Harnessing Image-Based Deep Learning for Advanced Malware Classification

by

Ahmed A. Abouelkhaire

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Harnessing Image-Based Deep Learning for
Advanced Malware Classification

by

Ahmed A. Abouelkhaire

Supervisory Committee

Dr. Issa Traoré, Co-Supervisor
(Department of Electrical and Computer Engineering)

Dr. Waleed A. Yousef, Co-Supervisor
(Department of Electrical and Computer Engineering)

ABSTRACT

This thesis explores the application of image-based deep learning models for malware classification, leveraging a subset of the extensive MalNet-Image dataset, which includes around 87,000 binary images from a base of 1.2 million binary images based on Android APK files. The core contribution of this work lies in the innovative use of multiple components that, as far as we know, have not been used before to tackle the malware classification problem. Harnessing the power of deep neural networks (DNNs), which have demonstrated exceptional capabilities in various classification tasks, we aim to enhance the accuracy and efficiency of malware detection. These include Feature Pyramid Networks (FPN) to handle the file size scale issue when converting to images and the application of data augmentation techniques like Mixup and Trivial Augment. We employ transfer learning with pre-trained models on ImageNet and optimize them using the AdamW Schedule-Free optimizer. Our experimental results show that the integration of these techniques achieves remarkable improvement in classification accuracy, with our best model achieving an F1 score of 0.6927 compared to 0.65 reported on the provided split for MalNet-Tiny. This could be considered a step forward in the field of malware classification using image-based deep learning models.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my wife for her unwavering support and motivation. Her willingness to take on many of my responsibilities and bear numerous burdens has been crucial to my success in this Master's program.

I extend my heartfelt thanks to Professor Waleed for shaping my thinking, instilling rigor in my approach to definitions, and laying the foundations of my understanding of machine learning. His life guidance has been invaluable to me.

I am also grateful to Andrew Nagiub and Ahmed Farag for their assistance during the application process and their companionship while I was away from my family. Their support has been a great source of comfort and encouragement.

A special thanks goes to Professor Issa for his continuous support throughout the entirety of this Master's program. His provision of the opportunity to pursue my dreams and continue my academic research at the ISOT laboratory has been instrumental.

Lastly, I would like to extend my appreciation to all my professors and the administrative team at the University of Victoria for their guidance and support.

DEDICATION

For my family, who have always been there for me.

Chapter 1

Introduction

1.1 Motivation and Problem Statement

Malware, short for malicious software, refers to software specifically designed to disrupt, damage, or gain unauthorized access to computer systems. The growing sophistication and volume of malware pose significant challenges to cybersecurity, necessitating advanced methods for effective detection and classification. On one hand, traditional signature-based malware detection techniques are becoming increasingly inadequate due to the rapid evolution of malware variants. On the other hand, behavioral and anomaly detection models which potentially could detect novelty, involves manual feature extraction, which can be time-consuming and prone to missing subtle malicious patterns. In this context the focus of this thesis is on utilizing static analysis through deep learning methods to enhance malware detection and classification. By employing deep learning techniques, we aim to bypass the manual feature extraction process and instead leverage the power of image-based models to identify generalization patterns in malware binaries. This approach involves converting malware binaries into images and training deep learning models to recognize and classify these images, thereby enabling a more automated and efficient analysis. Also image-based methods for static analysis removes the constraint of requiring a sandboxed environment for execution, as the analysis is conducted on the converted image files rather than the executable binaries themselves. Once the binary file has been converted into an image, it can be analyzed without the risks associated with executing potentially harmful code.By converting malware binaries into images, we can leverage powerful image recognition techniques to identify and categorize malicious software. The MalNet-Image dataset (chapter 3), the

largest publicly available cybersecurity image database, offers a substantial resource for this purpose. This dataset includes over 1.2 million binary images, categorized into various malware families and types, providing a rich basis for developing and evaluating image-based malware detection models.

1.2 Summary of Contributions

The primary contributions of this thesis are as follows:

1. **Novel Application of Feature Pyramid Networks (FPN)**: We introduce the use of FPN to handle varying malware object sizes when converting malware binaries into images, addressing the scale issue effectively.

2. **Advanced Data Augmentation Techniques**: We apply Mixup and Trivial Augment techniques, enhancing the robustness and generalization capabilities of our models.

3. **AdamW schedule-free and Transfer Learning**: Exploring pre-trained models on ImageNet for Malware image classification and utilizing a novel AdamW schedule-free optimizer we demonstrate performance improvements with lower training run time.

4. **Comprehensive Experimental Evaluation**: Our experiments on the MalNet-Image-Tiny dataset show that the integration of these techniques leads to a notable increase in classification accuracy, achieving an F1 score of 0.6927, compared to 0.65 reported on the provided split for MalNet-Image-Tiny (Freitas et al., 2022).

These contributions collectively exceed the state-of-the-art (SOTA) on MalNet-Tiny, offering a strong foundation for applying image-based deep learning approaches to malware classification.

1.3 Outline

The structure of this thesis is as follows:

- **Chapter 2: Background and Literature Review** - This chapter provides an overview of malware types and analysis methods, and reviews existing literature on malware detection techniques, focusing on image-based approaches.

- **Chapter 3: Dataset and Baseline Model Selection** - This chapter details the MalNet-Image and MalNet-Tiny datasets used, the image conversion process, and the backbone model selection for malware classification tasks.
- **Chapter 4: Methodology** - This chapter explains the deep learning models and techniques employed in this study, including Feature Pyramid Networks (FPN), data augmentation methods, transfer learning, and the schedule-free Adam optimizer.
- **Chapter 5: Experiment Setup and Results** - This chapter describes the experimental setup, evaluation metrics, and presents the results of our experiments.
- **Chapter 6: Conclusion and Future Work** - This chapter discusses the findings, their implications, and potential directions for future research.

Chapter 2

Background and Literature Review

2.1 Background

Malware, short for malicious software, refers to any software intentionally designed to cause damage to a computer, server, client, or computer network. The term encompasses a variety of hostile or intrusive software, including viruses, worms, trojans, ransomware, spyware, adware, among other malicious programs. These programs can execute unauthorized actions on a victim's system, such as stealing sensitive information, corrupting data, disrupting system performance, or gaining unauthorized access to network resources. Malware is often distributed through email attachments, infected websites, or malicious downloads, exploiting vulnerabilities in software and user behavior to propagate and inflict harm.

Malware operates by executing harmful code that can manipulate, disrupt, or destroy system functionality and data integrity. It can be programmed to perform various malicious activities, such as logging keystrokes, encrypting files for ransom, or using infected systems to launch further attacks. The motivations behind malware creation range from financial gain and espionage to ideological objectives and sheer disruption. Due to its diverse forms and evolving nature, malware poses significant challenges to cybersecurity, necessitating continuous advancements in detection, prevention, and response strategies to mitigate its impact effectively.

2.1.1 Malware Types

Malware can be classified into various types based on their behavior and execution methods , each posing unique threats and challenges to cybersecurity. These classifications

help in understanding the specific mechanisms through which malware operates, spreads, and impacts systems (Pachhala et al., 2021) . As illustrated in Table 2.1, the diverse range of malware includes viruses, worms, trojans, ransomware, spyware, adware, rootkits, keyloggers, botnets, and backdoors. Each type exhibits distinct characteristics and attack vectors, necessitating tailored detection and mitigation strategies to effectively combat the different nature of these malicious programs.

Malware Type	**Description**
Virus	A type of malware that attaches itself to a legitimate program or file and spreads to other programs and files when executed, causing harm to the system.
Worm	A standalone malware that replicates itself to spread to other computers, often using a network, without needing to attach to a host program.
Trojan Horse	Malicious software that disguises itself as a legitimate application or file to trick users into installing it, allowing unauthorized access to the system.
Ransomware	Malware that encrypts a victim's files or locks their system, demanding a ransom payment to restore access.
Spyware	Software that secretly monitors and collects information about a user's activities without their consent, often used for data theft.
Adware	Unwanted software that displays advertisements on a user's system, often bundled with legitimate software, and may track browsing habits.
Rootkit	A set of tools used by an attacker to gain unauthorized root or administrative access to a computer and hide the presence of other malware.
Keylogger	A type of spyware that records keystrokes on a user's keyboard to capture sensitive information such as passwords and credit card numbers.
Botnet	A network of infected computers controlled by an attacker, often used to launch distributed denial-of-service (DDoS) attacks or send spam emails.
Backdoor	Malware that creates a hidden entry point into a system, allowing an attacker to bypass normal authentication and gain remote access.

Table 2.1: Common Malware Types and Their Descriptions

2.1.2 Malware Analysis and Detection

Malware analysis is a critical process in the field of cybersecurity, aimed at understanding the functionality, origin, and potential impact of malicious software. This process involves dissecting malware samples to uncover their underlying code, behavior, and intent (Sibi Chakkaravarthy et al., 2019). There are two primary approaches to malware analysis: static analysis and dynamic analysis.

Static analysis entails examining the malware's code without executing it, using techniques such as reverse engineering and code decompilation to identify its structure and embedded functionalities. This approach can reveal valuable information about the malware's capabilities, such as payload delivery mechanisms, obfuscation techniques, and communication protocols.

The static analysis process typically begins with the disassembly or decompilation of the malware binary. Disassembly converts the binary code into assembly language, which is a low-level representation that can be analyzed by security researchers. Decompilation, on the other hand, translates the binary into a higher-level programming language, making it easier to understand the malware's logic and control flow. These steps are crucial for identifying the specific functions and routines within the malware, such as encryption algorithms, network communication protocols, and file manipulation procedures.

Advanced static analysis techniques also involve the use of automated tools and scripts to scan for known malicious signatures and patterns. These tools can quickly detect common indicators of compromise, such as unusual API calls, suspicious strings, and obfuscation techniques. Obfuscation, a common tactic used by malware developers to evade detection, involves altering the code to make it difficult to analyze. Static analysis tools can deobfuscate code, revealing the underlying malicious intent and functionality.

Dynamic analysis on the other hand, involves executing the malware in a controlled environment, such as a sandbox, to observe its behavior in real-time. This method allows analysts to monitor the malware's interactions with the system, network communications, and any changes it makes to the environment. By capturing these activities, dynamic analysis can provide insights into the malware's execution flow, persistence mechanisms, and the specific actions it performs upon infection. Additionally, mem-

ory analysis is often integrated within dynamic analysis to examine the volatile memory (RAM) of a system. This aspect of analysis can uncover hidden processes, injected code, and in-memory exploits that do not leave traces on the disk, offering a deeper understanding of the malware's behavior and impact.

2.2 Literature Review

Leveraging deep learning represents a significant advancement in the field of malware detection, providing capabilities that surpass traditional methods (Vinayakumar et al., 2019). By utilizing neural networks, deep learning models can automatically learn and extract complex patterns from datasets with different sizes, enhancing the accuracy and robustness of malware detection systems.

2.2.1 Malware Detection Based on Natural language Processing (NLP) Techniques

In the realm of malware detection, (NLP) techniques offer a novel approach by treating sequences of API calls, opcode sequences, and other code-related patterns as text data. These methods leverage advanced NLP models such as transformers, and recurrent neural networks to analyze and classify malware based on the behavioral patterns and sequences observed in the code.

Demirknran et al. (2022) presented an approach to multiclass malware classification by employing an ensemble of pre-trained transformer models. Utilizing API call sequences as features, the study contrasts traditional machine learning models with transformer-based models, highlighting the superior performance of the latter due to their ability to capture sequence relationships. Experiments demonstrated that pre-trained transformer models, specifically BERT (Devlin et al., 2019) and CANINE (Clark et al., 2022), outperform traditional models in handling imbalanced datasets. The proposed Random Transformer Forest (RTF) model, an ensemble of BERT (Devlin et al., 2019) and CANINE (Clark et al., 2022), achieved state-of-the-art results on three out of four datasets: Catak (7,107 samples), Oliveira (40,566 samples), VirusSample (9,732 samples), and VirusShare (13,849 samples), showcasing its efficacy in multiclass malware classification tasks.

Yesir and Sogukpinar (2021) investigated the application of NLP techniques, specifically fastText (Joulin et al., 2016) and BERT, for the detection and classification of mal-

ware. Their study employs dynamic analysis to extract API call sequences from malware, which are then optimized by removing redundant and noisy calls. These optimized sequences are used as input for classification tasks using fastText and BERT algorithms. The authors' experiments, conducted on three distinct open datasets : CSDMC (388 samples), APIMDS (1,329 samples (Ki et al., 2015)), and a custom dataset (5,000 samples), revealed that the fastText model generally achieved higher accuracy and efficiency in malware detection and classification compared to the BERT model. This work underscored the effectiveness of NLP methods in handling the complex and repetitive nature of malware API call sequences for improved malware detection.

