University of Stirling

Faculty of Natural Sciences

Division of Computing Science and Mathematics

Understanding Malware using Ontology-based Knowledge Graph and Deep Learning Techniques

Ipshita Roy Chowdhury

A thesis submitted in fulfilment of the requirements for the degree of Master of Philosophy in Computing Science

March 2025

UNIVERSITY of STIRLING



Preface

This thesis presents the outcome of a sustained programme of Ph.D. research conducted in the field of Computing Science at the University of Stirling. The work was originally undertaken and submitted with the objective of fulfilling the requirements for the degree of Doctor of Philosophy and was developed over several years of full-time research.

Following the initial submission, the thesis underwent a formal examination process after which a period of substantial revision was required. The revised thesis was resubmitted for examination after an extended period of further research, redevelopment and consolidation of original contributions. The scope, depth and ambition of the work reflect the objectives of doctoral-level research, encompassing novel methodological development, theoretical integration and applied investigation at the intersection of cybersecurity, artificial intelligence and semantic technologies.

This final version of the thesis incorporates all amendments requested during the examination process. It represents the culmination of several years of independent research effort and reflects the author's academic intent to contribute original and rigorous scholarship to the field.

Abstract

Due to rapid growth and advancement of Big Data and Internet of Things (IoT), industries and organizations are being the main targets of cyber criminals. As IoT enables the linking of all types of 'smart' devices industrial appliances, personal information or sensitive data are being constantly attacked by novel malware variants. Therefore, in recent years cyber security has become a major concern due to the emergence of new advanced malware mutants which are causing harm to the users for financial gain to personal information loss. Due to transportation and sharing of huge amount of data, internet-enabled devices and social media platforms are becoming more vulnerable to new generation malware attacks. Because of the limitations of traditional Machine Learning based approaches, it's very difficult to deal with the new malware generations with complex behaviour and varying code structures.

Different categories of malware families represents complex, dynamic behaviours and characteristics which can cause a novel and targeted attack in a cyber-system. Existence of large volume of malware types with frequent new additions hinders the cyber resilience effort. This motivation leads to develop a new ontology driven framework that can capture recent malware behaviours and construct novel malware detection and classification framework for new generation malware. This research built an ontology-based knowledge graph to capture metamorphic malware behaviour in the form of API call sequences, and it constructed a dynamic model for implementing an API-based knowledge graph from the domain ontology. The ontological structure are used as the domain knowledge to develop several deep learning algorithms have been incorporated for the prediction of new generation malware classes. In this research a number of deep learning techniques were developed for malware analysis by incorporating CNNs, ResNet-50, GANs, achieving higher accuracies than state-of-the-art models. CNNs and other deep learning architectures were implemented as standalone models and by integrating with other models as well.

To address the limitations of static deep learning models, this work includes continual learning approaches to avoid catastrophic forgetting while

promoting adaptability in dynamic threat environments. It further addresses two new approaches bringing ontology to Deep Learning: Domain Adaptation, transferring knowledge from one source to one target domain with results pooled in the subsequent stage and then Building Knowledge Graphs using Deep Learning-based Image Classification and Ontology Integration to better support semantic representation through structured integration of the output of classifications into structured knowledge.

The overall research work has been carried out to built a meaningful domain specific ontological framework, a wide range of Deep Learning-based neural architectures have been implemented on publicly available benchmark datasets. Employment of various advanced deep learning techniques to improve accuracy and address key challenges such as catastrophic forgetting. Integration of deep learning and ontology with Domain Adaptation and construction of three deep learning based malware knowledge graphs for enhancing interpretability and advancing AI-driven malware analysis in cybersecurity.

COVID impact statement

Pre-Covid-19 research plan

- Research problem identification.
- Identification of 'Aims and Objectives'.
- Trying to achieve first objective.
- Preparation for first year report and viva.

Details of Covid-19 related disruption

- Due to the Covid-19 pandemic research work was being little slow because we couldn't access our labs and other facilities that University provides the students.
- Research needs collaboration and sharing ideas with peers. Due to the Covid-19 lockdown this meeting with peers was not possible at all.

Summary of the decision taken

- Trying to achieve other objectives to get the work done.
- I have taken three months extension to complete the whole thesis work.

Attestation

I understand the nature of plagiarism, and I am aware of the University's policy on this.

I certify that this thesis demonstrates original work by me except the following:

- The developed ontologies and evaluation results in the 'Methodology' section of Chapter 4 are taken from my own 'Capturing Malware Behaviour with Ontology-based Knowledge Graphs' paper.
- The developed methodologies in the 'Methodology' section of Chapter 5 are taken from my own 'Image-Based Malware Detection Using Deep Learning: Performance Evaluation and Comparative Analysis'.
- The developed methodologies in Chapter 6 are taken from my own 'Combating Catastrophic Forgetting: A Continual Learning Approach to Deep Learning-based Malware Detection'.
- The developed methodologies in Chapter 7 are taken from my own 'Integration of Ontology and Deep Learning using Domain Adaptation and Knowledge Graph Construction'.

Signature: *Ipshita Roy Chowdhury* **Date: 27.03.2025**

Acknowledgements

First and foremost, I express my deepest gratitude to my parents for their unconditional love, unwavering faith and constant support throughout this long and challenging academic journey. Their sacrifices, encouragement and belief in the value of perseverance made it possible for me to continue this research through years of intellectual effort and uncertainty.

I would like to sincerely thank my supervisor, Dr Deepayan Bhowmik, for his guidance, academic discussions, and support over the course of this research. His feedback and supervision contributed to shaping the direction of the work, and I appreciate the time and effort invested during the supervision period.

My sincere thanks go to the School of Computing Science and the University of Stirling for providing the academic environment, research facilities and administrative support necessary to conduct this research.

I would also like to thank Sam and Graham of the Computing Support Group at the University of Stirling for their invaluable technical assistance which ensured that the computational and research infrastructure supporting this work functioned smoothly throughout my studies. I sincerely appreciate their immense technical support during the pandemic period when I was working remotely from my home country.

My thanks go to Grace and Gemma of the administrative team for their consistent administrative support and assistance throughout my research journey.

Finally, I acknowledge my own perseverance and commitment in completing this research. This thesis represents several years of sustained effort, independent inquiry and original contributions.

Contents

Preface i

Abstract i

COVID impact statement iii

Attestation iv

Acknowledgements v

1 Introduction 2

1.1 General Context and motivation 3

1.2 Aim and Objectives 4

1.3 Contributions 5

1.4 Research Questions 5

1.5 Overview of Report 6

1.6 List of Publications 6

1.7 Conference Achievements 7

2 Background 8

2.1 Sandbox Detection 9

2.2 Signature-based Detection 10

2.3 Anomaly-based Malware Detection 10

2.4 Statistical Modeling for predicting Cyber attacks 10

2.4.1 Using Distributions 10

2.4.2 Copula-based Models 12

2.4.3 Bayesian Networks 12

2.5 Time-Series Modeling 13

2.6 Machine Learning-based Methods 14

2.6.1 Zero day attack defence 15
2.6.2 Machine Learning/Data Mining Approaches 15
2.7 Deep Learning-based Methods 16
2.8 Ontology-based Methods 17
2.8.1 Ontology Engineering 18
2.9 Conclusions 19

3 State-of-the-Art 20
3.1 Ontology in Malware Detection 21
3.1.1 Modeling Malware Behaviour: 22
3.1.2 Ontology-based Frameworks: 26
3.2 Machine Learning based Malware Detection 32
3.3 Deep Learning based Malware Detection 43
3.3.1 Behaviour-based Malware Detection 44
3.3.2 Image-based Malware Detection 48
3.3.3 Using Opcode Sequences 55
3.3.4 Intelligent Framework for Malware Detection 59
3.4 Continual Learning based Malware Detection 63
3.5 Domain Adaptation in Malware Detection 67
3.6 Ontology in Cybersecurity Knowledge Graphs 68
3.7 Discussions 72
3.8 Conclusions 73

4 An Ontology-based Knowledge graph to Capture Malware Behaviour 74
4.1 Contributions 75
4.2 Problem Statement 76
4.3 Methodology 77
4.3.1 Ontology Engineering 84
4.3.2 Handling Multiple Classification or Inheritance 90
4.3.3 Ontology-oriented programming 91
4.3.4 Algorithm of the Experiment 96

4.4 Evaluation of the Ontology 97
4.5 Discussions 104
4.6 Conclusions 105

5 Malware Analysis using Deep Learning based Image Processing 107
5.1 Introduction 107
5.1.1 Problem Statement 107
5.1.2 Contributions 108
5.2 Methodology and Experimental Apparatus 108
5.2.1 Dataset 1: Malimg 109
5.2.2 Dataset 2: Virus MNIST 110
5.2.3 Dataset 3: MaleVis 112
5.2.4 Methodology 113
5.2.5 Detection with CNN Model using Keras Tuner 115
5.2.6 Detection with TPU 127
5.2.7 Classification with ResNet-50 Transfer Learning 136
5.2.8 Classification with 3D CNN 143
5.2.9 CNN Classification with Data Augmentation 152
5.2.10 Image Generation Using Autoencoder 160
5.2.11 Classification with Ensembling CNNs 172
5.2.12 ResNet-50 with Attention Network 196
5.2.13 Classification with CNN Normalization 211
5.2.14 Classification with CNN and RNN 223
5.2.15 ANN Implementation 244
5.2.16 GAN Implementation 247
5.2.17 Implementation with Conditional GAN (cGAN) 254
5.2.18 Visualization with t-SNE plots 261
5.3 Conclusions 269

6 Malware Analysis using Deep Learning with Continual Learning 271
6.1 Malware Detection with Continual Learning 273

6.1.1 GANs for Data Augmentation with Continual Learning . . 273
6.1.2 U-Net with Continual Learning 281
6.1.3 An Adaptive Malware Detection Using Continual Learning: Integrating Elastic Weight Consolidation and ResNet-50 288
6.2 Conclusions 294

7 Combination of Ontology and Deep Learning 297
7.1 Introduction 297
7.1.1 Combining Ontology and Deep Learning with Domain Adaptation 298
7.2 Integration of Ontology and Deep Learning: Source Domain: Ontology; Target Domain: Malware Image Dataset 301
7.2.1 Methodology 303
7.3 Integration of Deep Learning and Ontology: Source Domain: Malware Image Dataset; Target Domain: Ontology 309
7.4 Novel Malware Knowledge Graphs 319
7.4.1 Methodology 321
7.5 Conclusions 335

8 Discussions 339
8.1 Malware Behaviour Ontology 339
8.2 Deep Learning based Techniques 340
8.3 Deep Learning with Continual Learning 344
8.4 Combination of Ontology and Deep Learning 347

9 Conclusions & Future Work 351
9.1 Conclusions 351
9.2 Future Work 357

List of Figures

1.1 Malware Code Structure 3
1.2 Windows malware and PUA 4

2.1 Attacker motives for espionage and financial gain 8
2.2 Ontology Engineering 18

3.1 State-of-the-art in malware detection 20
3.2 Schematics of CRATELO [1]. 26

4.1 Word Cloud of API Calls 78
4.2 High Frequency API Calls 79
4.3 Low Frequency API Calls 81
4.4 The base (or domain) Ontology 84
4.5 HermiT reasoner by Protégé 86
4.6 Individuals of Trojan 87
4.7 Individuals of other classes 87
4.8 OWL Object Properties 88
4.9 OWL Data Properties 89
4.10 Overall visualization of the ontology 90
4.11 'rdf:type' for classes 91
4.12 with object property 91
4.13 Source Code 92
4.14 Classes of the Ontology 93
4.15 Sequential API calls added as malware behaviour 94
4.16 Property of the Behaviour 94
4.17 Individuals of the output ontology 95

5.1 Sample of Malimg Dataset 109
5.2 Malware Families in Malimg Dataset [2] 110

5.3 Virus MNIST dataset with classes 112
5.4 MaleVis dataset with classes 112
5.5 Virus MNIST dataset with labels 121
5.6 Plot of Train Images of Virus MNIST 121
5.7 Average shape of Virus MNIST images 122
5.8 Prediction of Test Images with CNN 122
5.9 Confusion Matrix for CNN 123
5.10 Keras Tuner function 123
5.11 Confusion Matrix of CNN with Keras Tuner 124
5.12 MaleVis Confusion Matrix 125
5.13 Train, Validation and Test 129
5.14 Train Images of Virus MNIST 130
5.15 Virus MNIST Confusion Matrix 131
5.16 Virus MNIST Model Accuracy 131
5.17 Predicted Labels of Test Images 132
5.18 MaleVis TPU Accuracy and Loss 133
5.19 MaleVis Prediction Results 134
5.20 Malimg TPU Accuracy and Loss 135
5.21 Malimg Dataset Predictions 135
5.22 A ResNet-50 architecture 136
5.23 Accuracy and Loss Curves of ResNet-50 Transfer Learning 137
5.24 Prediction on Test Images for ResNet-50 Transfer Learning . . . 138
5.25 Accuracy and Loss Curve of MaleVis Dataset 139
5.26 Train images for 3D CNN 147
5.27 Loss and Accuracy Curve of 3D CNN 148
5.28 MaleVis Prediction Results 149
5.29 Malimg Accuracy and Loss Curve 150
5.30 Malimg Dataset Prediction Results 150
5.31 Plot of Train Images 154
5.32 Prediction of Trojan 155

5.33 Prediction of Downloader 155
5.34 Prediction of test images of Virus MNIST 156
5.35 MaleVis Accuracy and Loss 156
5.36 MaleVis Predicted Images 157
5.37 Malimg Accuracy and Loss 158
5.38 Malimg Predicted Images 158
5.39 Function of Autoencoder 160
5.40 Neural Network of Autoencoder 161
5.41 Train data with labels 165
5.42 Network architecture of autoencoder 166
5.43 Encoder Compresses the Original Image into a Lower-dimensional Representation 166
5.44 Reconstructed Images 167
5.45 Virus MNIST Autoencoder Reconstruction Error Distribution . . 167
5.46 MaleVis Autoencoder Recnstructed Images 168
5.47 MaleVis Autoencoder Reconstruction Error Distribution 168
5.48 Malimg Reconstructed Images 169
5.49 Malimg Autoencoder Reconstruction Error Distribution 169
5.50 Malware-trained Autoencoder Novelty Result 172
5.51 Images with labels 173
5.52 Multiple Images with labels 174
5.53 Image with pixels 174
5.54 Confusion Matrix of Random Forest Classifier 183
5.55 Confusion Matrix of Decision Tree Classifier 183
5.56 KNN Confusion Matrix 184
5.57 Confusion matrix of CNN model 185
5.58 Accuracy of Models 186
5.59 Test Dataset Prediction 186
5.60 10 CNN Confusion Matrix 187
5.61 MaleVis KNN Results 188

5.62 MaleVis Decision Tree Results 188
5.63 MaleVis Random Forest Results 189
5.64 MaleVis 10 CNN Accuracies 190
5.65 MaleVis 10 CNN Prediction Results 191
5.66 Malimg KNN Results 192
5.67 Malimg Decision Tree Results 192
5.68 Malimg Random Forest Results 193
5.69 Malimg 10 CNN Accuracy Plot 194
5.70 Malimg Prediction Results 195
5.71 Spatial and Channel Attention Modules 197
5.72 ResNet with Attention Module 200
5.73 Top-1 Accuracy and Loss 205
5.74 Prediction results of Virus MNIST 205
5.75 Accuracy Curve 206
5.76 Loss Curve 206
5.77 MaleVis Prediction Results 207
5.78 Confusion Matrix 207
5.79 Malimg Accuracy Curve 208
5.80 Malimg Dataset Prediction Results 208
5.81 Train Images to feed into the Model 211
5.82 Accuracy Curve of CNN 215
5.83 Confusion Matrix of CNN 215
5.84 Predicted class labels after the experiment 216
5.85 MaleVis Accuracy and Loss 217
5.86 MaleVis Confusion Matrix 218
5.87 MaleVis Prediction Results 219
5.88 Malimg Accuracy and Loss 220
5.89 Malimg Confusion Matrix 221
5.90 Malimg Prediction Results 222
5.91 Confusion Matrix of Virus MNIST 229

5.92 Confusion Matrix of MaleVis 230
5.93 MaleVis CNN Confusion Matrix 237
5.94 MaleVis RNN Confusion Matrix 238
5.95 Virus MNIST CNN Confusion Matrix 239
5.96 Virus MNIST RNN Confusion Matrix 240
5.97 Malimg Dataset CNN Confusion Matrix 241
5.98 Malimg Dataset RNN Confusion Matrix 242
5.99 ANN Model 244
5.100Input Images 247
5.101Generated Images of Virus MNIST dataset 250
5.102Real and Fake images of MaleVis dataset 251
5.103Grid of Real and Fake images of Malimg Dataset 251
5.104Real and Fake images of Malimg dataset 252
5.105Flowchart of Conditional GAN 255
5.106Parameters of Discriminator 255
5.107Parameters of Generator 256
5.108Sample Images of Virus MNIST 258
5.109Model Testing of Virus MNIST dataset 259
5.110MaleVis Generated Images 259
5.111Malimg Generated Images 260
5.112Visualization with Virus MNIST 262
5.113Visualization with Malimg dataset 264
5.114Visualization with MaleVis dataset 266

6.1 GAN with Continual Learning 274
6.2 Confusion Matrix of Malimg 278
6.3 Confusion Matrix of Virus MNIST 279
6.4 Confusion Matrix of MaleVis 280
6.5 Confusion Matrix of Malimg 285
6.6 Confusion Matrix of MaleVis 286
6.7 Model Architecture 289

7.1 Domain Adaptation from Ontology to Malware Image 301
7.2 Domain Adaptation from Ontology to Malware Image 304
7.3 Domain Adaptation from Ontology to Malware Image 310
7.4 MaleVis Confusion Matrix 316
7.5 Virus MNIST Confusion Matrix 317
7.6 Malimg Confusion Matrix 318
7.7 Behaviour of Trojan 322
7.8 Behaviour of Virus 322
7.9 MobileNetV2 Architecture to process the Image Dataset 323
7.10 Knwledge Graph Population to create the ontology with Image Dataset and .owl file 325
7.11 Architecture of the Experiment for Novel Knowledge graph Generation 326
7.12 Visualization of MaleVis Dataset with Malware Ontology 329
7.13 Classes of MaleVis Dataset with Malware Ontology 329
7.14 Classes of Virus MNIST Dataset with Malware Ontology 330
7.15 Visualization of Virus MNIST Dataset with Malware Ontology . . 331
7.16 Classes of Malimg Dataset with Malware Ontology 332
7.17 Visualization of Malimg Dataset with Malware Ontology 333

1 Introduction

Because of the digital infrastructure of the Internet and cyber system many dominant cyber threats are horrifying the interconnected industries. Cyber criminals are writing new malware variants to damage the end users computer and network system with advanced attack vectors. Because of the ubiquitous nature of today's Internet there are always new emerging cyber threats that are causing significant growth of new generation malware and attack variants with multiple mutants [3].

Malware can compromise a computer system in any form such as Internet worms, viruses, Trojans etc. There exists a wide variety of malware types, including Trojan horses, ransomware, viruses, spyware, adware, worms, DDoS, zombies, backdoors and so on. This vast diversity of malware types makes detection and classification increasingly challenging, especially as cybercriminals continuously evolve their tactics. To evade traditional security measures, malware authors employ sophisticated techniques that alter the code structure, making it harder to identify through signature-based detection methods. There are three types of malware depending upon the code structure: Basic, Polymorphic and Metamorphic. Advanced techniques like Camouflage, Packing or code obfuscation are adopted by the malware developers to make the code unreadable and avoid detection. Figure 1.1 is showing the types of malware according to the code structure.

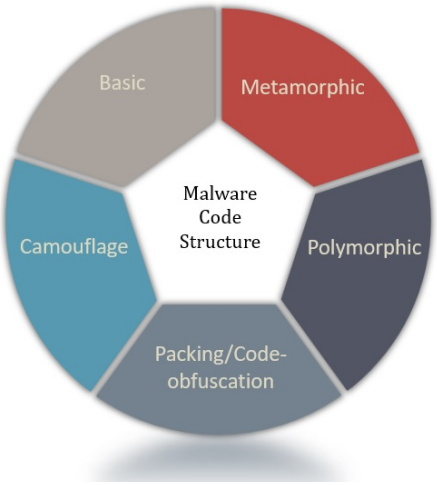


Figure 1.1: Malware Code Structure

1.1 General Context and motivation

Detection of new generation malware is a challenging task because of advanced anti-detection techniques and ability to change the malicious code in every iteration (polymorphism and metamorphism). Behaviours of polymorphic and metamorphic malware are difficult to detect because of their complex and dynamic nature.

According to AV-TEST report there is a constant distribution of Windows malware and PUA (Potentially Unwanted Applications) of last twelve months is shown in Figure 1.2 [1]. The chart visualizes the number of new malware samples detected over different time intervals, from the past 12 months down to a per-second rate,

- The chart illustrates the sheer scale of malware proliferation, emphasizing how frequently new threats emerge.
- The funnel shape suggests progressive filtering or breakdown of malware statistics over time.
- The per-second rate (3.3 new malware samples) highlights the rapid and continuous nature of malware evolution, posing a significant challenge for cybersecurity defenses.

[1] https://portal.av-atlas.org/malware

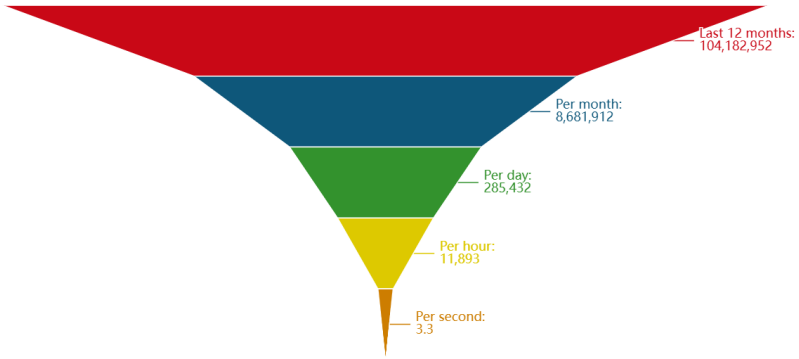


Figure 1.2: Windows malware and PUA

New generation malware can produce ever-changing malicious behaviour so domain knowledge reusability is not possible to analyse them. Also encryption and decryption of polymorphic malware code and translation of metamorphic malware code in a new form in every iteration are challenging to detect [4].

Ontology oriented knowledge framework is required to improve the dynamic analysis of malware behaviour and to have a clear understanding about the behaviour of malicious code. Due to exploration of Deep Learning techniques and Computer Vision Image-based Malware Detection is growing popularity because image files are mostly benign, code visualization friendly and informative. Both Ontology and Deep Learning can facilitate malware behaviour analysis in terms of knowledge specific and visualization purposes.

1.2 Aim and Objectives

This research aims to develop a malware detection framework based on an ontology-driven knowledge graph and deep learning-based image processing to capture malware behaviour and to analyze malware images respectively. For this purpose, this thesis is trying to achieve following objectives:

- To develop an ontological framework for capturing malware behaviour dynamically.

- To understand the features of malware families and behaviour of the malware.
- To process malware images with deep learning architectures for finding novel detection and classification methods.
- To combine the Ontological framework with Deep Learning.

1.3 Contributions

This main contributions of this research are:

- An Ontology-driven knowledge graph was developed to capture the behaviour of metamorphic malware in the form of API call sequences.
- Deep Learning based malware image processing architectures were developed for the detection and classification of malware families.
- Novel detection frameworks by combining continual learning with deep learning models.
- Combination of Ontology and Deep learning with two different methods: “Domain Adaptation" and “Building Knowledge Graphs with Deep Learning-based Image Classification and Ontology Integration".

1.4 Research Questions

Few research questions have to be worked out to complete the objectives of the research.

- What are the key features and behavioural patterns that distinguish different malware families and how can these be effectively modeled and analyzed?
- How can deep learning architectures be employed for malware image processing to achieve accurate and robust malware detection and classification?
- How can continual learning be integrated with deep learning models to improve adaptability and reduce catastrophic forgetting in malware detection tasks?
- In what ways can ontology and deep learning be effectively combined to enhance malware understanding?

1.5 Overview of Report

The thesis is divided into following sections to describe the progress, overview and objective of the research:

- Chapter 1 - This chapter describes the Introduction, motivation, objectives and research questions.
- Chapter 2 - This chapter describes the previous malware detection systems, background of deep learning and ontology in malware detection.
- Chapter 3 - State-of-the-art describes the relevant research papers of malware detection techniques.
- Chapter 4 - An Ontology-based Knowledge graph to Capture Malware Behaviour describes the overall process to build the ontology in an object oriented view.
- Chapter 5 - Malware Analysis using Deep Learning based Image Processing describes the deep learning based approaches to analyze the malware images.
- Chapter 6 - Implementation of Continual Learning with deep learning techniques.
- Chapter 7 - The combination of ontology and deep learning were exhibited in this chapter.
- Chapter 8 - This chapter describes the discussions about the research.
- Chapter 9 - Conclusion and Future Work describes the overview of the thesis and plan for the future scope of this research.

1.6 List of Publications

1. Ipshita Roy Chowdhury and Deepayan Bhowmik. “Capturing Malware Behaviour with Ontology-based Knowledge Graphs”. [Published: 5th IEEE Conference on Dependable and Secure Computing (IEEE DSC 2022), Edinburgh, UK] [LINK]
2. Ipshita Roy Chowdhury and Deepayan Bhowmik. “Image-Based Malware Detection Using Deep Learning: Performance Evaluation and Comparative Analysis." [Ongoing]

3. Ipshita Roy Chowdhury and Deepayan Bhowmik. “Combating Catastrophic Forgetting: A Continual Learning Approach to Deep Learning-based Malware Detection." [Ongoing]
4. Ipshita Roy Chowdhury and Deepayan Bhowmik. “Integration of Ontology and Deep Learning using Domain Adaptation and Knowledge Graph Construction." [Ongoing]

1.7 Conference Achievements

- SICSA Cyber Security PhD Forum: 3rd December 2021: Winner
- SICSA Cyber Security PhD Forum: 4th June 2020: Second runner up.

2 Background

In the era of Big Data, smart devices are always being used to generate and exchange huge amount of data and information, among the users, which is extremely difficult to handle with traditional processes. In the platform of complex data structure it's not easy to overcome the advanced attack patterns and threat techniques. Because of massive usage of interconnected systems cyber related threats are constantly increasing due to AI-based systems, phishing emails and most importantly novel malware variants. With advanced techniques malware can be installed in a system or network at any moment, even in real-time. According to Positive Technologies[1] dataset 69% of the malware sample were used for espionage to create a long term stealth also rest of the samples were used for financial gain (Figure 2.1).

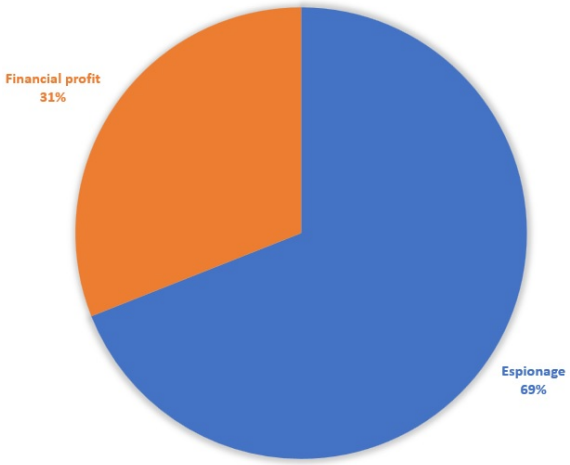


Figure 2.1: Attacker motives for espionage and financial gain

[1] https://www.ptsecurity.com/ww-en/

Traditional machine learning based malware detection and classification approaches involves with human intervention for feature learning and domain expertise to understand the features. Though Machine Learning depends on less data points rather than Deep Learning but it doesn't work well to recognize novel malware behaviour and requires more structured data. On the other hand, Deep Learning is less time consuming and an automatic process in terms of feature engineering, learning, self-correction and decision making.

Some traditional methods of malware detection are discussed in this section like:

- Sandbox Detection
- Signature-based Detection
- Anomaly-based malware detection
- Statistical Modeling for predicting Cyber attacks
- Time-Series Modeling

Few Machine Learning and Deep Learning based works have also been discussed in this section.

2.1 Sandbox Detection

A sandbox is an isolated environment to execute and run the code virtually to check behaviour and activity of the code whether the file or software is malicious or not [5]. Sandbox is a proactive measure which facilitates the Anti Virus to identify the malware. But the malware writers developed a number of techniques to evade sandbox analysis and spot the virtual environment [6]. Threat actors apply different methods and combine multiple techniques to avoid sandbox analysis and detection in a virtual environment.

Though Sandbox detection comes under Behaviour-based malware detection but there are several anti-sanboxing and anti-virtualization techniques to avoid its analysis processes.

2.2 Signature-based Detection

Signature-based malware detection method depends on pattern or signature matching. The unique digital signature of every app or software gets identified by the Anti Virus to add in the database if it's malicious. But this method cannot detect new generation malware signatures with complex behaviour and anti-detection techniques [7]. More than 3.5 million malware are being generated every day so detection of zero-day attack is almost impossible with this method.

2.3 Anomaly-based Malware Detection

Anomaly-based detection technique uses its knowledge to distinguish the deviation between normal and malicious behaviour during program inspection. This technique is generally used to develop Intrusion Detection Systems (IDSs). The goal of this technique is to characterize the normal behaviour and identify attacks based on deviations from normalcy [8]. Those behaviours are learnt by machine learning approach with the current and past interactions with the network, server or user's machine. This technique is based on the ability to predict network traffic flow. The major shortcomings of anomaly-based malware detection are high false alarm or false positive rate, time complexity and difficulty in feature selection for training phase [9].

2.4 Statistical Modeling for predicting Cyber attacks

2.4.1 Using Distributions

- **Pareto Distribution:** Wheatley *et al.*, 2015 investigated the extreme risk of personal data breaches in [10] where breached data from 2000 through 2015 was studied. The severity of breach sizes was modeled by an extremely heavy tailed truncated Pareto distribution, with tail exponent parameter decreasing linearly from 0.57 in 2007 to 0.37 in 2015.

Table 2.1: Previous Malware Detection Methods

Statistical Modeling for predicting Cyber attacks		
Paper	Strengths	Weaknesses
Wheatley et al., 2015 [10] and Maillart and Sornette, 2010 [11]	– In [10], severity of breach sizes was modeled by an extremely heavy tailed truncated Pareto distribution. – In [11], the severity of ID losses was described by a heavy-tailed power-law distribution.	– Statistical methods were developed for detection. – In recent times, more advanced type of cyber related incidents are being occurred by new malware generations.
Peng et al., 2018 [12] and Zhan et al., 2015 [13]	– In [12], Copula-GARCH models were used to model multivariate cyber security risks. – FARIMA and GARCH were used in [13] for short term attack prediction.	In [12] and [13] the data was collected from honeypots which may not identify the attacker and attack patterns leading the attacker to gain access to the legitimate system.
Edwards et al., 2015 [14] and Xu et al., 2018 [15]	– In [14], size of data beaches is following a log-normal distribution and daily breach frequency was modeled by a negative binomial distribution. – In [15] cyber hacking breach incident inter-arrival times and breach sizes were modeled by stochastic processes.	The Privacy Rights Clearinghouse (PRC) [16] dataset was used in [14] and [15]. Though the dataset may have some missing values which may not represent all the breach information and detection of attack patterns are becoming crucial in recent times.

- **Power Law Distribution**

  Another research related to investigation of personal identity losses was found in [11] where Maillart and Sornette (2010) described the severity of ID losses by a heavy-tailed power-law distribution.

- **Negative Binomial Distribution**

  A popular cyber breach incident dataset [16] was used in [14] to model the trend of data breaches. Size of data beaches is following a log-normal distribution and daily breach frequency was modeled by a negative binomial distribution.

  Forecasting of cyber incidents was performed by using Negative Binomial Distribution with one week lag in [3].

2.4.2 Copula-based Models

In [12] researchers initiated a study to model multivariate cyber security risks using Copula-GARCH models. To model high dimensional cyber security risks vine copula approach was used. In [15] cyber hacking breach incident inter-arrival times and breach sizes were modeled by stochastic processes. A positive dependence between the incidents inter-arrival times and the breach sizes has been discovered. This dependence can be adequately described by a particular copula.

2.4.3 Bayesian Networks

In [17] Bayesian Networks was used to identify types of uncertainty in enterprise networks. A Bayesian Network was built to justify the approach. A Bayesian State Space Model (BSSM) [3] was used to predict the number of future cyber attacks one week ahead of time. BSSM is a sort of machine learning approach and it has several advantages over time series models (like ARIMA and ARMA). BSSM can handle overdispersed data by using negative binomial distribution rather than normal distribution.

In [18] Sun et al., 2018 proposed a probabilistic approach to implement a prototype system named ZePro to identify zero-day attack path. Researchers described the zero-day attack path as a graph. This is the first approach until 2018, where probabilistic approaches were incorporated to identify zero-day attack path. This system constructs a Bayesian network based on the object instance graph.

2.5 Time-Series Modeling

The time series models are constructed to account for patterns in past movements in a manner useful for forecasting future behavior.

Autoregressive Integrated Moving Average (ARIMA) model is one of the most commonly used models for univariate time series. The ARIMA model is a special case of the ARMA model but includes a parameter to difference the transformation of the data. This model was used in [19] to forecast number of attacks in a future date.

FARIMA (Fractional Autoregressive Integrated Moving Average) and GARCH (Generalized Autoregressive Conditional Heteroskedasticity) are two widely used time series models. This model was used in [13] for short term attack prediction.

ARMA-GARCH model can precisely describe the evolution of the hacking breach sizes, where ARMA is acronym for "Autoregressive and Moving Average" and GARCH is acronym for "Generalized Autoregressive Conditional Heteroskedasticity." These stochastic process models can forecast the inter-arrival times and the breach sizes [15] by using the dataset from [16]. This is the first approach until 2018, where stochastic models were used to model cyber attacks rather than distribution.

In [10] and [11] the statistical methods were developed to detect personal data breaches and personal identity losses respectively. Though, in recent times there are more advanced type of cyber related incidents are being occurred by new malware generations.

In [12] and [13] the data was collected from honeypots which may not identify the attacker and attack patterns leading the attacker to gain access to the legitimate system. The Privacy Rights Clearinghouse (PRC) [16] dataset was also used in [14] and [15] to identify the patterns of data breaches and to analyze inter-arrival time and breach size of the incidents respectively. Though the dataset may have some missing values which may not represent all the breach information and detection of attack patterns are becoming crucial in recent times. In [17] and [18], graph based models were used to analyze the security events. The graph representations need to be more informative and structured to identify the attributes and properties. In [19] ARIMA model was

used for forecasting but it's a quite older model and doesn't fit for long-term events.

2.6 Machine Learning-based Methods

Machine Learning models had widely been used in malware detection field because it is able to cope with evolution of malware and attack variants. Several static and dynamic feature analysis approaches were implemented using machine learning algorithms.

Table 2.2 is showing some works related to static analysis based malware detection with their feature types and algorithms also Table 2.3 is showing few dynamic analysis for malware detection.

Table 2.2: Static Analysis for malware detection

Static Analysis		
Paper	Feature	Algorithm
Fuyong and Tiezhu, 2017 [20]	3-g (bytes)	1-NN
Yuxin and Siyi, 2019 [21]	opcodes	DBN, SVM, K-NN
Ahmadi et al., 2016 [22]	API Calls	GB
Wojnowicz et al., 2016 [23]	String features	LR
Hassen and Chan, 2017 [24]	Function Call Graph	RF, meta-classifier

Table 2.3: Dynamic Analysis for malware detection

Dynamic Analysis		
Paper	Feature	Algorithm
Carlin et al., 2017 [25]	Opcode	RF, Hidden Markov models
O'Kane et al., 2016 [26]	PCA	SVM
Bekerman et al., 2015 [27]	Correlation Feature Selection Algorithm	Naive Bayes, J48 DT, RF
Galal et al., 2016 [28]	Hand-crafted Heuristics	DT, RF, SVM
Salehi et al., 2017 [29]	Fisher Score, SVM	RF, J48 DT, Bayesian Logistics Regression

2.6.1 Zero day attack defence

To overcome the limitations of signature-based and anomaly-based detection system a new approach was proposed in [30] to detect zero-day attacks. A machine learning based framework ZDAR (ZERO-DAY ATTACK REMEDY) was proposed to sense the organization's network traffic for identifying anomalous behaviour that can cause zero-day exploit. In this approach supervised classification was used for known samples and unsupervised learning was used to identify new malware samples.

A Zero-day malware detection framework based on data mining techniques was developed in [31]. Different training files and classifiers for machine learning based zero-day malware detection were described in [32] to distinguish malware and benign samples.

2.6.2 Machine Learning/Data Mining Approaches

Based on the literature in [33] a survey of machine learning (ML) and data mining (DM) methods has been described for cyber analytics in support of intrusion detection. ML algorithms need a goal i.e. problem formulation from the domain (e.g., dependent variable to predict). DM focuses on the discovery of previously unknown characteristics in the data. It does not need a specific

goal from the domain, alternatively it focuses on finding new and interesting knowledge.

Related work can be found in [34] where a hardware based architecture, GuardOL, has been developed to perform online malware detection. GuardOL is a combination of processor and field-programmable gate array (FPGA). A machine learning approach (using MLP) has been implemented to capture the malicious behavior i.e., high level semantics of malware. A bipartite graph-based machine learning algorithm was used in [35] to classify benign and malicious network activities depending upon the behaviour to enable the defense systems successfully respond to novel attacks.

In [30], [31], [32] zero-day attack was detected using machine learning algorithms and in [34] and [35] machine learning methods were used with data mining approaches but feature selection is a time-consuming phase in these works.

Because of the rapid growth of new generation malware with varying complexity and anti-detection techniques cyber-security field needs to be secured and more advanced. As feature selection and extraction are the time consuming processes in machine learning also human intervention is a requirement in ML to understand the features. Recently, Computer Vision and Natural Language Processing are being incorporated in cyber security field so ML-based approaches are being replaced by Deep Learning based methods to overcome the limitations and solve the problems of malware detection and classification.

Due to several drawbacks of previously mentioned methods new research dimension is required in malware detection field and to detect malicious behaviour.

2.7 Deep Learning-based Methods

Deep Learning is being widely used in many real-world scenarios like Computer Vision, Robotics, Speech Recognition, Google Assistant, image processing and so on. Nowadays Deep Learning along with Artificial Neural Networks (ANNs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are being used in malware detection and classification where

data were used in the form of byte sequences, API call sequences, OpCode sequences, network traffic and gray-scale images. As Deep Learning dominates Computer Vision so it is very useful in image-based malware analysis for it's automatic feature extraction and classification purposes. In case of feed-forward neural networks the flow comes from input to output but in backpropagation the flow moves in opposite direction that is useful to calculate the error and fit the algorithm properly.

Table 2.4 is showing some deep learning based research works with algorithms and features.

Table 2.4: Deep Learning for malware detection

Deep Learning Works		
Paper	Feature	Algorithm
Saxe et al., 2015 [36]	PE import features, PE metadata features	Feed-forward network
Huang and Strokes, 2016 [37]	API 3-g, Sequence of API calls	Feed-forward network
Gilbert et al., 2019 [38]	gray-scale image	CNN
Rezende et al., 2017 [39]	gray-scale image	ResNet-50
Athiwaratkun et al., 2017 [40]	API call sequence	LSTM, GRU
AL-Hawawreh et al., 2018 [41]	Network behavior	AE , NN

2.8 Ontology-based Methods

Ontologies are being used in cyber security field to share knowledge [42], [43], mitigate the complexity related with taxonomy [44], [45], semantically analyze the attacks and attack patterns [46], [47], [48], [49] and vulnerabilities [50], classify the attacks [51] and to model the security requirements [52], [53].

2.8.1 Ontology Engineering

"Ontological Engineering" can be viewed as the set of activities to develop the ontology, the vocabularies, methodologies for building ontologies, and the languages which support them [54].

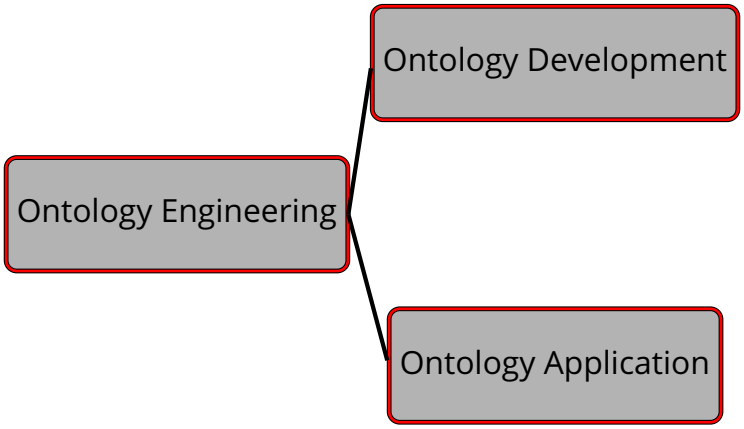


Figure 2.2: Ontology Engineering

Ontology engineering helps the domain experts to understand and explore their field more specifically. Ontology engineering is the field of applied ontology (Figure 2.2). It can be viewed as a large-scale representation of abstract concepts like, actions, time, physical objects etc. would be the examples of ontology engineering.

Due to the ever-changing nature of cyber domain, ontologies are needed to be incorporated as for security purposes as it can figure out the logic behind a cyber realated incident and what type of malware variants are being used by the attackers also to find out information about any hidden motive of the attack vector. Ontologies can help to analyze any organizational infrastructure to identify the vulnerabilities and to take measurement for developing security patches. Section 3.1 of Chapter 3 is describing a literature review for ontology based applications in cyber security field. Chapter 4 has described the contribution of this research towards cyber security by developing an ontology based malware behaviour knowledge graph.

2.9 Conclusions

This chapter has described the previous methodologies of cyber security domain from Section 2.1 to Section 2.5. Applications of Machine Learning, Deep Learning and Ontologies in cyber security field have also been discussed in this chapter.

This research aims to identify novel research gaps in this field to overcome the challenges related to previous methods to contribute new concepts for malware behaviour analysis also for malware detection and classification.

3 State-of-the-Art

To complete the state-of-the-art (Figure 3.1) of this thesis following literature survey was carried the for critical analysis of the research papers and to find the gaps in the literature. At first the relevant papers of the ontology-based malware detection were described after that several machine learning and deep learning based malware detection approaches were studied for the discussion.

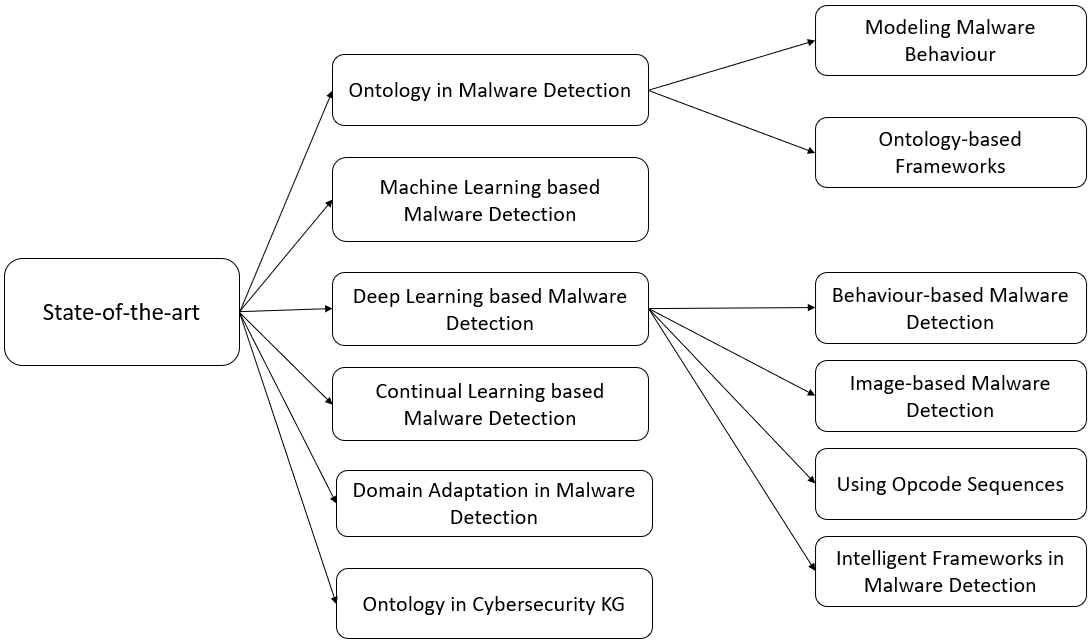


Figure 3.1: State-of-the-art in malware detection

The discussed ontology based frameworks in this chapter, were used for several research directions like, malware behaviour modeling, combining cy-

ber security ontologies, taxonomical and semantical structures, development of intelligent systems and security of mobile devices. Machine Learning based systems also provided new insights to protect end users and Android devices against cyber attacks. Several Deep Learning based malware analysis methods have also been discussed as deep learning has become a useful resource for the security researchers to handle large datasets, multidimensional and multimodal datasets also to analyze API sequences and opcodes.

3.1 Ontology in Malware Detection

Ontology could be a key building block of malware detection and modeling malware behavior through the provision of a structured, semantic description of malicious activity, enabling knowledge-driven analysis and automated reasoning. It formalizes relations between malware attributes such as API calls, system modifications, dissemination methods, persistence strategies, and evasion techniques to enable detection based on semantic reasoning, rule-based deduction and cyber threat intelligence interoperability.

By creating ontology-based knowledge graphs, security systems are able to model malware behaviour in a format amenable to advanced analytics, auto-classification and real-time threat identification. Ontology-based representations support detection of advanced attack patterns, tracking of malware development and correlating multiple attack surfaces' threat indicators to make sharing of cyber threat intelligence more effective.

In addition, ontology-based approaches enable proactive defense through the characterization of malware families, their variants and behavioral traits in a systematic manner, aiding in zero-day threat detection. By leveraging formal logic and reasoning mechanisms, ontologies enable decision-making in cybersecurity through the inference of new attack patterns by analysts and automated systems, risk assessment and the strengthening of defensive mechanisms. In the era of malware threat complexity, ontology-based methods provide a scalable, interpretable and adaptive foundation for intelligent malware analysis and cybersecurity defense.

This section is describing a number of application of ontology in the field of cyber security and malware behaviour analysis:

3.1.1 Modeling Malware Behaviour:

Table 3.1: Ontology in Malware Detection

Modeling Malware Behaviour		
Paper	Strengths	Weaknesses
Grégio et al. (2014) [55]	A malware ontology was proposed based on their exhibited behaviour, • to identify the unknown malware samples, • to distinguish the suspicious program from the benign.	This paper is based on modeling suspicious malware behaviours but this requires to be more sophisticated to detect new malware behaviours.
Grégio et al. (2016) [56]	• Malware Behaviour Ontology (MBO) was proposed to model the knowledge of malware behaviour. • Used over two thousand malicious samples and 400 benign samples.	• MBO produced some false positive results. • Machine Learning techniques need to be incorporated with MBO rules to detect new threats.

Table 3.2: Ontology in Malware Detection (Cont.)

Modeling Malware Behaviour		
Paper	Strengths	Weaknesses
Kiwia et al., 2018 [57]	• A cyber-kill chain based taxonomy of banking Trojan was proposed. • 127 banking Trojans from December 2014 to January 2016 of a UK-based financial organisation was used.	This a taxonomy-based approach, not an ontology.
Tafazzoli and Sadjadi, 2008 [58] and N. Rastogi, 2020 [59]	• In [58] a malware ontology was proposed to represent analyzed malware characteristics and their relationships. • MALOnt [59] is an open-source malware threat intelligence ontology was developed for knowledge-graph generation and information extraction.	• Fuzzy representation [58] has major limitations in data handling and requires human intervention for inference. • For both of these ontologies in [58] and [59] richness still needs to be improved.

Proper understanding and comprehension of domain knowledge of malicious behaviour are required to mitigate security related problems. In [55] Grégio et al. (2014) proposed a malware ontology based on their exhibited behaviour to identify the unknown malware samples and to distinguish the suspicious program from the benign. This work was also done to model authors knowledge of malware behaviour to find a solution from new generation malware samples which can easily compromise the detection techniques of users' systems. This paper is based on modeling suspicious malware behaviours but this requires to be more sophisticated to detect new malware behaviours. This framework was later extended into the Malware Behaviour Ontology (MBO) which was proposed in [56] by Grégio et al. (2016). They proposed an ontol-

ogy to model the knowledge of malware behaviour by using over two thousand malicious samples and almost 400 benign samples to test the rules of suspicious behaviours. MBO is an extensible ontology which can accommodate new malware features and implemented as a collaborative framework. This ontology is flexible and rule based to determine advanced malicious behaviours. But MBO produced some false positive results for real time malware detection. Machine Learning techniques need to be incorporated with MBO rules to detect new threats.

Malware detection is a challenging task in financial industries. Detection of banking Trojans is a great challenge due to advanced anti-detection techniques like obfuscation. A cyber-kill chain based taxonomy of banking Trojan features was proposed in [57] to improve the mitigation techniques. Kiwia et al. (2018) proposed the CKC-taxonomy of banking Trojan features based on the evolutionary computational intelligence. In this work 127 banking Trojans from December 2014 to January 2016 of a UK-based financial organization have been used to validate the taxonomy. This work is useful for the cyber security practitioners to develop mitigation techniques but this a taxonomy-based approach which is not an ontology.

In [58] a malware ontology was proposed to represent analyzed malware characteristics and their relationships. A semantic relation mapping was presented and used in semantic search engines. A malware fuzzy ontology was developed to describe the malware relationships. This is the first malware ontology which was created manually with a fuzzy representation of the framework to be used in semantic web. MALOnt [59] is an open-source malware threat intelligence ontology was developed for knowledge-graph generation and information extraction. Both of these ontologies in [58] and [59] presented detail understanding of malware and their characteristics but richness of the ontologies still needs to be improved. Though MALOnt [59] has a great significance for this thesis as it was compared with the developed ontology based knowledge graph in Chapter 4.

Table 3.3: Ontology in Malware Detection (Cont.)

Modeling Malware Behaviour		
Paper	Strengths	Weaknesses
Mundie and Mcintire, 2013 [60]	• In this framework an ontology-based competency model was built. • Consists of vocabularies, taxonomy, static ontology of malware analysis.	More conception of cyber security field and a dynamic sense are required to improve the analysis by the ontology.
Abdoli et al., 2008 [61]	An ontology of denial of service attack was developed using Protégé software.	Constructed based on denial of service attack only but more fields of cyber security domain need to be included.

Lack of well structured database is a challenge in cyber security field and domain knowledge needs to be semantically structured in Information Security field also. Iannacone et al. (2015) proposed an ontology in [62] to combine various publicly available datasets with internal information for the analysts and automated tools in order to overcome the lack of structured datasets. This work is a contribution towards cyber security domain with the available datasets and relevant information in a structured format. This ontology is general-purpose and lacked specialization for behaviour-based malware analysis. Additionally, ontology development in specific attack domains was reflected in [61]. Ontology of network and computer attacks are also essential to understand the attack pattern. In [61] an ontology of denial of service (DoS) attack was developed using Protégé software. The accuracy of the ontology was tested using Racer software and KDD cup99 test dataset. This ontology was constructed based on denial of service attack only but more fields of cyber security domain need to be included.

An OWL-based malware analysis ontology was built in [60]. A malware analysis dictionary and taxonomy were built and by combining these two a competency model was developed to create an ontology-based competency framework. This ontology consists of vocabularies, taxonomy and static ontology of malware analysis but more conception of cyber security field and a

dynamic sense are required to improve the analysis by the ontology.

All of the previous studies proclaim the evolution of malware and cybersecurity knowledge that has been semantically structured — from rule-based systems [55], [56], to domain-specific taxonomy [57], to fuzzy and open-source ontologies [58], [59], and finally the efforts were made towards dataset structuring and competency modeling [62], [60]. But no existing ontology-based framework fully integrates behaviour-level reasoning (e.g., API call sequences), dynamic knowledge graph generation, and image-based malware classification. The previously developed systems contributed important application-specific insights but were limited by their focus on static features and mobile based systems, and did not address metamorphic malware behaviour and semantic graph integration.

3.1.2 Ontology-based Frameworks:

CRATELO is an ontological framework constitutes the domain ontology of cyber operations (OSCO: Ontologies of Secure Cyber Operations), designed on the basis of DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) top ontology and middle-level ontology (SECCO: Security Core Ontology) [1].

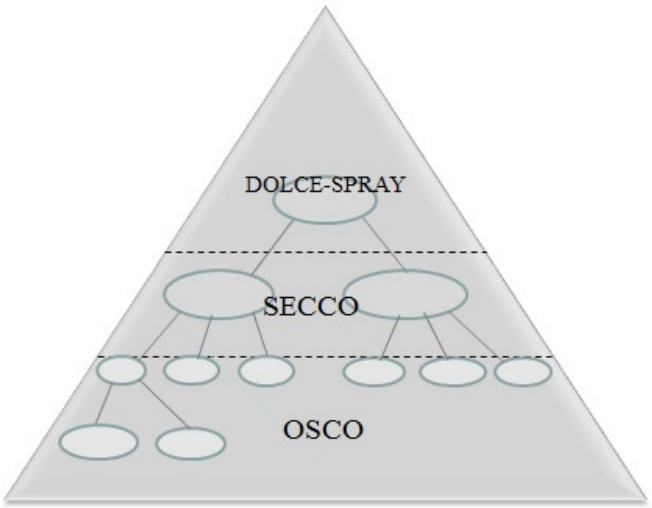


Figure 3.2: Schematics of CRATELO [1].

Table 3.4: Ontology in Malware Detection (Cont.)

Ontology-based Frameworks		
Paper	Strengths	Weaknesses
Oltramari et al., 2014 [1] and Syed et al., 2016 [63]	• CRATELO [1] is an ontological framework constitutes the domain ontology of cyber operations. • Syed et al., 2016 proposed an Unified Cybersecurity Ontology (UCO) in [63].	These two ontologies need to be improved with other real-world cyber security aspects for vulnerability assessment and interaction with other sources.
Huang et al., 2010 [64]	An ontology based intelligent system named Taiwan Malware Analysis Net (TWMAN) was proposed to analyze malware behaviour.	Ontology was created with three types of malware families but recent malware generations have more advanced types malware threats.

The three levels of CRATELO currently include 223 classes and 131 relationships encoded in OWL-DL. This framework was developed to improve situation awareness and ensures the depth of semantic structure of the cyberspace. To mitigate the need for semantic integration across diverse security systems, Syed et al. (2016) proposed an Unified Cybersecurity Ontology (UCO) in [63] to integrate heterogeneous data and domain knowledge of different existing cybersecurity systems. This is the first cyber security ontology until 2016 where mappings between UCO and general world ontologies were generated using use-cases for ontology sharing and reuse. These two ontologies need to be improved with other real-world cyber security aspects for vulnerability assessment and interaction with other sources.

In contrast to the abstract models, Huang et al., 2010 proposed an ontology based intelligent system named Taiwan Malware Analysis Net (TWMAN) in [64] to analyze malware behaviour. The behaviour information was stored in ontology repository which is used by the malware behaviour analysis agent and ontology agent to keep the system safe from Viruses and Trojans. The malware behaviour ontology was constructed by defining File modifications,

Network communications and Registry run key values which are beneficial to comprehend common reactions and characteristics of a malware but behavioural analysis needs to be more specific and dynamic in nature. Though this ontology was created with three types of malware families but recent malware generations have more advanced types malware threats.

A similar static approach was implemented in [65] where detection of malware by its behaviour was demonstrated using Ontologies and rules. Infected computer systems were used to develop a host level detection mechanism to identify obfuscated malware codes. Different automated malware detection and prevention approaches were developed previously. This is a malware hunting framework based on rules therefore it is time consuming and needs human intervention to set the rules.

Mobile devices are the repository of users' financial information, social networking activities, banking and emailing. Mapping the relationship among permissions, malware, and benign apps is troublesome and cannot be done manually. To facilitate the application testing in Android ecosystem an ontology based framework was developed in [66] by Navarro et al., 2018 to map the relationship between application and system and a machine learning framework to analyze the malware features. But incorporating a large amount of data could be an issue in this ontology and machine learning based system. Similarly, Chiang et al., 2010 proposed an ontology based approach for mobile malware behaviour analysis in [67] to provide information about a new mobile malware signature in a proper time to the users and organizations to keep their mobile devices safe when the device is not able to obtain the information because of limited resources and outdated malware signature within the device. The limitation is, this method is to some extent signature based therefore users' need enough precautions against malware.

Other mobile based frameworks such as an intelligent model for mobile malware detection was build in [68]. Based on the static features an Apps Feature Ontology (AFO) was built. A concept vector was created and features were selected using optimization algorithms. This approach was evaluated on benchmark datasets. This framework is based on static features from the app therefore inclusion of new malware features is a limitation for this method. A more network-focused approach was introduced in [69] ontology was used to develop a model which can detect attack profile of a malware and prevent

network threats but not useful to determine targeted attack which can compromise mobile security.

Table 3.5: Ontology in Malware Detection (Cont.)

Ontology-based Frameworks		
Paper	Strengths	Weaknesses
Jasiul et al., 2014 [65]	• Detection of malware by its behaviour was demonstrated using Ontologies and rules. • Infected computer systems were used to develop a host level detection mechanism.	This malware hunting framework based on rules therefore it's time consuming and needs human intervention to set the rules.
Navarro et al., 2018 [66]	• An ontology based framework was developed to map the relationship between application and system. • Facilitating the application testing in Android ecosystem.	But incorporating a large amount of data could be an issue in this ontology and machine learning based system.
Chiang et al., 2010 [67]	• An ontology based approach for mobile malware behaviour analysis was proposed. • To provide information about a new mobile malware signature in a proper time to the users and organizations.	This method is too some extent signature based therefore users' need enough precautions against malware.

An advanced semantic decision making system was proposed in [70] to identify malicious programs. In this work, the integration of semantic technologies and computational intelligence methods was done by integrating the

Fuzzy Ontologies and Fuzzy Markup Language (FML). But fuzzy ontologies are more domain specific in nature therefore formal model development is not obvious with it. To standardize the semantic layering in cybersecurity, Obrst et al., 2012 [71] developed a cyber ontology architecture based on an initial malware ontology to construct a semantic model in cyber security domain. This architecture is composed of three levels domain ontologies, middle-level and upper level ontologies. This ontology consists of less number of malware types which are not considering new generation malware families also behavioural aspect of malware needs to be included in this ontology.

Table 3.6: Ontology in Malware Detection (Cont.)

Ontology-based Frameworks		
Paper	Strengths	Weaknesses
J. N. O.S and M. S. B. S, 2021 [68]	• Ontology-based intelligent model for mobile malware detection was built. • Based on the static features an Apps Feature Ontology (AFO) was built.	Framework is based on static features from the app therefore inclusion of new malware features is a limitation.
Wang et al., 2017 [69]	Ontology was used to develop a model which can detect attack profile of a malware and prevent network threats.	This method is not useful to determine targeted attack which can compromise mobile security.

Table 3.7: Ontology in Malware Detection (Cont.)

Ontology-based Frameworks		
Paper	Strengths	Weaknesses
Huang et al., 2011 [70]	Integration of semantic technologies and computational intelligence methods was done by integrating the Fuzzy Ontologies and Fuzzy Markup Language (FML).	But fuzzy ontologies are more domain specific in nature therefore formal model development is not obvious with it.
Obrst et al., 2012 [71]	A cyber ontology architecture was developed based on an initial malware ontology to construct a semantic model in cyber security domain.	Consists of less number of malware types which are not considering new generation malware families also behavioural aspect of malware needs to be included.
W. Han, 2020 [72]	An ontology-based knowledge framework was built to represent malicious behaviour.	The behaviours were defined in an abstract form which are not enough dynamic in nature.

An automated system of malware analysis and evasion detection was proposed in [73]. In this study AEMS (Analysis Evasion Malware Sandbox) was developed using ontologies and MAEC (Malware Attribute Enumeration and Characterization). This architecture consists of detection of malware evasion technique and a countermeasure to force the malware for expressing its complete behaviour. A more recent approach was developed in [72] where a dynamic malware behaviour detection was established based on system calls by studying APT malware. Here an ontology-based knowledge framework was built to represent malicious behaviour. In this ontology the behaviours were defined in an abstract form which are not enough dynamic in nature.

CRATELO [1] is a structurally rich ontology but not designed for behavioural detection. The UCO ontology [63], aligns heterogeneous cybersecurity knowledge sources but not dealing with system events (API call traces), that are critical for detecting sophisticated malware.

Taiwan Malware Analysis Net (TWMAN) [64] and Jasiul et al., [65] proposed static and rule-based techniques respectively. The reliance on manually de-

fined rules limits scalability and not suitable to accommodate behavioural features, a core feature of metamorphic malware.

In mobile domain, models developed in [66–68] incorporated machine learning techniques along with ontologies. However, dynamic enrichment and real-time integration of new cyber incident data are still not properly explored. In addition, establishment of deep semantic links between malware exhibited behaviours and higher-level datasets is a new avenue to contribute.

All these studies collectively showcase a consistent trend of work. Despite the increasing advancement of ontology-based frameworks in cybersecurity, most remain fundamentally constrained by static modeling approaches, limited behavioural articulation and inadequate integration with data driven learning techniques. Rule-based and taxonomy-based frameworks often fail to capture the dynamic, evasive nature of modern malware generations, while ontology and machine learning systems are not necessarily capable of semantic reasoning and automated enrichment. This precise gap enhances the need for an ontology framework that is not only semantically robust but also behaviourally adaptive and interoperable with deep learning pipelines. This thesis addresses that gap by implementing a dynamic OWL-based malware behaviour ontology which is capable of modeling API call sequences, supporting ontology-oriented programming, and integrating with deep learning-based malware image classification. The framework further proposes RDF-based knowledge graph construction offering an interpretable, extensible, and resilient approach for malware detection.

3.2 Machine Learning based Malware Detection

Machine learning has become a fundamental key for modern malware detection, offering robust alternatives rather than traditional signature-based methods. A machine learning based malware detection approach was developed in [74] based on Random Forest classifier. This classifier was used to predict the cyber threats and identify a given data benign or malicious depending upon the behaviour. This approach was developed on Linux environment. Though several new generation Windows malware are also threatening the end-users and this paper is using only the Random Forest classifier

but there exist other advanced machine learning methods to be considered in future.

Another Machine Learning based malware detection method, Opem [75] was developed to integrate both static and dynamic techniques of malware detection. This hybrid method is promising for unknown malware detection. But static-dynamic approach may be a time consuming process where adversaries are getting more advanced in recent days. Likewise, in [76] an anomaly based malware detection method was developed and one-class Support Vector Machine was used for classification. The proposed method produced very high accuracy but the experiment was carried out on Windows/x86 platform and older versions of Internet Explorer and Adobe PDF reader. Nowadays latest versions of Operating Systems are becoming more vulnerable to the cyber attacks.

Table 3.8: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Joshi et al., 2018 [74]	• Random Forest classifier. • Classifier was used to predict the cyber threats and identify a given data is benign or malicious. • Focuses on process list data structure from a virtual environment.	• Focuses solely in Linux environments. • List data structures may not capture all type of nuanced behavior. • Focuses on RF, lacks a comparative study.
Santos et al., 2013 [75]	• Opem, a hybrid approach was developed. • By utilizing both opcode sequences (static features) and dynamic behavior analysis this method is promising for unknown malware detection.	• Acknowledges that dynamic analysis techniques can still be evaded by sophisticated malware.
Tang et al., 2014 [76]	• Introduces a new class of malware detectors that use anomaly-based detection. • By using hardware features the paper proposes a detection method.	• Experiment was carried out on Windows/x86 platform and older versions of Internet Explorer and Adobe PDF reader. • Acknowledges that sophisticated attackers can still bypass detection.

Table 3.9: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Narudin et al., 2016 [77]	• Five ML classifiers have been used. • Research emphasizes feature selection and extraction, identifying 11 key features from network traffic data.	• Focuses only on network traffic, which might limit its effectiveness. • Dataset is quite outdated. The private dataset consists of only 30 samples.
Peiravian and Zhu, 2013 [78]	• Combines permission requests and API calls. • Evaluates its approach on a balanced dataset of over 2,500 applications. • Can identify unknown or new malware.	• Static analysis has limitations in detecting more sophisticated malware. • Relies on a fixed set of permissions and API calls. Permissions can be misleading.
Sanz et al., 2013 [79]	• Android permission analysis as primary feature. • Applies a variety of ML classifiers to the permission-based feature set.	• Relies solely on static permissions. • Focuses on permissions alone overlooks behavioral indicators.

Table 3.10: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Wu et al., 2016 [80]	• Android malware detection based on dataflow APIs. • Reduces computational overhead by 40%. • Improves the KNN model by a weighted API-based Mahalanobis distance.	• Static analysis (dataflow API), which may struggle to detect more sophisticated malware. • Malware that doesn't use these specific APIs might evade detection.
Wang and Wang, 2015 [81]	• Developed by focusing on behavioral signatures. • Employment of SVM classifier enhances accuracy. • Key strength is the integration of a digital vaccine (DV) concept.	• Acknowledges that the accuracy in detecting entirely new or significantly altered malware variants is still limited. • Focuses on SVMs, does not explore advanced ML methods.
Saracino et al., 2016 [82]	• MADAM: a multi-level approach to detect malware. • MADAM implements a behaviour-based approach of detection.	• Collects signatures from known malware. • Relies on predefined behavioral patterns. • Effectiveness can vary based on the Android device and version.

Table 3.11: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Li et al., 2018 [83]	• SigPID reduces the number of permissions needed for malware detection by 84%. • Works effectively across a wide range of supervised learning algorithms (67 tested).	• Relies on permissions as the main feature. • Effectiveness in detecting unknown or new malware might be limited if the patterns or behaviors shift significantly over time.
Chen et al., 2016 [84]	• Combination of contributed features. • Uses a "streaminglized" process, which is suitable to handle large-scale datasets.	• Features were selected randomly from a large dataset. • Accuracy of detection could be improved.
Yerima et al., 2014 [85]	• Proactive approach for android malware detection based on parallel ML classifiers. • Proactive detection of zero-day malware. • Static analysis for feature extraction. • Selected classifiers are computationally efficient.	• Focuses on static analysis which may miss certain malware behaviors. • Uses a dataset from McAfee's internal repository, may not fully represent the diversity of Android malware found in the wild.

Table 3.12: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Rieck et al., 2011 [86]	• Clustering and classification used for behaviour analysis. • Incremental approach reduces both memory and run-time. • Focusing on dynamic behavior analysis rather than static file analysis.	• In recent times malware writers are being more advanced to develop sandbox evasion techniques. • System rejects clusters with fewer than a certain number of samples.
Firdausi et al., 2010 [87]	• Evaluates five different ML classifiers. • Dynamic analysis by monitoring malware in a sandbox. • Feature selection using Correlation-based Feature Selection (CFS).	• Uses a relatively small dataset, consisting of 220 unique malware samples and 250 benign software samples. • Relying on a single sandbox environment may limit the detection.
Sayadi et al., 2018 [88]	• Lightweight, hardware-assisted detection (HMD) approach. • Effective feature reduction process using PCA. • Key strength is it allows for monitoring low-level hardware events.	• Focuses on only three malware classes. • Does not compare results with software based detection.

Table 3.13: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Shamili et al., 2010 [89]	• A distributed SVM model. • System is lightweight in terms of bandwidth usage and computational load. • Framework is shown to train quickly while maintaining reliable performance.	• Focuses on well known Trojans: Viver, Beselo. • Reliance on a SVM model, may not scale well for complex, multi-class detection. • Does not provide detailed comparisons with other detection methods.
Ye et al., 2008 [90]	• Objective-Oriented Association (OOA) mining technique for malware detection. • OOA_Fast_FP-Growth algorithm, an enhancement over the traditional FP-Growth algorithm.	• Focuses on Windows PE files, which limits its applicability to other file formats or OS. • Only offers binary classification, does not classify different types of malware.

Table 3.14: Machine Learning based Malware Detection

Machine Learning based Works		
Paper	Strengths	Weaknesses
Ye et al., 2007 [91]	• Objective-Oriented Association (OOA) mining technique introduced. • Composed of three major modules: a PE parser, an OOA rule generator, and a rule-based classifier.	• Designed specifically for Windows PE files. • Uses static analysis of API sequences, which may not capture the full range of advanced malware behaviors.
Hu and Tan, 2017 [92]	• MalGAN: GAN-based algorithm developed to generate black-box malware detector. • Attackers do not have direct access to the model used in black-box model.	• Assumes that attacker knows features, even though they are unaware of the model parameters. • Acknowledges that training of MalGAN, is prone to instability.

Several machine learning approaches have been used to overcome the difficulties deal with hardware limitations [76] and detect Android malware [77], [93], [78], [79], [80], [81], [82] due to limited resource availability in mobile devices.

Due to increasing use of Android devices there is a considerable rising of cyber hacking incidents that generates malicious applications. Five machine learning classifiers have been used in [77] to evaluate anomaly-based malware detection. 49 malware families and 1000 Android malware samples were considered as dataset. This experiment extracts features from network traffic of mobile applications but in a network traffic data may come from different sources like routers, firewalls etc. and in different forms which makes it difficult to analyze the data accurately in a given time. Data that are passing through the network traffic may be spoofed by the attackers and may produce false alarms. Recognizing the context-sensitive nature of mobile platforms, another Machine Learning based Android malware detection approach was

found in [93] to mitigate context-sensitive nature of smartphones. An automation framework STREAM (System for Automatically Training and Evaluating Android Malware Classifiers) was developed for profiling malware and training classifiers.

A significant body of work has focused on feature-based detection techniques, particularly those using permissions and API calls, which were found in [78] and [79]. Detection of malicious apps on mobile devices was found in [78] based on permission and API (Application Programming Interface) calls. Open source nature of Android devices and increasing number of mobile apps are the significant reason of vulnerable nature of mobile devices to malware attacks. Machine learning classifiers have been used to analyze and classify the Android apps whether a particular app is malicious or not. Similarly, PUMA (Permission Usage to Detect Malware in Android) is another permission based machine learning approach which was developed in [79] to detect Android malicious applications by investigating the extracted permissions from the apps. The features were extracted from .apk (Android Application Package) files. Both of the approaches in [78] and [79] are accurate in malware detection but these are not dynamic analysis of malware.

Another Android malware detection approach was developed in [80] based on dataflow APIs by using machine learning classifiers. Though analysis of dataflow API is a static method and this paper proposed semantic based approaches for classification and behaviour analysis in future.

Support Vector Machines (SVMs) have been favored in malware detection research for their effectiveness in handling high-dimensional feature spaces. Wang and Wang introduced a malware behaviour detection using SVMs in [81]. This is an automatic malware detection approach which was developed by training SVM classifier for mobile security. A digital vaccine was proposed to prevent the malware infection from hosts. This paper also used the suspicious .apk files for behavioural analysis but SVM requires longer training time and it may consume large memory when the dataset is huge with more number of features. Another SVM based hybrid model classifier was developed by using static analysis and machine learning for Android malware detection in [94]. In this experiment the SVM accuracy was improved by using fusion logics after combining two decision models.

Several host- and anomaly-based frameworks have also been proposed.

MADAM (Multi-Level Anomaly Detector for Android Malware) a host-based and anomaly-based malware detector was developed in [82] to detect misbehaviour of Android apps. MADAM implements a behaviour-based approach of malware detection but collects signatures from known malware behaviours. Though being a host-based system it may not see the network level threats which attack the host. To refine classification further, Li et al. (2018) [83] proposed a Significant Permission IDentification (SigPID) an Android malware detection system. To distinguish benign and malicious apps most significant permissions were used. This system uses ML classifiers to classify malware and benign apps. This paper used a three level data pruning to identify the significant permissions. But data pruning may lead to loss of accuracy and increase training period.

Another machine learning based Android malware detection system is StormDroid [84] was developed by Chen et al. (2016). Contributed features were combined to develop a streaminglized malware detection framework. Though in this work, features were selected randomly from a large dataset and accuracy of the detection approach could be improved. In contrast, a novel and proactive approach for Android malware detection based on parallel ML classifiers was developed in [85]. This approach was tested on real-world samples to identify malware and benign applications. This paper proposed a parallel combination of five classifiers which made the process cost effective but static app features may provide limited information and classification with diverse ML classifiers makes the process time consuming.

Sandbox-based behavioural detection was reimplemented by Rieck et al. [86] and Firdausi et al. [87], who used machine learning classifiers to evaluate behavioural traces extracted from a sandbox environment. Whatever, as malware writers increasingly adopt sandbox evasion techniques, the long-term viability of these approaches is uncertain.

In [86], clustering and classification techniques were used for malware behaviour analysis where malware behaviours were monitored using sandbox environment. But in recent times malware writers are being more advanced to develop sandbox evasion techniques, they can manipulate their codes to hide the behaviour of malware. Different machine learning classifiers were tested in [87] based on their performance. In this paper also the behaviour of malware was collected using sandbox environment.

Beyond traditional detection methods, a hardware assisted malware detection approach was developed in [88]. It's a machine learning based method for embedded and required less number of features but this approach detects only the runtime malware. A distributed SVM-based approach for malware detection on mobile devices was presented in [89]. But noises and outliers can affect the distributed SVM-based approach. Though both of these approaches may struggle with resource constraints.

An intelligent malware detection system based on association mining was developed in [90] and object oriented association mining was developed in [91]. Both of [90] and [91] need to identify the features and families of malware more precisely.

Finally, MalGAN, a generative adversarial network (GAN) based algorithm was developed in [92] to generate a black-box malware detector. In this experiment, same probability distribution was followed by both the train and test dataset but in a complex scenario malware writers may modify the probability distribution to disable the machine learning based detection.

3.3 Deep Learning based Malware Detection

This section has described several deep learning frameworks for malware detection and analysis:

3.3.1 Behaviour-based Malware Detection

Table 3.15: Behaviour-based Malware Detection

Deep Learning Works		
Paper	Strengths	Weaknesses
Tobiyama et al., 2016 [95]	• Two-stage deep neural network (DNN): both RNN and CNN. • Focuses on the behavior of processes. • Captures temporal and structural patterns. • Can mitigate damage after an attack.	• Dataset has 81 malware process log files and 69 benign process log files, which is much smaller. • Feature extraction process relies on fixed-length sequences of process behavior.
Jeon and Moon, 2020 [96]	• Combines CNNs and RNNs for detection using opcode sequences. • Integrates static and dynamic analysis.	• Dataset consists of only 1000 benign files and 1000 samples. • Focuses on opcode sequences may not fully capture other important characteristics.

Behaviour based malware detection uses a number of techniques like machine learning, AI, graph-based analysis, Deep Learning etc. to execute suspicious behaviour of a software or an Android application. In [95] a malware detection method was proposed using process behaviour to discover an infected terminal rather than network traffic. Process behaviour was recorded in the form of API call sequences and Long Short-Term Memory (LSTM) was used to construct feature extractor for API call based language modeling. Two types of Deep Neural Network (DNN) were used: Recurrent Neural Network (RNN) to extract feature and to train them. After training feature vectors were extracted and validated with trained RNN. After that, converted to an image. Convolutional Neural Network (CNN) was used to classify these images based

on local features to identify malware and benign processes. But this work is highly dependent on API calls which leaves it vulnerable to anti-analysis techniques such as API obfuscation and additional image conversion layer may be time consuming.

Similarly, a Convolutional Recurrent Neural Network (CRNN) based malware detection method was proposed in [96]. Here a mathematical method was used to extract opcode sequences without execution. A convolutional autoencoder was used to compress the opcode sequence in much shorter than the original length. A Dynamic Recurrent Neural Network (DRNN) classifier was used in the back-end to classify the compressed code generated by the autoencoder. A CNN and RNN based approach may achieve higher accuracy but it may not detect smaller objects from a dataset. Also DRNN could be a complex model which poses challenges in large-scale deployments. SMASH [97] is a multi-feature malware detection method which combines software and hardware features. Software features were extracted from the malware samples in the form of API call sequence using dynamic sandbox. API call sequence features were used to detect malware with a multi-layer Bi-GRU network. The hardware feature was extracted in the form of grayscale image by converting the memory dump data. Memory dump file was used in integrated computing to detect whether a process is malicious or not. BiGRU network is faster than LSTM but it has two gated recurrent units so it may be costly and may limit its real-time applicability.

A straightforward Android malware detection method was proposed in [98] where an Android application was presented in the form of bytecode file. From the Android package (.apk file) the classes.dex was extracted to acquire function codes and behaviours of the application. Here a Visualization step was involved to convert the classes.dex file into an image. After that, CNN was used to analyze the features of bytecode images for detecting malware. This method is effective in handling static bytecode structures but limited in its ability to capture dynamic code loading or network-based payload.

Table 3.16: Behaviour-based Malware Detection (Cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Dai et al., 2019 [97]	• A combination of software and hardware features. • Method shows strong anti-evasion performance. • Incorporates advanced neural networks, such as BiGRU, VGG19.	• Sandbox can be bypassed by malware that detects sandbox environment. • Focused on detecting malware that runs as executables.
Ding et al., 2020 [98]	• Transformed Android bytecode files into 2-D bytecode images. • Resilient to malware with packing, encryption, or obfuscation techniques. • Automates feature extraction by CNNs. • Poses a challenge for traditional static analysis methods.	• Detection method is only CNN based. • Dataset contains 3962 malware samples and 1000 benign samples, which is smaller. • Accuracy decreases as percentage of packed samples increases. • Does not explore hybrid approaches.

Table 3.17: Behaviour-based Malware Detection (Cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Xiao et al., 2019 [99] and Hardy et al., [100]	• Constructed behavior graphs from API calls and OS resources. • SAEs used to acquire high level feature extraction. • Integrates SAEs with ML classifiers. • Focused on protecting IoT devices. • More resistant to evasion techniques.	• Dataset contains only 1760 samples (880 malware and 880 benign), which is smaller. • Cloud-based implementation introduces issues related to data transmission delays and privacy concerns.

BDLF is a behavior-based deep learning framework was proposed in [99] for IoT (Internet of Things) environment. Here API calls were extracted to represent malware behaviour and provide information about the behaviour graphs. Stacked AutoEncoders (SAEs) were used to acquire high level feature information of malware behaviours. Another SAE-based approach was developed in [100]. Here, a deep learning based intelligent malware detection approach using Stacked AutoEncoders (SAEs) model was developed based on Windows Application Programming Interface (API) calls.

Both SAE-based approaches are effective in feature abstraction capability but were resource-intensive and vulnerable to overfitting.

The above mentioned systems are mostly relied API calls based detection. But APIs are much prone to some other security concerns and one must need to understand the structure of the APIs before using it. Some of the detection methods are only CNN based which may need to be integrated with other deep learning based methods for better advancement.

3.3.2 Image-based Malware Detection

Table 3.18: Image-based Malware Detection

Deep Learning Works		
Paper	Strengths	Weaknesses
Venkatraman et al., 2019 [101]	• Combines deep learning and image analysis. • Effectively utilizes both supervised and unsupervised learning. • Addresses the challenge of detecting obfuscated malware.	• Pre-trained CNN used for feature extraction. • The datasets used in the study, although large, might not represent the full diversity of malware.
K. He and D.-S. Kim, 2019 [102]	• Spatial Pyramid Pooling (SPP) allows the CNN to handle input images of varying sizes. • Provides valuable insights into the limitations and potential of SPP layers within CNNs.	• CNN has only 3 layers. • Uses a dataset with 906 malicious files and 1,776 benign files, it is relatively small. • Acknowledges SPP layer did not perform well as expected.

Table 3.19: Image-based Malware Detection

Deep Learning Works		
Paper	Strengths	Weaknesses
Darem et al., 2021 [103]	• Semi-supervised approach to integrate deep learning, feature engineering, and image processing. • Targets obfuscated malware. • Integration of XGBoost and CNN.	• Needs to be implemented on colored images of malware. • Dataset may not fully represent the diversity of malware.
Ni et al., 2018 [104]	• Resilient to obfuscation techniques. • Does not require disassembly or code execution. • Tested on a large dataset.	• Relies heavily on visual similarities. • Acknowledges confusion between certain families which are visually similar but functionally different.

Due to exploration of Deep Learning techniques and Computer Vision Image-based Malware Detection is growing popularity because image files are mostly benign, code visualization friendly and informative.

A hybrid deep learning based method was developed in [101] where detection and classification of obfuscated malware were proposed. Visualization of images of malware and benign datasets was demonstrated to identify their distinct behaviour patterns. In this hybrid approach self-learning system was used to detect known and unknown malware samples. CNN and bi-directional networks were used for classification of malware. Similarly, another CNN-based malware detection method was developed in [102] to deal with redundant API injection. In this method input malware files were transformed into images to train the CNN and Spatial Pyramid Pooling (SPP) was used to cope with the malware images which are extremely varying in size.

In [101] and [102] CNN-based approaches have been developed for de-

tection and classification. Though a pre-trained CNN was used for feature extraction in [101] and in [102] the CNN has only 3 layers for classification.

Other studies integrated opcode analysis with image conversion, such as a semi-supervised approach for obfuscated malware identification was developed in [103] where few approaches like OpCode analysis, image conversion, feature engineering and deep learning were integrated. Malware OpCodes were converted into grayscale images for classification and visualization. Though in this work, images were generated using another algorithm developed in [105] and this work needs to be implemented on colored images of malware.

SimHash-based methods were developed in [104] and [106]. In [104], a MCSC (Malware Classification using SimHash and CNN) was developed to convert opcode into grayscale images. After feature extraction, fingerprint images of malwares were generated in visualization phase and trained with CNN for identification. These researchers used SimHash to generate gray scale images and to improve classification accuracy but SimHash may be highly sensitive to specific features and may not identify subtle differences between two images as it may count two very similar (not identical) images as completely different. Another SimHash-based Anroid malware detection approach was developed in [106]. Codes were extracted from an apk file and words were digitized to form an image. Convolutional Neural Network was used to check the apk file whether it is malicious or not. This CNN architecture could be improved to achieve better accuracy.

Table 3.20: Image-based Malware Detection (Cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Yen and Sun, 2019 [106]	• Detect malware by visualizing the importance of words in code. • Handles code obfuscation effectively. • Leverages text mining to convert code into images. • CNN was used to detect malware.	• Conducted on a relatively small dataset of 1,440 APK files. • CNN accuracy could be improved. • Overlooked features like API calls, dynamic behaviors.
Vinayakumar et al., 2019 [107]	• ScaleMalNet integrates static and dynamic analysis with deep learning and image processing. • Tested on both public and private datasets. • Both CNN and LSTM were used. • Two-stage detection.	• Primarily focuses on malware affecting Windows systems, with less consideration given to other platforms, like Android, macOS, or Linux.

Table 3.21: Image-based Malware Detection (Cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Cui et al., 2018 [108]	• CNNs allows for automatic feature extraction. • Addresses data imbalance among different malware families. • Effective in handling obfuscated code.	• Success depends on quality and consistency of image representation. • Does not explore the benefits of hybrid approaches.
Naeem, Hamad, et al. [109]	• Hybrid model combining image visualization and DCNN. • Optimized DCNN is computationally efficient. • Resistant to polymorphic malware obfuscation. • Used color image transformation to extract features.	• Transforming malware binaries into images may limit the approach's effectiveness to deal with sophisticated malware. • Does not provide detailed comparison with deep learning techniques.

Later research explored hybrid architectures to improve robustness. For example, a scalable deep learning based framework ScaleMalNet [107] was developed to overcome the limitations of conventional machine learning algorithm and to develop an effective solution for zero-day attack. This framework can process large malware samples. Here, a novel image processing technique was proposed for classification of malware. Both CNN and LSTM were used to develop a hybrid image processing framework which can properly detect packed malware samples. Though the CNN is 1-dimensional in this experiment and the 1D-CNN was combined with LSTM, which may require more training data with high computational cost and large memory. An idea of hybridization was also proposed in [109] where a cost-effective and hybrid malware detection approach was implemented for Industrial Internet of Things (MD-IIOT). In this hybrid approach color image visualization and deep

convolution neural network (DCNN) were used for in-depth malware analysis.

Further advanced the hybrid architecture was developed in [110] by integrating both static and dynamic perspectives, in this study a visualization and deep learning based Windows malware detection approach was proposed. This hybrid approach is an integration of visualization and convolutional neural network. In a combination of static and dynamic analysis this paper uses neural networks for detection and vectorize the input in two approaches: feature-based and image-based. In static analysis malware was visualized in RGB image to acquire more information than grayscale images. In dynamic analysis API calls were represented to visualize behaviour of a program. Cuckoo Sandbox was used to visualize the APIs of a program and to convert it into an image. Both static and dynamic images were combined in a single image to carry out the hybrid visualization. Though models of this paper didn't perform well on test dataset of virussign.com[1] and older malware samples. This is also a sandbox detection method which doesn't allow network interactions as some malware variants require network environments to exhibit their malicious behaviour.

Along with the hybrid methods, other researchers explored more streamlined CNN-based frameworks. As malicious codes are growing rapidly and becoming main threats in the field of cyber security, a novel deep learning based approach for malicious code detection was developed in [108]. Codes were converted into grayscale images and CNN was used to extract features from the images for classification and identification. Though converting the images into a fixed size was a task in this work.

To extend the image-based paradigm into mobile platforms, an image-based mobile malware detection method was proposed in [111]. Mobile applications were represented in grayscale images. A fine-grained method was proposed to distinguish malware and legitimate mobile samples and to identify the family variant. Though, in this experiment size of train and test dataset split is 50:50 and older samples were used in training phase.

[1] https://www.virussign.com/

Table 3.22: Image-based Malware Detection (Cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Huang et al., 2021 [110]	• CNN based on VGG16. • Combines both static and dynamic visualization techniques. • In static analysis malware was visualized in RGB image. • In dynamic analysis API calls were represented to visualize behaviour of a program.	• Didn't perform well on test dataset of virussign.com[2] and older malware samples. • Struggles with certain malware, like QQ password stealer trojans. • May require regular updates. • Drop in accuracy on unknown malware.
Mercaldo and Santone, 2020 [111]	• Platform-independent. • Validated using 50,000 Android applications and 230 Apple applications. • Fine-grained method was proposed to distinguish malware and legitimate mobile samples.	• Relies on visual similarity. • Size of train and test split is 50:50, older samples were used in training. • Proposed system relies purely on static analysis.

For image-based malware detection, CNNs have been widely applied [101], [102], often in combination with methods such as LSTMs [107], SimHash [104], [106] or static–dynamic hybrid [110] visualization. All these approaches improve robustness against obfuscation and packing, they are often computationally expensive and may not generalize well across diverse or evolving malware families. Basic CNN-only methods are much faster but encounter difficulties with resizing and limited feature representation. In general, current methods emphasize a trade-off between efficiency and accuracy, indicating that more sophisticated deep learning architectures such as attention-based CNNs, residual networks, or ensemble models might produce faster, scalable, and more accurate solutions for real-world malware detection.

