FEW-SHOT LEARNING FOR MALWARE DETECTION

A THESIS PRESENTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY
IN
COMPUTER SCIENCE
AT MASSEY UNIVERSITY, AUCKLAND,
NEW ZEALAND.

Jinting Zhu

2024

Contents

List of Tables

List of Figures

Abstract

The amount of malware is growing as the electric equipment thrives, which is exacerbated by malware's diversity and uniqueness. Under such circumstances, efficient detection by artificial intelligence (AI) has recently emerged and inspired researchers to pay attention to this field. At present, the designed AI model based on large-scale training can effectively detect known types of malicious attacks. However, malware differs from the natural images that other AI models access. In particular, zero-day attacks are scarce in number and frequently updated, and they generally are packaged with obfuscation techniques to avoid being detected. This thesis demonstrates some novel approaches from advanced artificial intelligence technology to overcome these challenges.

Our first research investigates a few-shot learning model applied in malware detection with scarce data and utilizes Siamese Neural Network (SNN) based on the metric space to detect malware. Our model addresses the optimization problem that tends to overfit in the few-shot training phase, in which feature embedding space is optimized with the objection function of binary cross-entropy loss to improve detection accuracy. We then explored the specificity between malware in the presence of obfuscation techniques affecting the malware signature and proposed a novel Task-Aware Meta-Learning-based Siamese Neural Network that generates task-specific weights based on the entropy value. With the weights that contribute to the different classes, this model efficiently captures the unique signatures of different malware families.

Along with initial success in few-shot learning for malware detection, we take into account the characteristics of malicious signatures in entropy patterns. We first proposed a model that utilizes the entropy feature directly obtained from binary ransomware files to retain more fine-grained features associated with different ransomware signatures. Benefiting from the robust features, a pre-trained network (e.g., VGG-16) combined with SNN, boosts feature representation along the frequency of malware signature and achieved a competitive outcome compared with the traditional deep learning method applied in malware detection. Next, we propose a triage approach using a Task Memory based on the Meta-Transfer Learning framework, which quantifies the malware threat level in the few-shot learning mechanism to prioritize different classes, which can also alert some suspicious software to human decision-making methods. Finally, we propose a novel Siamese Neural Network (SNN) designed to replace the distance scores but use relation-aware embeddings which can output better similarity probabilities based on semantics across different malware samples. Along with the use of entropy images as inputs, our proposed model can obtain better structural information and subtle differences in malware signatures despite the noises introduced by different obfuscation techniques.

I would like to dedicate this thesis to my wife and my daughter for their endless love, motivation, and encouragement.

Acknowledgements

Completing the Ph.D. study is not an easy task for anyone. Fortunately, I met my kindest supervisor, Julian Jang-Jaccard, throughout the study. To Julian, my supervisor is a remarkable woman who has compassion as a superior and leader, and she guided my Ph.D. study with her professionalism and wisdom.

And also, the cordial way our co-authors, Ian Welch, Harith Al-Sahaf, Seyit Camtepe, Paul Watters, Amardeep Singh, and Evelyn Sattlegger, work with us, who give the knowledge, comments, advice, and helpful contributions on our paper and make it published on a top journal. With the time we worked together, I learned a lot from them, which benefits me in further study. Especially thanks to their help have been indispensable to my doctoral career. Especially appreciate Ian Welch and Harith Al-Sahaf.

In addition, I would like to thank my lovely colleagues, Yuanyuan Wei, Sibghat Ullah Bazai, Hooman Alavizadeh, and Aeryn Dunmore. I still clearly remember the badminton time with them on the campus of Massey University. In particular, Aeryn's writing level and serious attitude amazed me in academic writing.

I am also grateful to my previous supervisor Xiaofeng Zhu for his valuable and constructive suggestions and his help as well as support before I moved to New Zealand and started my Ph.D. study.

Finally, the study life abroad is quite significant to me. The people I work with are across the nation and race, but what we're doing is changing the world and making it a better place. Finally, I wish all mankind can go hand in hand and move forward for a promising future. In the end, a tribute to Elon Musk and Nikola Tesla for their milestone achievements in human civilization

Publications Arising from Thesis

The core of this thesis was based on the following peer-reviewed journals and conferences.