Zhang et al. (2021) proposed CoDroid, a hybrid sequence-based Android malware detection method utilizing both static opcode sequences and dynamic system call sequences. This approach leverages a CNN (LeCun et al., 1998)-BiLSTM (Graves and Schmidhuber, 2005)-Attention (Bahdanau et al., 2015) model to classify Android applications as benign or malicious, treating sequence data as sentences in NLP. The model captures global sequence features through the BiLSTM layer and emphasizes critical features with the attention mechanism. The study evaluated CoDroid on a real-world dataset comprising 2,978 malicious applications from the DREBIN dataset and 2,707 benign applications from the PlayDrone dataset. The experimental results demonstrated that CoDroid outperformed several traditional machine learning algorithms and related advanced approaches, achieving an accuracy of 97% in malware detection.

2.2.2 Malware Detection Based on Graphs

Graph-based techniques for malware detection leverage the structural properties of code and system interactions, representing them as graphs to uncover complex relationships and patterns. By employing methods such as graph convolutional networks (GCNs) and graph embeddings, these approaches capture the interconnected nature of API calls, function calls, and other code components. This allows for a detailed analysis of malware behavior, enabling the identification and classification of malicious software based on the structural signatures embedded within the graph representations. Jiang et al. (2018) introduced DLGraph, a malware detection approach that integrates deep learning with graph embedding techniques. DLGraph utilized two stacked denoising autoencoders (SDAs) to learn latent representations of function-call graphs and Windows API calls. The function-call graphs are mapped to low-dimensional vectors using the node2vec (Grover and Leskovec, 2016) graph embedding technique, while the API calls

are transformed into binary vectors. These representations are combined and classified using a softmax regression layer to detect malware. The authors evaluated DLGraph on three datasets: Dataset 1 (2,434 Lollipop samples and 2,177 benign samples), Dataset 2 (2,584 Kelihos-ver3 samples and 2,177 benign samples), and Dataset 3 (5,018 malicious samples and 2,177 benign samples). The experimental results demdemonstrated that DLGraph achieved high accuracy across all datasets, outperforming the DL4MD method by leveraging function-call graphs for enhanced malware detection.

Pei et al. (2020) introduced AMalNet, a deep learning framework utilizing Graph Convolutional Networks (GCNs (Kipf and Welling, 2017)) and Independently Recurrent Neural Networks (IndRNNs (Li et al., 2018)) for malware detection and family attribution. The framework employs static analysis to extract API call sequences, which are then transformed into graph representations. These representations are processed using GCNs to capture high-level graphical semantics, and IndRNNs to decode deep semantic information, effectively leveraging remote dependencies between nodes. The authors evaluated AMalNet on multiple benchmark datasets, including DREBIN (5,560 malware samples), AMD (24,553 malware samples), a lab-built dataset (4,664 benign samples), and AndroZoo (100,000 samples). The experimental results demonstrated that AMalNet significantly outperforms other state-of-the-art techniques, showcasing its robustness and efficiency in handling large-scale malware detection tasks.

Gao et al. (2021) proposed GDroid, an approach for Android malware detection and familial classification based on Graph Convolutional Networks (GCNs). This method maps apps and Android APIs into a large heterogeneous graph, converting the detection problem into a node classification task. The authors utilized two types of edges, "App-API" and "API-API," based on invocation relationships and API usage patterns. Their prototype system, GDroid, was evaluated on several datasets, including GP-AMD with 3,300 samples (1,200 malware and 2,100 benign), and achieved a malware detection rate of 98.99% with a low false positive rate of less than 1%. Additionally, GDroid achieved an average accuracy of almost 97% in malware familial classification tasks.

2.2.3 Malware Detection Based on Image Processing

Nataraj et al. (2011a) proposed a novel method for malware visualization and classification by converting malware binaries into grayscale images. As far as we know, this is the first paper to introduced such a conversion, the majority of the literature adheres to the initial three steps outlined in section 3.1.2 following this propsed method. This

approach leverages the observation that malware samples from the same family exhibit similar visual patterns and textures. The authors employed standard image processing techniques, specifically using GIST descriptors, to extract texture features from these images for classification. Their experiments on a dataset of 9,458 samples spanning 25 different malware families yielded a classification accuracy of 98%. The study also highlighted the method's resilience to popular obfuscation techniques, such as section encryption, as it does not require disassembly or code execution. This image-based technique offers a significant computational advantage and robustness.

Alguliyev et al. (2024) presented a novel approach for malware classification in cyber-physical systems (CPS) using the Radon transform (Radon, 1917) and deep learning models. The study aims to address the increasing sophistication of malware targeting CPS by combining visual representations of malware and detection models based on transfer learning. The authors proposed converting malware binaries into grayscale images and applying the Radon transform to these images. The Radon transform, commonly used in computed tomography, captures the texture and structural information of the images by projecting them onto a space of line integrals, effectively highlighting the characteristic features of malware families. This transformation enhances the ability to differentiate between various malware types by focusing on their unique textural patterns. The framework integrates two pre-trained deep neural networks, AlexNet (Krizhevsky et al., 2012) and MobileNet (Howard et al., 2017), to process the grayscale and Radon-transformed images, respectively. The proposed model was evaluated on three datasets: Microsoft Malware Classification, IoT_Malware, and MalNet-Image. The Microsoft Malware Classification (Microsoft, 2015) dataset contains 10,868 training samples and 10,873 testing samples across nine malware families. The IoT_Malware dataset (Alasmary et al., 2020) includes 3,016 benign samples and 13,798 malicious samples spanning three malware families. The MalNet-Image dataset comprises over 1.2 million binary images, of which 60,437 malware samples and 19,736 benign samples are used for experiments. The experimental results demonstrated the superior classification accuracy of the proposed model, achieving high accuracy rates on the Microsoft malware dataset (99.89%), IoT_Malware dataset (99.95%), and MalNet-Image dataset (99.20%). This approach not only improves classification accuracy but also offers robustness against obfuscation techniques, making it a valuable tool for malware detection in CPS.

Chen et al. (2020) presented the STAMINA (STAtic Malware-as-Image Network Analysis) approach, which utilized deep learning for static malware classification. To handle

the highly skewed distribution of file sizes in their dataset, the authors implemented a file size gating mechanism. Files are categorized into ranges based on their pixel file sizes, with specific widths assigned for different size ranges, such as 32 pixels for files between 0 to 10 KB and up to 2048 pixels for files larger than 1500 KB. This method ensures that the structural integrity of the images is maintained for effective classification. Two models were developed using this file size gating approach with 3.9MB as the cut-off gate. The first model, STAMINA Model I, converts malware binaries into grayscale images and resizes them according to the predefined pixel file size table. Using the Inception-v1 model for fine-tuning, this model achieved a 99.07% accuracy with a 2.58% false positive rate, a precision of 99.09%, and a recall of 99.66%. The second model, STAMINA Model II, segments large benign files into multiple images while resizing malicious files into single images. This segmentation helps address data imbalance between benign and malicious files. Trained on 31,965 segmented benign images and 58,630 resized malware images, this model achieved a 95.97% accuracy, a 4.49% false positive rate, a precision of 83.88%, and a recall of 97.90%.

Kumar and Panda (2023) proposed the SDIF-CNN (Stacking Deep Image Features using Fine-tuned Convolutional Neural Network models) for real-world malware detection and classification. The study utilized pre-trained CNN models, specifically VGG16, VGG19, ResNet50, and InceptionV3, which were fine-tuned to enhance malware detection. The feature maps extracted from these models were horizontally concatenated to create a single feature map. The datasets employed include the publicly benchmarked MalImg dataset, consisting of 9339 images from 25 malware families, and malware samples collected from honeypots.

Tekerek and Yapici (2022) introduced a malware classification and augmentation model based on convolutional neural networks (CNNs). The study introduced a method called B2IMG for transforming byte files into grayscale and RGB image formats for classification. Additionally, the authors implemented a CycleGAN-based (Zhu et al., 2017) data augmentation technique to address the imbalanced data distribution among malware families. The proposed system was evaluated on two datasets: The Microsoft Malware Classification, and the DumpWare10 (Bozkir et al., 2021) dataset, which includes 4294 samples across 11 categories (10 malware families and 1 benign class). The Microsoft Malware Classification dataset was converted into JPG images using the B2IMG method, and the experiments were conducted on both RGB and grayscale images. Experimental results demonstrated that the proposed CycleGAN-based data augmentation improves classification performance. For the Microsoft Malware Classification, the

accuracy increased to 99.86% for RGB images and 99.73% for grayscale images with augmentation. Without augmentation, the accuracy was 99.76% for RGB images and 99.58% for grayscale images; These results highlight the effectiveness of data augmentation and that it works better with RGB images than grayscale images.

Discussion

The existing literature on image-based deep learning models for malware classification presents several significant advancements and insights, yet it also reveals notable gaps that warrant further exploration.

Firstly, the issue of varying file lengths impacting the scales of code blocks within malware binaries is largely unaddressed, with the exception of the STAMINA approach Chen et al. (2020). After converting bytecode to images and resizing them to a standard size for training, the multi-scale object issue emerges, affecting the consistency of features across different samples. STAMINA implemented a file size gating mechanism, categorizing files into different pixel width ranges based on their sizes and employing two separate models for classification. This manual segmentation approach acknowledges the challenge posed by file length variations. However, no other studies have systematically tackled this problem, potentially leading to inconsistent scales and features that can adversely affect classification accuracy. An end-to-end arcitecture for handling different file sizes uniformly is needed to improve the generalizability of malware classifiers and avoding deploying multiple models.

Secondly, while the MalNet-Image dataset is the largest public repository of malware images, it remains underutilized in the current body of research. Alguliyev et al. (2024) made a contribution by employing this dataset, but they deviated from the standard split typically used for benchmarking. This deviation complicates direct comparison with other studies. Additionally, given the scale of the MalNet-Image dataset, exploring advanced optimization techniques is crucial. Recent developments in optimization algorithms, such as the Schedule-Free optimizer (Defazio et al., 2024), promise improved performance and stability over traditional scheduled optimizers. Incorporating these newer optimizer could enhance the efficiency and effectiveness of deep learning models in handling large-scale datasets like MalNet-Image.

Lastly, data augmentation techniques play a critical role in enhancing the performance of deep learning models, particularly in scenarios with imbalanced datasets. While GAN-based augmentations, as demonstrated by Tekerek and Yapici (2022), have shown

slight improvement in classification accuracy, alternative techniques such as MixUp (Zhang et al., 2018) and TrivialAugment (Mueller and Hutter, 2021), which have proven successful in other domains, remain unexplored in malware classification. These database-independent augmentation methods provide a robust way to enhance model generalization by creating training samples through data mixing or slight alterations. Despite their widespread use in other fields, no studies have yet investigated the potential of MixUp and TrivialAugment in the context of malware detection, representing an important avenue for future research.

2.2.4 Datasets

Table 2.2: Summary of Image-Based Malware Datasets
bold dataset is used

Dataset	Public/Private	Images	Classes
MalNet-Image (Freitas et al., 2022)	Public	1,262,024	696
MalNet-Image Tiny (Freitas et al., 2022)	**Public**	**87,430**	43
Virus-MNIST (Noever and Noever, 2021)	Public	51,880	10
Malimg (Nataraj et al., 2011b)	Public	9,458	25
AndroDex (Khan et al., 2024)	Public	24,746	179
Stamina (Chen et al., 2020)	Private	782,224	2
McAfee (Labs, 2020)	Private	367,183	2
Kancherla (Kancherla and Mukkamala, 2013)	Private	27,000	2
Choi (Choi et al., 2017)	Private	12,000	2
Fu (Fu et al., 2018)	Private	7,087	15
Han (Han et al., 2015)	Private	1,000	50
IoT DDoS (Lab, 2020)	Private	365	3

In this section, we summarize various image-based malware datasets used for developing and benchmarking malware detection and classification algorithms. MalNet-Image (Freitas et al., 2022) is the largest publicly available image database for cybersecurity research, containing over 1.2 million images categorized into 696 malware families and 47 types. The dataset provides a comprehensive resource for developing and benchmarking image-based malware detection and classification algorithms. It aims to democratize access to large-scale malware image data, facilitating advancements in machine learning and cybersecurity research.

MalNet-Image Tiny (Freitas et al., 2022) is a subset of the MalNet-Image dataset, designed for rapid prototyping and experimentation. It contains 87,430 images across

43 classes, excluding the four largest types from the full MalNet-Image dataset. This reduced dataset allows researchers to quickly test new ideas and methodologies on a smaller scale while still maintaining a diverse set of malware images.