3.3.3 Using Opcode Sequences

Table 3.23: Using Opcode Sequences

Deep Learning Works		
Paper	Strengths	Weaknesses
Zhao et al., 2018 [112]	• CNNs were used in learning patterns. • Introduces k-max pooling in CNN architecture. • An effective feature extraction method by focusing on Dalvik opcode sequences.	• The dataset is relatively smaller with approx. 2,400 samples, around 1,200 benign and 1,200 malicious samples. • Compares its approach with traditional ML methods but does not compare with DL models.

Table 3.24: Using Opcode Sequences (cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Yan et al., 2018 [113]	• Ensemble method: combines CNN and LSTM. • Evaluation is conducted with 21,736 malware samples and 20,650 benign samples. • Addresses challenges like gradient vanishing problem in LSTM.	• Dataset comprises nearly equal distribution of malware and benign samples. • Compares with some traditional methods. • Does not benchmark against advanced DL models.
Pektaş and Acarman, 2020 [114]	• Combines static and dynamic analysis. • Used instruction call graphs and opcodes. • CNN and LSTM were combined for classification. • Utilizes a dataset consisting of 24,650 malicious samples and 25,000 benign samples. • Perform comprehensive hyperparameter tuning.	• Accuracy of the model could be improved. • Focuses on CNN and LSTM models without comparing them to other advanced deep learning architectures.

Table 3.25: Using Opcode Sequences (cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
HaddadPajouh et al., 2018 [115]	• Innovative use of RNNs, specifically LSTMs. • Addresses unique challenges of securing IoT networks. • Can be applied to large-scale IoT networks.	• Does not provide detailed information about the diversity of the dataset. • Needs more discussion about feature engineering.
Ding et al., 2016 [116]	• DBNs are well-suited for detecting patterns. • Opcodes can provide detailed insights into the behavior of both benign and malicious software. • By focusing on opcodes, the paper addresses a critical aspect of malware analysis.	• Focuses on experimental results within a controlled environment. • DBNs, with multiple layers, can be computationally expensive. • Does not provide detailed information about the diversity of the dataset.

Table 3.26: Using Opcode Sequences (cont.)

Deep Learning Works		
Paper	Strengths	Weaknesses
Yazdinejad et al., 2020 [117]	• Cryptocurrency malware detection with Deep Recurrent Neural Network (RNN), specifically LSTM. • LSTMs are effective in handling sequential data. • Used an embedding technique to address the issue of computational complexity.	• Dataset comprises only 500 cryptocurrency malware samples and 200 benign samples. • LSTM was not compared against deep learning models, such as CNNs or GANs.

For mobile and lightweight environments, opcode sequences have become an important feature for malware detection. For example, in [112] an optimized convolutional neural network was trained a number of times by the extracted opcode sequences from the apk files. The opcode sequence features were extracted using static analysis and the CNN was used to learn from the raw opcode sequences to classify and detect malware more accurately. K-max pooling was used with the CNN to improve the classification and detection performance of the system.

CNN and LSTM based methods were developed in [113] and [114]. MalNet is a deep neural network based malware detection method which was developed using both CNN and LSTM [113]. Here CNN and LSTM were used to develop automatic feature learning from raw data. Grayscale images were generated from the malware files and opcode sequences were extracted using decompilation tool. This paper approached to solve some practical problems and used stacking ensemble method to improve detection accuracy. Similarly, in [114] CNN and LSTM were combined for classification and features were extracted from the instruction call sequences. This paper examines malicious execution paths and benign execution paths through code analysis and trained deep learning model from scratch and a grid search method was applied to evaluate the combination of parameters also to discover the best

combination of hyper-parameters (i.e. varying network configuration).

Opcode-based detection methods are not only limited to mobile malware. Internet of Things (IoT) devices are being main target of the malware writers because of their openness and vulnerability. A Recurrent Neural Network based malware hunting approach was proposed in [115] for IoT platform. This paper analyzed the OpCodes of the ARM-based IoT malwares and evaluates their trained model with three LSTM configurations. The second configuration shows highest accuracy to detect new malware variants with the 10-fold cross validation. To extend beyond the shallow models, a Deep Belief Network (DBN)-based malware detection approach was developed in [116] to overcome the limitations of shallow neural networks. Malware was represented as opcode sequences and the detection system was developed using DBN. As DBN consists of several Restricted Boltzmann Machines (RBMs) so unlabeled data was trained using RBMs.

Opcode sequences have also been explored in the context of financial threats. Increasing popularity of Cyptocurrency is leveraging more vulnerability for malware developers. A Deep Recurrent Neural Network (RNN) based cryptocurrency malware detection approach was proposed in [117] for Windows platform. In this method RNN was used to analyze Windows Opcodes as a case study. For training of dataset five different LSTM structures were used and the the model was evaluated by 10-fold cross-validation.

These research works are based on, CNN, LSTM and RNN. CNNs are efficient in capturing local opcode patterns and LSTMs and RNNs can model sequential dependencies, making them suitable for complex execution paths. But LSTM may take longer time to train and use more memory for computation also RNN suffers from vanishing gradient problems and other computational complexities. In case of opcode based detection, one must check whether the opcodes are of same length or not because a single opcode may be meaningless.

3.3.4 Intelligent Framework for Malware Detection

DL4MD is a deep learning based intelligent malware detection framework which was developed to overcome the limitations of shallow learning methods [100]. This work extracted API calls from Windows OS to use as input

in stacked AutoEncoders (SAEs) for malware detection. This is the first deep learning based work until 2016 to use stacked AutoEncoders (SAEs) in Windows API platform for detecting malware in industrial application. For evaluation of this paper samples were used from Comodo Cloud Security Center. Almost similar work was found in [118] where a heterogeneous deep learning based work (DeepAM) was developed by stacked up AutoEncoder and multilayer restricted Boltzmann machines (RBMs). The deep learning model was trained layer by layer followed by fine-tuning of the parameters. This model can detect unknown malware using associative memory and allows unlabeled data samples (malware or benign) for feature learning in training phase. For the comparison this work used real samples from the Comodo Cloud Security Center.

Deep4MalDroid is a deep learning based dynamic analysis method which was developed in [119] to propose a mitigation technique for packaging and

Table 3.27: Intelligent Framework for Malware Detection

Stacked AutoEncoder (SAE) Methods		
Paper	Strengths	Weaknesses
Hardy et al., 2016 [100]	• DL4MD outperforms traditional shallow learning methods. • Explores the use of Stacked AutoEncoders (SAEs) for detection. • Leverages both unsupervised pre-training and supervised backpropagation.	• Models with multiple hidden layers, require significant computational resources. • The model relies heavily on Windows API calls for feature extraction.

Table 3.28: Intelligent Framework for Malware Detection (cont.)

Stacked AutoEncoder (SAE) Methods		
Paper	Strengths	Weaknesses
Ye et al., 2018 [118]	• DeepAM: heterogeneous deep learning model. • Utilize both labeled and unlabeled file samples. • Outperforms methods, like SAEs and DBN. • Shows improvements over shallow learning-based methods.	• The model is relatively complex. • May require increased computational costs and longer training times. • There might be concerns regarding the generalization to different types of malware.
Hou et al., 2016 [119]	• A novel method: “Component Traversal" was introduced. • Designed for dynamic behavior extraction. • Constructs weighted directed graphs. • Explores the use of SAEs, to learn patterns.	• Dealing with large datasets could be a challenge. • The method was tested on a set of 3,000 Android apps, this may not fully represent the wide variety of malware.

Table 3.29: Intelligent Framework for Malware Detection 3

Stacked AutoEncoder (SAE) Methods		
Paper	Strengths	Weaknesses
Zhu et al., 2020 [120]	• SEDMDroid: a novel stacking ensemble framework. • Combines static analysis and deep learning. • Emphasizes both individual accuracy and diversity among the base learners. • Uses a multi-level static feature extraction process.	• Relies on static analysis for feature extraction. • Stacking ensemble approach is powerful but can be computationally intensive.

obfuscation of malware signatures. This paper proposed a novel behaviour extraction method to use Linux kernel system calls rather than API call based behaviour extraction also an automatic Android app code execution process named Component Traversal to overcome the limitations of Android Testing Tool (ADT). From the extracted system calls a weighted directed graph was constructed and a deep learning based Stacked AutoEncoders (SAEs) model was applied on the graph features to detect unknown malware.

SEDMDroid is another Android malware detection method [120] based on a stacking ensemble method. This approach used Principal Component Analysis (PCA), bootstrapping and deep learning to improve individual accuracy and diversity. To get the prediction result Support Vector Machine (SVM) was used in the fusion stage and Multi-Layer Perception (MLP) was used to train the base learner.

These experiments are based on stacked AutoEncoders which are useful to identify complex relationships but they use much resources and time for training. Computation is also expensive with autoencoders. Moreover, dynamic methods such as Deep4MalDroid [119] is highly dependent on controlled execution environments and ensemble methods like SEDMDroid [120]

add additional computational overhead.

3.4 Continual Learning based Malware Detection

In the ever-evolving domain of cybersecurity, malware does not just mutate—it multiplies, evolves, and emerges in new forms. Previous machine learning models, which are trained once and applied forever, fail in such a scenario. This created a turning point towards Continual Learning (CL) - a method designed to allow models to learn step by step, retain past knowledge, and learn new threats without having to retrain from scratch.

Deep learning with continual learning (CL) is an advanced AI approach that enables models to learn new tasks while retaining previously acquired knowledge, preventing catastrophic forgetting. This is crucial for real-world applications where data evolves over time, such as malware detection, cybersecurity, and autonomous systems. In cybersecurity and malware classification, continual learning can be highly effective. For example, when new malware variants emerge, a CL-enabled model can adapt without forgetting older threats, ensuring robust and up-to-date threat detection. The Chapter 6 of this thesis illustrates the experiments with Continual Learning based Deep Learning.

But, the road to robust continuous malware detection has not been smooth. The following literature study describes the key breakthroughs and how our work advances the field further:

The most important inflection point was the study in [121] by Rahman *et al.* (2022). Their large-scale evaluation on 11 leading continual learning approaches (e.g., EWC, LwF, GEM, iCaRL, etc.) uncovered a disturbing trend: several of them perform worse than even simple joint training baselines on malware datasets. Malware classification is especially problematic: changing data distributions, visually similar but structurally different classes and extreme class imbalance. This article not only criticized — it established the ground upon which malware-specific continuous learning strategies must be developed.

The following paper introduced MalCL [122] (Park et al., 2025) — a GAN-generation-based approach that tackled catastrophic forgetting head-on. Instead of memorizing past data, MalCL generated realistic synthetic malware

images with Generative Adversarial Networks (GANs) that have feature matching loss, in a manner that enabled “replay" past data from scratch. The generative replay potentially improved class-incremental learning performance on Windows and Android malware datasets. However, GANs are resource-intensive and unstable to learn, especially in a continual setup. This resulted in MADAR [123] (Rahman *et al.*, 2025) — a more intuitive but effective concept. MADAR introduced a diversity- aware replay mechanism, selecting and holding onto a small, representative subset of previous data. Rather than sampling, it selected them optimally to ensure maximum feature diversity. The result: an ultra-lightweight, fast continual learner with stable performance and less computational overhead.

Meanwhile, Chen *et al.* (2023) [124] tackled another fundamental issue: concept drift. Their model, which was trained on Android malware and integrating hierarchical contrastive learning with active learning to cope with changes over time. Remarkably, it achieved a seven-year accuracy in malware evolution, with less false negatives and positives.

In another research, Chin and Corizzo (2024) [125] proposed a novel semi-supervised incremental learning architecture to solve the problem of malware detection, considering the frequent lack of labeled data under real-world cybersecurity setup. Their approach implements one-class classification method along with experience replay so that the model can learn incrementally and yet be minimal supervision-based.

By leveraging both labeled and unlabeled data across a sequence of learning stages, the method achieves state-of-the-art accuracy with maintenance of past task understanding — effectively avoiding catastrophic forgetting even when resources for labeling are limited.

Semi-supervised continual learning (SSCL) approaches face major challenges in dynamic environments like network intrusion detection (NID), that are class imbalance and attack traffic also need to store labeled samples from past tasks which causes privacy concerns. To address these challenges, Amalapuram *et al.* proposed a method SPIDER (Semisupervised Privacy-preserving Intrusion detection with Drift-aware continual LEaRning) in [126]. To mitigate catastrophic forgetting and preserve the past knowledge, this method integrates gradient projection memory (GPM) with SSCL and stored only unlabeled samples in the memory. Another Semi-supervised class-incremental

learning based approach was developed in [127] with Transformers. SSCL-TransMD was built to mitigate a critical issue called “model aging" which traditional machine learning models cannot deal with because of ever-evolving nature of new malware variants and less effectiveness of models. To handle class imbalance SSCL-TransMD introduces lifelong semi-supervised mixture algorithm (LUMP) and Learning with Local and Global Consistency algorithm (LLGC) to Compute similarity scores for unlabeled samples. Multilayer Perceptron (MLP) was used for malware classification. SSCL–TransMD was evaluated on CICMalDroid2020 dataset and outperformed other existing methods.

While Class-Incremental Learning (CIL) is a method which assumes that each class appears only once but malware families are rapidly evolving, therefore retraining previous samples is resource-intensive. To address this, Temporal-Incremental Malware Learning (TIML) was proposed in [128]. This paradigm models the temporal distribution shifts within recurring families and classifies new malware variants. Subsequently, few-shot class-incremental learning was addressed by MalFSCIL (2024) in [129]. Aware of the scarcity of malware detection data, MalFSCIL combined task-agnostic incremental learning with attention-based distillation of knowledge in such a way that the model learned effectively from few examples per class. This method mitigates catastrophic forgetting by employing a decoupled training strategy with a variational autoencoder.

Another Self-Paced Class Incremental Learning (SPCIL) approach was implemented in [130]. This method introduces a unique loss function that enables efficient learning of new classes and maintains compact model constraints. It proposes an innovative dual-branch network model integrates new malware classes while retaining knowledge of previously learned categories.

In the field of cybersecurity, researchers always need to adapt a paradigm shift as traditional models quickly become obsolete because of rapidly evolving novel malware generations. Traditional models often suffer from model aging and concept drift as malware samples transforms very fast through obfuscation, mutation and distributional shifts. To make security systems more resilient and progressive, continual learning and temporal adaptation-based approaches need to be explored.

One critical and often-overlooked perspective is “security regression" identified by Ghiani *et al.*, 2025 [131]. Regression refers to the cases where a sam-

ple previously detected as malware is misclassified after an update, which introduces vulnerability to a detection system. This study proposed a Positive Congruent Training (PCT) which adds a regression-aware penalty during training to preserve the previous correct predictions without hampering the adaptability.

In parallel, researchers have emphasized the challenge of “concept drift" in malware analysis. In [132] authors propose an Adaptive Behavioral-Based Incremental Batch Learning (ABIBL) model which focuses on malware behaviour features and uses incremental batch updates to balance old and new malware data knowledge. This model was designed to reduce catastrophic forgetting and learn new malware behaviour incrementally over time.

In cyber attack ecosystem, malicious domain is a critical part generally used to conduct phishing, malware delivery and other cyber attack events. A continuous learning pipeline for malicious domain detection was developed in [133]. This approach continuously gathers new domain data from threat intelligence feeds (Acquire), adapts detection models over time, and anticipates unseen malicious strategies. Therefore a realistic pipeline (Acquire–Adapt–Anticipate) was implemented for real-world deployment.

An earlier effort was made to avoid costly retraining was noted in machine learning–based solutions such as the Support Vector Machine based on Incremental Learning (SVMIL) in [134]. The proposed method divides the dataset into smaller pieces and built an incrementally updated classifier which is capable to adapt new malware samples while preserving the prior knowledge. To avoid redundant computation and maintain the prior knowledge effective, SVMIL performs a quadratic programming update only for the new classes except retraining the whole model. Therefore, this method can reduce the time and memory overhead.

But this method was trained only on 20,000 samples (10,000 malware + 10,000 normal) and tested on 10,000 additional samples which is quite smaller as compared to today’s large-scale malware generations and relied only on Windows API call sequences.

To avoid the retraining from scratch, another notable work was proposed in [135] for ransomware detection. As traditional machine learning approaches are batch-based, when new ransomware appears the retraining becomes slow,

resource-intensive and risky while ransomware acts very fast. The proposed solution SILRAD (Sysmon Incremental Learning System for Ransomware Analysis and Detection) incorporates Sysmon logs for monitoring of Windows event and introduced an incremental learning system that continuously updates the detection model when new data arrives without full retraining. Also implements ADWIN algorithm for concept drift detection to ensure the model's adaptability towards evolving ransomware behaviour. This research also applied Pearson Correlation Coefficient (PCC) for feature selection, to keep the model lightweight and efficient. The same researchers proposed another ransomware detection system in [136] where a batch-based incremental learning framework (iCNN-LSTM+) was proposed. This method also used Sysmon logs to extract system activity and the hybrid model was developed with Improved CNN (iCNN) for feature extraction and LSTM to capture sequential patterns in ransomware behaviour.

A shift in malware traffic detection was seen in [137] where a class incremental learning framework (CIL4EMTD) was designed for encrypted traffic detection as new new malware families are obvious in real-world scenario. To make the system adaptive and mitigate forgetting, this approach combines knowledge distillation and memory replay while maintaining strong performance on previously learned classes.

3.5 Domain Adaptation in Malware Detection

Several works have demonstrated the efficacy of DA in malware detection. For instance, MD-ADA by Bhardwaj *et al.* [138] proposes an adversarial DA framework using CNN and GAN architectures to align malware image distributions between known and unknown variants, achieving high performance even on heterogeneous datasets. Similarly, Wang *et al.* [139] introduces an unsupervised DA method that aligns both global and class-level feature distributions using residual networks and self-attention, significantly improving accuracy for unseen malware samples.

Xiong *et al.* [140] explore domain adaptation for Android malware detection by simulating distribution shifts, showing that DA improves the robustness of DL models in dynamic mobile environments. Meanwhile, Li *et al.* [141]

address concept drift using adversarial DA combined with graph neural networks (GNNs) to capture invariant features in malware control flow graphs. PROTEUS by Hou *et al.* [142] adapts to dynamic malware behaviour by analyzing API call sequences in runtime environments. These studies collectively show that domain adaptation plays a pivotal role in transferring knowledge across time, behaviour and distributional boundaries.

Recent malware detection advancements increasingly utilize domain adaptation (DA) methods to counter issues emerging from sparse labeled data and significant domain shifts. Wang *et al.* [139] proposed an unsupervised domain adaptation framework that translates world and class-level features across source and target spaces using adversarial learning, residual deep networks, and self-attention mechanisms with high detection scores and improved performance over traditional models.

In order to counter the evolving nature of Android malware, Xiong *et al.* [140] employed domain adaptation to enhance the generalization power of deep learning models on other distributions and gained an increase in detection rates for unseen examples of malware. In the context of organizational network security, an unsupervised domain adaptation technique was introduced in [143] to aim at identifying advanced persistent threats (APTs) by aligning feature spaces without the necessity for manual tagging, with dramatic improvement in network-specific distributional shift robustness.

Besides, in network security, Karel and Sofka, 2015 [144] proposed a domain adaptation-based approach for detection of malicious traffic in enterprise environments through learning of domain-invariant features that yield improved accuracy in varying deployment situations. Adapting this into the Industrial Internet of Things (IIoT) environment, Ning *et al.* [145] proposed a hybrid framework of domain adaptation and ladder network architecture for efficient feature learning from both labeled and unlabeled sets along with making it more resilient towards dynamic cyber threats.

3.6 Ontology in Cybersecurity Knowledge Graphs

Security analysts increasingly rely on comprehensive background knowledge to recognize new and emerging cyber threats. To arrange and precisely dis-

seminate this information, Cybersecurity Knowledge Cybersecurity knowledge graphs (CKGs) have emerged as crucial tools, organizing cybersecurity entities and their interrelations with semantic triples.

RelExt, brought forth by the research in [146], is a classic case of applying deep learning techniques to automatically extract semantic triples from cybersecurity documents to enhance CKG building. The ensuing graphs facilitate better detection and identification of cyber threats by integrating knowledge from multiple sources, which assist security analysts in identifying patterns between malware, tools, and vulnerabilities.

Drawing on the application of knowledge graphs, the research in [147] investigates into their incorporation into industrial intrusion detection systems (IDS). In this case, knowledge graphs model objects like devices, software components, and network events, recording communication, hierarchical, temporal, and dependency relationships. Such contextualization in a structured format enhances machine learning models by enhancing alert relevance and detection quality. Observing the heterogeneity of sources of data in CKGs resulting in inconsistency of information, research in [148] employs Graph Neural Networks (GNNs), namely Graph Convolutional Networks (GCNs), for assessing and strengthening the trustworthiness of relationships in CKGs in probabilistic multi-label classification.

Cybersecurity information dissemination across platforms is addressed in [149], wherein it is directed at constructing cybersecurity knowledge graphs for threat intelligence aggregation and thereby increasing situational awareness, threat perception, and decision-making. Such aggregated Cybersecurity Knowledge Graph (CSKG) may be employed for attack surface discovery, threat prediction, and comprehensive cybersecurity management.

The constructed CSKG may be utilized for a variety of applications such as attack surface discovery, situational awareness, and threat forecasting. It presents a helpful tool for cybersecurity professionals to manipulate and interpret complex cybersecurity data, making overall cybersecurity defense systems stronger. This research demonstrates the potential of combining ontology construction, knowledge representation learning, and graph-based models to create robust tools for cybersecurity analysis and decision-making.

Following the advent of 5G technologies, network complexity has risen and

new threats emerged. To tackle this, a standardized ontology of cybersecurity and a Cybersecurity Knowledge Base (CSKB) drawn from knowledge graphs is proposed in [150] to consolidate diverse multi-source threat information while keeping logical relationships among entities and attributes. Resolving the problem of misleading cybersecurity intelligence, research in [151] enhances CKGs by incorporating provenance data. By incorporating provenance-related entity and relationship classes into the CKG schema and employing machine learning (ML) and natural language processing (NLP) techniques, the system can infer and associate the credibility and source of cybersecurity threat intelligence.

In the field of cybersecurity education, paper [152] is a bottom-up approach to constructing knowledge graphs from unstructured course materials. By building ontology and entity-relation extraction with no pre-existing datasets, this approach enables structured knowledge representation in domains lacking benchmark corpora. Cybersecurity of Cyber-Physical Systems (CPS) is improved in [153] through the construction of knowledge graphs that integrate multi-source heterogeneous data. They suggest a multi-dimensional data association and dynamic clustering algorithm to perform real-time correlation analysis to enable efficient attack detection, reconstruction, and analysis in CPS environments.

Collectively, these works show the tremendous progress being made in the development of structured, scalable, and intelligent knowledge graphs for cybersecurity. However, challenges remain in dynamic knowledge base updating, data trustworthiness, and interfacing symbolic structures with real-time data-driven detection mechanisms.

Traditional intrusion detection systems (IDS) for Internet of Things (IoT) networks tend to have difficulties since they don't implement multiview feature fusion, nor do they possess much capacity for detecting semantic relations between the features of the network traffic. In an effort to bridge the gap, research in [154] created an effective, light-weight IDS which used deep learning combined with the knowledge graph technology. Through combining knowledge graph-based and statistical analysis elements with CNN-BiLSTM architectures, the system learns dynamically the feature weights and improves the detection rates of different types of attacks by a rate of 90.01% for threats such as DoS, Probe, R2L, and U2L. Realizing there must be more

formalized cybersecurity knowledge, the study in [155] focuses on establishing a cybersecurity knowledge base by combining structured and unstructured data sources. By developing an ontology that describes cybersecurity entities, concepts and relations and building an associated cybersecurity knowledge graph, this paper addresses a quintuple-based model to enable vast cybersecurity knowledge management.

Given the rapid rate of evolution of cyber threats and the enormous quantity of day-by-day unstructured data, Open-CyKG was proposed in [156] as an open Cyber Threat Intelligence (CTI) Knowledge Graph structure. Utilizing the leverage of advanced NLP techniques such as attention-based neural Open Information Extraction (OIE) models and named entity recognition (NER) with a focus on cybersecurity, Open-CyKG facilitates programmatic acquisition and structuring of information from Advanced Persistent Threat (APT) reports. Based on domain-specific threats, the work in [157] builds a social engineering ontology that outlines eleven essential concepts and twenty-two relation types necessary for the modeling of social engineering attacks. Derived is a knowledge graph supporting applications like attack scenario analysis, threat analysis, and prediction of probable targets and paths of attack, demonstrating the use of structured semantic modeling within the cybersecurity context.

To counter the rise in sophistication in security tools and their use in cyberattacks, SecTKG was introduced in [158] with a fine-grained security tool knowledge graph. It makes use of a BiLSTM-Attention-based named entity recognition model to successfully extract knowledge, and hence forms an enormous graph with 4 million entities and 10 million relations covering more than 15,000 security tools. Knowledge sparsity is one of the most critical challenges to addressing building a cybersecurity knowledge graph. CSEA, a novel method with projection and rotation operations and cooperative adversarial training for expressing complex entity relationships and graph completeness, is put forward by the work in [159]. The efficacy of the method in exposing unknown facts and boosting graph robustness is empirically tested using cybersecurity databases such as CVE, CWE, and CAPEC. Moreover, in relation to financial cybersecurity research, the article in [160] proposes a method to create Fintech mobile payment platforms' knowledge graphs for risk assessment and policy inference. By applying deep learning technology and unsupervised,

half-automatic knowledge acquisition procedures, the article is able to infer risk-oriented knowledge graphs from policy document corpora of large sizes effectively, which can aid countermeasures against mobile payment risks.

These developments are indicative of the growing expanded maturity and sophistication of knowledge graphs of cybersecurity and combining machine learning, deep learning, and domain-modeling to address the evolving environments of threats. However, there is still much to work through in automating large-scale knowledge graph construction, ensuring the quality and validity of the data, and blending symbolic reasoning and real-time threat identification.

3.7 Discussions

Malware detection is seriously threatened by emerging evasion methods such as polymorphism, metamorphism, and fileless execution that make the traditional signature-based techniques obsolete. Zero-day exploits attack unknown vulnerabilities, and adversarial attacks deceive AI models into masking their existence. Inadequate availability of well-quality labeled datasets, concept drift with evolution of malware, and unwarranted excess false positives also hinder detection. In addition, today's threats leverage encrypted traffic, steganography, and multi-stage infection processes, requiring state-of-the-art behavioral and AI-based methods. Scalable, real-time solutions integrating ontology-based knowledge graphs, continual learning are crucial for addressing these challenges in cybersecurity. This research is trying to mitigate the challenges in following ways:

- Developing an ontological framework enables dynamic malware behavior analysis, capturing API call sequences and semantic relationships.
- Processing malware images with deep learning architectures such as CNNs, ResNet, U-Net, and GANs improves detection robustness.
- By introducing continual learning, this research addresses concept drift and enables models to adapt dynamically to evolving threats.
- The combination of ontology and deep learning through domain adaptation and knowledge graph-based integration ensures a novel, scalable, and explainable framework, bridging the gap between AI-driven malware classification and semantic cybersecurity knowledge.

This chapter attempts to identify the strengths and weaknesses for different methods of malware analysis to find the motivation of this thesis. The strengths and weaknesses of different methods leads to identify the research questions of this thesis.

3.8 Conclusions

The State-of-the-art analysis discussed a number of recent and some previous works on malware detection field. The critical analysis of this chapter led this research to achieve the aim and objectives. The following chapters have discussed,

- the approach to build an object oriented ontological framework to capture malware behaviour,
- deep learning based algorithms have been implemented in a different way (by combining and fusion) to achieve good accuracy.
- Development of a novel malware knowledge graph in cyber security field.

4 An Ontology-based Knowledge graph to Capture Malware Behaviour

Detection of new generation malware is a challenging task because of advanced anti-detection techniques and ability to change the malicious code in every iteration (polymorphism and metamorphism). End-users are being constantly attacked by the malicious software (malware). Recent malware samples are very complex in nature which include anti-detection codes. Behaviours of polymorphic and metamorphic malware are difficult to detect because of their complex and dynamic nature. Ontology-based frameworks should be dynamic to adapt changes, to add new instances and to include new datasets which expresses new generation malware behaviour.

To address the gap this research aims to contribute a malware ontology to capture malware behaviour. Based on API call sequence of Windows PE (Portable Executable) malware a novel dataset was developed in [161] which was included with this base ontology to create and visualize a new ontological structure with it's reasoning. This output ontology will be able to include new individuals from the dataset. Ontology individuals are much easier to handle than classes [162]. Object properties were defined to establish relation between the classes. Data properties were defined to connect the classes to the literals. The Ontology is developed by following steps:

1. The Ontology has four main classes: Malicious Software, Behaviour, Malware Family and Malware Code-Structure.
2. Malware Code Structure has five sub-classes: Basic, Polymorphic, Metamorphic, Packing and Code-obfuscation. Where Packing and code obfuscation are equivalent to each other.
3. The class 'Malware Family' has 14 sub-classes.

4. The Behaviour class has one sub-class i.e. “Changes in Code”. This sub-class has two child classes: ‘Encryption and Decryption’, ‘Translation and Rewriting’.
5. Relation between ‘Behaviour’ and ‘Malware Code Structure’ was declared with an object property named ‘showsBehaviour’.
6. Relation between classes, sub-classes with literals was defined with data properties.

4.1 Contributions

To build this Ontology this paper focused on a huge literature survey [163], [164], [165], [166] and reports [1], [2] to gather information about new generation Malware Samples and their exhibited behaviour. Our contributions on this paper are:

- This chapter proposes a base ontology structure which divides Malware in four sub-categories: Malicious Software, Behaviour, Malware Family and Malware Code-Structure. Malware Families have 14 sub-classes. The sub-classes are included with instances or individuals which are new in this framework.
- The benchmark dataset [161] for metamorphic malware was included with the base ontology. This dataset was created by recording the behaviour of metamorphic malware depending upon the changes in API call sequences. In the proposed framework the output ontology can capture the sequences of API calls (behaviour). This would be a new contribution in this ontology.
- The sequences of API calls were added as ‘Individuals’ in this base ontology. The output ontology can be viewed and comprehend through a HTML website.
- Reasoning of the the output ontology was done through ‘HermiT’ reasoner. The output file can be saved in .owl or .rdf format. This file can also be viewed through a HTML website.

[1] https://searchsecurity.techtarget.com/definition/metamorphic-and-polymorphic-malware

[2] https://www.lastline.com/blog/polymorphic-malware-real-life-transformers/

4.2 Problem Statement

In the era of Big Data, smart devices are always being used to generate and exchange huge amounts of data and information which is extremely difficult to handle. In the platform of large amount of data it's not easy to overcome the advanced attack patterns and techniques. Due to massive usage of interconnected systems cyber related threats are constantly increasing due to AI-based systems, phishing emails and most importantly novel malware variants. With advanced techniques malware can be installed at any moment, even in real-time.

Signature-based detection techniques mostly rely on human expertise to create the signatures, to detect a malicious behaviour in the code, the malware detectors look for the previously defined signature in the code. Thus the signatures have been built through the expert knowledge to store it in the signature-based method's knowledge (repository). But the major drawback of the signature-based method is that it cannot detect new type of attacks like, zero-day [9]. Anomaly-based detection technique uses its knowledge to distinguish the deviation between normal and malicious behaviour during program inspection. This technique is generally used to develop Intrusion Detection Systems (IDSs). The goal of this technique is to characterize the normal behaviour and identify attacks based on deviations from normalcy [167]. Those behaviours are learnt by machine learning approach with the current and past interactions with the network, server or user's machine. This technique is based on the ability to predict network traffic flow and can detect zero-day attack. The major shortcomings of anomaly-based malware detection are high false alarm or false positive rate, time complexity and difficulty in feature selection for training phase [9]. Insufficient expertise and knowledge about ever-changing malware behaviour also cause new malware attack.

To address these gaps this chapter aims to incorporate ontology to explore domain knowledge of malware, create a dynamic ontology-driven framework to include new generation malware features from a current dataset.

4.3 Methodology

As an explicit specification of conceptualization ontology[3] describes the concepts of a particular domain through the classes, sub-classes, properties and relationships. A malware ontology provides knowledge of different types of malware based on their classes, sub-classes, properties and behaviour.

The ontology has been developed using Protégé OWL editor[4] which is a graphical user interface to build ontologies. The objective is to use this ontology to create a dynamic web-based ontology which will be able to add new characteristics from a given dataset.

1. **Ontology Definition**
   - A domain ontology which was created using Protégé is loaded using rdflib. This ontology models malware behavior, classification, and structural attributes.
   - Various classes (e.g., Malware, Trojan, Worm, Spyware, Backdoor etc.) and object/data properties are defined in RDF format.

2. **Data Collection and Preprocessing**
   - A metamorphic malware behaviour dataset[5] which contains API call sequences and corresponding malware classifications (e.g., Trojan, Worm, Adware etc.) in two CSV files was loaded.
   - The CSV files are merged to associate API call sequences with their respective malware class labels.

3. **Dataset Description**

   Figure 4.1 is showing the word cloud of different API calls. The largest and most prominent API calls are:
   - _exception_
   - ntquerydirectoryfile
   - enumwindows
   - ldrgetprocedureaddress
   - findfirstfileexw
   - ntclose
   - ntallocatevirtualmemory

[3] https://www.obitko.com/tutorials/ontologies-semantic-web/introduction.html
[4] https://protege.stanford.edu/
[5] https://raw.githubusercontent.com/ocatak/malware_api_class/master/sample_analysis_data.csv

- regopenkeyexw

Medium-sized API calls are:

- regqueryvalueexa
- ntwritevirtualmemory
- getsystemmetrics
- getcursorpos
- ntdelayexecution
- ntopenprocess

Smaller and less frequent API calls are:

- ldrloaddll
- getfileattributesw
- drawtextexw
- ntcreatekey
- regcreatekeyexw
- shutdown

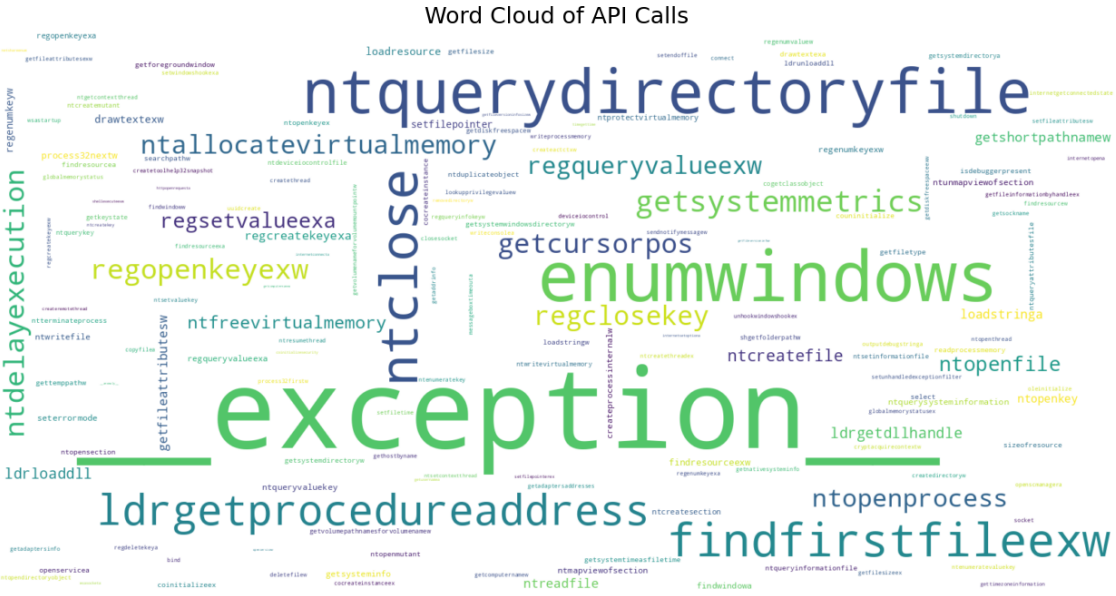


Figure 4.1: Word Cloud of API Calls

Figure 4.2 is showing the API calls that appear very frequently. The API token '_exception_' appears with the highest frequency compared to all other API calls.

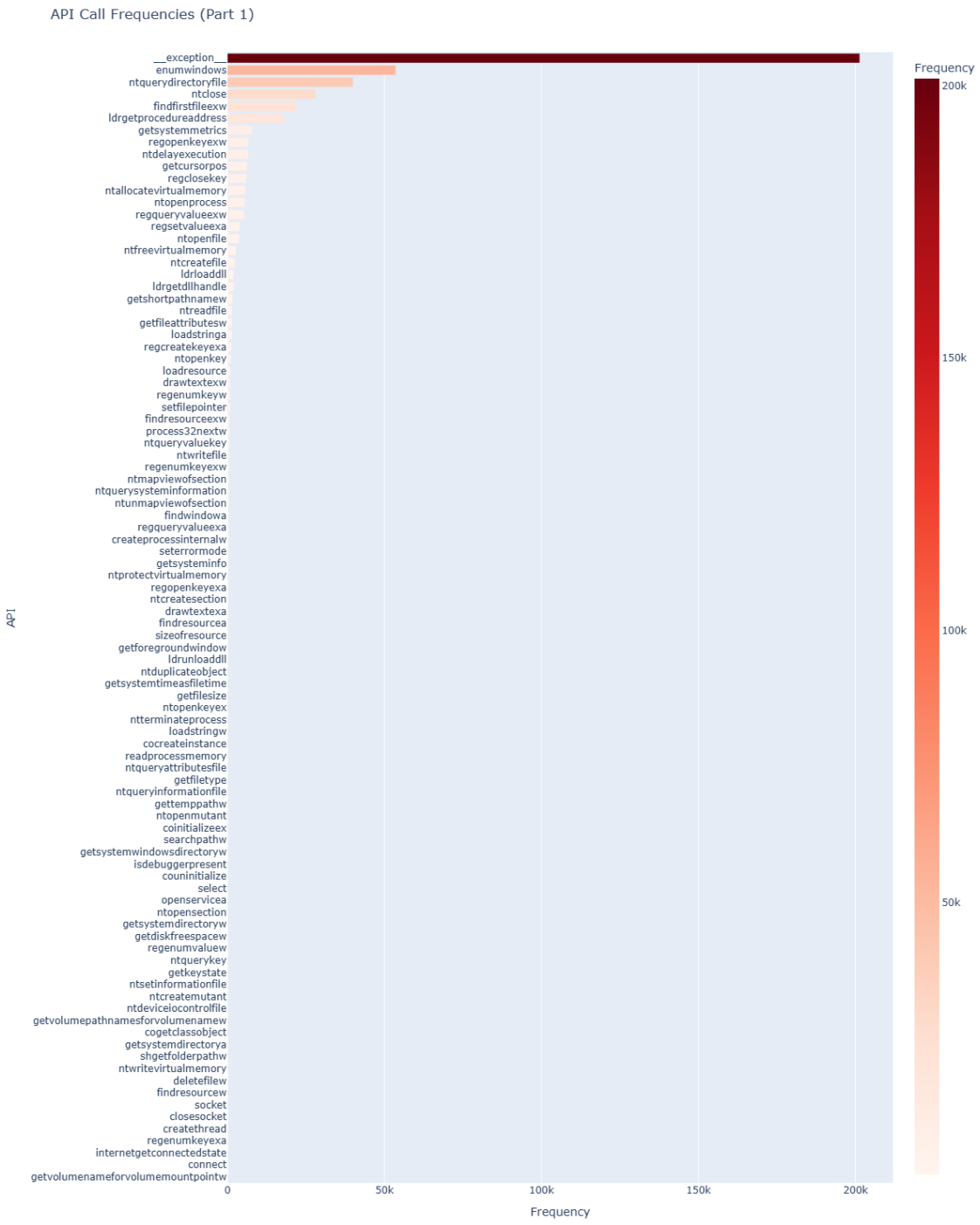


Figure 4.2: High Frequency API Calls

The top APIs are:

- **_exception_:** Exception handling routines for both error condition and control flow mechanisms [168].

- **enumwindows:** A Windows API which enumerates all visible top-level windows running on a system. Commonly used for system monitoring, to capture details of open programs[6].
- **ntquerydirectoryfile:** A native API call, can be accessed via 'ntdll.dll'. It retrieves information about files in a specific directory. Malware can use it for listing files in sensitive directories[7].
- **ntclose:** A native API which closes specific handles [169]. Malware can use this API call to close files to avoid detection and abnormal behaviour [170].
- **findfirstfileexw, getsystemmetrics:** Used for file enumeration and system discovery. 'findfirstfileexw' enumerates files and directories [171]. Malware can use it to discover files and folders. Ransomware may use it to encrypt all accessible files.

  'getsystemmetrics' is a system discovery API which can be used by malware to detect sandbox or virtual environments.

All the dominant APIs are fundamental building blocks, so malware samples can use them very frequently. Therefore, any detection system might need to filter out high-volume APIs to avoid false positives.

Figure 4.3 illustrates APIs which occur rarely (1 to 40 times). These APIs include:

- **setwindowshookexa, createdirectoryw, ntenumeratekey, gethostbyname:** These APIs are typically linked to persistence, file system changes or network communication.

  'setwindowshookexa' generally installs a hook procedure into a system. Malware can frequently use it for keylogging and DLL injection [172].

  'createdirectoryw' could be used by malware to create directories for storing payloads.

  'ntenumeratekey' is a native API, commonly used for enumerating registry keys and values. Attackers can use this API to manipulate registry metadata[8].

[6] https://hidocohen.medium.com/guloaders-anti-analysis-techniques-e0d4b8437195

[7] https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-ntquerydirectoryfile

[8] https://www.inversecos.com/2022/04/malicious-registry-timestamp.html

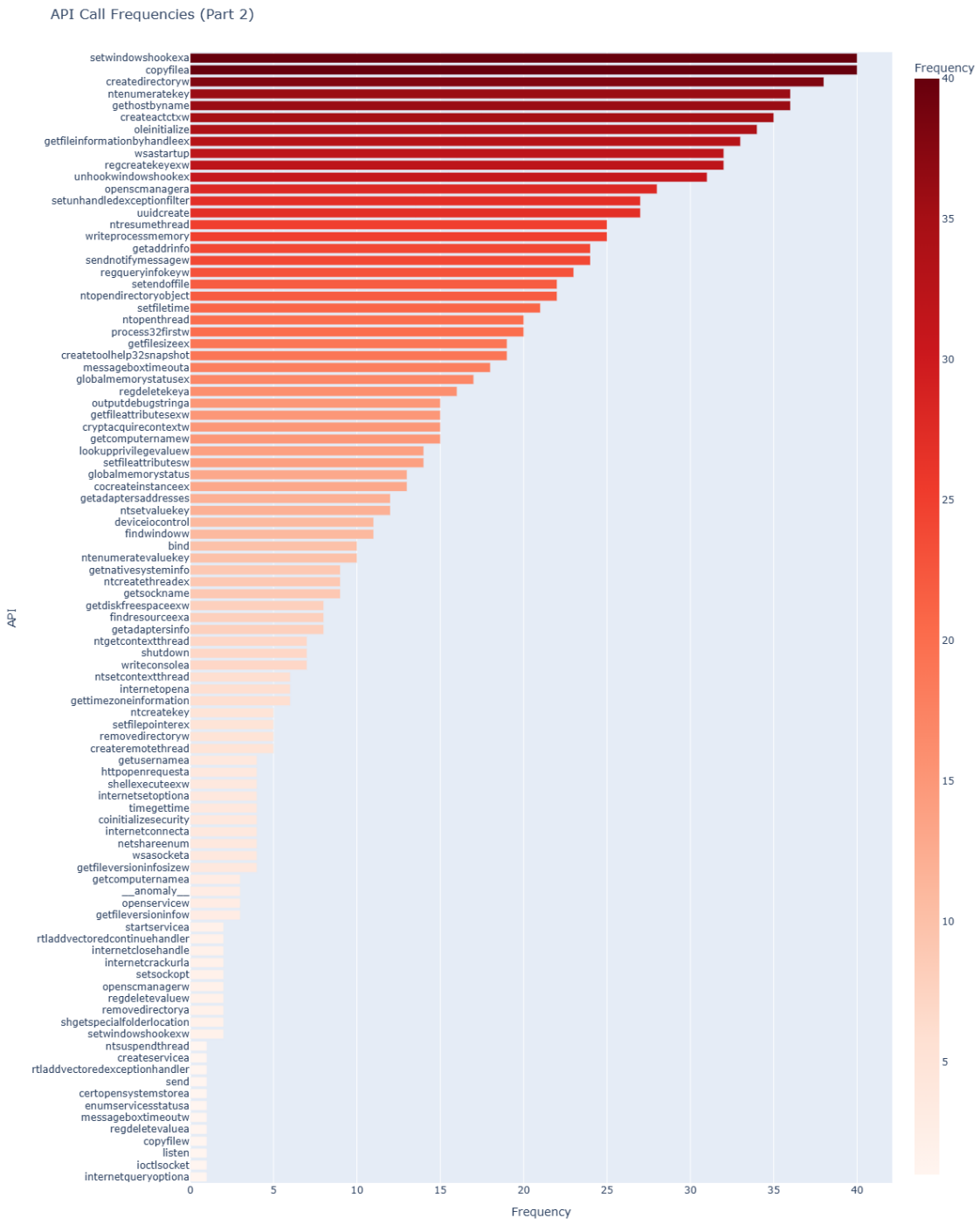


Figure 4.3: Low Frequency API Calls

'gethostbyname' is a function that retrieves host name corresponding to an IP address. Malware can use it to determine IP addresses and network services[9].

- **shellexecuteexw, internetopena, httpopenrequesta:**

  'ShellExecuteExW' is an API used for process creation and execution of malicious binaries. Commonly used by droppers and installers[10].

  'internetopena' creates HTTP or FTP session and 'httpopenrequesta' creates HTTP requests for downloading and uploading.
- **rtladdvectoredexceptionhandler:** A windows native API, often used by malware for code injection monitoring or control flow manipulation.

Many of these rare APIs are strongly associated with suspicious or malicious behavior. Even though they appear in less frequency, their presence in a sequence can be a strong malicious indicator.

4. **Knowledge Graph Construction**

   A function is defined to:
   - Create a unique URI for each API sequence.
   - Assign a malware type (e.g., Trojan, Backdoor, Virus) to the sequence in the ontology.
   - Store the API call sequence as a literal value linked to the malware class.

[9] https://learn.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-gethostbyname

[10] https://medium.com/@andreabocchetti88/malicious-api-1cee2100d6af

5. **Mapping API Sequences to Malware Classes**
    - The script iterates through different malware classes (Trojan, Worm, Adware, etc.).
    - Filters API calls based on their classification label.
    - Adds API sequences to the ontology under the corresponding malware class.
    - Prints the added entries to confirm ontology population.
6. **Ontology Serialization**
    - After processing all API sequences, the updated knowledge graph is saved.
    - The final ontology now contains API call traces mapped to malware families, making it useful for malware behaviour analysis and classification.

The following sections of this chapter are describing ontology engineering (how the ontology was built), how the ontology could handle multiple classification or inheritance, ontology oriented programming, algorithm development and evaluation of the ontology.

4.3.1 Ontology Engineering

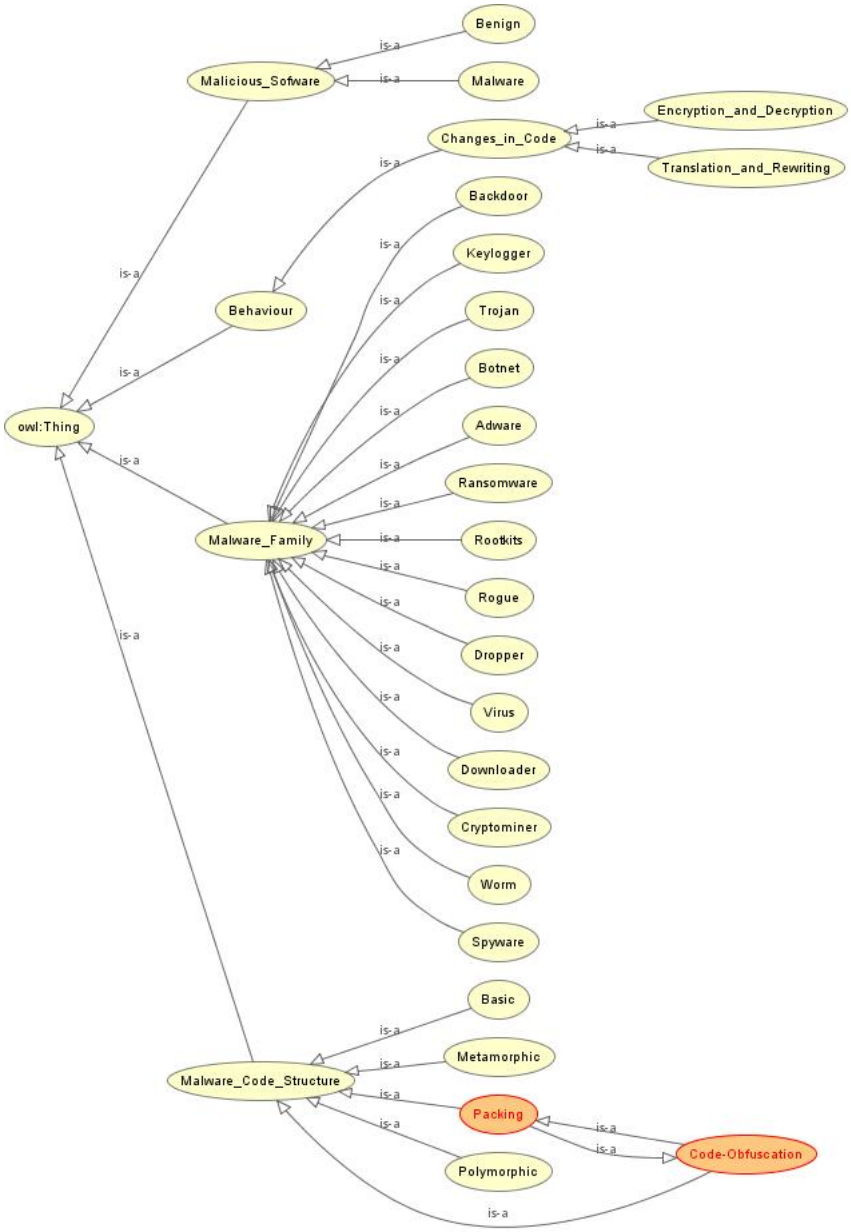


Figure 4.4: The base (or domain) Ontology

The process of developing an ontology is Ontology Engineering. The domain ontology was created by defining classes, sub-classes, data properties, object-properties, instances etc. The domain ontology consists of a superclass : Thing. The superclass has four sub-classes. The classes are connected with relations or object properties among them. Vocabularies were defined to de-

scribe the knowledge meaningfully and to relate classes and properties. “On the Semantic Web, vocabularies define the concepts and relationships (also referred to as “terms”) used to describe and represent an area of concern.”[11]

In this ontology, classes were defined based on malware behaviour, types of malicious software, malware families and malware code structure. There are 14 types of malware families were included to reflect recently available malware generations. Malware Code Structure includes five classes: Basic, Metamorphic, Polymorphic, Packing and Code obfuscation. Here the ontology shows Packing and Code obfuscation are equivalent to each other. Not all obfuscated code is packed: many obfuscation techniques do not involve packing. But packing can be considered as an obfuscation technique because packing does serve as an obfuscation-related function-hiding the original code from static analysis. In this sense, packing is a specific form of obfuscation.

The Behaviour class of the ontology is showing a specific behaviour i.e. Changes in Code. In the behaviour class, encryption and decryption is a behaviour of polymorphic malware and translation and rewriting is a behaviour of metamorphic malware: this relation has been defined using an object property “showsBehaviour". Within the sub-property ‘MetamorphicBehaviour’ is for the Metamorphic malware and ‘PolymorphicBehaviour’ is for the Polymorphic malware.

The reasoner (HermiT) produced this log file in Protégé. This file is showing that the the created ontology is consistent. HermiT is a reasoner for OWL 2 ontologies, which helps in checking consistency of the ontology.

Figure 4.5 is showing that HermiT reasoner has inferred:

- **Class hierarchy:** Checking subclass relationships.
- **Object property hierarchy:** Analyzing relationships between classes via properties.
- **Data property hierarchy:** Evaluating numerical or literal-based properties.
- **Class assertions:** Determining if individuals belong to specific classes.
- **Object property assertions:** Validating relationships between individuals.
- **Same individuals:** Identifying individuals that are logically equivalent.

[11] https://www.w3.org/standards/semanticweb/ontology

The reasoner processed the ontology in 216 milliseconds, which demonstrates efficient performance.

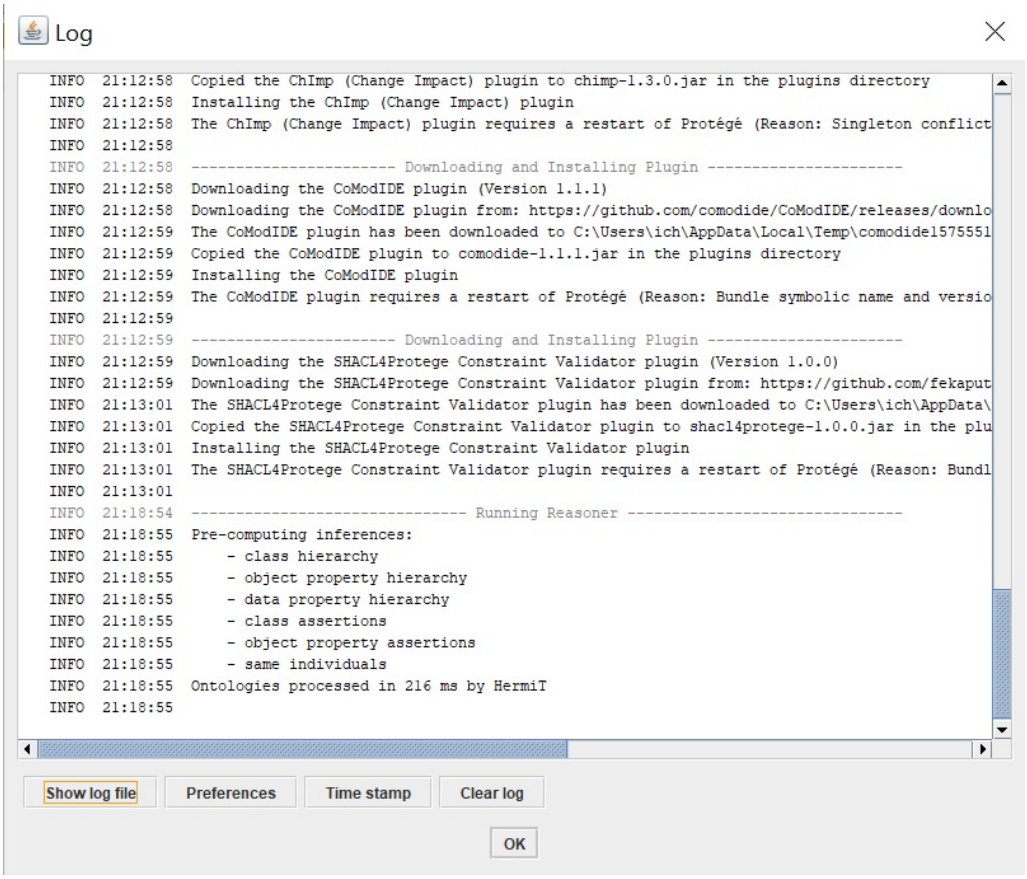


Figure 4.5: HermiT reasoner by Protégé

The sub-classes of the class ‘Malware Family’ are joined to the respective individuals. For example, Trojan has following individuals [173] as shown in Figure 4.6:

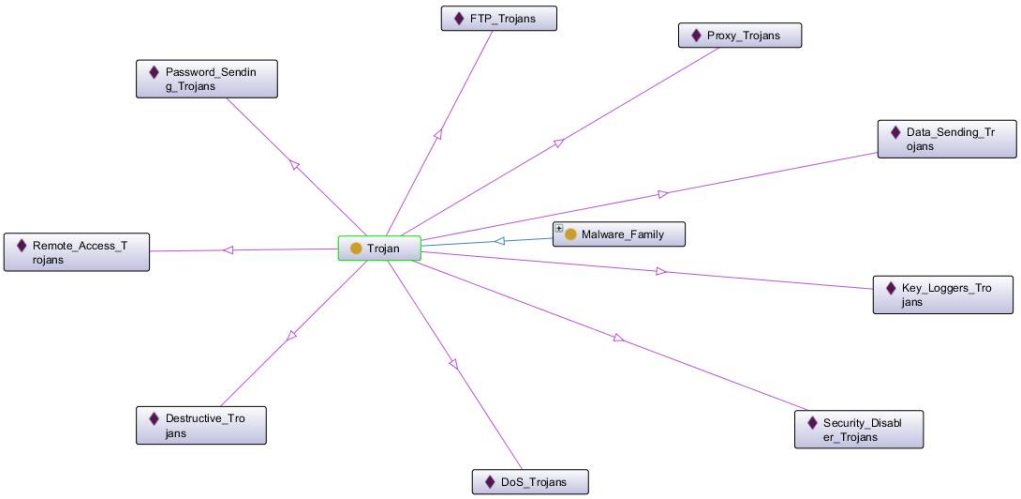


Figure 4.6: Individuals of Trojan

Instances of other sub-classes of 'Malware Family' are shown in Figure 4.7:

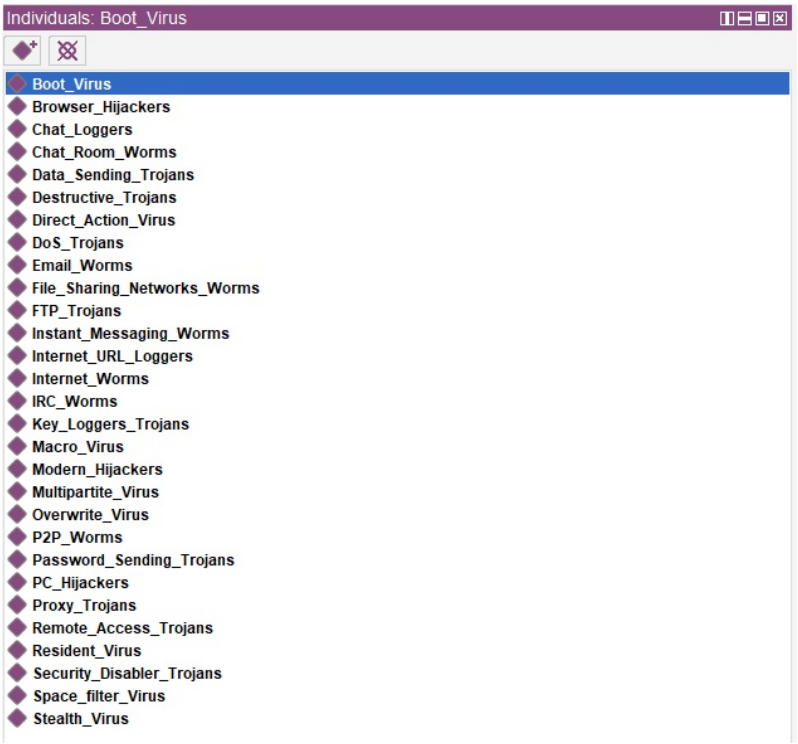


Figure 4.7: Individuals of other classes

Currently the ontology contains five object properties: hasClassification, hasCodeStructure, malclassification, second and showsBehaviour to establish connection between two classes.



Figure 4.8: OWL Object Properties

<Declaration>

<ObjectProperty IRI = “#hasClassification"/> </Declaration>

<Declaration>

<ObjectProperty IRI = “#hasCodeStructure"/> </Declaration>

<Declaration>

<ObjectProperty IRI = “#malclassification"/> </Declaration>

<Declaration>

<ObjectProperty IRI = “#showsBehaviour"/> </Declaration>

- hasClassification: Domain: Malware Family
  Range: Malicious Software
- hasCodeStructure: Domain: Malware Family
  Range: Malicious Code Structure
- malclassification: Domain: Malicious Software
  Range: Malware Family
- showsBehaviour: Domain: Malware Code Structure
  Range: Malicious Software

  The object property ‘showsBehaviour’ has two sub-properties: ‘MetamorphicBehaviour’ and ‘PolymorphicBehaviour’
- MetamorphicBehaviour: Domain: Metamorphic
  Range: Translation and Rewriting
- PolymorphicBehaviour: Domain: Polymorphic
  Range: Encryption and Decryption

The Ontology has five data properties: classification, requiredTime, hashValue, millisecond and hasOpCode to establish a relation between classes and literals.



Figure 4.9: OWL Data Properties

- classification: Domain: Malicious Software
  Range: string
- requiredTime: Domain: Malware Family
  Range: dateTime
- hashValue: Domain: Malware Code Structure
  Range: string
- millisecond: Domain: Malicious Software
  Range: dateTime
- hasOpCode: Domain: Malware Code Structure
  Range: string

Visualization of the whole ontology including classes, sub-classes, data properties, object properties and instances can be viewed from the Figure 4.10:

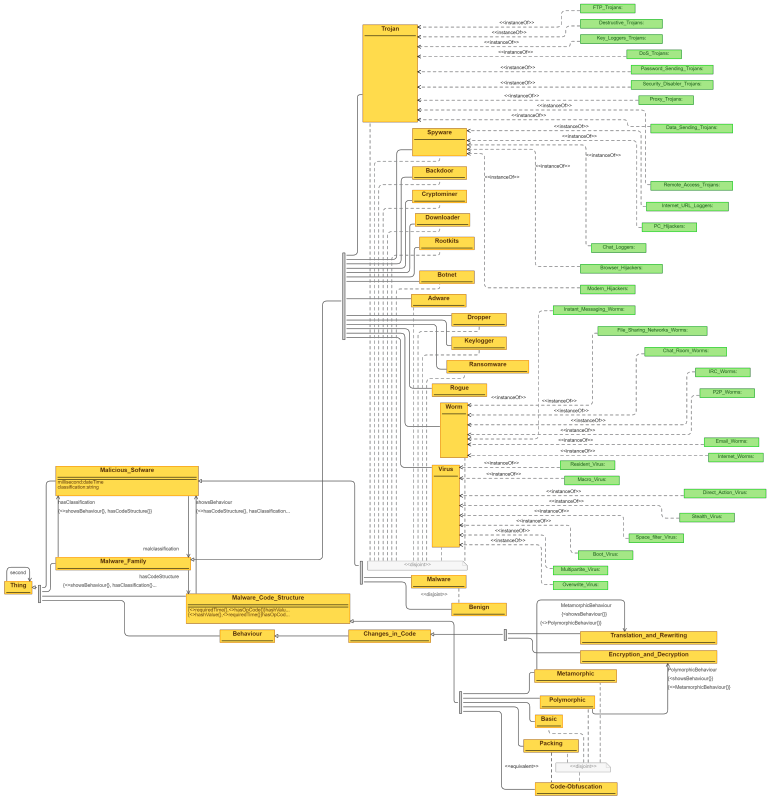


Figure 4.10: Overall visualization of the ontology

This structure was build using Graphical Ontology Editor: OWLGrEd [12].

4.3.2 Handling Multiple Classification or Inheritance

A specific malware sample may exhibit features from several malware types (e.g., Trojan, Worm, Backdoor etc.). If the malware sample belongs to two or more classes simultaneously, assigning multiple classification or inheritance will be indispensable to capture this complexity. When the behaviour or characteristics of a particular malware sample spanning multiple classes it leads to multiple classification or inheritance conflicts. Here are the ways to handle multiple classification or inheritance:

- **RDF Multiple Typing:**

  RDF provides the 'rdf:type' property which helps to associate with multiple classes. When a malware instance shows the characteristics of mul-

[12] http://owlgred.lumii.lv/

tiple types (Trojan, Virus, Worms etc.) the 'rdf:type' property will directly allow to support multiple types for an entity and leverages multiple inheritance capability (Figure 4.11).

```
g.add((API_Sequence, RDF.type, trojan_class))
g.add((API_Sequence, RDF.type, worm_class))
```

Figure 4.11: 'rdf:type' for classes

- **Using Object Properties:**

  OWL Object Properties are the most crucial features to connect a class or subclass with multiple relationships. Instead of relying only on class-based categorization, object property can relate a class with other behaviours and types of malware.

```
g.add((API_Sequence, showsBehaviour, virus_class))
g.add((API_Sequence, showsBehaviour, backdoor_class))
```

Figure 4.12: with object property

4.3.3 Ontology-oriented programming

The Ontology-oriented programming (OOP) was necessary because of following reasons:

1. **Semantic Linking of API Call Sequences to Malware Categories**
   - The ontology allows API sequences to be linked to malware classes (e.g., Trojan, Worm, Spyware) using object properties.
   - This structure ensures that API sequences are not just raw data but semantically categorized.
2. **Reusability and Extensibility**
   - The ontology-driven approach makes the knowledge base expandable. We can add new malware types and behavioral properties without modifying the entire structure.
   - For example, if a new type of malware emerges, it can be integrated seamlessly.
3. **Knowledge Graph Construction**

- By integrating API call data into an ontology, this research is effectively building a knowledge graph.
- This knowledge graph can be used for malware detection, classification, and explainability in AI-driven cybersecurity models.

4. **Integration with Domain Adaptation**
   - Since this research involves domain adaptation (Chapter 7) between image-based malware classification and ontology-based knowledge graphs, this approach provides a structured target domain.
   - It aligns malware image classification results with semantic knowledge in the ontology.

To establish an object oriented approach ontology oriented programming was designed to include new dataset in this domain ontology which has been used as a base ontology. A novel malware behaviour dataset has been collected from GitHub and included into this ontology to produce a new ontological structure. Ontology-oriented programming connects the classes, properties and instances of an ontology to a programming language [162]. In *static* approach of ontology oriented programming source code is generated from an ontology itself. An example of piece of code is shown in Figure 4.13.

```
<rdf:Description rdf:about="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Trojan">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
  <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Malware_Family"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Data_Sending_Trojans">
  <rdf:type rdf:resource="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Trojan"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
</rdf:Description>
<rdf:Description rdf:about="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#hasClassification">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Malicious_Sofware"/>
  <rdfs:domain rdf:resource="http://www.semanticweb.org/ich/ontologies/2021/5/untitled-ontology-19#Malware_Family"/>
  <rdfs:subPropertyOf rdf:resource="http://www.w3.org/2002/07/owl#topObjectProperty"/>
</rdf:Description>
```

Figure 4.13: Source Code

Because of static nature of this code it cannot be object-oriented and automatic.

The *dynamic* feature of an ontology facilitates data exchange, changes in data structure and automatic changes in real-time. It provides a 'reasoning engine' to check the consistency of the ontology. Dynamic models are much easier to accept changes in an ontology from it's background. In this study Python was used to carry out the ontology-oriented programming, because it's dynamic and object-oriented in nature.

In this approach, classes were defined from each class of the ontology. The process is same to define the object-properties and data-properties also. The classes of the ontology are shown in Figure 4.14.

```
#Ontology Classes

behaviour_class = URIRef(base_iri + 'Behaviour')

change_class = URIRef(base_iri + 'Changes_in_Code')

encrypt_class = URIRef(base_iri + 'Encryption_and_Decryption')
translate_class = URIRef(base_iri + 'Translation_and_Rewriting')

software_class = URIRef(base_iri + 'Malicious_Software')

malware_class = URIRef(base_iri + 'Malware')
benign_class = URIRef(base_iri + 'Benign')

code_class = URIRef(base_iri + 'Malware_Code_Structure')

basic_class = URIRef(base_iri + 'Basic')
polymorphic_class = URIRef(base_iri + 'Polymorphic')
metamorphic_class = URIRef(base_iri + 'Metamorphic')
packing_class = URIRef(base_iri + 'Packing')
obfuscation_class = URIRef(base_iri + 'Code-Obfuscation')

family_class = URIRef(base_iri + 'Malware_Family')
```

Figure 4.14: Classes of the Ontology

A dataset[13] was collected from GitHub to import into this base ontology. The sequences of API calls of the dataset were selected to add into the ontology as 'Individuals' with a data property 'hasOpCode'.

[13] https://raw.githubusercontent.com/ocatak/malware_api_class/master/sample_analysis_data.csv

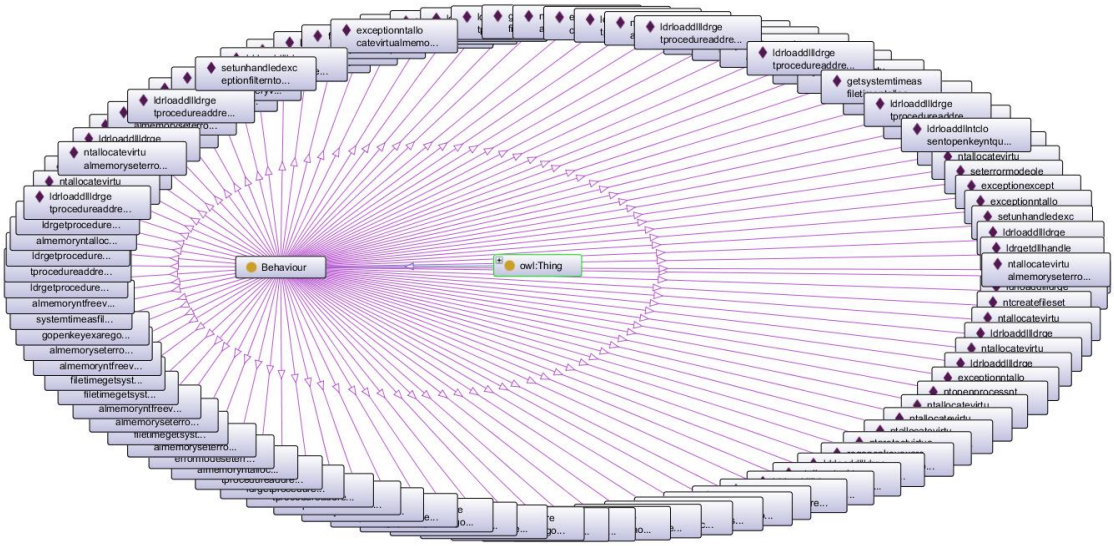


Figure 4.15: Sequential API calls added as malware behaviour

The output ontology was generated using the dataset including the individuals. Figure 4.15 and Figure 4.17 are showing the 'instances' or the 'individuals' of the output ontology. In this experiment, the behaviour of metamorphic malware were captured and presented as 'individuals' of this ontology. In the Figure 4.15, the individuals were connected with a property named 'has individual' (Figure 4.16).

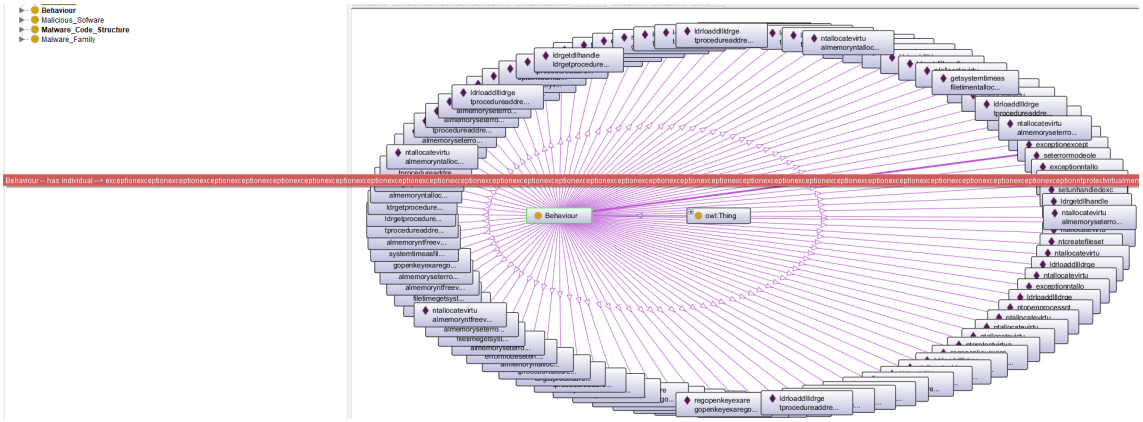


Figure 4.16: Property of the Behaviour



Figure 4.17: Individuals of the output ontology

This output ontology can be viewed and downloaded from a HTML website. A web-based version of the output ontology was developed by defining classes, object properties and data properties.

4.3.4 Algorithm of the Experiment

Algorithm 1 Ontology Enrichment Process

Require: `Base ontology file`, `Malware API data`, `Malware API labels`
Ensure: `Output Ontology`

Load the ontology into a graph G
Define the malware classes and their properties
Import the API call dataset (D_1) and the corresponding labels (D_2)
Combine D_1 and D_2 into a single dataset D
for each malware family $F \in \{Trojan, Backdoor, Virus, ...\}$ **do**
    Extract $D_F \leftarrow$ rows of D where *API_Labels* = F
    for each record $r \in D_F$ **do**
        Preprocess and clean the API call sequence
        Create a unique URIRef individual I
        Add the triple $(I, \texttt{rdf:type}, F)$ to G
        Add the triple $(I, \texttt{malclassification}, F)$ to G
        Add the triple $(I, \texttt{hasOpCode}, API\_Calls)$ to G
    end for
end for
Save (serialize) the enriched graph as `Output Ontology`
End

Algorithm 1 outlines the ontology enrichment process with the integration of API call sequences from the metamorphic malware behaviour dataset. The objective of this experiment is to create a semantically enriched ontology that could capture malware behaviour effectively.

1. **Data Loading:**
    - The manually created base ontology file was loaded into the graph structure G. This ontology will serve as the backbone of the knowledge representation as new classes, relationships and dataset will be added with this ontology.
    - Malware classes and their properties were also defined at this stage.
2. **Data Preparation:**
    - Two input files of the metamorphic malware dataset were imported: the first .csv file containing API sequences and the second one contains malware family labels.
    - These files were merged into a unified dataset D to ensure that each API sequence is associated with the correct malware class.

3. **Enrichment per Malware Family:**

   The dataset D is partitioned by malware families and for each family F:
   - A subset D_F was created containing the records labeled as belonging to F.
   - For every record r in D_F, the API call sequence is cleaned and preprocessed.
   - A unique individual I (represented as a URIRef in RDF) is generated to represent this malware instance in the ontology.

4. **Ontology Population:**

   For each individual I, three types of RDF triples were added into the graph:
   - Type Assignment: $(I, \texttt{rdf:type}, F)$ to link the instances or individuals to its malware family class.
   - Classification Relation: $(I, \texttt{malclassification}, F)$ to record the family categorization.
   - API Call Annotation: $(I, \texttt{hasOpCode}, API\_Calls)$ to associate the malware instance or individuals with its sequence of API calls.

5. **Serialization:** After the processing of all malware families and their associated individuals, the enriched ontology was serialized and saved in a new .owl file.

   The updated ontology is semantically enriched with real-world malware behaviour dataset.

4.4 Evaluation of the Ontology

Ontology evaluation is a process of measuring the quality and competency of an ontology. The proposed ontology was compared with other ontologies of this field. There are some terminologies of ontology metrics:

1. **Class Complexity:** Average number of paths to reach a class from the superclass *Thing*.
2. **Property Complexity:** Average number of semantic relations for object properties per class.
3. **Inheritance richness:** Average number of subclasses in a class.

4. **Attribute richness:** Ratio of the total number of data type properties by the number of classes.
5. **Relationship richness:** 'reflects the diversity of relations', by comparing the number of non-subsumption relations to the number of subsumption relations.
6. **Comprehension of classes:** Percentage of annotation of the classes in ontology.

Metrics of the output ontology were calculated from OntoMetrics[14]. Table 4.1 is showing the base metrics of the ontology:

Base Metrics	**Value**
Axioms	555
Logical axioms count	257
Class count	29
Total classes count	29
Object property count	8
Total object properties count	8
Data property count	6
Total data properties count	6
Properties count	14
Individual count	142
Total individuals count	142
DL expressivity	ALCR(D)

Table 4.1: Base Metrics

Table 4.2 and Table 4.3 are showing the Class axioms and Object property axioms of the output ontology, respectively.

Axiom Type	**Count**
SubClassOf axioms	28
Equivalent classes axioms	1
Disjoint classes axioms	5
GCI Count	0
Hidden GCI Count	2

Table 4.2: Class Axioms Summary

[14] https://ontometrics.informatik.uni-rostock.de/ontologymetrics/

Axiom Type	Count
SubObjectPropertyOf axioms	4
Equivalent object properties axioms	0
Inverse object properties axioms	0
Disjoint object properties axioms	2
Functional object properties axioms	0
Inverse functional object properties axioms	0
Transitive object property axioms	0
Symmetric object property axioms	0
Asymmetric object property axioms	0
Reflexive object property axioms	0
Irreflexive object property axioms	0
Object property domain axioms	8
Object property range axioms	12
SubPropertyChainOf axioms	0

Table 4.3: Object Property Axioms Summary

Table 4.4 and Table 4.5 are showing the Data property axioms and Individual axioms of the Ontology, respectively.

Axiom Type	Count
SubDataPropertyOf axioms	1
Equivalent data properties axioms	0
Disjoint data properties axioms	1
Functional data property axioms	0
Data property domain axioms	6
Data property range axioms	5

Table 4.4: Data Property Axioms Summary

Axiom Type	Count
Class assertion axioms	184
Object property assertion axioms	0
Data property assertion axioms	0
Negative object property assertion axioms	0
Negative data property assertion axioms	0
Same individuals axioms	0
Different individuals axioms	0

Table 4.5: Individual Axioms Summary

Table 4.6 is showing the Annotation axioms of the proposed ontology.

Axiom Type	Count
Annotation axioms count	0
Annotation assertion axioms count	216
Annotation property domain axioms count	0
Annotation property range axioms count	0

Table 4.6: Annotation Axioms Summary

Here are the ontology metrics [174] that help assess the quality, complexity, and expressiveness of an ontology.

1. **Schema Metrics (Ontology Structure)**
   - **Attribute Richness:** Measures the proportion of classes that have attributes (or properties) [174].

     Interpretation: A value close to 1 means most classes have attributes, while 0 means no attributes are assigned.
   - **Inheritance Richness:** Measures how well classes are structured hierarchically (i.e., parent-child relationships) [175].

     Formula: (Total number of subclasses) / (Total number of classes)

     Interpretation:

     Close to 1: Highly hierarchical.

     Close to 0: Flat ontology, meaning little hierarchy.
   - **Relationship Richness:** Measures how well relationships (object properties) are used in comparison to attributes (data properties) [176].

     Formula: (Number of relationships) / (Number of attributes + relationships)

     Interpretation:

     Closer to 1: More object properties (better knowledge representation).

     Closer to 0: Only data properties are used.
   - **Attribute Class Ratio:** Measures the number of attributes per class.

     Interpretation:

     Higher value: More properties per class.

     Zero: No attributes assigned.

- **Equivalence Ratio:** Measures the proportion of classes that have equivalence axioms (i.e., defined classes).

  Interpretation:

  Higher value: More defined classes, making the ontology more expressive.

- **Axiom/Class Ratio:** Measures the average number of axioms per class.

  Formula: (Total axioms) / (Total classes)

  Interpretation:

  Higher value: More logical statements in the ontology, indicating a richer structure.

- **Inverse Relations Ratio:** Measures how many relationships have an inverse relation defined.

  Interpretation:

  Higher value: More inverse relations, making the ontology more expressive.

- **Class/Relation Ratio:** Measures how many classes exist per relationship.

  Interpretation:

  Higher value: Many classes per relationship.

2. **Knowledgebase Metrics (Data in the Ontology)**

   - **Average Population:** The average number of instances per class [176].

     Interpretation:

     Higher value: More instances (better real-world representation).

   - **Class Richness:** Measures the proportion of classes that have instances [176].

     Interpretation:

     Higher value: More populated classes.

Metric	Value
Attribute richness	0.206897
Inheritance richness	0.965517
Relationship richness	0.333333
Attribute class ratio	0.0
Equivalence ratio	0.034483
Axiom/class ratio	19.206897
Inverse relations ratio	0.0
Class/relation ratio	0.690476

Table 4.7: Schema metrics of the Ontology

Table 4.7 and Table 4.8 are showing the Schema metrics and Knowledge-base metrics of the output ontology, respectively.

Metric	Value
Average population	4.896552
Class richness	0.310345

Table 4.8: Knowledgebase metrics of the Ontology

The Schema metrics and Knowledge metrics of MALOnt [59] and Swimmer's Ontology [177] are shown in Table 4.9, Table 4.10 and Table 4.11, Table 4.12, respectively.

Metric	Value
Attribute richness	0.191176
Inheritance richness	0.676471
Relationship richness	0.402597
Attribute class ratio	0.0
Equivalence ratio	0.0
Axiom/class ratio	20.661765
Inverse relations ratio	0.032258
Class/relation ratio	0.883117

Table 4.9: Schema metrics of MALOnt

Metric	Value
Average population	3.897059
Class richness	0.970588

Table 4.10: Knowledgebase metrics of MALOnt

Metric	Value
Attribute richness	0.0
Inheritance richness	0.941176
Relationship richness	0.0
Attribute class ratio	0.0
Equivalence ratio	0.0
Axiom/class ratio	2.735294
Inverse relations ratio	0.0
Class/relation ratio	1.0625

Table 4.11: Schema metrics of Swimmer's Ontology

Metric	Value
Average population	0.0
Class richness	0.0

Table 4.12: Knowledgebase metrics of Swimmer's Ontology

The developed ontology was compared with another two ontologies: MALOnt [59] and Swimmer's Ontology [177]. Table 4.13 is showing the terminology box (Tbox) for the comparison of the ontologies.

Table 4.13: Evaluation Results

Results	**Output Ontology**	MALOnt	Swimmer's Ontology
Attribute richness	**0.206897**	0.191176	0.0
Class richness	0.310345	**0.970588**	0.0
Inheritance richness	**0.965517**	0.676471	0.941176
Relationship richness	0.333333	**0.402597**	0.0
Average population	**4.896552**	3.897059	0.0

Table 4.13 demonstrates the performance evaluation of the proposed ontology with MALOnt and Swimmer's Ontology but these ontologies are fundamentally different from each other in terms of their conceptual focus and

scope. MALOnt was designed for malware threat intelligence report and focused on malware entities, attacker groups and indicators; Swimmer's Ontology provides a taxonomic classification malware and their characteristics but limited presentation of relationships and dynamic behavioural aspect. Our proposed ontology is behaviour-based which represents malware behaviour at a semantic level with API-call-driven relations. Regarding the graph structures such as node count, relationship richness, and depth are significantly different for each ontology. Therefore, numerical comparison of individual scores is quite difficult in this case.

4.5 Discussions

The dynamic ontology demonstrated in this Chapter, is flexible to adapt new malware behaviour and other cyber security datasets. The structure of this ontology can accommodate new malware types, behaviour, and relationships by following the same pattern used to define existing classes and sub-classes and instances. RDF-based ontologies provide flexibility to evolve with new knowledge and inventions for a particular domain.

The Chapter 7 of this thesis presented the extended versions of this ontology with other cyber security datasets. The ontology based knowledge graphs in Chapter 7 proves the flexibility of this ontology. Working with actual malware dataset is little difficult and risky in this scenario, consequently this ontology is further extended with other malware datasets. Following key steps were taken to extend this ontology and make it more adaptable:

- **Dataset Integration with Ontology:**

  The API call based ontology was enriched with large, diverse set of malware samples such as Malimg, MaleVis, and Virus MNIST. The image based malware classification was integrated to the API based ontology to develop three novel ontology based knowledge graphs.
- **Mapping Between Ontology and Malware Dataset:**

  The image-based malware classification results (from deep learning model) are mapped to the behaviour based ontology and added as 'Classes' with other malware families.
- **Integration with Deep Learning Models:**

To integrate the ontology with real-world datasets the image-based malware datasets were used as additional data and results from the deep learning based classification were added to enrich the ontology.

- **Identification of Malware Strains:**
  1. The final three ontologies in Chapter 7 are incorporated with malware behaviour dataset and malware image datasets. Enrichment of the ontology with large-scale malware datasets will enable it to identify different malware strains.
  2. Implementation of Domain Adaptation (with ontology features and image features) can handle different malware strains across domains.

4.6 Conclusions

The proposed ontology is an approach to build a knowledge of a malware domain which is trying to contribute new idea in cyber-security domain. This ontological structure will be beneficial for knowledge sharing and knowledge management for the particular field. This Ontology includes following contributions:

- A detailed understanding of different malware families with their behaviour.
- This work focused on capturing the changes of API call sequences (Behaviour) of metamorphic malware and to comprehend how this type of malware can produce absurd opcodes. This is an approach to understand and provide a well-structured malware behaviour knowledge.
- The base ontology was divided with different instances to classify malware types. 'Malware Family' has a number of 'Individuals' for each subclasses.
- A web-based ontology may provide proper understanding of the ontology building and consistency of the ontology by providing the reasoner. This study tried to establish an object-oriented approach of ontology building and its handling in a dynamic sense.

In future, this ontology would be included with more number of classes and individuals to improve the evaluation results and richness of the ontology.

5 Malware Analysis using Deep Learning based Image Processing

5.1 Introduction

Deep Learning has become an useful method for malware detection and analysis because of its advanced feature engineering, time saving and automated properties to generalize novel malware variants. Image and visualization-based detection methods are useful to analyze malicious codes. In case of data driven Deep Learning different types of dataset e.g. grayscale images, bytes sequences, API call sequence etc. are being used to feed into the model. Deep Learning models are efficient to learn the patterns from the images and extract features automatically to identify and categorize the malware classes and novel threats. In this research, grayscale images of malware variants have been used as input of models for detection and classification purposes. Decisions of the models have been visualised through plots, heatmaps and confusion matrix.

5.1.1 Problem Statement

Deep Learning is being widely used in cyber security field as it provides proactive measures to identify massive cyber threats by training the data points in a small amount of time. Existing cyber defence methods fail to detect novel malware threats, modern attack patterns, DDoS attacks, zero-day attacks and voice based systems. Deep Learning based models are able to learn from specific features and analyze network traffic more precisely than Machine Learning models. Deep Learning models are reusable to work on the newly intro-

duced datasets.

Despite all the advantages, Deep Learning models need to be applied on wide range of cyber incident and malware related datasets to find out novel methodologies to improve accuracy of the model and find out what type of algorithms could be integrated to detect anomalies and behavioural characteristics of the dataset.

To address this gap, this chapter aims to incorporate a number of deep learning based models on the malware image datasets to find out novel algorithms for malware detection and classification.