Virus-MNIST (Noever and Noever, 2021) is a benchmark malware dataset comprising approximately 51,880 images of 10 classes, including nine malware families and one benign class. The dataset mimics the MNIST format for ease of use with existing image classification algorithms. It serves as a benchmark for evaluating malware detection and classification methods.

Malimg (Nataraj et al., 2011b) is a publicly available dataset consisting of 9,458 malware images categorized into 25 different families. This dataset is widely used for malware visualization and classification research, providing a smaller-scale yet diverse set of malware images for evaluating various machine learning techniques.

AndroDex (Khan et al., 2024) is a dataset consisting of 24,746 Android malware samples spanning 179 malware families. The dataset includes images of `.dex` files, which are crucial for understanding malware behavior. The dataset features obfuscated malware, benign applications, and various obfuscation techniques to enhance the robustness of malware detection systems. AndroDex aims to aid researchers in developing advanced machine learning models for Android malware classification and detection by providing a rich and diverse set of malware images.

The Stamina dataset (Chen et al., 2020) is a private collection of 782,224 malware images, divided into two classes. It was developed through a collaboration between Intel and Microsoft and focuses on scalable deep learning approaches for malware classification. The dataset is not publicly accessible but is used extensively in industry research.

The McAfee dataset (Labs, 2020) comprises 367,183 images and is used for malware detection. It is a private dataset maintained by McAfee Labs and includes two classes: malicious and benign. This dataset supports the development of machine learning models for identifying malware within large-scale enterprise environments.

The Kancherla dataset (Kancherla and Mukkamala, 2013) is a private collection of 27,000 malware images, categorized into two classes. It is used for research in image-based malware detection and visualization. The dataset supports studies in transforming malware binaries into images for classification using machine learning techniques.

The Choi dataset (Choi et al., 2017) contains 12,000 images and is privately maintained. It includes two classes and is utilized for research in malware detection using deep learning and image processing methods. The dataset provides a resource for developing algorithms that analyze malware images.

The Fu dataset (Fu et al., 2018) consists of 7,087 images classified into 15 malware families. It is a private dataset used for fine-grained classification of malware through image visualization techniques. This dataset supports detailed analysis and categorization of various malware types.

The Han dataset (Han et al., 2015) includes 1,000 images and is privately maintained, featuring 50 different classes. It is used for malware analysis through visualized images and entropy graphs. The dataset aids in the study of diverse malware types and their visual characteristics.

The IoT DDoS dataset (Lab, 2020) is a private collection of 365 images, categorized into three classes. It focuses on detecting Distributed Denial of Service (DDoS) attacks in IoT environments. The dataset supports research in securing IoT devices against large-scale network attacks.

In conclusion, we will utilized the MalNet-Image Tiny dataset for our experiments. Its smaller size allows for more rapid prototyping and experimentation, significantly reducing computational requirements without compromising the diversity of the malware images. Despite its reduced scale, MalNet-Image Tiny remains the second largest publicly available image-based dataset for cybersecurity research.

Chapter 3

Dataset and Baseline Model Selection

3.1 Dataset Description

The primary dataset referenced in this study is the MalNet-Image dataset (Freitas et al., 2022), recognized as the largest publicly available cybersecurity image database designed for malware detection and classification. MalNet-Image comprises over 1.2 million binary images of malware, organized into 47 types and 696 families. This study specifically utilizes the MalNet-Image Tiny version, a smaller subset designed to enable faster experimentation while maintaining a representative diversity of the larger dataset. Both the MalNet-Image and MalNet-Image Tiny versions exhibit class imbalances, with the imbalance being more pronounced in the full MalNet-Image dataset, as illustrated in fig. 3.2 and fig. 3.3, respectively. This comprehensive dataset is a critical resource for advancing research in image-based malware detection. In the dataset section, we will describe the MalNet-Image and MalNet-Image Tiny datasets, outlining their structure, data collection methods, image conversion processes, and dataset splits. We will also discuss the class imbalances observed in both datasets.

3.1.1 Data Collection

MalNet-Image was constructed using Android APK files sourced from the AndroZoo repository. The decision to focus on the Android ecosystem is driven by its substantial market share and the rich diversity of malicious software targeting this platform. The dataset includes APKs with both family and type labels, obtained using Euphony, a malware labeling system that aggregates and learns from the labeling results of up to 70 antivirus vendors from VirusTotal. This comprehensive labeling approach ensures a

good level of accuracy and detail in the dataset.

3.1.2 Image Conversion Process

The image conversion process fig. 3.1 involves transforming Android APK files into binary images suitable for analysis by deep learning models. This process consists of several key steps, detailed as follows:

1. **bytecode extraction - DEX File Extraction**: The first step is to extract the Dalvik Executable (DEX) file from the APK. The DEX file contains the compiled bytecode for the Android application, which represents the actual executable instructions of the app. This step is crucial because the bytecode encapsulated in the DEX file is what we aim to analyze and convert into an image format. The extraction is typically done using tools such as `dex2jar` or similar utilities that can parse the APK and retrieve the DEX content.

2. **Byte Array Generation**: Once the DEX file is extracted, it is converted into a one-dimensional array of 8-bit unsigned integers. This array represents the binary content of the DEX file, with each entry in the array corresponding to a byte. The values in this array range from 0 (black) to 255 (white), which are the grayscale values used in the subsequent image formation step.

3. **Image Formation**: The one-dimensional byte array is reshaped into a two-dimensional array using standard linear plotting techniques. This reshaping converts the linear byte sequence into a grid-like structure, forming the basis of the image. The width of the image is fixed based on file pixel size mapping table, while the height of the image varies depending on the total number of bytes in the DEX file.

4. **Scaling and Coloring**: The resulting images are scaled to a standard size of 256x256 pixels using a Lanczos filter. This resizing ensures consistent image dimensions across the dataset, which is important for feeding into deep learning models that expect uniform input sizes. Each byte in the image is color-coded according to its position within the DEX file structure—header, identifiers, class definitions, and data sections. This color-coding adds semantic information to the images, providing additional context about the structure and content of the malware.

By following these steps, we transform the raw bytecode of Android applications into

a visual format that can be analyzed using image-based deep learning techniques, facilitating the detection and classification of malware.

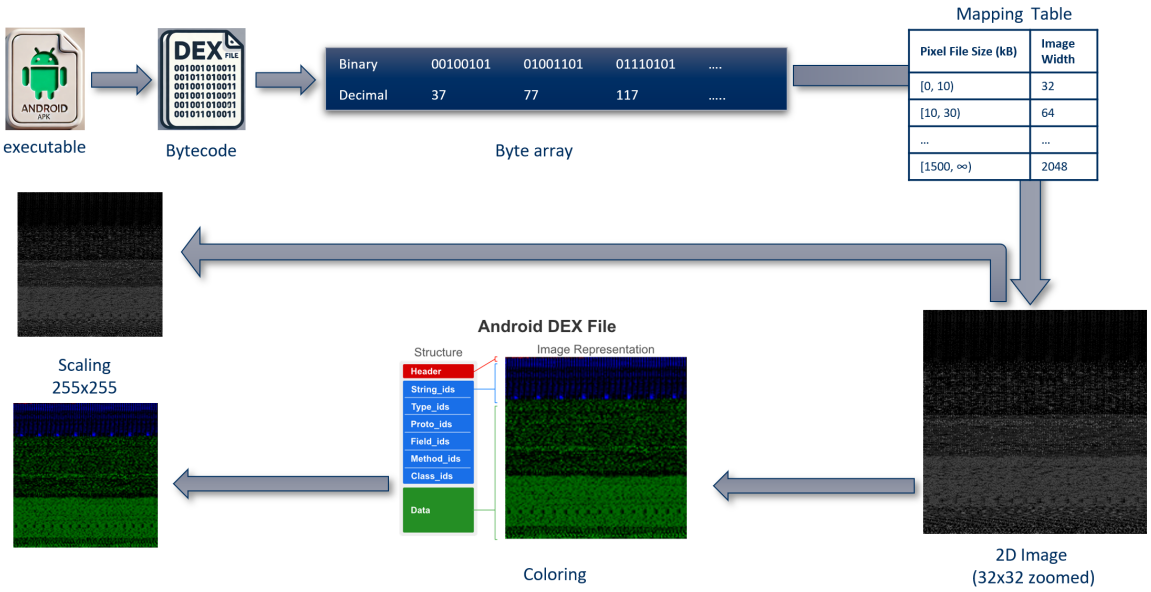

Figure 3.1: Image Conversion Process

3.1.3 Dataset Splits

In this subsection, we describe the division of the MalNet-Image and MalNet-Image Tiny datasets into training, validation, and test sets, as well as highlight key issues such as class imbalance and labeling discrepancies.

The MalNet-Image dataset is stratified into three subsets to facilitate effective training, validation, and testing of machine learning models:

- **Training Set (70%)**: This subset comprises 883,417 images and is used for model training.
- **Validation Set (10%)**: This subset contains 126,202 images and is used for hyperparameter optimization.
- **Test Set (20%)**: This subset includes 252,405 images and is used to evaluate the generalization performance of the trained models.

Similarly, the MalNet-Image Tiny version, designed for rapid prototyping and experimentation, is divided into three subsets:

- **Training Set (70%)**: This subset comprises 61,201 images.

- **Validation Set (10%)**: This subset contains 8,743 images.
- **Test Set (20%)**: This subset includes 17,486 images.

The MalNet-Image Tiny subset was created by excluding the four largest types from the full dataset, resulting in a more balanced dataset for classification tasks.

fig. 3.2 and **fig. 3.3** illustrate the class distribution of the MalNet-Image dataset and its tiny subset, focusing on the relationship between malware types and their largest families. The y-axis represents different malware types, while the x-axis lists various malware families. The color intensity indicates the log-transformed count of samples for each type-family pair, with darker colors representing higher counts. fig. 3.2 the class distribution for the full MalNet-Image dataset. This figure reveals a significant class imbalance, as indicated by the scattered and varied intensity of the blue areas across the plot. Certain malware types and families have high representation, shown by the dark blue streaks, while many others are underrepresented, as evidenced by the lighter blue areas. For example, types like ”adware,” ”trojan,” and ”riskware” have high counts across several families, leading to a concentration of dark blue areas. On the other hand, types such as ”monitor” and ”smsend” have fewer samples, shown by lighter blue regions. The uneven distribution suggests that some type-family pairs dominate the dataset, leading to challenges in model training and potential bias towards the more frequent classes. While fig. 3.3 exhibits a more balanced distribution, as indicated by the relatively consistent and intense blue areas along the diagonal. The more even intensity of the blue regions suggests that the tiny subset has a more equitable representation of different types and families, with fewer extreme imbalances

Additionally, there are labeling issues within the dataset, such as inconsistencies between similar malware types, like “adsware” and “adware.” These discrepancies necessitate careful data preprocessing and potential relabeling to ensure the accuracy and reliability of the classification models.