5.1.2 Contributions

This Chapter outlined the following contributions:

- Existing Deep Learning architectures are being used to find out novel methods and improve accuracy.
- For malware categorization ResNet-50 coupled with attention network performed well.
- CNN with RNN methods also produced promising accuracy for classification.
- A comparison with the existing works was presented in terms of features and accuracy.

5.2 Methodology and Experimental Apparatus

By carrying out all the experiments in this section, the research aims to find out data driven approaches for malware detection and classification methods by training the created models to learn statistically relevant features for better prediction of classes.

5.2.1 Dataset 1: Malimg

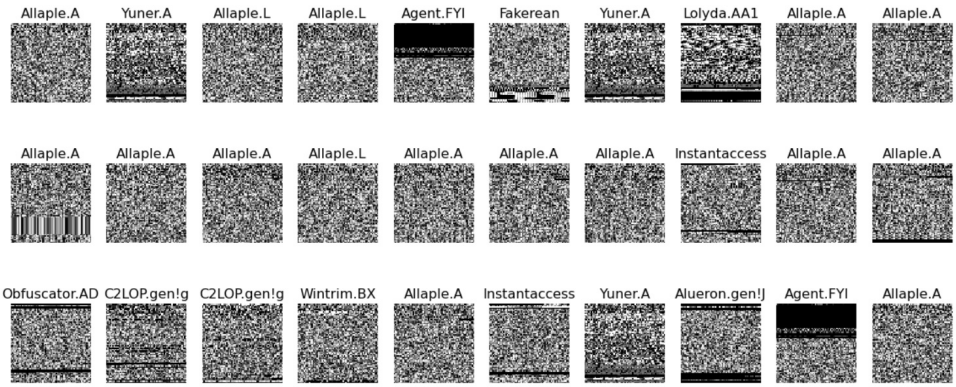


Figure 5.1: Sample of Malimg Dataset

Malimg dataset[1] is a benchmark dataset for image-based malware analysis has been used in this thesis. This dataset was introduced by Nataraj *et al.* for visualizing and classifying malware binaries as grayscale images. A sample of the dataset is shown in Figure 5.1.

The dataset contains 9339 images of 25 Malware Families [164] (Figure 5.2). The images of the dataset have varying sizes.

- One of the most widely used malware image datasets.
- Contains grayscale images of malware binaries converted using visual representation techniques.
- Helps researchers explore deep learning-based malware classification without relying on traditional handcrafted features.

[1] https://www.dropbox.com/s/ep8qjakfwh1rzk4/malimg_dataset.zip?dl=0

No.	Family	Family Name	No. of Variants
01	Dialer	Adialer.C	122
02	Backdoor	Agent.FYI	116
03	Worm	Allaple.A	2949
04	Worm	Allaple.L	1591
05	Trojan	Alueron.gen!J	198
06	Worm:AutoIT	Autorun.K	106
07	Trojan	C2Lop.P	146
08	Trojan	C2Lop.gen!G	200
09	Dialer	Dialplatform.B	177
10	Trojan Downloader	Dontovo.A	162
11	Rogue	Fakerean	381
12	Dialer	Instantaccess	431
13	PWS	Lolyda.AA 1	213
14	PWS	Lolyda.AA 2	184
15	PWS	Lolyda.AA 3	123
16	PWS	Lolyda.AT	159
17	Trojan	Malex.gen!J	136
18	Trojan Downloader	Obfuscator.AD	142
19	Backdoor	Rbot!gen	158
20	Trojan	Skintrim.N	80
21	Trojan Downloader	Swizzor.gen!E	128
22	Trojan Downloader	Swizzor.gen!I	132
23	Worm	VB.AT	408
24	Trojan Downloader	Wintrim.BX	97
25	Worm	Yuner.A	800

Figure 5.2: Malware Families in Malimg Dataset [2]

5.2.2 Dataset 2: Virus MNIST

A benchmark malware dataset Virus-MNIST (Modified National Institute of Standards and Technology) [178] was used in this research. This dataset has 48422 number of train samples and 3458 number of test samples.

In case of Virus MNIST dataset each image consists of 32 pixels in height and width, the number of total pixels of images is 1024. The lightness and darkness of the images depend on the single value of pixel assigned with it. The images were stored in 0 to 255 pixel values. The datasets have 1026 columns where first column represents the labels 10 classes which is the type of malware and the last column is the hash values of the images. The columns from 1 to 1024 represents the pixel values of the images. Table 5.1 is showing the details of Virus MNIST dataset. The dataset has 10 executable malware

files in a format of gray scale images with 32x32 pixel size Figure 5.3. Although this is a smaller dataset but has been extensively used in the literature.

- Inspired by MNIST, this dataset offers malware images in a 32×32 format, making it useful for lightweight experiments.
- Suitable for quick model prototyping and evaluating deep learning architectures on malware images.

Table 5.1: Details of Virus MNIST dataset [178]

Virus MNIST Class Distributions				
Class	**Count**	**Group**	**Type**	**Example**
0	2516	Beneware	Good	putty.exe
1	7684	Malware	Adware	IESettings
2	3037	Malware	Trojan	Supreme.exe
3	2404	Malware	Trojan	myfile.exe
4	796	Malware	Installer	myfile.exe
5	6662	Malware	Backdoor	myfile.exe
6	15377	Malware	Crypto	Powershell
7	7494	Malware	Backdoor	BitTorrent.exe
8	2571	Malware	Downloader	myfile.exe
9	3339	Malware	Heuristic	myfile.exe

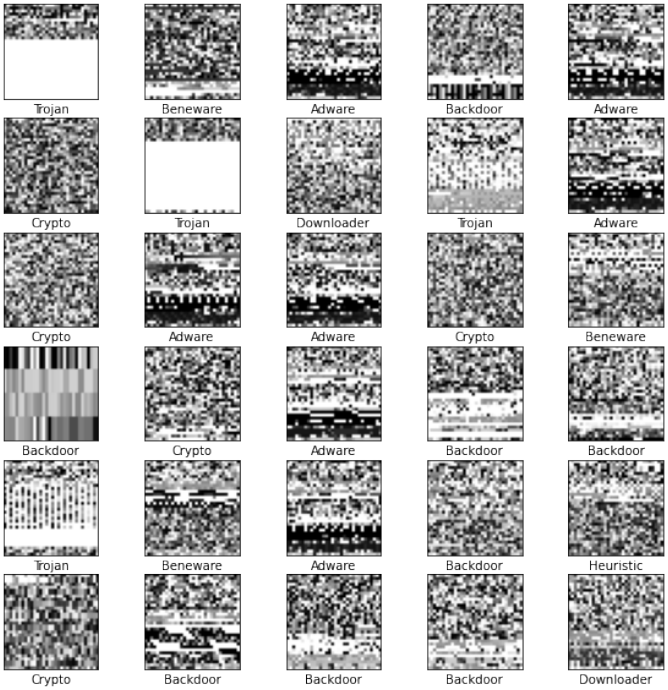


Figure 5.3: Virus MNIST dataset with classes

5.2.3 Dataset 3: MaleVis

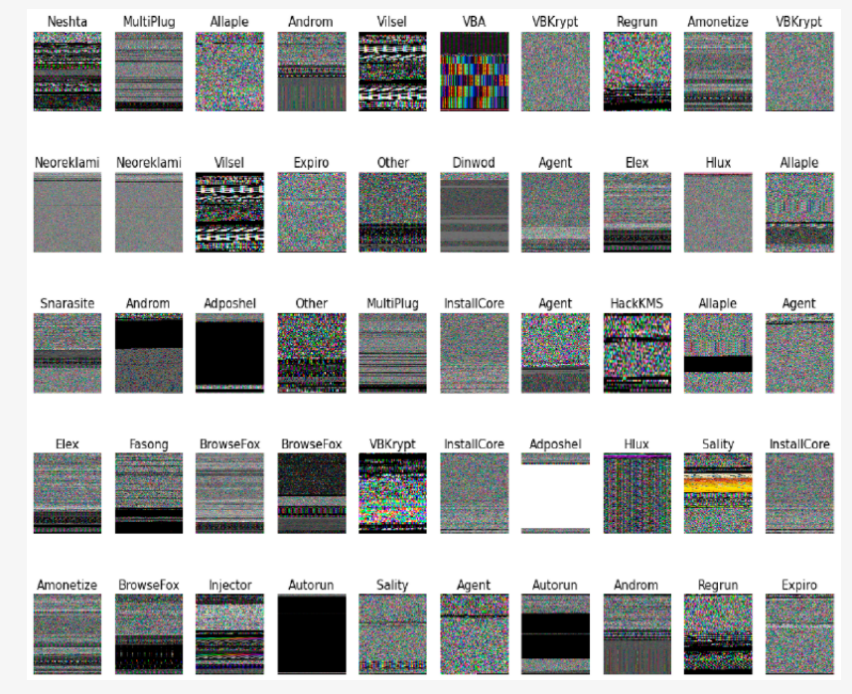


Figure 5.4: MaleVis dataset with classes

The MaleVis (Malware Evaluation with Vision) dataset[2] is designed for vision-based malware recognition and includes images generated from 25 malware classes and 1 legitimate software class. This dataset offers 9,100 training and 5,126 validation RGB images in resolutions of 224x224 and 300x300 pixels (Figure 5.4). In this thesis the (224x224) image size was used for experiments.

- Designed for a more diverse malware classification task, covering a wide range of malware families.
- Offers higher resolution images (e.g., 224×224) compared to Malimg and Virus MNIST.
- Provides realistic malware image distributions, making it closer to real-world malware classification scenarios.

For this research, the 224×224 resolution of MaleVis dataset was chosen to maintain the uniformity across models and compatibility established deep learning architectures. The larger resolution 300x300 was tested initially but that didn't improve the performance of the models significantly.

Both Malimg and Virus-MNIST contain greyscale malware images while MaleVis provides an RGB image domain. By incorporating both greyscale and RGB data, this research evaluated and tested the robustness of proposed models across different modalities.

5.2.4 Methodology

Deep learning approaches play a crucial role in various domains, especially in computer vision, image processing, predictive analytics and cybersecurity. These models leverage neural networks with multiple layers to extract complex patterns from data, enabling tasks such as classification, detection, and generation. In this research, following deep learning models were implemented:

1. **Detection with CNN Model using Keras Tuner**
    - **Importance:** Hyperparameter tuning helps optimize CNN performance by selecting the best architecture, learning rate, and other parameters.

[2] https://web.cs.hacettepe.edu.tr/~selman/malevis/

- **Rationale:** Finding the best configuration improves accuracy and generalization, which is crucial for malware classification.

2. **Classification with ResNet-50 Transfer Learning**
   - **Importance:** ResNet-50 is a powerful deep network with skip connections to address vanishing gradients.
   - **Rationale:** Transfer learning reduces training time and improves accuracy by leveraging pre-trained knowledge on large datasets.
3. **Classification with 3DCNN**
   - **Importance:** 3D CNNs capture spatiotemporal features, making them useful for sequential or volumetric data.
   - **Rationale:** Useful for analyzing malware evolution over time or patterns in dynamic behavior analysis.
4. **Image Generation Using Autoencoder**
   - **Importance:** Autoencoders learn efficient latent representations and generate synthetic data.
   - **Rationale:** Useful for data augmentation, feature extraction, and adversarial sample generation in cybersecurity.
5. **Classification with Ensembling CNNs**
   - **Importance:** Ensembling combines multiple models to enhance prediction accuracy and robustness.
   - **Rationale:** Helps reduce variance and mitigate errors by leveraging different perspectives on the same data.
6. **ResNet-50 with Attention Network**
   - **Importance:** Attention mechanisms improve focus on relevant features, enhancing classification accuracy.
   - **Rationale:** Helps malware classifiers prioritize critical patterns in images, improving interpretability.
7. **Classification with CNN Normalization**
   - **Importance:** Normalization techniques stabilize training, speed up convergence, and improve generalization.
   - **Rationale:** Essential for deep networks to maintain numerical stability and prevent overfitting.
8. **Classification with CNN and RNN**

- **Importance:** Combining CNNs (for spatial feature extraction) with RNNs (for sequential dependencies) enhances learning.
- **Rationale:** Effective for malware behavior analysis, where sequences or actions need modeling.

9. **ANN Implementation**
   - **Importance:** Although CNN is the most promising approach for image recognition, the ANN model provides critical insights for how malware image features can be learned without relying on spatial feature extraction process.
   - **Rationale:** The ANN model captures non-linear relationships across the feature space as malware patterns may vary in structure and texture.
10. **GAN Implementation**
    - **Importance:** GAN architectures provide an effective generative modeling approach that can learn underlying distribution of malware samples, enabling the generation of synthetic but realistic malware images.
    - **Rationale:** By generating synthetic malware images, this experiment directly serves in data augmentation and class balance in malware image classification tasks.
11. **Implementation with Conditional GAN (cGAN)**
    - **Importance:** The cGAN architecture was designed to generate class-specific synthetic malware images, therefore, to diversify the training dataset with diverse and realistic samples.
    - **Rationale:** The conditional adversarial learning was leveraged to augment the malware image datasets and improve the robustness of malware detection systems.

5.2.5 Detection with CNN Model using Keras Tuner

In Computer Vision and Image Processing field, Convolutional Neural Networks are most effective to deal with image classification and object detection problems. Convolutional Neural Network (CNN) is the most popular deep

learning based neural network which is being used to solve problems in machine learning, natural language processing (NLP), Computer Vision with great results and accuracy [179]. CNN is a fully connected neural network with convolutinal layers, pooling layers and dropout layers. CNN is built of different sizes of kernels for feature extraction and several neurons in different layers.

In case of a Deep Learning model finding the set of optimal parameters (hyperparameters) is the most crucial part to attain a high accuracy by the model. Hyperparameters are tunable parameters that affects the training and learning process of a model. Learning rates, number of epochs, batch sizes are some examples of hyperparameters.

Manual process of hyperparameter tuning is time consuming and not convenient at all. "KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search[3]". Keras tuner is an open source library for hyperparameter tuning with Deep Learning architectures. Keras tuner is being implemented with algorithms like Bayesian Optimization, Hyperband, Sklearn and Random Search.

In this experiment malware image datasets were used to implement with CNN for detection and keras tuner to find different hyper parameters. A custom CNN model was developed to implement on malware image datasets. This CNN model has following layers:

- Uses 4 Convolutional Layers with increasing filters.
- Batch Normalization and Dropout improve stability.
- Max Pooling (2×2) reduces spatial dimensions.
- Fully Connected Layers refine features before classification.
- Adam Optimizer + Categorical Crossentropy for best performance.

[3] https://github.com/keras-team/keras-tuner

Algorithm 2 CNN Training and Hyperparameter Tuning

Input: Dataset $D = \{(x_i, y_i)\}$, learning rate η, epochs E, batch size B
Output: Trained CNN model f_θ
Normalize $x_i \leftarrow x_i/255$, encode y_i
Apply data augmentation (shift, rotation, flip)
Split training set for validation
Initialize CNN f_θ:
  Conv layers + ReLU + batch norm
  Pooling layers
  Dense layers + dropout
  Output softmax layer
for each epoch $e = 1 \ldots E$ **do**
  Sample mini-batch $(x_{\text{batch}}, y_{\text{batch}}) \sim D$
  $\hat{y}_{\text{batch}} \leftarrow f_\theta(x_{\text{batch}})$
  $L \leftarrow$ categorical_crossentropy$(\hat{y}_{\text{batch}}, y_{\text{batch}})$
  Backpropagate L, update $\theta \leftarrow \theta - \eta \nabla_\theta L$
end for
Predict: $\hat{y}_i \leftarrow \arg\max_c f_\theta(x_i)$
Evaluate accuracy and confusion matrix
Hyperparameter tuning:
  Define search space (filters, kernel sizes, learning rate)
  Search best hyperparameters with validation split
  Retrieve best model, retrain on full training set
return f_θ

Table 5.2: Terms and Notation for CNN and Hyperparameter Tuning

Symbol / Term	Description
$D = \{(x_i, y_i)\}$	Dataset of input samples x_i and corresponding labels y_i
x_i	Input sample (image or feature vector)
y_i	Label for sample x_i
f_θ	CNN model parameterized by weights θ
θ	Trainable parameters of the CNN (weights and biases)
η	Learning rate used in gradient descent
E	Number of training epochs
B	Mini-batch size for training
x_{batch}	Mini-batch of input samples
y_{batch}	Mini-batch of labels corresponding to x_{batch}
$\hat{y}_{\text{batch}}$	Predicted labels for the mini-batch
L	Loss function (categorical cross-entropy)
$\nabla_\theta L$	Gradient of the loss with respect to CNN parameters
$\hat{y}_i$	Predicted label for input x_i
Filters / Kernel size	CNN hyperparameters defining number of convolution filters and kernel sizes
Dense units	Number of neurons in fully connected layers
Dropout rate	Fraction of neurons randomly dropped during training for regularization
Accuracy	Fraction of correctly predicted labels
Confusion matrix	Table summarizing classification performance across all classes
Keras Tuner	Tool for automated hyperparameter search
Validation split	Portion of training data reserved for evaluating model performance

Steps of CNN with Keras Tuner

The experiment follows several steps according to the Algorithm 2. Terms and notations were described in Table 5.2.

1. **Preprocess Input Data**
    - At first, normalization of pixel values to the range [0,1]:

$$x_i' = \frac{x_i}{255}$$

    - Encode labels using one-hot encoding for multi-class classification.

2. **Data Augmentation**
    - Augmentation methods like shifting, rotation, scaling, horizontal or vertical flipping were applied.
    - After that the dataset was splitted into training and validation sets.
3. **Defining CNN Architecture**

    The CNN model was built with following layers:
    - Convolutional layers with ReLU activation and batch normalization to learn spatial features. After that Pooling layers were added (e.g., MaxPooling) to downsample the feature maps.
    - At the end, fully connected (dense) layers were added with dropout for regularization and output layer with softmax activation.
4. **Initialize Training Parameters**
    - The optimizer, learning rate, batch size, number of epochs and validation split were set.
5. **Training of Model**
    - Training the model on mini-batches of the augmented data.
    - Update parameters θ using gradient descent:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(f_\theta(x), y)$$

6. **Prediction of Class Labels**
    - For training and test data:

$$\hat{y}_i = \arg\max_c f_\theta(x_i)$$

7. **Evaluate Performance**
    - Model's performance was computed with metrics like accuracy.
    - After that, visualization of sample predictions and confusion matrix.
8. **Hyperparameter Tuning**
    - Hyperparameter Tuning was performed by using Keras Tuner to optimize the number of filters, kernel sizes, dense units and learning rate.
    - After that retraining the best model on full training data.
    - Comparison of the performance with baseline CNN to assess the improvement.

Proposition of CNN with Keras Tuner

1. **Effectiveness of Data Augmentation**
    - **Proposition:** Data augmentation [180] such as rotation, shifting were incorporated to this experiment, to enhance the generalization capability of the CNN model for producing higher validation accuracy.
    - **Rationale:** Variants of malware images differ in a very noticeable way in their texture and structural differences [105]; augmentation emulates these variations to make the model more robust.

2. **Model Depth and Complexity**
    - **Proposition:** The CNN model was built with several convolutional, dropout and batch normalization layers so that it becomes much deeper in order to learn features hierarchically and generate better classification results.
    - **Rationale:** Malware images usually contain complex patterns [181], hence the images need deeper networks to extract the features hierarchically.

3. **Tuned Hyperparameters Enhance Performance**
    - **Proposition:** Hyperparameter tuning with Keras Tuner enhances the performance of CNN.
    - **Rationale:** Systematic tuning can make the model better fit for the features of the dataset [182].

4. **Benchmarking and Comparison**
    - **Proposition:** The CNN model outperformed the baseline model (simple CNN with fewer layers).
    - **Rationale:** Showing improvements will validate the effectiveness of hyperparameter tuning and the CNN model.

Virus MNIST Dataset Results

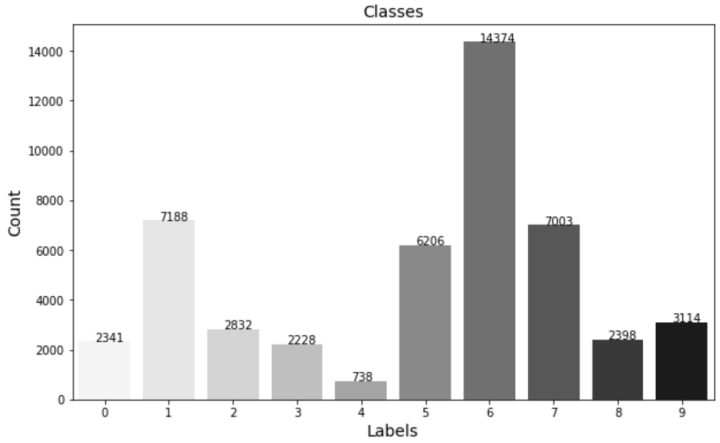


Figure 5.5: Virus MNIST dataset with labels

Figure 5.5 is showing the labels of the dataset with classes. The confusion matrix of the predicted classes is shown below (Figure 5.9). The plot of the train images with labels was shown in Figure 5.6.

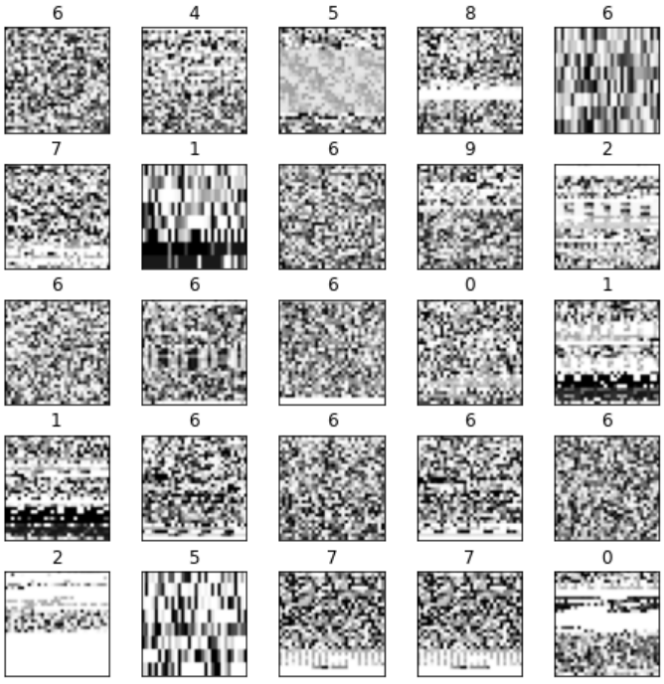


Figure 5.6: Plot of Train Images of Virus MNIST

The visualization of the average value of the image array is shown in Figure 5.7.

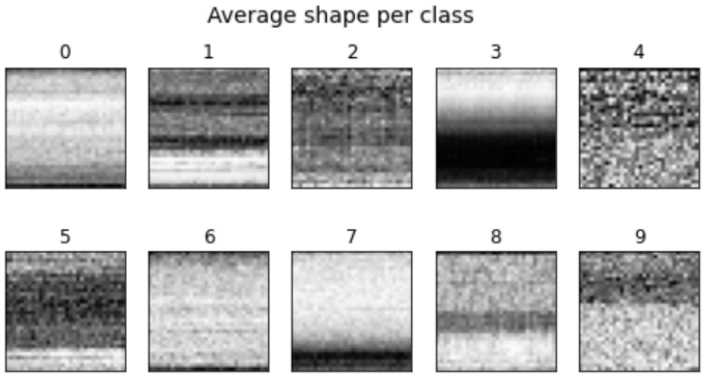


Figure 5.7: Average shape of Virus MNIST images

The Figure 5.8 is showing the prediction with test images.

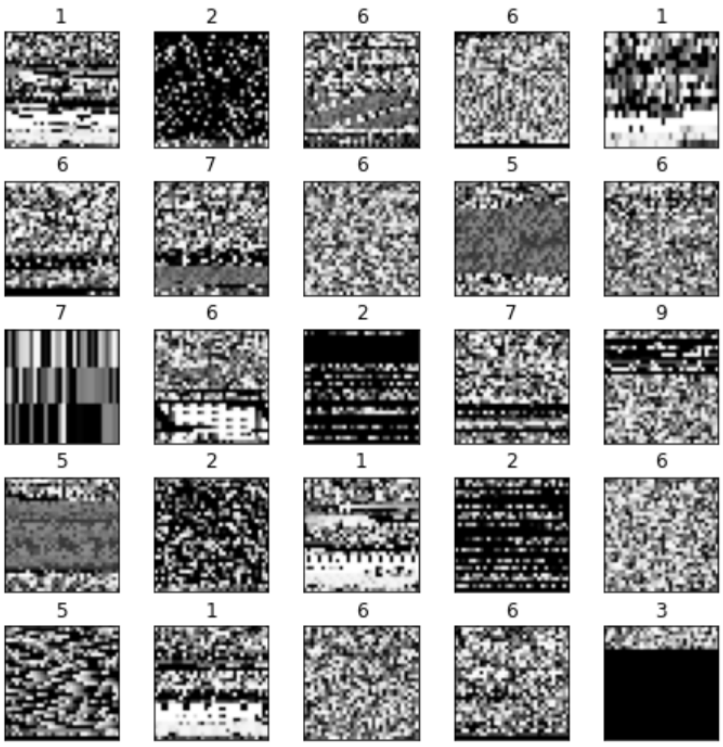


Figure 5.8: Prediction of Test Images with CNN

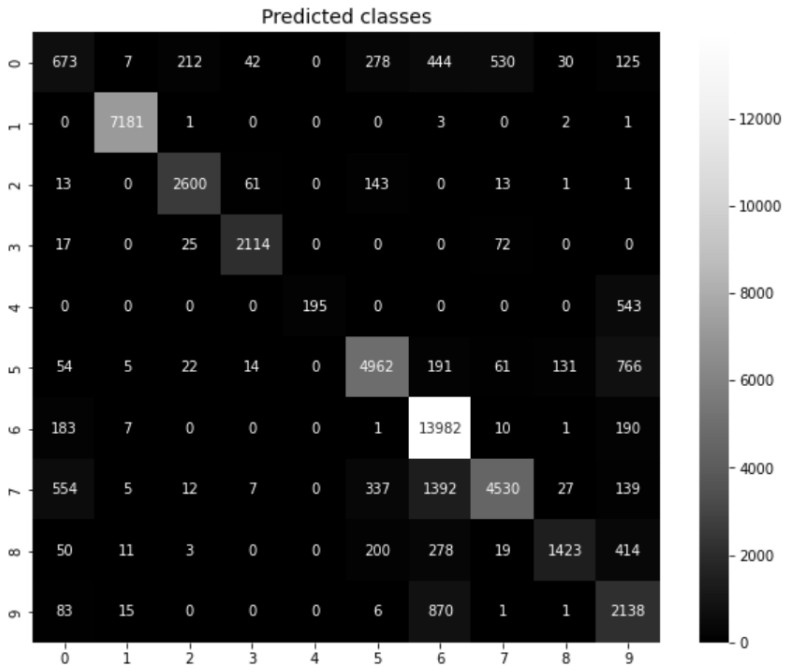


Figure 5.9: Confusion Matrix for CNN

With the above CNN model Keras Tuner was used for the hyperparameter tuning. To improve the model's performance hyperparameter tuning was used. Keras Tuner is the convenient way to search the best values of hyperparameters for the CNN model. To parameterize the variables of CNN keras tuner was used. 'RandomSearch' was selected as tuner object and the final step is to find the best values of hyperparameters for the model. Here is a diagram of Keras tuner function (Figure 5.10[4]):

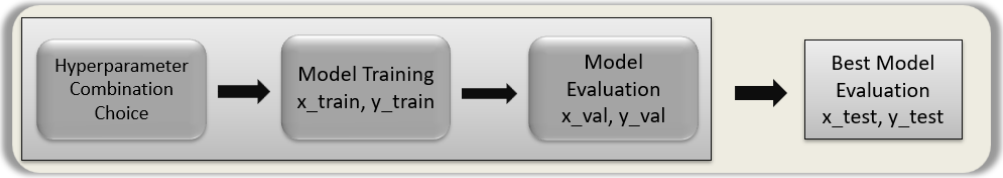


Figure 5.10: Keras Tuner function

[4] https://www.kdnuggets.com/2020/02/hyperparameter-tuning-keras-tuner.html

The model achieved better accuracy for each classes with Keras Tuner. Figure 5.11 is showing the confusion matrix of the result.

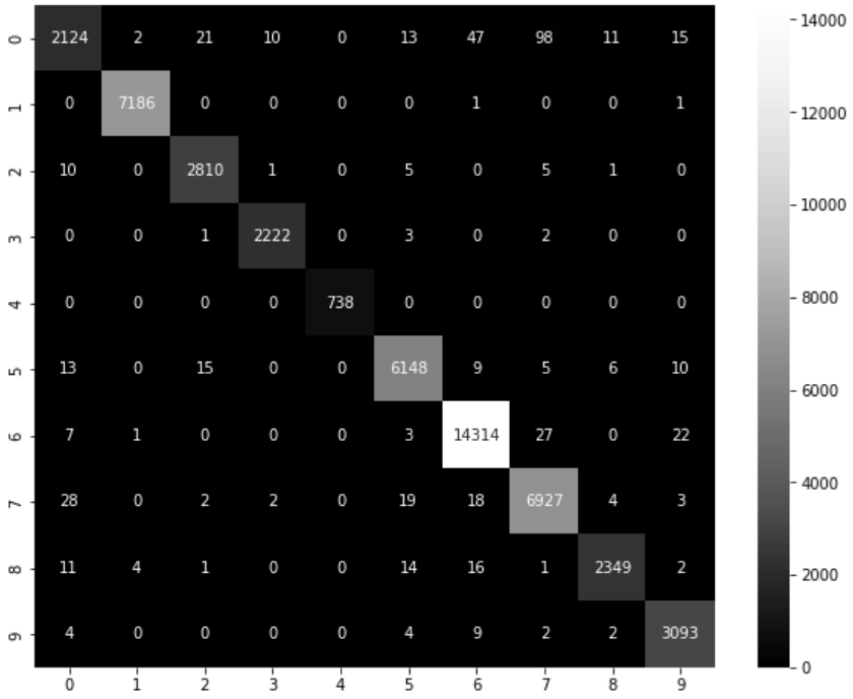


Figure 5.11: Confusion Matrix of CNN with Keras Tuner

MaleVis Dataset Results

Figure 5.12 is showing the confusion matrix of MaleVis dataset.

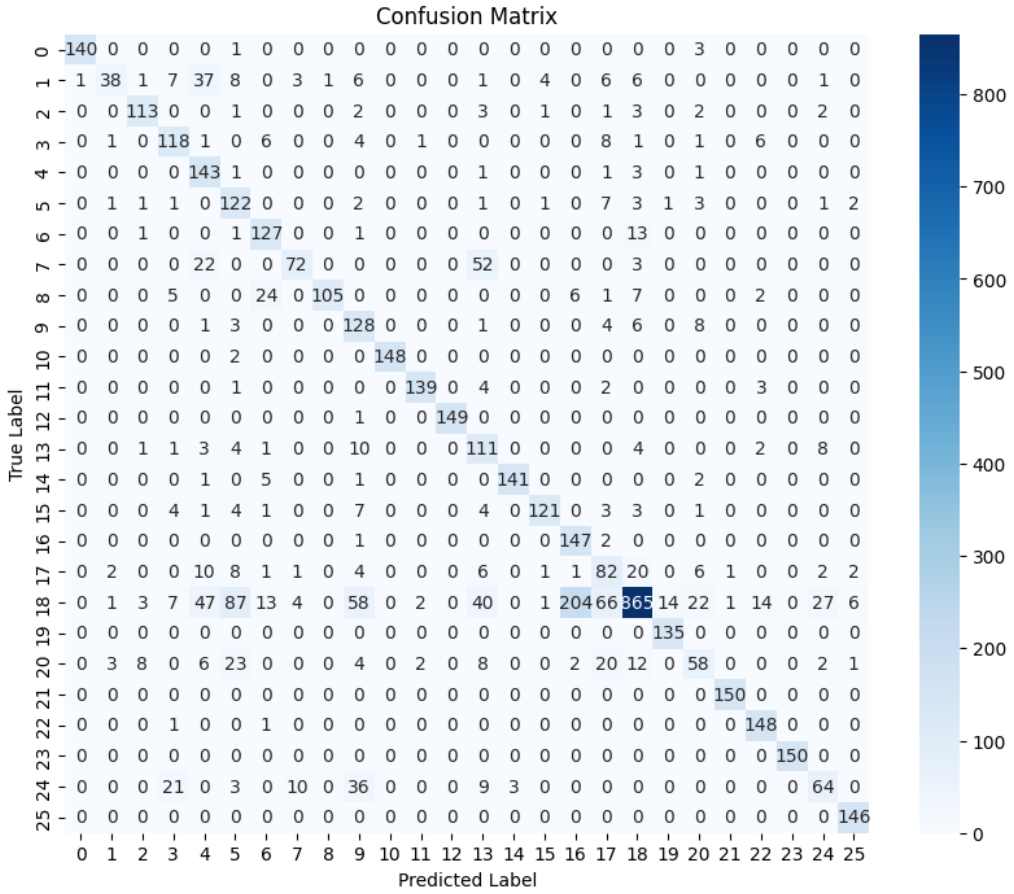


Figure 5.12: MaleVis Confusion Matrix

Table 5.3 and Table 5.4 are showing the results of three datasets with CNN and CNN with Keras Tuner, respectively.

Table 5.3: CNN Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	97.05%	94.51%
Virus MNIST	91.44%	90.50%
MaleVis	85.28%	85.16%

Table 5.4: CNN Results with Keras Tuner

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	98.81%	96.50%
Virus MNIST	99.73%	92.30%
MaleVis	86.25%	85.60%

- For the Malimg dataset, both models were trained for 50 epochs. The Keras Tuner model achieved 98.81% training and 96.50% validation accuracy, surpassing the baseline CNN (97.05% and 94.51%). This steady improvement suggests that the Keras Tuner achieved better feature extraction and convergence without overfitting.
- In the Virus MNIST experiment, the Keras Tuner CNN reached 99.73% training accuracy and 92.30% validation accuracy by using half the epochs of the baseline (50 vs. 100). The hyperparameter tuning accelerated training and maintained higher validation performance. Here the models was highly capable compare to the smaller dataset size.
- For MaleVis dataset both the models were trained for 30 epochs, improvements were 85.16% to 85.60%. Here the Keras Tuner model added marginal benefits.

Critical Analysis of Incorporating Keras-Tuner

The use of Keras Tuner in this experiment adds a layer of novelty to the hyperparameter tuning process:

- **Automated Hyperparameter Search:** By incorporating Keras Tuner this method becomes an automated and systematic process for the best hyperparameters. In our experiment, this efficiency was exercised, where a wide range of configurations needed human intervention.
- **Resource-Efficient Hyperparameter Tuning:** Keras Tuner efficiently uses available resource for hyperparameter tuning. It offers smart hyperparameter space search in such a way that it may find a good configuration without extreme computational costs [183].

- **Integration with Keras Models:** Keras Tuner was integrated with the Keras deep learning library. In this way, the experiment can use Keras functionalities to construct and train a model while performing hyperparameter optimization.
- **Enhanced Model Generalization:** Keras Tuner is used to generalize the model better by finding those hyperparameters that can give better results on the validation dataset.

5.2.6 Detection with TPU

TPUs or Tensor Processing Units are power efficient to train large scale machine learning and deep learning models. It can improve the performance of a neural network by allocating small number of resources [184]. TPUs are the hardware accelerators which train the models with matrix computations [185]. These are able to train huge ML/DL models with large batch sizes. Developed by Google, TPUs are available in Google-based platforms and being used for TensorFlow operations. TPUs provide high latency and can process complex deep learning models.

Algorithm 3 CNN Training with TPU Strategy

Input: $D = \{(x_i, y_i)\}$, learning rate η, epochs E, batch size B
Output: Trained CNN f_θ
Normalize $x_i \leftarrow x_i/255$
Encode y_i, split D into train/val
Initialize strategy: TPU
with strategy.scope():
  Initialize f_θ: Conv + ReLU + Pool + Flatten + Dense + Softmax
  Compile with optimizer and loss
for epoch $1 \dots E$ **do**
  Sample batch (x_b, y_b)
  $\hat{y}_b \leftarrow f_\theta(x_b)$
  $L \leftarrow loss(\hat{y}_b, y_b)$
  Update $\theta \leftarrow \theta - \eta \nabla_\theta L$
end for
Load best weights
Evaluate on val/test: accuracy, confusion matrix
return f_θ

Table 5.5: Terms and Notations for CNN Training with TPU

Symbol/Term	Description
D	Dataset of samples $D = \{(x_i, y_i)\}$
x_i	Input image or feature vector
y_i	True label of sample x_i
B	Batch size
E	Number of training epochs
η	Learning rate
f_θ	CNN model with parameters θ
θ	Trainable weights of the CNN
$\hat{y}_b$	Predicted labels for batch (x_b, y_b)
L	Loss function (e.g., categorical cross-entropy)
$\nabla_\theta L$	Gradient of loss w.r.t. model parameters
$strategy$	TPU distribution strategy
$train/val/test$	Training, validation, and test splits of D
Conv, Pool, Dense	Convolutional, pooling, and fully connected layers
Softmax	Output activation function for multi-class classification

Steps of CNN Training with TPU Strategy

According to Algorithm 3 the experiment was designed. Terms and notation were described in Table 5.5.

1. **Preprocess Input Data**
   - At first, normalize the pixel values to [0,1]:

$$x_i' = \frac{x_i}{255}$$

2. **Initialize TPU Strategy**
   - Connect to TPU:

$$\text{strategy} = \text{TPUStrategy}$$

   - Adjust batch size according to number of TPU replicas:

$$B_{\text{scaled}} = B \times \text{strategy.num\_replicas\_in\_sync}$$

3. **CNN Architecture**

   The CNN model was implemented with following layers:
   - Conv2D + ReLU + Pooling layers for feature extraction.

- Dense layers with dropout for regularization.
- Output layer with softmax activation.

4. **Compile Model**
    - Optimizer: Adam was selected with learning rate η.
    - Loss: Sparse or categorical crossentropy.
    - Metrics: accuracy.
5. **Train Model**
    - Fit the model on training dataset using TPU strategy.
    - Callbacks like ModelCheckpoint, ReduceLROnPlateau, EarlyStopping were applied.
6. **Predict and Evaluate**
    - Prediction of class labels for validation/test data.
    - Computation of metrics: accuracy, confusion matrix.
    - Visualization of predicted samples.

Virus MNIST Dataset Results

In this research, the experiment was carried out for the detection of malware using TPUs to develop the CNN architecture as TPUs are much faster than GPUs in terms of processing neural networks. At first, the images were reshaped and divided into train, test and validation datasets. The size of train, validation and test split was shown in Figure 5.13 and the display of train images was shown in Figure 5.14.

```
Shape of training data: (43579, 32, 32, 1)
Shape training labels: (43579, 10)
Shape of validation data: (4843, 32, 32, 1)
Shape of valiation labels: (4843, 10)
Shape of testing data: (3458, 32, 32, 1)
```

Figure 5.13: Train, Validation and Test

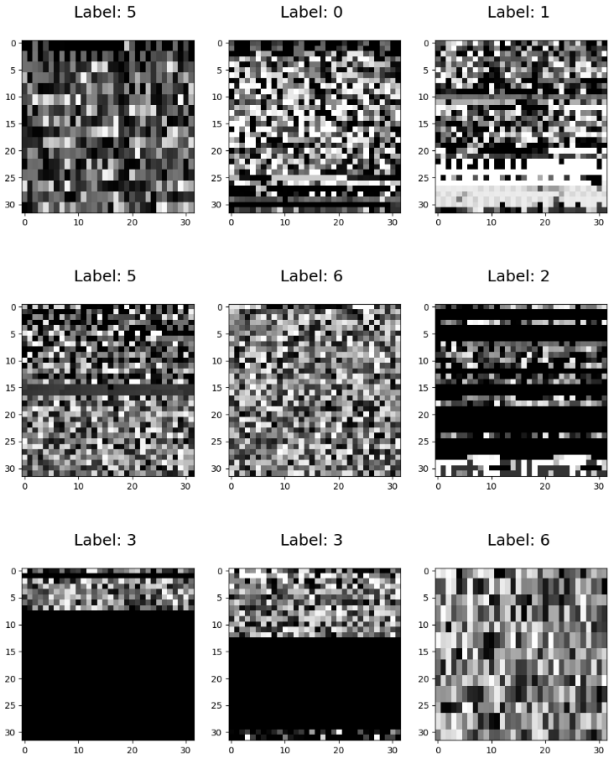


Figure 5.14: Train Images of Virus MNIST

Figure 5.15 is showing the confusion matrix and Figure 5.16 is showing the accuracy of the model. The plot of predicted images was shown in Figure 5.17.

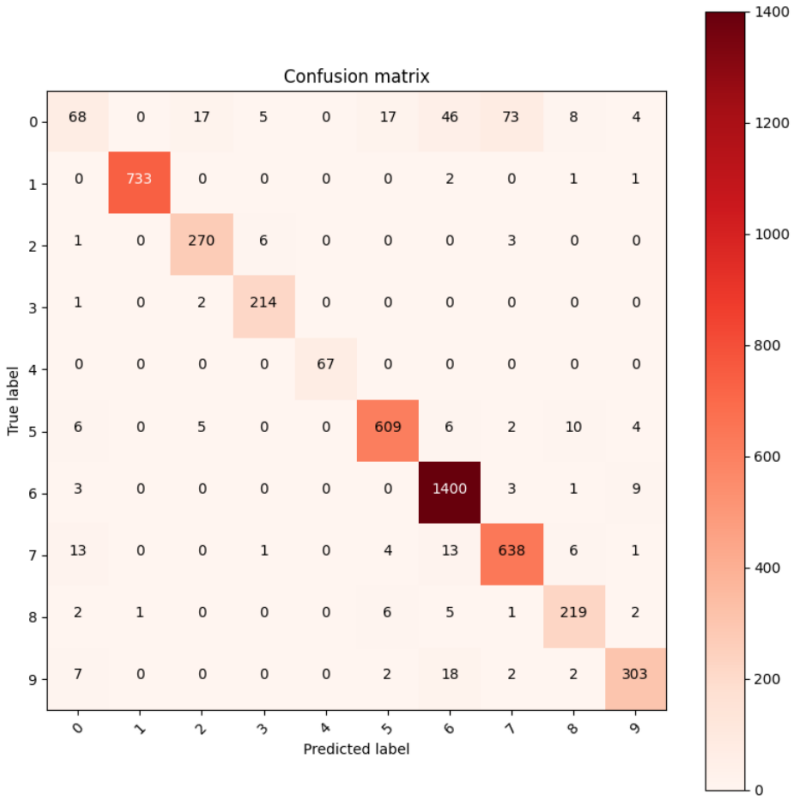


Figure 5.15: Virus MNIST Confusion Matrix

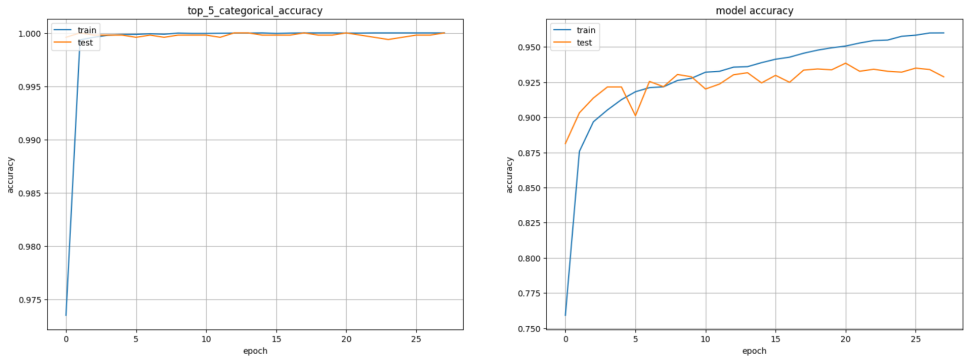


Figure 5.16: Virus MNIST Model Accuracy

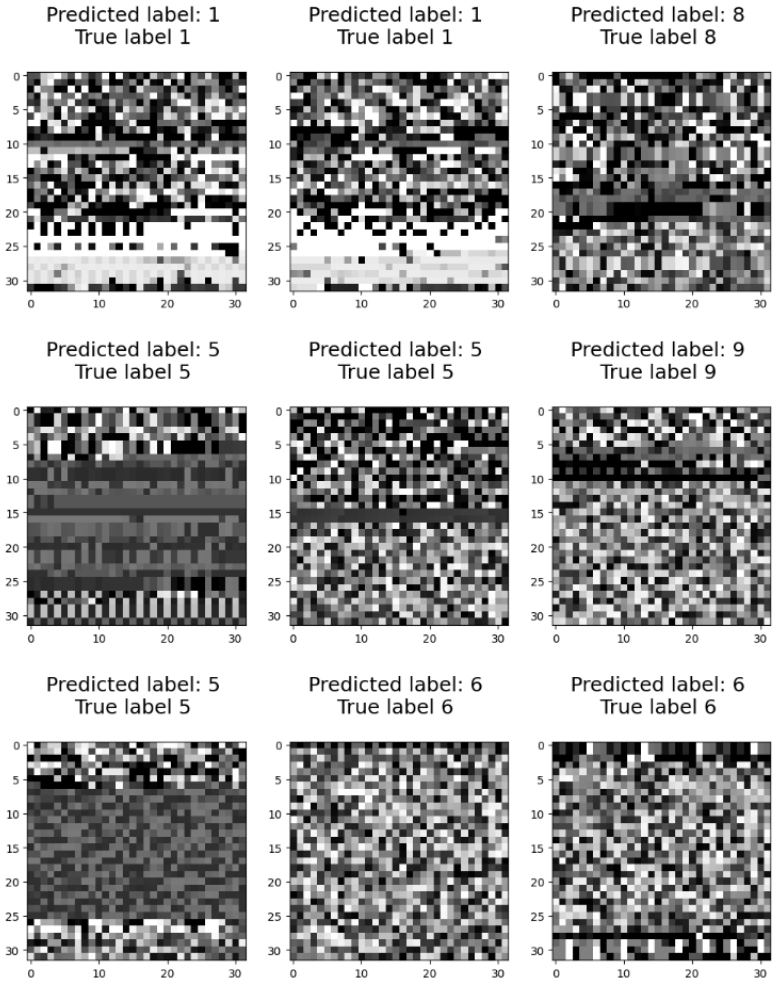


Figure 5.17: Predicted Labels of Test Images

Table 5.6: TPU Experiment Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	99.68%	98.05%
Virus MNIST	96.03%	93.78%
MaleVis	99.35%	71.62%

- All the experiments were trained for 50 epochs. Two datasets (Malimg, Virus MNIST) produced very high training and validation accuracies. Results indicate good generalization and convergence under TPU training.

- MaleVis produced perfect training accuracy but low validation performance which indicates overfitting and generalization gap.

MaleVis Dataset Results

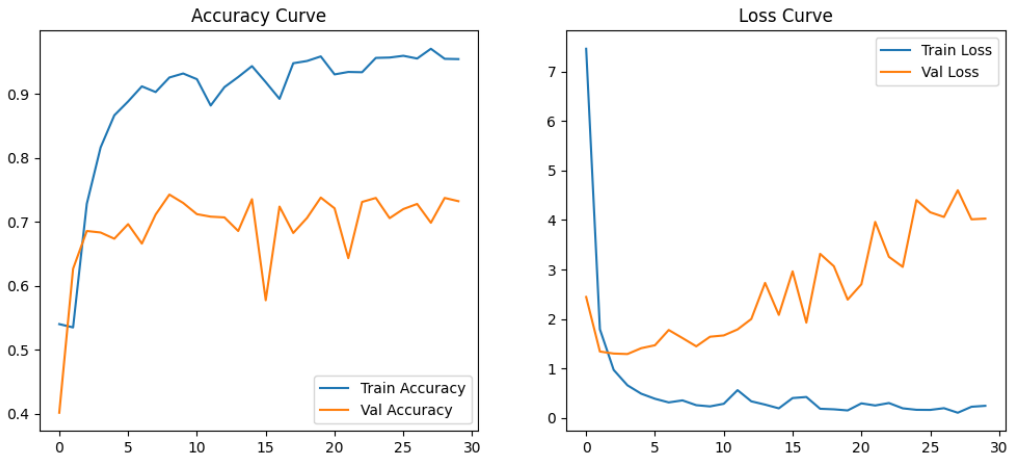


Figure 5.18: MaleVis TPU Accuracy and Loss

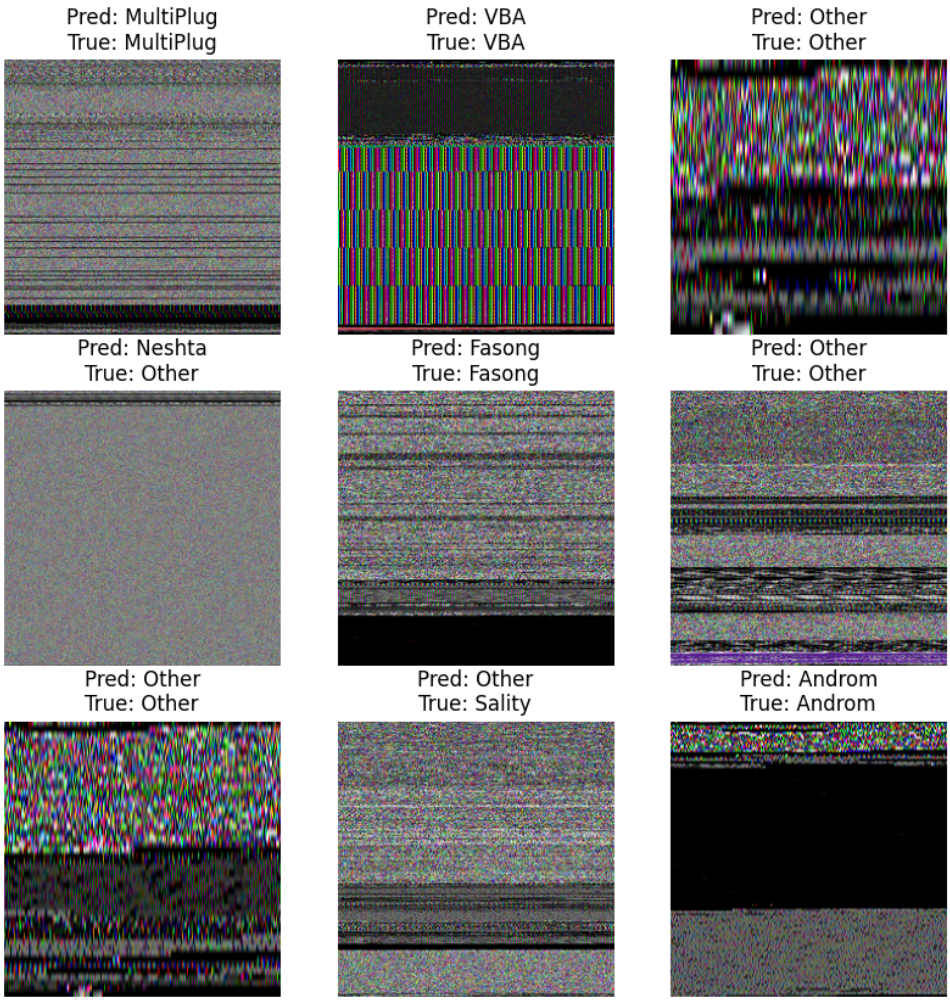


Figure 5.19: MaleVis Prediction Results

Malimg Dataset Results

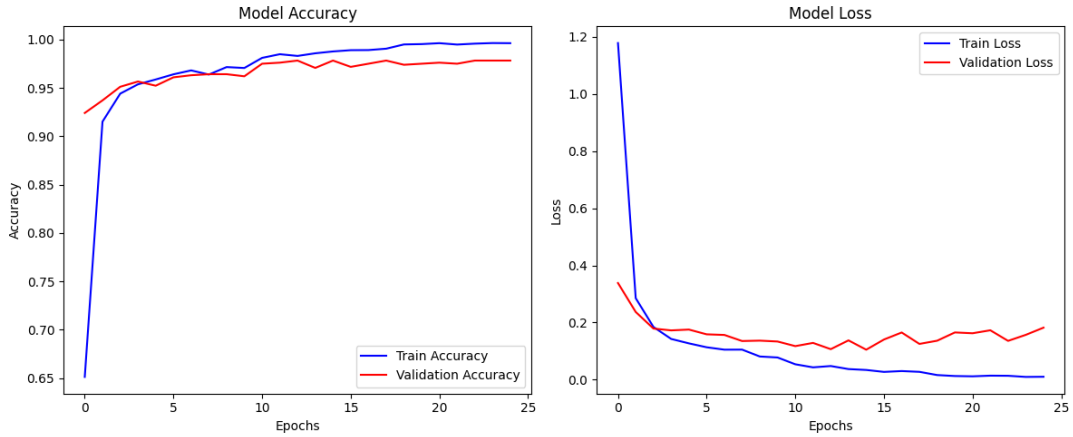


Figure 5.20: Malimg TPU Accuracy and Loss

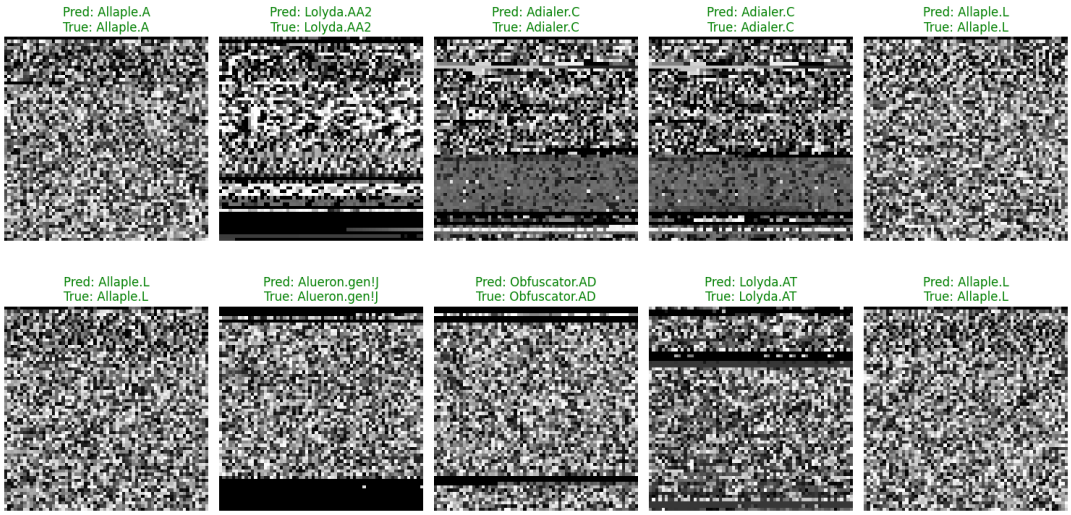


Figure 5.21: Malimg Dataset Predictions

Critical Analysis of TPU Utilization in Research

- **Performance Enhancement:** TPUs are custom-built hardware accelerators for deep learning work [185]. By utilizing TPUs, this deep learning

experiment significantly accelerated the computation, particularly the CNN architecture, enabling quicker experimentation and training.

- **Scalability:** TPU provides scalability in an experiment. In this experiment, it has made it possible to handle large datasets and the CNN model more efficiently.
- **Speed:** TPU probably reduced the training time compared to other traditional CPU or GPU settings [184]. Such acceleration is highly relevant for large-scale experiments and iterative model refinement.

5.2.7 Classification with ResNet-50 Transfer Learning

In Deep Learning ResNet architecture consists of the residual blocks. ResNet-50 is a type of Convolutional Neural Network with 50 layers to make the model extremely deep. It is part of the ResNet (Residual Network) family. The key innovation of ResNet is the introduction of residual connections or skip connections [186]. These connections allow the network to skip one or more layers, effectively "bypassing" them and passing the input directly to the later layers. This helps mitigate the vanishing gradient problem, where gradients become too small during backpropagation in deep networks, making it hard for the network to learn effectively. Figure 5.22[5] is showing a ResNet-50 architecture.

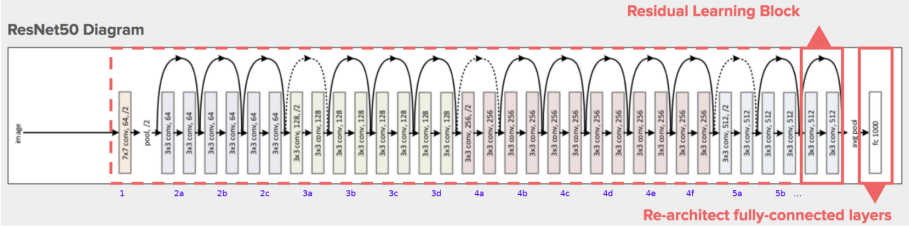


Figure 5.22: A ResNet-50 architecture

[5] https://stackoverflow.com/questions/54207410/how-to-split-resnet50-model-from-top-as-well-as-from-bottom

Virus MNIST Dataset Results

In this experiment, random erasing was used as it chooses a rectangular region from an image of any arbitrary size and assign the pixels of the selected region with random values. This is the process to generate the training images with various occlusions to reduce overfitting. Pre-trained ResNet-50 was leveraged with transfer learning which helps to build the model accurately and saves time.

The Virus MNIST .csv files were loaded at first for preprocessing. The image arrays were reshaped with 32*32 size as there are 1024 number of pixels. The image array was resized into (75,75) shape and converted into 'RGB' mode.

After that the dataset was split into 80% train and 20% validation images to fit into the model. Figure 5.23 is showing the accuracy and loss of the model performance.

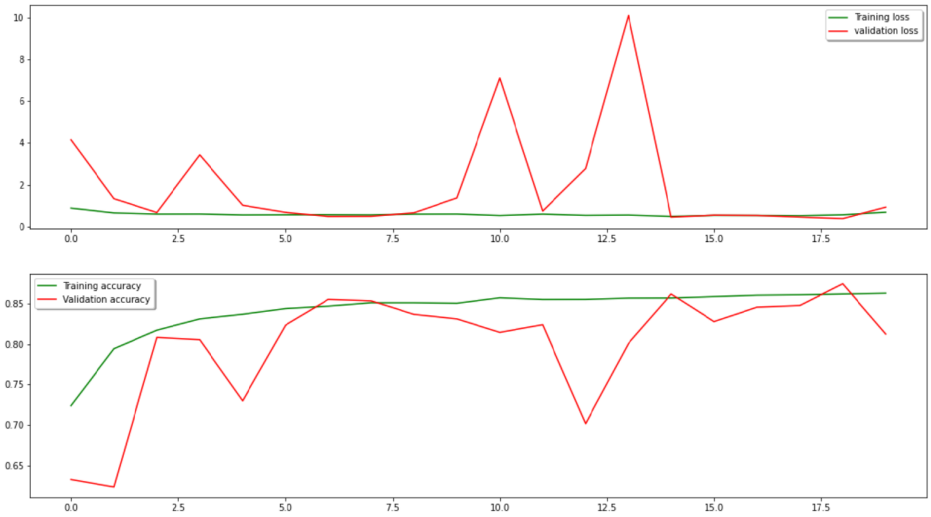


Figure 5.23: Accuracy and Loss Curves of ResNet-50 Transfer Learning

The model achieved 86.24% training accuracy and 81.22% validation accuracy. Figure 5.24 is showing the prediction of the model on test images where the model has predicted two images of label 2 and label 7.

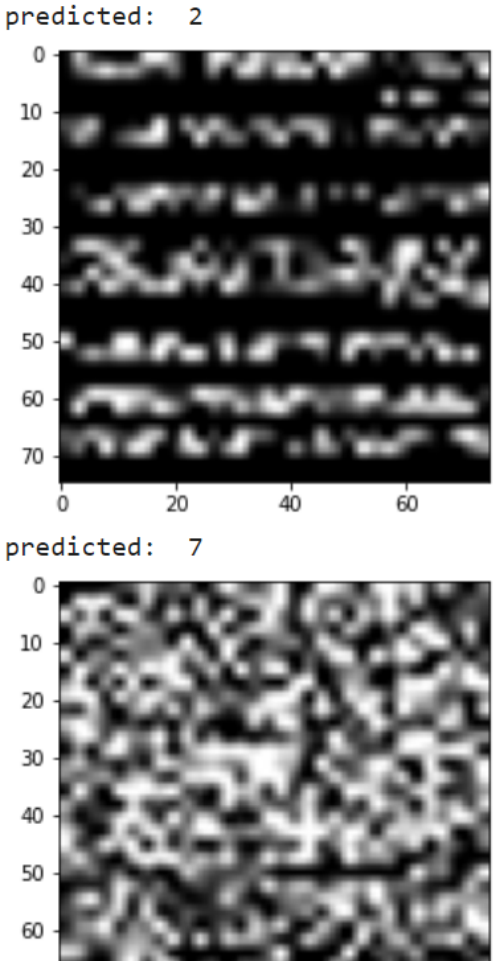


Figure 5.24: Prediction on Test Images for ResNet-50 Transfer Learning

MaleVis Dataset Result

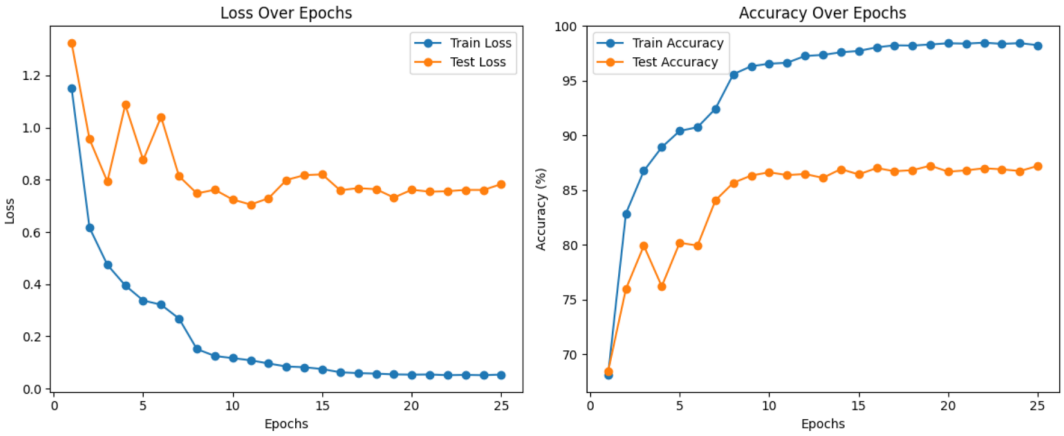


Figure 5.25: Accuracy and Loss Curve of MaleVis Dataset

Algorithm 4 Transfer Learning for Malware Image Classification

Require: Dataset D, pretrained model M, hyperparameters (B, η, E)
Ensure: Trained model M, evaluation metrics
  Split $D \rightarrow (D_{train}, D_{test})$
  Preprocess: resize (H, W, C), normalize
  Augment D_{train}
  Create mini-batches of size B
  Initialize M, freeze early layers
  Replace final layer $\rightarrow |classes|$
  Loss $L \leftarrow CE$, optimizer $\leftarrow$ Adam/RMSprop(η)
  Scheduler $\leftarrow$ init()
  for epoch $= 1 \ldots E$ **do**
    train mode(M)
    for batch $(x, y) \in D_{train}$ **do**
      $\hat{y} \leftarrow M(x)$
      $L \leftarrow CE(\hat{y}, y)$
      backprop(L), update(M)
    end for
    eval(M, D_{test})
    step(scheduler)
  end for
  acc, cm, report $\leftarrow$ evaluate(M, D_{test})
  return M, (acc, cm, report)

Table 5.7: Terms and Notation for Transfer Learning Pipeline

Symbol / Term	Description
D	Dataset of images and corresponding labels
D_{train}	Training split of the dataset
D_{test}	Test/validation split of the dataset
x, x_i	Input image or sample
y, y_i	Label for an input sample
M	Pretrained model used for transfer learning (e.g., ResNet-50)
B	Mini-batch size for training
E	Number of training epochs
η	Learning rate for optimizer
$\hat{y}, \hat{y}_i$	Predicted label or probability distribution from the model
L	Loss function (cross-entropy loss)
Optimizer	Algorithm for updating model parameters (Adam or RMSprop)
Scheduler	Learning rate scheduler to adjust η during training
Freeze layers	Early layers of pretrained model kept constant during fine-tuning
Final layer / $\lvert classes \rvert$	Output layer replaced to match the number of malware classes
Accuracy	Fraction of correctly predicted labels
Confusion matrix (cm)	Table summarizing classification performance across all classes

Steps of Transfer Learning for Malware Image Classification

The experiment was conducted according to Algorithm 4, where terms and notations were described in Table 5.7.

1. **Load and Split Dataset**
    - The dataset D containing malware images and labels was loaded.
    - After that it was splitted into training set D_{train} and validation or test set D_{test}.
2. **Preprocess Images**
    - The images were resized in adequate shape.
    - Normalization of pixel values to [0,1]. Here, the purpose is to stabilize training.

3. **Apply Data Augmentation**
    - Application of augmentation methods such as rotation, flipping, scaling on D_{train} to increase data diversity and reduce overfitting.
4. **Initialize Pretrained Model**
    - Loading of the pre-trained CNN model (ResNet-50 trained on ImageNet).
    - Freezing of early layers to preserve pre-trained feature representations and replace the final classification layer with a new output layer.
    - Here the motivation is to leverage pre-trained features while adapting the malware classification.
5. **Define Loss Function and Optimizer**
    - Cross-Entropy Loss L_{CE} was used for multi-class classification.
    - Optimizer (Adam/RMSprop) was applied with the learning rate η.
    - After that a learning rate scheduler was set up, if needed.
6. **Train the Model**
    - For each mini-batch (x, y) of size B:

$$\hat{y} = M(x), \quad L = \text{CE}(\hat{y}, y)$$

    - Performed backpropagation and update the trainable parameters of unfrozen layers using optimizer.
    - Evaluate the performance of model on the validation set after each epoch.
    - Adjust learning rate dynamically based on scheduler.
7. **Evaluate Performance**
    - Prediction of the labels on test/validation dataset.
    - Computation of metrics such as accuracy and confusion matrix.

Proposition for ResNet-50 Transfer Learning

1. **Transfer Learning with ResNet-50**
   - **Proposition:** In this experiment, the model relies on a pre-trained ResNet-50 with ImageNet weights and uses the rich feature representations from a large dataset for malware image classification.
   - **Rationale:** Transfer learning reduces training time and enhances model performance [187].
2. **Advanced Data Augmentation**
   - **Proposition:** Normally, different methods of image augmentation are applied with the purpose of increasing the size and variability of the dataset. In the current experiment, random erasing and geometric transformations such as rotation, shift were applied.
   - **Rationale:** Data Augmentation helps in reducing overfitting, improving generalization [188].
3. **Dynamic Image Resizing**
   - **Proposition:** Grayscales resized to RGB format according to the input requirement of ResNet-50 and retaining relevant features.
   - **Rationale:** This resizing and color conversion will make the dataset compatible with ResNet-50 without much loss of feature information.

Table 5.8: ResNet-50 Transfer Learning Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	84.17%	82.66%
Virus MNIST	86.24%	81.22%
MaleVis	98.25%	87.22%

- The experiment with Virus MNIST was trained for 20 epochs and experiments with other two datasets were trained for 25 epochs.
- ResNet-50 Transfer Learning produced most effective result for MaleVis as compared to other two datasets.
- Malimg and Virus MNIST achieved moderate results indicates that dataset characteristics (RGB vs grayscale) heavily influenced the results.

Critical Analysis of ResNet-50 Transfer Learning

- **Data Augmentation with Random Erasing:**

  The application of random erasing as a data augmentation technique is unique, in this experiment. Random erasing introduces randomness by erasing random patches of the image, encouraging the model to be more robust and invariant to certain transformations [189].

- **Image Resizing and Memory Optimization:**

  The resizing of images and the subsequent memory optimization are choices that can impact the model's performance. Resizing generally captures more intricate details and memory optimization that ensures efficient use of resources [190].

- **Selective Layer Freezing:**

  In this experiment, freezing of layers up to a specific point in the pre-trained ResNet-50 model was beneficial to fine-tune on malware-related features while preserving the knowledge gained from general image recognition.

5.2.8 Classification with 3D CNN

Convolutional Neural Networks have always generated promising solutions in any Deep Learning based problems. In recent years, 3D CNN [191] is being used to process the 3D volumetric data therefore the framework has a great importance to process Magnetic Resonance Imaging (MRI) data. In case of 3D CNN model the input shape is always 4 dimensional. The input and output data are also 4 dimensional.

3D Convolutional Neural Network Experiment

The following steps outline the procedure for conducting a 3D CNN experiment according to Algorithm 5 (notation were described in Table 5.9):

1. **Data Preparation**
   - After loading the dataset, the images were normalized by scaling pixel intensities to the range $[0, 1]$.

- The data were then reshaped into four-dimensional tensors in the form of (H, W, d, C).
- Here, H and W denote height and width, d represents the depth (third spatial dimension) and C indicates the number of channels (e.g., grayscale or RGB).
- The dataset was divided into training, validation and test subsets.

2. **Model Design**
   - A 3D CNN architecture was built to capture spatial dependencies across height, width and depth.
   - The model consists of 3D convolutional layers and 3D MaxPooling layers, the number of filters were increased gradually. After convolutional layers, a 3D GlobalAveragePooling layer or flattening operation was applied.
   - Finally, one or more fully connected layers were included and ended with a softmax layer for multi-class classification.
3. **Training Phase**
   - An appropriate optimizer (Adam or SGD) was selected along with a loss function.
   - The model was trained and for each epoch the model was processed into mini-batches, generated predictions and computes the loss.
   - Performance of both training and validation data was monitored to avoid overfitting.
4. **Evaluation** Once training is complete, the model is evaluated on the test dataset to assess its generalization capability.

Algorithm 5 3D CNN Model Pipeline

Require: Dataset D, input size (H, W), depth d, batch size B, number of classes C, epochs E
Ensure: Trained 3D CNN model M
  Normalize pixel values $\in [0, 1]$
  Split $D \rightarrow (D_{train}, D_{val})$
  for each image $x \in D$ **do**
    Expand image to depth d
    Reshape $x \in \mathbb{R}^{H \times W \times 1} \rightarrow x' \in \mathbb{R}^{H \times W \times d \times 1}$
  end for
  Initialize 3D CNN model M:
    Conv3D $\rightarrow$ ReLU $\rightarrow$ MaxPool3D
    Conv3D $\rightarrow$ ReLU $\rightarrow$ MaxPool3D
    Conv3D $\rightarrow$ ReLU $\rightarrow$ MaxPool3D
    Global Average Pooling
    Dense layer with C outputs + Softmax
  Compile model: Optimizer = Adam, Loss = Sparse Categorical Crossentropy, Metric = Accuracy
  for epoch = 1 ... E **do**
    Train M on D_{train} in batches of size B
    Validate M on D_{val}
  end for
  Save model M
  Plot accuracy and loss curves
  return M

Table 5.9: Terms and Notation for 3D CNN Model Pipeline

Symbol / Term	Description
D	Dataset of images
D_{train}	Training split of the dataset
D_{val}	Validation split of the dataset
x, x_i	Input image or sample
x'	Input image reshaped for 3D CNN with depth dimension
H, W	Height and width of the input image
d	Depth dimension for 3D expansion
B	Mini-batch size for training
C	Number of classes
E	Number of training epochs
M	3D CNN model
Conv3D	3D convolution layer
MaxPool3D	3D max pooling layer
Global Average Pooling	Reduces feature map to a single vector
Dense layer + Softmax	Fully connected layer with C outputs and softmax activation
Loss	Sparse Categorical Crossentropy
Optimizer	Adam optimizer
Accuracy	Metric to evaluate correct predictions

Proposition for using 3D CNN

1. **Leveraging 3D Convolutions**
   - **Proposition:** The focus is to improve the classification performance, 3D convolutional layers were used in this experiment to effectively capture the spatial patterns in malware images.

     Rationale: 3D convolutions can better model relationships across neighboring pixel groups [192].

2. **Importance of Data Normalization**
   - **Proposition:** Normalizing image pixel values to the range [0, 1] improves model convergence and accuracy.
   - **Rationale:** Normalization ensures that the neural network receives inputs within a consistent scale [193].

3. **Multi-Layer Architecture for Feature Hierarchy**

- **Proposition:** In this experiment, the layered 3D CNN architecture progressively extracts hierarchical features, from simple edges in early layers to complex patterns in deeper layers.
- **Rationale:** Malware images contain structural characteristics that can be better captured with multi-level feature abstraction with a 3D CNN [194].

Virus MNIST Results

In this experiment, 3D CNN was used to classify the malware images where the input shape was (32, 32, 1, 1). The fourth dimension is the colour channel as the images are grayscale the colour channel will be 1. The kernel size is (3, 3, 1) and the pooling size is (2, 2, 1). Figure 5.26 is showing the train images before feeding into the model.

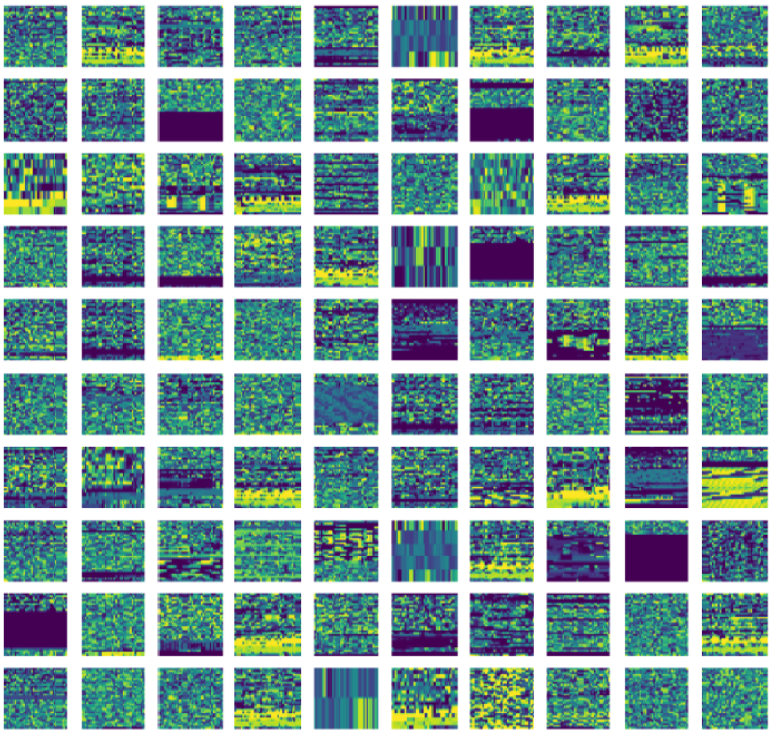


Figure 5.26: Train images for 3D CNN

Figure 5.27 is showing the loss and accuracy curve of the model performance after 30 epochs.

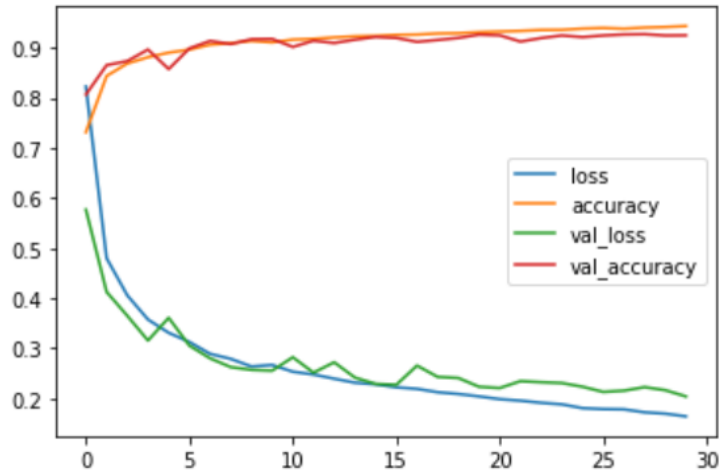


Figure 5.27: Loss and Accuracy Curve of 3D CNN

MaleVis Dataset Results

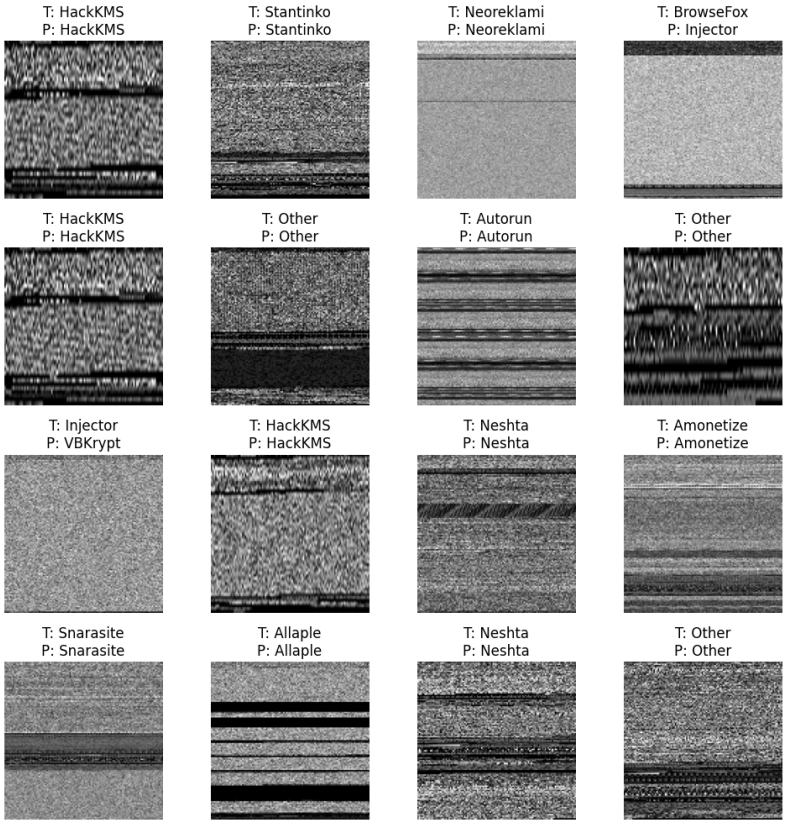


Figure 5.28: MaleVis Prediction Results

Malimg Dataset Results

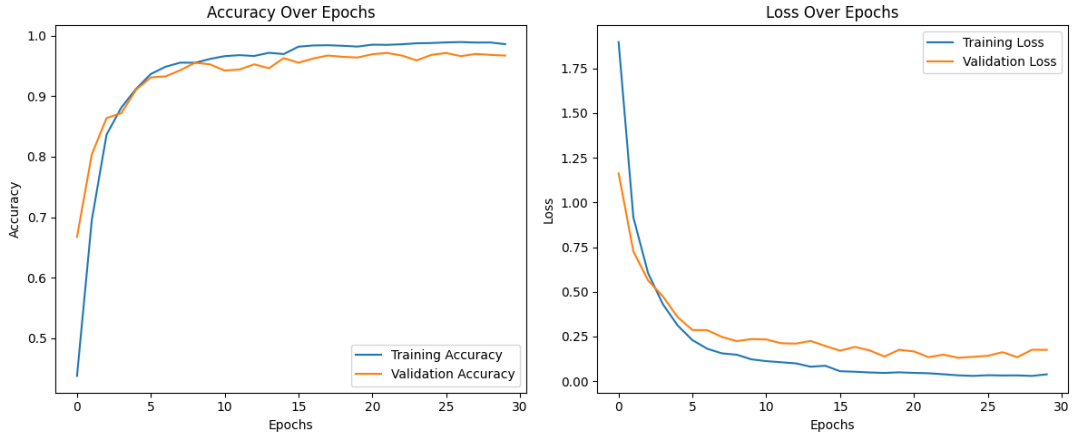


Figure 5.29: Malimg Accuracy and Loss Curve

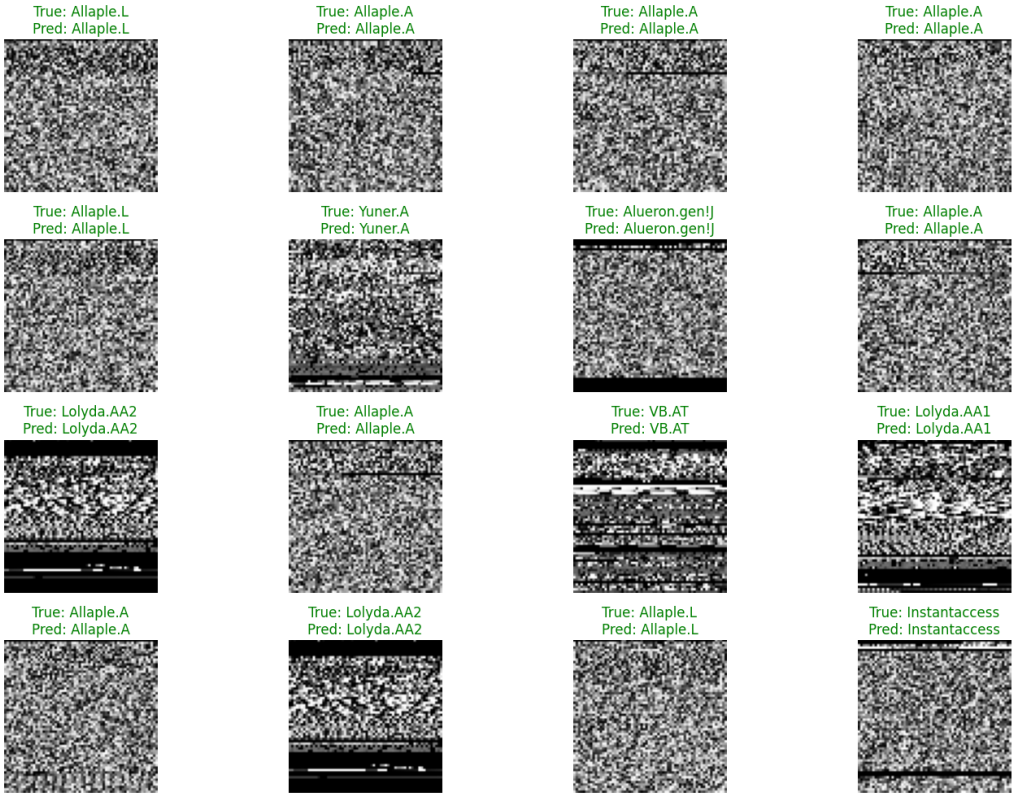


Figure 5.30: Malimg Dataset Prediction Results

Table 5.10: 3D CNN Experimental Results

Results		
Dataset	Training Accuracy	Validation Accuracy
Malimg	98.75%	96.72%
Virus MNIST	94.36%	92.51%
MaleVis	92.32%	81.39%

Table 5.10 is showing that:

- 3D CNNs performed best on Malimg, followed by Virus MNIST dataset. Both of these datasets were trained for 30 epochs.
- For MaleVis dataset, training accuracy is high but validation accuracy is much lower than Malimg and Virus MNIST, although it was trained for 40 epochs.
- Therefore, for complex dataset like MaleVis (RGB) with increasing feature space, 3D CNN may require selective regularization to prevent overfitting.

Critical Analysis of using 3D CNN

- **Use of 3D Convolutional Neural Network (CNN):** A 3D CNN being used implies that the dataset consists of three dimensional spatial information is especially suitable for those scenarios: where volumetric structures show up in the data [195].
- **Model Architecture for 3D Data:** The architecture of the neural network is oriented for 3D data: it contains Conv3D, max-pooling in 3D, and a global average pooling layer. With such a setting, this presents an attempt at the capturing of spatial relations in three dimensions other than conventional 2D CNN as used in image classification tasks.
- **Global Average Pooling in 3D:** Another interesting design follows the usage of a Global Average Pooling layer in 3D; it would allow a more compact representation of the spatial features in the 3D volume and could be useful in the process of feature extraction efficiently to reduce overfitting [196].

5.2.9 CNN Classification with Data Augmentation

Data Augmentation is a technique which can deal with the problems like overfitting of training data and limited dataset. This technique can minimise the validation error along with training error to make the Deep Learning model more accurate [180].

According to the Algorithm 6, the following steps are outlined with terms and notations mentioned in Table 5.11:

1. **Data Preparation**
   - Loading of the dataset and splitting into training and validation subsets (e.g., 80%-20%).
   - Preprocessing of images by resizing, normalizing pixel values to $[0, 1]$, and converting to grayscale or RGB, if required.
   - Data augmentation methods like: horizontal flips, rescaling or other transformations were applied on the training set.
2. **Model Design**
   - Construction of CNN architectures consisting the layers of Conv2D $\rightarrow$ ReLU $\rightarrow$ BatchNorm $\rightarrow$ MaxPooling.
   - After convolutional layers, feature maps were flattened and Dropout was applied for regularization.
   - At the end a Dense layer was added, using Softmax activation for multi-class classification.
3. **Training Phase**
   - Training of the model for a specified number of epochs using mini-batches from the training dataset.
   - Performance was monitored on the validation/test set to detect overfitting and adjust training as needed.
4. **Evaluation**
   - Evaluation of trained model on the validation dataset to measure performance (accuracy, loss).

Algorithm 6 CNN Training with Data Augmentation

Input: $D = \{(x_i, y_i)\}$, learning rate η, epochs E, batch size B
Output: Trained CNN f_θ
Normalize $x_i \leftarrow x_i/255$
Encode y_i, split D into train/val/test
Initialize data augmentation G_{train} (flip, rotation, shift, zoom)
Initialize G_{val}, G_{test}: rescale only
Initialize CNN f_θ: Conv+ReLU+BatchNorm+Pool $\times 3$, Flatten, Dense+ReLU+Dropout $\times 2$, Dense Softmax
Compile f_θ with Adam and Sparse Categorical Crossentropy
for epoch $1 \dots E$ **do**
  Sample batch (x_b, y_b) from G_{train}
  $\hat{y}_b \leftarrow f_\theta(x_b)$
  $L \leftarrow loss(\hat{y}_b, y_b)$
  Update $\theta \leftarrow \theta - \eta \nabla_\theta L$
  Validate on G_{val}
end for
Load best weights
Evaluate f_θ on G_{test}
return f_θ

Table 5.11: Terms and Notation for CNN with Data Augmentation

Symbol/Term	Description
$D = \{(x_i, y_i)\}$	Dataset of input images x_i and labels y_i
x_i	Input image (pixel values)
y_i	Corresponding class label of x_i
η	Learning rate for optimizer
E	Number of training epochs
B	Batch size
f_θ	CNN model parameterized by θ
$\hat{y}_b$	Predicted labels for batch b
L	Loss computed on a batch (Sparse Categorical Crossentropy)
G_{train}	Data augmentation generator for training set
G_{val}	Preprocessing generator for validation set
G_{test}	Preprocessing generator for test set
θ	Trainable parameters of the CNN
Conv+ReLU+Pool	Convolutional layer, activation, pooling sequence
BatchNorm	Batch normalization layer
Flatten	Flatten layer to convert feature maps to vector
Dense	Fully connected layer
Dropout	Dropout layer for regularization
Softmax	Output activation for multi-class probability

Virus MNIST Results

The train image shape was (48422, 32, 32). Figure 5.31 is showing plot of the train images to feed into the model.