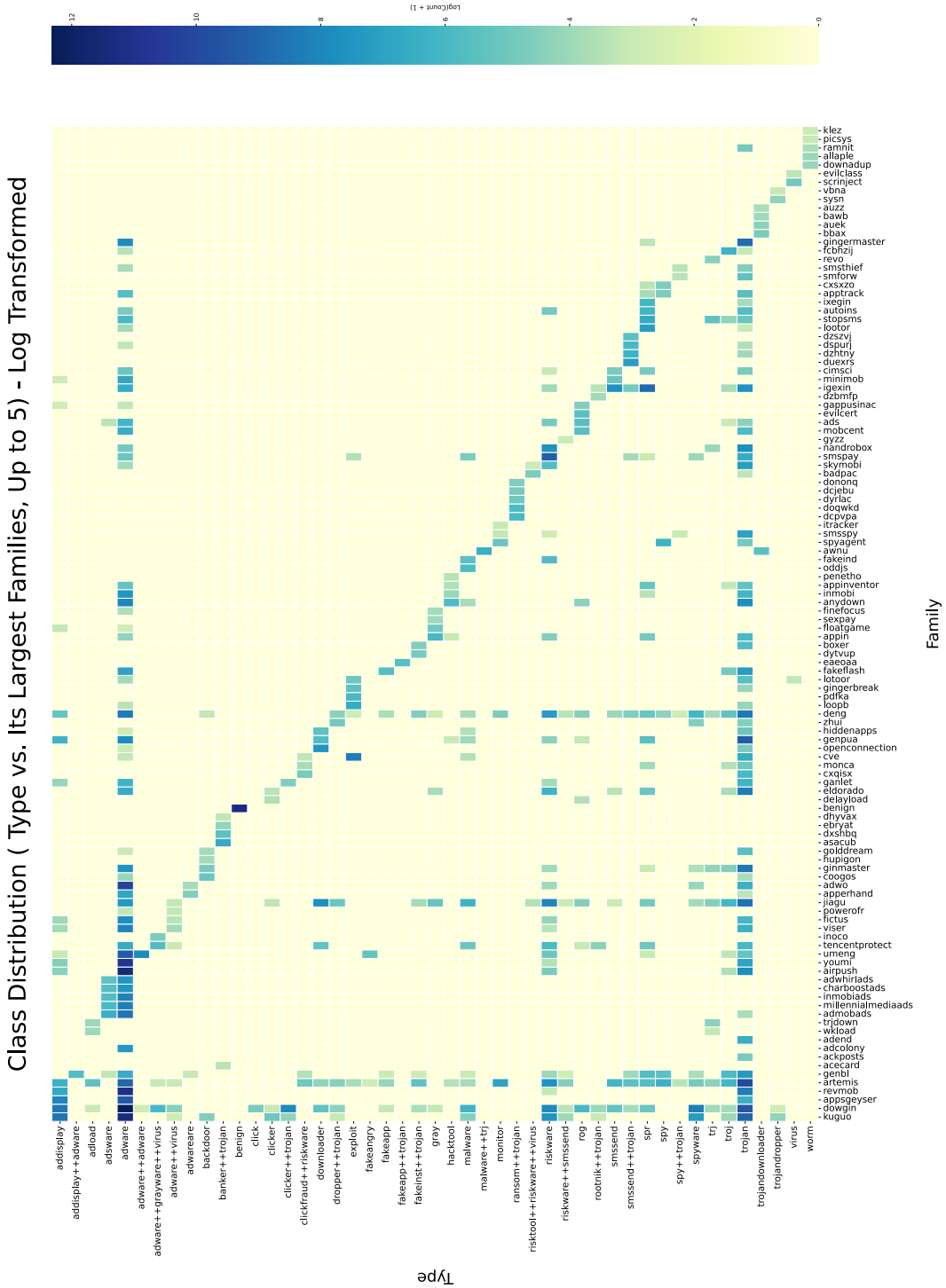

Figure 3.2: MalNet-Image Class Distribution

This figure reveals a significant class imbalance, with certain malware types and families showing high representation like adware and trojan, while many others are underrepresented

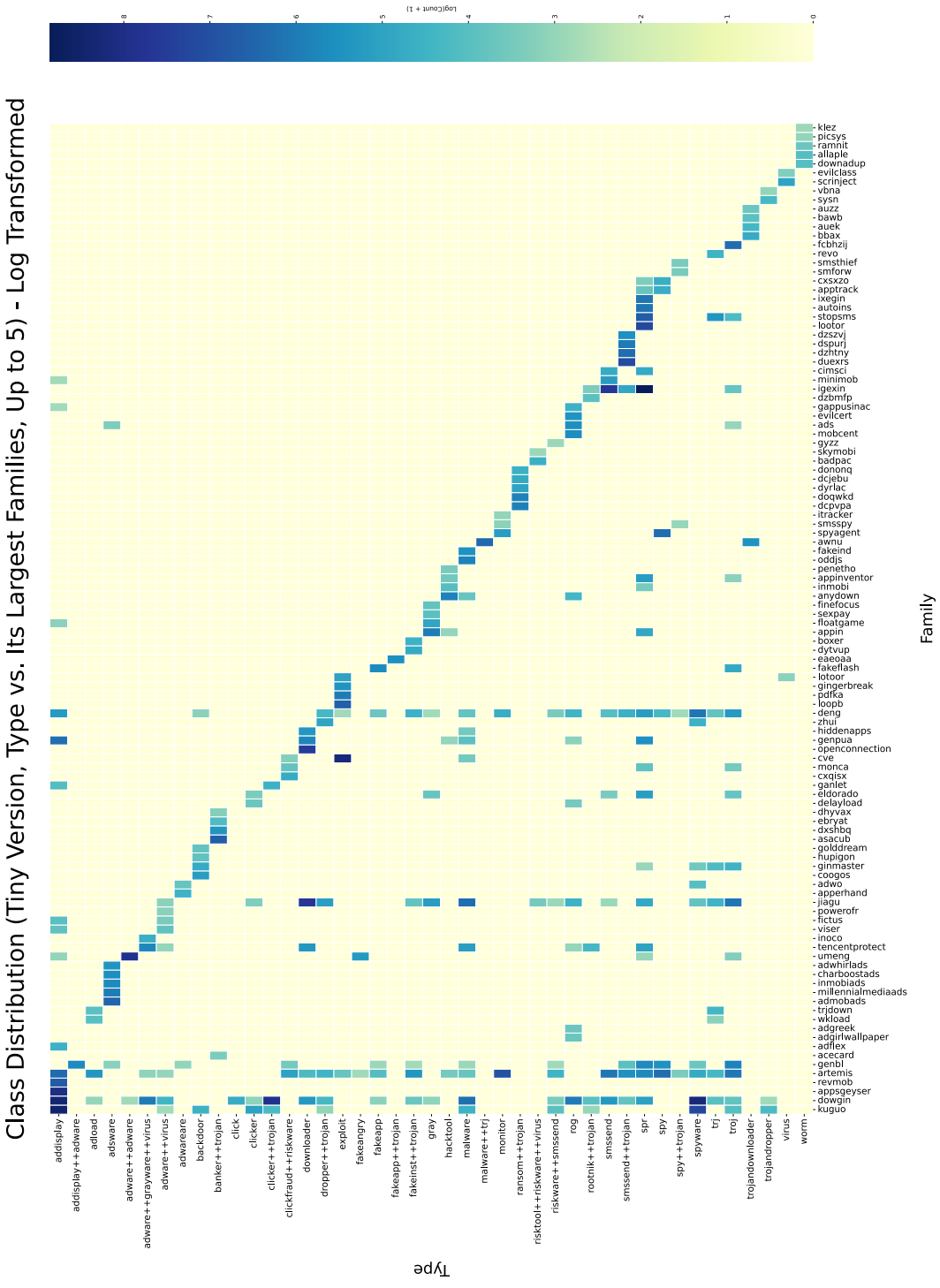

Figure 3.3: MalNet-Image-**Tiny** Class Distribution

This figure exhibits a more balanced distribution, with a relatively consistent and intense blue diagonal indicating comparable sample counts across different types and families

3.2 Backbone Model Selection: ResNet for Malware Classification

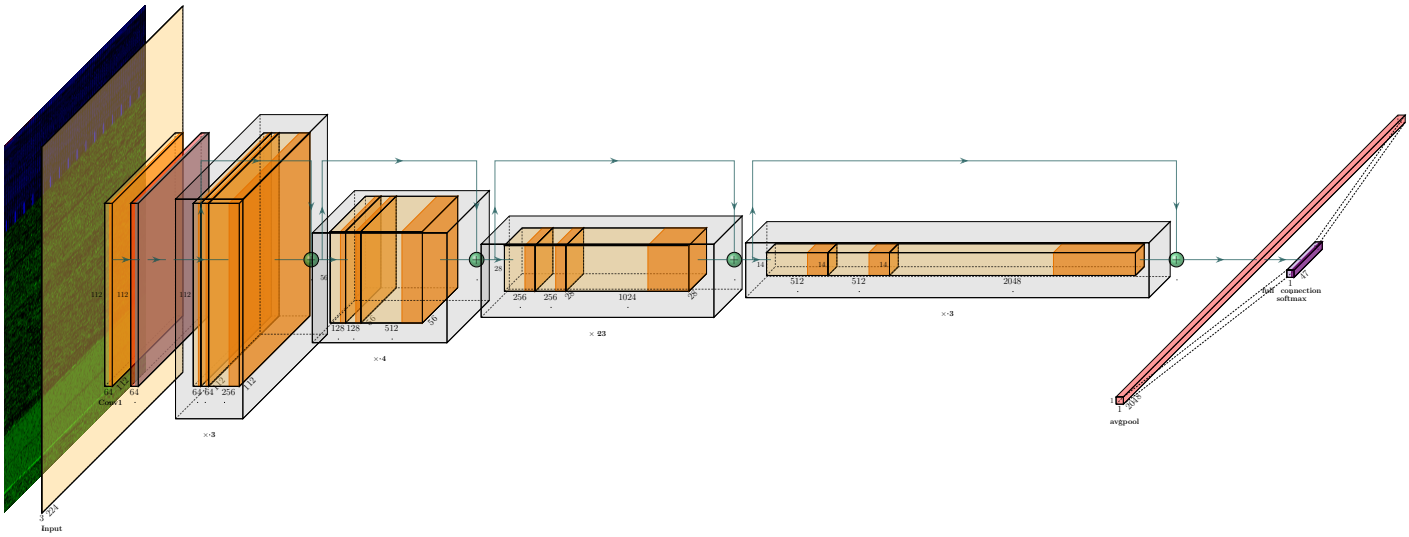

Figure 3.4: ResNet Architecture

The *MalNet-Image* dataset facilitates comprehensive research into malware classification using deep learning models, particularly focusing on Convolutional Neural Networks (CNNs) based architectures. The results of the experiments conducted in (Freitas et al., 2022) with ResNet (He et al., 2016), DenseNet (Huang et al., 2017), and MobileNetV2 (Sandler et al., 2018) architectures, are summarized in tables: 3.1. In this section, we delve into the rationale behind selecting ResNet18 as the backbone model for malware classification tasks.

3.2.1 Model Selection

Based on the results from Table 3.1 ResNet18 provides an optimal balance between performance and computational efficiency in malware classification tasks. Here's a detailed explanation:

Computational Efficiency: ResNet18 has significantly fewer parameters (12M) and lower GFlops (1.8) compared to deeper versions like ResNet50 (26M, 3.9 GFlops) and

Model	Params (M)	GFlops	Binary			Type			Family		
			F1	P	R	F1	P	R	F1	P	R
ResNet18	12	1.8	0.86	0.89	0.84	0.47	0.56	0.42	0.45	0.54	0.42
ResNet50	26	3.9	0.85	0.91	0.81	0.48	0.57	0.44	0.47	0.54	0.44
ResNet101	45	7.6	0.86	0.88	0.84	0.48	0.59	0.44	0.47	0.54	0.44
DenseNet121	7.9	2.9	0.86	0.90	0.83	0.47	0.56	0.43	0.46	0.53	0.44
DenseNet169	14	3.4	0.86	0.89	0.84	0.48	0.57	0.43	0.46	0.55	0.43
MobileNetV2(x0.5)	1.9	0.1	0.86	0.89	0.83	0.46	0.55	0.42	0.45	0.53	0.42
MobileNetV2(x1.0)	3.5	0.3	0.85	0.89	0.83	0.45	0.53	0.42	0.44	0.53	0.41

Table 3.1: Performance comparison for different model architectures and sizes on MalNet-Image dataset.
The table lists models (ResNet18, ResNet50, ResNet101, DenseNet121, DenseNet169, MobileNetV2 (x0.5 and x1.0)) along with their parameters (in millions) and GFlops (giga floating-point operations per second). Performance metrics include F1 score, precision (P), and recall (R) for each classification task: binary, type, and family. This comprehensive comparison highlights each model's strengths and weaknesses across different classification scenarios on the MalNet-Image dataset.

ResNet101 (45M, 7.6 GFlops). This makes ResNet18 less computationally demanding, which is advantageous for training and deploying models in resource-constrained environments.

Performance:

- **Binary Classification**: ResNet18 achieves a Binary F1 score of 0.86, which is on par with deeper models like ResNet101, but with much lower computational requirements.
- **Type Classification**: While the Type F1 score of ResNet18 (0.47) is slightly lower than ResNet101 (0.48), the difference is minimal, and the computational savings justify its use.
- **Family Classification**: ResNet18 again performs competitively with a Family F1 score of 0.45, closely matching the deeper models (0.47).

Optimal Balance: ResNet18 provides an optimal balance by delivering high performance (similar F1 scores to deeper models) while requiring fewer computational resources. This makes it an excellent choice for scenarios where computational efficiency is crucial without significantly compromising accuracy.

Conclusion

Based on the performance metrics and computational requirements outlined in Tables 3.1 ResNet18 is selected as the backbone model for this study. It achieves a balanced trade-off between performance and computational efficiency, making it suitable for malware classification tasks. This backbone will be integrated with our other components to assess their impact, ensuring consistent and equitable comparisons.

3.2.2 ResNet Architecture

The core idea of ResNet is the use of residual learning. A residual block can be defined as follows:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{3.1}$$

Here, $\mathbf{x}$ represents the input to the residual block, $\mathcal{F}(\mathbf{x}, \{W_i\})$ denotes the residual mapping to be learned, and $\mathbf{y}$ is the output. The function $\mathcal{F}$ typically consists of a series of convolutional layers with weights $\{W_i\}$.

In the case where the dimensions of $\mathbf{x}$ and $\mathcal{F}(\mathbf{x}, \{W_i\})$ are not equal, a linear projection W_s (using 1x1 convolutions) is used to match dimensions:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x} \tag{3.2}$$

3.2.3 Typical ResNet Block

A typical residual block consists of the following components:

1. **Convolutional Layer:** This layer applies a convolution operation to the input, which involves sliding a filter over the input to produce a feature map. The output of this layer can be expressed as:

$$\mathbf{z} = W_{\text{conv}} * \mathbf{x} + b_{\text{conv}} \tag{3.3}$$

where W_{conv} is the filter weights, b_{conv} is the bias, and $*$ denotes the convolution operation.

2. **Batch Normalization (BN):** This layer normalizes the output of the convolutional layer to improve training stability and performance. The normalized output is given by:

$$\mathbf{z}_{\text{norm}} = \frac{\mathbf{z} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{3.4}$$

where μ and σ^2 are the mean and variance of $\mathbf{z}$, respectively, and ϵ is a small constant for numerical stability.

3. **ReLU Activation:** The Rectified Linear Unit (ReLU) introduces non-linearity to the network, allowing it to learn complex patterns. The ReLU activation is defined as:

$$\mathbf{a} = \max(0, \mathbf{z}_{\text{norm}}) \tag{3.5}$$

4. **Pooling (optional):** Pooling layers reduce the spatial dimensions of the input, which helps in reducing the number of parameters and computational complexity. Common types include max pooling and average pooling. For max pooling:

$$\mathbf{p}_{i,j} = \max(\mathbf{a}_{i+k,j+l}), \quad \text{for } k, l \in \{0, \ldots, p-1\} \tag{3.6}$$

where p is the pooling window size.

The output of a residual block thus combines the input and the transformed input:

$$\mathbf{y} = \mathbf{x} + \mathbf{a}_{\text{block}} \tag{3.7}$$

3.2.4 Network Scaling

The depth of a ResNet is defined by the number of layers it contains. Common architectures include ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152, where the number indicates the total layers in the network.

The general form of the residual learning block is:

$$\mathbf{y}_l = h(\mathbf{y}_{l-1}) + \mathcal{F}(\mathbf{y}_{l-1}, W_l) \tag{3.8}$$

where $h(\mathbf{y}_{l-1})$ is an identity mapping (or a linear projection for matching dimensions), and $\mathcal{F}(\mathbf{y}_{l-1}, W_l)$ represents the residual function (typically a stack of two or three layers).

Chapter 4

Methodology

4.1 Feature Pyramid Networks (FPN)

Feature Pyramid Networks (FPN (Lin et al., 2017)) are a type of neural network architecture designed to address the challenges posed by detecting objects at multiple scales within an image. Traditional convolutional neural networks (CNNs) (LeCun et al., 1998) often struggle with multi-scale object detection due to the fixed receptive field sizes of convolutional layers. Although it extracts features from higher level layers implicitly via pooling that sometimes lead to loss in the details of the input, FPNs mitigate this issue by explicitly constructing a pyramid of features with rich semantic information at multiple scales, enabling more effective detection of objects of varying sizes.

4.1.1 Architecture of FPN

The core idea of FPN is to leverage a backbone network to extract feature maps at different scales, and then to build a top-down pathway with lateral connections to fuse these features. The architecture can be described as follows:

1. **Bottom-up pathway**: This is typically a convolutional neural network (such as ResNet) that extracts feature maps at different levels. Let $\{C_2, C_3, C_4, C_5\}$ denote the feature maps of the backbone network, where C_i is the output of the i-th stage.

2. **Top-down pathway and lateral connections**: A top-down pathway is built by upsampling higher-level feature maps and combining them with corresponding lower-level feature maps using lateral connections. Let $\{P_2, P_3, P_4, P_5\}$ represent the resulting feature maps at different pyramid levels.

Mathematically, the feature map at level P_i can be computed as:

$$P_i = \text{Conv}(C_i) + \text{Upsample}(P_{i+1}) \tag{4.1}$$

where Conv(C_i) denotes a 1×1 convolution applied to C_i to reduce the channel dimension, and Upsample(P_{i+1}) denotes upsampling of P_{i+1} by a factor of 2.

4.1.2 Handling Different Image Sizes

To address the issue that binary files have different sizes, and when converted to standard image sizes, the same code blocks can appear at different scales, we propose using Feature Pyramid Networks (FPN) to solve this problem.

First, all images are resized to a fixed dimension (H_r, W_r) before being fed into the network. This resizing ensures that the feature maps generated by the backbone network are consistent in spatial dimensions. However, this resizing can cause code blocks of the same size to appear at different scales across different images.

FPNs mitigate this issue by providing a feature pyramid that captures rich semantic information at multiple scales, allowing the network to effectively detect objects (malicious code blocks here) regardless of the scale variation caused by resizing. Each level of the feature pyramid focuses on a specific range of object sizes, making the network robust to scale variations.

By utilizing FPNs, the network can detect code blocks at multiple scales within the resized images, ensuring that the detection performance is not adversely affected by the resizing process. This approach enables consistent detection of code blocks regardless of the initial size differences in binary files.

4.2 Optimizers

Optimizers play a crucial role in training deep learning models by updating the model's weights to minimize the loss function. They determine how the weights are adjusted based on the gradient of the loss function with respect to the weights. Several optimizers have been developed, each with its own mechanisms to improve convergence speed and accuracy. Among the widely used optimizers are Stochastic Gradient Descent (SGD), RMSprop, and Adam.

4.2.1 Stochastic Gradient Descent (SGD)

SGD (Robbins and Monro, 1951) updates the weights by moving them in the direction of the negative gradient of the loss function. The update rule for SGD is given by:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t) \tag{4.2}$$

where θ_t are the model parameters at iteration t, η is the learning rate, and $\nabla_\theta L(\theta_t)$ is the gradient of the loss function with respect to the parameters.

4.2.2 RMSprop

RMSprop (Tieleman and Hinton, 2012) adapts the learning rate for each parameter by maintaining a moving average of the squared gradients. The update rule for RMSprop is:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho) g_t^2 \tag{4.3}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{4.4}$$

where $E[g^2]_t$ is the moving average of the squared gradients, ρ is the decay rate, g_t is the gradient at iteration t, and ϵ is a small constant to prevent division by zero.

4.2.3 Adam Optimizer

The Adam (Kingma and Ba, 2014) optimizer combines the advantages of both SGD with momentum and RMSprop. It maintains an exponentially decaying average of past gradients (momentum) and past squared gradients (RMSprop-like).

The update rules for Adam are:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{4.5}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{4.6}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{4.7}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{4.8}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.9}$$

where m_t is the first moment estimate, v_t is the second moment estimate, β_1 and β_2 are the decay rates for the moment estimates, and $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected estimates.

4.2.4 Learning Rate Scheduling

Learning rate scheduling adjusts the learning rate during training to improve convergence. Common scheduling techniques include:

- **Step Decay (Krizhevsky et al., 2012)** Reduces the learning rate by a factor at fixed intervals.

$$\eta_t = \eta_0 \times \text{factor}^{\left\lfloor \frac{t}{\text{step size}} \right\rfloor} \tag{4.10}$$

- **Exponential Decay (Goodfellow et al., 2016)** Continuously reduces the learning rate by an exponential factor.

$$\eta_t = \eta_0 \times e^{-\lambda t} \tag{4.11}$$

- **1Cycle Learning Rate (Smith, 2017)** Increases the learning rate linearly up to a maximum value and then decreases it.

$$\eta_t = \begin{cases} \eta_{min} + \frac{t}{t_{max}}(\eta_{max} - \eta_{min}) & \text{if } t \leq t_{max} \\ \eta_{max} - \frac{t - t_{max}}{T - t_{max}}(\eta_{max} - \eta_{min}) & \text{if } t > t_{max} \end{cases} \tag{4.12}$$

Adjusting the learning rate can significantly affect the training process, helping to avoid local minima and ensuring better convergence. Properly scheduled learning rates can lead to faster training times and improved model performance.

4.2.5 Schedule-Free Adam

Recent advancements in optimization algorithms have led to the development of the Schedule-Free Adam optimizer (Defazio et al., 2024), which eliminates the need for learning rate schedules while maintaining or exceeding the performance of traditional schedule-based methods.

The Schedule-Free Adam method modifies the standard Adam optimizer by incorporating an alternative form of momentum and iterate averaging, which together re-

move the dependency on learning rate schedules. The key equations governing this approach are as follows:

$$y_t = (1 - \beta_1)z_t + \beta_1 x_t \tag{4.13}$$

$$z_{t+1} = z_t - \frac{\gamma_t}{\sqrt{\hat{v}_t} + \epsilon} g_t - \gamma_t \lambda y_t \tag{4.14}$$

$$x_{t+1} = (1 - c_{t+1})x_t + c_{t+1} z_{t+1} \tag{4.15}$$

where:

- y_t is a weighted combination of the parameters z_t and x_t.
- z_{t+1} is updated using the gradient g_t, learning rate γ_t, and weight decay λ.
- x_{t+1} is the averaged parameter update.

The coefficients are defined as:

$$c_{t+1} = \frac{\gamma_t^2}{\sum_{i=1}^{t} \gamma_i^2} \tag{4.16}$$

The update rule for v_t, the second moment estimate, follows the standard Adam update:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.17}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4.18}$$

Theoretical Advantages

The Schedule-Free Adam optimizer boasts several theoretical benefits:

- It achieves the worst-case optimal convergence rate for convex Lipschitz functions.
- It eliminates the need for scheduling hyperparameters, simplifying the optimization process.
- It allows the use of larger learning rates, potentially speeding up convergence.

Implementation Details

The Schedule-Free Adam algorithm can be implemented as follows (Alguliyev et al., 2024):

Algorithm 1 Schedule-Free AdamW - (Defazio et al., 2024)

Require: x_1, learning rate γ, decay λ, warmup steps T_{warmup}, β_1, β_2, ϵ

$z_1 = x_1$
$v_0 = 0$
for $t = 1$ to T **do**
  $y_t = (1-\beta_1)z_t + \beta_1 x_t$
  $g_t \in \partial f(y_t, \zeta_t)$
  $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  $\gamma_t = \gamma \min(1, t/T_{\text{warmup}})$
  $z_{t+1} = z_t - \frac{\gamma_t}{\sqrt{\hat{v}_t}+\epsilon} g_t - \gamma_t \lambda y_t$
  $c_{t+1} = \frac{\gamma_t^2}{\sum_{i=1}^{t} \gamma_i^2}$
  $x_{t+1} = (1-c_{t+1})x_t + c_{t+1} z_{t+1}$
end for

return x_{T+1}

4.3 Dataset-independent Data Augmentation

4.3.1 Mixup

Mixup is a data augmentation technique that helps to improve the generalization of image classification models by generating new training examples through linear interpolation of existing ones. This method was introduced by Zhang et al. (2018) and can be mathematically formulated as follows:

Given two examples (x_i, y_i) and (x_j, y_j), where x denotes the input image and y represents the corresponding one-hot encoded label, a new training example $(\tilde{x}, \tilde{y})$ is generated using the following convex combination:

$$\tilde{x} = \lambda x_i + (1-\lambda) x_j \tag{4.19}$$

$$\tilde{y} = \lambda y_i + (1 - \lambda) y_j \tag{4.20}$$

Here, $\lambda \in [0, 1]$ is a random variable drawn from a Beta distribution, $\lambda \sim \text{Beta}(\alpha, \alpha)$ for $\alpha > 0$.

The mixup technique is data-independent as it does not require any additional information apart from the existing dataset. The random sampling of λ ensures that the generated samples cover a wide range of interpolations between pairs of original samples. This increases the diversity of the training data and helps the model to generalize better by learning smoother decision boundaries.

Mixup effectively regularizes the model by encouraging it to behave linearly in-between training examples, which helps in reducing overfitting. The interpolated labels $\tilde{y}$ are also smoothed versions of the original labels, making the training process more robust to noisy labels and adversarial examples.

In summary, mixup data augmentation creates new training examples by linearly combining pairs of original examples and their labels. This process introduces variability and helps the model generalize better, ultimately leading to improved performance in image classification tasks.