For the CNN model, the shape of the input image was (32,32,1) as the images are grayscale. The size of the kernel was (3, 3) in the convolution layer and (2, 2) in the pooling layer with dropout rate 0.5. The model achieved 97.51% training accuracy and 92.13% test accuracy.

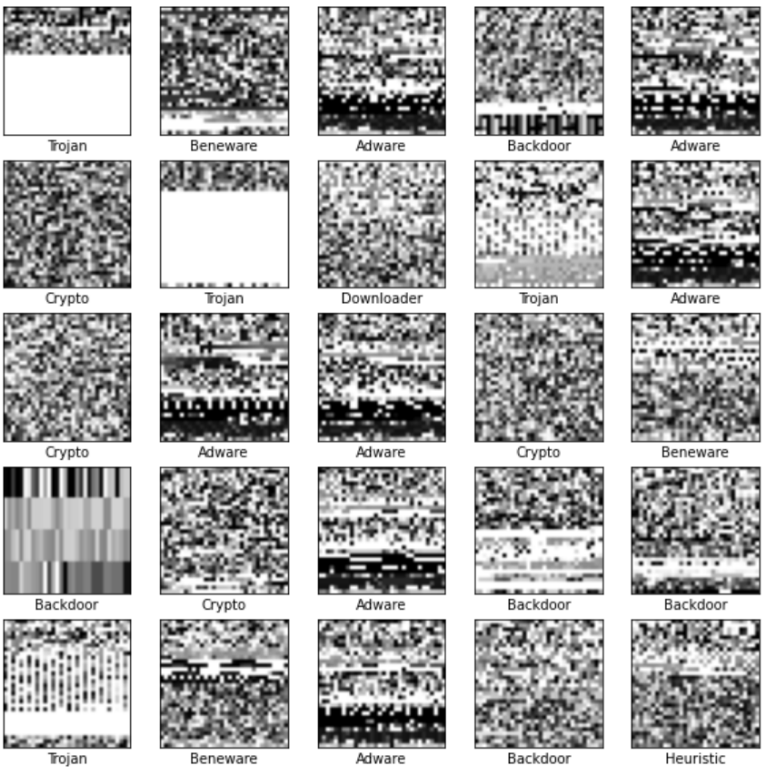


Figure 5.31: Plot of Train Images

The model was tested on the test dataset. To predict the true label of the test images colour 'blue' was used and for the incorrect label 'red' colour was used. Figure 5.32 and Figure 5.33 are showing the predictions of Trojan and Downloader for test images.

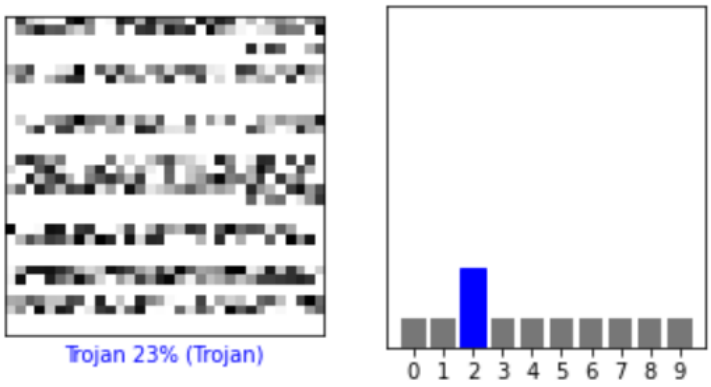


Figure 5.32: Prediction of Trojan

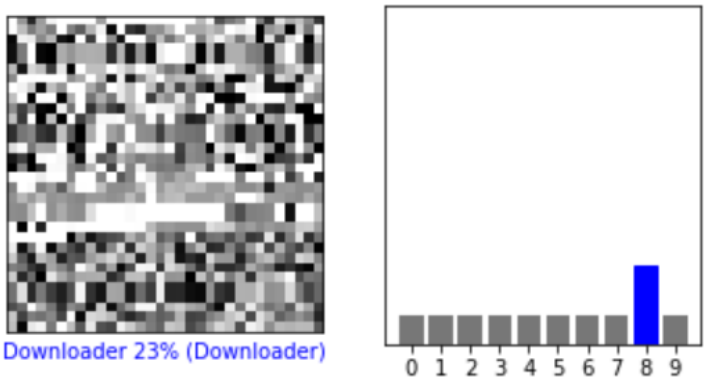


Figure 5.33: Prediction of Downloader

Figure 5.34 is showing the plot of predictions for the test images.

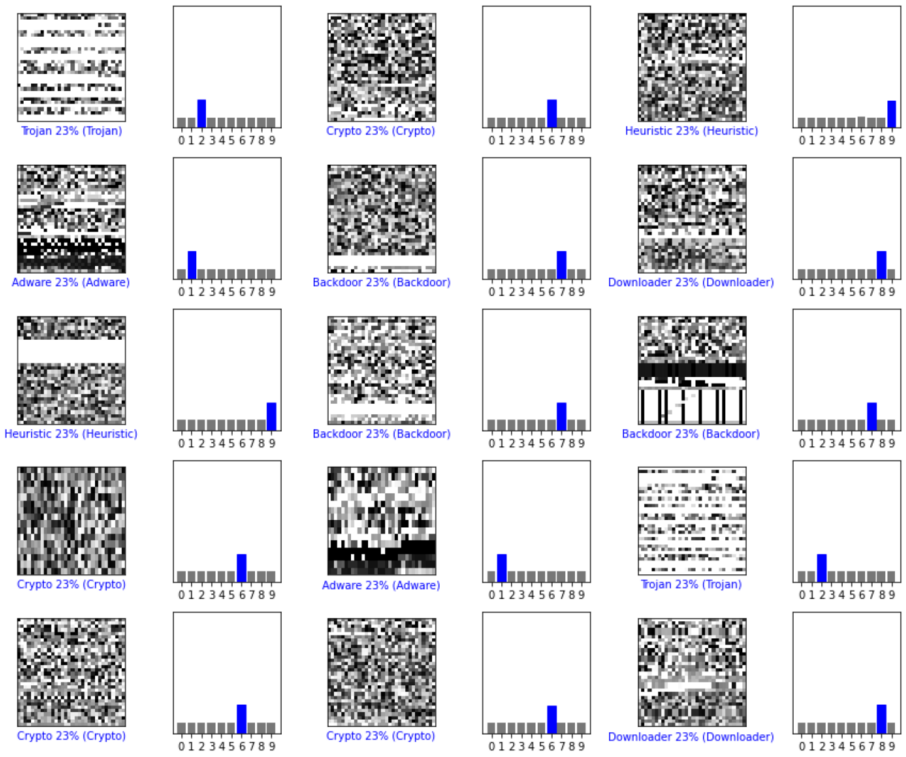


Figure 5.34: Prediction of test images of Virus MNIST

MaleVis Dataset Results

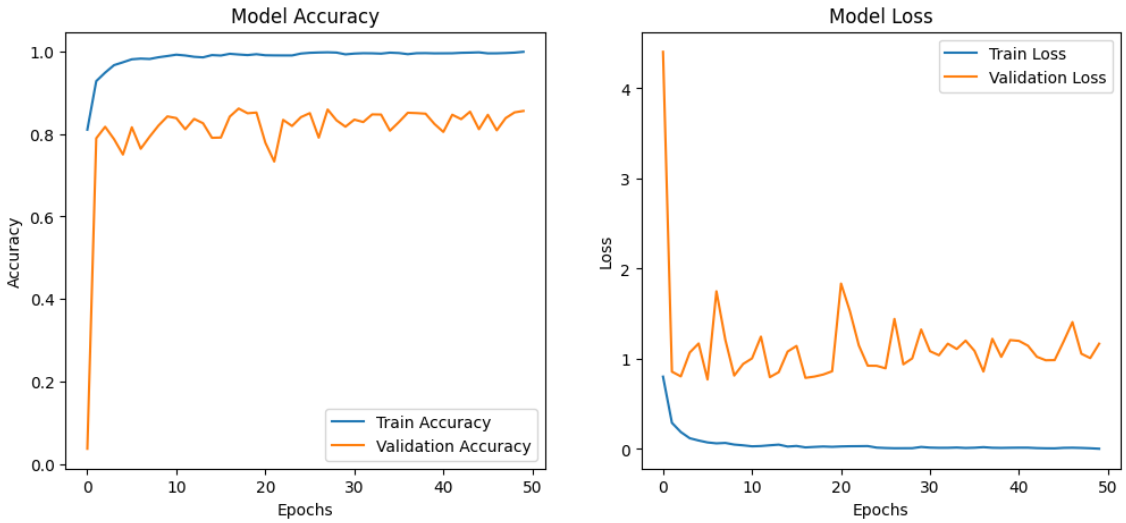


Figure 5.35: MaleVis Accuracy and Loss

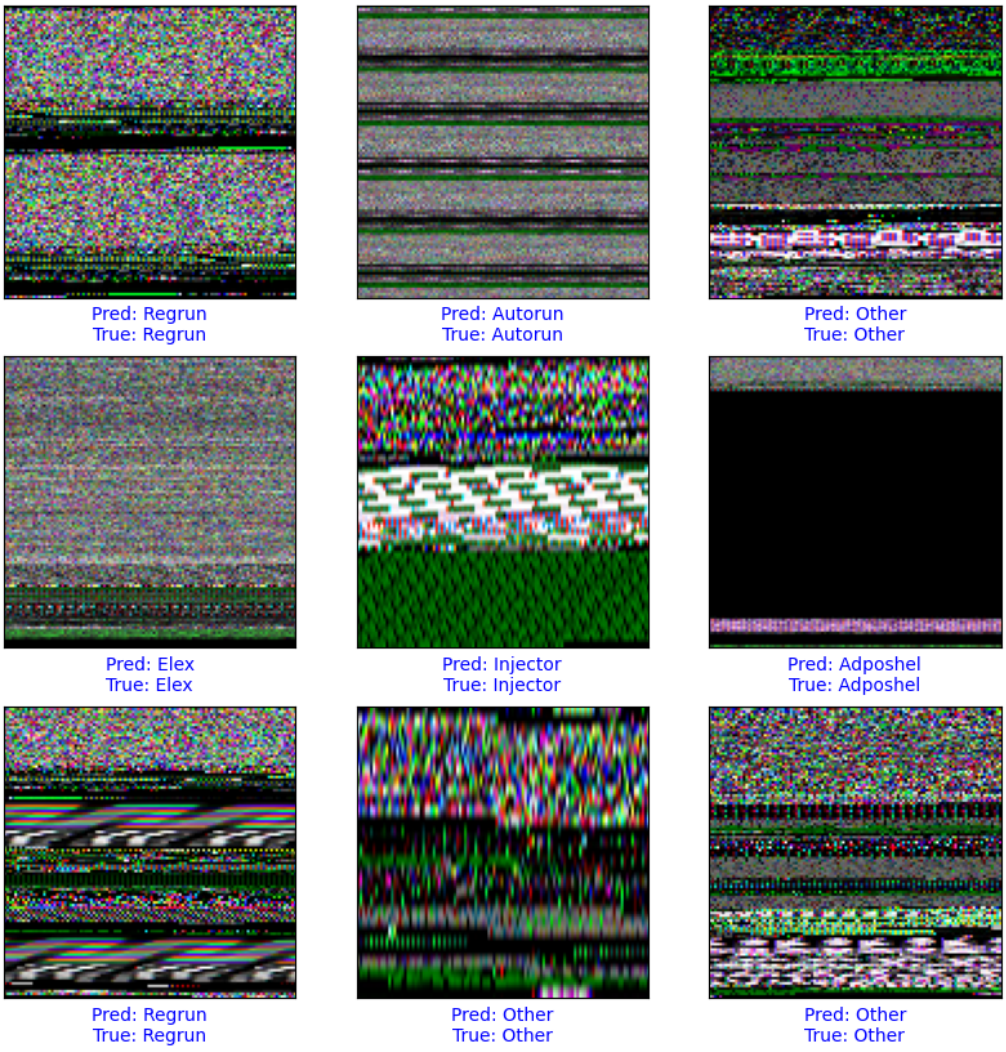


Figure 5.36: MaleVis Predicted Images

Malimg Dataset Results

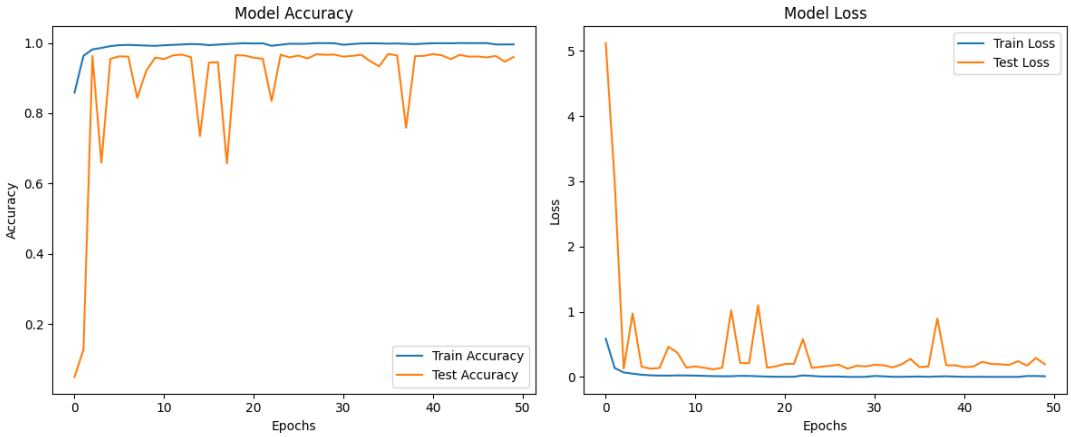


Figure 5.37: Malimg Accuracy and Loss

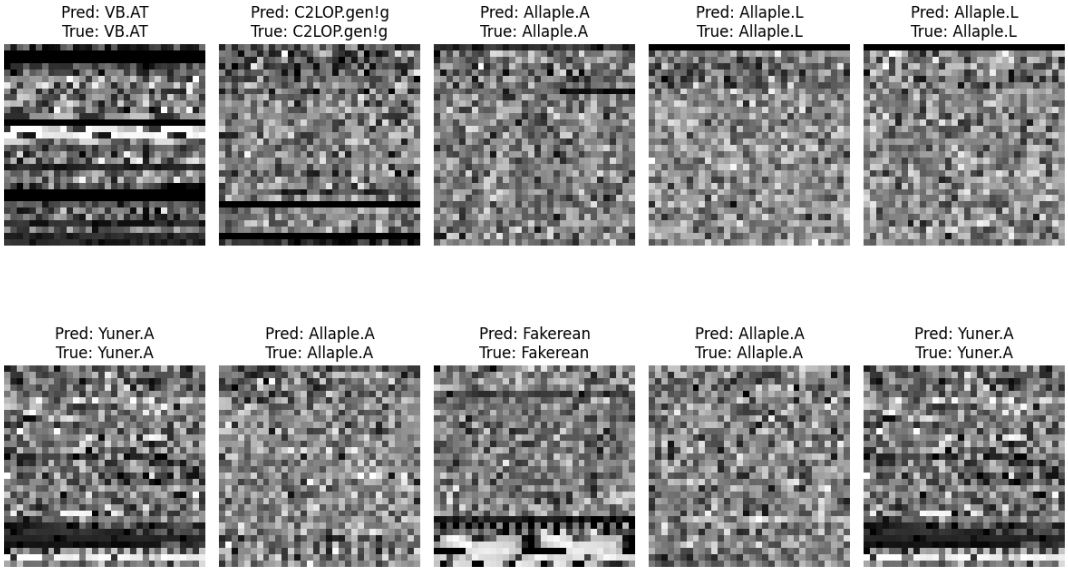


Figure 5.38: Malimg Predicted Images

Table 5.12: CNN with Data Augmentation

Results		
Dataset	Training Accuracy	Validation Accuracy
Malimg	99.91%	96.73%
Virus MNIST	97.51%	92.13%
MaleVis	99.96%	85.60%

Table 5.12 illustrates that:

- All the experiments were trained for 50 epochs.
- CNN with Data augmentation is effective for Malimg and Virus MNIST datasets, improving validation performance and mitigating overfitting.
- For RGB dataset like MaleVis, augmentation is insufficient; overfitting persists after extensive training because of dataset complexity.

Proposition for CNN with Data Augmentation

1. **Data Augmentation**
   - **Statement:** In this experiment, data augmentation techniques like horizontal flipping and rescaling were implemented on the dataset.
   - **Rationale:** Augmentation method increases dataset diversity and reduces overfitting.
2. **Model Compilation**
   - **Statement:** The model was compiled using the Adam optimizer, sparse categorical cross-entropy loss, and accuracy as a metric.
   - **Rationale:** Adam is well-suited for this experiment and sparse categorical cross-entropy handles integer class labels efficiently.
3. **Prediction Visualization**
   - **Statement:** Visualize the predictions against actual labels with confidence scores for both correct/incorrect classification.
   - **Rationale:** This method gives some idea about model confidence and gives a visual on where it misclassifies.

Critical Analysis for CNN with Data Augmentation

- **Data Augmentation for Malware Images:**

  This experiment is applying data augmentation techniques, such as horizontal flipping to malware images. This helps improve the model's ability to generalize and recognize various orientations of malware instances.

- **Class Imbalance Handling:**

  The datasets have several classes representing different types of malware. Handling class imbalance in the dataset is crucial for effective model training and generalization.

- **Use of ImageDataGenerator:**

  The utilization of the ImageDataGenerator from Keras for real-time data augmentation during training is a common practice, and its inclusion in the experiment contributes to the robustness of the model.

5.2.10 Image Generation Using Autoencoder

Autoencoder is an artificial neural network which consists of encoder and decoder network for data compression and representation with an unsupervised machine learning method and uses unlabeled data for training as it can reconstruct the input data by compression and decompression to generate the original input by using representation learning [197].

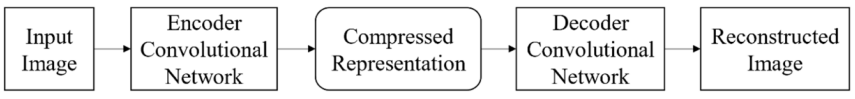


Figure 5.39: Function of Autoencoder

The fundamental function of the Autoencoder is shown in Figure 5.39. The useful application of Autoencoders are: Image denoising, Dimensionality reduction, Object detection, Semantic segmentation, Recommendation system and to generate new data [197]. The neural network of the autoencoder is

shown in Figure 5.40[6]. The images were normalized before putting into the model to minimize the errors.

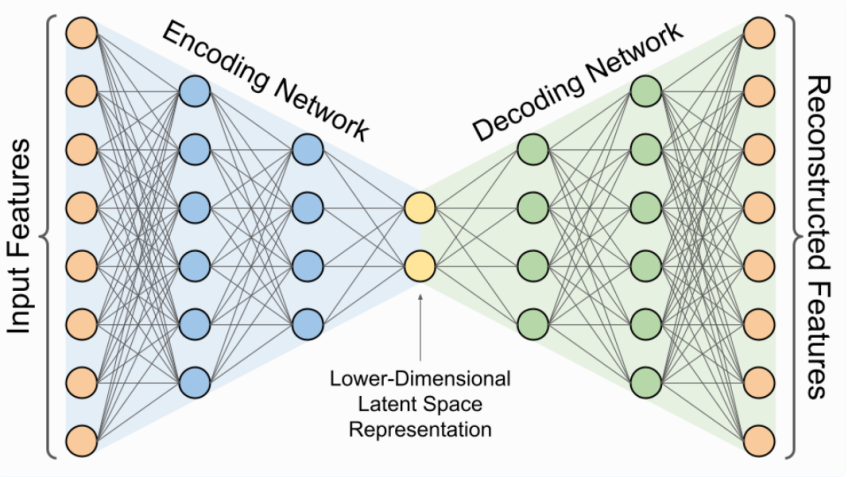


Figure 5.40: Neural Network of Autoencoder

[6] https://pyimagesearch.com/2024/11/18/build-a-network-intrusion-detection-system-with-variational-autoencoders/

Algorithm 7 Autoencoder Pipeline

Require: Dataset D, epochs E, batch size B, latent dimension k
Ensure: Trained autoencoder, reconstruction accuracy
Load dataset $D = \{x_i, y_i\}_{i=1}^{N}$
Convert images to grayscale, resize (H, W)
Flatten images: $x_i \in \mathbb{R}^d$, $d = H \times W$
Normalize pixels: $x_i \leftarrow x_i/255$
Split dataset into training and testing sets
Build autoencoder:
Encoder: $z = f_\theta(x) \in \mathbb{R}^k, k \ll d$
Decoder: $\hat{x} = g_\phi(z) \in \mathbb{R}^d$
Loss: $L = \frac{1}{N}\sum_{i=1}^{N} \|x^{(i)} - \hat{x}^{(i)}\|^2$
for epoch = 1 ... E **do**
Train on batches of size B, update θ, ϕ using optimizer (Adam/RMSProp)
Validate on test set
end for
Encode test samples: $z = f_\theta(x)$
Reconstruct: $\hat{x} = g_\phi(z)$
Visualize original vs reconstructed images
Compute reconstruction error: $MSE(x, \hat{x}) = \frac{1}{d}\sum_{j=1}^{d}(x_j - \hat{x}_j)^2$
Select threshold τ
Mark sample as well-reconstructed if $MSE(x, \hat{x}) < \tau$
Reconstruction accuracy: Accuracy $= \frac{\#\{i: MSE(x^{(i)}, \hat{x}^{(i)}) < \tau\}}{N} \times 100$
return Autoencoder (f_θ, g_ϕ), accuracy

Table 5.13: Terms and Notation for Autoencoder Pipeline

Symbol / Term	Description
D	Dataset of images
N	Total number of samples in the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample (if available)
H, W	Height and width of input images
d	Flattened input dimension, $d = H \times W$
B	Mini-batch size
E	Number of training epochs
k	Latent dimension of encoder representation ($k \ll d$)
f_θ	Encoder function with parameters θ
g_ϕ	Decoder function with parameters ϕ
z	Latent encoded representation of input x
$\hat{x}$	Reconstructed output from decoder
L	Reconstruction loss (Mean Squared Error)
Optimizer	Algorithm for updating θ, ϕ (Adam or RMSProp)
$MSE(x, \hat{x})$	Reconstruction error for each sample
τ	Threshold to decide if a sample is well-reconstructed
Accuracy	Fraction of samples with $MSE(x, \hat{x}) < \tau$

Steps of Autoencoder Experiment

The following steps were carried out the procedure for conducting the Autoencoder experiment according to Algorithm 7 with terms and notations described in Table 5.13:

1. **Data Preparation**
    - At first, loading of the dataset $D = \{x_i\}_{i=1}^{N}$ containing image samples.
    - Preprocessing of each image by resizing into a fixed size (H, W) and normalizing pixel values to the range $[0, 1]$.
    - Flattening of images into dimension of $d = H \times W$ if using fully connected layers.
    - Splitting of the dataset into training and testing subsets.
2. **Model Construction**
    - Encoder network was defined to map the input $x \in \mathbb{R}^d$ to a latent representation $z \in \mathbb{R}^k$, where $k \ll d$.

- After that, the decoder network was defined to reconstruct $\hat{x}$ from the latent vector z.
- Autoencoder's objective is to minimize the reconstruction loss:

$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

3. **Training Phase**
   - Training of the autoencoder for a fixed number of epochs E using mini-batches of size B.
   - Optimize model parameters using a suitable optimizer (Adam or RMSProp).
4. **Reconstruction Phase**
   - Encoding of the unseen test samples into latent representations.
   - Decoding the latent vectors to reconstruct the images.
   - Visualization of original vs reconstructed image samples for evaluation.
5. **Reconstruction Error Analysis**
   - Computation of the reconstruction error for each test sample using Mean Squared Error (MSE).
   - Define a threshold τ.
6. **Evaluation**
   - Computation of the reconstruction accuracy as:

$$\text{Accuracy} = \frac{\#\{i : \text{MSE}(x^{(i)}, \hat{x}^{(i)}) < \tau\}}{N} \times 100$$

   - Reconstruction accuracy detection.

Proposition for using Autoencoder

1. **Autoencoder Model Design**
   - **Proposition:** In this experiment, an autoencoder was constructed with [198]:
     - An encoder which is a module to compress the input data into low-dimensional latent space using Dense layers.

- A decoder to take that latent representation to reconstruct it the input from the encoded representation.

- **Rationale:** Autoencoders are helpful in dimensionality reduction and unsupervised learning procedures to learn from meaningful features.

2. **Model Compilation**
   - **Proposition:** Optimizer-'RMSprop', Loss function - 'binary cross-entropy' were used for compiling the autoencoder.
   - **Rationale:** RMSprop dynamically changes the learning rate while training and binary cross-entropy [199] is perfect for pixel-wise reconstruction.

3. **Encoding and Visualization**
   - **Proposition:** Use the trained encoder to compress test images into their lower-dimension versions and visualize them with their original versions.
   - **Rationale:** It reduces the dimensions and shows an encoder that learns to extract meaningful features data.

Virus MNIST Autoencoder Results

For the experiment with Virus MNIST dataset, the training dataset was plotted according to the labels and amount of data in each class Figure 5.41.

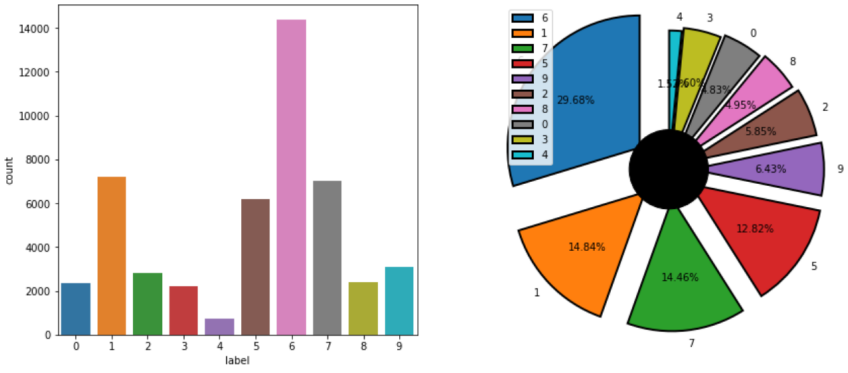


Figure 5.41: Train data with labels

Figure 5.42 is showing the whole architecture of the autoencoder with the input image and reconstructed image. As the autoencoder copies the values from input layer to the output layer, the output image will be the same size of the input image.

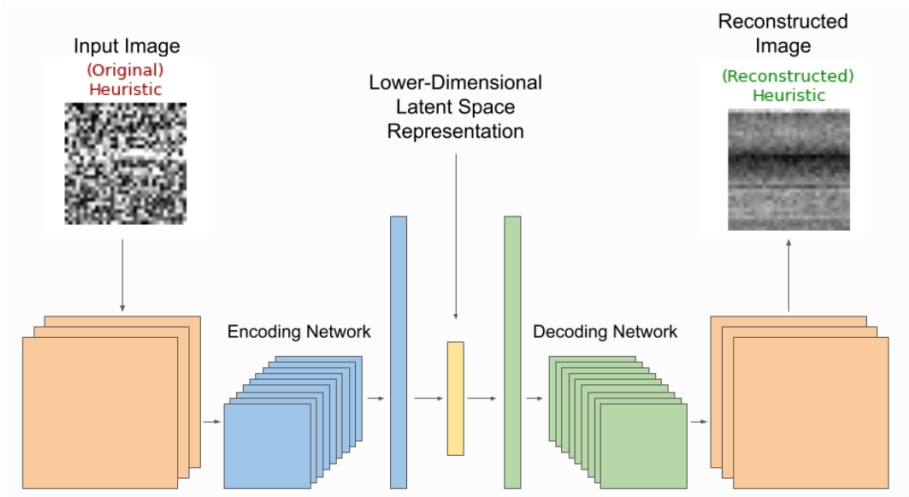


Figure 5.42: Network architecture of autoencoder

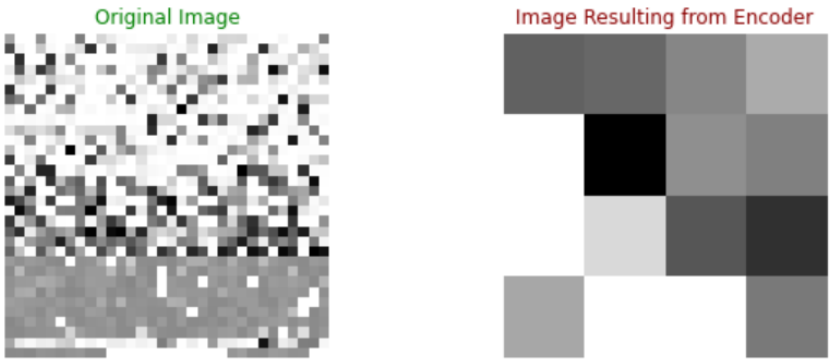


Figure 5.43: Encoder Compresses the Original Image into a Lower-dimensional Representation

After training the autoencoder model the encoded image will look like Figure 5.43. The reconstructed images from the autoencoder are shown in Figure 5.44.

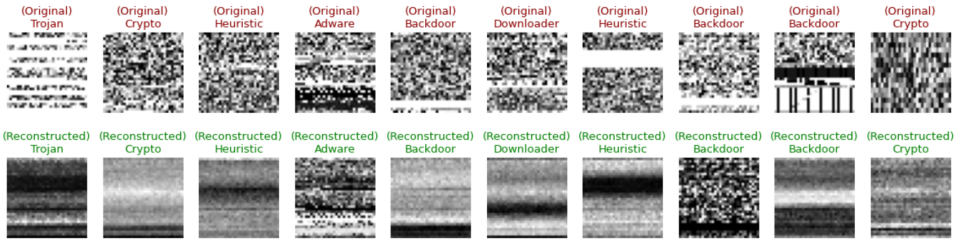


Figure 5.44: Reconstructed Images

Figure 5.45 is showing the reconstruction error distribution with Virus MNIST dataset.

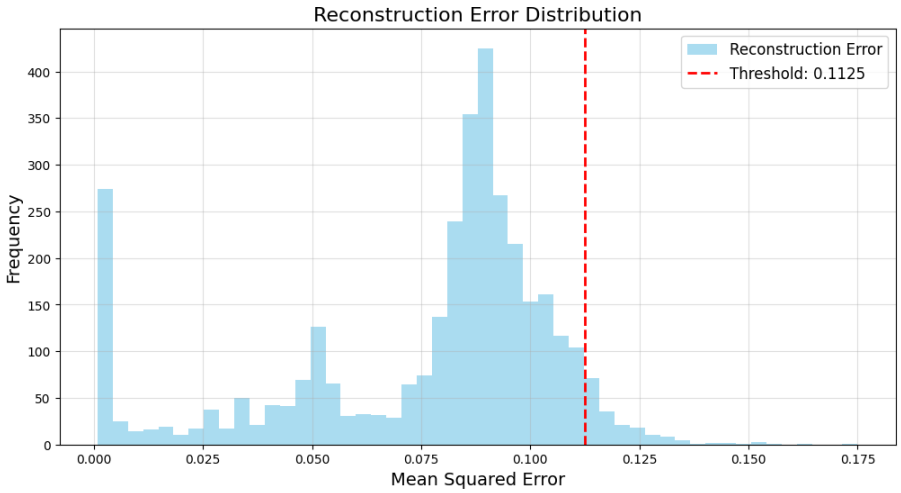


Figure 5.45: Virus MNIST Autoencoder Reconstruction Error Distribution

MaleVis Autoencoder Results

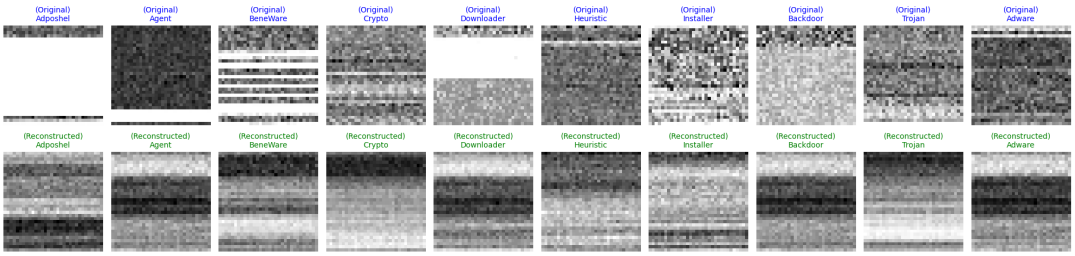


Figure 5.46: MaleVis Autoencoder Recnstructed Images

Figure 5.47 is showing the result of MaleVis dataset.

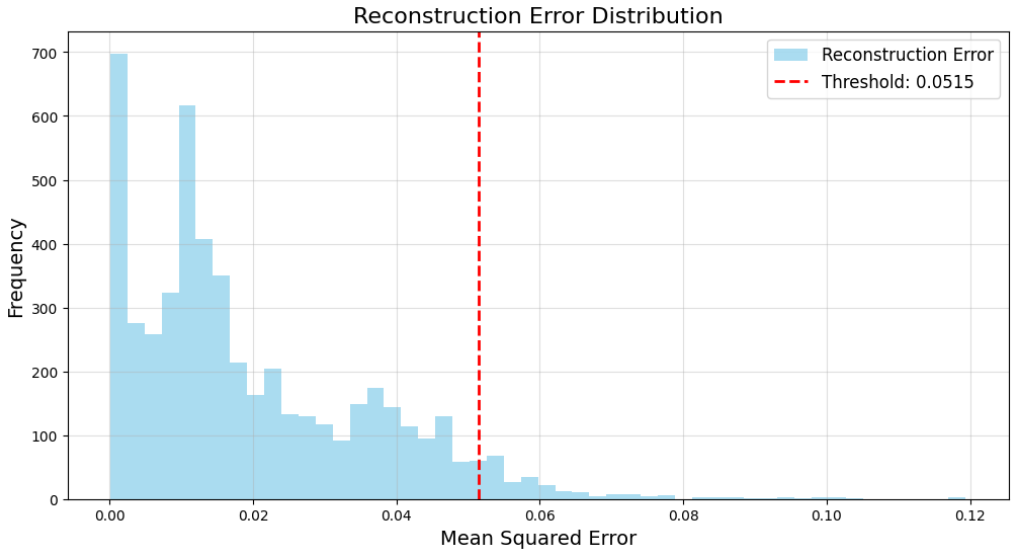


Figure 5.47: MaleVis Autoencoder Reconstruction Error Distribution

Malimg Autoencoder Results

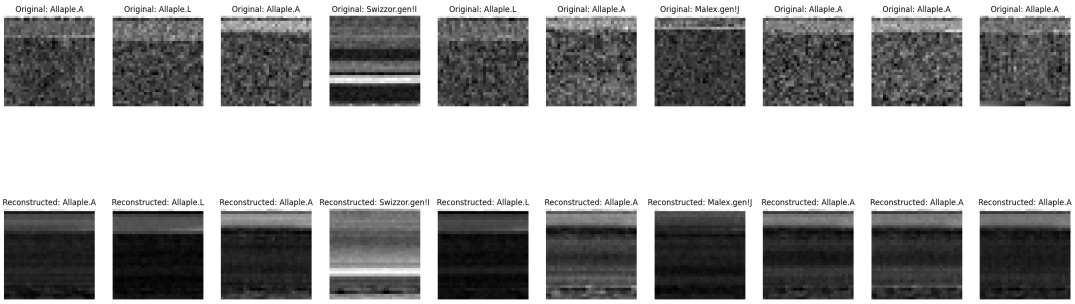


Figure 5.48: Malimg Reconstructed Images

Figure 5.49 is showing the result for Malimg dataset.

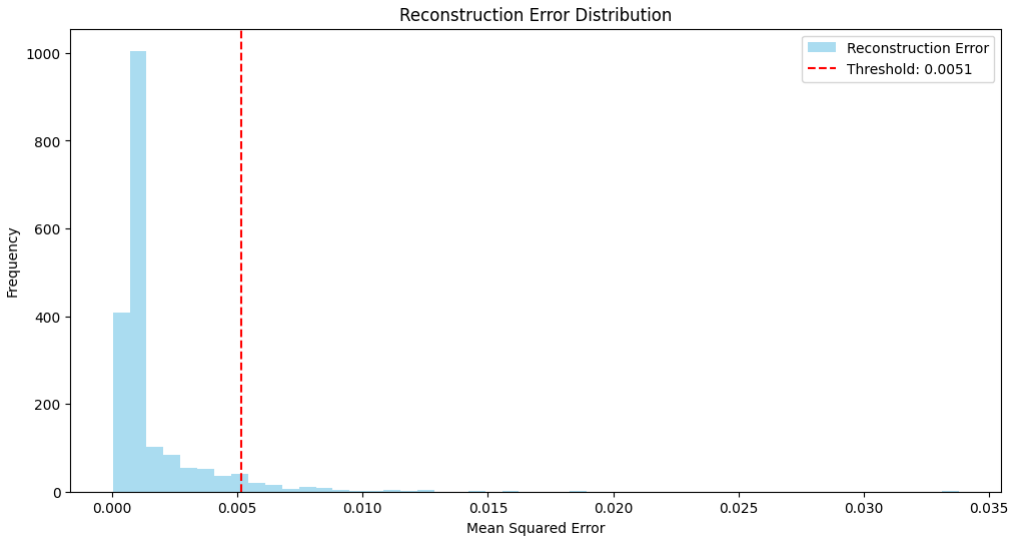


Figure 5.49: Malimg Autoencoder Reconstruction Error Distribution

Table 5.14: Autoencoder Experimental Results

Results	
Dataset	Reconstruction Accuracy
Malimg	94.97%
Virus MNIST	95.00%
MaleVis	94.99%

Table 5.14 is showing the results with all three datasets. In this experiment, all datasets were trained for 250 epochs:

- Reconstruction accuracies are consistent (95%) across all three datasets, which concludes that autoencoder is effective for both grayscale and RGB malware images.
- Therefore the autoencoders can serve as reliable feature extractors without overfitting.

Critical Analysis of using Autoencoder

- **Unsupervised Representation Learning:**

  Autoencoders are a kind of unsupervised learning algorithm [200]. In this experiment, they are also implemented to learn compact representation of malware images without explicit labels. This could be helpful for when labeled data is limited or expensive to get.

- **Visualization of Representations:**

  The experiment was conducted as an intuition for how an autoencoder embeds malware images into lower-dimensional space. The visualizations of the encoded images might be informative and useful for a domain expert in cybersecurity.

- **Potential for Feature Extraction:**

  Feature extractor part is used as the encoder section of the autoencoder, helping contribute toward feature engineering for the downstream tasks [201].

- **Interpretability:**

Labeling of the images and the visualizations provide some interpretability for this experiment. This will help the user to understand what features of the images are being captured by the autoencoder, whether related to malware or in general.

Malware-trained Autoencoder for Novelty Detection:

To find novelty, this Autoencoder was implemented differently on three datasets. At first the model was trained on the Malimg dataset which contains malicious samples only. After that, it was tested to identify benign files from the Virus MNIST and MaleVis datasets.

1. **Dataset Preparation:**
   - The Malimg dataset was splitted into 80% training data, 10% validation and 10% test data. Therefore, after splitting the size of data is Train: 7471 images, Validation: 934 images and Test: 934 images.
   - The image count of Virus MNIST benign older ("0") is 2341 and for MaleVis, the legitimate ("Other") folder contains 1482 images.
   - These two were combined as a benign set containing 3823 images.

2. **Experiment Setup:**
   - All the images were resized into 64×64 resolution and normalized to [0,1].
   - Random shuffling was applied before training.

3. **Model Training:**
   - The autoencoder model was trained for 50 epochs with Mean Squared Error (MSE) as loss function and optimizer: Adam.
   - After training the metrics Precision, Recall and F1-score were calculated.

4. **Visualization:**

   Figure 5.50 is showing the confusion matrix of the result, with Precision: 0.872, Recall: 0.751 and F1-score: 0.807. Out of 3823 benign samples 2871 were correctly classified as benign by the Autoencoder.

```
Confusion Matrix:
 [[ 512  422]
 [ 952 2871]]
Precision: 0.872, Recall: 0.751, F1: 0.807
```

Figure 5.50: Malware-trained Autoencoder Novelty Result

The interpretation is:

- True Negatives (Malware correctly detected): 512
- False Positives (Malware misclassified as benign): 422
- False Negatives (Benign misclassified as malware): 952
- True Positives (Benign correctly detected): 2871

High precision 87.2% and recall 75.1% with F1 score 80.7% interprets that the autoencoder effectively identifies the malicious and benign samples and patterns of the datasets.

5.2.11 Classification with Ensembling CNNs

The malware image datasets were used to implement with other neural networks to test their results. This experiment was done by ensembling 10 CNN architectures. Ensembling models are the process of combining multiple models to get a single prediction. Ensembling Convolutional Neural Networks are implemented to produce diversity in the models.

For the visualization of the dataset, some random images of Virus MNIST dataset were chosen with assigned labels. The images are shown in Figure 5.51.

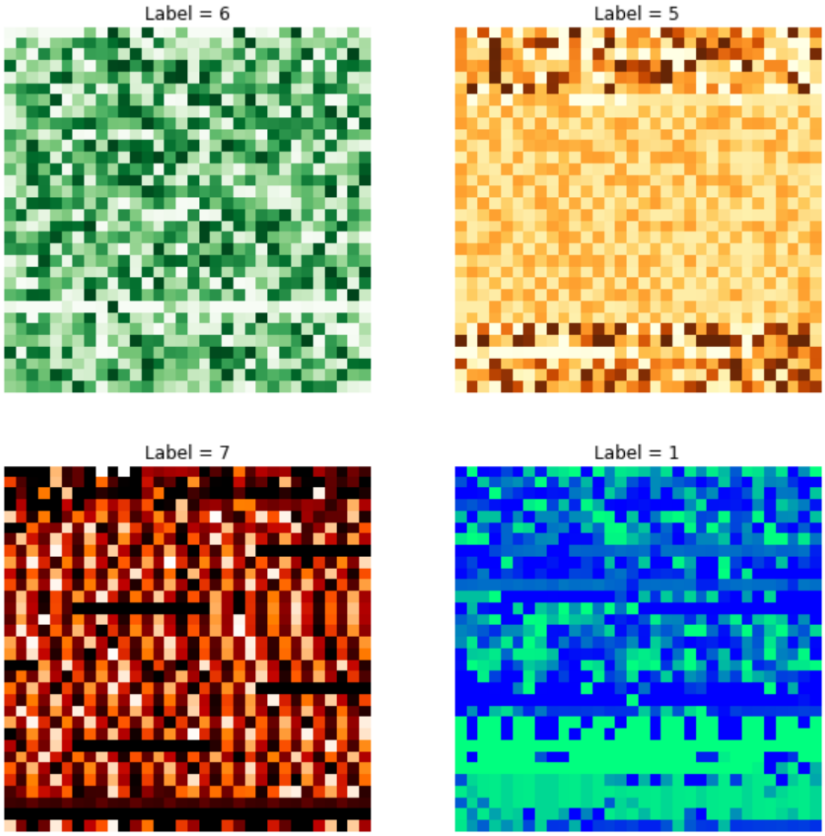


Figure 5.51: Images with labels

Figure 5.52 is showing the multiple images with their labels and Figure 5.53 has the pixels of an image of label 7.

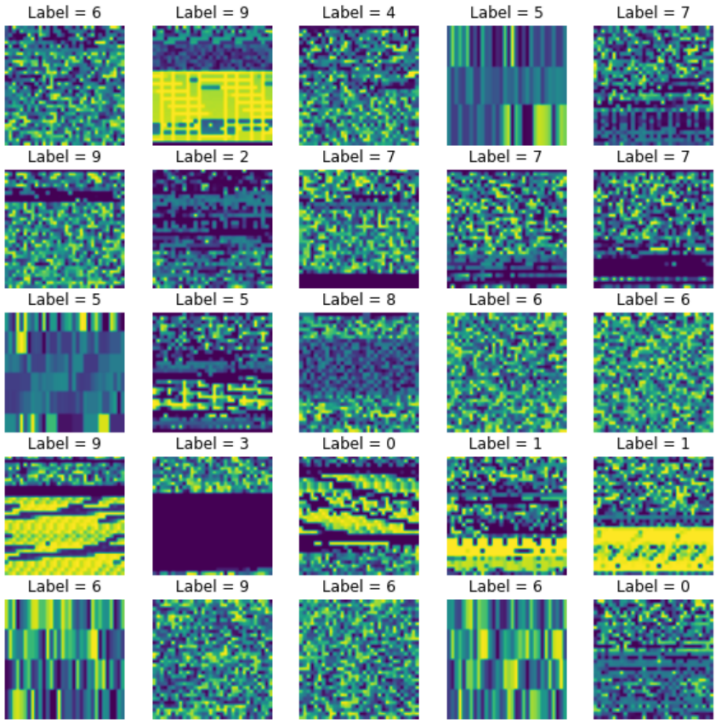


Figure 5.52: Multiple Images with labels

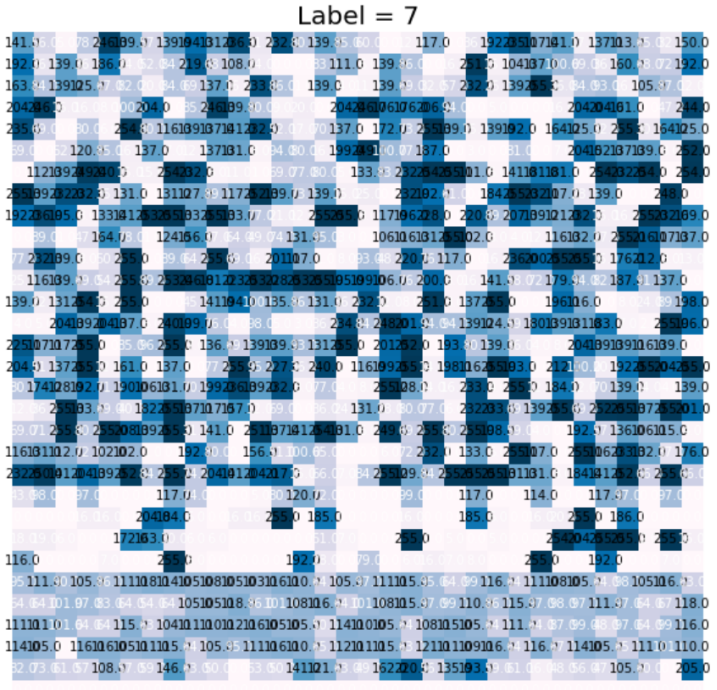


Figure 5.53: Image with pixels

Algorithm 8 CNN Ensemble Pipeline

Require: Dataset D, batch size B, epochs E, learning rate η, number of CNNs n

Ensure: Trained ensemble $\{M_1, \ldots, M_n\}$, ensemble predictions

```
Normalize images ∈ [0,1], resize (H, W, C)
Encode labels if needed
Split D → (D_train, D_val, D_test)
Apply data augmentation on D_train
Define CNN architecture function:
    Conv layers + ReLU + BatchNorm + MaxPool
    Dropout layers
    Flatten
    Dense + ReLU
    Output layer + Softmax
    Compile (optimizer=Adam(η), loss=CategoricalCrossentropy, metric=Accuracy)
for i = 1 ... n do
    M_i ← CNN architecture function
    for epoch = 1 ... E do
        Train M_i on batches of size B
        Validate on D_val
        Adjust learning rate using scheduler
    end for
    Save M_i to model list
end for
Initialize ensemble predictions P = 0
for each M_i do
    p_i ← predict probability on D_test
    P ← P + p_i
end for
Final predictions: y_pred = arg max(P)
Evaluate accuracy
Visualize training/validation curves, confusion matrix
return {M_1, ..., M_n}, y_pred
```

Table 5.15: Terms and Notation for CNN Ensemble Pipeline

Symbol / Term	Description
D	Dataset of images and corresponding labels
D_{train}	Training split of the dataset
D_{val}	Validation split of the dataset
D_{test}	Test split of the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample
H, W, C	Height, width and number of channels of input images
B	Mini-batch size for training
E	Number of training epochs
η	Learning rate for optimizer
n	Number of CNN models in the ensemble
M_i	Individual CNN model i in the ensemble
p_i	Predicted probability vector from model M_i on test set
P	Aggregated ensemble probability vector, $P = \sum_i p_i$
y_{pred}	Final predicted labels from ensemble, $y_{pred} = \arg\max(P)$
Optimizer	Algorithm used for training each CNN (e.g., Adam)
Loss	Categorical Crossentropy
Accuracy	Fraction of correctly predicted labels in D_{test}
Confusion matrix	Table summarizing classification performance across all classes

Steps for 10 CNN Training and Evaluation Pipeline

According to Algorithm 8 following steps were implemented with description of terms and notation in Table 5.15:

1. **Data Preparation**
   - At first, normalization of image pixel values to $[0, 1]$.
   - Resizing of the images into fixed dimensions (H, W).
   - Splitting of the dataset into training, validation and test sets.
   - Data augmentation methods (rotation, shifts etc.) were applied.
2. **Traditional Machine Learning Baselines**

   Some machine learning classifiers were implemented on the datasets initially. ML models like K-Nearest Neighbors (K-NN), Decision Tree and Random Forest classifier were implemented for training and evaluation.

3. **CNN Architecture Design**

   The CNN model was designed with:
   - Multiple convolutional layers with ReLU activation.
   - Batch normalization and max-pooling after convolutional blocks.
   - Dropout layers for regularization.
   - Flattening of the feature maps.
   - Fully connected dense layers with ReLU activation.
   - Adding output layer with softmax activation.

4. **Training 10 CNN Models**
   - At first, to initialize the 10 independent CNN models with same architecture, training of each CNN on mini-batches was performed. After that validation of the model.
   - Use of the learning rate scheduler (e.g., ReduceLROnPlateau) if needed.

Architecture of Ensembling CNNs

1. **CNN1:**
   - Layers: 1 Convolutional layer.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)
     - Output layer with a softmax activation for classification.
   - Purpose: A basic CNN with only one convolutional layer.

2. **CNN2:**
   - Layers: 2 Convolutional layers.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)
     - Output layer (softmax activation).
   - Purpose: A model with two convolutional layers, adding more abstraction of features.

3. **CNN3:**
    - Layers: 3 Convolutional layers.
    - Architecture:
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - MaxPooling2D(2x2)
        - Conv2D(64 filters, 3x3 kernel, ReLU activation)
        - Flatten layer
        - Dense layer (128 units, ReLU activation)
        - Output layer (softmax activation).
    - Purpose: Adds more feature extraction by introducing a third convolutional layer.

4. **CNN4:**
    - Layers: 4 Convolutional layers.
    - Architecture:
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - MaxPooling2D(2x2)
        - Conv2D(64 filters, 3x3 kernel, ReLU activation)
        - Conv2D(64 filters, 3x3 kernel, ReLU activation)
        - MaxPooling2D(2x2)
        - Flatten layer
        - Dense layer (128 units, ReLU activation)
        - Output layer (softmax activation).
    - Purpose: Introduces deeper layers for more complex feature extraction.

5. **CNN5:**
    - Layers: 5 Convolutional layers.
    - Architecture:
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - Conv2D(32 filters, 3x3 kernel, ReLU activation)
        - MaxPooling2D(2x2)
        - Conv2D(64 filters, 3x3 kernel, ReLU activation)
        - Conv2D(64 filters, 3x3 kernel, ReLU activation)
        - MaxPooling2D(2x2)

- Conv2D(128 filters, 3x3 kernel, ReLU activation)
- Flatten layer
- Dense layer (128 units, ReLU activation)
- Output layer (softmax activation).

- Purpose: Increases depth further by adding another convolutional layer.

6. **CNN6:**
   - Layers: 6 Convolutional layers.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)
     - Output layer (softmax activation).
   - Purpose: Adds more convolutional layers for better feature extraction.

7. **CNN7:**
   - Layers: 7 Convolutional layers.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)

- Output layer (softmax activation).

- Purpose: Adds even more convolutional layers to extract features more effectively.

8. **CNN8:**
   - Layers: 8 Convolutional layers.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)
     - Output layer (softmax activation).
   - Purpose: More layers for deeper feature extraction.

9. **CNN9:**
   - Layers: 9 Convolutional layers.
   - Architecture:
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - Conv2D(32 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - Conv2D(64 filters, 3x3 kernel, ReLU activation)
     - MaxPooling2D(2x2)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Conv2D(128 filters, 3x3 kernel, ReLU activation)
     - Flatten layer
     - Dense layer (128 units, ReLU activation)

- Output layer (softmax activation).

- Purpose: A deeper network for even more complex feature extraction.

10. **CNN10:**
    - Layers: 10 Convolutional layers.
    - Architecture:
      - Conv2D(32 filters, 3x3 kernel, ReLU activation)
      - Conv2D(32 filters, 3x3 kernel, ReLU activation)
      - MaxPooling2D(2x2)
      - Conv2D(64 filters, 3x3 kernel, ReLU activation)
      - Conv2D(64 filters, 3x3 kernel, ReLU activation)
      - MaxPooling2D(2x2)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Conv2D(128 filters, 3x3 kernel, ReLU activation)
      - Flatten layer
      - Dense layer (128 units, ReLU activation)
      - Output layer (softmax activation).
    - Purpose: The deepest model in this setup, allowing for the highest level of feature extraction.

Proposition for Ensembling CNNs

1. **Proposition:** This experiment explores the performance of 10 Convolutional Neural Networks (CNNs) for malware classification using image-based datasets:
    - The purpose is to deduce the efficacy and reliability of deep learning approaches to distinguish classes of malware.
    - Each model of CNN is thoroughly trained and tested against data augmentation techniques include rotation, scaling and flipping to increase the diversity of the dataset.

- This experiment also incorporates traditional machine learning models such as K-Nearest Neighbors, Decision Trees and Random Forest for comparison to establish a baseline for deep learning performance.

2. **Rationale:**
   - **Comparative Analysis:** This research examines the strengths, weaknesses and suitability of ten different CNN architectures on malware image datasets. The presence of traditional machine learning models like KNN, Decision Trees and Random Forests provides a comparison baseline that highlights the enhancement in performance provided by CNNs.
   - **Addressing Overfitting:** Overfitting is a common problem in deep learning models. To make sure that the CNNs have a balance between learning and generalization, some techniques such as dropout, early stopping and learning rate were implemented [202].

Virus MNIST Dataset Results

The dataset was tested on the Machine Learning models like K-Nearest Neighbors, Decision Tree Classifier and Random Forest Classifier. For the KNN the best score was 76.71%. The confusion matrix of KNN is shown in Figure 5.56. For the Decision Tree Classifier the score was 76.77%. The confusion matrix of Decision Tree and score are in Figure 5.55. The Random Forest Classifier produced the score 86.95% and the confusion matrix is shown in Figure 5.54.

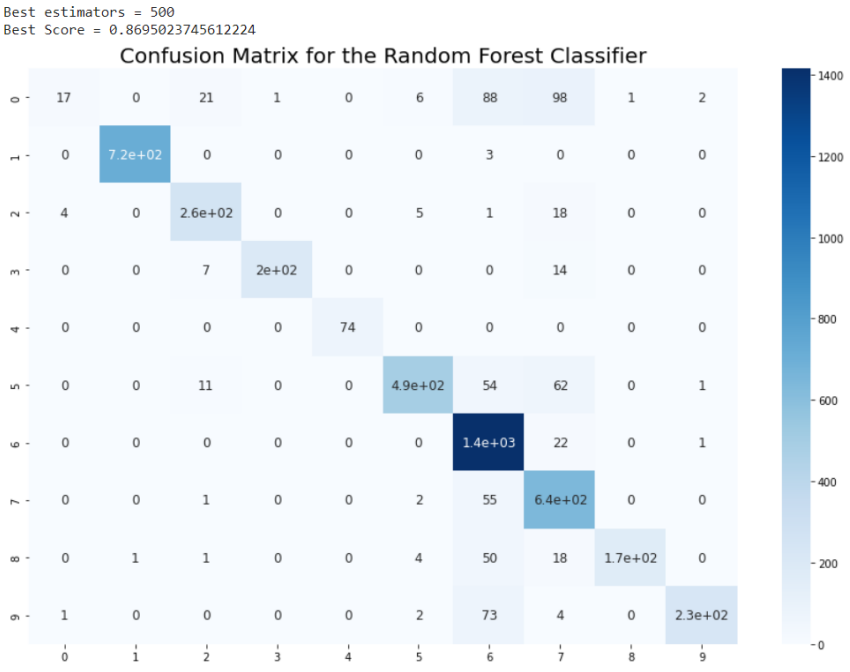


Figure 5.54: Confusion Matrix of Random Forest Classifier

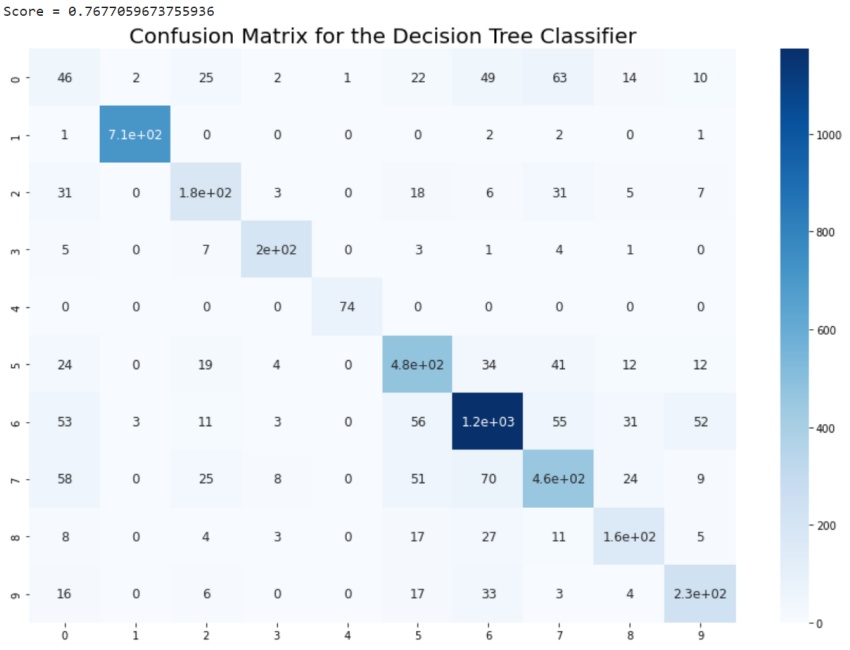


Figure 5.55: Confusion Matrix of Decision Tree Classifier

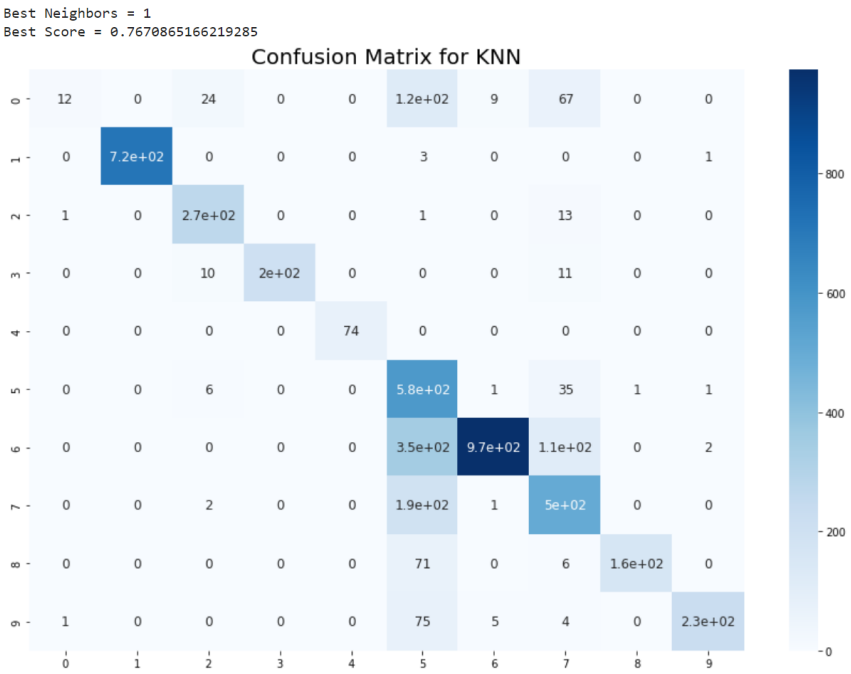


Figure 5.56: KNN Confusion Matrix

The dataset was prepared to feed into the CNN Model. Data Augmentation was done to expand the size of the dataset which will improve the model performance. The model was trained and produced 92.43% training accuracy and 90.56% test accuracy. The confusion matrix of the CNN model is shown in Figure 5.57.

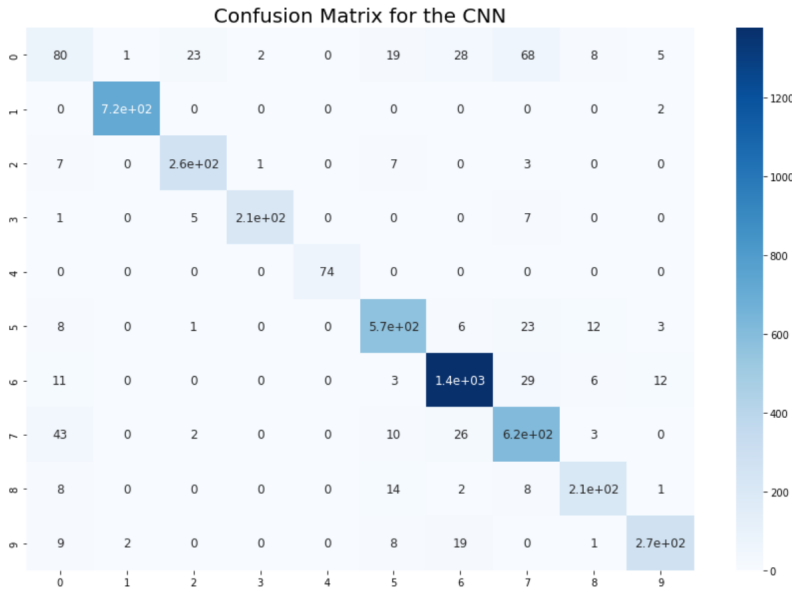


Figure 5.57: Confusion matrix of CNN model

Ensembling CNNs was done by building 10 CNN models for this experiment. Training and Validation accuracy of each model are shown in Table 5.16. The CNN accuracy curve was shown in Figure 5.58. The confusion matrix of 10 CNNs is shown in Figure 5.60.

Table 5.16: Virus MNIST 10 CNN model

Training and Validation Accuracy		
Model	Training Accuracy	Validation Accuracy
CNN1	91.03%	91.31%
CNN2	91.14%	90.91%
CNN3	91.27%	91.27%
CNN4	91.23%	91.37%
CNN5	90.44%	91.06%
CNN6	91.57%	91.20%
CNN7	91.64%	91.39%
CNN8	91.98%	91.53%
CNN9	90.46%	91%
CNN10	91.33%	91%

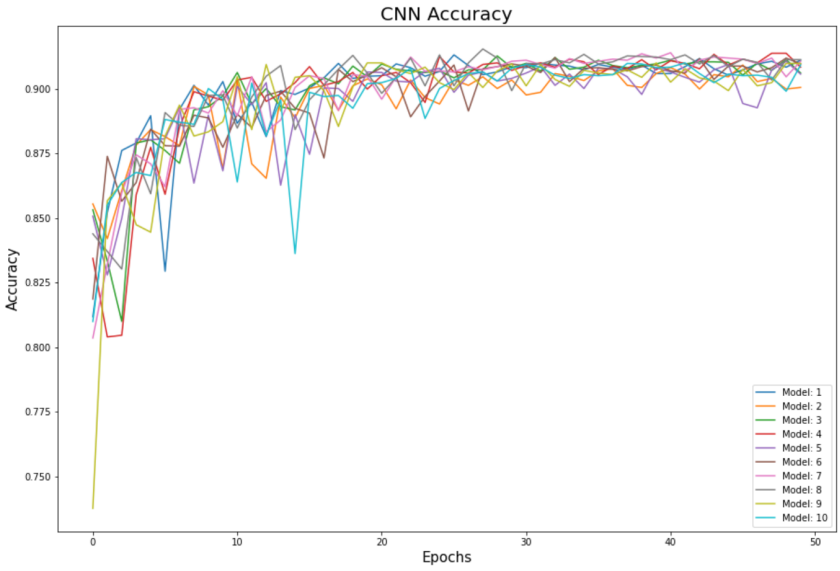


Figure 5.58: Accuracy of Models

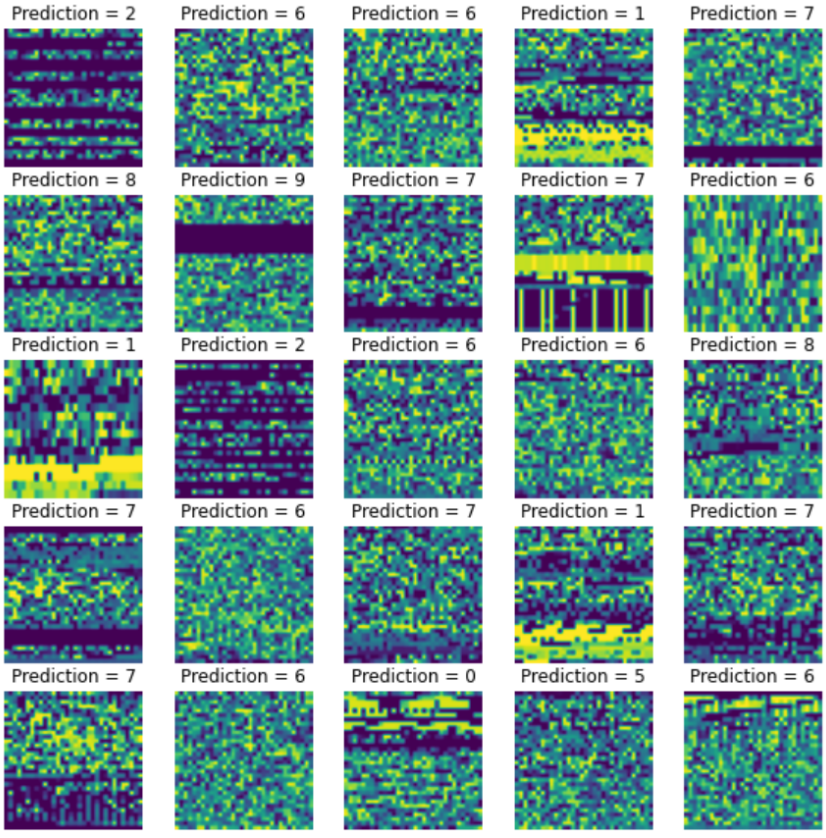


Figure 5.59: Test Dataset Prediction

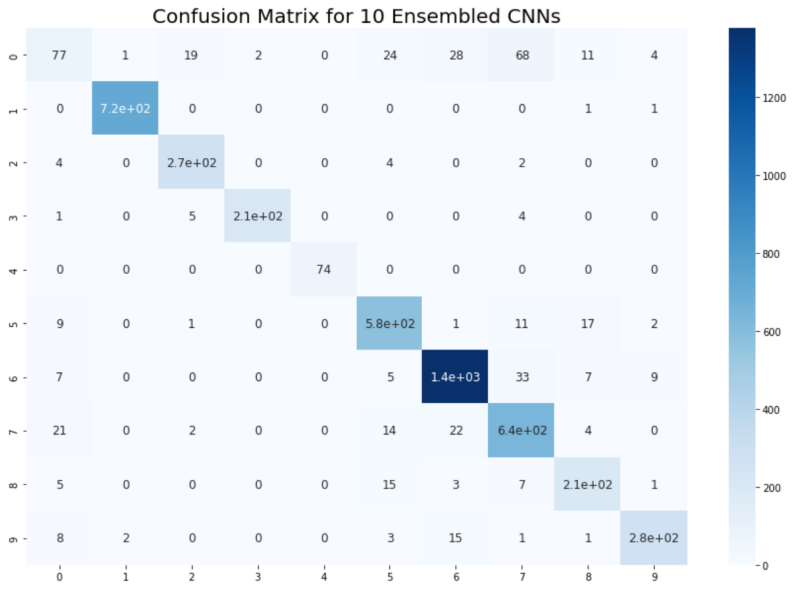


Figure 5.60: 10 CNN Confusion Matrix

MaleVis Dataset Ensembling CNNs

Figure 5.61, Figure 5.62, Figure 5.63 are showing the KNN, Decision Tree and Random Forest results of MaleVis dataset, respectively.

Table 5.17 is showing MaleVis 10 CNN results.

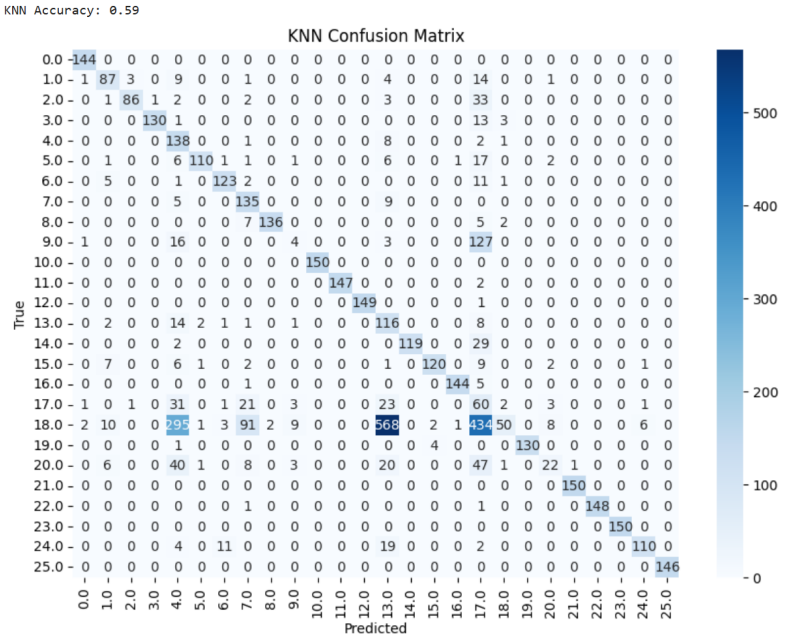


Figure 5.61: MaleVis KNN Results

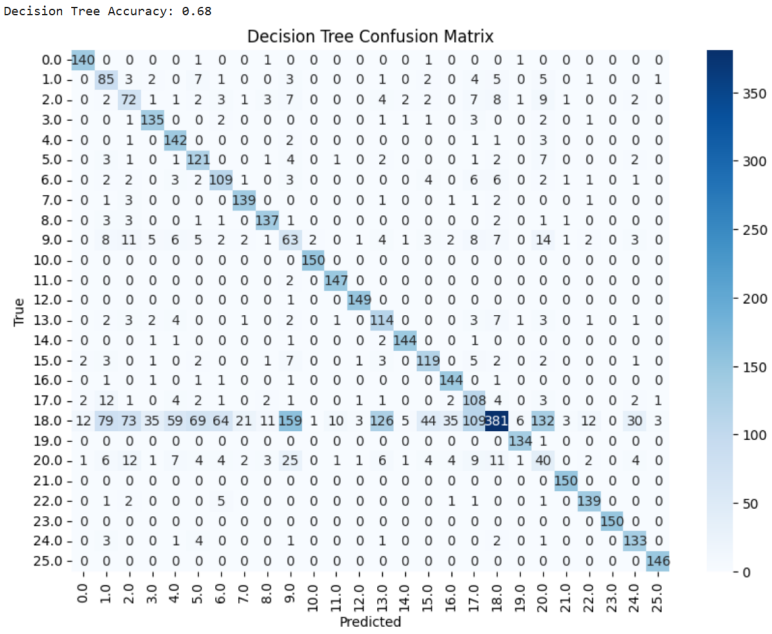


Figure 5.62: MaleVis Decision Tree Results

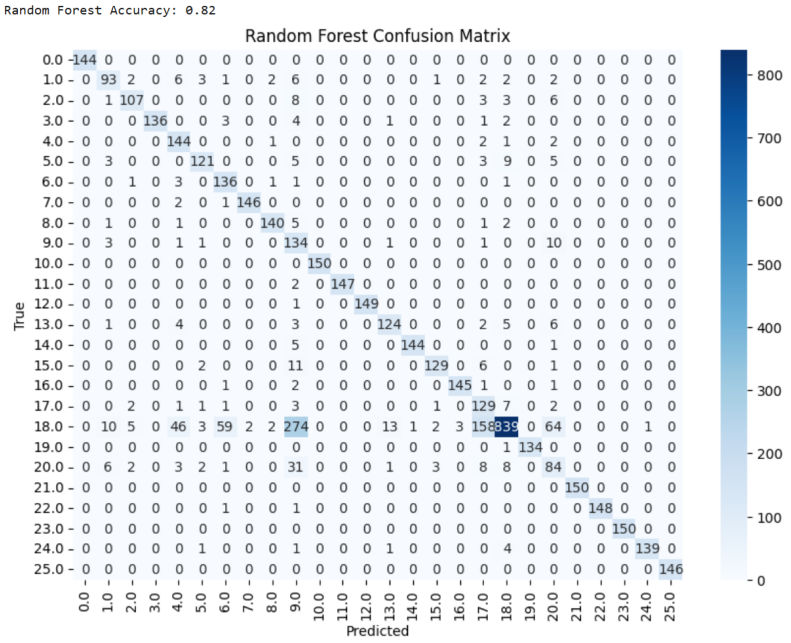


Figure 5.63: MaleVis Random Forest Results

Table 5.17: MaleVis 10 CNN model

Training and Validation Accuracy		
Model	Training Accuracy	Validation Accuracy
CNN1	99.57%	79.89%
CNN2	99.61%	85.27%
CNN3	99.07%	81.43%
CNN4	97.32%	81.12%
CNN5	95.16%	77.76%
CNN6	93.81%	81.56%
CNN7	90.92%	76.65%
CNN8	87.66%	73.53%
CNN9	84.51%	71.95%
CNN10	81.75%	64.71%

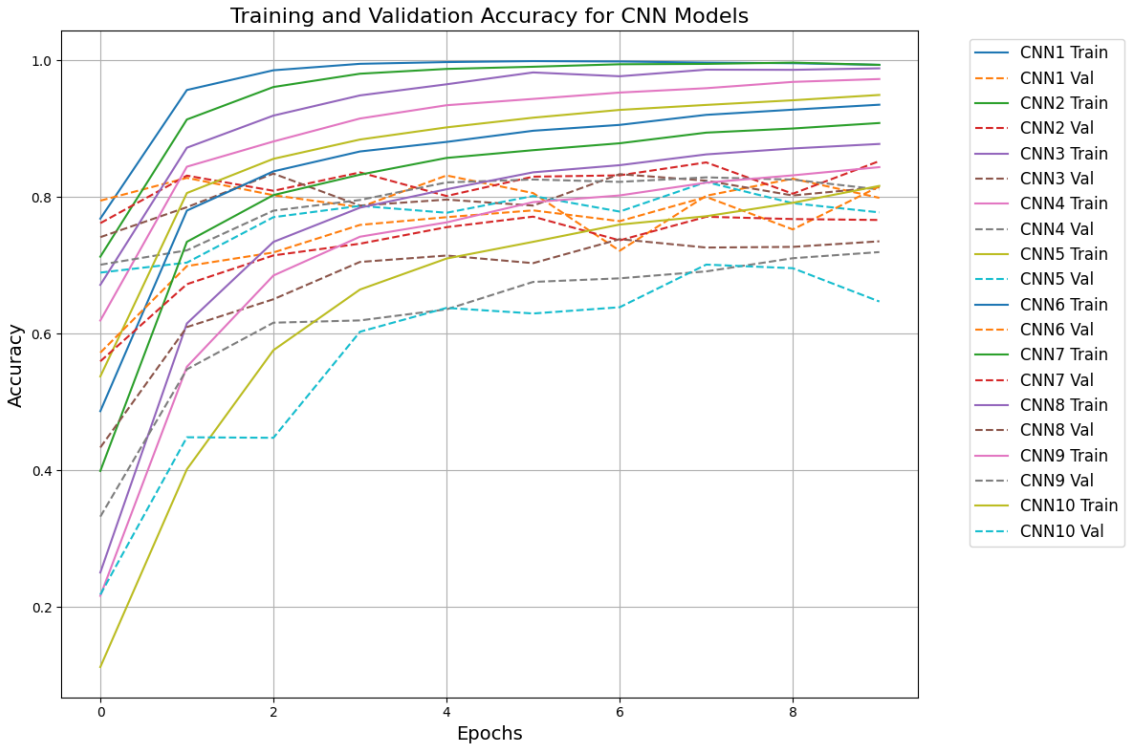


Figure 5.64: MaleVis 10 CNN Accuracies

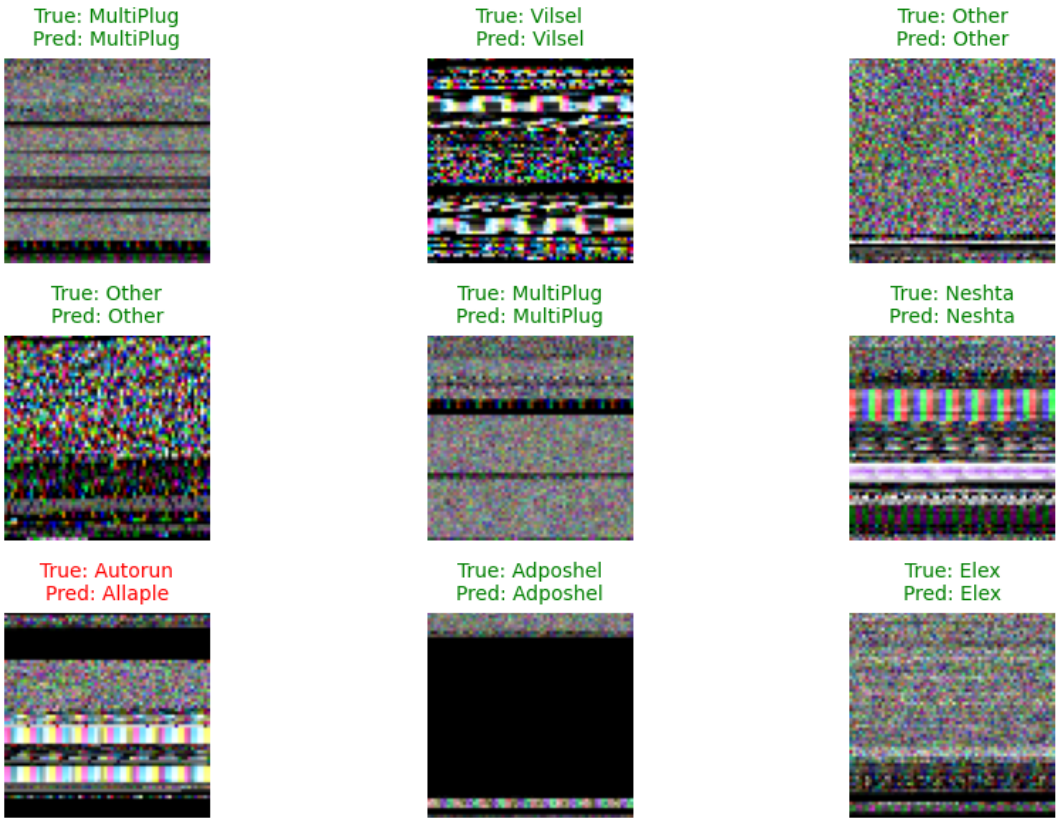


Figure 5.65: MaleVis 10 CNN Prediction Results

Malimg Dataset Ensembling CNNs

Figure 5.66, Figure 5.67 and Figure 5.68 are showing the KNN, Decision Tree and Random Forest results of Malimg dataset, respectively.

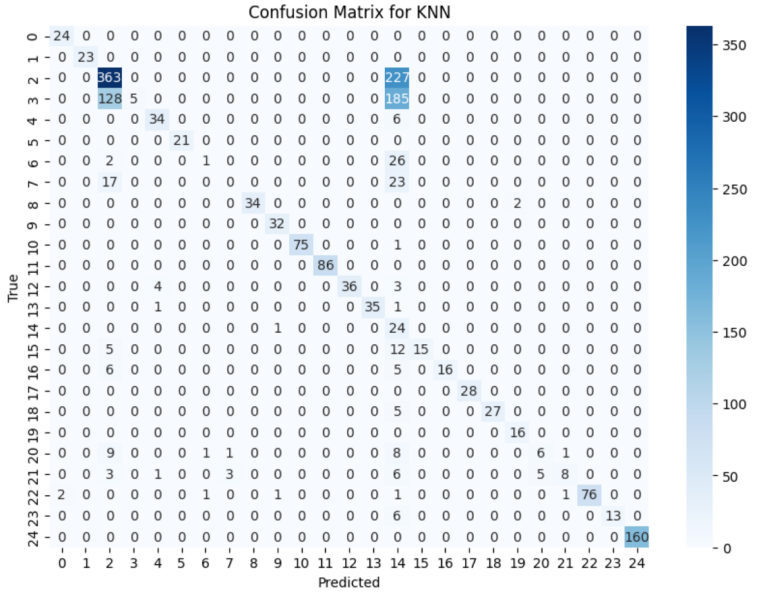


Figure 5.66: Malimg KNN Results

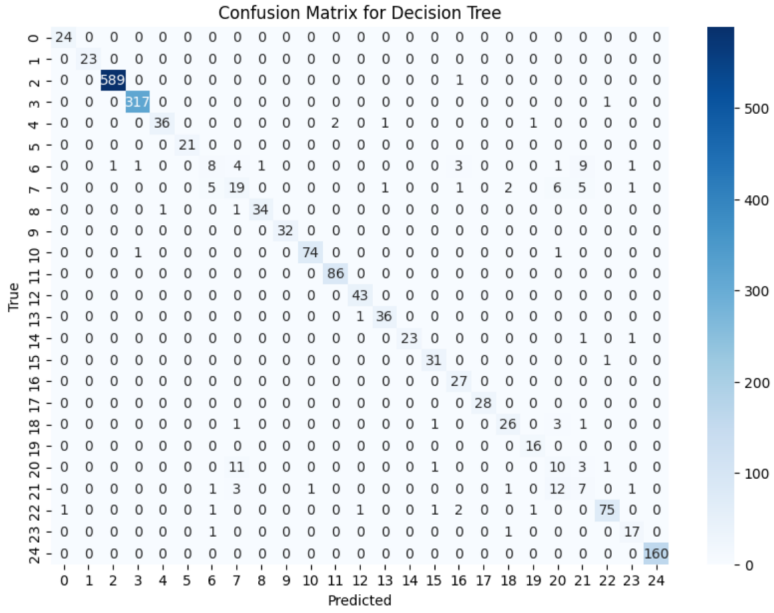


Figure 5.67: Malimg Decision Tree Results

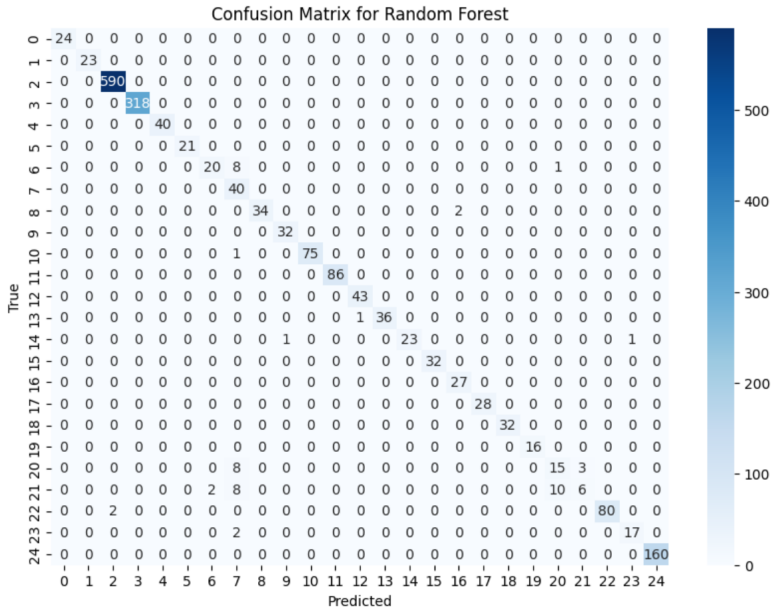


Figure 5.68: Malimg Random Forest Results

Table 5.18 is showing the results of 10 CNN models for Malimg dataset.

Table 5.18: Malimg 10 CNN model

Training and Validation Accuracy		
Model	Training Accuracy	Validation Accuracy
CNN1	97.80%	95.77%
CNN2	97.42%	95.13%
CNN3	96.89%	94%
CNN4	97.54%	95.45%
CNN5	97.82%	93.68%
CNN6	97.17%	95.72%
CNN7	96.90%	95.40%
CNN8	97.02%	95.24%
CNN9	96.66%	95.72%
CNN10	97.04%	95.34%

Table 5.16, Table 5.17 and Table 5.18 are showing that:

- Virus MNIST (grayscale, 10 classes) shows stable training and validation

performances; Malimg (grayscale, 25 classes) shows moderate stability but MaleVis (RGB, 26 classes) shows overfitting and variability.

- Each independent CNN models was trained for 10 epochs. Short training was sufficient for grayscale datasets (Virus MNIST, Malimg) but insufficient for complex RGB dataset (MaleVis).