4.3.2 TrivialAugment

TrivialAugment (Mueller and Hutter, 2021), is a data augmentation technique designed to simplify and enhance the training of deep neural networks. This method stands out for its minimalistic approach, requiring no extensive hyperparameter tuning.

TrivialAugment applies a single random augmentation operation to each image in the dataset during training. The operation is chosen from a predefined set, including rotations, translations, scaling, and color adjustments, with the strength of the augmentation randomly selected within a reasonable range. This approach eliminates the need for tuning multiple hyperparameters, which is often required in more complex strategies like AutoAugment and RandAugment.

Benefits and Performance One of the main advantages of TrivialAugment is its ease of use. It allows practitioners to achieve competitive results without extensive experimentation. The paper demonstrates that TrivialAugment can perform on par with or even outperform more sophisticated methods across various datasets and tasks, such as image classification on CIFAR-10, CIFAR-100, and ImageNet.

Implementation The implementation of TrivialAugment is straightforward:

1. **Select an Augmentation Operation**: Randomly choose an augmentation operation from the set.
2. **Determine the Strength**: Randomly select the magnitude of the augmentation.
3. **Apply the Augmentation**: Apply the selected augmentation operation with the determined strength to the image.

This process ensures that each image undergoes a unique transformation, increasing the diversity of the training data and helping the model generalize better.

4.4 Transfer Learning

Transfer learning is a machine learning technique where a model developed for a particular task is reused as the starting point for a model on a second task. It leverages the knowledge acquired from the source domain to improve learning efficiency and effectiveness in the target domain. This approach is particularly advantageous in scenarios where the target domain has limited labeled data, making it challenging to train a robust model from scratch.

In the context of malware detection using image-based deep learning models, transfer learning is employed to enhance model performance and reduce computational costs. Typically, a pre-trained convolutional neural network (CNN) model, such as VGG, ResNet, or Inception, trained on a large-scale image dataset like ImageNet, is fine-tuned on the malware dataset. The rationale behind this strategy is that even if the source dataset (natural images) is not similar to the target dataset (malware images), initializing the network with pre-trained weights is significantly better than starting with random weights. This has been demonstrated in various studies, including the work by Yosinski et al. (2014). Moreover, recent research, such as the study by Kumar and Panda (2023), provides further evidence supporting that pre-trained weights, originally trained on the ImageNet dataset, are effective for malware classification tasks by fine-tuning the VGG16 model and utilizing VGG19, ResNet50, and InceptionV3 solely as feature extractors without any fine-tuning.

4.4.1 Process of Transfer Learning

The transfer learning process involves several steps:

1. **Pre-trained Model Selection**: A suitable pre-trained model is selected based on its architecture and proven performance in similar tasks. Models like ResNet and Inception are popular choices due to their deep architecture and ability to capture complex features.

2. **Model Modification**: The pre-trained model is modified to suit the specific requirements of the malware detection task. This typically involves replacing the final classification layer with a new layer that matches the number of target classes in the malware dataset.

3. **Feature Extraction and Fine-tuning**:

   - **Feature Extraction**: The earlier layers of the pre-trained model are frozen to retain their learned weights, and only the final layers are trained on the new data. This approach reduces the risk of overfitting, especially when the target dataset is small.
   - **Fine-tuning**: In some cases, it is beneficial to unfreeze some of the deeper layers of the pre-trained model and fine-tune them alongside the new layers. This allows the model to adapt more closely to the specific features of the malware images.

4.5 Loss function and class imbalance

Cross-Entropy (Cover and Thomas, 2006) loss is a widely used loss function for classification tasks. It measures the performance of a classification model whose output is a probability value between 0 and 1. The Cross-Entropy loss increases as the predicted probability diverges from the actual label.

For a single instance, the Cross-Entropy loss is defined as follows:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{4.21}$$

where C is the number of classes, y_i is the binary indicator (0 or 1) if class label i is the correct classification for the given observation, and $\hat{y}_i$ is the predicted probability of the observation being of class i.

In the context of deep learning, the average Cross-Entropy loss over a batch of N instances is given by:

$$L = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} y_{ij}\log(\hat{y}_{ij}) \tag{4.22}$$

where y_{ij} and $\hat{y}_{ij}$ are the true label and predicted probability for the j-th instance and the i-th class, respectively.

4.5.1 Class Imbalance

Class imbalance occurs when certain classes are underrepresented compared to others. This imbalance can bias the model towards predicting the majority class. To mitigate this, class weighting can be applied to the Cross-Entropy loss function.

The weighted Cross-Entropy loss can be formulated as:

$$L = -\sum_{i=1}^{C} w_i y_i \log(\hat{y}_i) \tag{4.23}$$

where w_i is the weight associated with class i. These weights are often set to be inversely proportional to the class frequencies:

$$w_i = \frac{N}{C \cdot N_i} \tag{4.24}$$

where N is the total number of instances, C is the number of classes, and N_i is the number of instances of class i.

For a batch of N instances, the weighted Cross-Entropy loss is given by:

$$L = -\frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{C} w_i y_{ij}\log(\hat{y}_{ij}) \tag{4.25}$$

By applying class weights, the model is encouraged to treat all classes equally, thus addressing the imbalance and improving the performance on minority classes.

Chapter 5

Experiment Setup and Results

5.1 Evaluation and Metrics

In the evaluation of classification models, the choice of metrics is crucial, particularly in the presence of class imbalance. Accuracy, although a commonly used metric, can be misleading in imbalanced datasets. For instance, in a dataset where 95% of the samples belong to one class, a model that always predicts the majority class would achieve 95% accuracy, failing to capture the performance across all classes.

To address this issue, metrics such as Precision, Recall, F1-score, and Area Under the Curve (AUC) are employed, providing a more comprehensive evaluation of the model's performance.

5.1.1 Precision, Recall, and F1-score

Precision is the ratio of true positive predictions to the total predicted positives, defined as:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.1}$$

where TP is the number of true positives, and FP is the number of false positives.

Recall (or Sensitivity) is the ratio of true positive predictions to the total actual positives, defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.2}$$

where FN is the number of false negatives.

Specificity is the ratio of true negative predictions to the total actual negatives, de-

fined as:

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{5.3}$$

The **F1-score** is the harmonic mean of Precision and Recall, providing a single metric that balances both:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

5.1.2 Area Under the Receiver Operating Characteristic (AUC)

The **AUC** is a metric used to evaluate the performance of a binary classification model. It is the area under the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (Recall) against the false positive rate (1-Specificity). For multiclass classification, the AUC can be extended using the one-vs-all approach, where the ROC curve is calculated for each class against all other classes. In the one-vs-all mechanism, each class is treated as the positive class, and the rest as the negative class. This results in multiple ROC curves, and their AUC values can be averaged to obtain a single performance metric.

5.1.3 Weighting Mechanisms

To accurately evaluate the performance of models in the presence of class imbalance, different weighting mechanisms can be applied to the metrics. These mechanisms ensure that the evaluation accounts for the distribution of classes in the dataset. The three primary weighting mechanisms are:

- **Macro-Averaging**: Treats all classes equally by averaging the metric of each class.
- **Micro-Averaging**: Gives more weight to classes with more samples by considering the total true positives, false positives, and false negatives across all classes.
- **Weighted-Averaging**: Weighs the metric of each class by the number of true instances in that class.

Macro-Averaged Metrics

Following the evaluation used in MalNet, we will utilize the **Macro-Averaging** method for comparison purposes. The macro-averaged metrics for Precision, Recall, F1-score, and AUC are defined as follows:

The macro-averaged Precision is:

$$\text{Macro Precision} = \frac{1}{C}\sum_{i=1}^{C}\text{Precision}_i \tag{5.5}$$

The macro-averaged Recall is:

$$\text{Macro Recall} = \frac{1}{C}\sum_{i=1}^{C}\text{Recall}_i \tag{5.6}$$

The macro-averaged F1-score is:

$$\text{Macro F1-score} = \frac{1}{C}\sum_{i=1}^{C}\text{F1-score}_i \tag{5.7}$$

The macro-averaged AUC is:

$$\text{Macro AUC} = \frac{1}{C}\sum_{i=1}^{C}\text{AUC}_i \tag{5.8}$$

where C is the number of classes and Precision_i, Recall_i, F1-score_i, and AUC_i are the respective metrics for the i-th class.

By using macro-averaged metrics, we ensure that the performance evaluation accounts for the performance across all classes equally, thus providing a more balanced assessment in the context of class imbalance, regardless of the *class sample size*.

5.2 Experiment Setup

Our approach involves utilizing multiple components that, to the best of our knowledge, have not been previously employed or studied in malware classification, except for the use of pretrained weights from ImageNet. We will explore the impact of each component while also optimizing hyperparameters to measure their individual contributions.

Baseline Reproduction: We begin by reproducing the baseline results reported for the MalNet-Tiny dataset (Freitas et al., 2022), which achieved an F1 score of 0.65 for malware type classification. This baseline utilized a grayscale image input, a weighted cross-entropy loss function, and the AdamW optimizer.

AdamW with Free Scheduling: Our next step involves employing the new AdamW optimizer with free scheduling, aiming to accomplish two primary goals: (1) replicating the baseline results with fewer training epochs, and (2) potentially improving upon the baseline results, given the advantages of the new optimizer. We will conduct hyperparameter optimization focusing on learning rate, weight decay, warmup steps, and particularly the beta values, as the AdamW optimizer with free scheduling emphasizes that this new optimizer is more sensitive to beta values (Defazio et al., 2024).

Loss Function Evaluation: We will assess the performance of weighted versus non-weighted cross-entropy loss functions. The MalNet experiments did not definitively favor one over the other, so this evaluation aims to determine which works best with the new optimizer.

Optimization Target: While we prefer to optimize our model directly for the F1 score, as studies in the literature (Yousef, 2023) have shown that models optimized on a loss different from the targeted metric do not always guarantee the best performance for the targeted metric, even though there is a direct correlation (fig. 5.1), we acknowledge that F1 is not differentiable and thus not suitable as a direct optimization target. This study does not cover the approximation of F1 for optimization purposes.

Pretrained Weights: After identifying the optimal hyperparameters for the AdamW schedule-free optimizer and the appropriate loss function, we will evaluate the impact of using pretrained weights from ImageNet.

Data Augmentation Techniques: We will apply dataset-independent data augmentation techniques, MixUp and TrivialAugment, on both grayscale and RGB images. We will also explore the combination of these augmentation methods.

Feature Pyramid Network (FPN) Architecture: Next, we will integrate the Feature Pyramid Network (FPN) architecture and assess its impact on model performance.

Combined Approaches: Finally, we will evaluate the combination of FPN, pretrained weights, and data augmentation techniques. Given the impracticality of exhaustive experimentation due to computational constraints, we will employ a Bayesian optimization approach to navigate the different combinations, albeit manually.

By systematically, but not exhaustivilly, exploring these components, we aim to build on the baseline and achieve improved performance in malware classification using those non-used before components.

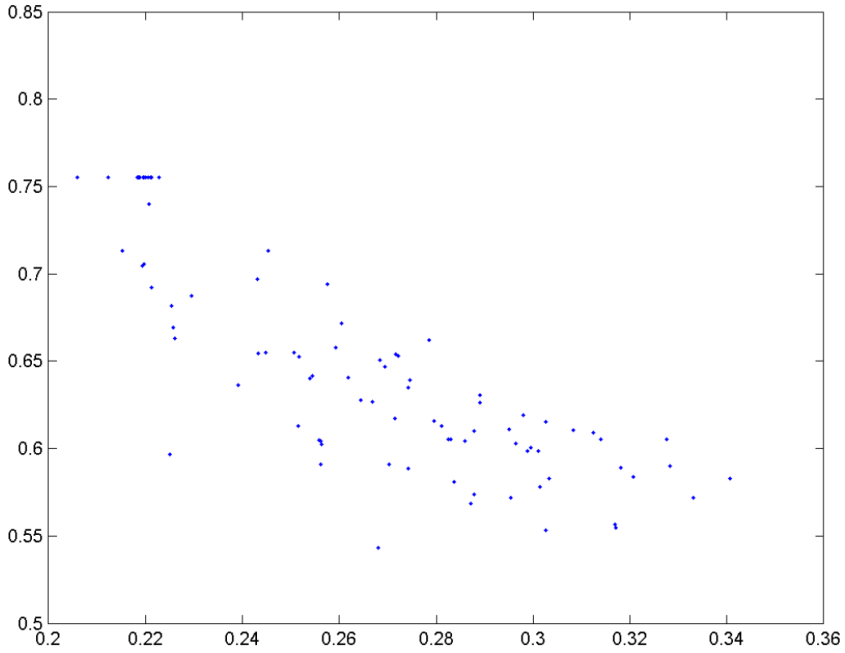

Figure 5.1: 100 pairs of true AUC vs. true MSE. - courtesy of (Yousef, 2023)
While a lower MSE often correlates with a higher AUC, this relationship is not consistently observed across all experiments

5.3 Results

This section presents the results of our 15 experiments, detailing the performance of various configurations and techniques on the MalNet-Tiny dataset. please find the results summarized in table 5.1, results are ordered by f1 score acendingly and we will use loss value to resolve ties, We will refer to different experiments using the experiment IDs mentioned in this table.

5.3.1 Baseline Reproduction

The reproduction of the baseline results using the MalNet-Tiny dataset achieved an F1 score of 0.6510 (Exp Id 5). This aligns closely with the reported baseline F1 score of 0.65,

Exp. ID	Pretrained	FPN	Precision	Recall	F1-Score	AUC	Loss	Optimizer	Loss Function	Channels	Trivial Augment	Mixup
1	×	×	0.6455	0.6266	0.6276	0.9487	1.7064	adamfree	crossentropy_weighted	1	×	×
2	✓	×	0.6978	0.5975	0.6315	0.9434	1.2115	adamW	crossentropy	3	×	×
3	×	×	0.6715	0.6083	0.6317	0.9409	1.7247	adamfree	crossentropy	1	×	×
4	×	×	0.6743	0.6320	0.6449	0.9475	1.1375	adamfree	crossentropy	1	×	×
5[a]	×	×	0.6651	0.6460	0.6510	0.9488	1.8410	adamW	crossentropy_weighted	1	×	×
6	×	×	0.6928	0.6301	0.6535	0.9451	1.5553	adamfree	crossentropy	1	×	×
7	✓	×	0.6907	0.6390	0.6536	0.9512	1.3389	adamW	crossentropy	3	×	×
8	✓	✓	0.6936	0.6499	0.6654	0.9492	1.3520	adamfree	crossentropy	3	×	×
9	✓	×	0.6951	0.6507	0.6656	0.9503	1.1690	adamfree	crossentropy	1	×	×
10	✓	×	0.7276	0.6411	0.6698	0.9519	0.8586	adamfree	crossentropy	3	×	✓
11	✓	×	0.7315	0.6424	0.6728	0.9514	0.8618	adamfree	crossentropy	1	×	✓
12	×	✓	0.7694	0.6269	0.6755	0.9521	0.9023	adamfree	crossentropy	3	✓	✓
13	✓	×	0.7098	**0.6582**	0.6764	0.9515	1.0704	adamfree	crossentropy	3	✓	×
14	✓	×	0.7670	0.6537	**0.6927**	0.9552	0.8731	adamfree	crossentropy	3	✓	✓
15	✓	✓	**0.7707**	0.6523	**0.6927**	**0.9556**	**0.8536**	adamfree	crossentropy	3	✓	✓

Table 5.1: Experiment Results - Summarizing the performance of using FPN, pretrained weights, data augmentation on RGP and grayscale, different optimizers, and loss functions.

[a]Reproducing the baseline (Freitas et al., 2022)

validating the integrity and reproducibility of the experimental setup.

5.3.2 Hyperparameter Optimization

We conducted hyperparameter optimization to identify the optimal configuration for the AdamW optimizer with free scheduling. The results of the hyperparameter optimization are summarized in table 5.2. The best-performing hyperparameters are highlighted in bold, we compared the learning rate values of 0.01 and 0.001 in Exp Id 3 and 4, and we managed to surpass the baseline when used beta 1 with a value of 0.95 in Exp Id 6 and afterwards; which align with the guidelines of how sensitive the new optimizer is to beta values (Defazio et al., 2024).

Table 5.2: Hyperparameters

Hyperparameter	Values
Learning Rate (lr)	0.01, **0.001**, **0.005**
Weight Decay (wd)	0.01
Warmup Steps (ws)	1000
Beta 1	0.9, **0.95**
Beta 2	0.999
Batch Size	128

5.3.3 AdamW with Free Scheduling

Our experiments with the AdamW optimizer with free scheduling aimed to replicate and potentially improve the baseline results with fewer epochs. Exp Id 6, which used AdamW with free scheduling and the cross-entropy loss function, achieved an F1 score of 0.6535, slightly surpassing the baseline. This indicates that the new optimizer can achieve comparable performance, but it managed to do it with 10 epochs of training, compared to the 96 epochs required by the baseline.

5.3.4 Loss Function Evaluation

We evaluated both weighted and non-weighted cross-entropy loss functions across different Exp Ids. The results from Exp Ids 1,3 and 6 showed that the cross-entropy loss function without weighting outperformed the weighted version when used with the AdamW free scheduling optimizer, but not with large margins, also the class imbalance in the tiny

dataset fig. 3.3 is not as severe as in the full dataset fig. 3.2, which could explain the lack of significant difference between the two loss functions.

Also , we reached the same observation that not using the f1 score as the optimization target does not guarantee the best performance for the targeted metric, as shown in fig. 5.2.

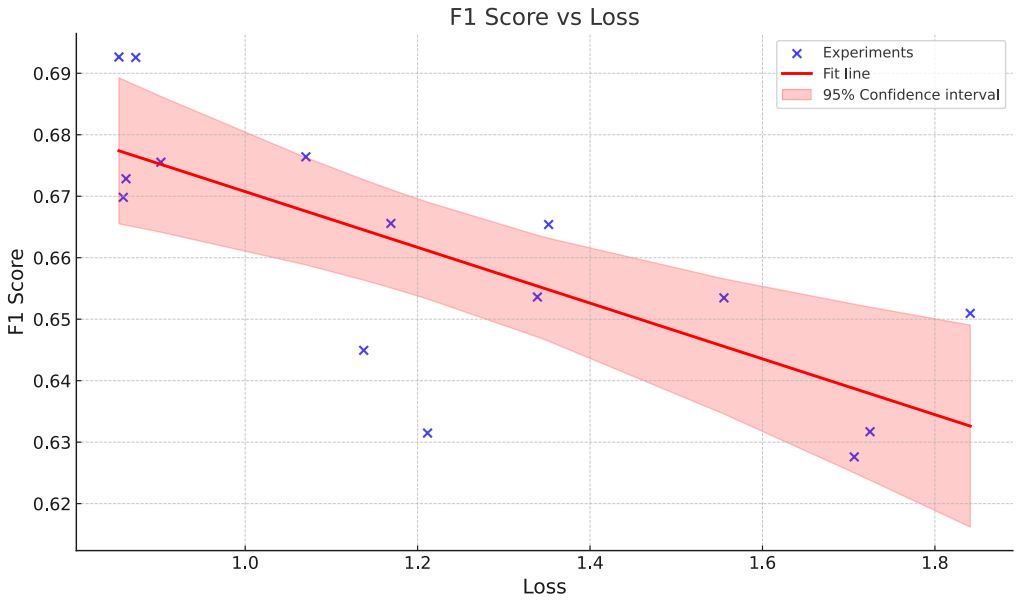

Figure 5.2: F1 Score vs Loss
getting a lower loss does not guarantee a higher F1 score, although there is an obvious correlation.

5.3.5 Optimization Target and Pretrained Weights

Experiments incorporating pretrained weights from ImageNet generally outperformed those without, check fig. 5.3. For instance, Exp Id 15 achieved the highest F1 score of 0.6927, showcasing the benefit of leveraging pretrained models.

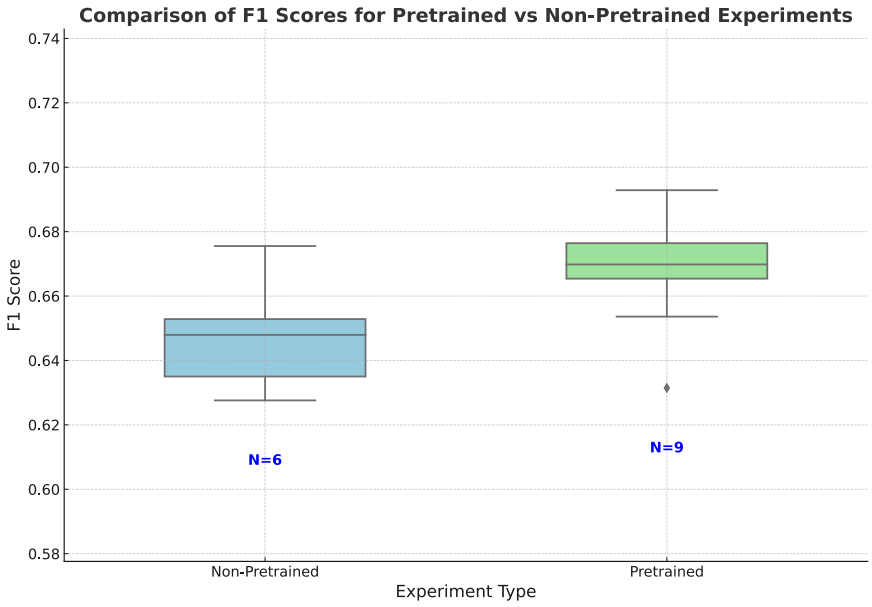

Figure 5.3: Experiments F1 Score with and without Pretrained Weights
Pretrained weights from ImageNet generally improved model performance.

5.3.6 Data Augmentation Techniques

The application of MixUp and TrivialAugment techniques showed notable performance improvements. Exp Id 14, which employed both augmentation techniques, achieved an F1 score of 0.6927, the highest among the configurations. This result underscores the effectiveness of robust data augmentation in enhancing model performance, also the number of channels used doesn't seem to have a significant impact on the performance.

5.3.7 Feature Pyramid Network (FPN) Architecture

The integration of the Feature Pyramid Network (FPN) architecture demonstrated performance gains in terms of slightly enhanced loss value, AUC and precision.

5.3.8 Combined Approaches

Our final experiments combined pretrained weights, data augmentation, and the FPN architecture. Exp Id 15, which included all these elements, achieved the highest overall performance with an F1 score of 0.6927, precision of 0.7707, AUC of 0.9556, and lowest

loss value of 0.8536. This highlights the complementary strengths of these techniques when used together. Also as we can see in fig. 5.4, it is evident that the model performs well for many categories, as indicated by the prominent diagonal line of bright cells. However, there are also several off-diagonal cells with varying intensities, highlighting instances where the model misclassified certain malware types. These misclassifications are spread throughout the matrix, suggesting that some categories may be more challenging for the model to distinguish, potentially due to similarities in features or insufficient training data for those categories.

5.3.9 Discussion

Even though none of the components yielded a significant advantage when used standalone, their combination achieved remarkable results. The best-performing model, Experiment 15, achieved an F1 score of **0.6927**, outperforming the baseline model, Exp Id. 5, which had an F1 score of **0.6510**. This demonstrates that the integration of FPN, pretraining, AdamFree optimizer, and augmentation techniques (trivial augment and mixup) can lead to improvements in performance.