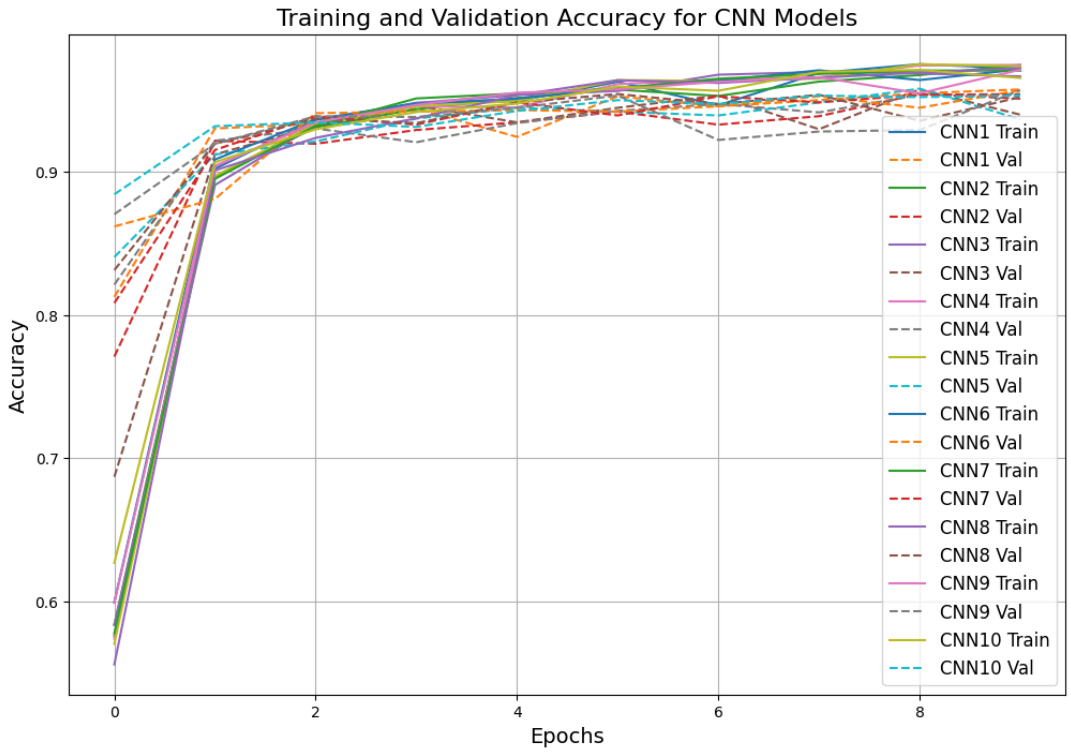


Figure 5.69: Malimg 10 CNN Accuracy Plot

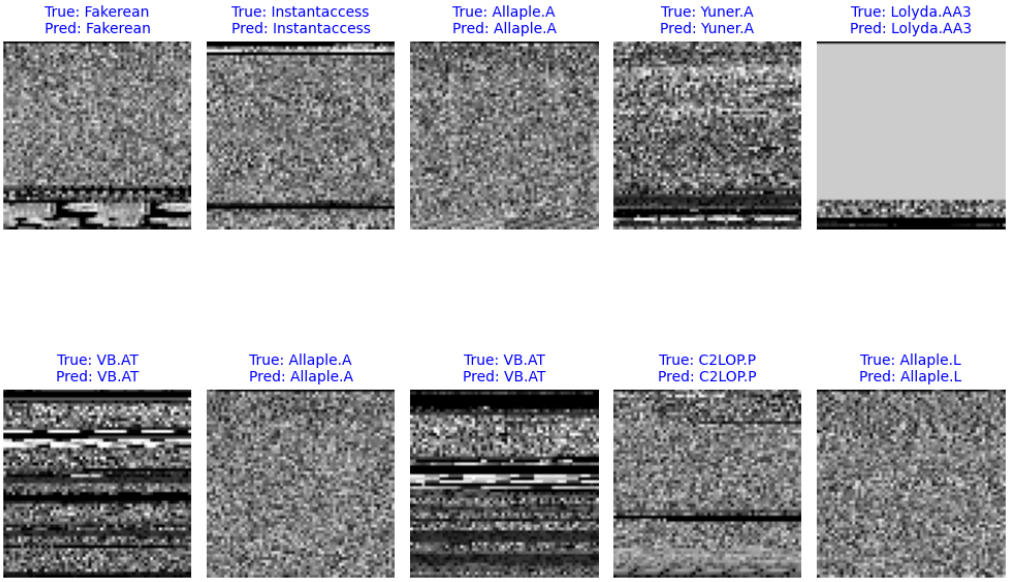


Figure 5.70: Malimg Prediction Results

Critical Analysis for Ensembling CNNs

- **Integration of Multiple Models:**
  1. The experiment includes several machine learning models such as from the more traditional algorithms like K-Nearest Neighbors and simple Decision Trees to more complex models such as Convolutional Neural Networks.
  2. Therefore, this integration opens up avenues for a relative performance comparison among different models on the malware image dataset.
- **Ensemble Learning with CNNs:**
  1. The experiment goes beyond the training of individual models by exploring ensemble learning by combining predictions from different instances of CNNs.
  2. Generally, ensemble learning is a very powerful technique in enhancing performance and robustness in predictive analytics [203].
- **Dynamic Learning Rate Adjustment:**
  1. The experiment uses dynamic learning rate tuning during CNN training, relying on a callback called ReduceLROnPlateau.

2. Adaptive learning rate contribute to better convergence and generalization of models [204].

Table 5.19: Malware Prediction Contributions

Contributions from Experiments	
Experiment	Important Contributions
CNN	• Inclusion of image data augmentation techniques using Keras' ImageDataGenerator is unique. • Use of early stopping and learning rate reduction on plateau callbacks during model training were used to optimize model performance and avoid overfitting.
ResNet-50 Transfer Learning	• Application of random erasing as a data augmentation technique is unique. Random erasing introduces randomness by erasing random patches of the image. • Freezing layers up to a specific point in the pretrained ResNet-50 model is a strategic decision. It allows the model to fine-tune on virus-related features while preserving the knowledge gained from general image recognition. • Use of a learning rate reduction strategy (using 'ReduceLROnPlateau' callback) helps in adapting the model's learning rate during training, potentially overcoming plateaus in the optimization landscape.

5.2.12 ResNet-50 with Attention Network

Due to the advancement of very deep neural networks learning features has become easier from lower layers to very complex deeper layers. However deep neural networks can suffer from vanishing gradient problems [205] where neural network may not count the earlier layers as it progresses towards the deeper layers. As a result, either the performance remains same or deterio-

rates quickly. ResNet models can greatly help to mitigate the vanishing gradient problems. As ResNet facilitates “skip connection" or “shortcut" which provides an alternative path to backpropagate the gradients with the earlier layers [206]. In the family of Residual Network there are ResNet-18, ResNet-50, ResNet-101 and ResNet-152, this research used ResNet-50 with an Attention Module which includes channel attention and spatial attention branches to generate 3D attention maps to produce feature maps from the input features. The attention network can be integrated with any CNN or feed forward network to produce a hierarchical attention to have a human perception like framework.

In CNN or any feed forward network attention modules are incorporated to make the model learn important features. Generally attention modules are implemented with 2D-convolutional layer, MLP (for channel attention) and a sigmoid function to constrain the input feature map.

The input of of an attention model is $CxHxW$ feature map to produce $1xHxW$ or $CxHxW$ (for 3D Attention) output attention map. After the element wise multiplication of input feature map and output attention map a refined output (feature map) is generated. There are two dimensions to apply attention mechanism: spatial and channel dimensions (Figure 5.71[7]). In this experiment the attention mechanism was implemented on ResNet-50 architecture [207].

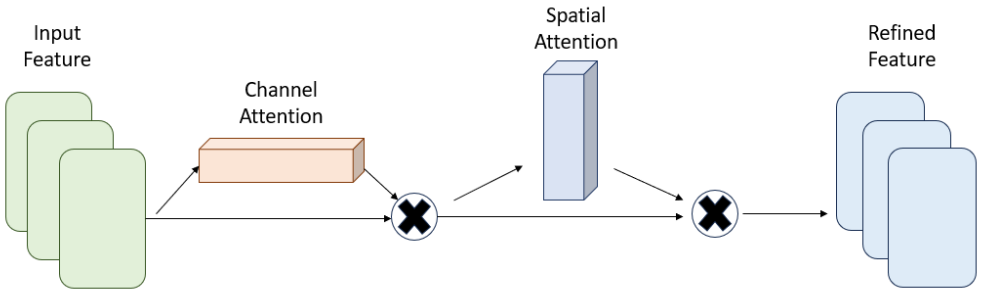


Figure 5.71: Spatial and Channel Attention Modules

[7] https://github.com/asdf2kr/BAM-CBAM-pytorch

Attention Module

To generate the Attention Module a 3D attention map $M(F) \in C * H * W$ is always generated from an input feature map $F \in$ C×H×W . The output feature map is calculated by adding the input feature map with the element-wise multiplied value of input feature and 3D attention map. The formula to calculate the output feature map F' is:

$$F' = F + F \otimes M(F) \tag{5.1}$$

The multiplication $\otimes$ was applied on the elements of input feature map F and the attention map M(F). To deal with the gradient vanishing this method was coupled with ResNet architecture. Channel attention $M_c(F) \in R^C$ was first calculated followed by the spatial attention $M_s(F) \in R^{H}{}^{*W}$ to compute the final attention map:

$$M(F) = \sigma(M_c(F) + M_s(F)) \tag{5.2}$$

The Sigmoid function σ is the attention vector here.

Channel Attention:

For this experiment, the Channel Attention model was designed to model the inter-channel dependencies. Channel Attention was developed by a global average pooling method to compress spatial dimensions of the feature map thereby producing a channel vector. A multi-layer perceptron (MLP) was added to generate channel-wise weights for rescaling the feature maps. After that a batch normalization (BN) was added with the output of spatial branch. The Channel Attention produces a channel-wise attention map to highlight the most informative filters.

Mathematically, the channel attention is defined as:

$$M_c(F) = \sigma\left(\text{MLP}\left(\text{AvgPool}(F)\right) + \text{MLP}\left(\text{MaxPool}(F)\right)\right), \tag{5.3}$$

where F denotes the input feature map, $M_c(F)$ is the channel attention

map, σ represents the sigmoid activation, and the two pooling operations extract global descriptors.

The refined feature map was computed with:

$$F' = M_c(F) \otimes F, \tag{5.4}$$

where $\otimes$ denotes element-wise multiplication between the attention weights and the original feature map.

Spatial Attention:

The Spatial Attention model focuses on spatially important regions of the feature maps. In the spatial attention branch, dilated convolution was used to construct spatial map. It learns spatial dependencies by focusing on key regions and captures local and contextual dependencies [207].

To generate the spatial attention map in the neural network the inter-spatial relation of the features are used. Spatial attention map is computed as $M_s(F) \in R^{H*W}$ and it focuses on the contextual information in different spatial locations. In case of spatial attention the main objective is to get the contextual information from large receptive field. To achieve this goal dilated convolution is more effective than the standard convolution.

Spatial Attention focuses on inter-spatial dependencies to allow the network to emphasize on informative regions of a feature map. If an intermediate feature representation is F, the spatial attention could be calculated as:

$$M_s(F) = \sigma\left(f^{k\times k}\left([\text{AvgPool}(F);\ \text{MaxPool}(F)]\right)\right), \tag{5.5}$$

- $f^{k\times k}$ denotes a convolution operation with kernel size $k \times k$),
- σ represents the sigmoid activation,
- $[\cdot;\cdot]$ indicates the concatenation along the channel axis.

The spatially refined output is:

$$F' = M_s(F) \otimes F, \tag{5.6}$$

where $\otimes$ denotes element-wise multiplication between the spatial attention map and the input feature representation.

Combination of Attention Modules

These two attention maps: Channel attention and Spatial attention were combined using the addition method rather than any other combining methods to facilitate the gradient flow during the backpropagation.

This attention network was implemented on the ResNet-50 architecture. In every layer of the ResNet-50 the attention module was integrated to have the hierarchical attention structure. Figure 5.72 is showing the Attention Network was integrated with the ResNet-50 layers [207].

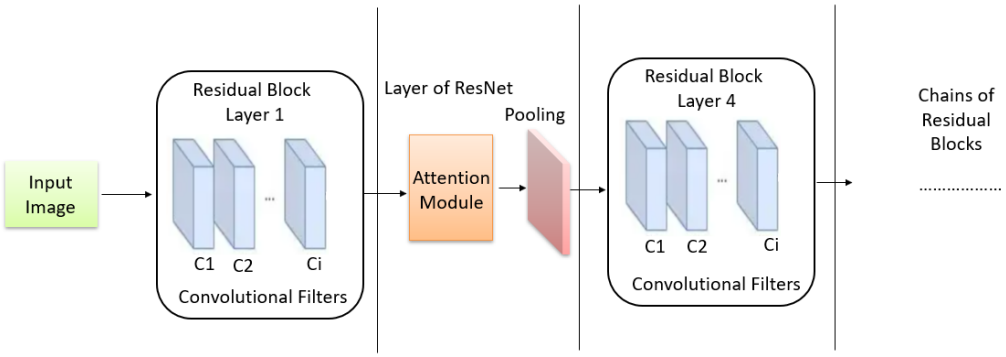


Figure 5.72: ResNet with Attention Module

Algorithm 9 ResNet-50 with Attention Module

Require: Dataset D, epochs E, batch size B, learning rate η, optimizer O, scheduler S, loss function L

Ensure: Trained classifier f_θ and evaluation metrics

```
Load dataset D, split into D_train, D_test
Preprocess images: resize (H × W), normalize, batch size B
Define Attention Module (ChannelGate + SpatialGate)
Initialize ResNet-50 backbone
Insert Attention Module after final convolutional block
Replace final FC layer with |C| outputs
Set loss L = CrossEntropy
Initialize optimizer O with learning rate η
Define scheduler S (e.g., StepLR)
for epoch = 1 ... E do
    for each mini-batch (x, y) ∈ D_train do
        ŷ ← f_θ(x)
        Compute loss L(ŷ, y)
        Backpropagate and update θ using optimizer O
    end for
    Evaluate f_θ on D_test
    Update learning rate using scheduler S
    Record accuracy and loss
end for
Generate accuracy/loss plots and confusion matrix
return f_θ and evaluation metrics
```

Table 5.20: Terms and Notation for ResNet-50 with Attention Module

Symbol / Term	**Description**
D	Dataset of images and labels
D_{train}	Training split of the dataset
D_{test}	Test split of the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample
H, W	Height and width of input images
B	Mini-batch size
E	Number of training epochs
η	Learning rate for optimizer
O	Optimizer (e.g., Adam, RMSProp)
S	Learning rate scheduler (e.g., StepLR)
L	Loss function (CrossEntropy)
f_θ	ResNet-50 model with attention module and parameters θ
C	Set of classes
Attention Module	Includes ChannelGate and SpatialGate to refine feature maps
$\hat{y}$	Predicted output of model f_θ
Accuracy	Fraction of correctly predicted labels on D_{test}
Confusion matrix	Table summarizing classification performance across all classes
Plots	Accuracy/loss curves for training and validation

Steps of ResNet-50 Attention Module

The experiment was conducted according to Algorithm 9 with terms and notation were described in Table 5.20:

1. **Dataset Preparation**
    - At firast, the dataset was loaded and divided into training and test (or validation) subsets.
    - All images were preprocessed by resizing them into a fixed resolution $(H \times W)$.
    - Pixel values were normalized to a standard range such as $[0, 1]$ and the data is organized into mini-batches.
2. **Model Definition**

- The backbone of the model was implemented with ResNet-50. To enhance the feature representation, an Attention Module was integrated to the model.
- The Attention Module consists of two mechanisms: channel attention and spatial attention.
- The Attention Module was added after the final convolutional block of ResNet-50.

3. **Model Training**

   To proceed with the training process following operations were performed:

   - For every mini-batch (x, y), a forward pass was performed to compute the predictions $\hat{y} = f_\theta(x)$.
   - The cross-entropy loss $\mathcal{L}(\hat{y}, y)$ was computed.
   - Backpropagation was incorporated and the model parameters θ were adjusted with optimizer.

4. **Evaluation**
   - After training, the model was tested on the test dataset.
   - Confusion matrices and plots of accuracy and loss curves were generated to provide detailed insights of the model.

Propositions of ResNet 50 with Attention Module

1. **Proposition 1: Attention Module to Enhance Feature Representation**
   - **Proposition:** In this experiment, Attention Module was integrated with ResNet-50 to improve the model's performance to capture both spatial and channel features for better discrimination of malware images.
   - **Rationale:** For every malware dataset, malware images may have precise visual patterns critical for malware classification. Attention Module explicitly enhances the model's accuracy compared to baseline the ResNet-50 models [208].
2. **Proposition 2: Effective Handling of Class Imbalance**

- **Proposition:** Attention mechanisms can reduce the effect of class imbalance by focusing on the informative features in the smaller sample-sized classes [209], with the aim of generating a well-balanced performance across the categories.
- **Rationale:** Some of these classes have an extremely lower number of samples within malware datasets. This approach reduces the bias towards majority classes in malware datasets

3. **Proposition 3: Generalization Across Training and Test Sets**
    - **Proposition:** The model, with an incorporated Attention Module, will generalize well for both training and test datasets, hence little or no overfitting and good performance.
    - **Rationale:** This generalization will make the model reliable for real-world applications, outside of malware datasets.
4. **Proposition 4: Visualization Validates Attention Mechanism**
    - **Proposition:** Visualization of results from prediction shows that the module focuses on malware-relevant features of the images.
    - **Rationale:** Interpretability and explainability are the key features of deep learning based cybersecurity solutions [210].

Virus MNIST Dataset Results

The whole ResNet-50 with the attention architecture was implemented on the Virus-MNIST dataset for prediction of classes. For the Top-1 accuracy the model achieved 99.99% training accuracy and 92.35% test accuracy and for the Top-5 accuracy the training accuracy was 100% and test accuracy was 99.31%. Figure 5.73 is showing the Top-1 accuracy and loss graphs of the model.



Figure 5.73: Top-1 Accuracy and Loss

Figure 5.74 is showing the prediction of classes on the test dataset.

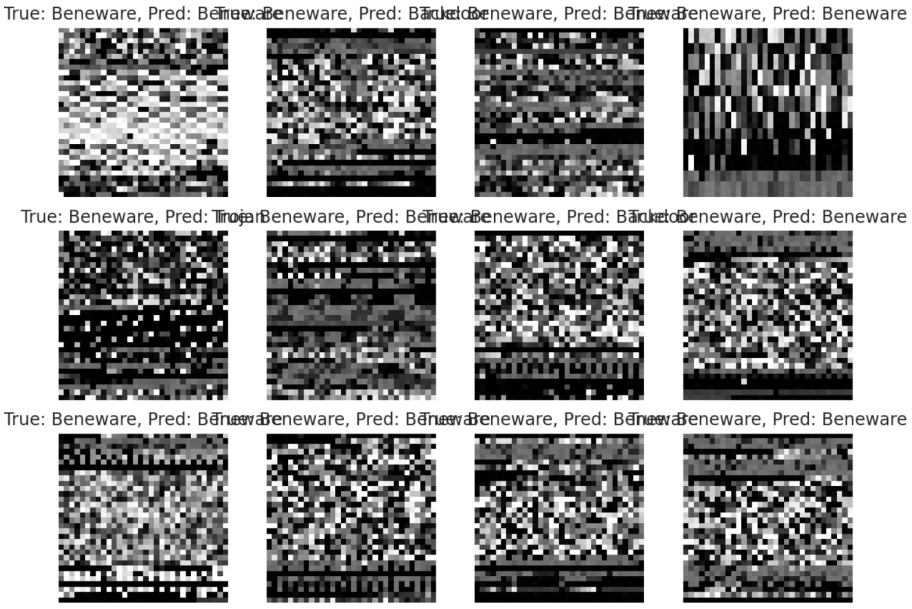


Figure 5.74: Prediction results of Virus MNIST

MaleVis Dataset Experiment



Figure 5.75: Accuracy Curve

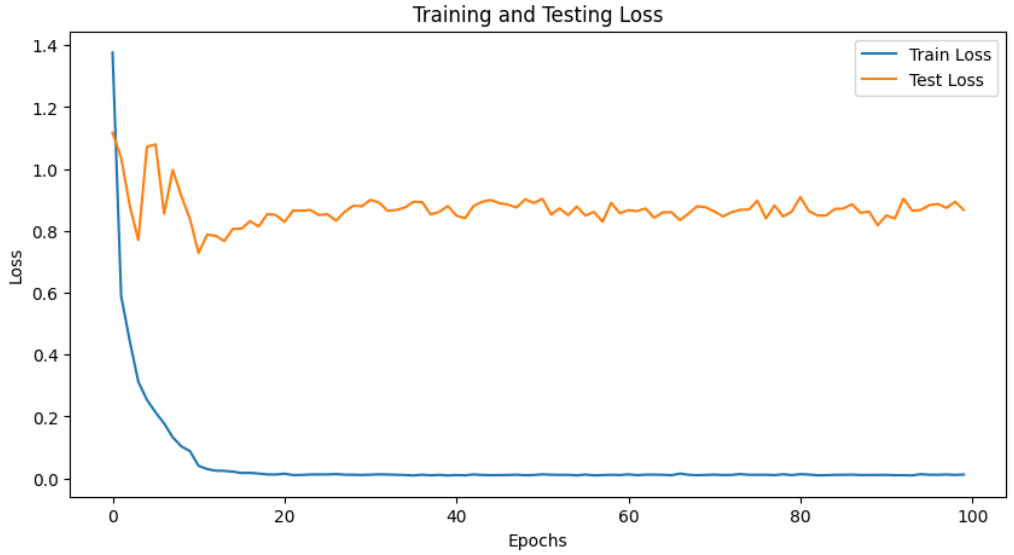


Figure 5.76: Loss Curve

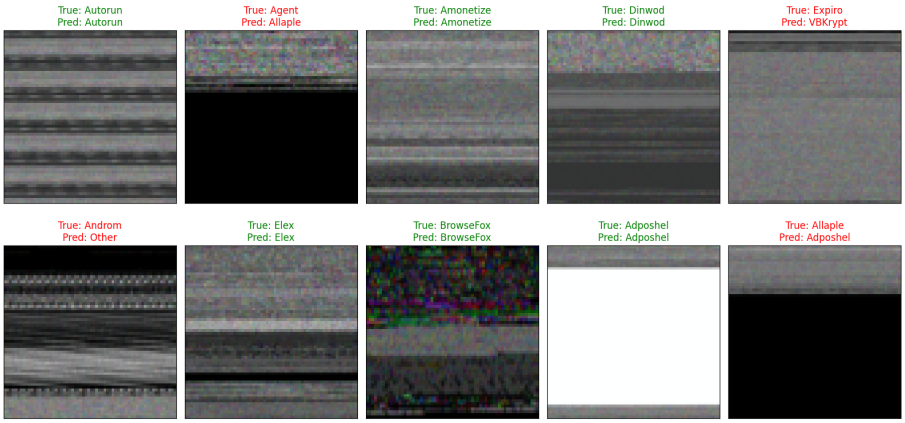


Figure 5.77: MaleVis Prediction Results

Malimg Dataset Results

Figure 5.78 is showing the confusion matrix of Malimg dataset.

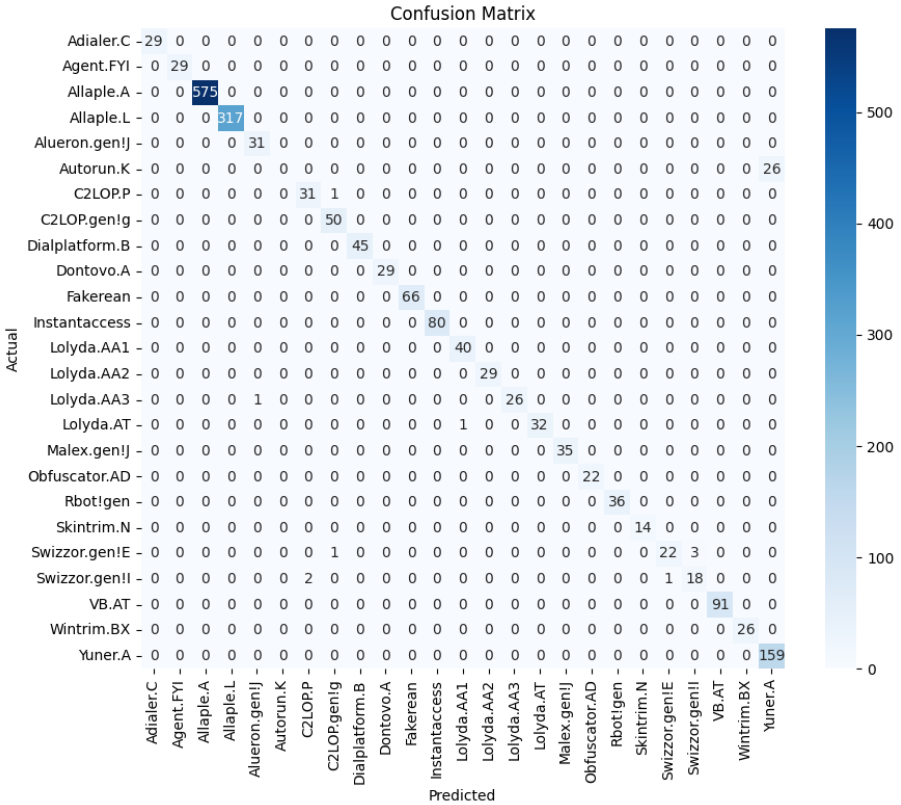


Figure 5.78: Confusion Matrix

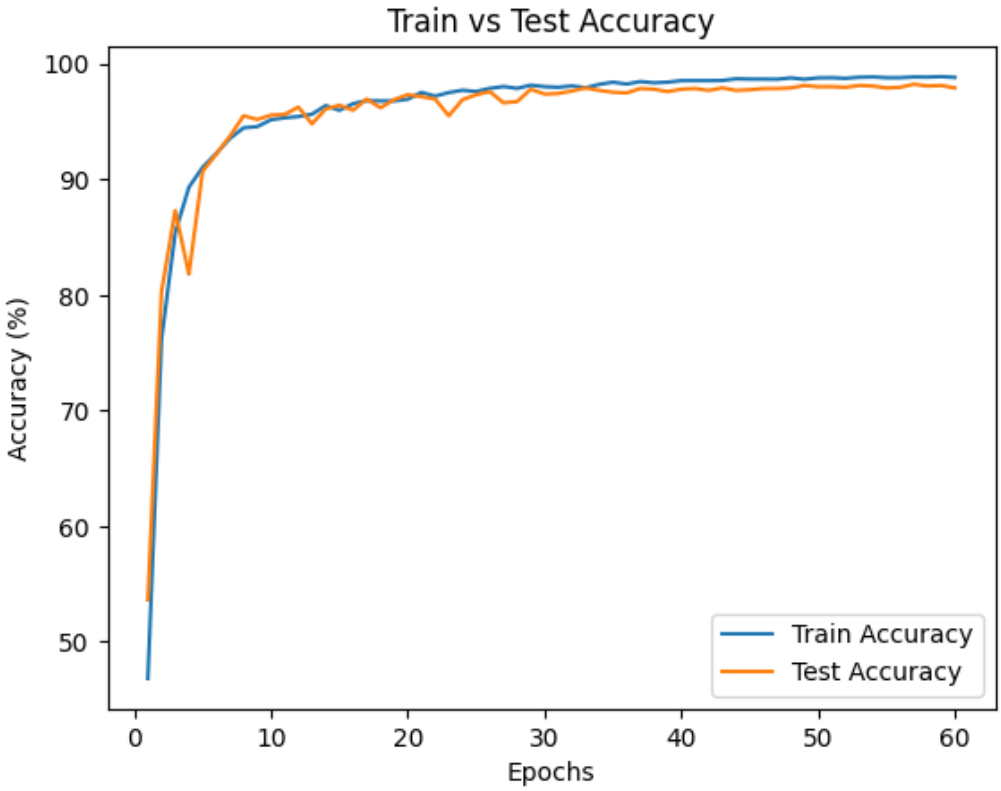


Figure 5.79: Malimg Accuracy Curve

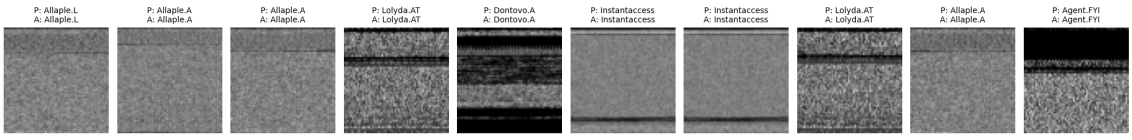


Figure 5.80: Malimg Dataset Prediction Results

Table 5.21: ResNet-50 Attention Results

Results		
Dataset	Training Accuracy	Test Accuracy
Malimg	98.82%	97.91%
Virus MNIST	99.99%	92.35%
MaleVis	99.67%	84.63%

Table 5.21 is showing that:

- Overall, the ResNet-50 Attention experiments have produced consistently superior training performances for all datasets.
- Malimg and Virus MNIST datasets were trained for 60 epochs and Male-Vis was trained for 100 epochs to achieve a promising accuracy.
- Malimg dataset produced highest test accuracy (97.91%), Virus MNIST produced near-perfect test accuracy (92.35%), which suggests attention mechanism effectively enhanced discriminative feature extraction.
- MaleVis dataset exhibited a challenging learning as it produced a modest test accuracy (84.63%). This is because of the RGB composition, complex nature and high heterogeneity.

Critical Analysis of ResNet-50 with Attention Module

- **Incorporation of Attention Module:**

  This work introduces a varied level of attention mechanisms through an attention network into the model, which may allow it to focus on more relevant spatial and channel-wise features, enhancing its discriminative power to improve the network.
- **Comprehensive Evaluation Metrics:**

  This experiment tests the model based not just on standard loss but also gives accuracies on both training and testing, which give a wider view of the performance of the model.
- **Visualization of Results:**

  The code also includes visualization for training and test losses, and top-1 accuracy over epochs. This plot provides insightful visual feedback for assessing the training dynamics.

Table 5.22: Malware Classification Experiment

Contributions from Experiments	
Experiment	Important Contributions
3D CNN	• The architecture of the neural network is specifically designed for 3D data, incorporating Conv3D layers, max-pooling in 3D, and a global average pooling layer. • This setup demonstrates an attempt to capture spatial relationships in three dimensions, which is a departure from conventional 2D CNNs used in image classification tasks. • The use of a Global Average Pooling layer in 3D is noteworthy. This layer provides a more compact representation of the spatial features in the 3D volume.
CNN with Data Augmentation (Horizontal flip)	• Applying data augmentation techniques, such as horizontal flipping, to malware images is a novel aspect. This helps improve the model's ability to generalize and recognize various orientations of malware instances. • There are 10 classes representing different types of malware. Handling class imbalance in the dataset is crucial for effective model training and generalization. • The preprocessing steps, including data normalization and reshaping, are crucial for preparing the input data for the neural network.

Table 5.23: Malware Classification Experiment

Contributions from Experiments	
Experiment	Important Contributions
ResNet-50 with Attention Module	• This attention mechanism introduces different levels of the network, allowing the model to focus on more relevant spatial and channel-wise features. • The experiment evaluates the model not only based on standard loss values but also includes top-1 accuracy during both training and testing phases. • The code includes visualizations of training and test losses, as well as top-1 accuracy over epochs. This kind of visual feedback is valuable for understanding the training dynamics.

5.2.13 Classification with CNN Normalization

In this experiment mean normalization was used by calculating the normalized score of the dataset. A visualization of train images was shown in Figure 5.81 to have a clear understanding of class labels as predicted test images need to be visualized accordingly after the experiment.

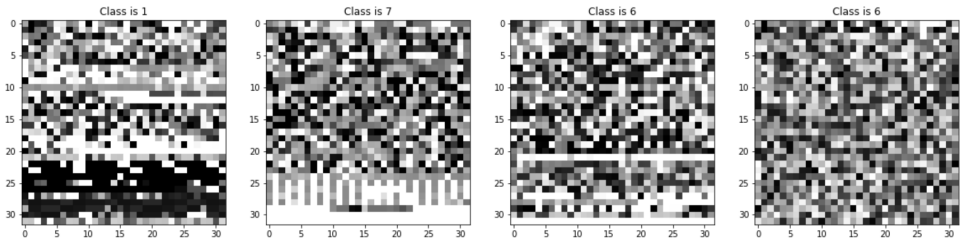


Figure 5.81: Train Images to feed into the Model

After calculating the mean and standard deviation of the train data all the columns have been standardized by using the formula: $(x - mean)/std$. The

CNN model was defined by using convolutional layers, pooling layers, batch normalization layers also flatten and dense layers at the end. In this experiment image augmentation was used by using Keras image augmentation API with 'ImageDataGenerator' to define data preparation and augmentation of images. For the 'ImageDataGenerator' the rotation range was 15 i.e. images of the dataset need to be randomly rotated up to 15 degrees during training, zoom range was 0.01 to randomly zoom the images also width shift range was 0.1 and height shift range was 0.1, here random shifts were used to artificially shift the off-center objects in the images of training data.

Algorithm 10 CNN-based Image Classification

Require: Dataset D, input size (H, W), batch size B, epochs E
Ensure: Trained CNN model, evaluation metrics
Load D, resize (H, W), normalize pixels
Convert labels to categorical
Split $D \rightarrow (D_{train}, D_{val})$
Apply data augmentation on D_{train}
Initialize CNN:
Conv + ReLU + MaxPool + BatchNorm layers
Flatten
Dense + ReLU layers
Softmax output layer
Compile: Loss = CategoricalCrossEntropy, Optimizer = Adam, Metric = Accuracy
Set callbacks: EarlyStopping, ReduceLROnPlateau
for epoch = 1 ... E **do**
Train on mini-batches of D_{train}
Validate on D_{val}
end for
Predict on D_{test}: $\hat{y}$
Compute confusion matrix
Plot training accuracy/loss curves
Display sample predictions
return Trained CNN model, evaluation metrics

Table 5.24: Terms and Notation for CNN-based Image Classification

Symbol / Term	Description
D	Dataset of images and corresponding labels
D_{train}	Training split of the dataset
D_{val}	Test/validation split of the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample
H, W	Height and width of input images
B	Mini-batch size for training
E	Number of training epochs
C	Set of classes
$\hat{y}$	Predicted labels of the model
CNN layers	Convolution + ReLU + MaxPool + BatchNorm + Dense + Softmax
Loss	Categorical Crossentropy
Optimizer	Adam
Callbacks	EarlyStopping, ReduceLROnPlateau
Accuracy	Fraction of correctly predicted labels on D_{test}
Confusion matrix	Table summarizing classification performance across all classes
Training curves	Accuracy and loss plots for training and validation

Steps of the Experiment

The following steps were carried out to develop the Convolutional Neural Network-based pipeline according to Algorithm 10 with symbols were described in Table 5.24:

1. **Dataset Preparation:**
   - The dataset D was loaded, which consists of malware images and labels.
   - After that splitting of data into training and validation sets.
   - Images were resized into a fixed resolution (H, W) and normalized to the range $[0, 1]$.
2. **Model Construction:**
   - A CNN model was constructed with multiple convolutional layers, ReLU activation functions, followed by pooling and batch normalization layers.

- After feature extraction the feature maps were flattened and connected to the fully connected layers.

3. **Compilation:** The model was compiled using the categorical cross-entropy loss function:
$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i),$$
   - $\hat{y}_i$ is the predicted probability,
   - The Adam optimizer was incorporated for accuracy.

4. **Training:** The model was trained for a certain number of epochs with a batch size of B. Callbacks such as Early Stopping was applied to prevent overfitting and ReduceLROnPlateau was applied to adjust learning rates during training.

5. **Evaluation:** After training the model was evaluated on the test dataset. A confusion matrix was generated to analyze classification performance for different classes.

Propositions for Classification with CNN Normalization

- **Effectiveness of Convolutional Neural Networks (CNNs):**

  The proposed CNN architecture consists of multiple convolutional blocks, batch normalization and dense layers, has achieved high accuracy.
- **Impact of Data Augmentation:**

  Data augmentation techniques (e.g., rotation, width/height shifts) were applied through *ImageDataGenerator* to enhance robustness of the model and improve validation accuracy [180].
- **Normalization of Inputs:**

  Normalizing the pixel values to the range [0, 1] and standardizing them based on the training dataset's mean and standard deviation will result in faster convergence during training.

Virus MNIST Dataset Results

After generating the train and test image data the model was trained for 50 epochs and achieved 96.09% training accuracy and 86.53% validation accuracy. The accuracy graph was shown in Figure 5.82.

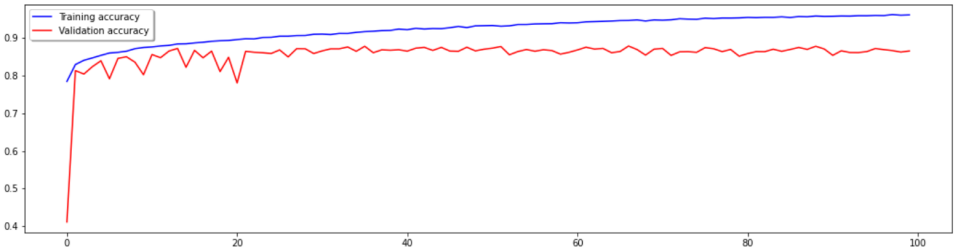


Figure 5.82: Accuracy Curve of CNN

The confusion matrix of the predicted result was shown in Figure 5.83 and the predicted result using the test dataset was shown in Figure 5.84.

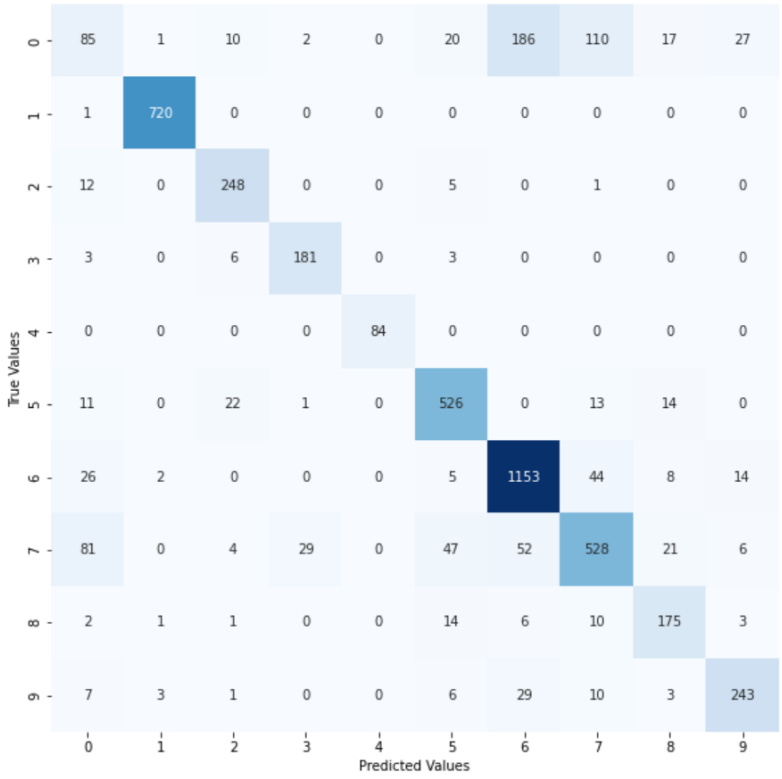


Figure 5.83: Confusion Matrix of CNN

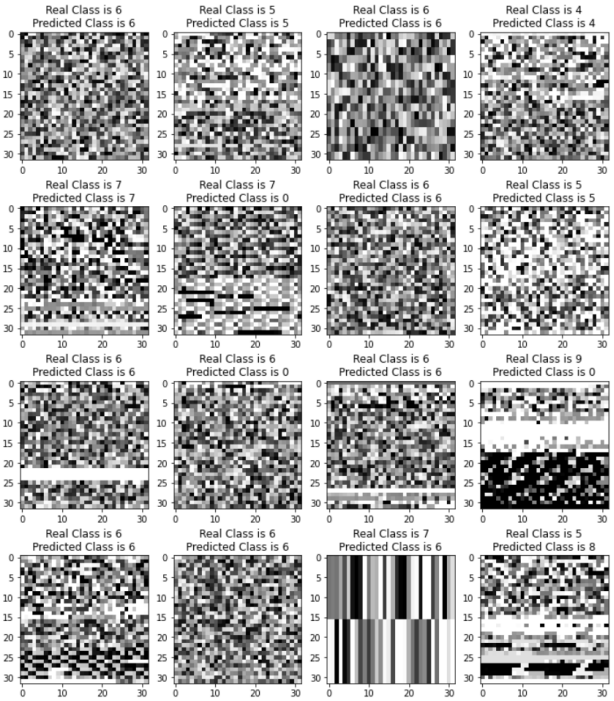


Figure 5.84: Predicted class labels after the experiment

MaleVis Dataset Results

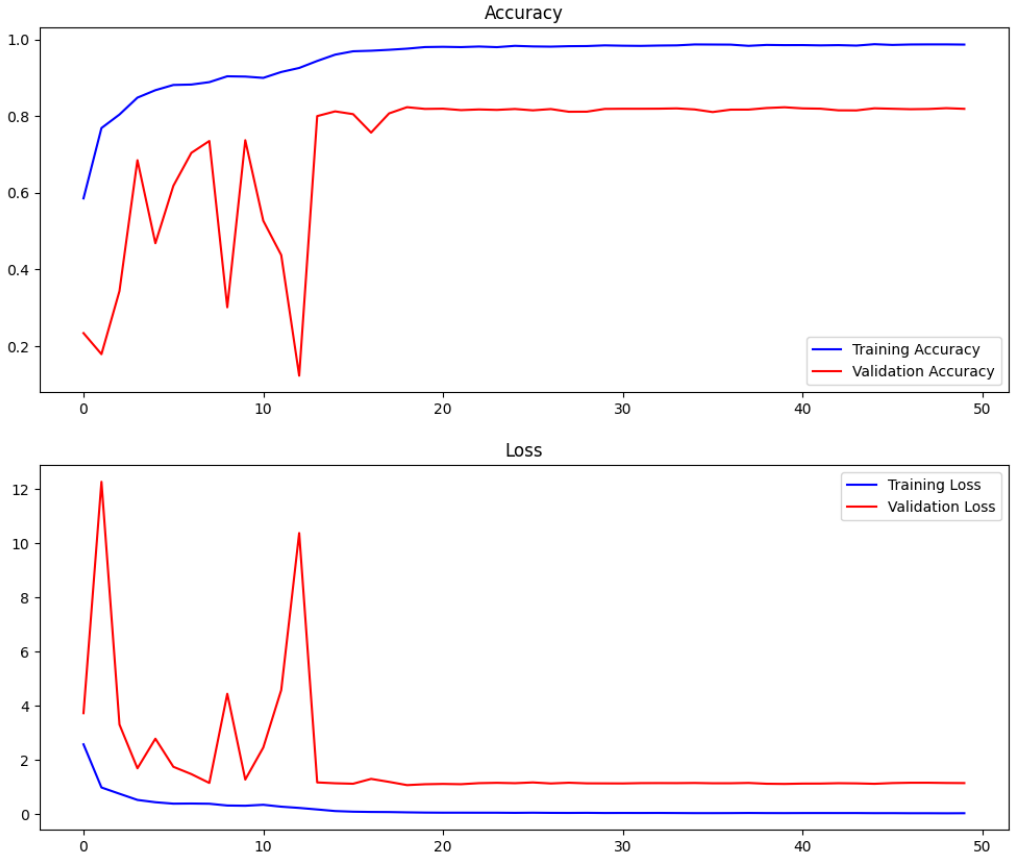


Figure 5.85: MaleVis Accuracy and Loss

Figure 5.86 is showing the confusion matrix of MaleVis dataset.

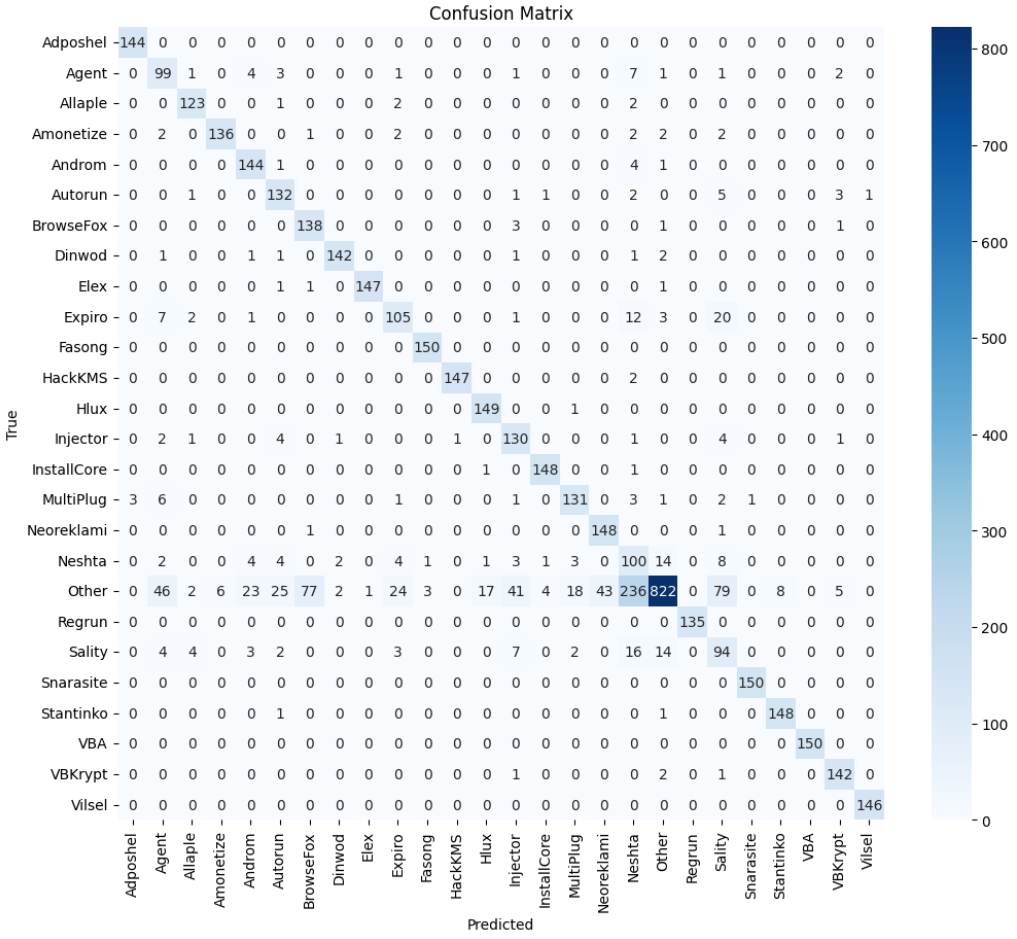


Figure 5.86: MaleVis Confusion Matrix

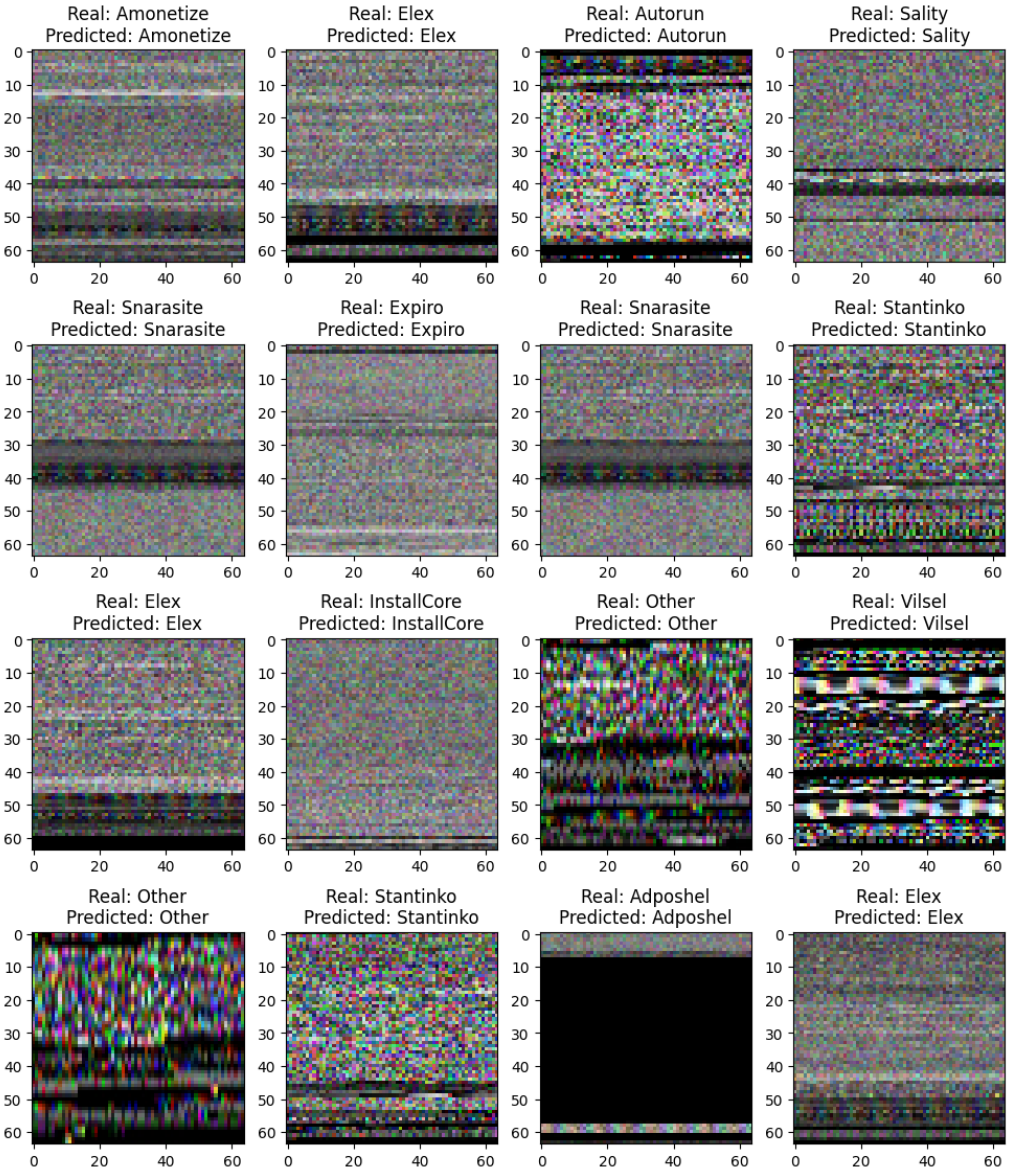


Figure 5.87: MaleVis Prediction Results

Malimg Dataset Results

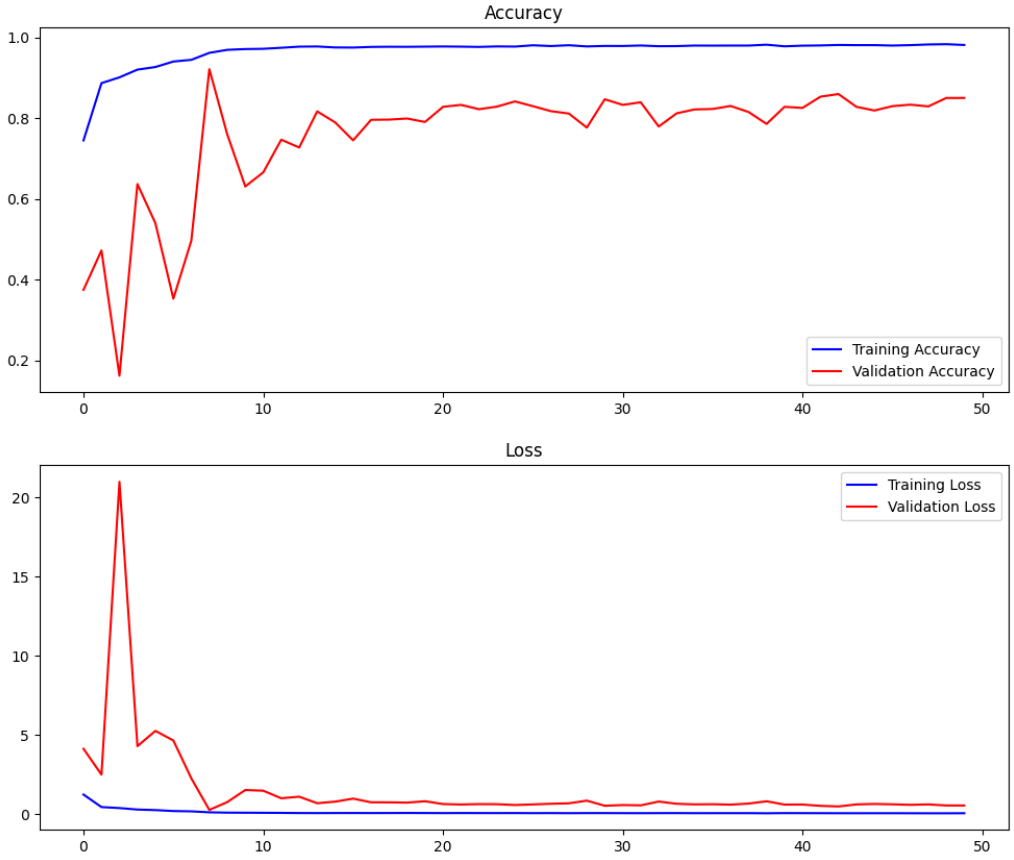


Figure 5.88: Malimg Accuracy and Loss

Figure 5.89 is showing the confusion matrix of Malimg dataset.

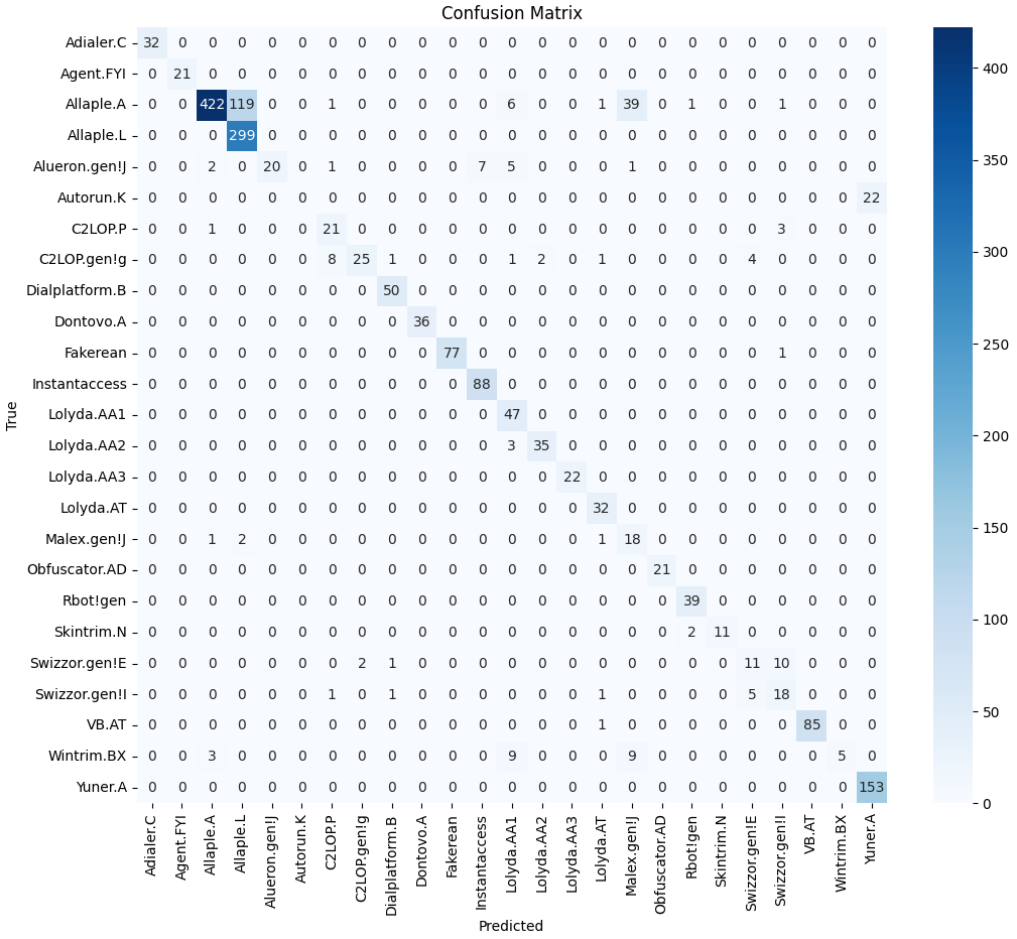


Figure 5.89: Malimg Confusion Matrix

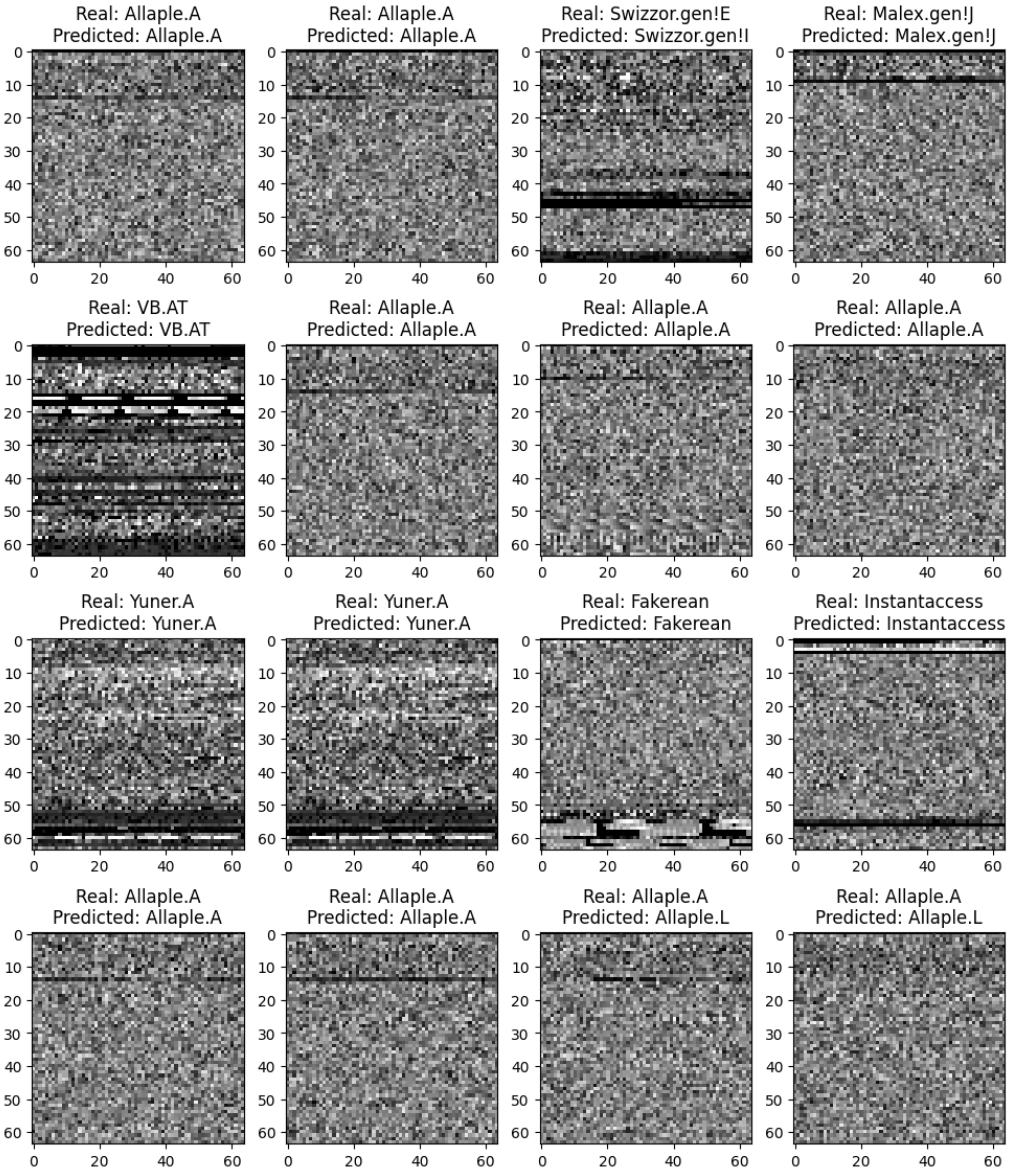


Figure 5.90: Malimg Prediction Results

Table 5.25 is showing the results with all three datasets.

Table 5.25: CNN Normalization Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	98.51%	85.01%
Virus MNIST	96.09%	86.53%
MaleVis	98.78%	81.94%

Table 5.25 compares the results of three datasets, all were trained for 50 epochs:

- In this experiment also, Virus MNIST and Malimg produced better generalization.
- The method was not sufficient for MaleVis because the dataset is RGB in nature and class complexity.

Critical Analysis for Classification with CNN Normalization

- **Data Augmentation:**

  The inclusion of data augmentation techniques like rotation, zoom, shift using Keras' ImageDataGenerator enhanced the model's ability to generalize by exposing it to a variety of augmented images during training.

- **Callbacks for Training:**

  The use of early stopping and learning rate reduction on plateau callbacks during model training indicates an effort to optimize model performance and avoid overfitting.

5.2.14 Classification with CNN and RNN

Combination of CNN and RNN

This experiment integrates Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) (specifically, Bidirectional GRU) to classify malware images by leveraging CNNs for spatial feature extraction and GRUs for temporal pattern recognition. The combination of deep learning, sequence

modeling, and optimization techniques makes it a robust approach for malware classification tasks.

CNNs extract spatial features using multiple convolutional layers, batch normalization, and max pooling, while GRUs model sequential dependencies within the extracted features. The Bidirectional GRU enhances learning by capturing patterns in both forward and backward directions, making it suitable for malware classification [211]. The model is optimized using categorical crossentropy loss, the RMSprop optimizer, and ReduceLROnPlateau to adjust the learning rate dynamically. Finally, performance is evaluated using accuracy, loss plots and confusion matrix, ensuring robustness in malware classification.

In CNN + GRU model, the temporal domain is introduced by treating spatial features extracted from images as sequences. The CNN layers extract spatial features from malware image datasets. These feature maps are flattened using TimeDistributed(Flatten()), essentially transforming a 2D image into a sequence of features. This sequence is then processed by a Bidirectional GRU to capture sequential dependencies. The Justification for Adding a Temporal Aspect are:

- **Mimicking Temporal Dependency**

  Although malware images do not have an explicit time-series nature, the GRU processes feature sequences, which can help learn hierarchical feature relationships across regions of the image.

- **Leveraging Recurrent Layers for Feature Learning**
  1. GRUs (and LSTMs) [212] are good at learning long-range dependencies.
  2. In malware classification, certain regions of an image might contain patterns that contribute more to classification [213].
  3. By adding a GRU layer, the model can focus on important sequential patterns across extracted features.

- **Alternative View of Image Data**
  1. Traditional CNNs process images as grids, but a CNN + RNN hybrid treats images as a sequence of extracted features [214].
  2. This could improve performance by capturing relationships between different feature regions.

Algorithm 11 CNN + RNN Training Pipeline

Require: Dataset D, batch size B, epochs E, learning rate η
Ensure: Trained CNN+BiGRU model M, evaluation metrics
Normalize images $\in [0,1]$, resize (H, W)
Split $D \rightarrow (D_{train}, D_{val}, D_{test})$
Apply data augmentation on D_{train}
Initialize CNN: Conv + ReLU + BatchNorm + MaxPool + Dropout
Flatten with TimeDistributed
Add BiGRU layer
Add Dense + ReLU layers
Add Softmax output layer
Compile: Optimizer = Adam/RMSprop(η), Loss = CategoricalCrossEntropy, Metric = Accuracy
for epoch = 1 ... E **do**
  Train on D_{train} batches
  Validate on D_{val}
  Update learning rate with ReduceLROnPlateau
end for
Predict on D_{test}
Compute accuracy and confusion matrix
Plot training/validation curves
Display confusion matrix
return Model M, evaluation metrics

Table 5.26: Terms and Notation for CNN + RNN Training Pipeline

Symbol / Term	Description
D	Dataset of images and corresponding labels
D_{train}	Training split of the dataset
D_{val}	Validation split of the dataset
D_{test}	Test split of the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample
H, W	Height and width of input images
B	Mini-batch size for training
E	Number of training epochs
η	Learning rate
CNN layers	Convolution + ReLU + MaxPool + BatchNorm + Dropout
TimeDistributed	Layer used to flatten feature maps across time steps
BiGRU	Bidirectional Gated Recurrent Unit layer for sequential dependencies
Dense layers	Fully connected layers with ReLU activation
Softmax	Output activation layer producing probability distribution over classes
Loss	Categorical CrossEntropy
Optimizer	Adam or RMSprop
ReduceLROnPlateau	Learning rate scheduler to adjust η during training
$\hat{y}$	Predicted labels of the model
Accuracy	Fraction of correctly predicted labels on D_{test}
Confusion matrix	Table summarizing classification performance across all classes
Training curves	Plots of loss and accuracy during training/validation

CNN + RNN Training Pipeline

The experiment was conducted according to Algorithm 11 and terms and notation were mentioned in Table 5.26:

1. **Data Preparation**
   - **Normalization:** For normalization, scale the pixel values of each image x to $[0, 1]$:

$$x_{\text{norm}} = \frac{x}{255}$$

- **Resize:** Images were resized to a fixed dimensions (H, W).
- **Dataset split:** The dataset D was divided into training (D_{train}), validation (D_{val}) and test (D_{test}) sets.

2. **Build Model Architecture**
   - **Convolutional layers:** For each convolutional layer, feature maps F were computed as:

$$F_{i,j,k} = \sigma\Big(\sum_{m=1}^{C_{\text{in}}} \sum_{p=1}^{K} \sum_{q=1}^{K} W_{p,q,m,k} \cdot X_{i+p-1,j+q-1,m} + b_k\Big)$$

     - σ is the ReLU activation,
     - K is the kernel size,
     - C_{in} is the input channel count,
     - W is the filter weight.
   - **Batch normalization and max-pooling:** To normalize activations and reduce spatial dimensions.
   - **Dropout:** This layer randomly deactivates neurons with probability p to prevent overfitting.
   - **Flattening:** `TimeDistributed` layer was used to convert feature maps into sequences for GRU input.
   - **Bidirectional GRU:** For sequence $\mathbf{x} = [x_1, x_2, ..., x_T]$, hidden states are computed as:

$$\overrightarrow{h_t} = \text{GRU}(x_t, \overrightarrow{h_{t-1}}), \quad \overleftarrow{h_t} = \text{GRU}(x_t, \overleftarrow{h_{t+1}})$$

$$h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$

   - **Fully connected layers:** To aggregate features for classification.
   - **Output layer:** Softmax activation for multi-class probability.

3. **Train Model**
   - Training of model for E epochs with mini-batches of size B.
   - Validate on D_{val} to monitor performance.
   - Adjust learning rate dynamically using a scheduler.

4. **Evaluate Model**
   - Prediction of model on D_{test}.
   - Calculation of the metrics: accuracy and confusion matrix.

Proposition for Combining CNN and RNN

1. **Proposition:** The hybrid CNN-GRU architecture improves malware image classification performance compared to CNN-only models.

   Rationale: CNNs are particularly good at learning spatial features and GRUs can learn potential sequential relations [215] within the malware frameworks, which leads to richer feature representation.
2. **Proposition:** Malware images exhibit hidden sequential dependencies that can be learned by a GRU-based model.

   Rationale: Although malware images are static but their pixel configurations can encode patterns relative to opcode sequences [216], instruction streams or file structures and these cen be utilized by a recurrent model.
3. **Proposition:** Bidirectional GRUs can enable better feature learning by capturing one relation in both directions forward and backward.

   Rationale: Malware families tend to have structural similarities and bididirectional processing can help detect contextual patterns a unidirectional model can overlook [217].
4. **Proposition:** The TimeDistributed layer aids in maintaining spatial features when switching from CNN to GRU.

   Rationale: Without this layer, directly flattening CNN outputs could lose crucial spatial information, while TimeDistributed(Flatten()) ensures sequential preservation before GRU processing.
5. **Proposition:** Dropout and Batch Normalization improve generalization of models and prevent overfitting.

   Rationale: Malware datasets are usually class imbalanced and regularization methods like Batch Normalization and Dropout make training stable and improve the adaptability of the model on unseen samples.

Virus MNIST Results

The confusion matrix of the model was shown in Figure 5.91.

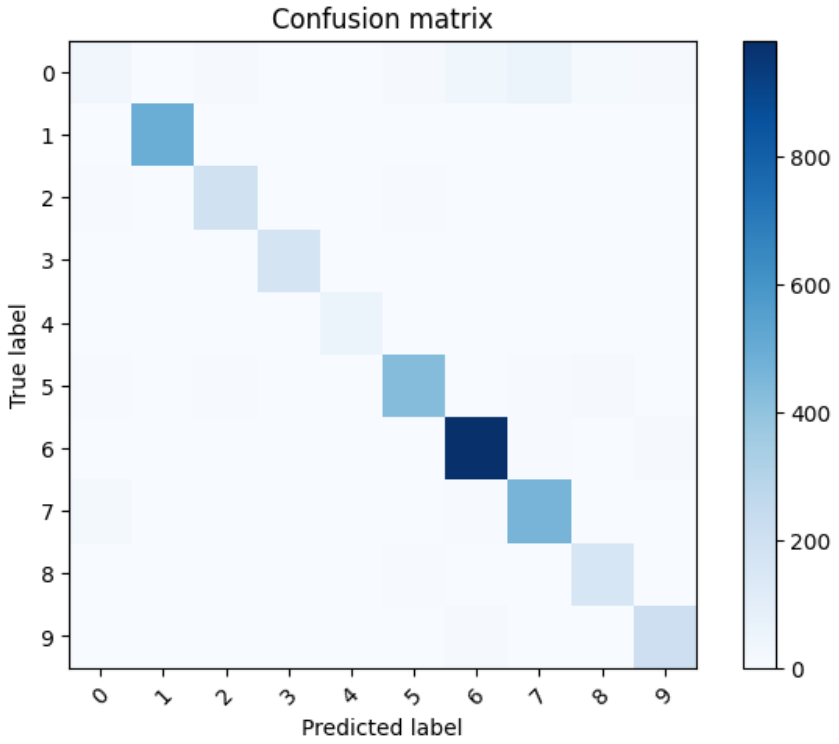


Figure 5.91: Confusion Matrix of Virus MNIST

MaleVis Dataset Results

Figure 5.92 is showing the confusion matrix of MaleVis dataset.

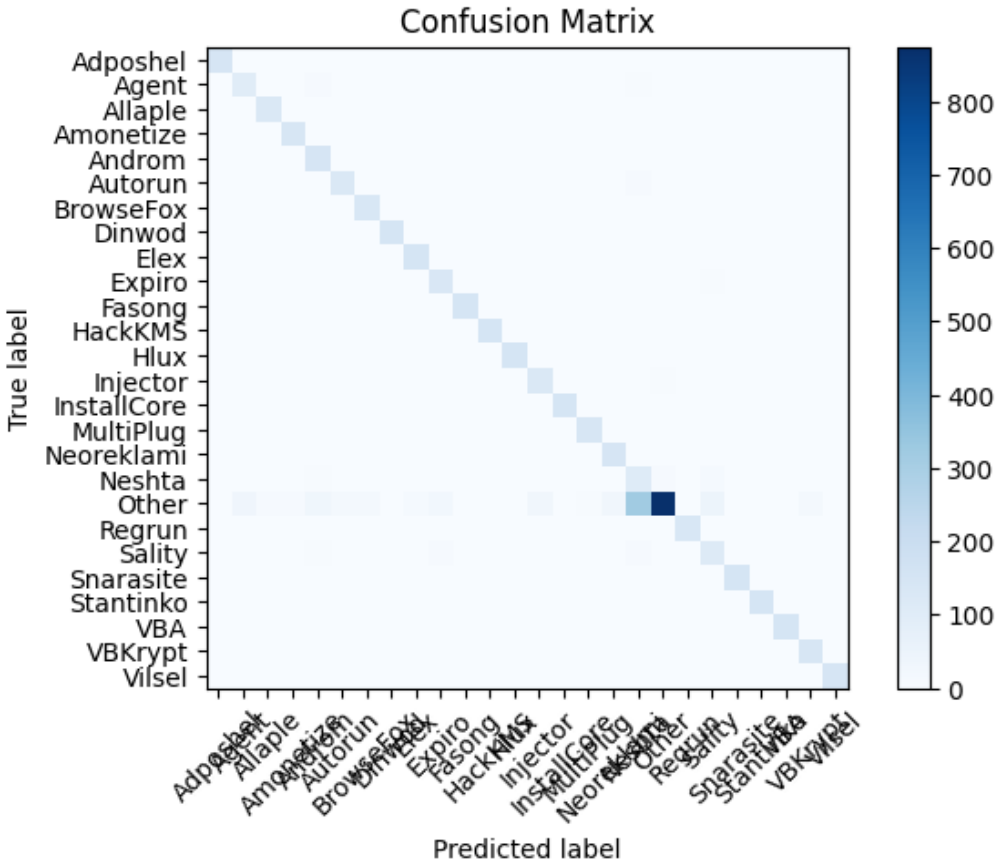


Figure 5.92: Confusion Matrix of MaleVis

Table 5.27: Results of Combining CNN and RNN

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	100%	98.05%
Virus MNIST	93.18%	92.72%
MaleVis	96.70%	94.51%

Table 5.27 is showing the results with three datasets, all were trained for 50 epochs:

- The combined CNN and RNN model achieves excellent performance for all three datasets.

- Malimg and MaleVis produced promising performance due to their rich texture.
- Virus MNIST is a simpler dataset but produced stable training and testing alignment with minimal overfitting.

Critical Analysis for Combining CNN and RNN

- **Hybrid CNN-GRU Architecture:** Combines Convolutional Neural Networks (CNNs) to extract spatial features with Bidirectional GRUs to model sequences, an unconventional task in malware image classification.
- **TimeDistributed Layer Integration:** Employs a TimeDistributed(Flatten()) layer to link CNN and GRU that preserves the spatial relationships before the sequential learning process, something used very little for malware image processing.
- **Bidirectional GRU for Malware Images:** This method is different from standard CNN-based malware classifiers since this experiment employs Bidirectional GRUs to detect dependency between features in both forward and backward directions [211], a novel intuition for static image classification.
- **Adaptive Learning Strategy:** Incorporates ReduceLROnPlateau, which dynamically adjusts the learning rate as a function of validation performance to enhance training stability and convergence.
- **Potential for Hierarchical Feature Learning:** The GRU module can unveil hidden sequential relationships [218] between malware families, which leads to a novel representation of malware behavior in cyber security.

Classification with CNN and RNN

This experiment was carried out to classify the dataset using CNN and then RNN. For this experiment, two deep learning models, one Convolutional Neural Network (CNN) and one Recurrent Neural Network (RNN) with LSTM layers, are trained and evaluated on malware image datasets. The CNN model for image classification uses convolutional layers, max pooling, batch normalization, and dropout to extract spatial features from grayscale images. The RNN,

with LSTM layers, treats every image as a sequence of pixels and learns time dependencies among them. Both the models are trained, and their performance is evaluated on test sets with accuracy, loss, and confusion matrices. The CNN should perform better because it is optimized for extracting spatial features, and the RNN is trying out the application of LSTMs to image classification.

The temporal domain is introduced through the RNN (LSTM-based) model, which treats each row of the 2D image as a time step. The images are originally 2D. For the CNN, treating the images as spatial feature maps. For the RNN, where each row is treated as a time step, resulting in a sequence of length. This allows the LSTM layers to process the image data sequentially, treating each row as a temporal event. Justification for Adding a Temporal Aspect are:

- **Exploring Spatial Correlations in a Sequential Manner:**

  Instead of analyzing the entire image at once (as CNN does), the RNN can capture dependencies between pixel rows, which might be useful if the malware image structures contain sequential patterns.
- **Potentially Capturing Malware Behavior Over Time:**

  If malware images encode execution traces, instruction sequences or structural variations in a way that follows a top-to-bottom pattern, LSTMs could extract hidden time-dependent patterns.
- **Alternative Feature Representation:**

  CNNs focus on spatial relationships, whereas LSTMs offer a different perspective by processing rows as sequences [219], possibly capturing long-range dependencies across the image.

Algorithm 12 CNN and RNN Malware Image Classification

Require: Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, hyperparameters (B, E, η, H, W)
Ensure: Trained model M, evaluation metrics
$x_i' \leftarrow x_i / 255$ {Normalize pixels}
Encode $y_i \in \{0,1\}^{\mathrm{C}}$, split $D \rightarrow D_{train}, D_{val}, D_{test}$
$f^{(l)} \leftarrow \sigma(W^{(l)} * f^{(l-1)} + b^{(l)})$ {Convolution + ReLU}
$p^{(l)} \leftarrow \text{maxpool}(f^{(l)})$ {Pooling}
Flatten $\rightarrow z \in \mathbb{R}^{F}$, reshape $z \rightarrow (T, F)$
$r_t \leftarrow \sigma(W_r x_t + U_r h_{t-1} + b_r)$
$z_t \leftarrow \sigma(W_z x_t + U_z h_{t-1} + b_z)$
$\tilde{h}_t \leftarrow \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$
$h_t \leftarrow (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
$h_t \leftarrow [h_t^{\rightarrow}; h_t^{\leftarrow}]$ {Bidirectional}
$o \leftarrow W_o h_T + b_o$
$\hat{y} \leftarrow \text{softmax}(o)$
$\mathcal{L} \leftarrow -\frac{1}{B} \sum_{i=1}^{B} \sum_{c=1}^{\mathrm{C}} y_{i,c} \log(\hat{y}_{i,c})$ {Cross-Entropy Loss}
$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}$ {Gradient update}
for epoch $= 1$ to E **do**
  Train on mini-batches $\{(x_i, y_i)\}_{i=1}^{B}$
  Validate on D_{val}
  Adjust η if plateau
end for
$\hat{y}_i \leftarrow \arg\max_c f_{\theta}(x_i)$ {Prediction}
Evaluate accuracy, loss, confusion matrix

Table 5.28: Terms and Notation for CNN and RNN Classification

Symbol / Term	Description
D	Dataset of malware images and labels
D_{train}	Training split of the dataset
D_{val}	Validation split of the dataset
D_{test}	Test split of the dataset
x, x_i	Input image or sample
y, y_i	Label of input sample
H, W	Height and width of input images
B	Mini-batch size for training
E	Number of training epochs
η	Learning rate
CNN layers	Conv2D + ReLU + BatchNorm + MaxPooling2D + Dropout + Flatten
Sequence reshape (T, F)	Flattened CNN features reshaped as sequence for RNN input
BiGRU/LSTM	Recurrent layer(s) to capture sequential dependencies in CNN features
Dense layers	Fully connected layers with ReLU activation
Softmax	Output activation producing class probabilities
Loss	Categorical CrossEntropy
Optimizer	Adam
ReduceLROnPlateau	Scheduler to adjust learning rate during training
$\hat{y}$	Predicted labels of the model
Accuracy	Fraction of correctly predicted labels on D_{test}
Confusion matrix	Table summarizing classification performance across classes
Training curves	Plots of loss and accuracy during training/validation

CNN and RNN Malware Classification Pipeline

The experiment followed Algorithm 12 with terms and notations were described in Table 5.28:

1. **Data Preparation**

   At first, the dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ was loaded, where x_i are images and y_i are labels. The preprocessing includes:

   - **Grayscale conversion**
   - **Resizing images** to fixed dimensions (H, W):

$$x_i^{resized} \in \mathbb{R}^{H \times W}$$

- **Normalization** to scale pixel values to $[0, 1]$:

$$x_i^{norm} = \frac{x_i^{resized}}{255}$$

- **One-hot encoding** of labels for C classes:

$$y_i^{onehot} \in \mathbb{R}^C$$

- **Train-validation-test split** to evaluate performance of the model.

2. **CNN Feature Extraction**

   CNN layers extract spatial features from the images:

   - **Convolution:**

$$f_l = \sigma(W_l * f_{l-1} + b_l)$$

     - $*$ denotes the convolution operation,
     - σ is ReLU,
     - W_l are kernels,
     - f_{l-1} is the previous feature map.
   - After that, Pooling and Dropout layers were added.
   - Flattening and reshaping for sequential modeling:

$$\mathbf{z}_i = \text{Flatten}(f_L), \quad \mathbf{S}_i = \text{Reshape}(\mathbf{z}_i) \in \mathbb{R}^{T \times F}$$

3. **BiGRU/LSTM Sequence Modeling**

   Sequential dependencies are captured by LSTM or BiGRU layers:

$$\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, S_{i,t}] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, S_{i,t}] + b_i) \\
o_t &= \sigma(W_o[h_{t-1}, S_{i,t}] + b_o) \\
\tilde{c}_t &= \tanh(W_c[h_{t-1}, S_{i,t}] + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

   For BiGRU, forward and backward sequences are concatenated:

$$h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$$

The fully connected layer and softmax output give class probabilities:

$$\mathbf{o}_i = \text{ReLU}(W_{fc} h_T + b_{fc}), \quad \hat{y}_i = \text{softmax}(\mathbf{o}_i) \in \mathbb{R}^C$$

4. **Model Training** Train in mini-batches of size B for E epochs:

$$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{batch}$$

Evaluate on the validation set and adjust learning rate using scheduler.

5. **Model Evaluation**
   - Predict on test set: $\hat{y}_{test} = M(x_{test})$
   - Compute accuracy:

$$\text{Accuracy} = \frac{1}{N_{test}} \sum_{i=1}^{N_{test}} \mathbf{1}(\hat{y}_i = y_i)$$

   - Generate confusion matrix.
6. **Visualization**
   - Plot training/validation loss and accuracy curves.
   - Display confusion matrix.

Proposition for CNN and RNN Implementation

- **Proposition: Combine CNNs and RNNs into a Hybrid Model**

  Rationale: By merging CNNs for spatial feature extraction and RNNs for sequential relationship learning, the hybrid model was able to learn both the spatial relationships inherent in image data as well as the temporal relationships RNNs are good at learning. This would improve classification performance, especially in use cases where both spatial and sequential patterns were important, e.g., image processing for malware.

- **Proposition: Conduct Hyperparameter Tuning for Optimal Model Performance**

  Rationale: Hyperparameters are tuned systematically (e.g., size of the kernel, number of filters in CNN, and number of units for LSTM) to significantly improve the performance of the model. Optimal tuning of them ensures the models are not underfitting or overfitting and hence provides better accuracy and generalization.

MaleVis Dataset Results

Figure 5.93 is showing the confusion matrix of CNN for MaleVis dataset.

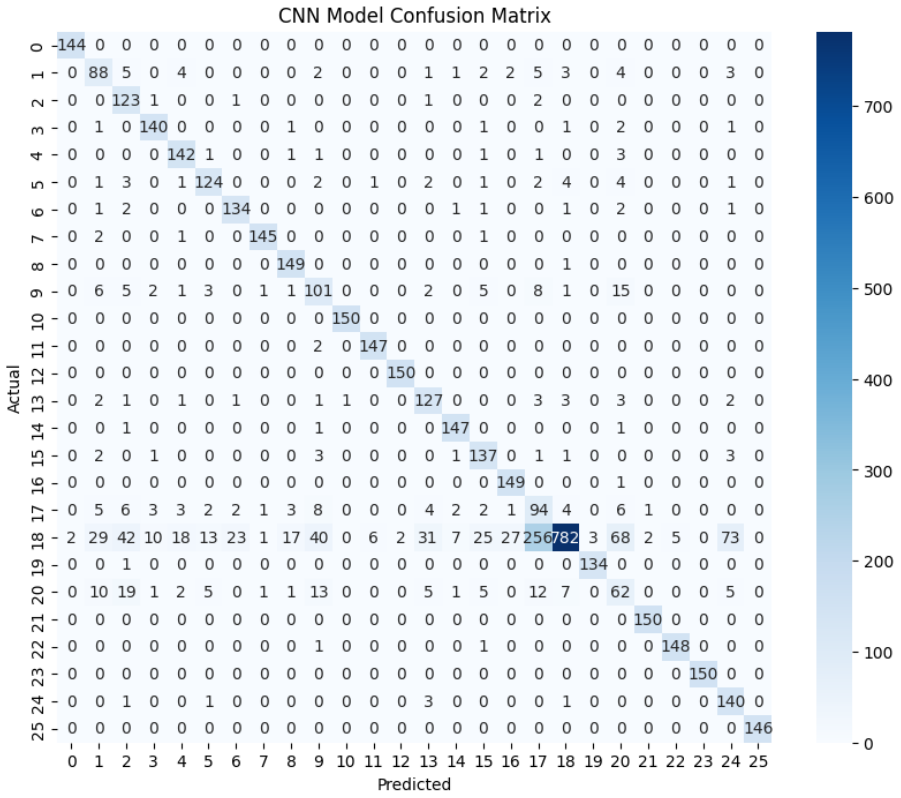


Figure 5.93: MaleVis CNN Confusion Matrix

Figure 5.94 is showing the confusion matrix of RNN for MaleVis dataset.

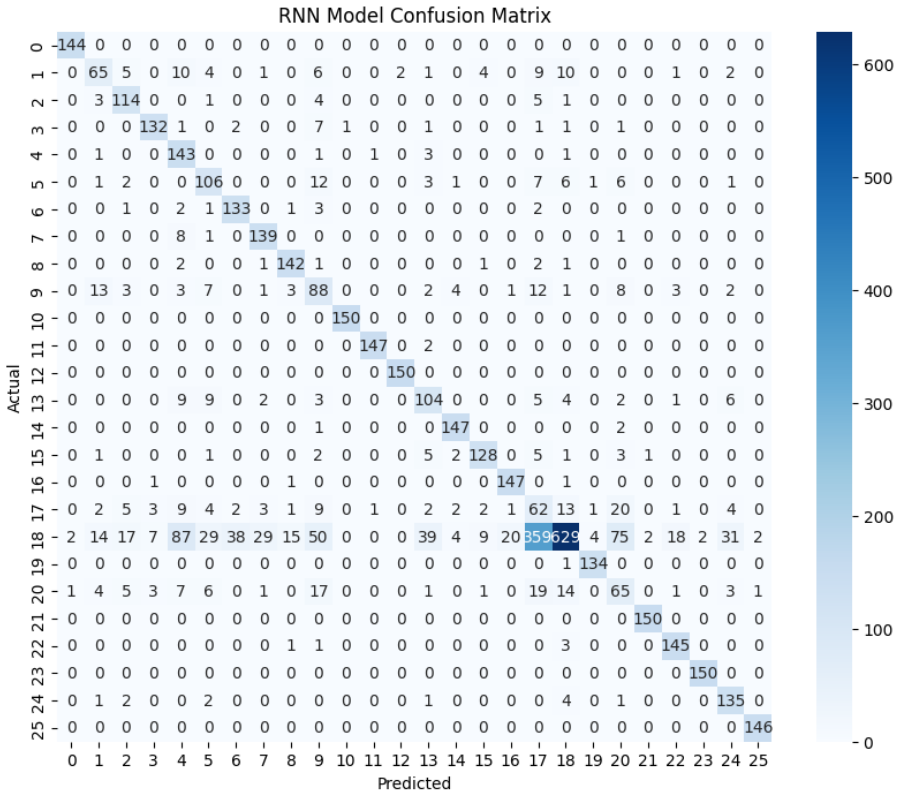


Figure 5.94: MaleVis RNN Confusion Matrix

Virus MNIST Results

Figure 5.95 is showing the confusion matrix of CNN for Virus MNIST dataset.

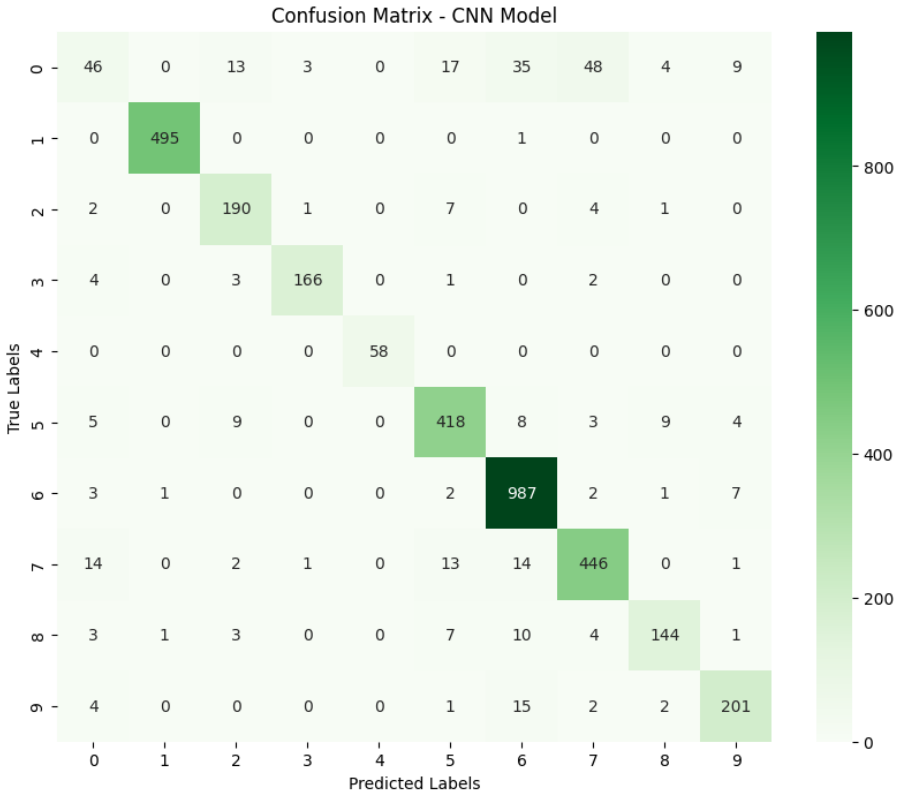


Figure 5.95: Virus MNIST CNN Confusion Matrix

Figure 5.96 is showing the confusion matrix of RNN for Virus MNIST dataset.

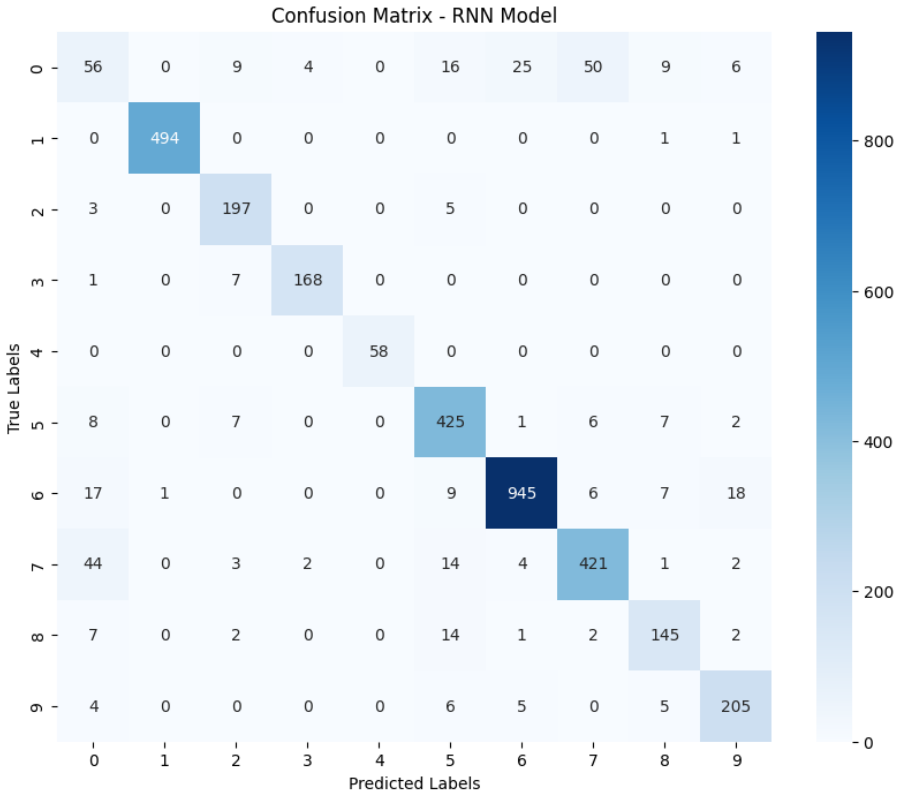


Figure 5.96: Virus MNIST RNN Confusion Matrix

Malimg Dataset Results

Figure 5.97 is showing the confusion matrix of CNN for Malimg dataset.

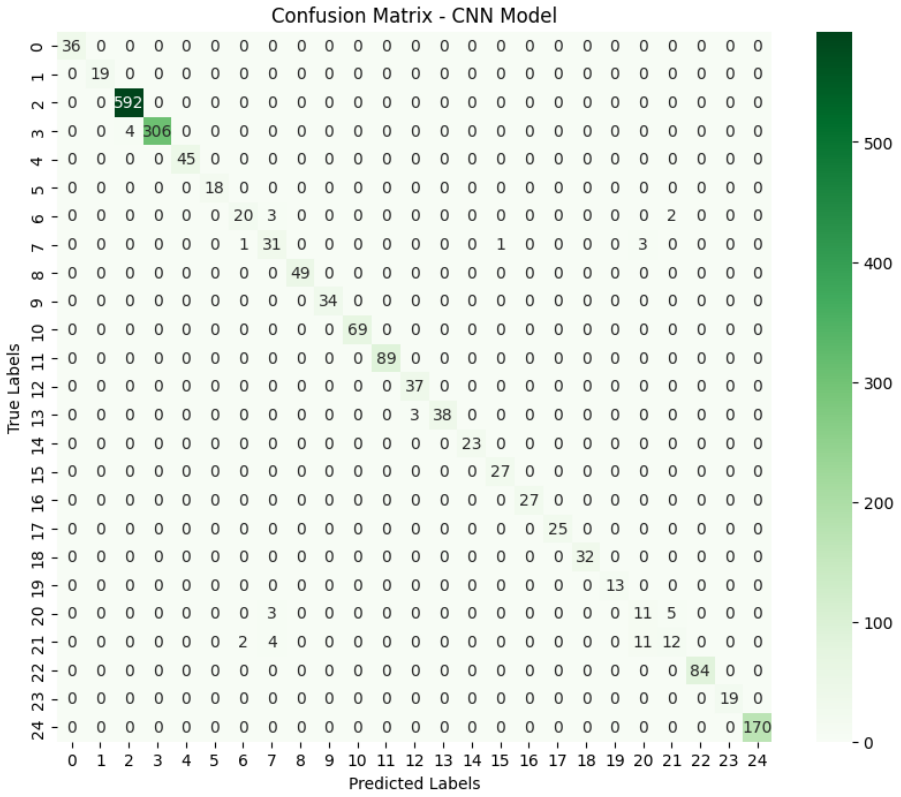


Figure 5.97: Malimg Dataset CNN Confusion Matrix

Figure 5.98 is showing the confusion matrix of RNN for Malimg dataset.

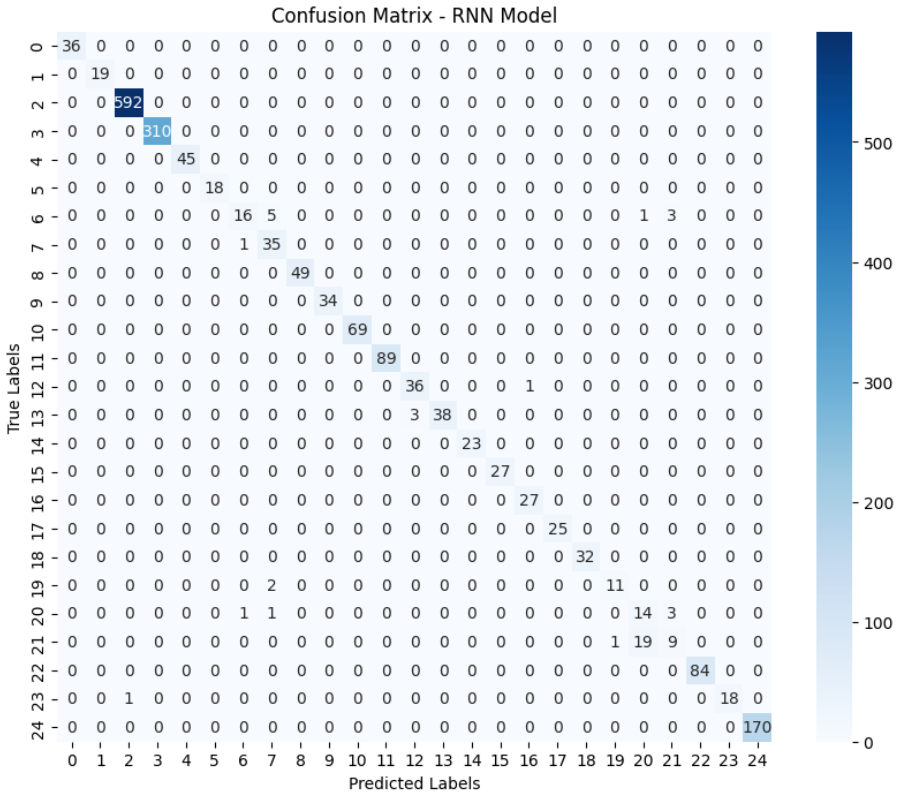


Figure 5.98: Malimg Dataset RNN Confusion Matrix

Table 5.29: CNN Experimental Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	100%	98.26%
Virus MNIST	99.71%	91.25%
MaleVis	99.82%	90.05%

Table 5.30: RNN Experimental Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	99.88%	97.79%
Virus MNIST	99.10%	92.09%
MaleVis	87.41%	83.96 %

Table 5.29 and Table 5.30 are showing the results with all three datasets. All the datasets were trained for 50 epochs with each of the model.

- Both architectures produced exceptionally high accuracies for Malimg dataset, which indicates that the dataset is highly learnable.
- For Virus MNIST, both models exhibited high training accuracy but mild drop in validation accuracy which suggests mild overfitting.
- As MaleVis is a more complex and diverse dataset, CNN achieves a very high training accuracy but the validation accuracy indicates strong overfitting.
- For MaleVis, RNN generalizes better which indicates more robust to intra-class variation.

Critical Analysis of CNN and RNN Implementation

- **Use of LSTM for Image Classification:** The experiment employs Long Short-Term Memory (LSTM), an instance of Recurrent Neural Network (RNN) for classification of images. LSTMs are typically utilized on sequential data like time-series or text but not images. The approach is novel in the sense that it explores how to adapt RNNs to acquire spatial dependencies among images by breaking them down as sequences of pixels.
- **Image Data Treated as Sequences:** Against the usual use of CNNs for extracting spatial patterns, in the experiment images are converted into sequential data so as to feed them into the LSTM model. This defies the traditional teaching of handling images as 2D data and rather the rows of pixels are handled as sequential inputs.
- **Comparison of CNN and RNN Models:**

  The experiment contrasts CNN and RNN with LSTM with each other within the same scenario of image classification, providing knowledge about the performance of these structures in comparison regarding malware image classification.
- **Exploration of Alternative Models for Malware Classification:**

  With the application of LSTMs for classifying malware images, the experiment investigates a different method of handling cybersecurity and

malware detection challenges that could be instrumental in coming up with innovative solutions in this area.

- **Focus on Sequential Learning for Image Data:**

  The experiment points towards the ability of sequential learning models such as LSTMs to identify long-range dependencies in image data, which presents new opportunities for research in computer vision and cybersecurity in the future.

5.2.15 ANN Implementation

Figure 5.99 demonstrates the implementation of an Artificial Neural Network with 400 units at the first hidden layer, 300 units for the second, third and fourth hidden layers. In this experiment, a 1024 x 400 x 300 x 300 x 300 matrix of weights may produce the risk of overfitting which was controlled by incorporating normalization, dropout regularization and optimizer selection.

The experiment follows Algorithm 13 with terms and notation were described in Table 5.31.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense (Dense)                (None, 400)               410000

 dense_1 (Dense)              (None, 300)               120300

 dense_2 (Dense)              (None, 300)               90300

 dense_3 (Dense)              (None, 300)               90300

 dense_4 (Dense)              (None, 100)               30100

 dense_5 (Dense)              (None, 10)                1010

=================================================================
Total params: 742,010
Trainable params: 742,010
Non-trainable params: 0
```

Figure 5.99: ANN Model

Algorithm 13 ANN Training Algorithm

Require: Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, batch size B, epochs E, learning rate η, dropout p

Ensure: Trained ANN model M, accuracy, loss

$x_i \leftarrow x_i / 255$
$y_i \leftarrow$ one-hot(y_i)
Split $D \rightarrow D_{train}, D_{val}$
Flatten images: $z_i \leftarrow$ reshape(x_i)
Initialize weights $\theta = \{W^{(l)}, b^{(l)}\}$
for epoch = 1 to E **do**
  for batch $\{(z_i, y_i)\}_{i=1}^{B}$ in D_{train} **do**
    $h^{(0)} \leftarrow z_i$
    for layer $l = 1$ to $L - 1$ **do**
      $h^{(l)} \leftarrow \text{ReLU}(W^{(l)} h^{(l-1)} + b^{(l)})$
      $h^{(l)} \leftarrow \text{Dropout}(h^{(l)}, p)$
    end for
    $\hat{y} \leftarrow \text{softmax}(W^{(L)} h^{(L-1)} + b^{(L)})$
    $L \leftarrow \text{CrossEntropy}(\hat{y}, y_i)$
    Update $\theta \leftarrow \theta - \eta \nabla_\theta L$
  end for
  Evaluate M on D_{val} (accuracy, loss)
end for
$\hat{y}_i \leftarrow \arg\max_c \hat{y}$
Output M, accuracy, loss

Table 5.31: Terms and Notations for ANN Algorithm

Symbol	Description
D	Dataset consisting of image-label pairs: $D = \{(x_i, y_i)\}_{i=1}^{N}$
x_i	Input image for the i-th sample
y_i	Corresponding label for the i-th sample
B	Mini-batch size for training
E	Number of training epochs
η	Learning rate used in gradient updates
p	Dropout probability for regularization
z_i	Flattened version of input image x_i
$h^{(l)}$	Activation of layer l after applying weights, bias, and non-linearity
$W^{(l)}, b^{(l)}$	Weight matrix and bias vector of layer l
$\hat{y}$	Predicted output vector after softmax activation
L	Loss function (categorical cross-entropy)
θ	Set of all trainable parameters $\{W^{(l)}, b^{(l)}\}$
$\arg\max_c \hat{y}$	Class prediction based on maximum probability
D_{train}, D_{val}	Training and validation splits of the dataset

Table 5.32: ANN Experimental Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	96.18%	92.86%
Virus MNIST	97.80%	90.72%
MaleVis	76.70%	74.45 %

Table 5.32 shows that:

- Malimg dataset achieved high training and validation accuracy after 45 epochs, which indicates that ANN effectively learned discriminative features from the dataset and generalized well.
- Virus MNIST reached higher training accuracy but relatively lower validation accuracy after 20 epochs suggesting minor overfitting.
- MaleVis produced comparatively lower accuracy after 50 epochs, which indicates dataset complexity and high intra-class variability.

5.2.16 GAN Implementation

Generative Adversarial Networks (GANs) were developed by Ian Goodfellow *et al.* (2014) [220], are the architectures for estimating generative models using deep learning with an adversarial process to generate images.

GANs are unsupervised learning algorithms where the 'Generators' learn the distribution from a random input vector to generate synthetic images from the training data and the 'Discriminators' are the binary functions which are responsible to identify the real images from the training data and the fake images produced by the Generator [221]. Figure 5.100 is showing a plot of input images.

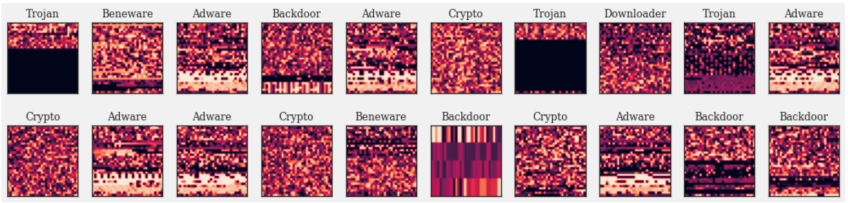


Figure 5.100: Input Images

After defining the generator and discriminator the activation functions have been defined to normalize the images. The forward generator and forward discriminator were defined with the activation functions. The backward discriminator was defined to input the fake images and to combine the gradients of real and fake images. The backward generator was defined for the back-propagation with generator.

Algorithm 14 GAN Training

Require: Dataset $X = \{x_i\}_{i=1}^{N}$, epochs E, batch size B, latent dim d_z
Ensure: Trained Generator G and Discriminator D
  for epoch = 1 to E **do**
    Sample mini-batch $x \subset X$, $|x| = B/2$
    Sample noise $z \sim p_z(z)$
    $\hat{x} \leftarrow G(z)$
    $y_{\text{real}} \leftarrow 1$, $y_{\text{fake}} \leftarrow 0$
    $\mathcal{L}_D \leftarrow \text{BinaryCrossEntropy}(D(x), y_{\text{real}}) + \text{BinaryCrossEntropy}(D(\hat{x}), y_{\text{fake}})$
    $\theta_D \leftarrow \theta_D - \eta \nabla_{\theta_D} \mathcal{L}_D$
    Sample noise $z \sim p_z(z)$
    $y_{\text{mislead}} \leftarrow 1$ {Mislead D}
    $\mathcal{L}_G \leftarrow \text{BinaryCrossEntropy}(D(G(z)), y_{\text{mislead}})$
    $\theta_G \leftarrow \theta_G - \eta \nabla_{\theta_G} \mathcal{L}_G$
  end for
  return G, D

Table 5.33: Terms and Notations for GAN Algorithm

Symbol	Meaning
$X = \{x_i\}_{i=1}^{N}$	Real dataset of images
x_i	Real image sample, $x_i \in \mathbb{R}^{H \times W \times C}$
y_i	Label of x_i
$z \sim p_z(z)$	Latent noise vector, $z \in \mathbb{R}^{d_z}$
d_z	Dimension of latent space
$G(z;\theta_G)$	Generator network producing fake images
$D(x;\theta_D)$	Discriminator network predicting real/fake
$\hat{x} = G(z)$	Generated (fake) image
B	Mini-batch size
E	Number of epochs
η	Learning rate
$\mathcal{L}_D$	Discriminator loss
$\mathcal{L}_G$	Generator loss (mislead D)
$D(x) \approx 1$	D predicts real image correctly
$D(\hat{x}) \approx 0$	D predicts fake image correctly
Mislead D	Training G to fool D so $D(G(z)) \approx 1$

Virus MNIST Results

After preprocessing and splitting the data the model was trained for 100 epochs. Figure 5.101 is showing a sample of generated images.

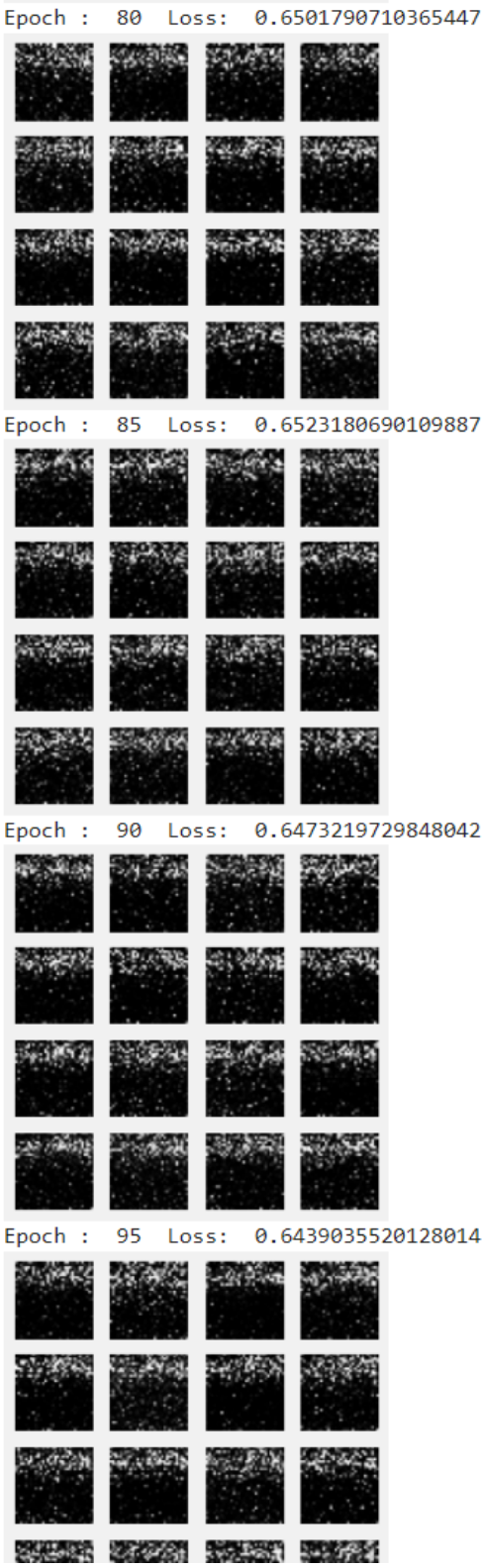


Figure 5.101: Generated Images of Virus MNIST dataset

MaleVis Dataset Images



Figure 5.102: Real and Fake images of MaleVis dataset

Malimg Dataset Images

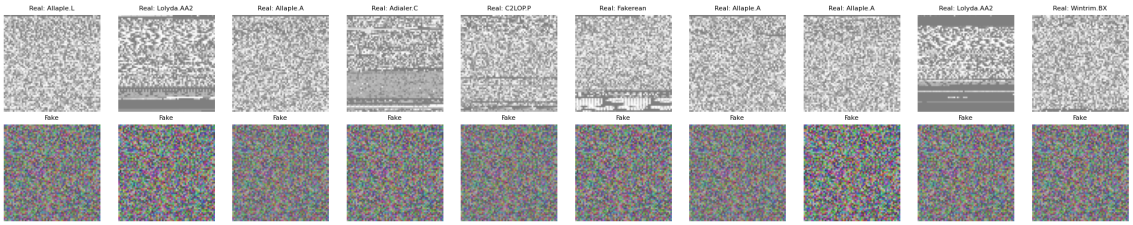


Figure 5.103: Grid of Real and Fake images of Malimg Dataset

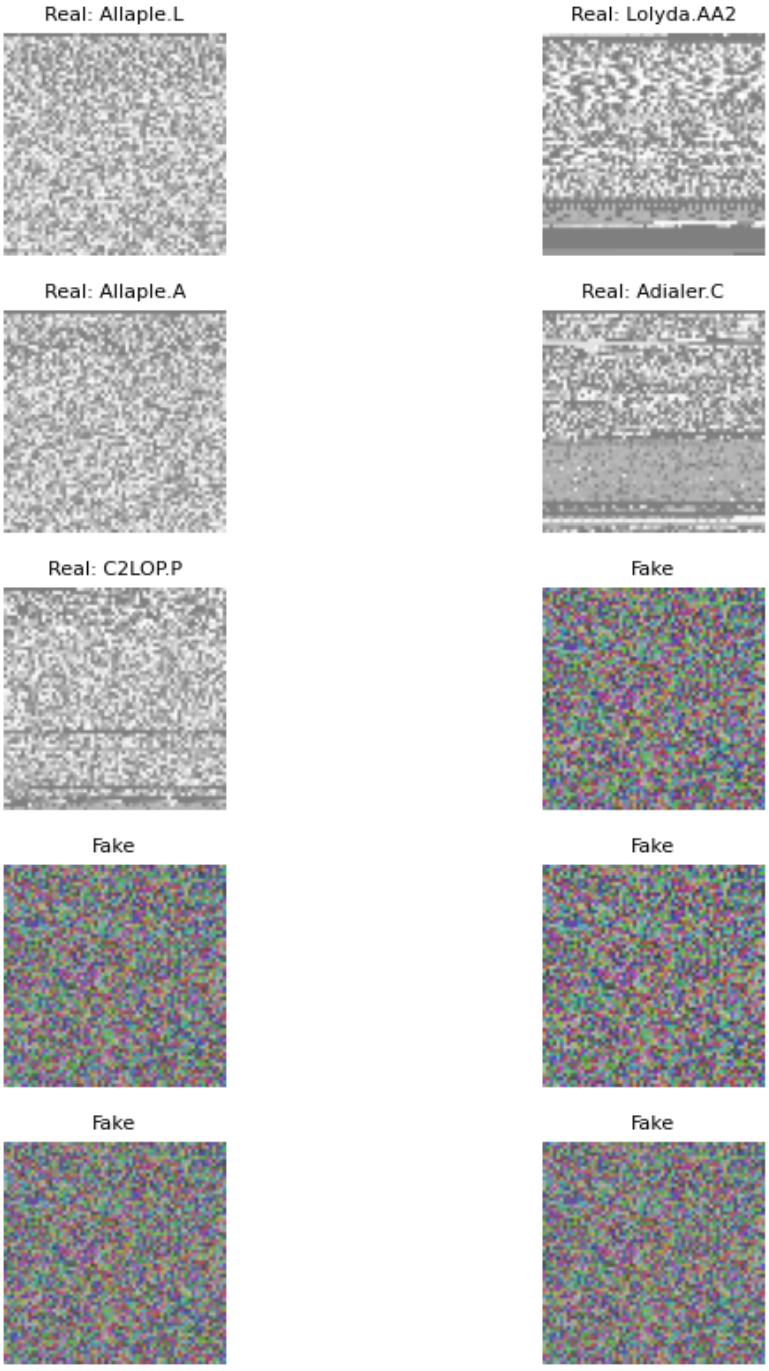


Figure 5.104: Real and Fake images of Malimg dataset

Proposition for GAN Implementation

1. **Synthetic Malware Generation**

- **Proposition:** This experiment demonstrates the generation of synthetic malware samples for the "Trojan" category using a custom Generative Adversarial Network (GAN).
- **Rationale:** GANs are the most convenient techniques to augment dataset with synthetic samples [222]. This can be used to improve data availability and diversity without compromising security guidelines as availability of diverse and representative malware datasets is critical in cybersecurity field.

2. **Dataset Augmentation for Classification**
   - **Proposition:** These can improve the performance of a classification model if the original dataset is integrated with the synthetic samples that the GAN has generated.
   - **Rationale:** Data augmentation of the dataset using high-quality synthesized images. It can mitigate data imbalance and scarcity, helping to improve the model performance.

Steps of GAN Implementation

The steps of this experiment were constructed according to Algorithm 14 with terms and notations were described in Table 5.33:

- **Model Architecture:**
  1. The GAN architecture consists of a generator and a discriminator.
  2. Generator and discriminator have respective weight matrices (theta) initialized with random values.
  3. The activation functions used are Leaky ReLU (lrelu), hyperbolic tangent (tanh), and sigmoid.
  4. While the generator tries to generate realistic images, the discriminator tries to distinguish between real and generated images.

- **Training Loop:**
  1. The training loop iterates over epochs and batches.
  2. The discriminator should be trained, at each batch, to discriminate the real from whereas generated images, while the generator is trained to generate images that fool the discriminator.
  3. Calculation and storing the loss functions of the generator and discriminator.

- **Backpropagation and Optimization:**
    1. Backward passes are implemented both for the discriminator and generator using backpropagation.
    2. The discriminator calculates the gradients for real and fake images, respectively, followed by an update of the model parameters.
- **Learning Rate Schedule:**
    1. Learning rate decreases over epochs to stabilize and improve the training performance.
    2. This prevents oscillation or divergence that may occur in the model during training.

Critical Analysis of GAN Implementation

- **Malware Image Generation:** GANs are usually used for generative tasks; in this experiment, it's relatively new because it's being used to generate malware images. This is very useful for malware detection models, helping to create diverse samples.
- **Specific Malware Categories:** The code emphasizes generating images for certain malware categories such as 'Trojan', 'Adware', etc., and may have some practical implications when trying to understand and detect diversified new cyber threats.
- **Learning Rate Decay:** This experiment also utilizes a learning rate decay strategy during training. This can be important for the stability of GAN training in finding better convergence over time.

5.2.17 Implementation with Conditional GAN (cGAN)

Conditional GAN is more handy and specific to generate images. By using Conditional GAN the generator can be assigned to produce specific numbers of images with specified labels or classes (if needed). A flowchart of cGAN was shown in Figure 5.105.

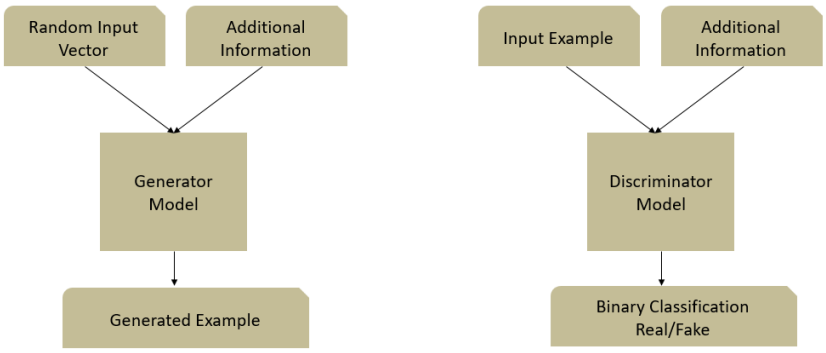


Figure 5.105: Flowchart of Conditional GAN

In this experiment the generator and discriminator have been defined as neural network architectures. The labels were assigned as additional information. The parameters of the discriminator and generator were shown in Figure 5.106 and Figure 5.107.

```
(label_emb): Embedding(10, 10)
(model): Sequential(
  (0): Linear(in_features=1034, out_features=2048, bias=True)
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
  (2): Dropout(p=0.3, inplace=False)
  (3): Linear(in_features=2048, out_features=512, bias=True)
  (4): LeakyReLU(negative_slope=0.2, inplace=True)
  (5): Dropout(p=0.3, inplace=False)
  (6): Linear(in_features=512, out_features=256, bias=True)
  (7): LeakyReLU(negative_slope=0.2, inplace=True)
  (8): Dropout(p=0.3, inplace=False)
  (9): Linear(in_features=256, out_features=1, bias=True)
  (10): Sigmoid()
)
```

Figure 5.106: Parameters of Discriminator