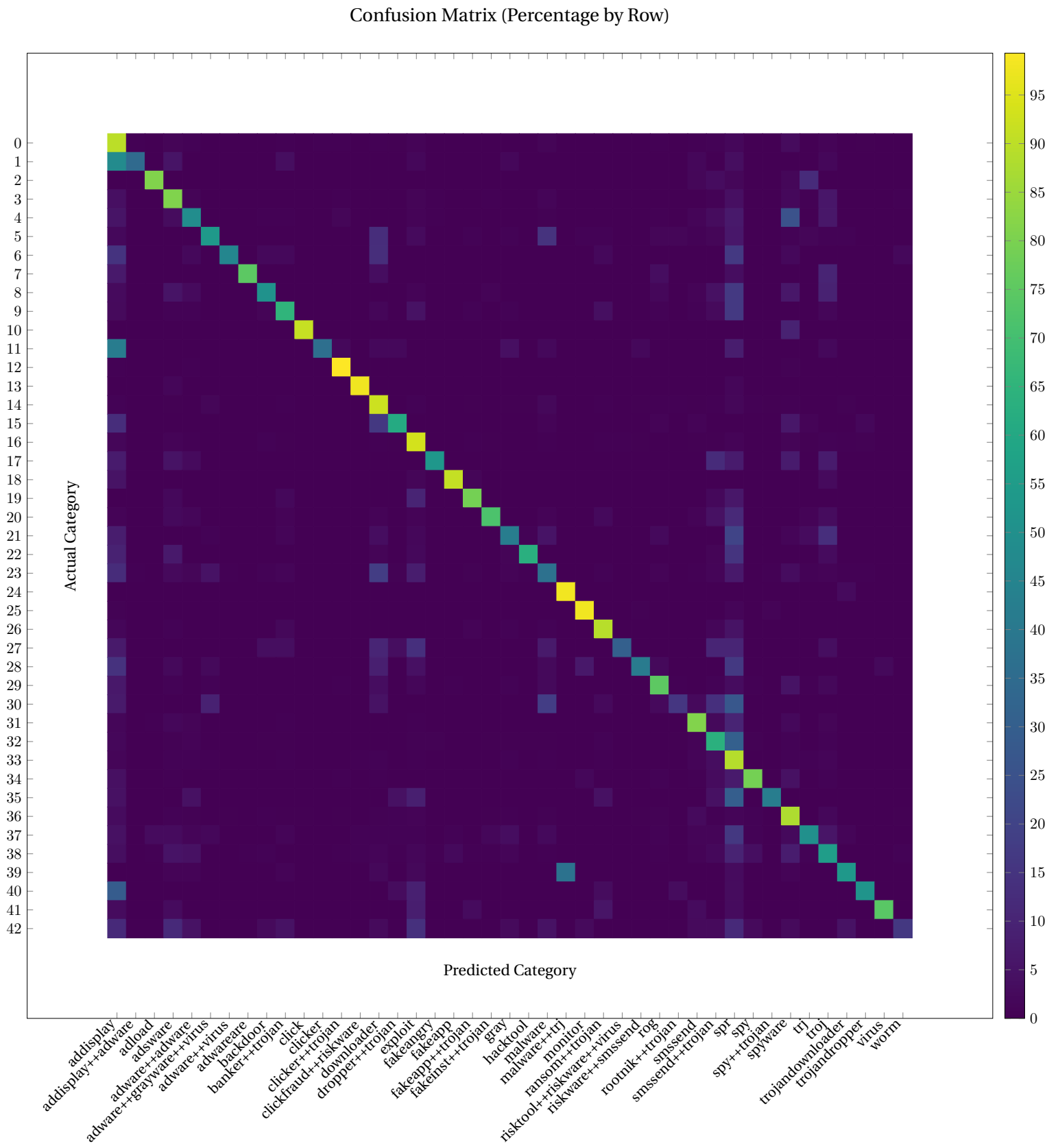

Figure 5.4: Confusion Matrix for Malware Classification: Visual representation of actual versus predicted categories across 43 malware classes, indicating the model's performance and error distribution

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This thesis presents a novel approach to malware classification using image-based deep learning models. By leveraging a subset of the MalNet-Image dataset, our study introduces several innovative components to enhance the effectiveness of malware detection. The integration of Feature Pyramid Networks (FPN), advanced data augmentation techniques such as Mixup and Trivial Augment, and the use of the schedule-free AdamW optimizer improves the classification performance.

Our experiments demonstrate that the combination of these components leads to a notable increase in F1 Score. Specifically, our best-performing model achieved an F1 score of 0.6927, outperforming the baseline models. The application of FPNs allows the model to effectively handle varying scales of malware object sizes, while data augmentation techniques improve the model's generalization capabilities. The use of the schedule-free AdamW optimizer facilitates efficient and stable training, further contributing to the model's superior performance.

Despite these advancements, several areas require further research and investigation. The techniques used in this study, such as data augmentation methods, need more tailored approaches to better capture the variability in malware images. Additionally, the process of transforming binary files to images is relatively underexplored, suggesting a potential avenue for innovative methods to improve the representation of malware in visual formats. Moreover, while the MalNet-Image dataset provides a robust foundation, its significant class imbalance presents an ongoing challenge that needs to be addressed with more sophisticated solutions. There are also discrepancies in the labels

within MalNet, indicating a need for further data cleansing to enhance the dataset's reliability.

6.2 Future Work

Building on the results of this thesis, several directions for future research are suggested:

- **Refinement of Data Augmentation Techniques**: There is a need for further investigation into advanced data augmentation methods. Developing new augmentation strategies or combining existing techniques more effectively could enhance the model's robustness and better capture the variability in malware images.
- **Binary to Image Transformation**: The process of converting binary files into images is relatively underexplored. Future work could focus on devising more effective methods for this transformation, improving the representation and classification of malware.
- **Enhanced Handling of Class Imbalance**: The significant class imbalance in the MalNet-Image dataset poses an ongoing challenge. Future research should aim to develop more sophisticated methods to address this issue, such as using generative models for creating synthetic data for underrepresented classes.
- **Label Discrepancies and Data Cleansing**: The presence of discrepancies in the labels within the MalNet-Image dataset suggests a need for thorough data cleansing. Ensuring accurate and consistent labeling is crucial for developing reliable classification models.
- **Exploration of Different Neural Network Architectures**: While Feature Pyramid Networks with CNN backbone have shown promising result, exploring other architectures, such as Transformer-based models, could introduce a breakthrough.

In conclusion, this thesis lays a strong foundation for future research in image-based malware classification, demonstrating the potential of advanced deep learning techniques in enhancing cybersecurity. Continued innovation and collaboration will be essential in addressing the evolving challenges posed by malicious software.

Bibliography

Alasmary, H., A. Abusnaina, R. Jang, M. Abuhamad, A. Anwar, D. Nyang, and D. Mohaisen (2020). Soteria: Detecting adversarial examples in control flow graph-based malware classifiers. In Proceedings of the IEEE 40th International Conference on Distributed Computing Systems (ICDCS), pp. 888–898.

Alguliyev, R., R. Aliguliyev, and L. Sukhostat (2024, March). Radon transform based malware classification in cyber-physical system using deep learning. Results in Control and Optimization 14, 100382.

Bahdanau, D., K. Cho, and Y. Bengio (2015). Neural machine translation by jointly learning to align and translate. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Bozkir, A. S., E. Tahillioglu, M. Aydos, and I. Kara (2021). Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. Computers & Security 103, 102166.

Chen, L., R. Sahita, J. Parikh, and M. Marino (2020). Stamina: Scalable deep learning approach for malware classification. Technical report, Intel and Microsoft.

Choi, S., S. Jang, Y. Kim, and J. Kim (2017). Malware detection using malware image and deep learning. In International Conference on Information and Communication Technology Convergence (ICTC), pp. 1193–1195.

Clark, J. H., D. Garrette, I. Turc, and J. Wieting (2022). Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. Transactions of the Association for Computational Linguistics 10, 73–91.

Cover, T. M. and J. A. Thomas (2006). Elements of Information Theory. Wiley-Interscience.

Defazio, A., Xingyu, Yang, H. Mehta, K. Mishchenko, A. Khaled, and A. Cutkosky (2024, May). The Road Less Scheduled. arXiv:2405.15682 [cs, math, stat].

Demirknran, F., A. Cayhr, U. Ünal, and H. Dağ (2022, October). An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. Computers & Security 121, 102846.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019, May). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs].

Freitas, S., R. Duggal, and D. H. Chau (2022, September). MalNet: A Large-Scale Image Database of Malicious Software. arXiv:2102.01072 [cs].

Fu, J., J. Xue, Y. Wang, Z. Liu, and C. Shan (2018). Malware visualization for fine-grained classification. IEEE Access 6, 14510–14523.

Gao, H., S. Cheng, and W. Zhang (2021, July). GDroid: Android malware detection and classification with graph convolutional network. Computers & Security 106, 102264.

Goodfellow, I., Y. Bengio, and A. Courville (2016). Deep Learning. MIT Press.

Graves, A. and J. Schmidhuber (2005). Framewise phoneme classification with bidirectional lstm networks. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., Volume 4, pp. 2047–2052 vol. 4. IEEE.

Grover, A. and J. Leskovec (2016). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864. ACM.

Han, K. S., J. H. Lim, B. Kang, and E. G. Im (2015). Malware analysis using visualized images and entropy graphs. International Journal of Information Security 14(1), 1–14.

He, K., X. Zhang, S. Ren, and J. Sun (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778.

Howard, A. G., M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger (2017). Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4700–4708.

Jiang, H., T. Turki, and J. T. L. Wang (2018, December). DLGraph: Malware Detection Using Deep Learning and Graph Embedding. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1029–1033.

Joulin, A., E. Grave, P. Bojanowski, and T. Mikolov (2016). Bag of tricks for efficient text classification.

Kancherla, K. and S. Mukkamala (2013). Image visualization based malware detection. In IEEE Symposium on Computational Intelligence in Cyber Security (CICS), pp. 40–44.

Khan, A., M. Usama, B. B. Kamal, A. Ahmad, H. Malik, and S. Lee (2024). Androdex: Android dex images of obfuscated malware. Scientific Data.

Ki, Y., E. Kim, and H. K. Kim (2015). A novel approach to detect malware based on api call sequence analysis. International Journal of Distributed Sensor Networks 2015, 9 pages.

Kingma, D. P. and J. Ba (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kipf, T. N. and M. Welling (2017). Semi-supervised classification with graph convolutional networks. In Proceedings of the 5th International Conference on Learning Representations (ICLR).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NIPS), pp. 1097–1105.

Kumar, S. and K. Panda (2023, October). SDIF-CNN: Stacking deep image features using fine-tuned convolution neural network models for real-world malware detection and classification. Applied Soft Computing 146, 110676.

Lab, I. S. (2020). Iot ddos detection dataset.

Labs, M. (2020). Mcafee dataset for malware detection.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. In Proceedings of the IEEE, Volume 86, pp. 2278–2324. IEEE.

Li, S., W. Li, C. Cook, C. Zhu, and Y. Gao (2018). Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5457–5466.

Lin, T.-Y., P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2117–2125.

Microsoft (2015). Microsoft malware classification challenge (big 2015). https://www.kaggle.com/c/malware-classification. Accessed: 2024-07-01.

Mueller, R. and F. Hutter (2021). Trivialaugment: Tuning-free yet state-of-the-art data augmentation. arXiv preprint arXiv:2103.10158.

Nataraj, L., S. Karthikeyan, G. Jacob, and B. S. Manjunath (2011a, July). Malware images: visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, VizSec '11, New York, NY, USA, pp. 1–7. Association for Computing Machinery.

Nataraj, L., S. Karthikeyan, G. Jacob, and B. S. Manjunath (2011b). Malware images: visualization and automatic classification. International Symposium on Visualization for Cyber Security.

Noever, D. and S. E. M. Noever (2021). Virus-mnist: A benchmark malware dataset. arXiv preprint arXiv:2103.00602.

Pachhala, N., S. Jothilakshmi, and B. P. Battula (2021, October). A Comprehensive Survey on Identification of Malware Types and Malware Classification Using Machine Learning Techniques. In 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), pp. 1207–1214.

Pei, X., L. Yu, and S. Tian (2020, June). AMalNet: A deep learning framework based on graph convolutional networks for malware detection. Computers & Security 93, 101792.

Radon, J. (1917). On the determination of functions from their integral values along certain manifolds. IEEE transactions on medical imaging 5(4), 170–176.

Robbins, H. and S. Monro (1951). A Stochastic Approximation Method, Volume 22.

Sandler, M., A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510–4520.

Sibi Chakkaravarthy, S., D. Sangeetha, and V. Vaidehi (2019, May). A Survey on malware analysis and mitigation techniques. Computer Science Review 32, 1–23.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. In Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV), pp. 464–472.

Tekerek, A. and M. M. Yapici (2022, January). A novel malware classification and augmentation model based on convolutional neural network. Computers & Security 112, 102515.

Tieleman, T. and G. Hinton (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Coursera: Neural Networks for Machine Learning.

Vinayakumar, R., M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman (2019). Robust Intelligent Malware Detection Using Deep Learning. IEEE Access 7, 46717–46738.

Yesir, S. and I. Sogukpinar (2021, June). Malware Detection and Classification Using fastText and BERT. In 2021 9th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–6.

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014). How transferable are features in deep neural networks? Advances in neural information processing systems 27, 3320–3328.

Yousef, W. A. (2023). Machine learning construction: Implications to cybersecurity. In I. Traore, I. Woungang, and S. Saad (Eds.), Artificial Intelligence for Cyber-Physical Systems Hardening, pp. 45–80. Cham: Springer International Publishing.

Zhang, H., M. Cisse, Y. N. Dauphin, and D. Lopez-Paz (2018, April). mixup: Beyond Empirical Risk Minimization. arXiv:1710.09412 [cs, stat].

Zhang, N., J. Xue, Y. Ma, R. Zhang, T. Liang, and Y.-a. Tan (2021). Hybrid sequence-based Android malware detection using natural language processing. International Journal of Intelligent Systems 36(10), 5770–5784.

Zhu, J.-Y., T. Park, P. Isola, and A. A. Efros (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2223–2232.