```
(label_emb): Embedding(10, 10)
(model): Sequential(
  (0): Linear(in_features=110, out_features=256, bias=True)
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
  (2): Linear(in_features=256, out_features=512, bias=True)
  (3): LeakyReLU(negative_slope=0.2, inplace=True)
  (4): Linear(in_features=512, out_features=2048, bias=True)
  (5): LeakyReLU(negative_slope=0.2, inplace=True)
  (6): Linear(in_features=2048, out_features=1024, bias=True)
  (7): Tanh()
)
```

Figure 5.107: Parameters of Generator

The generator was trained to generate the images and the discriminator was trained with the real images and the fake images. Optimizer 'Adam' was used to update the whole network weight while training.

The experiment followed the steps of Algorithm 15 with terms and notations were described in Table 5.34.

Algorithm 15 Conditional GAN Training

Require: Dataset $X = \{(x_i, y_i)\}_{i=1}^{N}$, epochs E, batch size B, latent dim d_z, classes C

Ensure: Trained Generator G and Discriminator D

for epoch = 1 to E **do**

for mini-batch $(x, y) \subset X$ **do**

$z \sim p_z(z)$

$\hat{x} \leftarrow G(z, y)$

$y_{\text{real}} \leftarrow 1, y_{\text{fake}} \leftarrow 0$

$\mathcal{L}_D \leftarrow \text{BCE}(D(x, y), y_{\text{real}}) + \text{BCE}(D(\hat{x}, y), y_{\text{fake}})$

$\theta_D \leftarrow \theta_D - \eta \nabla_{\theta_D} \mathcal{L}_D$

$z \sim p_z(z)$

$\hat{x} \leftarrow G(z, y)$

$y_{\text{mislead}} \leftarrow 1$

$\mathcal{L}_G \leftarrow \text{BCE}(D(\hat{x}, y), y_{\text{mislead}})$

$\theta_G \leftarrow \theta_G - \eta \nabla_{\theta_G} \mathcal{L}_G$

end for

end for

return G, D

Table 5.34: Terms and Notations for Conditional GAN

Symbol/Term	Description
$X = \{(x_i, y_i)\}_{i=1}^{N}$	Dataset of N samples with data x_i and labels y_i
G	Generator network
D	Discriminator network
z	Latent noise vector sampled from prior $p_z(z)$
y_i	Class label associated with sample x_i
C	Total number of classes
$\hat{x} = G(z, y)$	Fake/generated sample conditioned on label y
y_{real}	Label for real samples (usually 1)
y_{fake}	Label for fake/generated samples (usually 0)
y_{mislead}	Target label for generator to fool discriminator (usually 1)
$\mathcal{L}_D$	Discriminator loss
$\mathcal{L}_G$	Generator loss
θ_D	Parameters of the discriminator
θ_G	Parameters of the generator
η	Learning rate
B	Mini-batch size
E	Number of epochs
d_z	Dimension of the latent noise vector z

Virus MNIST Results

The model was trained for 10 epochs to produce the sample images (Figure 5.108).

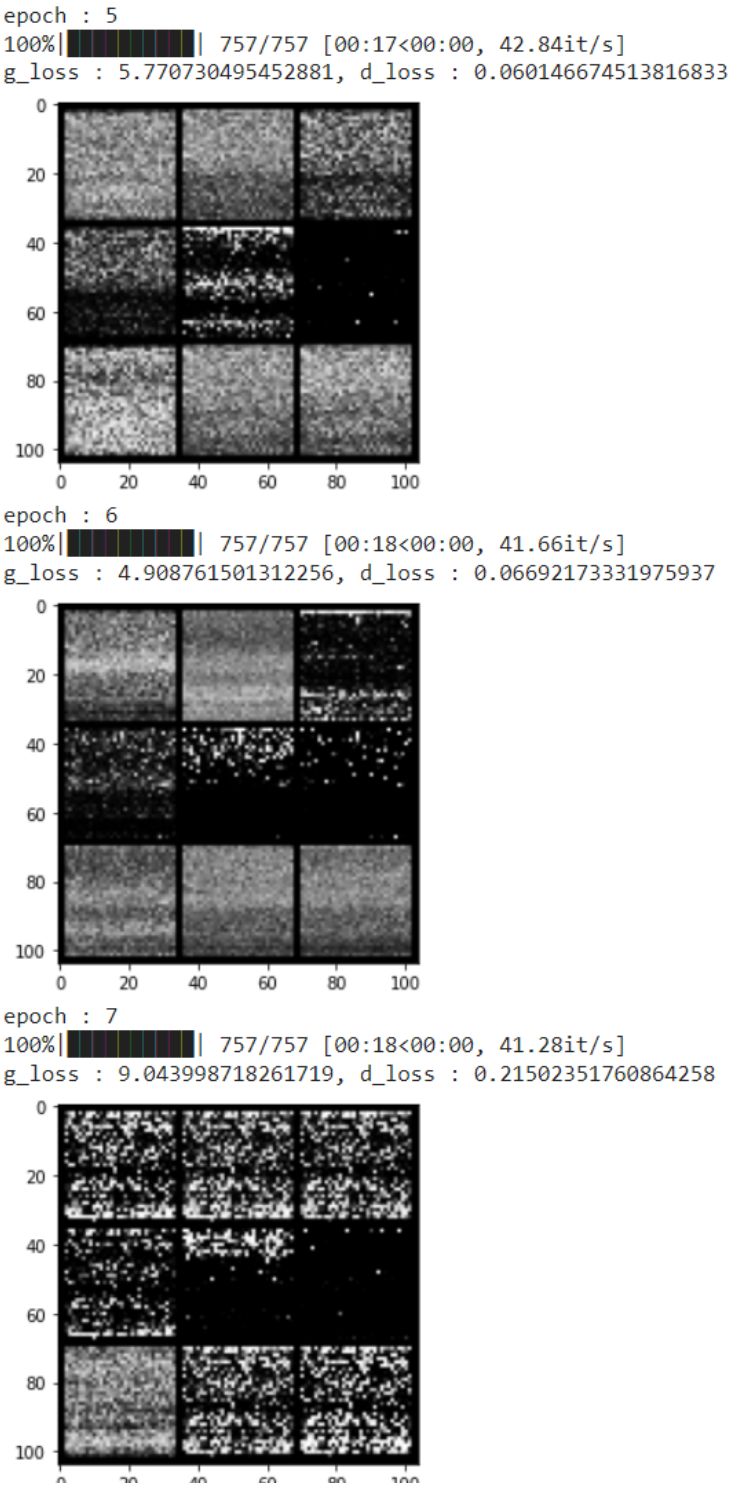


Figure 5.108: Sample Images of Virus MNIST

The model was tested to generate the grid of the sample images (Figure

5.109).

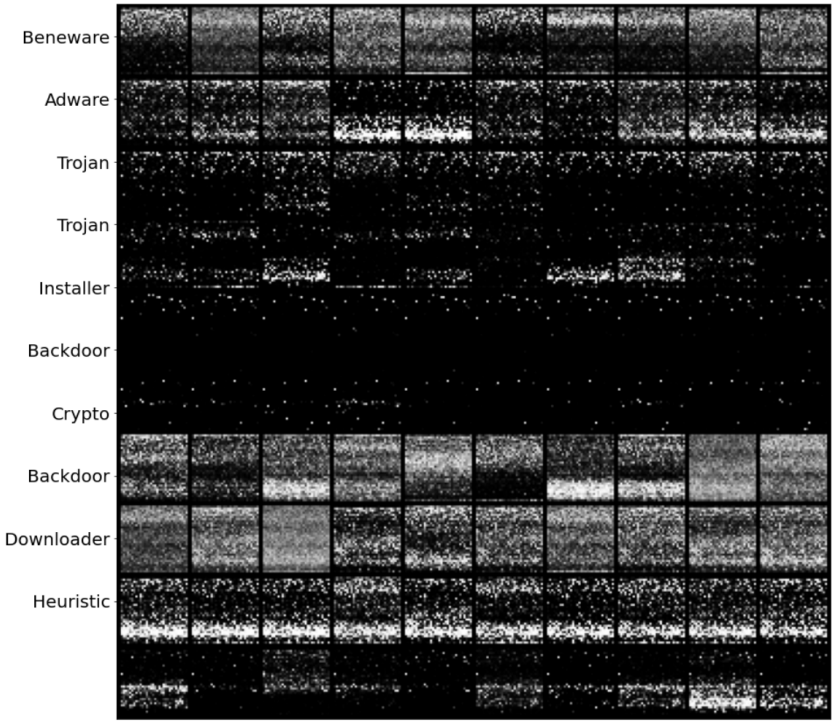


Figure 5.109: Model Testing of Virus MNIST dataset

MaleVis Dataset Conditional GAN Images

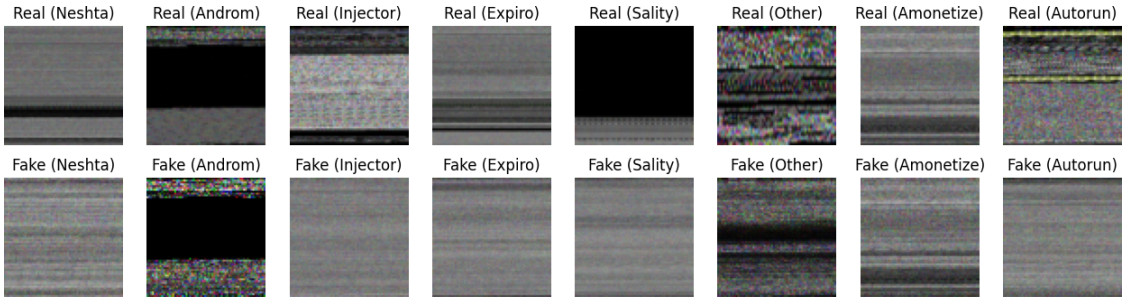


Figure 5.110: MaleVis Generated Images

Malimg Dataset Conditional GAN Images

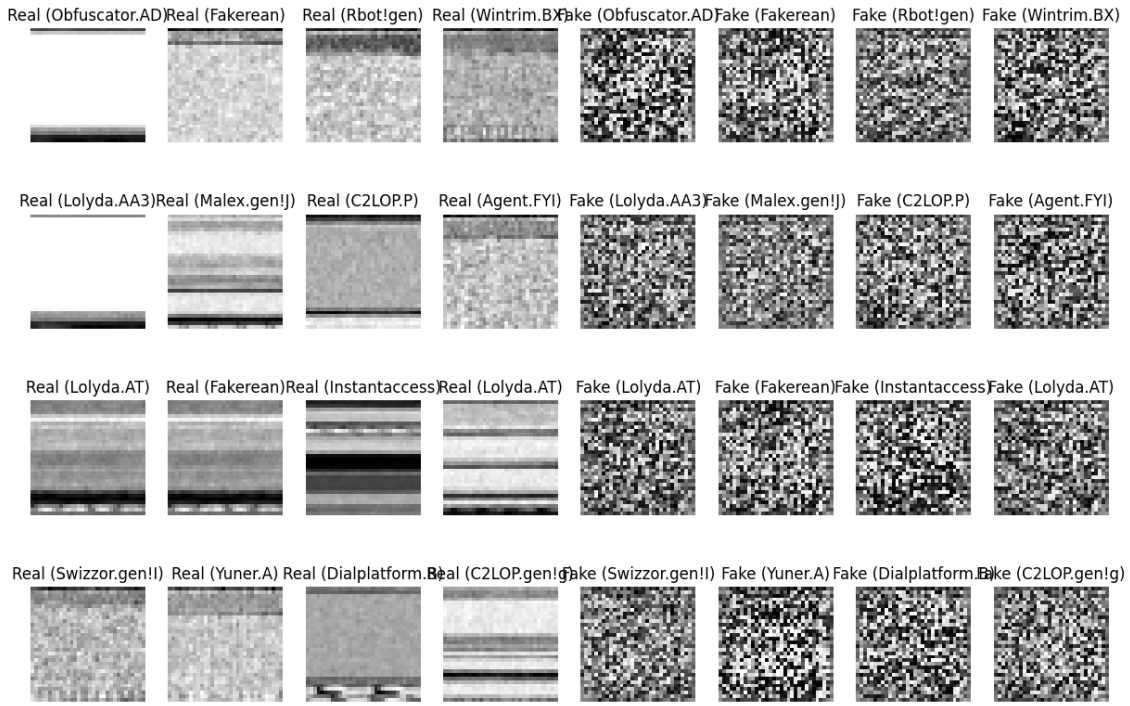


Figure 5.111: Malimg Generated Images

Propositions for Conditional GAN

- **Statement:**
  1. The Conditional Generative Adversarial Network (cGAN) methodology was implemented to generate synthetic malware variants [223]. The generated malware samples address two major challenges: class imbalance and scarcity of data within malware datasets.
  2. These will increase the malware image dataset and enhance the diversity and categories of malware.
- **Rationale:**
  1. **Challenge of Class Imbalance:** Augmenting the dataset with synthetic samples addresses class imbalance.

2. **Benefits of Conditional GANs:** Conditional GANs allow for label conditioned image generation [224], making them suitable for synthesizing malware samples specific to each category.
3. **Enhanced Data Diversity:** Traditional augmentation techniques (e.g., rotations, flips) provide limited diversity and do not capture the intricate variations in malware patterns. A cGAN-generated dataset introduces greater variability while preserving category specific features.

Critical Analysis of Conditional GAN

- **Conditional GAN with Labels:**
  1. The cGAN has an architecture in which both the generator and discriminator are conditioned not only by random noise but also class labels.
  2. This enables the generation of images, conditioned on specific classes and giving the user more control over the generated samples.
- **Label Embedding:**
  1. Label embedding was used both in the discriminator and generator models, are a distinguishing feature.
  2. Label embedding lets the models encode information about the class of a generated or real image; therefore, it enables more meaningful and diverse generations of images.
- **Conditional Image Generation:**
  1. The conditional generation of images allows the creation of images to be more specific to the given labels.
  2. That will be of particular help where there is the requirement for image generation belonging to certain categories or classes.

5.2.18 Visualization with t-SNE plots

To understand the structure and distribution of datasets, t-distributed Stochastic Neighbor Embedding (t-SNE) is employed as a powerful visualization technique. Here, t-SNE was chosen because of its strong ability to preserve local structure and visualize high-dimensional data.

Visualization with Virus MNIST dataset

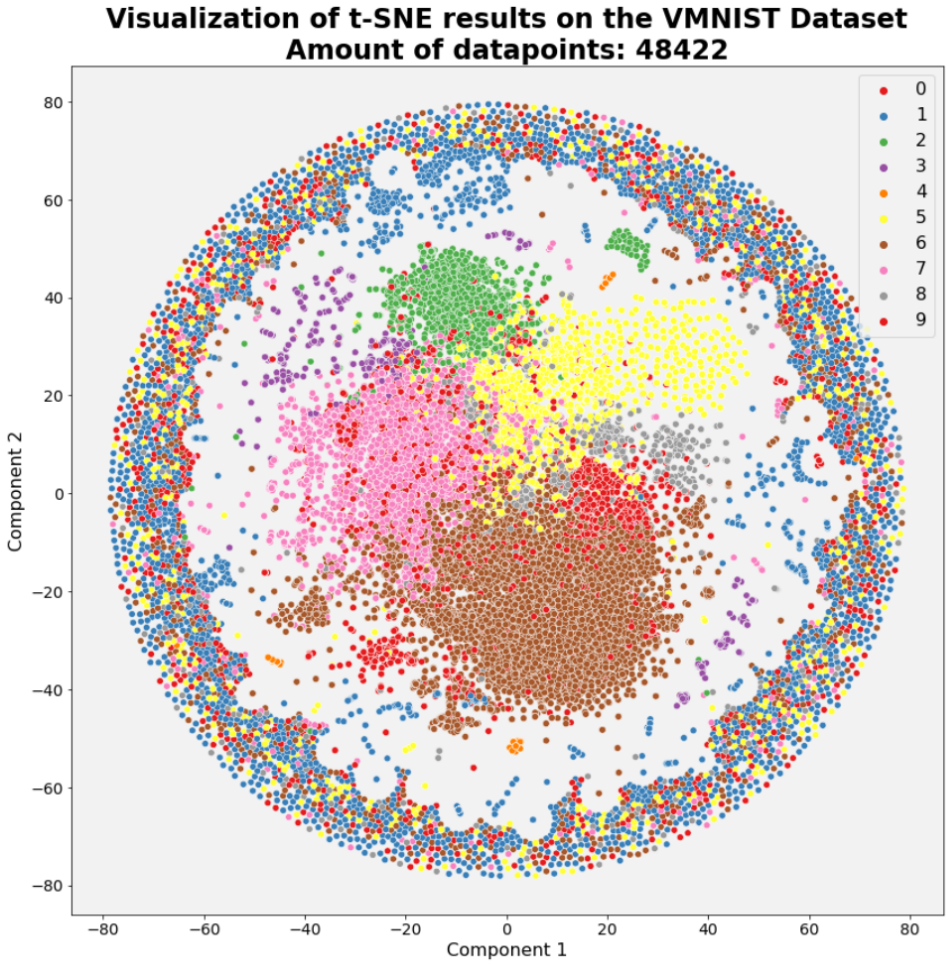


Figure 5.112: Visualization with Virus MNIST

Analysis of t-SNE Visualization

In this visualization the Virus MNIST dataset is presented with a two dimensional t-SNE plot. The high-dimensional data points have been projected into two components: **Component 1** and **Component 2**.

- The training dataset of Virus MNIST contains 48422 number of data points, spreading across the plane with clusters of different classes.

- Each colour presents a distinct class (from 0 to 9), and the points correspond to individual images.

1. **Clusters of Data:**
    - Class 2 (green) forms a distinct cluster at the top center.
    - Class 0 (light red) and Class 3 (purple) form clusters around the borders.
    - Other Classes like 4 (orange), 1 (blue) have their own distinct clusters.
2. **Inter-cluster Relationships:**
    - Couple of classes such as (Class 7 (pink) and 5 (yellow)) have shown considerable overlap which indicates that those classes have visual similarities in their high-dimensional representation.
    - Potential visual similarities reflected on Classes like 6 (brown), 8 (grey) and 9 (dark red) as they overlap slightly.
3. **Distribution of Data:**
    - In the high-dimensional space the local neighborhood structures are preserved in the 2D plane as data points are distributed over the space.
    - The outer ring is representing the outliers or points that the model finds difficult classify distinctly that's the reason the outer ring is less clear or ambiguous.
4. **Global Structure and Local Structure:**
    - For preserving local structure (how close points are to their nearest neighbors) t-SNE is a better option which has been shown in the compact clusters.
    - But t-SNE is not always good to preserve global structure that is evident that some clusters are squeezed or separated in arbitrary directions.

Visualization with Malimg dataset

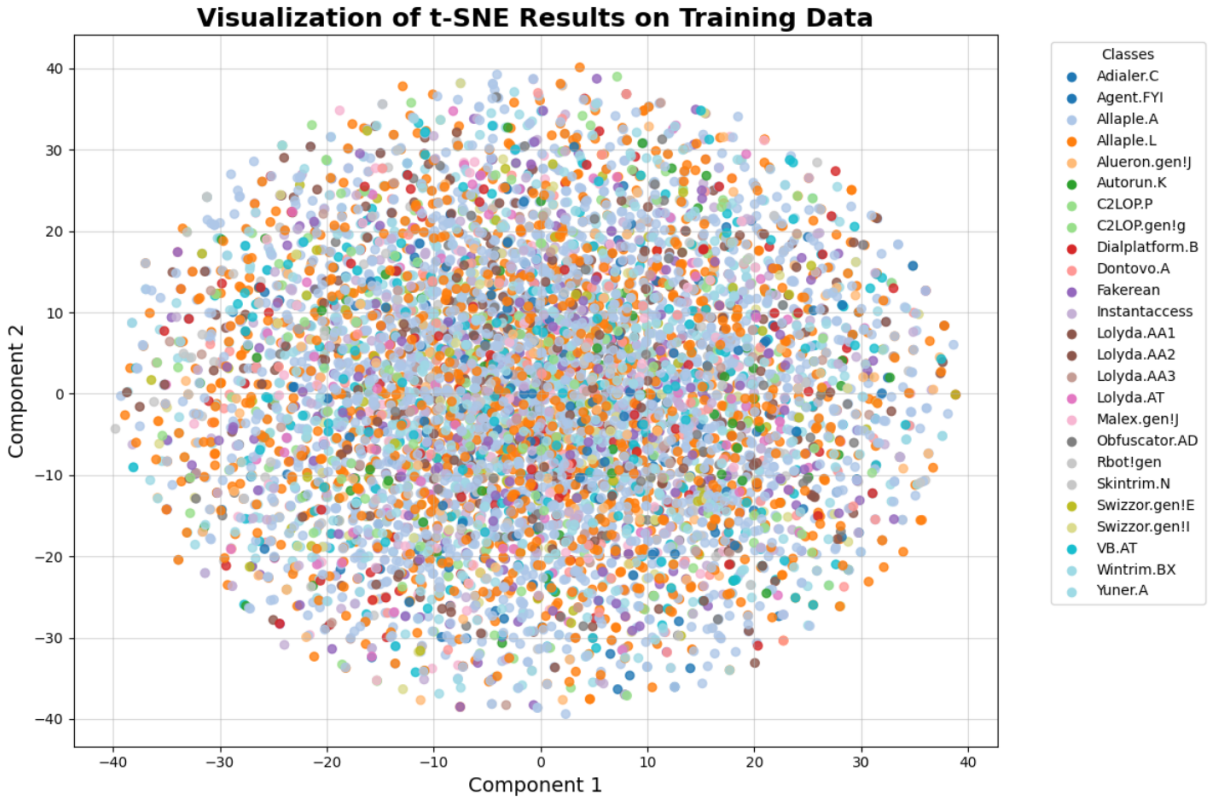


Figure 5.113: Visualization with Malimg dataset

Analysis of t-SNE Visualization for Malimg

1. **Lolyda.AA1, Lolyda.AA2, Lolyda.AA3:**
   - These classes are highly overlapping, meaning that their feature representations are not easily separable.
   - This is expected since these classes are part of the same Lolyda family, implying that they may share some common characteristics or have similarities in their malware behavior or visual patterns.
   - The overlapping nature may result in a higher rate of misclassification among these subfamilies.
2. **Autorun.K and C2LOP.gen!g:**

- These classes are highly spread out in the plot, indicating a high degree of intra-class variability.
- This may be due to the fact that samples in these classes tend to behave differently or look differently, and it will not be easy for any clustering technique to fit them tightly.
- For instance, Autorun.K consists of various polymorphic samples belonging to the same malware family.

3. **Adialer.C:**
    - In this case, this class looks a little denser compared to others but still overlaps with the classes surrounding it.
    - This relatively small size might indicate low entropy in the samples for this family, which would make it slightly more distinguishable than others.

4. **Swizzor.gen!E and Swizzor.gen!I:**
    - These classes are within the same Swizzor family. Like in Lolyda subfamilies, there are a great deal of overlap.
    - It could mean that the features for these classes do not make much differentiation between the two variants.

5. **Rbot!gen and Obfuscator.AD**
    - These classes seem spread all over the plot and don't appear to have a tight or sharp cluster.
    - This scattering could imply that samples belonging to this class have many commonalities with more than one of other classes. Perhaps due to the functionalities overlapped and generic malware behaviors.

6. **Wintrim.BX and Fakerean:**
    - These families exhibit moderate overlap with surrounding classes.
    - This behavior might represent an intermediate class of separability, capturing something unique related to these families.

7. **Lolyda.AT**
    - This is slightly more diffused than Lolyda.AA subfamily indicating a bit greater variability across this class's samples.
    - The spreading across could further refer to general broader behavioral differences, or in simpler terms it possibly indicates weaker correlations with other representatives of Lolyda.

Visualization with MaleVis dataset

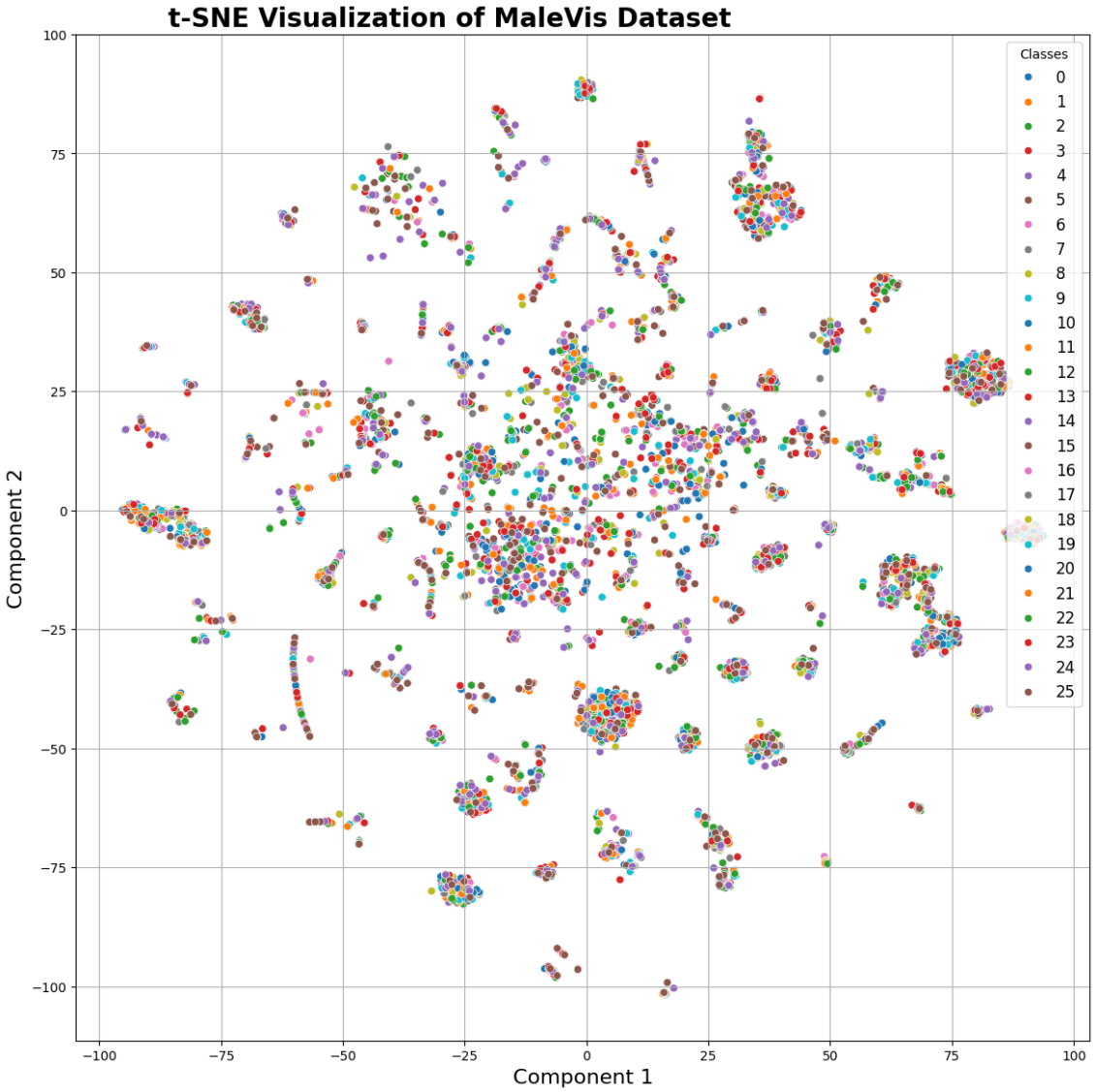


Figure 5.114: Visualization with MaleVis dataset

The labels are:

1. 0: 'Adposhel',
2. 1: 'Agent',
3. 2: 'Allaple',
4. 3: 'Amonetize',

5. 4: 'Androm',
6. 5: 'Autorun',
7. 6: 'BrowseFox',
8. 7: 'Dinwod',
9. 8: 'Elex',
10. 9: 'Expiro',
11. 10: 'Fasong',
12. 11: 'HackKMS',
13. 12: 'Hlux',
14. 13: 'Injector',
15. 14: 'InstallCore',
16. 15: 'MultiPlug',
17. 16: 'Neoreklami',
18. 17: 'Neshta',
19. 18: 'Other',
20. 19: 'Regrun',
21. 20: 'Sality',
22. 21: 'Snarasite',
23. 22: 'Stantinko',
24. 23: 'VBA',
25. 24: 'VBKrypt',
26. 25: 'Vilsel'

Analysis of t-SNE Visualization for MaleVis

1. **Cluster Sizes and Distribution:**
    - Classes like Allaple, Sality, and Agent have larger, more dense clusters. This shows that these malware families have more samples in the dataset and are probably much more homogeneous with regard to the feature representation.
    - Classes like Vilsel, Neshta, and HackKMS appear as small, disjoint groups, meaning that these classes probably have fewer samples or possibly much more varied characteristics.

2. **Overlapping Clusters:**

   Overlaps are observable between classes such as VBKrypt, Injector, MultiPlug, and many others. This suggests overlap in feature representations within the reduced dimensional space. This may be due to:
   - The structural patterns in their behavior being similar.
   - There is overlap in the visual patterns of the malware images.

3. **Class Other:** This class, being a mixed category, is widely spread over the plot, overlapping with several other clusters. The wide spread depicts its heterogeneous nature.
4. **Cluster Shapes and Patterns:**
   - Some classes, such as BrowseFox or Regrun, are more or less well-rounded, circular clusters, indicating significant intra-class similarity.
   - Other classes, such as Autorun or Stantinko, tend to be more elongated or fragmented, suggesting either variation within the class or overlap with other classes.

Critical Analysis for Visualization with t-SNE

1. **Class Distribution Visualisation:**
   - The purpose is to understand the distribution of classes in the dataset and identify the class imbalances.
   - This will be crucial for selecting appropriate evaluation metrics.
2. **t-SNE Visualization:**
   - The purpose of a t-SNE plot is to visualize the dataset in two dimensions using t-SNE.
   - This will provide insights into the structure and relationships between data points.
3. **Scatterplot of t-SNE Results:**
   - Here the purpose is to visualize the t-SNE results in a scatterplot.
   - It offers a visual representation of the data distribution in a reduced-dimensional space.

5.3 Conclusions

The deep learning experiments conducted in this study demonstrate the effectiveness of deep learning based techniques for malware classification using the malware image datasets. The model was trained using a structured approach, including data preprocessing, exploratory data analysis and hyperparameter tuning to optimize performance. Techniques such as learning rate scheduling, early stopping and data augmentation contributed to improved generalization and classification accuracy. The evaluation metrics, including accuracy and confusion matrix analysis, provided insights into the model's strengths and areas for improvement. These results confirm the potential of deep learning in cybersecurity applications, particularly for malware detection and classification.

In this chapter we propose, different applications of the preexisted deep learning models on malware image datasets to improve accuracy from the literature survey. The experiments have achieved very good accuracy and some combinations of models performed very well to detect and classify malware samples. Table 5.35 is showing comparison of our methods with the literature.

Table 5.35: Experimental Results with Accuracy

Results			
Experiment	Malimg	Virus MNIST	MaleVis
Thosar et al., 2021 [225] **CNN**	X	93.45%	X
Wong et al., 2021 [226] **ShuffleNet and DenseNet 201 with Transfer Learning**	Training Accuracy: 99.14%, Test Accuracy: 98.87%,	Training Accuracy: 86.36%, Test Accuracy: 85.79%	Training Accuracy: 95.01%, Test Accuracy: 93.91%
Sharma et al., 2020 [227] **CNN**	Test accuracy: 97.5%	X	X
Pant and Bista, 2021 [228]	**CNN:** 98.2%, **VGG-16:** 88.40%, **ResNet-18:** 90.21%	X	X
Combination of CNN and RNN	**Training Accuracy: 100%, Test Accuracy: 98.05%**	**Training Accuracy: 93.18%, Test Accuracy: 92.72%**	**Training Accuracy: 96.70%, Test Accuracy: 94.51%**
CNN	**Training Accuracy: 100%, Test Accuracy: 98.26%**	Training Accuracy: 99.71%, Test Accuracy: 91.25%	Training Accuracy: 99.82%, Test Accuracy: 90.05%
ResNet-50 Attention Network	**Training Accuracy: 98.82%, Test Accuracy: 97.91%**	**Training Accuracy: 99.99% Test Accuracy: 92.35%**	Training Accuracy: 99.67%, Test Accuracy: 84.63%

6 Malware Analysis using Deep Learning with Continual Learning

For the dynamic fields like (Cyber Security and IoT), network data can be highly volatile. Generally traditional deep learning models lose information (while in batch learning mode) learned from older tasks when retrained the model with new malware data, leading to "catastrophic forgetting" [122]. The root cause behind catastrophic forgetting is the changing of weights during training of new malware data. Because this process certainly modifies weights that were added during the earlier malware detection task. To overcome catastrophic forgetting, Continual Learning needs to be incorporated to enable the models to adapt to new data while preserving older knowledge, which is crucial for applications that need to evolve with incoming information.

Based on the State-of-the-art provided in Subsection 3.4, this chapter is trying to achieve following research gaps:

- **Limited Focus on Image-Level Malware Representations:** Most of the existing approaches utilize raw byte strings, API traces, or hand-designed features with minimal exploration of malware image-based continual learning. Our method fills this particular gap uniquely through the utilization of deep image representations but the field is not fully explored in overall literature.
- **Lack of Unified Framework Handling:** The majority of studies focus on a single continual learning scenario (e.g., class-incremental or task-incremental) without suggesting a generalizable method that possesses the ability to dynamically switch or combine continual learning regimes depending on data characteristics.

- **Limited Scalability Across Diverse Malware Datasets:** Existing methods are commonly validated against small or domain-specific datasets or a specific malware type (e.g., ransomware). Exhaustive benchmarking across various, visually diverse malware image datasets has been a poorly explored research avenue where our work is an enriching contribution.
- **Minimal Use of Adaptive Replay Strategies for Deep Malware Models:** Most replay-based methods reuse exemplars or latent representations, while image-level replay (used in our method with U-Net and ResNet) offers greater fidelity but is not widely applied. Adaptive, quality-sensitive replay mechanisms designed specifically for deep malware imaging may be discovered through research.
- **Lack of Explainability in Continual Learning Decisions:**

  Current models behave more like black boxes, making little effort to describe why certain variants of malware are remembered and forgotten. Explainable continuous learning especially for security-critical use cases is an emerging research gap.

Our approach substantially improves the solutions for the major loopholes of malware detection in continual learning. Unlike traditional methods that depend on raw bytes or API calls, our research is the first to use image-level features of malware and leverage powerful deep models like ResNet and U-Net to extract powerful spatial and textural cues from malware images. This not only increases the modality but also enables the system to learn fine-grained visual features, such as structural patterns and layout behaviors, which are generally neglected in feature-level representations. Furthermore, our experiments are based on realistic malware evolution scenarios rather than artificially balanced task splits. Instead, we mimic real-world scenarios where new malware families suddenly appear and in imbalanced distributions. This real-world design is further supported by our use of multiple, heterogenous datasets such as Malimg, MaleVis and Virus MNIST to ensure good scalability and generalizability.

6.1 Malware Detection with Continual Learning

Deep learning with continual learning (CL) is an advanced AI approach that enables models to learn new tasks while retaining previously acquired knowledge, preventing catastrophic forgetting [229]. This is crucial for real-world applications where data evolves over time, such as malware detection, cybersecurity, and autonomous systems. In cybersecurity and malware classification, continual learning can be highly effective. For example, when new malware variants emerge, a CL-enabled model can adapt without forgetting older threats, ensuring robust and up-to-date threat detection.

6.1.1 GANs for Data Augmentation with Continual Learning

1. **GAN Architecture Definition:**
   - **Generator:** Generator is a neural network model which generates synthetic samples by taking random noise vectors or latent variable as input. The latent vector comes from a latent space which consists of compressed feature space where each each random noise represents a potential feature [230].

     In this experiment also, the generator learns to produce realistic malware-like images by using random noises.
   - **Discriminator:** Discriminator is another neural network which is trained to distinguish between real images and fake images. In this experiment also the discriminator ensures that the generator produces malware-like images that closely resemble the real images. It evaluates the authenticity of images by taking the input and produces a probability to indicate whether the image is real or fake.
2. **GAN Training:** A function was defined to train the GAN by alternating between training the generator and the discriminator. It uses adversarial loss to train both components. The generator aims to minimize this loss while the discriminator aims to maximize it.

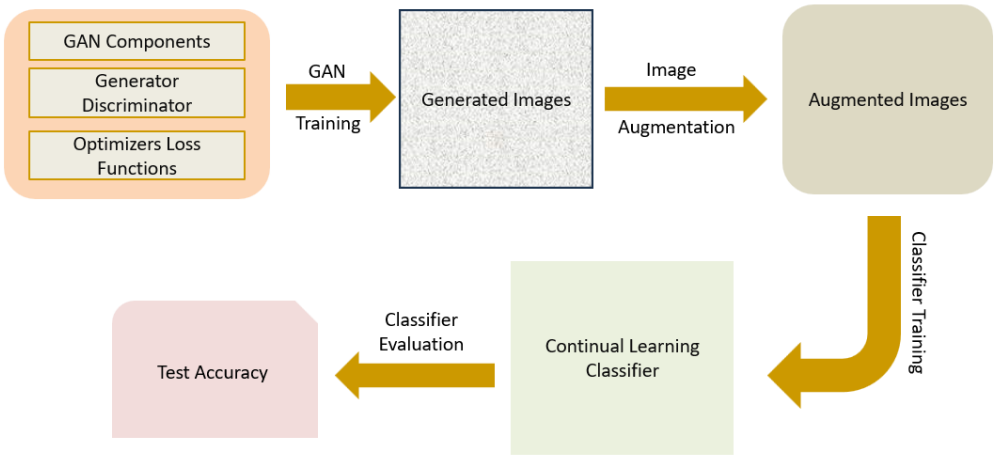


Figure 6.1: GAN with Continual Learning

3. **Continual Learning Classifier:**
   - In this GAN based experiment, a classifier was designed to be trained in a continual learning set-up, so that it can learn from data without forgetting the knowledge from previous tasks. The Continual Learning classifier serves as an architecture suitable for image classification and operates a conjunction with the GAN.
   - This classifier also estimates and updates Fisher information which is used to mitigate catastrophic forgetting during training.
4. **Classifier Training:** The train_classifier function uses the augmented images for training the classifier. It optimizes the classifier's parameters to minimize a standard cross-entropy loss between predicted and true labels.
5. **Evaluation:** After training, the classifier is evaluated on a separate test dataset to assess its performance in terms of accuracy.
6. **Visualization:** The experiment includes a function to visualize the augmented images generted by the GAN during training.

This experiment follows the Algorithm 16 with terms and notations were provided in Table 6.1.

Algorithm 16 GAN + Continual Learning Classifier

Input: $D = \{D_{\text{train}}, D_{\text{test}}\}$, hyperparameters $(B, E_{GAN}, E_{CLS}, \eta)$
Data Preparation:
$x \in \mathbb{R}^{H \times W} \quad \rightarrow \quad x_{norm} = x/255$
$(x_{batch}, y_{batch}) \sim D_{\text{train}}$
GAN Training:
for $epoch = 1 \rightarrow E_{GAN}$ **do**
  for $(x, y) \in D_{\text{train}}$ **do**
    $z \sim \mathcal{N}(0, I)$
    $\hat{x} = G(z; \theta_G)$
    $L_D = -[\log D(x; \theta_D) + \log(1 - D(\hat{x}; \theta_D))]$
    $L_G = -\log D(\hat{x}; \theta_D)$
    $\theta_D \leftarrow \theta_D - \eta \nabla_{\theta_D} L_D$
    $\theta_G \leftarrow \theta_G - \eta \nabla_{\theta_G} L_G$
  end for
end for
Continual Learning Classifier:
for $epoch = 1 \rightarrow E_{CLS}$ **do**
  $p(y|x) = C(x; \theta_C)$
  $L_{cls} = -\sum y \cdot \log p(y|x)$
  $L_F = \sum F(\theta) \cdot (\theta - \theta^{old})^2$
  $L_{total} = L_{cls} + \lambda \cdot L_F$
  $\theta_C \leftarrow \theta_C - \eta \nabla_{\theta_C} L_{total}$
end for
Evaluation:
$\hat{y} = \arg\max p(y|x)$
Accuracy $= \frac{1}{N_{test}} \sum 1(\hat{y}_i = y_i)$
Confusion Matrix $= \text{count}(\hat{y}, y)$

Table 6.1: Terms and Functions for GAN with Continual Learning Classifier

Term / Symbol	Description
$D = \{D_{\text{train}}, D_{\text{test}}\}$	Dataset with training and test sets.
B	Batch size for training.
E_{GAN}	Number of GAN training epochs.
E_{CLS}	Number of classifier training epochs.
η	Learning rate for parameter updates.
$x \in \mathbb{R}^{H \times W}$	Input image of height H and width W.
x_{norm}	Normalized image: $x_{\text{norm}} = x/255$.
$(x_{\text{batch}}, y_{\text{batch}})$	Mini-batch of images and labels sampled from D_{train}.
$z \sim \mathcal{N}(0, I)$	Latent noise vector for GAN generator.
$\hat{x} = G(z; \theta_G)$	Generated (fake) image by the GAN generator.
$D(x; \theta_D)$	Discriminator output: probability that x is real.
L_D	Discriminator loss: $L_D = -[\log D(x) + \log(1 - D(\hat{x}))]$.
L_G	Generator loss: $L_G = -\log D(\hat{x})$.
θ_D, θ_G	Discriminator and generator parameters.
$C(x; \theta_C)$	Classifier function outputting class probabilities.
$p(y\|x)$	Predicted class probability vector from classifier.
L_{cls}	Cross-entropy classification loss: $L_{\text{cls}} = -\sum y \cdot \log p(y\|x)$.
L_F	Continual learning regularization: $L_F = \sum F(\theta) \cdot (\theta - \theta^{\text{old}})^2$.
λ	Regularization coefficient for L_F.
L_{total}	Total loss for classifier: $L_{\text{total}} = L_{\text{cls}} + \lambda L_F$.
θ_C	Classifier parameters updated using gradient descent on L_{total}.
$\hat{y} = \arg\max p(y\|x)$	Predicted label for input x.
Accuracy	Accuracy $= \frac{1}{N_{\text{test}}} \sum 1(\hat{y}_i = y_i)$.
Confusion Matrix	Counts of predicted vs true labels: ConfusionMatrix $= \text{count}(\hat{y}, y)$.

Propositions for GANs for Data Augmentation with Continual Learning

1. **Proposition 1:**
   - **Statement:** The Continual Learning Classifier is trained on augmented images produced by the GAN.
   - **Rationale:** The classifier is trained both on real and GAN-synthesized malware images so that it learns from a broader dataset and gains its ability to classify new-class malware more effectively.

2. **Proposition 2:**
    - **Statement:** Fisher Information is incorporated into the Continual Learning Classifier to prevent catastrophic forgetting [231].
    - **Rationale:** Fisher Information is applied within the classifier to retain prior knowledge while adding new classes of malware, thereby preventing the model from losing current classes as it adapts to new data.
3. **Proposition 3:**
    - **Statement:** Data augmentation is a key component for improving classifier performance.
    - **Rationale:** Applying images of malware developed by GAN as augmentation information enables the Continual Learning Classifier to generalize better, particularly while learning on imbalanced datasets or a small sample sizes.

Malimg Dataset Results

Figure 6.2 is showing the confusion matrix of Malimg dataset with the experiment.

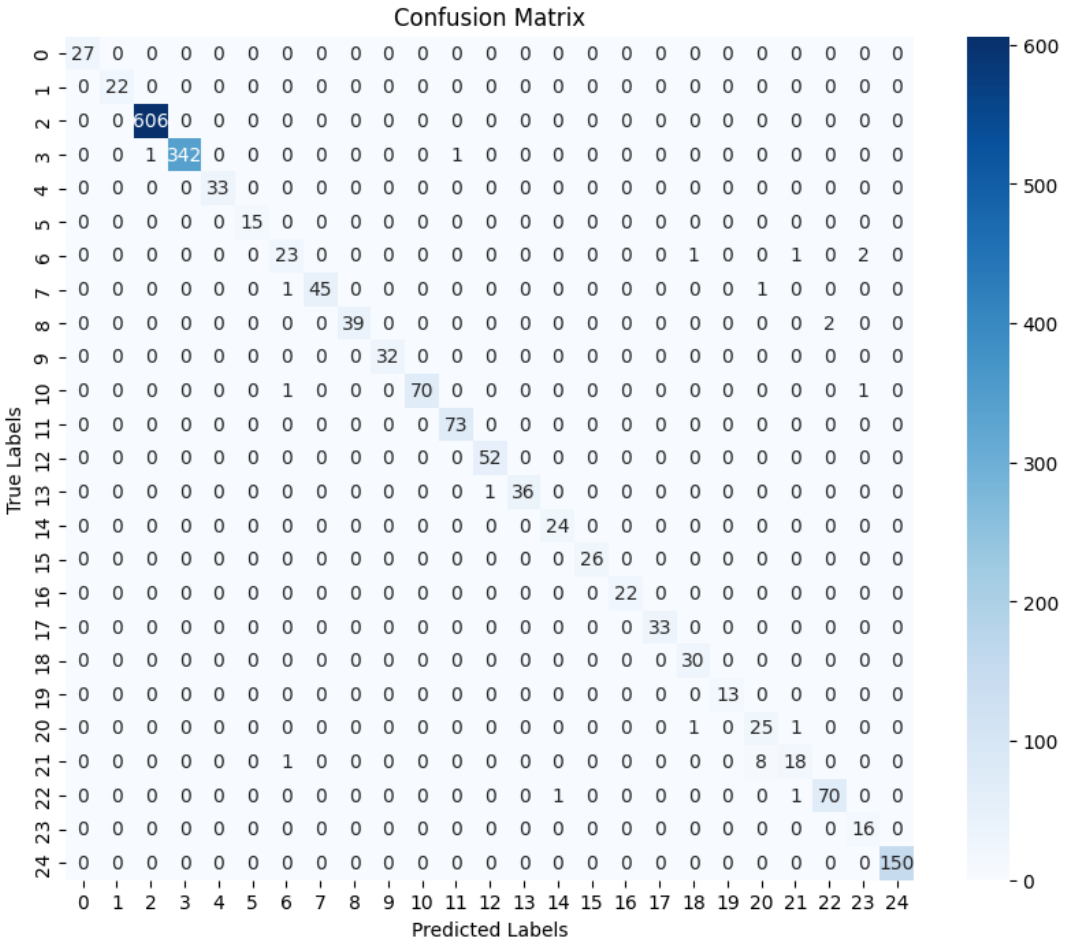


Figure 6.2: Confusion Matrix of Malimg

Virus MNIST Dataset Results

Figure 6.3 is showing the confusion matrix of Virus MNIST dataset.

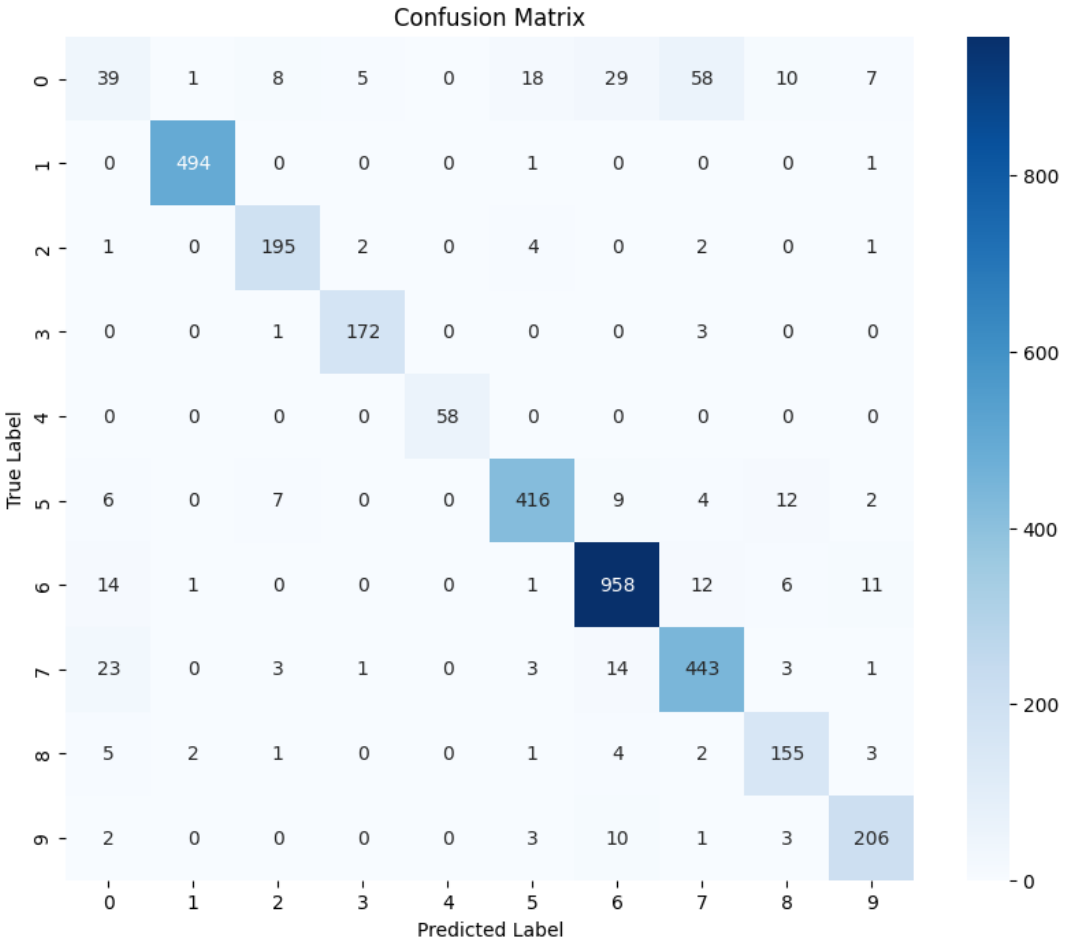


Figure 6.3: Confusion Matrix of Virus MNIST

MaleVis Dataset Results

Figure 6.4 is showing the confusion matrix of MaleVis dataset.

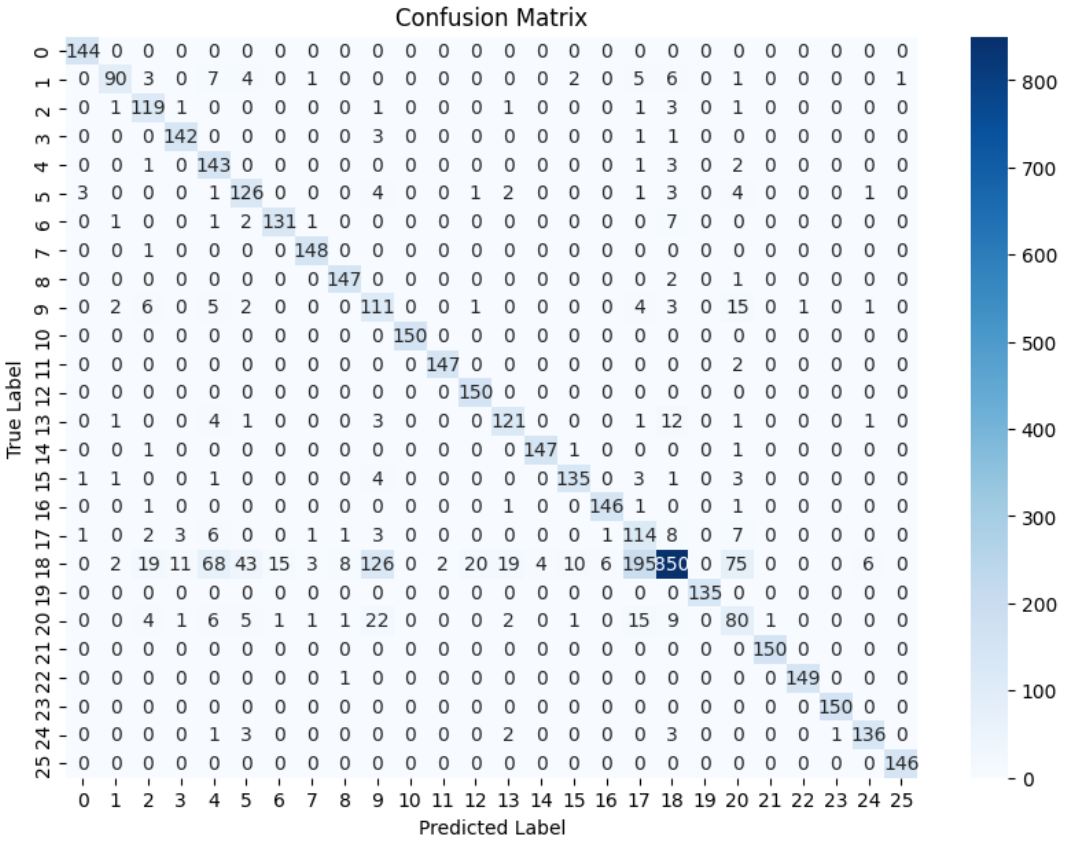


Figure 6.4: Confusion Matrix of MaleVis

Novelty

The novelty of this experiment lies in its integration of two distinct techniques - Generative Adversarial Networks (GANs) for data augmentation and Continual Learning for preserving knowledge over time.

- **Combination of GANs and Continual Learning:** It's generally common to combine GANs with continual learning method, yet this combination in image classification is quite less explored. This experimentation has linked two techniques: strengths in generating synthetic images with GANs and Continual Learning capabilities that are designed to adapt new data while retaining previous knowledge.
- **Data Augmentation with GANs:** In this experiment, GANs are uniquely used for the purpose of data augmentation. GANs provide an opportu-

nity to enlarge and diversify the training data set, therefore enhancing the classifier's robustness [232].

- **Continual Learning for Classifier Adaptation:** The inclusion of Continual Learning Classifier will retain its previously learned knowledge while learning from the augmented data produced by GAN.
- **Dynamic Fisher Information Estimation:** Fisher information is estimated dynamically during training and helps in computing penalty terms that help to prevent forgetting.

The novelty of this experiment is to incorporate the GAN-based data augmentation with Continual Learning techniques for image classification, and dynamic Fisher information estimation, that contributes to progress in both the data augmentation and continual learning research fields.

6.1.2 U-Net with Continual Learning

In this experiment, a U-Net based framework was implemented by combining with a EWC-based Continual Learning (CL) or malware image analysis tasks.

- **Define U-Net Architecture:**

  U-Net is a special CNN architecture which is primarily utilized for image segmentation. The designed U-Net architecture was composed of an encoder to downsample and a decoder to upsample [233].
- **Define Classifier on Top of U-Net:**

  The classifier consists of fully Connected Layers with activation function LeakyReLU and Batch Normalization layer. In the experiment, classifier was used to accept the output of the U-Net model for classification.
- **Continual Learner Class:** The continual learner class encompasses methods for training tasks, and the computation of Elastic Weight Consolidation (EWC) loss, as well as updates Fisher Information with regularization to manage the continual learning process.
- **Training Loop:** The Continual Learner is trained by using the train_task method. It runs the epochs and batches, measures the loss, performs back-propagation, and updates its model parameters.

- **Evaluation:** Following the training process, accuracy over epochs is plotted. Then, the trained classifier is tested on test dataset by the evaluate_classifier function.

The experiment steps were provided in Algorithm 17 with terms and notations in Table 6.2.

Algorithm 17 U-Net + Continual Learning Classifier Training Pipeline

Input: Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, number of classes C, learning rate η, epochs E, regularization λ
Output: Trained model with continual learning capability
Initialize U-Net feature extractor $\mathcal{F}_\theta$
Initialize classifier head g_ϕ
Initialize optimizer and loss function (cross-entropy)
for each task $t = 1$ to T **do**
  for each epoch **do**
    for mini-batch $(x, y) \subset D_t$ **do**
      $F \leftarrow \mathcal{F}_\theta(x)$
      $\hat{y} \leftarrow g_\phi(F)$
      $\mathcal{L}_{cls} = -\sum y \log(\hat{y})$
      $\mathcal{L}_{ewc} = 0$
      if $t > 1$ **then**
        $\mathcal{L}_{ewc} = \frac{\lambda}{2} \sum_i F_i(\theta_i - \theta_i^*)^2$
      end if
      $\mathcal{L}_{total} = \mathcal{L}_{cls} + \mathcal{L}_{ewc}$
      $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{total}$
    end for
  end for
  Store Fisher information F_i and parameter means θ_i^* for task t
end for
Evaluate accuracy on validation/test set

Table 6.2: Terms and Notations for U-Net with Continual Learning

Term / Symbol	Description
$D = \{(x_i, y_i)\}_{i=1}^{N}$	Dataset with N samples, each consisting of an input image x_i and label y_i.
C	Number of classes.
η	Learning rate for gradient updates.
E	Number of epochs per task.
λ	Regularization coefficient for Elastic Weight Consolidation (EWC).
$\mathcal{F}_\theta$	U-Net feature extractor with parameters θ.
g_ϕ	Classifier head with parameters ϕ.
x	Input image.
y	Label (one-hot encoded).
$F = \mathcal{F}_\theta(x)$	Feature vector extracted from U-Net.
$\hat{y} = g_\phi(F)$	Predicted label probabilities from the classifier head.
$\mathcal{L}_{cls}$	Cross-entropy classification loss: $-\sum y \log(\hat{y})$.
$\mathcal{L}_{ewc}$	EWC regularization to prevent forgetting: $\frac{\lambda}{2}\sum_i F_i(\theta_i - \theta_i^*)^2$.
$\mathcal{L}_{total}$	Total loss: $\mathcal{L}_{cls} + \mathcal{L}_{ewc}$.
$\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_{total}$	Gradient descent update of U-Net parameters.
F_i	Fisher information (importance) of parameter θ_i for previous tasks.
θ_i^*	Parameter mean from previous tasks used in EWC.

Propositions for U-Net with Continual Learning

1. **Proposition 1: U-Net as a Feature Extractor for Malware Image Classification**
   - **Statement:** U-Net can be utilized effectively as a feature extractor [234] for malware images, capturing the global as well as spatial relationships which enhance classification accuracy.
   - **Rationale:** CNN models are usually based on hierarchical feature extraction but may or may not capture fine-grained details needed in malware image classification. U-Net's encoder-decoder architecture holds spatial data, which will be utilized during the discrimination among malware families based on structural difference and similarity [235].
2. **Proposition 2: Continual Learning Prevents Catastrophic Forgetting in Malware Detection**

- **Statement:** The integration of Continual Learning (CLClassifier) with Elastic Weight Consolidation (EWC) enables the system to retain knowledge of previously learned malware patterns while adapting to new threats [236].
- **Rationale:** Security models will fail when challenged with new variants of malware because they have a tendency to forget past experience [123]. With Continual Learning, the model remembers past malware patterns while acquiring new ones, maintaining consistent and strong threat detection over time.

3. **Proposition 3: U-Net + CLClassifier Enables Real-Time Threat Detection and Response**
    - **Statement:** The feature extraction facilitated by the U-Net and continual learning process support real-time malware categorization, which improves incident response in cybersecurity systems.
    - **Rationale:** Conventional malware detection systems must be periodically retrained and are largely unable to cope with complex dynamic threats [237]. U-Net has fast feature extraction capacity which was combined with CLClassifier's adaptive learning that makes the system effective in improved malware detection and classification.

Novelty

- **Integration of U-Net with Continual Learning:** U-Net is a popular framework which is being widely used for image segmentation tasks, combination of U-Net with Continual Learning (CL) for malware image classification tasks is a unique idea. U-Net is mainly used for medical image segmentation, but in this experiment it's being implemented for feature extraction in a CL scenario, which is relatively less explored.
- **Continual Learning Approach:** Continual Learning is an active area of research where the model learns from continual stream of data to mitigate catastrophic forgetting. In this experiment, a Continual Learning based approach was developed with U-Net and applied for feature extraction then subsequent classification, this context is comparatively less common.

- **Regularization with Elastic Weight Consolidation (EWC):** EWC can be seen as a beneficial sequential learning technique since it prevents catastrophic forgetting and leverages the learned knowledge from earlier tasks [238]. Its incorporation as part of the EWC regularization, particularly for the U-Net based feature extraction of malware images, is considered significant.

Malimg Dataset Results

Figure 6.5 is showing the confusion matrix of Malimg dataset for U-Net based Continual Learning.

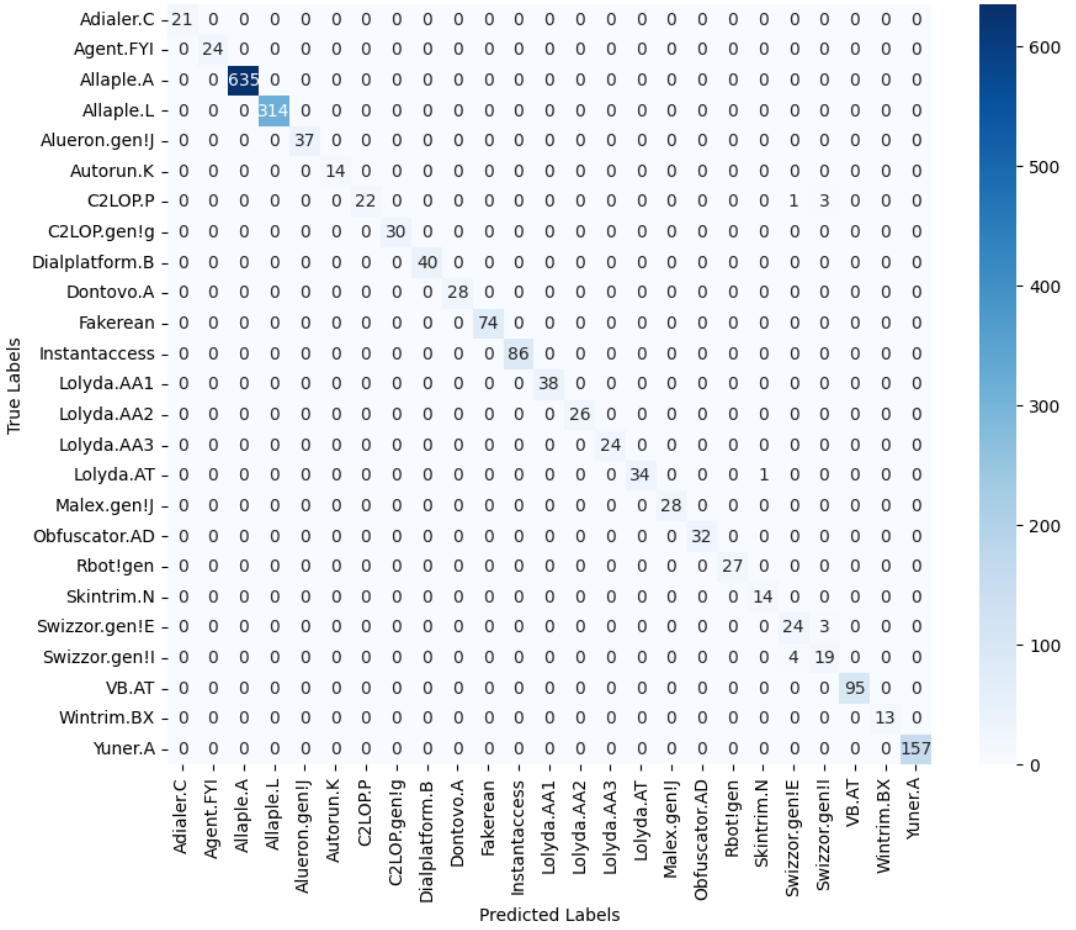


Figure 6.5: Confusion Matrix of Malimg

MaleVis Dataset Results

Figure 6.6 is showing the confusion matrix of MaleVis dataset with U-Net based Continual Learning.

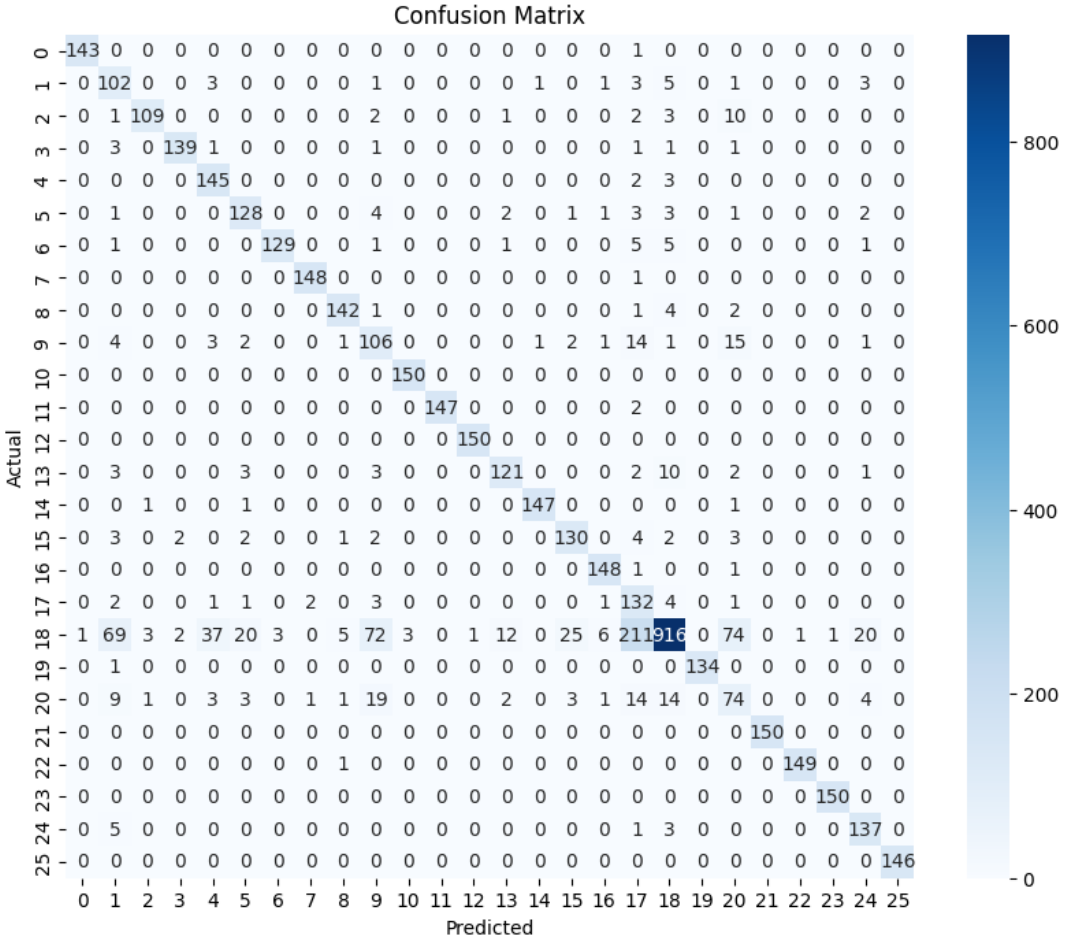


Figure 6.6: Confusion Matrix of MaleVis

Results of Datasets

This experiment was conducted on three different malware image datasets: Malimg, MaleVis and Virus MNIST. Table 6.3 is showing the results:

Table 6.3: Experimental Results with Continual Learning

Results			
Experiment	Malimg	Virus MNIST	MaleVis
GANs with Continual Learning	**Training Accuracy:** 100%, **Test Accuracy:** 98.61%	**Training Accuracy:** 99.44%, **Test Accuracy:** 90.69%	**Training Accuracy:** 99.49%, **Test Accuracy:** 82.07%
U-Net with Continual Learning	**Training Accuracy:** 100%, **Test Accuracy:** 99.41%	**Training Accuracy:** 99.43%, **Test Accuracy:** 90.98%	**Training Accuracy:** 99.57%, **Test Accuracy:** 99.30%

All the datasets were trained for 50 epochs with both the models. Table 6.3 is showing:

- For Malimg dataset, training accuracy is 100% for both GAN with CL and U-Net with CL models and it produced very high test accuracies. This demonstrates strong representation learning.
- Both the model produced stable performance for Virus MNIST dataset, which indicates that continual learning preserved previous task knowledge on a dataset which has fewer visual variations.
- For MaleVis U-Net with CL achieved high (99.30%) test accuracy which outperforms GAN with CL (82.07%) performance. This is because U-Net's skip connections and encoder-decoder structure that can effectively capture color and texture information in RGB dataset.
- Among all three datasets, MaleVis dataset presented the most challenging dynamics due to its RGB structure and high inter-class similarity across 26 malware families. But the U-Net with Continual Learning (U-Net + CL) model achieved an exceptional 99.30% test accuracy, this result was not previously reported in the literature for this dataset.

6.1.3 An Adaptive Malware Detection Using Continual Learning: Integrating Elastic Weight Consolidation and ResNet-50

This experiment was conducted to build an adaptable and resilient malware detection system which will be capable to handle the ever growing array of malware variants. Deep Learning models are highly efficient to extract and learn complex features from a dataset, but exposure to new dataset often leads to lacking of mechanisms to retain knowledge from older tasks also may require large balanced dataset for better performance. In cybersecurity domain novel malware generations are emerging frequently, therefore it's not always possible to retain a model from scratch.

A known phenomenon "Catastrophic Forgetting", arises when a Deep Learning model is being deployed across different datasets or tasks sequentially. Deep Learning models can learn the new tasks quickly but there is a tendency to forget the previously acquired knowledge which leads to a problem when the model should retain the knowledge across various malware variants [127]. To overcome this limitation Continual Learning specifically Elastic Weight Consolidation (EWC) is a potential solution to enable the model to retain essential features learned from previous tasks.

For this experiment, we have taken the three datasets: Malimg, MaleVis, and Virus MNIST to implement the EWC based transfer learning with ResNet-50 to create a robust classification framework so that the model can retain malware knowledge across three diverse datasets. The experiment explores whether this approach effectively mitigates catastrophic forgetting while adapting to new malware classes.

Experimental Set-up

The experiment follows Algorithm 18, terms and notations were provided in Table 6.1.3.

- **Load Malware Datasets:** With all the image folders Malimg, MaleVis and Virus MNIST datasets were loaded.
- **Apply Data Transforms:** Resizing of images to 64x64 to apply normalization, and convert them to tensors.

- **Initialize ResNet 50:** A pre-trained ResNet 50 was set up and the output layer was modified to classify the malware classes.
- **Initialize EWC:** Elastic Weight Consolidation (EWC) was initialized for each new task (dataset) to retain previously learned knowledge.
- **For Each Dataset (Task):** A DataLoader was created for each dataset, then train the model and apply EWC.
- **Evaluate Train/Test Accuracy:** After training, evaluate and record the train and test accuracy.

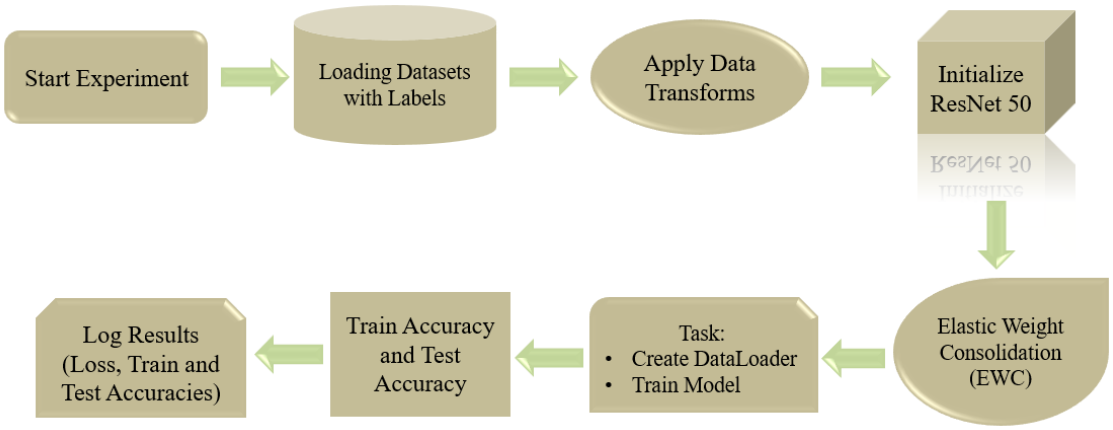


Figure 6.7: Model Architecture

Algorithm 18 Continual Learning with ResNet-50 + EWC

Input: Datasets $\{D_1, D_2, D_3\}$, learning rate η, epochs E, regularization λ
Output: Trained ResNet50 model with continual learning
Initialize pretrained ResNet50
Replace final FC layer with task-specific output
$M_{prev} \leftarrow None$, $F \leftarrow None$
for each dataset D_i **do**
  Split D_i into training T_i and test S_i
  Build dataloaders $\mathcal{L}_{train}, \mathcal{L}_{test}$
  apply_ewc = False if $i = 1$, else True
  for each epoch **do**
    for each minibatch $(x, y) \in \mathcal{L}_{train}$ **do**
      $z \leftarrow f(x; \theta)$
      $L_{task} \leftarrow -\sum_c y_c \log p(c|x; \theta)$
      $L \leftarrow L_{task}$
      if apply_ewc **then**
        $L \leftarrow L_{task} + \lambda \sum_j F_j(\theta_j - \theta_j^*)^2$
      end if
      Backpropagate L and update θ
    end for
    Evaluate accuracy on $\mathcal{L}_{train}$ and $\mathcal{L}_{test}$
  end for
  Estimate Fisher Information F for θ
  Save $\theta^* \leftarrow \theta$
end for

Table 6.4: Terms and Notations for ResNet-50 + EWC Continual Learning

Term / Symbol	Description
D_i	Dataset for task i.
T_i, S_i	Training and test splits of D_i.
$\mathcal{L}_{train}, \mathcal{L}_{test}$	Dataloaders for training and testing.
E	Number of epochs per task.
η	Learning rate for gradient updates.
λ	Regularization coefficient for EWC penalty.
$f(x;\theta)$	Forward pass of ResNet50 with parameters θ producing feature representation z.
y	Label (one-hot encoded).
$p(c\|x;\theta)$	Predicted probability of class c given input x and parameters θ.
L_{task}	Cross-entropy loss for current task: $L_{task} = -\sum_c y_c \log p(c\|x;\theta)$.
L	Total loss including EWC penalty: $L = L_{task} + \lambda \sum_j F_j(\theta_j - \theta_j^*)^2$.
F_j	Fisher information for parameter θ_j, representing its importance for previous tasks.
θ_j^*	Parameter value from previous task used in EWC regularization.
M_{prev}	Previous model (weights from last task).
θ	Parameters of current ResNet50 model.
θ^*	Saved parameters of current task for EWC.

Propositions

1. **Proposition 1:**
   - **Statement:** Model Performance across Multiple Datasets.
   - **Rationale:** In this experiment, the ResNet-50 model exhibited varying performance on three different malware (Malimg, MaleVis, Virus MNIST) dataset, when assessed in terms of continual learning using Elastic Weight Consolidation (EWC), depicting to what extent the model is able to learn new data sets but retain learned knowledge.
2. **Proposition 2:**
   - **Statement:** Effectiveness of Elastic Weight Consolidation (EWC).
   - **Rationale:** The incorporation of EWC mitigates catastrophic forgetting, which was indicated through the comparison of accuracy on datasets. Precisely, performance on the second and further tasks

(datasets) will have very little decline when EWC is utilized, confirming its success on continual learning.

3. **Proposition 3:**
    - **Statement:** Cross-Dataset Generalization Ability.
    - **Rationale:** The ResNet-50 model will generalize to the Malimg, MaleVis and Virus MNIST datasets well, showing that when training on a single data set, good performance on unseen data sets will occur, which suggests the robustness of the model in dealing with different kinds of malware images.

Novelty

1. **Continual Learning for Malware Classification**
    - **Unique Application of EWC:**

      Elastic Weight Consolidation (EWC) was used to deal with catastrophic forgetting, ensuring the model to retain essential knowledge from previously learned tasks while it's learning from new data [239]. Here the model is retaining knowledge across distinct malware datasets: Malimg, MaleVis, Virus MNIST, while mitigating catastrophic forgetting. This is innovative because it's challenging the model to adapt to different malware classes sequentially without the loss of past knowledge.
    - **Practical Utility in Cybersecurity:**

      This Continual Learning approach is beneficial for real world cybersecurity environment because in this era of ever evolving and diverse malware strains a model with retaining and adapting capability is enriching the security system without forgetting the past threats.

2. **Cross-Dataset Malware Classification Using Transfer Learning**
    - **Transfer Learning in Malware Context:**

      In this experiment, a transfer learning was incorporated with ResNet-50 model and applied to malware images which have different texture and pattern characteristics, is an innovative use case.

- **Handling Different Dataset Distributions:**

  In this experiment three distinct malware datasets: Malimg, MaleVis, and Virus MNIST were incorporated where each of them represents different data distributions and characteristics. Novelty of transfer learning across datasets is seen in the generalization and robustness of the model across the forms of malware.

3. **Experimenting with Malware-Specific ResNet-50 Customization**
   - **Adjustments for Malware Dataset-Specific Classifications:**

     The experiment deals with 61 malware classes across three datasets (Malimg: 25 classes, MaleVis: 26 classes, Virus MNIST: 10 classes) that were accommodated with the ResNet-50 model. Here the malware specific characteristics were being adapted with the model.
   - **EWC's Use in Preserving Malware Feature Distributions:**

     In case of malware images, subtle and intricate feature distributions are often noticed, exhibiting a difference from natural textures [240]. For this experiment, EWC was leveraged to preserve the critical feature distributions across tasks, which may not have been precisely explored in malware research.

Results and Discussions

ResNet-50 EWC	Dataset	Training Accuracy	Test Accuracy
Experiment 1	Malimg	96.62%	96.62%
	MaleVis	91.49%	88.41%
	Virus MNIST	90.89%	90.54%
Experiment 2	MaleVis	98.88%	86.09%
	Malimg	97.31%	94.97%
	Virus MNIST	87.52%	85.45%
Experiment 3	Virus MNIST	97.80%	90.08%
	Malimg	95.44%	95.29%
	MaleVis	89.48%	79.50%

Table 6.5: Training and Test Accuracy for Different Tasks

1. **Experiment 1: (Malimg $\rightarrow$ MaleVis $\rightarrow$ Virus MNIST) is the Best Order**
   - Test accuracy remained high across all tasks, suggesting minimal forgetting.

- Malimg was the most stable dataset, as its training and test accuracies matched.
- Virus MNIST retained 90.54% accuracy, indicating effective EWC application.

2. **MaleVis Suffers the Most Forgetting**

   In Experiment 3, when MaleVis was trained last, it dropped to 79.50% test accuracy, indicating significant forgetting.
3. **Virus MNIST Experiences Moderate Forgetting**
   - When trained first (Experiment 3), it retained 90.08% accuracy.
   - When trained last (Experiment 2), its accuracy dropped to 85.45%, suggesting some knowledge retention challenges.
4. **Malimg is the Most Generalizable Dataset**
   - Malimg consistently maintained high test accuracy (94.97% - 95.29% - 96.62%).
   - This suggests that Malimg's malware family features are more distinctive and robust.

A consistent high accuracy is observed for all three datasets signifies that ResNet-50 with EWC regularization, is much effective for Continual Learning for malware classification without sacrificing accuracy from previous tasks.

The performance gap is much smaller across the tasks, illustrates that EWC effectively mitigated catastrophic forgetting. This method suggests that the selected model architecture and continual learning strategy are well-suited for handling new malware types or evolving datasets in cybersecurity.

6.2 Conclusions

The experiments with GANs and U-Net with Continual Learning Classifier (CLClassifier) demonstrated promising advancements in malware classification. GANs helped generate synthetic malware images for augmentation, improving class balance and model generalization. U-Net with CLClassifier proved highly effective in extracting detailed malware features and retaining prior knowledge while learning new malware classes, addressing the challenge of catastrophic

forgetting. The success of continual learning in maintaining accuracy across evolving malware families highlights its critical role in real-world cybersecurity, where malware constantly evolves.

The experiments in subsection 6.1.3 conducted across the Malimg, MaleVis, and Virus MNIST datasets evaluated the effectiveness of deep learning and continual learning in malware image classification. ResNet-50 achieved high accuracy across all datasets, demonstrating strong feature extraction capabilities. However, accuracy varied, with Malimg performing the best due to distinct malware family features, while MaleVis exhibited slightly lower accuracy due to greater intra-class variations. The Virus MNIST dataset also showed promising results despite its smaller image size. To address catastrophic forgetting, Elastic Weight Consolidation (EWC) was implemented, allowing the model to retain prior knowledge while adapting to new datasets, ensuring stable performance across sequential learning tasks. Although EWC introduced a slight computational overhead, it significantly improved long-term knowledge retention, reinforcing the importance of continual learning in dynamic cybersecurity environments, where malware constantly evolves.

The Table 6.6 is showing the comparison with literature and performance of continual learning with all three datasets.

Table 6.6: Experimental Results with Accuracy

Results			
Experiment	Malimg	Virus MNIST	MaleVis
Thosar et al., 2021 [225] **CNN**	X	93.45%	X
Wong et al., 2021 [226] **ShuffleNet and DenseNet 201 with Transfer Learning**	Training Accuracy: 99.14%, Test Accuracy: 98.87%,	Training Accuracy: 86.36%, Test Accuracy: 85.79%	Training Accuracy: 95.01%, Test Accuracy: 93.91%
Sharma et al., 2020 [227] **CNN**	Test accuracy: 97.5%	X	X
Pant and Bista, 2021 [228]	**CNN:** 98.2%, **VGG-16:** 88.40%, **ResNet-18:** 90.21%	X	X
Combination of CNN and RNN	Training Accuracy: 100%, Test Accuracy: 98.05%	Training Accuracy: 93.18%, Test Accuracy: 92.72%	**Training Accuracy: 96.70%, Test Accuracy: 94.51%**
CNN	**Training Accuracy: 100%, Test Accuracy: 98.26%**	Training Accuracy: 99.71%, Test Accuracy: 91.25%	Training Accuracy: 99.82%, Test Accuracy: 90.05%
ResNet-50 Attention Network	Training Accuracy: 98.82%, Test Accuracy: 97.91%	Training Accuracy: 99.99% Test Accuracy: 92.35%	Training Accuracy: 99.67%, Test Accuracy: 84.63%
GANs with Continual Learning	**Training Accuracy: 100%**, **Test Accuracy: 98.61%**	**Training Accuracy:** 99.44%, **Test Accuracy:** 90.69%	**Training Accuracy:** 99.49%, **Test Accuracy:** 82.07%
U-Net with Continual Learning	**Training Accuracy: 100%**, **Test Accuracy: 99.41%**,	**Training Accuracy:** 99.43%, **Test Accuracy:** 90.98%	**Training Accuracy: 99.57%**, **Test Accuracy: 99.30%**

7 Combination of Ontology and Deep Learning

7.1 Introduction

This chapter explores the innovative approaches to combine ontology and deep learning. This combination will bridge the gap between two different domains: Semantic Web and Deep Learning. At first, this chapter is trying to combine these two fields with Domain Adaptation techniques. For that purpose, the first experiment was carried out with ontology as source domain and deep learning as target domain; the second Domain Adaptation experiment utilizes Deep Learning as source domain and ontology as target domain. For both the experiments results were discussed.

The next section of this chapter has produced three deep learning based novel malware knowledge graphs by incorporating the developed ontology from Chapter 4 with three distinct malware image datasets, respectively.

By analyzing the state-of-the-art provided in section 3.5 and section 3.6, this chapter is trying to achieve following research gaps:

- Domain adaptation methods, although effective in coordinating visual features across domains, are related only to statistical distribution matching. They have no semantic awareness, explainability, and the capacity to incorporate domain knowledge.
- No existing literature integrated heterogeneous data sources like images and ontologies into a unified learning framework. Motivated by this gap, our approach implemented domain adaptation techniques to align image-based and ontology-based feature spaces, enabling the model to

leverage both visual and semantic features and improve generalization across two inherently different domains.

- Cybersecurity knowledge graphs and ontologies provide rich semantic structures to represent system-level interactions and malware behavior. However, these systems tend to be static, manually composed, and decoupled from real-time detection models. They do not learn or scale with high-dimensional data like malware images.
- Existing malware ontologies reach only as far as behavioural rule modeling and are not integrated into data-driven learning pipelines. Most knowledge graphs are constructed from text sources and do not engage with deep learning outputs, so they are of lesser value for dynamic classification tasks.

Motivated by the gaps, this research introduced novel frameworks which combine the strengths of both paradigms. The first one performs domain adaptation from one source domain to another target domain. The second one leverages the ontology not only as a classification target but also as a foundation for generating semantically rich knowledge graphs. Both of the methods enable interpretable, cross-domain malware detection, grounded in both statistical learning and semantic reasoning.

7.1.1 Combining Ontology and Deep Learning with Domain Adaptation

Ontologies offer the way of consistently representing knowledge, concepts, and relationships of any kind on a given domain. In fact, consistent representation provides a coherent way in which the knowledge will be systematically structured and organized, therefore easily understood and shared. It is a useful tool in knowledge management, sharing of knowledge, and the building up of knowledge bases; knowledge transfer within the same and different domains can easily be adapted [241]. Through the technique of ontology alignment, knowledge from one domain can be mapped to another and thus allows insights, best practices, and lessons learned across diverse domains. Ontologies promote knowledge reusability by encapsulating domain-specific knowledge in a reusable format [242]. Once developed, ontologies can be

reused across different applications, projects, and organizations within the same domain, eliminating the need to recreate knowledge from scratch.

Scalability, versatility and high performance have made deep learning algorithms an indispensable tool in a range of computer vision problems, from simple image classification to a more complex task of synthesizing and understanding images [243]. Examples of deep learning models to learn hierarchical representations of the input data include CNN, RNN or transformers. These models automatically extract features from raw data, capturing both low-level and high-level representations relevant for the task at hand. Deep convolutional neural networks have become the de facto standard for image classification tasks [244]. Object tracking algorithms based on deep learning make use of RNNs, such as LSTM networks or newer transformer-based architectures like Transformer, that model temporal dependencies for object tracking between video frames. These include generative models, such as GANs and Autoencoders for image generation, image inpainting, super-resolution, and style transfer, synthesizing realistic images from noise or incomplete inputs.

But in the case of Deep Learning algorithms, the requirement of labeled data and domain shift between training and test data distributions are the major challenges. Deep Learning models perform well if the training and test data are coming from the same data distribution. Deep Learning models often fail to combine two datasets coming from significantly different domains [245]. Here comes the concept of “Domain Adaptation". This approach aligns the distributions of the source and target domains by learning domain-invariant representations.

Definition 1. *“Domain Adaptation is a technique to improve the performance of a model on a target domain containing insufficient annotated data by using the knowledge learned by the model from another related domain with adequate labeled data* [1]*."*

Domain adaptation tries to bridge the gap in distribution between the source and target by aligning their feature spaces or adapting classifiers to the new domain [246]. Feature alignment approaches usually aim at mapping the features from both domains to a common latent space where the distributions

[1] https://www.v7labs.com/blog/domain-adaptation-guide

are more aligned [247]. This can be achieved by various techniques, adversarial learning among them, where a domain discriminator tries to distinguish source and target domain samples while the feature generator tries to confuse it by minimizing this discrimination.

On the other hand, classifier adaptation aims at adapting a source-trained classifier to target-domain performance, which typically involves fine-tuning it using a target domain with limited labeled data or re-weighting these samples in an attempt to find minimum source-target distribution discrepancies.

Both approaches try to minimize both marginal (overall distribution) and conditional mismatch (distribution conditioned on the class labels of domains) for better performance, improving model performance when then applied to new, unseen data from the target domain.

Therefore we can assume domain adaptation as a game like structure between three players: the feature extractor, the domain discriminator and the classifier. The feature extractor performs like a strategist which aims to learn features that are useful for classification in the source domain while confusing the domain discriminator about whether the sample is coming from the source or target domain. The domain discriminator is attempting to correctly distinguish the origin of the feature, i.e., whether it belongs to the source domain or target domain. While the classifier's action is to focus on improving the classification performance.

7.2 Integration of Ontology and Deep Learning: Source Domain: Ontology; Target Domain: Malware Image Dataset

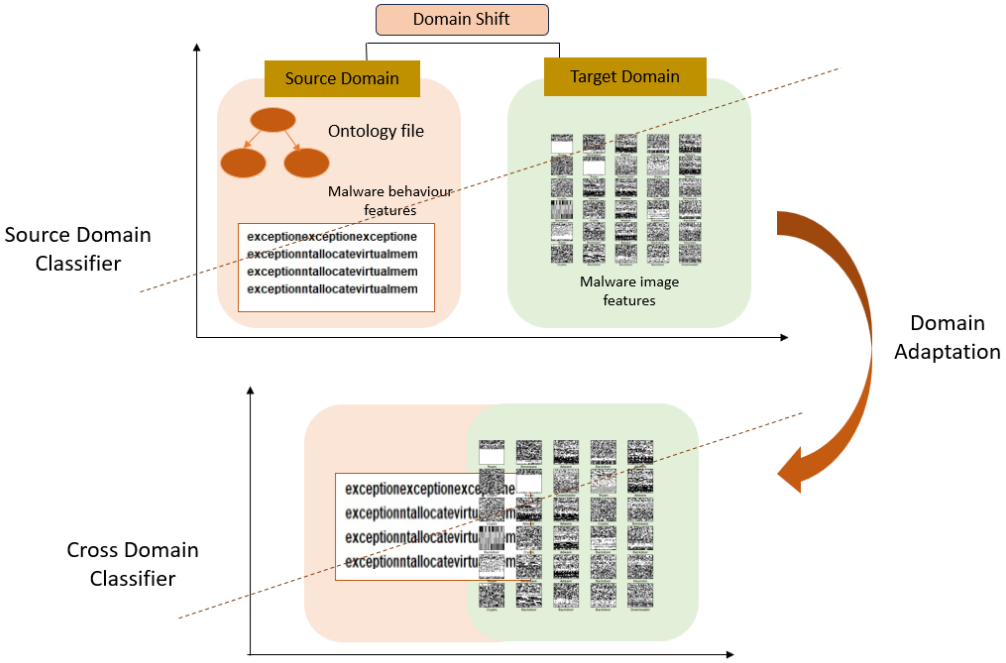


Figure 7.1: Domain Adaptation from Ontology to Malware Image

Benefits:

Combining ontology-based knowledge representation with deep learning techniques for domain adaptation offers several advantages in addressing the challenges of transferring knowledge between different domains.

- **Semantic Alignment:** Ontologies are structured representations of domain knowledge defined by concepts, relationships and axioms[2]. Aligning ontology concepts with the features extracted by deep learning models may reveal semantic similarities across different domains. By captur-

[2] https://www.obitko.com/tutorials/ontologies-semantic-web/what-is-ontology.html

ing shared semantic information, it makes knowledge transfer from the source domain to the target domain effective.

- **Feature Fusion:** Deep learning models extract high-dimensional features from raw data, which capture intricate patterns and representations [248]. Besides, ontology-based features are supposed to incorporate into deep learning features for more completeness of feature representation. It enhances the capability of adaptation to the target domain by introducing domain-specific semantic information from the ontology.
- **Ontology-Aware Loss Functions:** These can be guided into the loss function designs to encourage the learning of domain-invariant representation by incorporating ontology-based constraints or objectives. For instance, semantic similarity metrics derived from the ontology could be incorporated into loss functions to urge the model to learn consistent representations between domains. This helps in minimizing the domain gap and improves the adaptation performance of the model.
- **Domain Specific Knowledge Injection:** Ontologies capture domain-specific knowledge regarding hierarchical relationships, constraints and rules. Deep learning models can make use of such knowledge by incorporating domain-specific constraints or regularization terms during training. For example, consistency with ontology-defined relationships may be used to regularize the learning process to yield domain-adaptive representations that adhere to domain-specific semantics.
- **Enhanced Interpretability:** Ontology knowledge representation integrated into deep models can enhance interpretability in domain adaptation. By aligning the concepts and relationships defined in ontologies with learned representations, interpretation of model decisions and predictions are possible in terms of domain-specific semantics. This situation improves the process of comprehension and validation of the adaptation process at a higher degree.

Domain adaptation, which has combined ontology with deep learning, has been pretty synergistic, enabling the strengths of both methodologies. Ontology guided deep learning models can handle challenges in knowledge transfer across diverse domains by integrating semantic knowledge representation with data-driven feature learning while preserving domain-specific semantics.

7.2.1 Methodology

The purpose of this Chapter is to integrate 'Ontology' and 'Deep Learning' by combining the features from the ontology (.owl) file and malware image dataset. The behaviour (sequences of API calls) from the Ontology file which was created in Chapter 4, was used as features for 'source domain' and features from malware image dataset was used as 'target domain'.

Domain adaptation is a technique that used to address the challenge of learning from a source domain with labeled data and applying that knowledge to a different but related target domain where labeled data might be scarce or unavailable. In summary, it's about transferring knowledge from a source domain to a target domain even when there are differences in their distributions.

Though our experiment is quite challenging as it's an effort to implement Domain Adaptation in between two inherently different domains: Ontology and deep learning (malware image datasets).

Domain Adaptation from ontology to malware image datasets

1. **Define Source and Target Domains:**
    - **Source domain:** Malware behaviour ontology (e.g., ontology embeddings).
    - **Target domain:** The target domain for this experiment is malware image datasets (each of Malimg, MaleVis and Virus MNIST dataset).
2. **Data Preparation:**
    - **Source domain data:**

      Extraction or generation of feature representation from the source domain e.g., ontology embeddings.
    - **Target domain data:**

      After loading the malware image data, preprocessing techniques like resize, normalization etc. were applied.
3. **Define Feature Encoders:**
    - **Source encoder:** It's a simple neural network for embeddings designed to map the source domain features into a shared latent space.

- **Target encoder:** A CNN model for images, was implemented to map target domain data into the same latent space.

4. **Define Classifier:**

   The classifier was defined to predict the classes.

5. **Domain Discriminator:**
   - It's a neural network which tries to predict whether a feature is coming from the source domain or the target domain.
   - Domain Discriminator works simultaneously with Gradient Reversal Layer (GRL) which enables the encoder to make source and target features indistinguishable.

   Figure 7.2 is showing the framework of domain adaptation from ontology to malware image.

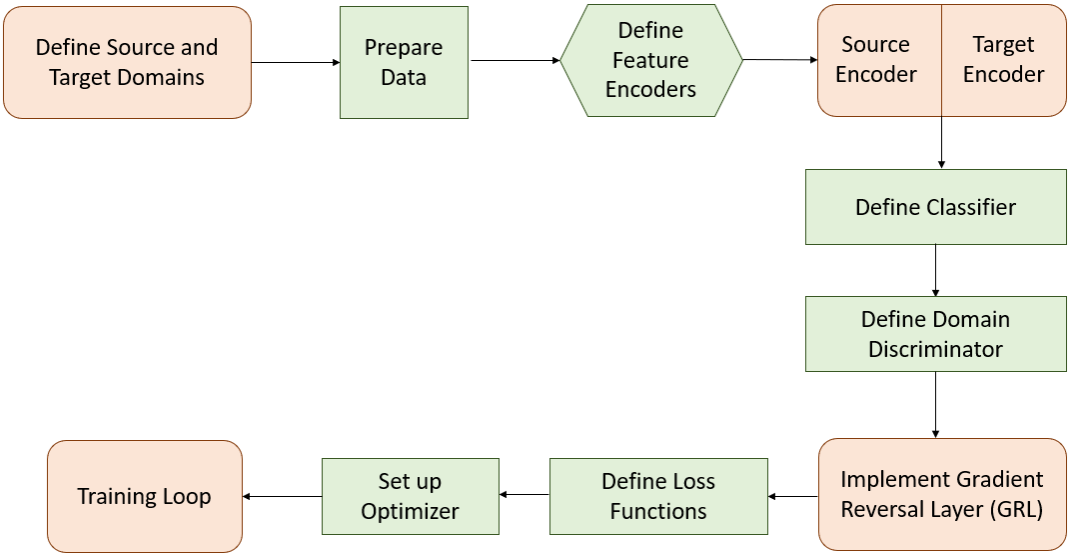


Figure 7.2: Domain Adaptation from Ontology to Malware Image

6. **Gradient Reversal Layer (GRL):**
   - It's a special layer in domain adaptation, which works during the backward pass to multiply the gradients.

- By doing this GRL enables the feature encoder to make source and target features indifferentiable.

7. **Training Loop:**
   (a) **Forward pass:**
      - Pass source and target features to the latent space through the source and target encoders respectively.
      - Concentrating source and target latent features for the domain discriminator input.
      - Gradient Reversal Layer (GRL) was applied before feeding into domain discriminator.
      - Discriminator tries to predict source and target features.

   (b) Computation of total loss after calculating classification loss and domain loss.

   (c) **Backward pass:**
      - Backpropagation of total loss.
      - During backward pass, the GRL's purpose is to ensure that the encoder should get reversed gradients for domain loss.
      - Thus the encoder produces domain-invariant features.

   (d) **Metrics:**
      - Classification accuracy.
      - Domain discriminator accuracy.

The experimental steps followed Algorithm 19 with terms and notation provided in Table 7.1.

Algorithm 19 Domain Adaptation from ontology to malware image

Input: Source dataset D_s, Target dataset D_t, learning rate η, epochs E, balancing parameter λ
Output: Trained feature extractor F with classifier C
Initialize feature extractor $F(\cdot;\theta_f)$
Initialize label classifier $C(\cdot;\theta_c)$
Initialize domain classifier $D(\cdot;\theta_d)$
for each epoch $e = 1 \dots E$ **do**
  Sample minibatch $(x_s, y_s) \sim D_s$, $(x_t) \sim D_t$
  $z_s \leftarrow F(x_s;\theta_f)$
  $z_t \leftarrow F(x_t;\theta_f)$
  $\hat{y}_s \leftarrow C(z_s;\theta_c)$
  $\hat{d}_s \leftarrow D(\mathrm{GRL}(z_s);\theta_d)$
  $\hat{d}_t \leftarrow D(\mathrm{GRL}(z_t);\theta_d)$
  $L_{cls} \leftarrow -\sum_c y_{s,c} \log p(c|z_s;\theta_c)$
  $L_{dom} \leftarrow BCE(\hat{d}_s, 0) + BCE(\hat{d}_t, 1)$
  $L \leftarrow L_{cls} + \lambda L_{dom}$
  Backpropagate L and update $\theta_f, \theta_c, \theta_d$
  Evaluate source classification accuracy on D_s
  Evaluate domain accuracy on $\{D_s, D_t\}$
end for
return F, C

Table 7.1: Notation and Terms for Domain Adaptation Algorithm

Notation / Term	**Meaning / Description**
D_s	Source dataset (labeled)
D_t	Target dataset (unlabeled or labeled)
η	Learning rate for optimization
E	Number of training epochs
λ	Balancing parameter or weighting factor between classification loss and domain loss
$F(\cdot;\theta_f)$	Feature extractor or encoder mapping input to latent space; parameters θ_f
$C(\cdot;\theta_c)$	Label classifier to predict labels; parameters θ_c
$D(\cdot;\theta_d)$	Domain classifier predicting source vs. target domain; parameters θ_d
(x_s, y_s)	Minibatch of source inputs and labels
x_t	Minibatch of target inputs
z_s, z_t	Latent feature representations of source and target after feature extractor
$\hat{y}_s$	Predicted source labels from classifier C
$\hat{d}_s, \hat{d}_t$	Predicted domain labels from domain classifier D after GRL
GRL$(\cdot)$	Gradient Reversal Layer; multiplies gradients by $-\alpha$ during backpropagation to confuse domain classifier
L_{cls}	Classification loss (cross-entropy) on source labels
L_{dom}	Domain loss (e.g., BCE) for source vs target discrimination
L	Total loss: $L = L_{cls} + \lambda L_{dom}$
$\theta_f, \theta_c, \theta_d$	Trainable parameters of feature extractor, classifier and domain classifier
Source classification accuracy	Accuracy of $C(F(x_s))$ on source dataset
Domain accuracy	Accuracy of $D(F(x))$ in predicting domain label (source vs target)

Results

Table 7.2: Results of Ontology to Image Domain Adaptation

Dataset	Train (%)	Test (%)	Dom. Train (%)	Dom. Test (%)
MaleVis	99.56	87.53	55.73	47.00
Malimg	97.06	97.00	70.26	57.83
VirusMNIST	95.14	90.92	81.32	52.04

Table 7.2 is showing the experimental results with all three image datasets.

- The models achieved high training and testing accuracies for all datasets (MaleVis: 87.53%, Malimg: 97%, VirusMNIST: 90.92%).
- **MaleVis Dataset:**
  1. Domain Train Accuracy (55.73%) and Domain Test Accuracy (47%), which suggest a good domain adaptation.
  2. Here the feature extractor succeeded in confusing the discriminator and makes ontology features and MaleVis image features domain-invariant.
- **Malimg Dataset:**
  1. Domain discriminator achieved 70.26% on training and 57.83% on testing.
  2. This indicates model learnt distinction between ontology and Malimg features but full invariance is not yet achieved.
- **Virus MNIST Dataset:**
  1. The discriminator achieved high (81.32%) training accuracy. So ontology and Virus MNIST features are clearly distinguishable in training phase.
  2. Domain test accuracy is 52.04%, that's a positive sign which suggests the feature extractor successfully confused the discriminator during testing and leads to a more balanced feature alignment.

Novelty

In this experiment, labels are different for both ontology file and the malware image datasets. The ontology (.owl) file was created by incorporating .csv files which contain malware behaviour information and malware image datasets have the images of several malware types. The ontology file and the malware image files are not very much related to each other. This section is describing the novelty of this experiment:

- **Source-Target Modality Shift:** Common domain adaptation experiments generally align “different but related" domains e.g., image-to-image, text-to-text. This study implements the “ontology-to-image adaptation" which bridges structured knowledge representations with visual data.
- **Ontology as Source Domain:** A unique concept of this experiment is to use ontology embeddings as source domain. Leveraging semantic knowledge graph as the source domain is novel in malware analysis field.
- **Integration of GRL:** Incorporation of Gradient Reversal Layer [249] to align semantic embeddings with image features enables the encoder to confuse the domain discriminator and learn a shared latent space.

  This process can effectively handle heterogeneous data types.

7.3 Integration of Deep Learning and Ontology: Source Domain: Malware Image Dataset; Target Domain: Ontology

Cybersecurity threats are evolving fast especially malware which in the current landscape renders traditional signature-based detection ineffective against metamorphic and zero-day attacks. While deep learning demonstrates much promise for malware classification, generalization and explainability are usually problematic in these models.

This experiment considers Domain Adaptation, where the source domain will be malware image datasets, for example Malimg, MaleVis and Virus MNIST, and an ontology-based knowledge graph as a target domain. This deep learning with ontology fusion will provide the capability to transfer knowledge across

these domains for improvement in accuracy, adaptability and interpretability of malware detection.

1. **Motivation**

   This experiment leverages Domain Adaptation, where malware image datasets act as the source domain and an ontology-based knowledge graph serves as the target domain. Figure 7.3 is showing the methodology of the experiment. By integrating malware image with ontology, the objective was:

   - **Enhance Generalization**: Adapt features from malware images to ontology-based knowledge graph that improves the detection performance on different datasets.
   - **Improve Explainability**: Integrating deep learning predictions with structured knowledge graphs so that model decisions are more understandable.

   This novelty closes the gap between purely data-driven and purely knowledge-driven cybersecurity solutions and advances toward more adaptive, interpretable and effective malware classification.

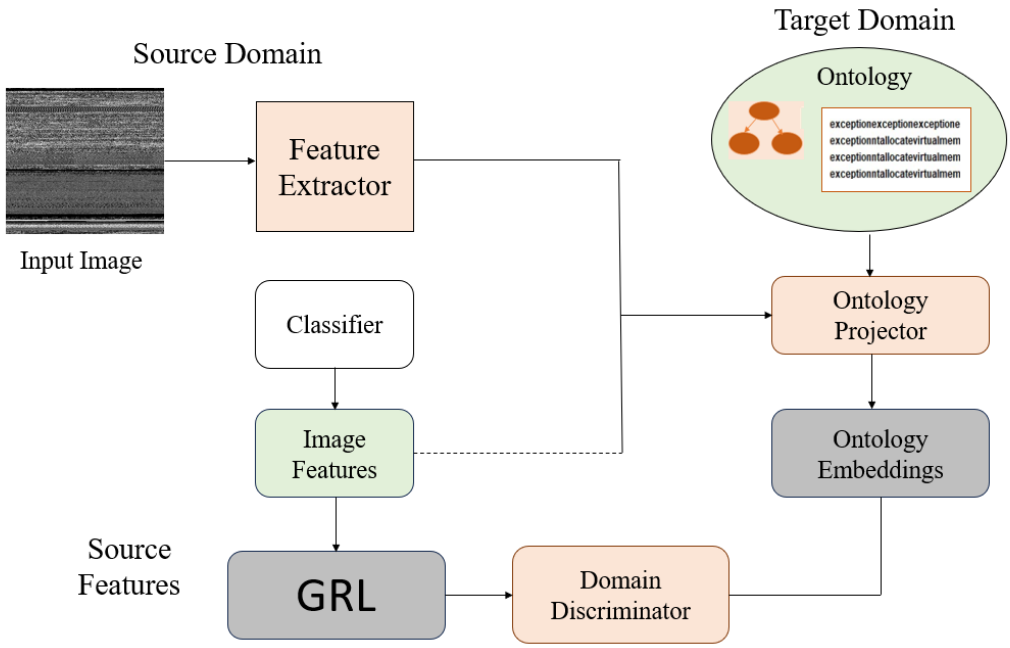


Figure 7.3: Domain Adaptation from Ontology to Malware Image

2. **Methodology**

(a) **Dataset Preparation**

In this experiment,

- **Source domain** is malware image dataset (either Malimg or MaleVis or Virus MNIST).
- **Target domain** is ontology embeddings.
- The motivation is to learn feature representation which should be discriminative for the classifier but domain-invariant for the domain discriminator.

(b) **Data Preprocessing**

- Resizing of images and normalizing the pixel values.
- Parsing the ontology file in Turtle or RDF format.
- Extraction of all entities (subjects and objects) from the triples of the ontology.
- **Assign embeddings:** Each entity was assigned a vector of fixed dimension.
- This embedding can be projected into the feature space of the image domain using a Ontology Projector.

(c) **Model Development**

Three main components of the model:

- **Feature Extractor** is a CNN or pre-trained ResNet model which extracts feature vector from an image.
- **Ontology Projector** is a feed-forward network was incorporated to map ontology embeddings into the same feature space as the image features.
- **Domain Discriminator** is a classifier which predicts whether the feature is coming from the source domain (image dataset) or the target (ontology) domain.
  In this experiment also the domain discriminator is connected through the Gradient Reversal Layer (GRL) which encourages domain-invariant features.

(d) **Loss Functions**

Total loss was calculated by classification loss and domain adversarial loss.

(e) **Training Loop**

- **Forward pass** which computes source features, ontology features and classification logits.
- Classification logits apply the classifier head on the source features (images) only to predict the malware class.
- Follwing the computation of classification and domain adversarial losses, the backpropagation of total loss was conducted and networks (feature extractor, classifier, domain discriminator) were updated.

(f) **Feature Space Alignment**

- Ontology embeddings were mapped to the same feature space with the image features by the ontology projector.
- Domain discriminator along with GRL ensures feature extractor to produce features that should be indistinguishable between domains.

(g) **Metrics**

- Classification accuracy was evaluated for the source domain.
- Domain discriminator accuracy was calculated to evaluate domain alignment (lower accuracy is better, which indicates better domain-invariance).

The experiment follows Algorithm 20 with terms and notations were described in Table 7.3.

Algorithm 20 Image $\rightarrow$ Ontology Domain Adaptation

Input: Image dataset D_s, Ontology triples T, learning rate η, epochs E, balancing parameter λ
Output: Trained networks F, C, P, D
Extract entities $\mathcal{E}$ from T
Initialize embeddings $e \in \mathbb{R}^{d_{emb}}$ for $e \in \mathcal{E}$
Define ontology projector $P : \mathbb{R}^{d_{emb}} \rightarrow \mathbb{R}^{d_{feat}}$
for epoch $= 1 \dots E$ **do**
  for $(x_s, y_s) \sim D_s$, $e_t \sim \mathcal{E}$ **do**
    $f_s = F(x_s)$
    $f_t = P(e_t)$
    $z_s = C(f_s)$
    $d_s = D(\mathsf{GRL}(f_s))$
    $d_t = D(f_t)$
    $L_{cls} = \mathsf{CE}(z_s, y_s)$
    $L_{dom} = \mathsf{BCE}(d_s, 0) + \mathsf{BCE}(d_t, 1)$
    $L = L_{cls} + \lambda L_{dom}$
    Backpropagate L and update $\{F, C, P, D\}$
  end for
end for
Return F, C, P, D

Table 7.3: Terms and Notations for Image to Ontology Domain Adaptation

Term / Notation	Description
D_s	Source domain dataset (image dataset)
T	Set of ontology triples (subject, predicate, object)
$\mathcal{E}$	Set of unique entities extracted from ontology (subjects $\cup$ objects)
$e \in \mathcal{E}$	An entity from the ontology
d_{emb}	Dimensionality of ontology embeddings
$e_t \in \mathbb{R}^{d_{emb}}$	Embedding of ontology entity e
F	Feature extractor network (maps images to feature space)
C	Classifier network (predicts image class from features)
P	Ontology projector network (maps ontology embeddings to feature space)
D	Domain discriminator network (predicts source/target domain)
$x_s \in D_s$	Image sample from source domain
y_s	Class label of image sample x_s
$f_s = F(x_s)$	Image feature vector from feature extractor
$f_t = P(e_t)$	Ontology feature vector from projector
$z_s = C(f_s)$	Classification logits for image sample
$d_s = D(\mathsf{GRL}(f_s))$	Domain logits for image features (source domain)
$d_t = D(f_t)$	Domain logits for ontology features (target domain)
$\mathsf{GRL}(\cdot)$	Gradient reversal layer for domain adaptation
L_{cls}	Classification loss (Cross-Entropy between z_s and y_s)
L_{dom}	Domain loss (Binary Cross-Entropy between d_s, d_t and domain labels)
λ	Balancing parameter between classification and domain loss
L	Total loss: $L = L_{cls} + \lambda L_{dom}$
η	Learning rate
E	Total number of epochs

3. **Novelty**

   (a) **Cross-modal Domain Adaptation**

   This experiment connects two inherently different domains: image-based source domain with a structured ontology-based target domain and effectively aligns the heterogeneous data modalities.

   (b) **Ontology-driven Feature Alignment**

Ontology projector was used to map ontology embeddings to the same feature space as image features, that enables semantic alignment between visual and knowledge-based features.

(c) **Domain Discriminator with GRL**

Employing a gradient reversal layer (GRL) with a domain discriminator enforces domain invariance.

Results

Table 7.4: Results of Image → Ontology Domain Adaptation

Dataset	Train	Test	Dom. Train	Dom. Test
MaleVis	99.98	87.01	52.01	52.01
Malimg	98.77	98.55	53.32	49.73
Virus MNIST	95.34	91.64	53.32	53.32

Table 7.4 is showing the results of the image to ontology domain adaptation experiments.

- The table presents that classification pipeline works fine for all three experiments.
- Domain train and test Accuracy for all datasets is around 50–53%. Therefore, the feature extractor is successfully confusing the domain discriminator.
- This indicates image features and ontology embeddings are aligned in the shared feature space.
- The objective of Domain adaptation is achieved for all three datasets. The discriminator can't distinguish the image and ontology features.
- In this experiment, Malimg dataset performed better than other two datasets as it produces high classification accuracy with good adaptation performance.

MaleVis Experimental Results

Figure 7.4 is showing the confusion matrix of MaleVis dataset.

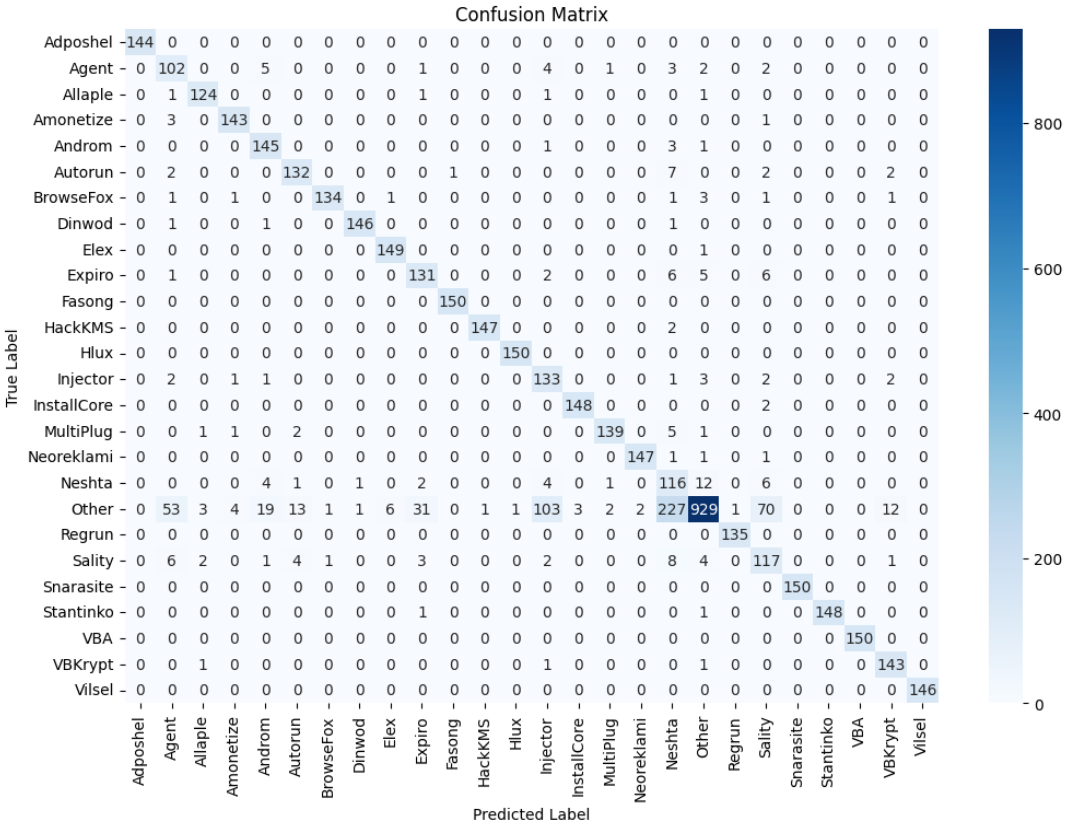


Figure 7.4: MaleVis Confusion Matrix

Virus MNIST Experimental Results

Figure 7.5 is showing the confusion matrix of Virus MNIST dataset.

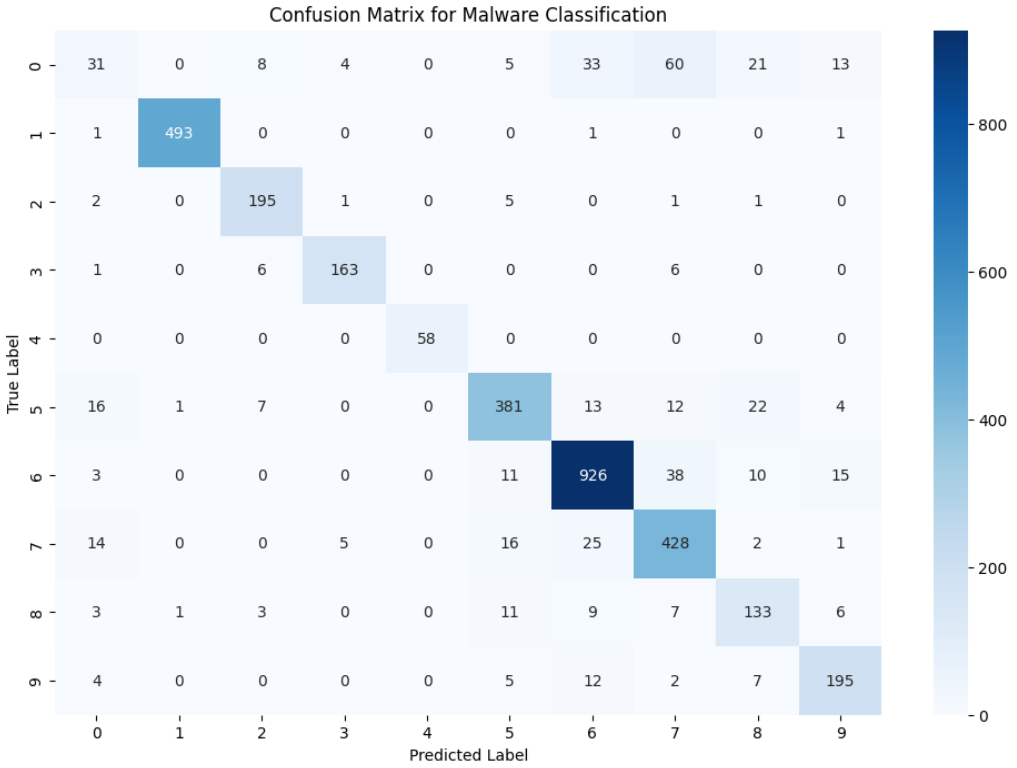


Figure 7.5: Virus MNIST Confusion Matrix

Malimg Experimental Results

Figure 7.6 is showing the confusion matrix of Malimg dataset.

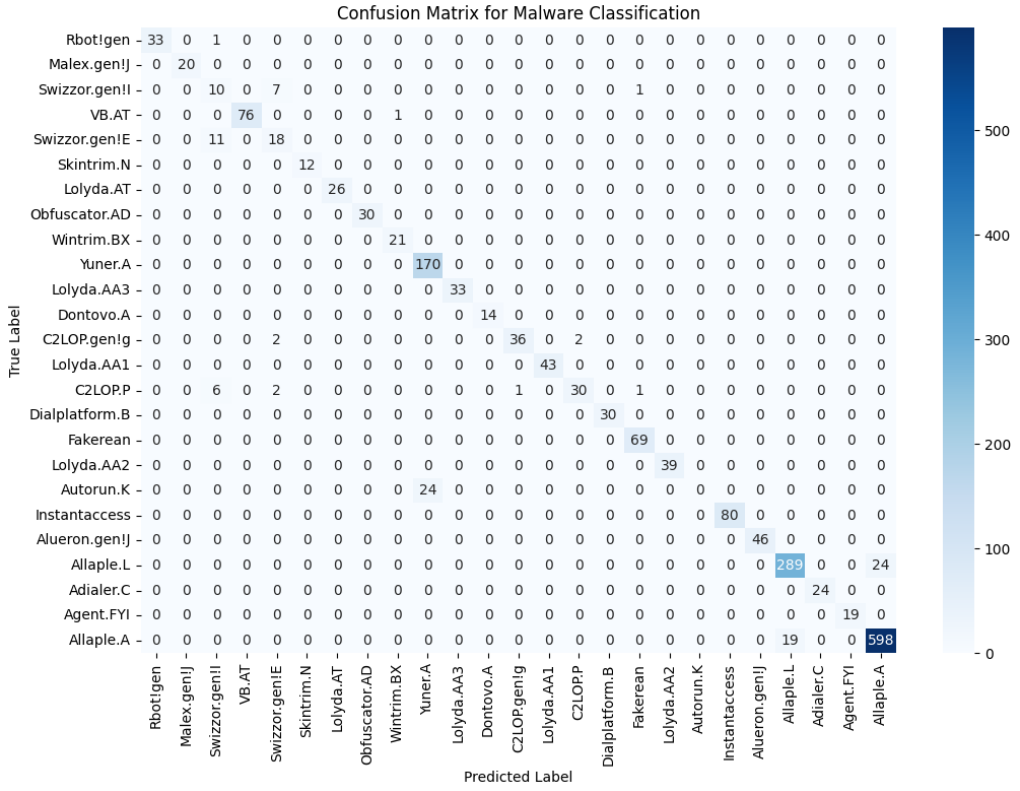


Figure 7.6: Malimg Confusion Matrix

This Domain Adaptation experiment manages to combine malware classification using deep learning with knowledge representation using ontologies to offer an enhanced form of malware detection. Using the malware image dataset as the source domain and ontology-based knowledge graphs as the target domain, the generalization, interpretability, and robustness of malware classification have been achieved in this framework.

These results confirm that structured knowledge combined with deep learning could solve some critical challenges in data dependency, interpretability, and adaptability within an evolving threat landscape of malware. This experiment hence proved a proof-of-concept for hybrid AI techniques in cybersecurity and moved one step further toward intelligent, interpretable, and adaptive malware detection systems.

Future work can be devoted to an update of the structure of the ontology, feature alignment across domains, and the generalization of this approach

for real-world threat detection.

7.4 Novel Malware Knowledge Graphs

This section presents advanced deep learning techniques combined with the power of ontological frameworks to contribute significantly to semantic image classification or on populating a malware knowledge graph related to cybersecurity. Therefore, this section presents one more effort using these technologies with contributions concerning enhanced cybersecurity threat identification and threats intelligence organization. This research investigates how deep learning can be combined with ontological knowledge representation to further improve the semantic classification of malware-related images for populating an extensive malware knowledge graph. Using the CNN based on the architecture of MobileNetV2, we realize the image classification into predefined malware behavior categories. These then will be integrated into a knowledge graph encoded in RDF/OWL and further enhanced with information about malware behaviors. This work narrows the gap between unstructured image data and structured semantic representations, improving the interpretability and usability of the results of classification in cybersecurity applications. In this work, the proposed method now promises to populate cybersecurity knowledge graphs more accurately and with contextual information.

This research has designed three malware knowledge graphs using one behaviour-based malware ontology by fusing it with three different malware datasets. A knowledge graph can feed lots of rich, structured data into machine learning models. Similarly, patterns in the graph could be used to train models to do a better job of malware detection and classification. This hybrid approach combines the strengths of deep learning (for image recognition) with semantic technologies (understanding relationships and context). Knowledge representation in the form of malware data allows for intuitiveness of understanding and reporting. This, therefore, will let the security analysts see the relations among various samples of malware, their behaviors, and visual signatures, thus communicating findings to stakeholders easily, as well as making decisions based on data. Here are some key aspects of the novelty of this experiment:

1. **Integration of Deep Learning with Knowledge Graphs:**
    - **Image Classification to RDF:** This experiment combines image classification using a deep learning model: MobilenetV2 with the construction of a knowledge graph. The results from the classification of images by the model populate an RDF graph, linking image instances to ontology classes.
    - **Automated Knowledge Graph Population:** The novelty of automatically populating a knowledge graph based on the predictions of image classification is that this allows dynamic and scalable Knowledge graph generation from visual data [250].
2. **Semantic Enrichment of Visual Data:**
    - **Use of Ontologies:** The experiment is designed by using an OWL ontology file to set the structural form of the knowledge graph. This ontology provides a formal framework for describing the various relationships among concepts, thus populating semantically enriched classified images.
    - **Connection Between Visual and Semantic Domains:** Using visual data with associated semantic concepts from the ontology creates a direct link between the visual or image classification domain and semantic or ontology-based domain [251]. Such a link enables further advanced queries and inferences to be derived from the visual data.
3. **Scalability and Flexibility:**
    - **Scalable Knowledge Graph Construction:** The approach is scalable, as it can be used on large datasets; with the processing and classification of new images, it automatically updates the knowledge graph.
    - **Flexibility in Ontology Use:** The method is flexible because it can be applied to different ontologies depending on the domain of interest. The same framework can then be adapted to various domains by changing the underlying ontology and the dataset used for training the image classification model.
4. **Visualization and Interaction with Knowledge Graphs:**

- **NetworkX and PyVis for Visualization:** The experiment leverages NetworkX to transform the RDF data into a graph format and PyVis to visualize the resulting knowledge graph. This visualization assists in understanding the relationships between classified images and the semantic concepts they represent.
- **Interactive Exploration:** With the generation of a knowledge graph that can be visualized and explored interactively, the users gain insight into the relationships between images and their classifications, perhaps not as readily apparent in traditional image classification tasks.

7.4.1 Methodology

The methodology of this experiment mainly includes the following parts: data preparation, model training, knowledge graph population, and visualization. In particular, each of the above-mentioned steps is described as follows:

1. **Dataset Preparation**
    - **Data Collection:** An ontology-based malware behavior knowledge graph was developed in Chapter 4 to capture the behavior of metamorphic malware. In this experiment, this ontology is used for integrating with three malware image datasets: Malimg, MaleVis, and Virus MNIST, respectively.

      This domain ontology was incorporated with the a benchmark metamorphic malware behaviour dataset[3] to produce the final output ontology[4] which is capturing the malware behaviour. Figure 7.7 and 7.8 are showing the behaviour of Trojan and Virus respectively.

[3] https://raw.githubusercontent.com/ocatak/malware_api_class/master/sample_analysis_data.csv
[4] https://github.com/ipsrychow/Malware-Ontology-Graph

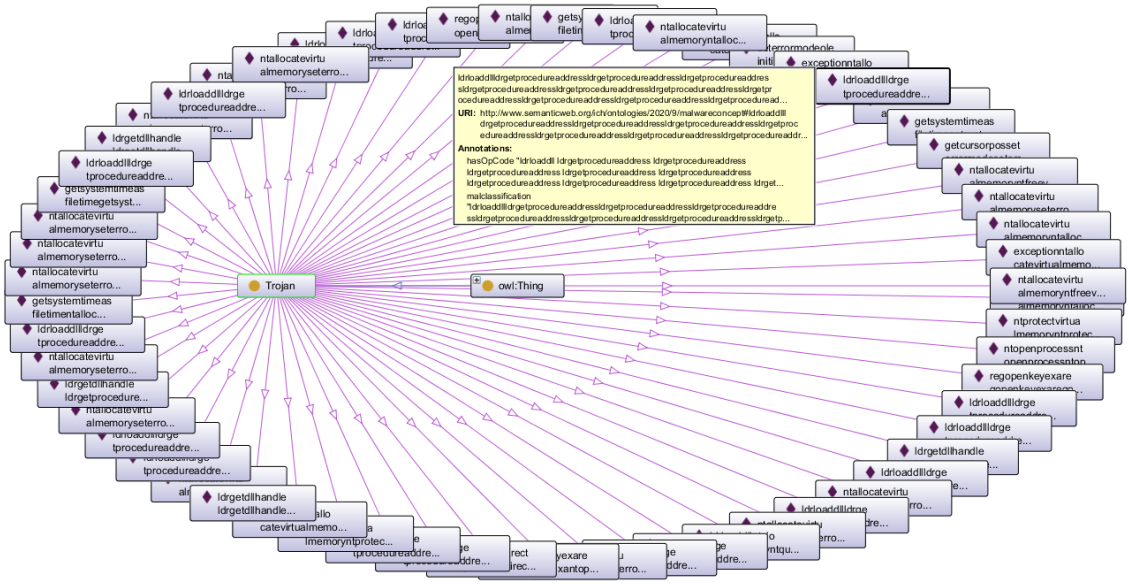


Figure 7.7: Behaviour of Trojan

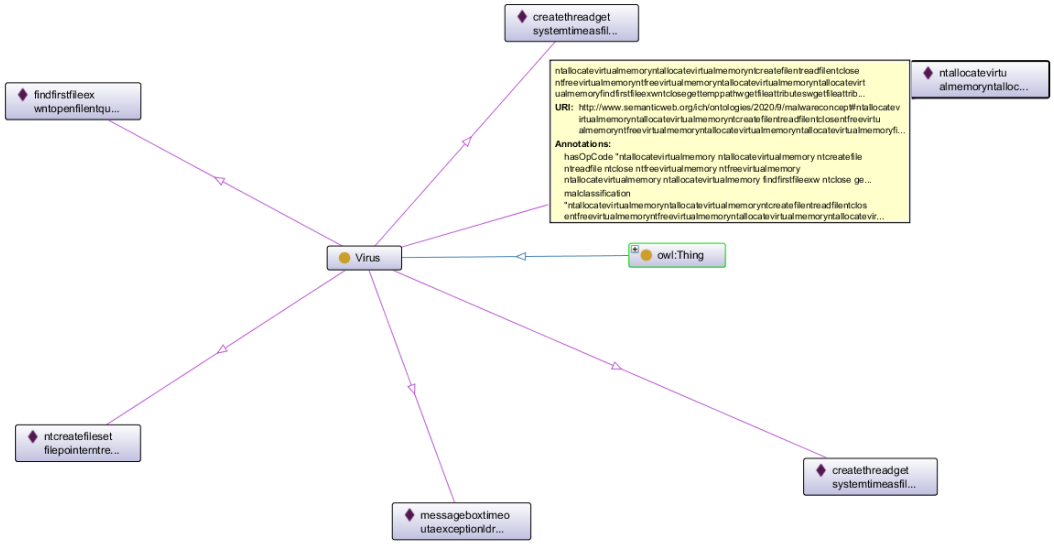


Figure 7.8: Behaviour of Virus

- In this work, malware image datasets were gathered and organized. Each image is categorized into the pre-defined malware behavior classes.

- **Data Preprocessing:** Resize images to a uniform size and normalize pixel values to the range [0, 1].

2. **Model Architecture and Training**
   - **Model Selection:** MobileNetV2 was selected for its efficiency and strength. Initialize it with weights pretrained on ImageNet, excluding the top layer.
   - **Model Customization:** Added a global average pooling layer and a dense layer with softmax activation to adapt the model for the specific classification task.

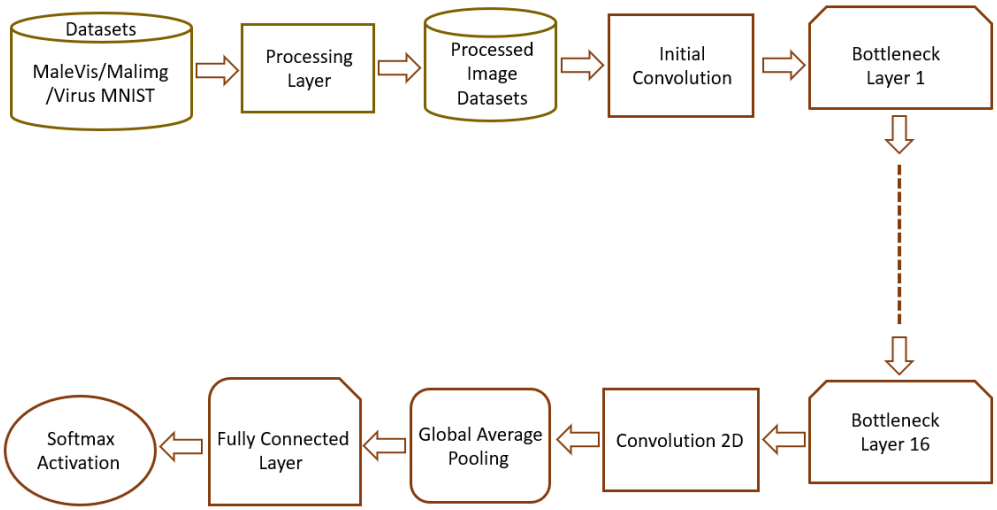


Figure 7.9: MobileNetV2 Architecture to process the Image Dataset

Figure 7.9 is showing the MobileNetV2 architecture for this experiment.

- **Compilation:** Compiled the model with the Adam optimizer and sparse categorical cross-entropy loss function.
- **Training:** Train the model on the training dataset and validate its performance on the testing dataset over some pre-defined number of epochs.

Table 7.5 is showing the classification results of three image datasets after training.

Table 7.5: MobileNetV2 Experimental Results

Results		
Dataset	Train Accuracy	Validation Accuracy
Malimg	87.98%	83.83%
Virus MNIST	85.39%	79.53%
MaleVis	95.67%	77.88%

The motivation of this experiment is to produce a meaning knowledge graph with ontology and malware image datasets. Because of this reason, this image classification result (Table 7.5) was not compared to the other experimental results in previous chapters.

3. **Image Preprocessing for Prediction Image Loading:** Design a function for loading images to be predicted and ensure standardization in image size and normalization.
4. **Knowledge Graph Construction:**
    - **RDF Graph Initialization:** Initializing a RDF graph with the help of RDFlib library.
    - **Ontology Loading:** Load the ontology from an OWL file to an RDF graph.
    - **Graph Population:** Populate the RDF graph with nodes representing images and edges representing predicted classes along with their relations to broader malware behavior ontology nodes. Figure 7.10 is showing the knowledge graph population for the experiment.

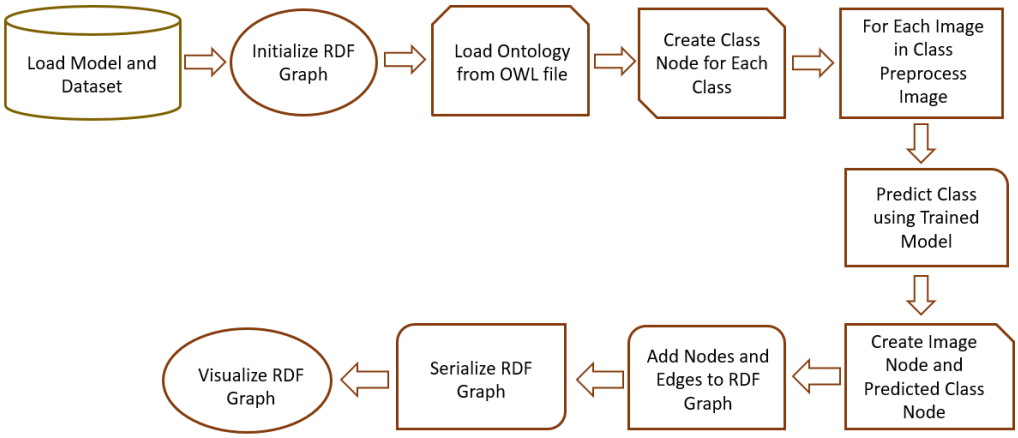


Figure 7.10: Knwledge Graph Population to create the ontology with Image Dataset and .owl file

5. **Knowledge Graph Visualization:**
   - **Conversion to NetworkX:** The RDF graph is converted into a NetworkX graph to facilitate its visualization.
   - **Graph Visualization:** Visualize the NetworkX graph using PyVis library by showing nodes and edges along with their respective labels and relationships.
6. **Integration and Analysis**
   - **Semantic Integration:** Include the malware behavior data in the knowledge graph along with image classifications for better semantic analysis.
   - **Evaluation and Analysis:** Assess the performance of the model and analyze the populated knowledge graph to gain insights on malware behavior and detection.

Figure 7.11 is showing the whole architecture of the experiment.

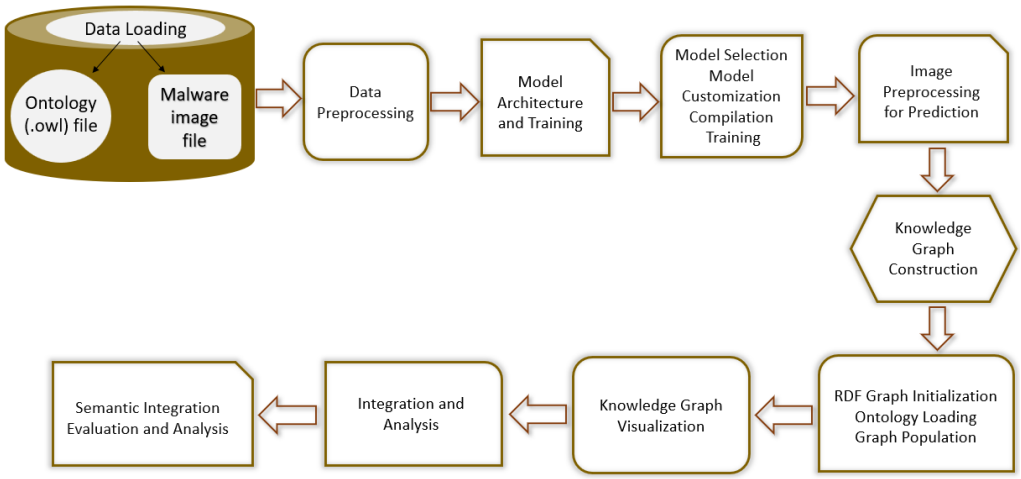


Figure 7.11: Architecture of the Experiment for Novel Knowledge graph Generation

The experiment was built according to Algorithm 21 with terms and notations were described in Table 7.6.

Algorithm 21 Malware Image Classification $\rightarrow$ Ontology Integration

Input: Dataset D, Ontology O, base model M, epochs E, batch size B, image size (H, W)
Output: Trained model M^*, RDF graph G
Load ontology O into RDF graph G
Extract classes $\mathcal{C}$ from D and encode labels
Split D into D_{train}, D_{test}
Define preprocessing preprocess$(\cdot)$
Initialize model M with pretrained weights
Add classification head
Compile M with optimizer, loss, metrics
for epoch $= 1 \ldots E$ **do**
  Train M on D_{train}, validate on D_{test}
end for
for each image $x_i \in D$ **do**
  $y_i^{pred} = M^*(\text{preprocess}(x_i))$
  $img_i = \text{URIRef}(\text{image\_id})$
  $cls_i^{pred} = \text{URIRef}(\text{predicted\_class})$
  $cls_i^{true} = \text{URIRef}(\text{true\_class})$
  Add triples: $(img_i, rdf:type, cls_i^{pred})$
  Add triples: $(cls_i^{pred}, rdf:type, cls_i^{true})$
  Add image data URI
end for
Serialize RDF graph G
Return M^*, G

Table 7.6: Description for Image Classification to Ontology

Symbol / Term	Description
D	Image dataset containing malware images and labels
D_{train}, D_{test}	Training and testing subsets of D
$\mathcal{C}$	Set of class labels in D
M	Base CNN model (e.g., MobileNetV2)
M^*	Trained CNN model
E	Number of training epochs
B	Batch size for training
(H, W)	Height and width of input images
x_i	Individual input image from D
y_i^{pred}	Predicted class label for image x_i
y_i^{true}	True class label for image x_i
O	Ontology file (OWL/Turtle)
G	RDF knowledge graph integrating images and ontology
img_i	RDF node representing image x_i
cls_i^{pred}	RDF node representing predicted class of x_i
cls_i^{true}	RDF node representing true class of x_i
$imageDataURI$	Base64-encoded image stored in RDF
preprocess$(\cdot)$	Function to resize, normalize and augment images

Results and Discussion

This research produced three novel knowledge graphs with malware behaviour based ontology and malware image files using deep learning. Here are the visualizations of the knowledge graphs:

- **MaleVis Dataset with Ontology Visualization**

  The MaleVis dataset[5] is designed for vision-based malware recognition and includes images generated from 25 malware classes and 1 legitimate software class. It offers 9,100 training and 5,126 validation RGB images in resolutions of 224x224. The dataset can be used with various deep learning frameworks.

  Figure 7.12 is showing the images of a Class ("Allaple") of MaleVis dataset with a Class ("Worm") of the malware behaviour ontology. Figure 1.1 is showing all the classes of MaleVis dataset and malware ontology.

[5] https://web.cs.hacettepe.edu.tr/~selman/malevis/

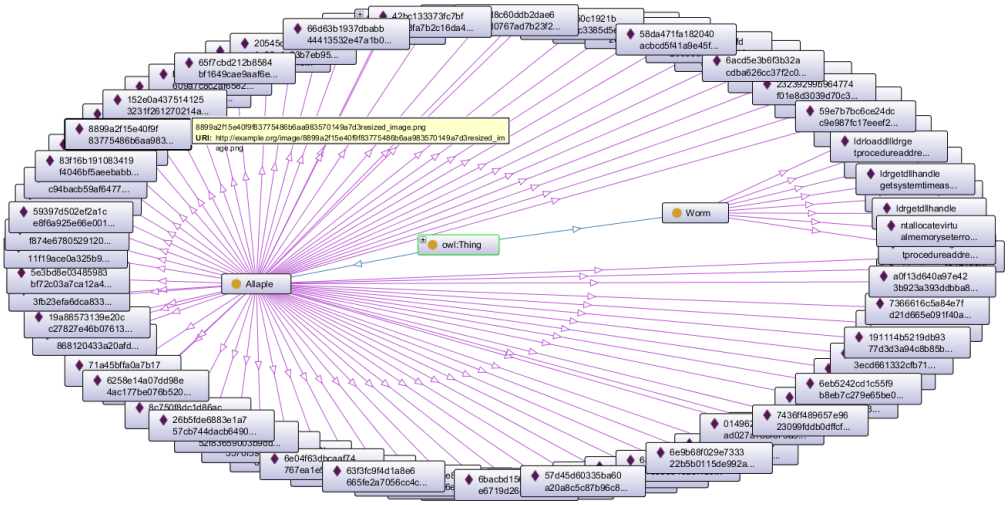


Figure 7.12: Visualization of MaleVis Dataset with Malware Ontology

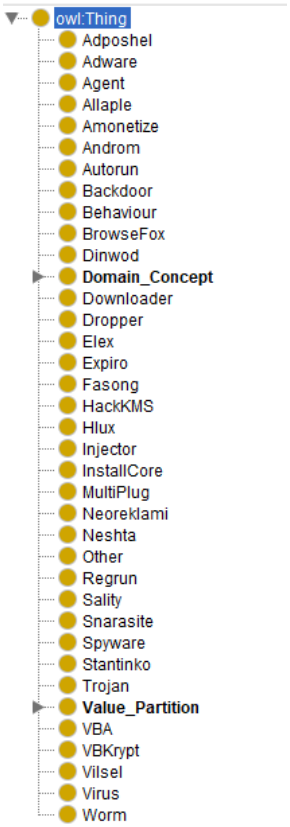


Figure 7.13: Classes of MaleVis Dataset with Malware Ontology

- **Virus MNIST Dataset with Ontology Visualization**

  The Virus-MNIST dataset [178] comprises grayscale images that were created from malware files designed for training and testing of machine learning models in malware detection. This dataset is structured similarly as newer MNIST dataset that includes images from virus binaries. It provides an avenue for both researchers and practitioners to study all the various techniques of cybersecurity classification.

  Normally, all images are of size 32x32 pixels to allow the employment of most standard image classification models. Virus-MNIST constitutes images corresponding to some malware family or type, with a class corresponding to one type of malware. Just like the MNIST for classes per number from 0 to 9. Figure 7.15 is showing the images of a Class ("9") of Virus MNIST dataset with a Class ("Virus") of the malware behaviour ontology. Figure 7.14 is showing all the classes of Virus MNIST dataset and malware ontology.

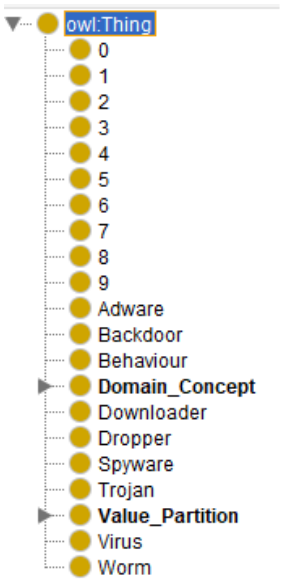


Figure 7.14: Classes of Virus MNIST Dataset with Malware Ontology

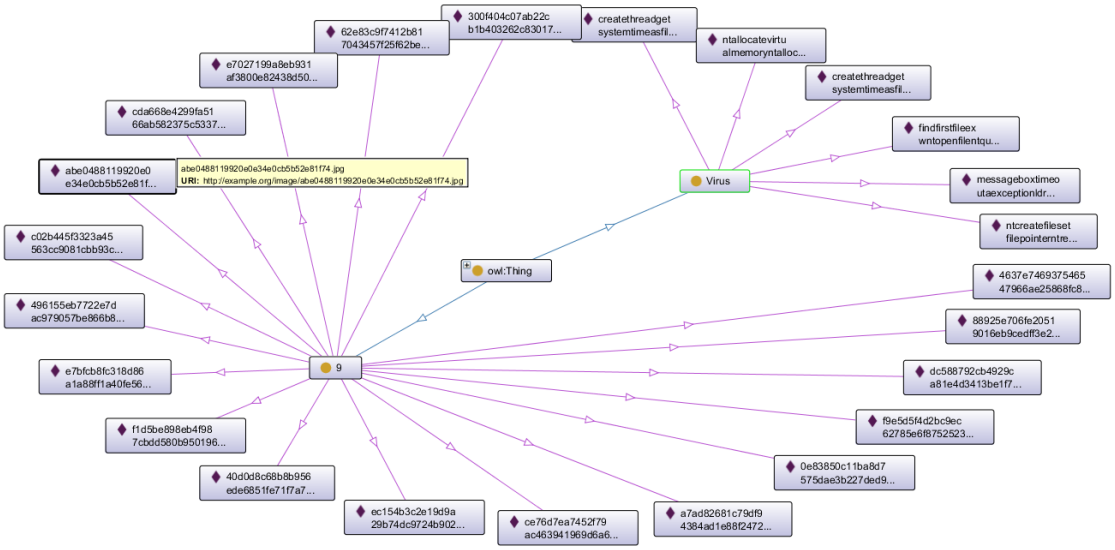


Figure 7.15: Visualization of Virus MNIST Dataset with Malware Ontology

- **Malimg Dataset with Ontology Visualization**

  The Malimg dataset[6] is one of the popular datasets used in malware classification. It contains 9,339 grayscale images representing 25 different malware families. The images in the dataset are generated from malware binary files and this gets converted into visual representations, which makes it useful for machines learning and deep learning applications in cybersecurity.

  Figure 7.17 is showing the images of a Class ("Lolyda.AA1") of Malimg dataset with a Class ("Virus") of the malware behaviour ontology. Figure 7.16 is showing all the classes of Malimg dataset and malware ontology.

[6] https://www.dropbox.com/s/ep8qjakfwh1rzk4/malimg_dataset.zip?dl=0

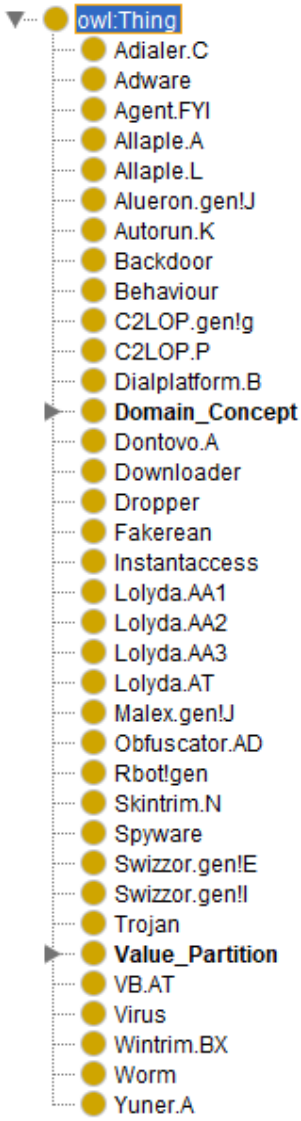


Figure 7.16: Classes of Malimg Dataset with Malware Ontology

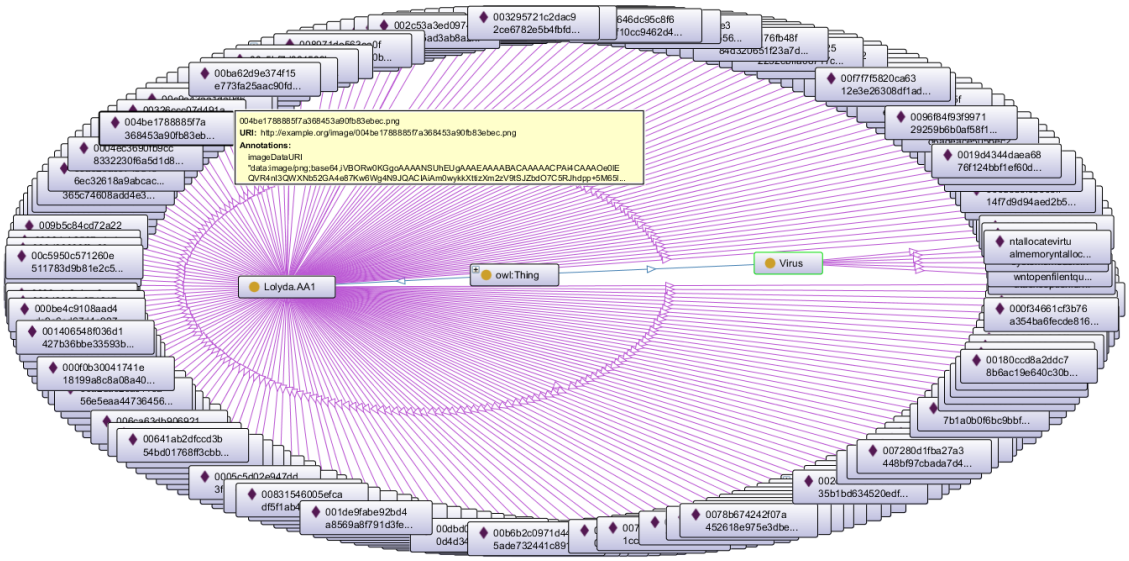


Figure 7.17: Visualization of Malimg Dataset with Malware Ontology

Evaluation Results

In this experiment, the knowledge graphs were saved in '.rdf' format. Evaluation was done with a neural network model. Here are the steps:

1. **Loading Ontologies:** The notebook loads different ontologies represented in RDF (Resource Description Framework) format.
2. **Generating Embeddings:** It generates RDF2Vec embeddings for the entities in the ontologies. These embeddings represent the entities in a vector space.
3. **Preparing Labeled Data:** It prepares labeled data from the embeddings for a binary classification task.
4. **Creating Graph Data:** It constructs graph data structures for the use with neural networks.
5. **Model Definition and Training:** It defines, compiles and trains a neural network model (e.g. an LSTM model) on the graph data.
6. **Evaluation:** The model is evaluated with training and validation accuracies.

This chapter developed three novel ontologies with malware image and behaviour dataset. These ontologies and another open source malware ontology (MALOnt ([252])) were evaluated with the LSTM network.

Table 7.7: Evaluation Results

Results	**MaleVis Ontology**	**Virus MNIST Ontology**	**Malimg Ontology**	**MALOnt**
Training Accuracy	53.62%	51.90%	53%	**63.79%**
Validation Accuracy	**50.93%**	**50.92%**	**50.86%**	38.16%

Our developed ontologies produced more Validation Accuracy than 'MALOnt'. These three malware image and behaviour ontologies contain more data points than 'MALOnt'.

Shared knowledge graphs help an organization to share useful information across their network and promote collaborative Threat Intelligence. The nature of the semantic graph ensures that the data will keep on representing meaningful relations even outside an individual environment, which simplifies further the task of various teams to grasp the data underlying the malware for much-needed coordinated defense.

Knowledge graphs will make the automated security tools more capable of performing threat detection, correlating events with security, and automated response actions. The structured data in the knowledge graph makes informed decisions easier to reach.

Malware Strain Identification

Malware image datasets were trained by the MobileNetV2 model to categorize different malware strains with their corresponding class labels. Based on the visual patterns the model learns to classify the malware images according to its predefined classes. Here are the steps involved:

1. **Ontology Integration:**

The RDF graph that was created using OWL file was structured to present different malware families and their behaviours (API call sequences). The ontology was structured to represent different malware strain knowledge as it can capture the relationships between the classes and behaviours.

2. **Populating the Knowledge Graph:**

   The model is able to predict different malware classes (strains) to which it belongs after training and processing. Images were added to the knowledge graph with corresponding nodes. The nodes were enriched with information like image's base64 data URI for visualization after linking to the strain classes.

   RDF triples represent the relationship between the images and predicted malware strains.

3. **Behavioural and Strain Differentiation:**

   Malware strains can be identified and differentiated by the knowledge graph depending upon visual patterns (learned by the deep learning model) and predefined behaviors (encoded in the ontology). The ontology's hierarchical structure includes the predicted malware strains linked with associated behaviours so the system can identify visually similar malware strains based on their behaviour.

The current experiment is not directly performing the prediction of new generation malware classes, however, it can efficiently capture and map the malware behaviour and malware image datasets. Therefore, the proposed framework implements a strong foundation for predicting new-generation malware classes by aligning image features and ontology embeddings within a shared semantic space which will allow the framework for generalization to unseen or evolving malware behaviours.

7.5 Conclusions

The developed experiment on Domain Adaptation shows the integration of an ontology file with malware image datasets. It connects structured ontological data with unstructured image data through a semantic bridge, thereby enhancing complex queries and deeper insights from both domains. This exper-

iment demonstrated that such integration is feasible and profitable, enabling enhanced semantic understanding and the possibility of exploiting these connections for a variety of applications, such as semantic search, automated reasoning, and data enrichment.

This approach has underlined the need for careful alignment among different data sources, detailed annotation of data, and enrichment with semantic metadata. Though compatibility and scalability challenges are there, this experiment will open the way for sophisticated domain adaptation techniques that could enhance knowledge discovery and interoperability across diverse datasets.

One of the effective methods for domain shift involves domain adaptation CNNs, which allow a model trained in one domain to generalize to another. Alignment of features, adversarial training, and transfer learning are effective approaches that empower CNNs with the capacity to adapt to new domains with very few labeled data. This is thus very helpful in practical applications where domain-specific data could be scarce or very different from the source domain. Integrating ontologie files can further enrich the adaptation process by incorporating domain-specific knowledge into the learning process of the CNN.

The combination of Ontology (or Knowledge graph) with Deep Learning to develop a malware knowledge graph using malware image datasets will try to contribute a new knowledge based and data driven framework in the security domain. Previous malware knowledge graphs [253] and [254] presented threat intelligence and based on RDF triples.

In this research, the approach was taken to develop a knowledge graph from a benchmark malware dataset by integrating deep learning and knowledge graph based algorithms. This malware knowledge graph will be helpful

- To provide malware knowledge or cyber knowledge in a structured format to accommodate massive volumes of cyber data.
- Relationships among the entities and attributes of the image dataset will facilitate the machine to learn automated reasoning on malware knowledge.
- As cybersecurity domain is always heterogeneous and complex a well-structured knowledge representation framework will be able to allocate huge amount of cyber information.

- As Knowledge graph is a network infrastructure it can be used to understand information flow, new malware knowledge, attack patterns also fusion of other data.
- Incorporation of deep learning to develop a malware knowledge graph will facilitate the anomaly detection method to understand anomaly patterns and anomaly description.
- An image based malware knowledge graph in cyber security domain will be a structured format to have knowledge about malware image identification, hash values and pixel values.

A knowledge graph can store historical data, which would allow for analysis of malware trends over time. The historical perspective will be of great use in predicting future threats and understanding how the TTPs (tactics, techniques and procedures) of malware have changed [255]. This Chapter presents a new approach to improving cybersecurity intelligence by taking advantage of deep learning for semantic image classification and using ontology for populating the malware knowledge graph. These technologies, when put together, provide unparalleled detection and contextualization of cyber threats.

Our experiment is novel in constructing the knowledge graph by integrating deep learning-based image classification with semantic web technologies, particularly the construction of the RDF-based knowledge graph. This approach allows the experiment to bridge the gap between unstructured visual data-images-and structured, machine interpretable knowledge representations-knowledge graphs. The novelty of this experiment is the seamless integration that will be achieved between a task of image classification and semantic web technologies for constructing a knowledge graph. Such an approach will, apart from automating the population of a knowledge graph, enrich it with semantic relationships for powerful advanced information retrieval, analytics, and decision-making upon varied domains. This knowledge graph that would result can be queried using SPARQL or any other query language, which allows the construction of very complex queries based on the visual content and semantic relationships defined by the ontology.

It also integrates malware behaviour with the images in a knowledge graph to finally provide a rich dataset for further research. These fully open new ways of researching how visual characteristics and behavioral patterns of mal-

ware are connected and therefore can be useful when building new cybersecurity technologies and methodologies.

8 Discussions

8.1 Malware Behaviour Ontology

In Chapter 4, the research addresses a critical aspect of malware detection by implementing an ontology-driven framework to capture dynamic malware behaviour, with a distinct focus on metamorphic malware. In the ever-evolving era of Big Data and IoT, understanding new generation malware behaviour is a complicated but crucial task for security domain. Common detection methods, such as signature based and anomaly based techniques, cannot cope up properly with the rapidly evolving nature of anti-detection malware.

Malware writers use sophisticated techniques like code obfuscation, encryption, decryption and translation, rewriting to generate polymorphic and metamorphic malware samples. These malware generations can continuously alter their code in every iteration to evade traditional detection mechanisms, making them extremely challenging to detect by the conventional systems. Different malware families exhibited their behaviour by producing absurd opcodes which have been captured by the dynamic and adaptable ontology-based model proposed in Chapter 4.

The proposed ontological approach is novel due to its structured and formal representation of malware behaviour knowledge. The existing signature-based and anomaly-based detection methods show a number of limitations like, false positive rates, feature selection for training, time complexity and incapable to detect new attack. This dynamic ontological model can be expanded and adaptable to accommodate new datasets like, API call sequences, malware images etc.

Chapter 4 highlighted the use of ontology-oriented programming, enabling real-time data integration and updating of malware behaviors, which provides a level of automation and adaptability which is missing from many other approaches. Other ontological frameworks like, MALOnt has provided a constructive architecture of threat intelligence which includes information about different cyber threat use cases. But didn't include any real-time malware datasets which limits this frameworks to handle new generation malware variants with advanced evasion techniques. In comparison, our proposed framework is designed to capture complex malware behaviour, by adding new individuals in the model, making the architecture highly adaptable for the cyber security datasets.

The significant contribution in this ontology is to incorporate metamorphic malware behaviour in the form of sequential API calls. This ontology presents a concrete and novel understanding for malware behaviour through the changes of absurd opcodes in every iteration with the morphing engine. This framework could help the cyber security professionals to develop sophisticated detection mechanisms. Future enhancements could include continuous refinement and expansion of this ontology to integrate more real-time datasets. This ontology could adapt the continuous emerging threats and help in identifying and mitigating advanced malware related threats.

The comparison with existing ontologies like MALOnt and Swimmer's Ontology shows that the proposed ontology performs well in terms of attribute richness, class richness, and inheritance richness. However, continuous evaluation and refinement are necessary to ensure the effectiveness of the ontology to capture more dynamic and complex behaviour and exploring its integration with other cyber security datasets. Chapter 7 of this thesis has a contribution towards more expanded versions of this ontology.

8.2 Deep Learning based Techniques

In Chapter 5, the experiments reflect the transformative nature of AI in malware classification. In this section, some of the key insights were found by using these experiments on advanced deep learning architectures, particularly their strengths and limitations as well as the opportunities in the application

of these advanced architectures in cybersecurity.

- **Model Performance and Generalization:** The deep learning models used are ResNet-50, CNNs, which can highly capture patterns within the images used in these datasets. High performances between the training and testing phases also represent generalization by the model, though there's a slight decline of performance at testing, indicating an opportunity for the refinement of further modeling.
- **Feature Extraction:** Automated feature extraction has made deep learning models excel where earlier models, depending on the necessity of feature engineering, struggled in the earlier domain-specific process. This ability could be easily noticed in experiments when models would perfectly identify patterns hidden in malware images. It was possible because hierarchical features, which may range from edges to more complex textures, helped classify such finely nuanced data in deep networks.
- **Architectural Advancements:** The experiments utilize state-of-the-art architectures, which are scalable and efficient when dealing with high-dimensional data. Through residual connections, ResNet-50 was able to counter the vanishing gradient problem and allowed deeper networks to learn. Meanwhile, CNNs captured spatial and structural dependencies very effectively, suited for image-based classification of malware.
- **Data augmentation and synthetic data:** Enriched datasets were produced by using data augmentation techniques and synthetic data generation via GANs in order to alleviate class imbalances and data scarcity. GANs not only augmented the datasets but also increased model robustness by exposing networks to diverse data distributions. Such an aspect is very important for cybersecurity, in which the nature of malware makes it dynamic; hence, it requires adaptable yet robust models.
- **Challenges faced:** Several challenges were also encountered despite these promising results: overfitting, especially in smaller datasets, indicates that regularization techniques and more effective strategies in augmenting data are required.

  Overall, deep learning experiments depicted enormous potential that sophisticated AI techniques bring to automation as well as better malware detection. Although results seem promising, further exploration

on the innovative architecture of models, effective training strategy, and adversarial robustness becomes necessary as well to be capable of evolving and overcoming security issues in cyberspace.

In Subsection 5.2.12, the experiment involved, Residual Networks (ResNet-50) with an Attention Module were used to classify malware images from the datasets using PyTorch. The model adopted convolution layers, combined with batch normalization, residual connections, for extracting the features and also classifying them. Results showed that ResNet obtained high accuracy, thereby explaining its efficiency in terms of distinguishing between different malware types.

1. **Channel Attention and Spatial Attention:**
   - **Channel Attention:** The experiment includes a mechanism of channel attention, a technique that emphasizes the network sensitivity to the most informative feature maps. This mechanism basically aggregates spatial information across each channel employing global average pooling and then allows a Multi Layer Perceptron (MLP) learning a set of weights to pinpoint important channels of interest. Weighted feature map results are derived and applied where significant features in the given feature maps amplify while suppressing less relevant ones. This helps the model focus on which channels (features) are more critical for classification, thus enhancing the overall feature represented.
   - **Spatial Attention:** Along with channel attention, a spatial attention mechanism was introduced to aid the model in determining the most relevant spatial regions within an image. This mechanism applies convolutional layers to the input feature maps to generate a spatial attention map that highlights important regions in the image. This model focuses more on spatially important regions that can distinguish more between classes in the presence of complex patterns or backgrounds in the malware images. This approach becomes very helpful whenever specific spatial patterns are used as indicators of some malware types.
2. **Impact of Attention Mechanisms on Model Performance:**

- The inclusion of modules of channel and spatial attention in ResNet helped focus on the vital features and localizing the input regions, resulting in better discrimination ability. Those mechanisms enhanced good feature extraction of images by their focus on a few critical areas, which brought accuracy in many classification tasks into focus.
- The model might learn more robust representations of the malware images by dynamically adjusting the focus on different channels and spatial regions during training. This adaptability is particularly valuable in malware classification, where subtle differences in texture, shape, and structure can be the difference between classes.

3. **Model Training and Evaluation:**
   - The experiment utilized stochastic gradient descent (SGD) with momentum to train the model, along with a learning rate scheduler to fine-tune the learning process. The use of attention mechanisms added minimal computational overhead while significantly enhancing the model's performance.
   - The model's evaluation showed high accuracies, indicating that the attention mechanisms successfully contributed to improving the classification results. The attention-enhanced model was able to identify and focus on the most informative features, leading to better generalization on the test set.
4. **Visualization and Insights:**
   - Visualizations of the model's performance, including training and test loss, as well accuracies, provided insights into how the attention mechanisms influenced learning. The consistent reduction in loss and improvement in accuracy over epochs demonstrated the effectiveness of incorporating attention into the model.
   - Visual testing of test images with their true and predicted labels were also presented to show that difficult cases are accurately classified by the model. A qualitative evaluation Highlighted the attention mechanism and its importance in extracting significant features in images.

In summary, the experiment well showed the usability of channel and spatial attention integration in a ResNet architecture that would be in malware image classification. These channels of attention actually enhanced the strength of the feature focusing ability while the model exploited the spatiality of regions significantly to improve classifications. This underlines the significant role of possible applications of using attention mechanisms, deep learning applied in cybersecurity-based approaches, refined ways of defining and classifying malware.

8.3 Deep Learning with Continual Learning

In Subsection 6.1.2, the experiment investigated the use of a U-Net-based architecture for a CL classifier. The U-Net model acted as a feature extractor, whose output was fed into a simple fully connected network to classify images across multiple tasks. The experiment aimed to evaluate the classifier's performance using transfer learning on a dataset of images while employing techniques like Elastic Weight Consolidation (EWC) to mitigate catastrophic forgetting, a common issue in continual learning.

In the case of image classification, U-Net-the model originally devised for semantic segmentation was reemployed here by its encoder-decoder structure to derive deep features from the images. A classifier trained on a tasks-divided data set was allowed to retain prior task knowledge as the model would use EWC to update a model with the addition of new tasks. This will be achieved by retaining the learned parameters of the previous tasks while adding a regularization term to the loss function.

The results of this experiment proved that the architecture of the U-Net model, combined with continual learning strategies, yields good classification accuracy over tasks. It is evidenced in the smooth and steady increase of performance throughout training. However, while the experiment showed promising results, there are some areas that could be further improved:

- **Performance Enhancement:** The accuracy of the classifier could be possibly enhanced with extensive hyperparameter tuning, a more sophisticated classifier network, or through using a much larger dataset that better captures the variability.

- **Model Complexity:** The complexity of the U-Net model itself, particularly the encoder-decoder structure, would have to be optimized in terms of the actual segmentation task as compared to just using the classifier model.

Experiment also demonstrates a potential integration point of the U-Net into a Feature extractor within the CL framework. Future work may include using more extensive datasets, extended EWC tuning, and the use of other advanced continual learning algorithms to improve a robust classifier model.

In Subsection 6.1.1, the experiment stresses the potential advantages of combining GANs with continual learning frameworks. One important benefit of applying GANs in this case is the possibility of generating further training samples, especially when there are limited amounts of data available. The GAN successfully produced synthetic samples that supplemented the original dataset, preventing overfitting and allowing the classifier to better learn complex patterns.

However, several challenges exist. The quality of GAN-generated images has much importance in determining the success of data augmentation; noisy training data can degrade classifiers' performance if images are poorly generated. Furthermore, fine-tuning both the parameters of GAN and the classifier is crucial; producing suboptimal configurations may deter convergence or unstable training.

Moreover, the continual learning classifier employed elastic weight consolidation (EWC) to mitigate catastrophic forgetting. The use of EWC helps preserve knowledge from previously learned tasks by penalizing changes in the model's important parameters. While this approach worked well in this experiment, scaling it to more complex tasks or datasets might require further optimization of the regularization hyperparameters and careful selection of Fisher information updates.

Overall, the experiment demonstrates good results of a combination of GANs with continual learning. Future work might be focused more on sophisticated techniques for balancing real and synthetic contributions and improving stability and scalability within the learning framework.

The experiment demonstrated the use of GANs for data augmentation and enhancing a classifier's performance in a setup of continual learning. The gen-

erator was trained very well to output new, real images from noisy random inputs; these images formed the augmented version of the data set. Thus, the same classifier was updated on both data sets: one the original and the other the augmented by GAN, which improved generalization and robustness.

The performance of the classifier, measured both in terms of training loss and accuracy, suggests a steady improvement, which means the augmentation strategy improved the learning process. The utilization of GAN-generated samples adds more training data to the model, allowing the model to pick up on features that are generally hidden in the dataset. The test accuracy, as reported at the end of the experiment, indicates the model generalized very well to unseen validation data and reinforces the effectiveness of the approach with data augmentation.

In Subsection 6.1.3, the experiment focused on the use of continual learning techniques, such as Elastic Weight Consolidation (EWC), for malware classification across three different datasets: Malimg, MaleVis, and Virus MNIST. All the datasets were challenging due to their image styles, resolutions, and class distributions. The experiment used ResNet-50 architecture, which can handle catastrophic forgetting in sequential learning tasks while maintaining robust classification performance. Key findings include:

- **Continual Learning with EWC:** The introduction of EWC offers a significant innovation in preserving knowledge across multiple tasks. The model penalizes the deviation from the parameters that are important to the prior tasks, as calculated from the Fisher Information matrices. The method is highly effective in alleviating catastrophic forgetting, which typically affects traditional deep learning models trained on sequential tasks. The experiment demonstrates that after training on the subsequent datasets, the model still enjoys high accuracy for earlier tasks and therefore reflects strong knowledge retention.
- **Unified Model for Diverse Datasets:** All malware sample classification is now performed using only one ResNet-50 on datasets with fundamentally different characteristics: Malimg metamorphic malware focuses on visual patterns, MaleVis on real obfuscation and Virus MNIST on very fine-grained classification of artificial malware images. Consistent good performance across those datasets shows generalization and adapting to different kinds of malware images.

- **Use of Transfer Learning:** The experiment uses a pretrained ResNet-50 architecture by exploiting the convenience of transfer learning, which actually does considerably speed up convergence and allows for higher performance. Such a method also balances out lower computational overhead associated with training from scratch and benefits considerably from features learned on large-scale natural-image databases. This innovative application of transfer learning in malware image classification represents a step forward in applying computer vision techniques to cybersecurity challenges.

8.4 Combination of Ontology and Deep Learning

In Chapter 7, we successfully developed a Domain Adaptation CNN in Subsection 7.1.1 by integrating the ontology file with the three malware image datasets, respectively. These datasets respectively underwent understanding of the ontology structure, annotating the image dataset, converting annotations into RDF format, and linking these annotations to relevant concepts within the ontology. This integration allows knowledge to be presented uniformly from both Domains, in addition to the improved semantic capabilities to facilitate querying and insights into the datasets.

The experiment highlights several key points in the context of integration with domain adaptation:

- **Ontology and Dataset Alignment:** Predefined ontology Integration with a dataset from another domain is aligned carefully in terms of concepts and properties. In this experiment, the integration was completely dependent on meaningful links found between the ontology classes in the ontology file and the annotations in the malware image dataset. This would involve manual mapping and domain expertise, which is challenging and time-consuming.
- **Scalability and Automation:** As the complexity and size of datasets grow, scalability becomes a concern. Automating parts of the annotation process, for instance, using machine learning models for image classification, could facilitate the creation of larger and more detailed frameworks.

- **Challenges and Limitations:** One of the major challenges of this experiment was to ensure the semantic compatibility between ontology and image database. Granularity, terminology, and conceptualization might vary across the entities and thus tend to cause challenges of integration.

This experiment proves the viability and benefits of a Domain Adaptation-based integration combining ontological data with images, as developed here. Such integration is to increase the possibilities of deriving meaningful insights across domains toward more advanced semantic analysis and knowledge discovery.

In section 7.4, the integration of deep learning with ontologies alleviates the malice of dealing with malware identification and analysis processes in several manners. First of all, an approach classifying malware-related images is based on MobileNetV2, which integrates visual data lacking in traditional methods of signature-based approaches. Secondly, the inclusion of ontologies provides semantic enrichment so that image data can be brought into predefined categories relating to malware behaviors and form the comprehensive knowledge graph. This approach of duality provides a deeper contextual understanding of malware, closing the gap between unstructured image data and structured semantic representations.

It further enhances the interpretability and usability of classification results. Malware data visualized in a knowledge graph helps security analysts to see relationships between various malware samples, behaviors, and visual signatures. It helps make more effective decisions and response during incidents.

The development of knowledge graphs in section 7.4 represents a significant leap in the area of cybersecurity, especially in representing how this kind of merging in deep learning with ontological framework can create a comprehensive and semantically enriched malware knowledge base. Implementation of a multi-dimensional perspective in understanding and analyzing cyber threats can be offered by the knowledge graphs equipped with malware behavior and image data.

- **Semantic Enrichment and Contextual Understanding:** Combining ontology with deep learning-based image classification allows the knowledge graph within this study to surpass just mere categorization; it brings

in a layer of semantic enrichment that creates more nuanced relationships between malware entities. Using RDF/OWL for the construction of the knowledge graph allows for the structured representation of both visual data and behavioral characteristics. Such semantic richness gives in-depth contextual understanding and makes it easier to trace relationships between different malware samples, their behaviors, and visual signatures.

- **Dynamic and Scalable Knowledge Graph Population:** The experiment presents an automated procedure to populate the knowledge graph using deep learning-based image classification results. This dynamic method not only facilitates the scalability to accommodate large data but also ensures the freshness of the knowledge graph due to the ongoing discovery of new malware images and their behaviors. The ability to update and expand knowledge graph automatically, with minimal intervention, is quite important for updating the landscape of cyber threats rapidly.
- **Improving Interpretability and Usability:** One of the primary advantages of this approach is that it makes the malware data more interpretable. The visualization of malware behavior and image data within a knowledge graph can help security analysts identify interconnections and patterns that would not be readily visible through traditional methods. The combination of deep learning outputs with ontological structure gives a more intuitive understanding of complex malware relationships, thereby making the communication of findings better and enabling more informed decision-making.
- **Facilitating Advanced Queries and Inferences:** Knowing that the knowledge graph is semantics based which makes it feasible to be queried about specific insights relating to malware behaviors and relationships. This feature, in turn, enables security professionals to perform complex analyses involving visual and semantic aspects of malware data. This opens up significantly broader opportunities for a knowledge graph as an instrument in carrying out research in the cybersecurity space and operational threat intelligence.
- **Potential for Collaborative Threat Intelligence:** Structured nature of knowledge graph makes it the perfect ground for sharing and collaboration on threat intelligence across different organizations. A semantic

framework that keeps the data context and meaning makes for more effective collaboration in identification and mitigation of cyber threats. Coordination in this area is imperative in developing defenses against advanced malware.

In conclusion, the knowledge graph development in this Chapter demonstrates the potential of integrating deep learning and ontologies to create a semantically enriched and dynamic knowledge base for malware analysis. This approach not only enhances the interpretability and usability of malware data but also provides a robust foundation for future research and collaboration in cybersecurity.

9 Conclusions & Future Work

This research aims to contribute in the field of malware behaviour analysis to observe how the sequences of opcodes are being changed for each type of new generation malware variants. Deep Learning based malware image processing and visualization has helped to develop a deeper insight for malware detection and classification process. A new knowledge based graph framework is being developed by combining ontology and deep learning.

9.1 Conclusions

In Chapter 1, the research problem was identified and aims and objectives were discussed to achieve the contributions. Knowledge graphs are beneficial to unify and integrate the data points and mapping of relationship among them. A knowledge graph based malware behaviour mapping was needed to enable the capturing of metamorphic malware behaviour. To address the gap of a dynamic malware behaviour knowledge graph an ontology driven malware behaviour capturing system was developed in Chapter 4 for deeper insights and understanding of malware behaviour analysis. The knowledge graph was based on a domain ontology and a benchmark metamorphic malware behaviour dataset. The Ontology-driven knowledge graph was developed to build an ontology for dynamic analysis of malware behaviour. The proposed framework includes 14 malware families with their sub-families and 3 types of malware code-structure with their individuals. With a focus on malware behaviour the proposed ontology depicts the relation among malware families and malware code-structures with their respective behaviour. The

proposed ontological framework can capture behaviour of metamorphic malware from a benchmark dataset which was generated based on API calls of Windows malware.

The proposed ontology includes different kinds of malware families, code structures and behavioural changes within it, therefore structuring a knowledge base that can help in malware detection and analysis. By implementing a dynamic ontology with a malware dataset based on API call sequences, the framework provides an innovative approach towards modeling and understanding complex malware behaviors.

The key contributions include:

- An domain ontology structure foundation, which classifies malware into their types and code structures.
- Sequences of API calls to automatically capture malware behavioral dynamics, with enhanced capabilities on ontology representation towards dynamic changes of malware code.
- Evaluation of the ontology through metrics and comparison of the ontology with available existing ones for its functionality and further usage.

The proposed ontology successfully models the advanced malware behavior where changes in code structure and behavior were represented and especially for the metamorphic types of malware because they are really known to adapt their code in each iteration by changing their original or initial behavior with the traditional signature-based detection. Capturing these nuanced aspects, there is a very robust ontological framework created for understanding as well as the analysis of the malware behavior.

The ontology has a great value for researchers and practitioners in the field of cybersecurity, as it allows them to conceptualize and analyze the behaviour of new-generation malware. It helps in knowledge sharing and management of malware domain, thereby leading to better detection and mitigation strategies.

Because of proactive and advanced cyber threats which are being major concern day by day, potential malware threats has become crimeware. Every malicious behaviour or cyber threat related incidents cannot be automatically resolved by machine learning. Recently, Deep Learning models are performing excellent to solve Computer Vision problems as deep learning models are

more dependent on feature learning rather than being task specific. In Chapter 5 several deep learning based experiments were carried out for malware detection and classification purposes to find out the best possible results. The deep learning based experiments are focusing on developing image-based malware detection and classification methods to construct the novel frameworks than the state-of-the-art. Convolutional Neural Networks (CNNs) are the most promising deep learning models as it can analyze multidimensional data (images) and detect malicious and benign files. CNN-based methods performed well to develop detection and classification methods and produced very good accuracy.

The deep learning experiments conducted in this research emphasize the transformative role that advanced AI methodologies play in malware classification and cybersecurity. To accomplish this, using advanced architectures like ResNet-50, CNNs, it demonstrated how deep learning could potentially automate the extraction and analysis of complex features from malware images with high efficiency. This is the crux in cybersecurity since sheer volume and diversity of malware call for scalable detection systems.

The findings of these experiments contribute to a larger goal towards the integration of AI into cybersecurity; they emphasize resilience and adaptability in the struggle against evolving threats from malware. Deep learning models are well equipped to handle dynamic and diverse data sets, thereby positioning them to be central pieces in the automated malware detection system of the future.

While significant progress has been made, however the path to fully automated, resilient and interpretable cybersecurity solutions remains long. This exercise moves forward an essential step in the use of AI to address one of the most significant challenges in the digital world-malware detection and classification.

In Subsection 6.1.1, We successfully train and implement a Generative Adversarial Network (GAN) and a classifier in continual learning on a collection of images. The main aim was to have the GAN generate augmented images and then subsequently train a classifier with the images to enhance performance. This would prove the practicality of data augmentation using GANs on image classification. The generated images improved the classifier's training set, leading to better accuracy and robustness. Integration of the continual

learning technique also ensured the performance of the classifier during successive iterations of the training process.

In Subsection 6.1.2, we also had the experiment showing implementation and testing a Continual Learning (CL) classifier built from the architecture of U-Net for the task of a multi-class classification problem. This task used U-Net that performs feature extraction for input images as it offers robust encoder-decoder structure, within a classifier formed by incorporating it into the sequential neural network to perform a CL classification problem. Plot of training accuracies over epochs with a clear model learning curve indicated the improvement over epochs. In this experiment, a U-Net-based feature extractor was used to integrate with a Continual Learning classifier with promising results in a single-task classification problem.

In Subsection 6.1.3, this study shows the potential for the combination of ResNet-50 and EWC in the malware image classification across multiple datasets. The approach to continual learning was able to avoid catastrophic forgetting while maintaining a high accuracy in the tasks performed, thus showing potential for a dynamic cybersecurity environment. The experiments indicate the relevance of task-specific optimization and leave open further investigation for improvement in performance, such as for more complex datasets as MaleVis.

This experiment integrates transfer learning within a unified framework with continual learning to serve as a new paradigm for malware classification. The novel application of EWC improves on the model's ability to retain and adapt knowledge across sequential tasks, hence solving the critical gap in the existed malware detection systems.

The method differs from other traditional approaches where models are trained from scratch on new datasets or malware variants and it improves the efficiency of learning by preserving critical knowledge of previous tasks selectively while adapting to new ones. This balance between retention and plasticity is very important in the cybersecurity domain because of the dynamic nature of threats, which always require adaptation.

Furthermore, it incorporates transfer learning using the ResNet50 architecture, which enhances the speed of training and applies pre-existing knowledge from large-scale natural image datasets. The integration of transfer

learning with continual learning ensures that accuracy and efficiency in malware detection are increased while setting up a scalable base for future work.

The experiment results prove that deep learning approaches can be effectively applied to varied malware datasets without compromising the level of accuracy and generalization.

The Chapter 7 of this thesis is trying to combine these two methods: “Domain Adaptation" and “Building Knowledge Graphs with Deep Learning-based Image Classification and Ontology Integration".

In Subsection 7.1.1, the integration of ontology file with the malware image data through the use of a “Domain Adaptation" model suggests a very good approach towards more accurate and more robust classification in malware classification. This experiment exploited the use of rdflib library in manipulating ontology data such that classification becomes possible regarding their respective types in malware as against their behavior.

The outcome of this experiment indicates that the inclusion of ontology data in conjunction with malware image data improves the understanding and the classification capabilities of the model. By leveraging deep semantic information from the ontology, the model is able to capture the subtleties of the available different types of malware, which further implies enhanced accuracy of classifying malware. Moreover, the use of the proposed approach yields robustness in the model toward different data sources, thereby making this useful tool for malware detection and analysis in real-world scenarios.

Future work will encompass further improvements of the model like adding more features to the ontologies, running experiments on architectures, and pushing the approach over larger and different datasets. Thus, integrating the semantic knowledge base with visual information opens up ways for developing advancements in malware-detection technologies to improve the robustness of these cybersecurity measures.

In section 7.4, the experiment for the integration of Ontology and Deep Learning by “Building Knowledge Graphs" has demonstrated the integration of deep learning and semantic web technologies to build an enriched knowledge graph for image classification.

This experiment implements an interesting novel integration of deep learn-

ing with ontology for enriching semantic image classification and knowledge graph population concerning malware. With the combination of convolutional neural networks (CNNs) with ontological frameworks, the work introduces a systematic way of explaining the behavior and visual features of malware. Through this integration, three different types of malware knowledge graphs are formed with enrichment using both behavioral data and visual signatures. The approach demonstrates the potential for more accurate threat detection and a more intuitive understanding of cybersecurity data.

The proposed framework enhances the capability of detecting, analyzing, and understanding malware by integrating semantic image classification with malware behavior data. By using convolutional neural networks (CNNs) for image classification and RDF/OWL knowledge graphs, the unstructured visual data is linked to structured, semantically enriched information.

The development of the knowledge graph is novel in several aspects:

1. **Dynamic Population:** This framework would allow the dynamic and automated population of knowledge graphs by organizing image data using classifiers. Scalability and flexibility would then be introduced in a manner where knowledge graphs evolve with new input, ensuring that knowledge is always current and comprehensive.
2. **Semantic Enrichment:** The use of ontologies further enriches the image data classified with semantic relationships, thus making it capable of deeper context and meaningful connections between different types of malware, behaviors, and their visual signatures.
3. **Integration of Visual and Behavioral Data:** Thus, combination of malware image data set and behavioural ontologies yields a holistic graph that encompasses much more than only the malware signature; now it combines behavioural patterns into a significantly richer dataset.
4. **Interactive and Visualized Graphs:** The use of tools such as NetworkX and PyVis allows users to visually interact with the knowledge graph in order to further understand and decide on matters involving cybersecurity.

Finally, the developed knowledge graph framework is a robust tool for malware analysis as it provides greater capabilities for the organization, analysis

and visualization of complex data related to malware. This also opens new directions for future research in knowledge graphs, especially the integration of heterogeneous data types and support for more advanced cybersecurity applications. This experiment showcases a comprehensive pipeline that integrates image classification with semantic web technologies, creating a robust knowledge graph enriched with both semantic data and visual representations.

9.2 Future Work

Future work of this research has the following objectives to achieve:

- To expand the ontology with other available datasets to improve its class richness and add more classes and attributes in it. This approach will make the ontology more dynamic to act as a malware behaviour knowledge base.
- Investigate methods to scale ontology-based knowledge graphs to accommodate the continuously growing and evolving nature of malware. This includes optimizing the storage, retrieval and updating processes to maintain efficiency and accuracy.
- Including implementing more complex ontologies, more malware related datasets, or different machine learning models to yield better classification precision and graph quality.
- To develop self-generating ontologies for generating relevant knowledge of different domains. Self-generating ontologies are relatively new ideas in AI field, incorporation of sel-generating ontology in cyber security field will be a novel concept to understand attack patterns, zero-day attack prevention, malware behaviour recording etc.
- Multimodal cyber incident datasets need to be incorporated to develop more advanced frameworks. Incorporation of multimodal dataset in cyber security field will be beneficial to secure personal and sensitive information.
- Develop and implement real-time malware detection systems that leverage the proposed methodologies. This involves addressing challenges related to computational efficiency, latency, and resource constraints in real-world environments.

- Fine-tuning the architecture of the GAN and the hyperparameters involved in it as well as checking its applicability on other datasets and classification problems.
- Continuing to develop a continual learning framework for more novel tasks and introducing mechanisms to avoid catastrophic forgetting.
- Extend the domain adaptation techniques to a broader range of related domains and investigate their applicability to other types of cyber threats beyond malware, such as phishing or ransomware.
- Create interactive visualization tools that allow cybersecurity analysts to dynamically explore and interpret malware datasets, facilitating more intuitive and efficient threat analysis.
- Combine the proposed methods with external threat intelligence sources to enhance the contextual understanding of malware and improve predictive capabilities.
- Expand the scope of analysis to include user and system behavior patterns, integrating these with malware behavior captured by knowledge graphs and deep learning models to provide a holistic view of cybersecurity threats.
- Signal processing needs to be incorporated with the image processing techniques for performing noise reduction, image sharpening etc.
- Novel malware attack patterns and vulnerabilities need to be considered to build more advanced detection methods.
- Novel Deep Learning based approaches need to be proposed independently to incorporate more number of datasets as a futute scope of this thesis.
- Federated Learning based methods need to be developed to improve remote work experiences. Virtual and remote works has grown rapidly after last two years, which led to more cyber related incidents. Federated Learning could be an option to secure the remote work spaces.

By pursuing these future directions, the research community can build upon the foundation laid by this thesis, advancing the state-of-the-art in malware detection and contributing to the development of more robust and adaptive cybersecurity solutions.

Bibliography

[1] A. Oltramari, L. F. Cranor, R. J. Walls, and P. D. McDaniel, "Building an ontology of cyber security." in *STIDS*. Citeseer, 2014, pp. 54–61.

[2] A. F. Agarap, "Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (svm) for malware classification," 2019. [Online]. Available: https://arxiv.org/abs/1801.00318

[3] J. Z. Bakdash, S. Hutchinson, E. G. Zaroukian, L. R. Marusich, S. Thirumuruganathan, C. Sample, B. Hoffman, and G. Das, "Malware in the future? forecasting of analyst detection of cyber events," *Journal of Cybersecurity*, vol. 4, no. 1, p. tyy007, 2018.

[4] A. Sharma and S. Sahay, "Evolution and detection of polymorphic and metamorphic malwares: A survey," *International Journal of Computer Applications*, vol. 90, 06 2014.

[5] S. R. Swamy, "Sandbox: A secured testing framework for applications," vol. 4, pp. 1–8, 09 2020.

[6] A. Mills and P. Legg, "Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques," *Journal of Cybersecurity and Privacy*, vol. 1, no. 1, pp. 19–39, 2021. [Online]. Available: https://www.mdpi.com/2624-800X/1/1/3

[7] Y. Guo, "A review of machine learning-based zero-day attack detection: Challenges and future directions," *Computer Communications*, vol. 198, pp. 175–185, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366422004248

[8] G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, 2019.

[9] R. Zhou, J. Pan, X. Tan, and H. Xi, "Application of clips expert system to malware detection system," in *CIS*, 2008.

[10] S. Wheatley, T. Maillart, and D. Sornette, "The extreme risk of personal data breaches and the erosion of privacy," *The European Physical Journal B*, vol. 89, 05 2015.

[11] T. Maillart and D. Sornette, "Heavy-tailed distribution of cyber-risks," *The European Physical Journal B*, vol. 75, no. 3, pp. 357–364, 2010.

[12] C. Peng, M. Xu, S. Xu, and T. Hu, "Modeling multivariate cybersecurity risks," *Journal of Applied Statistics*, vol. 45, no. 15, pp. 2718–2740, 2018.

[13] Z. Zhan, M. Xu, and S. Xu, "Predicting cyber attack rates with extreme values," *IEEE Transactions on Information Forensics and Security*, vol. 10, pp. 1666–1677, 2015.

[14] B. Edwards, S. Hofmeyr, and S. Forrest, "Hype and heavy tails: A closer look at data breaches," 06 2015.

[15] M. Xu, K. M. Schweitzer, R. M. Bateman, and S. Xu, "Modeling and predicting cyber hacking breaches," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 11, pp. 2856–2871, 2018.

[16] R. C. P., "Privacy rights clearinghouse's chronology of data breaches," https://privacyrights.org/, nov. 2017.

[17] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, "Using bayesian networks for cyber security analysis," in *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2010, pp. 211–220.

[18] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen, "Using bayesian networks for probabilistic identification of zero-day attack paths," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2506–2521, 2018.

[19] G. Werner, S. Yang, and K. McConky, "Time series forecasting of cyber attack intensity," in *Proceedings of the 12th Annual Conference on cyber and information security research*, 2017, pp. 1–3.

[20] Z. Fuyong and Z. Tiezhu, "Malware detection and classification based on n-grams attribute similarity," in *2017 IEEE international conference on computational science and engineering (CSE) and IEEE international conference on embedded and ubiquitous computing (EUC)*, vol. 1. IEEE, 2017, pp. 793–796.

[21] D. Yuxin and Z. Siyi, “Malware detection based on deep learning algorithm,” *Neural Computing and Applications*, vol. 31, no. 2, pp. 461–472, 2019.

[22] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel feature extraction, selection and fusion for effective malware family classification,” in *Proceedings of the sixth ACM conference on data and application security and privacy*, 2016, pp. 183–194.

[23] M. Wojnowicz, G. Chisholm, M. Wolff, and X. Zhao, “Wavelet decomposition of software entropy reveals symptoms of malicious code,” *Journal of Innovation in Digital Ecosystems*, vol. 3, no. 2, pp. 130–140, 2016.

[24] M. Hassen and P. K. Chan, “Scalable function call graph-based malware classification,” in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 239–248.

[25] D. Carlin, A. Cowan, P. O’kane, and S. Sezer, “The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes,” *IEEE Access*, vol. 5, pp. 17 742–17 752, 2017.

[26] P. Okane, S. Sezer, and K. McLaughlin, “May, 2016. detecting obfuscated malware using reduced opcode set and optimised runtime trace. security informatics 5 (1), 2.”

[27] D. Bekerman, B. Shapira, L. Rokach, and A. Bar, “Unknown malware detection using network traffic classification,” in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 134–142.

[28] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, “Behavior-based features model for malware detection,” *Journal of Computer Virology and Hacking Techniques*, vol. 12, no. 2, pp. 59–67, 2016.

[29] Z. Salehi, A. Sami, and M. Ghiasi, “Maar: Robust features to detect malicious activity based on api calls, their arguments and return values,” *Engineering Applications of Artificial Intelligence*, vol. 59, pp. 93–102, 2017.

[30] U. Singh, C. Joshi, and S. Singh, “Zero day attacks defense technique for protecting system against unknown vulnerabilities,” 02 2017.

[31] M. Alazab, S. Venkatraman, P. Watters, M. Alazab *et al.*, "Zero-day malware detection based on supervised learning algorithms of api call signatures," 2010.

[32] B. R. Avasarala, J. C. Day, D. Steiner *et al.*, "System and method for automated machine-learning, zero-day malware detection," Mar. 22 2016, uS Patent 9,292,688.

[33] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications surveys & tutorials*, vol. 18, no. 2, pp. 1153–1176, 2015.

[34] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE transactions on information forensics and security*, vol. 11, no. 2, pp. 289–302, 2015.

[35] R. Colbaugh and K. Glass, "Proactive defense for evolving cyber threats," *Proceedings of 2011 IEEE International Conference on Intelligence and Security Informatics*, pp. 125–130, 2011.

[36] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 11–20.

[37] W. Huang and J. W. Stokes, "Mtnet: a multi-task neural network for dynamic malware classification," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2016, pp. 399–418.

[38] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 1, pp. 15–28, 2019.

[39] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2017, pp. 1011–1014.

[40] B. Athiwaratkun and J. W. Stokes, “Malware classification with lstm and gru language models and a character-level cnn,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2482–2486.

[41] A.-H. Muna, N. Moustafa, and E. Sitnikova, “Identification of malicious activities in industrial internet of things based on deep learning models,” *Journal of Information security and applications*, vol. 41, pp. 1–11, 2018.

[42] A. Vorobiev and J. H. J. Han, “Security attack ontology for web services,” in *2006 Semantics, Knowledge and Grid, Second International Conference on*. IEEE, 2006, pp. 42–42.

[43] N. Abrek, “Attack taxonomies and ontologies,” in *Seminar Future Internet SS2014, Network Architectures and Services*, 2015.

[44] J. Undercofer, A. Joshi, T. Finin, J. Pinkston *et al.*, “A target-centric ontology for intrusion detection,” in *Workshop on Ontologies in Distributed Systems, held at The 18th International Joint Conference on Artificial Intelligence*, 2003.

[45] J. Undercoffer and J. Pinkston, “Modeling computer attacks: a target-centric ontology for intrusion detection,” in *Proceedings of the 2002 UMBC Center for Architectures for Data-Driven Information Processing Research Symposium*. Citeseer, 2002.

[46] R. Luh, S. Marschalek, M. Kaiser, H. Janicke, and S. Schrittwieser, “Semantics-aware detection of targeted attacks: a survey,” *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 47–85, 2017.

[47] W. Znaidi, M. Minier, and J.-P. Babau, “An ontology for attacks in wireless sensor networks,” 2008.

[48] A. Simmonds, P. Sandilands, and L. Van Ekert, “An ontology for network security attacks,” in *Asian Applied Computing Conference*. Springer, 2004, pp. 317–323.

[49] O. Can, M. O. Unalir, E. Sezer, O. Bursa, and B. Erdogdu, “An ontology based approach for host intrusion detection systems,” in *Research Conference on Metadata and Semantics Research*. Springer, 2017, pp. 80–86.

[50] J.-b. Gao, B.-w. Zhang, X.-h. Chen, and Z. Luo, "Ontology-based model of network and computer attacks for security assessment," *Journal of Shanghai Jiaotong University (Science)*, vol. 18, no. 5, pp. 554–562, 2013.

[51] R. P. van Heerden, B. Irwin, and I. Burke, "Classifying network attack scenarios using an ontology," in *Proceedings of the 7th International Conference on Information-Warfare & Security (ICIW 2012)*, 2012, pp. 311–324.

[52] F. Massacci, J. Mylopoulos, F. Paci, T. Tun, and Y. Yu, "An extended ontology for security requirements," vol. 83, 01 2011, pp. 622–636.

[53] A. Souag, C. Salinesi, and I. Comyn-Wattiau, "Ontologies for security requirements: A literature survey and classification," in *International conference on advanced information systems engineering*. Springer, 2012, pp. 61–69.

[54] O. Corcho, M. Fernández-López, and A. Gomez-Perez, "Ontological engineering: What are ontologies and how can we build them?" *Semantic Web Services: Theory, Tools and Applications*, 01 2007.

[55] A. Grégio, R. Bonacin, O. Nabuco, V. M. Afonso, P. L. De Geus, and M. Jino, "Ontology for malware behavior: A core model proposal," in *2014 IEEE 23rd International WETICE Conference*. IEEE, 2014, pp. 453–458.

[56] A. Grégio, R. Bonacin, A. C. de Marchi, O. F. Nabuco, and P. L. de Geus, "An ontology of suspicious software behavior," *Applied Ontology*, vol. 11, no. 1, pp. 29–49, 2016.

[57] D. Kiwia, A. Dehghantanha, K.-K. R. Choo, and J. Slaughter, "A cyber kill chain based taxonomy of banking trojans for evolutionary computational intelligence," *Journal of Computational Science*, 11 2017.

[58] T. Tafazzoli and S. H. Sadjadi, "Malware fuzzy ontology for semantic web," *International Journal of Computer Science and Network Security*, vol. 8, no. 7, pp. 153–161, 2008.

[59] N. Rastogi, "Malont: An ontology for malware threat intelligence," 08 2020.

[60] D. A. Mundie and D. M. Mcintire, “An ontology for malware analysis,” in *2013 International Conference on Availability, Reliability and Security*. IEEE, 2013, pp. 556–558.

[61] F. Abdoli, N. Meibody, and R. Bazoubandi, “An attacks ontology for computer and networks attack,” in *SCSS*, 2008.

[62] M. Iannacone, S. Bohn, G. Nakamura, J. Gerth, K. Huffer, R. Bridges, E. Ferragut, and J. Goodall, “Developing an ontology for cyber security knowledge graphs,” in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–4.

[63] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, “Uco: A unified cybersecurity ontology,” in *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[64] H.-D. Huang, T.-Y. Chuang, Y.-L. Tsai, and C.-S. Lee, “Ontology-based intelligent system for malware behavioral analysis,” in *International Conference on Fuzzy Systems*. IEEE, 2010, pp. 1–6.

[65] B. Jasiul, J. Sliwa, K. Gleba, and M. Szpyrka, “Identification of malware activities with rules,” *2014 Federated Conference on Computer Science and Information Systems*, pp. 101–110, 2014.

[66] L. C. Navarro, A. K. W. Navarro, A. Grégio, A. Rocha, and R. Dahab, “Leveraging ontologies and machine-learning techniques for malware analysis into android permissions ecosystems,” *Comput. Secur.*, vol. 78, pp. 429–453, 2018.

[67] H.-S. Chiang, W.-J. Tsaur *et al.*, “Ontology-based mobile malware behavioral analysis,” in *IEEE Second International Conference on Social Computing (SocialCOM 2010)*, vol. 10, 2010.

[68] J. N. O.S and M. S. B. S, “Detection of malicious android applications using ontology-based intelligent model in mobile cloud environment,” *Journal of Information Security and Applications*, vol. 58, p. 102751, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212621000041

[69] P. Wang, K.-M. Chao, C.-C. Lo, and Y.-S. Wang, "Using ontologies to perform threat analysis and develop defensive strategies for mobile security," *Information Technology and Management*, vol. 18, no. 1, pp. 1–25, 2017.

[70] H.-D. Huang, G. Acampora, V. Loia, C.-S. Lee, and H.-Y. Kao, "Applying fml and fuzzy ontologies to malware behavioural analysis," in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*. IEEE, 2011, pp. 2018–2025.

[71] L. Obrst, P. Chase, and R. Markeloff, "Developing an ontology of the cyber security domain," in *STIDS*, 2012.

[72] W. Han, "Aptmalinsight: Identify and cognize apt malware based on system call information and ontology knowledge framework," *Information Sciences*, vol. 546, pp. 633–664, 09 2020.

[73] M. Noor, H. Abbas, and W. B. Shahid, "Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis," *J. Netw. Comput. Appl.*, vol. 103, pp. 249–261, 2018.

[74] S. Joshi, H. Upadhyay, L. Lagos, N. S. Akkipeddi, and V. Guerra, "Machine learning approach for malware detection using random forest classifier on process list data structure," in *Proceedings of the 2nd International Conference on Information System and Data Mining*, ser. ICISDM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 98–102. [Online]. Available: https://doi.org/10.1145/3206098.3206113

[75] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 271–280.

[76] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 109–129.

[77] F. A. Narudin, A. Feizollah, N. B. Anuar, and A. Gani, "Evaluation of machine learning classifiers for mobile malware detection," *Soft Computing*, vol. 20, no. 1, pp. 343–357, 2016.

[78] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *2013 IEEE 25th international conference on tools with artificial intelligence*. IEEE, 2013, pp. 300–305.

[79] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "Puma: Permission usage to detect malware in android," in *International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions*. Springer, 2013, pp. 289–298.

[80] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow apis and machine learning," *Information and software technology*, vol. 75, pp. 17–25, 2016.

[81] P. Wang and Y.-S. Wang, "Malware behavioural detection and vaccine development by using a support vector model classifier," *Journal of Computer and System Sciences*, vol. 81, no. 6, pp. 1012–1026, 2015.

[82] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[83] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3216–3225, 2018.

[84] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting android malware," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 377–388.

[85] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. IEEE, 2014, pp. 37–42.

[86] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[87] I. Firdausi, A. Erwin, A. S. Nugroho *et al.*, "Analysis of machine learning techniques used in behavior-based malware detection," in *2010 Second international conference on advances in computing, control, and telecommunication technologies*. IEEE, 2010, pp. 201–203.

[88] H. Sayadi, H. M. Makrani, O. Randive, S. M. PD, S. Rafatirad, and H. Homayoun, "Customized machine learning-based hardware-assisted malware detection in embedded devices," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1685–1688.

[89] A. S. Shamili, C. Bauckhage, and T. Alpcan, "Malware detection on mobile devices using distributed machine learning," in *2010 20th International Conference on Pattern Recognition*. IEEE, 2010, pp. 4348–4351.

[90] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining," *Journal in computer virology*, vol. 4, no. 4, pp. 323–334, 2008.

[91] Y. Ye, D. Wang, T. Li, and D. Ye, "Imds: Intelligent malware detection system," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 1043–1047.

[92] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[93] B. Amos, H. Turner, and J. White, "Applying machine learning classifiers to dynamic android malware detection at scale," in *2013 9th international wireless communications and mobile computing conference (IWCMC)*. IEEE, 2013, pp. 1666–1671.

[94] H.-Y. Chuang and S.-D. Wang, "Machine learning based hybrid behavior models for android malware analysis," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 201–206.

[95] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *2016 IEEE 40th annual computer software and applications conference (COMPSAC)*, vol. 2. IEEE, 2016, pp. 577–582.

[96] S. Jeon and J. Moon, “Malware-detection method with a convolutional recurrent neural network using opcode sequences,” *Information Sciences*, vol. 535, pp. 1–15, 2020.

[97] Y. Dai, H. Li, Y. Qian, R. Yang, and M. Zheng, “Smash: A malware detection method based on multi-feature ensemble learning,” *IEEE Access*, vol. 7, pp. 112 588–112 597, 2019.

[98] Y. Ding, X. Zhang, J. Hu, and W. Xu, “Android malware detection method based on bytecode image,” *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–10, 2020.

[99] F. Xiao, Z. Lin, Y. Sun, and Y. Ma, “Malware detection based on deep learning of behavior graphs,” *Mathematical Problems in Engineering*, vol. 2019, 2019.

[100] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, “Dl4md: A deep learning framework for intelligent malware detection,” in *Proceedings of the International Conference on Data Science (ICDATA)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2016, p. 61.

[101] S. Venkatraman, M. Alazab, and R. Vinayakumar, “A hybrid deep learning image-based analysis for effective malware detection,” *Journal of Information Security and Applications*, vol. 47, pp. 377–389, 2019.

[102] K. He and D.-S. Kim, “Malware detection with malware images using deep learning techniques,” in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2019, pp. 95–102.

[103] A. Darem, J. Abawajy, A. Makkar, A. Alhashmi, and S. Alanazi, “Visualization and deep-learning-based malware variant detection using opcode-level features,” *Future Generation Computer Systems*, vol. 125, pp. 314–323, 2021.

[104] S. Ni, Q. Qian, and R. Zhang, “Malware identification using visualization images and deep learning,” *Computers & Security*, vol. 77, pp. 871–885, 2018.

[105] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, ser. VizSec '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2016904.2016908

[106] Y.-S. Yen and H.-M. Sun, "An android mutation malware detection based on deep learning using visualization of importance from codes," *Microelectronics Reliability*, vol. 93, pp. 109–114, 2019.

[107] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[108] Z. Cui, F. Xue, X. Cai, Y. Cao, G.-g. Wang, and J. Chen, "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 7, pp. 3187–3196, 2018.

[109] H. Naeem, F. Ullah, M. R. Naeem, S. Khalid, D. Vasan, S. Jabbar, and S. Saeed, "Malware detection in industrial internet of things based on hybrid image visualization and deep learning model," *Ad Hoc Networks*, vol. 105, p. 102154, 2020.

[110] X. Huang, L. Ma, W. Yang, and Y. Zhong, "A method for windows malware detection based on deep learning," *Journal of Signal Processing Systems*, vol. 93, no. 2, pp. 265–273, 2021.

[111] F. Mercaldo and A. Santone, "Deep learning for image-based mobile malware detection," *Journal of Computer Virology and Hacking Techniques*, pp. 1–15, 2020.

[112] L. Zhao, D. Li, G. Zheng, and W. Shi, "Deep neural network based on android mobile malware detection system using opcode sequences," in *2018 IEEE 18th International Conference on Communication Technology (ICCT)*. IEEE, 2018, pp. 1141–1147.

[113] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, 2018.

[114] A. Pektaş and T. Acarman, "Learning to detect android malware via opcode sequences," *Neurocomputing*, vol. 396, pp. 599–608, 2020.

[115] H. HaddadPajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, "A deep recurrent neural network based approach for internet of things malware threat hunting," *Future Generation Computer Systems*, vol. 85, pp. 88–96, 2018.

[116] Y. Ding, S. Chen, and J. Xu, "Application of deep belief networks for opcode based malware detection," in *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016, pp. 3901–3908.

[117] A. Yazdinejad, H. HaddadPajouh, A. Dehghantanha, R. M. Parizi, G. Srivastava, and M.-Y. Chen, "Cryptocurrency malware hunting: A deep recurrent neural network approach," *Applied Soft Computing*, vol. 96, p. 106630, 2020.

[118] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "Deepam: a heterogeneous deep learning framework for intelligent malware detection," *Knowledge and Information Systems*, vol. 54, no. 2, pp. 265–285, 2018.

[119] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," in *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*. IEEE, 2016, pp. 104–111.

[120] H. Zhu, Y. Li, R. Li, J. Li, Z.-H. You, and H. Song, "Sedmdroid: An enhanced stacking ensemble of deep learning framework for android malware detection," *IEEE Transactions on Network Science and Engineering*, 2020.

[121] M. S. Rahman, S. E. Coull, and M. Wright, "On the limitations of continual learning for malware classification," 2022. [Online]. Available: https://arxiv.org/abs/2208.06568

[122] J. Park, A. Ji, M. Park, M. S. Rahman, and S. E. Oh, "Malcl: Leveraging gan-based generative replay to combat catastrophic forgetting in malware classification," 2025. [Online]. Available: https://arxiv.org/abs/2501.01110

[123] M. S. Rahman, S. Coull, Q. Yu, and M. Wright, "Madar: Efficient continual learning for malware analysis with diversity-aware replay," *arXiv preprint arXiv:2502.05760*, 2025.

[124] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1127–1144.

[125] M. Chin and R. Corizzo, "Continual semi-supervised malware detection," *Machine Learning and Knowledge Extraction*, vol. 6, no. 4, pp. 2829–2854, 2024.

[126] S. K. Amalapuram, B. R. Tamma, and S. S. Channappayya, "Spider: A semi-supervised continual learning-based network intrusion detection system," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 571–580.

[127] L. Kou, D. Zhao, H. Han, X. Xu, S. Gong, and L. Wang, "Sscl-transmd: Semi-supervised continual learning transformer for malicious software detection," *Applied Sciences*, vol. 13, no. 22, 2023. [Online]. Available: https://www.mdpi.com/2076-3417/13/22/12255

[128] T. Sun, N. Daoudi, W. Pian, K. Kim, K. Allix, T. F. Bissyande, and J. Klein, "Temporal-incremental learning for android malware detection," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 4, pp. 1–30, 2025.

[129] Y. Chai, X. Chen, J. Qiu, L. Du, Y. Xiao, Q. Feng, S. Ji, and Z. Tian, "Malfscil: A few-shot class-incremental learning approach for malware detection," *IEEE Transactions on Information Forensics and Security*, vol. 20, pp. 2999–3014, 2025.

[130] X. Xu, X. Zhang, Q. Zhang, Y. Wang, B. Adebisi, T. Ohtsuki, H. Sari, and G. Gui, "Advancing malware detection in network traffic with self-paced class incremental learning," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 06 2024.

[131] D. Ghiani, D. Angioni, G. Piras, A. Sotgiu, L. Minnei, S. Gupta, M. Pintor, F. Roli, and B. Biggio, "Regression-aware continual learning for android malware detection," 2025. [Online]. Available: https://arxiv.org/abs/2507.18313

[132] A. Darem, F. Ghaleb, A. Al-Hashmi, J. Abawajy, S. Alanazi, and A. AL-Rezami, "An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning," *IEEE Access*, vol. PP, pp. 1–1, 06 2021.

[133] I. Arnaldo, A. Arun, S. Kyathanahalli, and K. Veeramachaneni, "Acquire, adapt, and anticipate: continuous learning to block malicious domains," 12 2018, pp. 1891–1898.

[134] W. Zhuang, L. Xiao, J. Cui, and W. Zhuang, "Support vector machine based on incremental learning for malware detection," in *2015 International Conference on Computer Science and Intelligent Communication*. Atlantis Press, 2015, pp. 204–207.

[135] J. Ispahany, M. Islam, M. A. Khan, and M. Islam, "A sysmon incremental learning system for ransomware analysis and detection," *arXiv preprint arXiv:2501.01089*, 2025.

[136] J. Ispahany, M. R. Islam, M. A. Khan, and M. Z. Islam, "icnn-lstm+: A batch-based incremental ransomware detection system using sysmon," *IEEE Access*, 2025.

[137] J. Li, L. Du, Y. Chen, L. Zeng, H. Li, and Z. Gu, "Cil4emtd: A novel class incremental learning method for encrypted malware traffic detection," in *2024 IEEE 9th International Conference on Data Science in Cyberspace (DSC)*, 2024, pp. 300–307.

[138] S. Bhardwaj, A. S. Li, M. Dave, and E. Bertino, "Overcoming the lack of labeled data: Training malware detection models using adversarial domain adaptation," *Computers & Security*, vol. 140, p. 103769, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404824000701

[139] F. Wang, G. Chai, Q. Li, and C. Wang, "An efficient deep unsupervised domain adaptation for unknown malware detection," *Symmetry*, vol. 14, no. 2, p. 296, 2022.

[140] S. Xiong, X. Chen, H. Zhang, and M. Wang, "Domain adaptation-based deep learning framework for android malware detection across diverse distributions," *Artificial Intelligence Advances*, vol. 6, no. 1, pp. 13–24, 2024.

[141] A. S. Li, A. Iyengar, A. Kundu, and E. Bertino, "Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples," 2024. [Online]. Available: https://arxiv.org/abs/2407.13918

[142] Y. Hou, T. Truong-Huu, Z. Chen, C.-K. Kwoh, and S. Teo, "Proteus: Domain adaptation for dynamic features in ai-assisted windows malware detection," 12 2023.

[143] R. Coulter, J. Zhang, L. Pan, and Y. Xiang, "Domain adaptation for windows advanced persistent threat detection," *Computers & Security*, vol. 112, p. 102496, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404821003205

[144] K. Bartos and M. Sofka, "Robust representation for domain adaptation in network security," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015, pp. 116–132.

[145] J. Ning, G. Gui, Y. Wang, J. Yang, B. Adebisi, S. Ci, H. Gacanin, and F. Adachi, "Malware traffic classification using domain adaptation and ladder network for secure industrial internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 058–17 069, 2021.

[146] A. Pingle, A. Piplai, S. Mittal, A. Joshi, J. Holt, and R. Zak, "Relext: relation extraction using deep learning approaches for cybersecurity knowledge graph improvement," in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '19. New York, NY, USA: Association for Computing Machinery, 2020, p. 879–886. [Online]. Available: https://doi.org/10.1145/3341161.3343519

[147] J. S. Garrido, D. Dold, and J. Frank, "Machine learning on knowledge graphs for context-aware security monitoring," in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*. IEEE, Jul. 2021, p. 55–60. [Online]. Available: http://dx.doi.org/10.1109/CSR51186.2021.9527927

[148] S. Dasgupta, A. Piplai, P. Ranade, and A. Joshi, "Cybersecurity knowledge graph improvement with graph neural networks," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 3290–3297.

[149] H. Li, Z. Shi, C. Pan, D. Zhao, and N. Sun, "Cybersecurity knowledge graphs construction and quality assessment," *Complex & Intelligent Systems*, vol. 10, 08 2023.

[150] K. Li, H. Zhou, Z. Tu, and B. Feng, "Cskb: A cyber security knowledge base based on knowledge graph," in *International Conference on Security and Privacy in Digital Economy*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:226069656

[151] S. Mitra, A. Piplai, S. Mittal, and A. Joshi, "Combating fake cyber threat intelligence using provenance in cybersecurity knowledge graphs," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 3316–3323.

[152] G. Agrawal, Y. Deng, J. Park, H. Liu, and Y.-C. Chen, "Building knowledge graphs from unstructured texts: Applications and impact analyses in cybersecurity education," *Information*, vol. 13, no. 11, 2022. [Online]. Available: https://www.mdpi.com/2078-2489/13/11/526

[153] Y. Qi, Z. Gu, A. Li, X. Zhang, M. Shafiq, Y. Mei, and K. Lin, "Cybersecurity knowledge graph enabled attack chain detection for cyber-physical systems," *Computers and Electrical Engineering*, vol. 108, p. 108660, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004579062300085X

[154] X. Yang, G. Peng, D. Zhang, and Y. Lv, "An enhanced intrusion detection system for iot networks based on deep learning and knowledge graph," *Security and Communication Networks*, vol. 2022, no. 1, p. 4748528, 2022.

[155] Y. Jia, Y. Qi, H. Shang, R. Jiang, and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53–60, 2018, cybersecurity. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809918301097

[156] I. Sarhan and M. Spruit, "Open-cykg: An open cyber threat intelligence knowledge graph," *Knowledge-Based Systems*, vol. 233, p. 107524, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705121007863

[157] Z. Wang, H. Zhu, P. Liu, and L. Sun, "Social engineering in cybersecurity: a domain ontology and knowledge graph application examples," *Cybersecurity*, vol. 4, no. 1, Aug. 2021. [Online]. Available: http://dx.doi.org/10.1186/s42400-021-00094-6

[158] S. Sun, C. Huang, T. Wu, and Y. Shen, "Sectkg: A knowledge graph for open-source security tools," *International Journal of Intelligent Systems*, vol. 2023, no. 1, p. 4464974, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2023/4464974

[159] P. Wang, J. Liu, D. Hou, and S. Zhou, "A cybersecurity knowledge graph completion method based on ensemble learning and adversarial training," *Applied Sciences*, vol. 12, no. 24, p. 12947, 2022.

[160] H. Xia, Y. Wang, J. Gauthier, and J. Z. Zhang, "Knowledge graph of mobile payment platforms based on deep learning: Risk analysis and policy implications," *Expert Systems with Applications*, vol. 208, p. 118143, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422013264

[161] F. O. Catak, A. F. Yazı, O. Elezaj, and J. Ahmed, "Deep learning based sequential model for malware analysis using windows exe api calls," *PeerJ Computer Science*, vol. 6, p. e285, Jul. 2020. [Online]. Available: https://doi.org/10.7717/peerj-cs.285

[162] J.-B. Lamy, "Owlready: Ontology-oriented programming in python with automatic classification and high level constructs for biomedical ontologies," *Artificial Intelligence in Medicine*, vol. 80, pp. 11–28, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0933365717300271

[163] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Comput. Surv.*, vol. 52, no. 5, Sep. 2019. [Online]. Available: https://doi.org/10.1145/3329786

[164] S. Alam, I. Traore, and I. Sogukpinar, "Annotated control flow graph for metamorphic malware detection," *Computer Journal*, vol. 58, 10 2015.

[165] S. Venkatachalam and M. Stamp, "Detecting undetectable metamorphic viruses," in *Proceedings of 2011 International Conference on Security & Management (SAM'11)*. Citeseer, 2011, pp. 340–345.

[166] R. Mirzazadeh, M. H. Moattar, and M. V. Jahan, "Metamorphic malware detection using linear discriminant analysis and graph similarity," in *2015 5th International Conference on Computer and Knowledge Engineering (ICCKE)*. IEEE, 2015, pp. 61–66.

[167] G. Fernandes, J. Rodrigues, L. F. Carvalho, J. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, pp. 447–489, 2019.

[168] L. P. Lima, L. S. Rocha, C. I. M. Bezerra, and M. Paixao, "Assessing exception handling testing practices in open-source libraries," 2021. [Online]. Available: https://arxiv.org/abs/2105.00500

[169] D. Z. Syeda and M. N. Asghar, "Dynamic malware classification and api categorisation of windows portable executable files using machine learning," *Applied Sciences*, vol. 14, no. 3, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/3/1015

[170] C.-T. Lin, N.-J. Wang, H. Xiao, and C. Eckert, "Feature selection and extraction for malware classification," *Journal of Information Science and Engineering*, vol. 31, pp. 965–992, 05 2015.

[171] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804512002214

[172] A. Case, R. D. Maggio, M. Firoz-Ul-Amin, M. M. Jalalzai, A. Ali-Gombe, M. Sun, and G. G. Richard, "Hooktracer: Automatic detection and analysis of keystroke loggers using memory forensics," *Computers & Security*, vol. 96, p. 101872, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301450

[173] I. Georgiev, S. Eng, and Schaaf, "Cyber security fraud prevention using data analytics developing a layered framework with preconditions to enable fraud identification in bank sector," Ph.D. dissertation, 12 2017.

[174] S. Aydina and M. N. Aydina, “Development and quality evaluation of hazelnut ontology.”

[175] A. Agárdi and L. Kovács, “Property-based quality measures in ontology modeling,” *Applied Sciences*, vol. 12, no. 23, p. 12475, 2022.

[176] B. Ben Mahria, I. Chaker, and A. Zahi, “A novel approach for learning ontology from relational database: from the construction to the evaluation,” *Journal of Big Data*, vol. 8, no. 1, p. 25, 2021.

[177] M. Swimmer, “Towards an ontology of malware classes,” https://dokumen.tips/documents/towards-an-ontology-of-malware-classes.html, jan. 2008.

[178] D. A. Noever and S. E. M. Noever, “Virus-mnist: A benchmark malware dataset,” *CoRR*, vol. abs/2103.00602, 2021. [Online]. Available: https://arxiv.org/abs/2103.00602

[179] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[180] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[181] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, “Malware classification with deep convolutional neural networks,” *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pp. 1–5, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4569169

[182] M. Claesen and B. D. Moor, “Hyperparameter search in machine learning,” *ArXiv*, vol. abs/1502.02127, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:17147092

[183] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, “Hyperband: A novel bandit-based approach to hyperparameter optimization,” *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018. [Online]. Available: http://jmlr.org/papers/v18/16-558.html

[184] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, p. 1–12, Jun. 2017. [Online]. Available: https://doi.org/10.1145/3140659.3080246

[185] C. Silvano, D. Ielmini, F. Ferrandi, L. Fiorin, S. Curzel, L. Benini, F. Conti, A. Garofalo, C. Zambelli, E. Calore, S. Schifano, M. Palesi, G. Ascia, D. Patti, N. Petra, D. De Caro, L. Lavagno, T. Urso, V. Cardellini, G. C. Cardarilli, R. Birke, and S. Perri, "A survey on deep learning hardware accelerators for heterogeneous hpc platforms," *ACM Comput. Surv.*, vol. 57, no. 11, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3729215

[186] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[187] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[188] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, pp. 1–48, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:195811894

[189] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," *Proceedings of the AAAI Conference on Artificial*

Intelligence, vol. 34, no. 07, pp. 13 001–13 008, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/7000

[190] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, May 2017. [Online]. Available: https://doi.org/10.1145/3065386

[191] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[192] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 4489–4497.

[193] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[194] J. Li, Z. Pan, and K. Jiang, "A three-dimensional convolutional neural network for dark web traffic classification based on multi-channel image deep learning," *Computers*, vol. 14, no. 8, 2025. [Online]. Available: https://www.mdpi.com/2073-431X/14/8/295

[195] O. Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2016, p. 424–432. [Online]. Available: https://doi.org/10.1007/978-3-319-46723-8_49

[196] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, and S. Almotairi, "A comparison of pooling methods for convolutional neural networks," *Applied Sciences*, vol. 12, no. 17, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/17/8643

[197] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: a survey," *Artificial intelligence review*, vol. 57, no. 2, p. 28, 2024.

[198] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

[199] U. Ruby, V. Yendapalli *et al.*, "Binary cross entropy with deep learning technique for image classification," *Int. J. Adv. Trends Comput. Sci. Eng*, vol. 9, no. 10, 2020.

[200] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *ICML Unsupervised and Transfer Learning*, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:10921035

[201] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[202] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[203] O. Sagi and L. Rokach, "Ensemble learning: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49291826

[204] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[205] S. O. Haroon-Sulyman, S. S. Kamaruddin, F. K. Ahmad, S. Abdul-Rahman, and N. I. Aziz, "Techniques for mitigating the vanishing gradient problem in deep neural networks," in *Digital Innovation in Knowledge Management*, Z. Mat Aji, R. Romli, S. F. Packeer Mohamed, M. A. Saip, and T. Herawan, Eds. Cham: Springer Nature Switzerland, 2025, pp. 428–438.

[206] G. Xu, X. Wang, X. Wu, X. Leng, and Y. Xu, "Development of skip connection in deep neural networks for computer vision and medical

image analysis: A survey," *Eng. Appl. Artif. Intell.*, vol. 142, p. 109890, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:269588210

[207] J. Park, S. Woo, J.-Y. Lee, and I. S. Kweon, "Bam: Bottleneck attention module," 2018. [Online]. Available: https://arxiv.org/abs/1807.06514

[208] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha, M. Li, and A. Smola, "Resnest: Split-attention networks," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022, pp. 2735–2745.

[209] Y. Wu, R. Fu, T. Xing, and F. Yin, "Loss aware feature attention mechanism for class and feature imbalance issue." *Computers, Materials & Continua*, vol. 82, no. 1, 2025.

[210] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv preprint arXiv:1702.08608*, 2017.

[211] X. Wu, Y. Song, X. Hou, Z. Ma, and C. Chen, "Deep learning model with sequential features for malware classification," *Applied Sciences*, vol. 12, no. 19, p. 9994, 2022.

[212] Y. Sari, Y. Arifin, N. Novitasari, and M. R. Faisal, "Deep learning approach using the gru-lstm hybrid model for air temperature prediction on daily basis," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 10, p. 430–436, 09 2022.

[213] T. V. Dao, H. Sato, and M. Kubo, "An attention mechanism for combination of cnn and vae for image-based malware classification," *IEEE Access*, vol. 10, pp. 85 127–85 136, 2022.

[214] P. Dhruv and S. Naskar, "Image classification using convolutional neural network (cnn) and recurrent neural network (rnn): A review," in *Machine Learning and Information Processing*, D. Swain, P. K. Pattnaik, and P. K. Gupta, Eds. Singapore: Springer Singapore, 2020, pp. 367–381.

[215] K. O. Adefemi, M. B. Mutanga, and O. A. Alimi, "A hybrid cnn–gru deep learning model for iot network intrusion detection," *Journal of Sensor and Actuator Networks*, vol. 14, no. 5, 2025. [Online]. Available: https://www.mdpi.com/2224-2708/14/5/96

[216] A. Bensaoud and J. Kalita, "Cnn-lstm and transfer learning models for malware classification based on opcodes and api calls," *Knowledge-Based Systems*, vol. 290, p. 111543, 2024.

[217] R. Chaganti, V. Ravi, and T. D. Pham, "Deep learning based cross architecture internet of things malware detection and classification," *Computers & Security*, vol. 120, p. 102779, 2022.

[218] H. Kauser. Sk and M. Anu. V, "Hybrid deep learning model for accurate and efficient android malware detection using dbn-gru," *PloS one*, vol. 20, no. 5, p. e0310230, 2025.

[219] R. Lu, "Malware detection with lstm using opcode language," *ArXiv*, vol. abs/1906.04593, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:184488098

[220] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[221] ——, "Generative adversarial nets," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 2672–2680.

[222] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification," *Neurocomputing*, vol. 321, p. 321–331, Dec. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.neucom.2018.09.013

[223] E. Baghirov, "Building robust malware detection through conditional generative adversarial network-based data augmentation," *Program Systems: Theory and Applications*, vol. 15, pp. 97–110, 12 2024.

[224] L. C. Ribas, W. Casaca, and R. T. Fares, "Conditional generative adversarial networks and deep learning data augmentation: A multi-perspective data-driven survey across multiple application fields and classification architectures," *AI*, vol. 6, no. 2, 2025. [Online]. Available: https://www.mdpi.com/2673-2688/6/2/32

[225] K. Thosar, P. Tiwari, R. Jyothula, and D. Ambawade, "Effective malware detection using gradient boosting and convolutional neural network," in *2021 IEEE Bombay Section Signature Conference (IBSSC)*. IEEE, 2021, pp. 1–4.

[226] W. K. Wong, F. H. Juwono, and C. Apriono, "Vision-based malware detection: A transfer learning approach using optimal ecoc-svm configuration," *IEEE Access*, vol. 9, pp. 159 262–159 270, 2021.

[227] G. Sharma, K. Singh, and M. Singh, *A Deep Learning Approach to Image-Based Malware Analysis*, 03 2020, pp. 327–339.

[228] D. Pant and R. Bista, "Image-based malware classification using deep convolutional neural network and transfer learning," in *2021 3rd International Conference on Advanced Information Science and System (AISS 2021)*, 2021, pp. 1–6.

[229] G. M. van de Ven, N. Soures, and D. Kudithipudi, *Continual learning and catastrophic forgetting*. Elsevier, 2025, p. 153–168. [Online]. Available: http://dx.doi.org/10.1016/B978-0-443-15754-7.00073-0

[230] T. P. Van, T. M. Nguyen, N. N. Tran, H. V. Nguyen, L. B. Doan, H. Q. Dao, and T. T. Minh, "Interpreting the latent space of generative adversarial networks using supervised learning," in *2020 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE, Nov. 2020. [Online]. Available: http://dx.doi.org/10.1109/ACOMP50827.2020.00015

[231] G. Kim, Y. S. Kang, S. M. Yang, J. G. Choi, G. Hwang, H. W. Park, and S. Lim, "Fisher-informed continual learning for remaining useful life prediction of machining tools under varying operating conditions," *Reliability Engineering & System Safety*, vol. 253, p. 110549, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832024006215

[232] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, "Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification," *Neurocomputing*, vol. 321, pp. 321–331, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218310749

[233] B.-L. Jian, H.-L. Chang, and C.-L. Chen, "Enhanced u-net with multi-module integration for high-exposure-difference image restoration," *Sensors*, vol. 25, no. 4, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/4/1105

[234] H. Dong and I. Kotenko, "Enhancing malware detection resilience: A u-net gan denoising framework for image-based classification." *Computers, Materials & Continua*, vol. 82, no. 3, 2025.

[235] D. Saadati, O. N. Manzari, and S. Mirzakuchaki, "Dilated-unet: A fast and accurate medical image segmentation approach using a dilated transformer and u-net architecture," *ArXiv*, vol. abs/2304.11450, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:258298561

[236] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[237] B. P. Gond and D. P. Mohapatra, "Deep learning-driven malware classification with api call sequence analysis and concept drift handling," 2025. [Online]. Available: https://arxiv.org/abs/2502.08679

[238] L. Álvarez, S. Valverde, Àlex Rovira, and X. Lladó, "Mitigating catastrophic forgetting in multiple sclerosis lesion segmentation using elastic weight consolidation," *NeuroImage: Clinical*, vol. 46, p. 103795, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213158225000658

[239] M. S. Rahman, S. Coull, and M. Wright, "On the limitations of continual learning for malware classification," in *Conference on Lifelong Learning Agents*. PMLR, 2022, pp. 564–582.

[240] M. Ashawa, N. Owoh, S. Hosseinzadeh, and J. Osamor, "Enhanced image-based malware classification using transformer-based convolutional neural networks (cnns)," *Electronics*, vol. 13, no. 20, p. 4081, 2024.

[241] N. V. Do, "Ontology cokb for knowledge representation and reasoning in designing knowledge-based systems," in *Intelligent Software Method-*

ologies, Tools and Techniques, H. Fujita and A. Selamat, Eds. Cham: Springer International Publishing, 2015, pp. 101–118.

[242] P. Sowiński, K. Wasielewska-Michniewska, M. Ganzha, M. Paprzycki, and C. Bădică, *Ontology Reuse: The Real Test of Ontological Design*. IOS Press, Sep. 2022. [Online]. Available: http://dx.doi.org/10.3233/FAIA220292

[243] J. Chai, H. Zeng, A. Li, and E. W. Ngai, "Deep learning in computer vision: A critical review of emerging techniques and application scenarios," *Machine Learning with Applications*, vol. 6, p. 100134, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827021000670

[244] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial intelligence review*, vol. 53, no. 8, pp. 5455–5516, 2020.

[245] G. Wilson and D. J. Cook, "A survey of unsupervised deep domain adaptation," *ACM Trans. Intell. Syst. Technol.*, vol. 11, no. 5, Jul. 2020. [Online]. Available: https://doi.org/10.1145/3400066

[246] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[247] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[248] J. Singh, D. Thakur, F. Ali, T. Gera, and K. S. Kwak, "Deep feature extraction and classification of android malware images," *Sensors*, vol. 20, no. 24, p. 7013, 2020.

[249] Y. Sun, D. Dai, and S. Xu, "Rethinking adversarial domain adaptation: Orthogonal decomposition for unsupervised domain adaptation in medical image segmentation," *Medical Image Analysis*, vol. 82, p. 102623, 2022.

[250] L. Chen, Z. Chen, W. Yang, S. Liu, and Y. Li, "From pixels to insights: Unsupervised knowledge graph generation with large language model," *Information*, vol. 16, no. 5, 2025. [Online]. Available: https://www.mdpi.com/2078-2489/16/5/335

[251] J. Yuan, A. Le-Tuan, M. Nguyen-Duc, T.-K. Tran, M. Hauswirth, and D. Le-Phuoc, "Visionkg: Unleashing the power of visual datasets via knowledge graph," 2024. [Online]. Available: https://arxiv.org/abs/2309.13610

[252] W. Lee, J. Saxe, and R. Harang, *SeqDroid: Obfuscated Android Malware Detection Using Stacked Convolutional and Recurrent Neural Networks*, 08 2019, pp. 197–210.

[253] R. Christian, S. Dutta, Y. Park, and N. Rastogi, "An ontology-driven knowledge graph for android malware," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2435–2437. [Online]. Available: https://doi.org/10.1145/3460120.3485353

[254] S. Dutta, N. Rastogi, D. Yee, C. Gu, and Q. Ma, "Malware knowledge graph generation," *CoRR*, vol. abs/2102.05583, 2021. [Online]. Available: https://arxiv.org/abs/2102.05583

[255] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu, "Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources," 12 2017, pp. 103–115.