- Zhu, Jinting, Julian Jang-Jaccard, and Paul A. Watters. ”Multi-loss Siamese neural network with batch normalization layer for malware detection.” *IEEE Access 8 (2020): 171542-171550.*

- Zhu, Jinting, Julian Jang-Jaccard, Amardeep Singh, Ian Welch, AI-Sahaf Harith, and Seyit Camtepe. ”A few-shot meta-learning based siamese neural network using entropy features for ransomware classification.” *Computers Security 117 (2022): 102691.*

- Zhu, Jinting, Julian Jang-Jaccard, Amardeep Singh, Paul A. Watters, and Seyit Camtepe. ”Task-aware meta learning-based siamese neural network for classifying obfuscated malware.” *Future Internet 15 (6), 214*

- Zhu, Jinting, Julian Jang-Jaccard, Ian Welch, Harith Al-Sahaf, and Seyit Camtepe. ”A Ransomware Triage Approach using a Task Memory based on Meta-Transfer Learning Framework.” Submitted to *IEEE transactions on information forensics and security* (Under review).

- Zhu, Jinting, Julian Jang-Jaccard, Ian Welch, Harith Al-Sahaf, and Seyit Camtepe. ”Relation-Aware Based Siamese Denoising Autoencoder for Malware Few-Shot Classification” Submitted to *Expert Systems With Applications* (Major revision).

Chapter 1

Introduction

Using artificial intelligence (AI) to detect malware has gained consensus and attention in cybersecurity. Especially today's heightened security requirements raise the demands for AI, in particular, as zero-day exploits that have not been previously encountered and vulnerability scanners fail to block them because no patches or antivirus signatures exist for the zero-day attacks. Moreover, the requirements for the highest level of cybersecurity from national security agencies and the private environment are expanding the need for the unambiguous capability of strength defense ability to various malware. With this competitive advantage of the AI's automation mechanism, it will be easier to augment human capability in government and department of defense cybersecurity roles while expanding its impact and efficiency. Specifically, integrating AI capabilities into manual and semi-manual processes can minimize errors and inconsistencies through a risk analysis system.

This emerging technology integrating cybersecurity with AI will continue to expand its impact across the industry and only grow with time. AI's cyber applications offer major advantages for the government and business leaders responsible for protecting people, systems, organizations, and communities from today's relentless cyber adversaries. Moreover, AI acts as a force multiplier for cyber professionals and spreads its functions across the cyber lifecycle including monitoring vast swaths of data to detect nuanced adversarial attacks, quantifying the risks associated with known vulnerabilities, and powering decision-making with data during threat hunts, such as a triage system. Meanwhile, it increases time savings as AI expedites the detection and response cycle time, rapidly quantifying risks and accelerating analyst decision-making with data-driven mitigation measures so that cybersecurity professionals can focus on higher-level tasks instead of time-consuming, manual actions.

Therefore, reasonably making AI integrated with cybersecurity is the core task of all problems. Recently, AI for malware detection techniques has been proposed in extant literature. These approaches introduce feature extraction and processing to performance evaluation for malware detection. The underlying ideas behind the AI strategies try to project the raw features onto the feature embedding optimized by the objective loss function, which semantic of malicious code can then be understood by the artificial intelligence paradigm. Utilizing image-based machine learning (ML) techniques in malware solutions presents multiple benefits. This strategy converts malware data into visual formats, enabling the application of sophisticated image-processing technologies. Such an approach significantly improves visual pattern recognition, facilitating the easier identification of distinct malware traits. It is notably scalable and efficient, which is essential for managing voluminous data sets. Image-based ML particularly shines in anomaly detection, adeptly identifying unusual patterns that could signify malware presence. Additionally, it leverages transfer learning, employing pre-existing models to expedite the development process. This technique simplifies the interpretation of intricate data, offers automated feature extraction, and demonstrates resilience against variations in malware. In essence, it signifies an innovative blend of cybersecurity and cutting-edge ML, introducing fresh methodologies for tackling cyber threats.

Although these approaches achieve efficient results for known malware detection, which does not reflect the situation in reality that has less sample or zero-day malware, it also does not consider the obfuscation affecting the malware signature learned from the deep learning model, and the features are still easily intercepted as well. Bridging this gap between AI and cybersecurity down to the real world, then an artificial intelligence paradigm in cybersecurity does not need access to large datasets and has the ability to rely on a few examples to defend against known and unknown malicious attacks. Therefore, a new machine learning paradigm called Few-shot Learning (FSL) for malware detection is proposed in this paper, as the FSL uses prior knowledge to reduce the hypothesis space size; it could alter the search for the best hypothesis in the given hypothesis space. With this taxonomy, the model for detecting malware samples can leverage previous experience with semantic information about malicious code.

The AI based on few-shot learning for malware detection can benefit cybersecurity at the moment and in the long term. With the advanced mechanism, FSL is the artificial intelligence technology closest to being applied practically. Therefore, we provide the research questions we aim to address throughout this thesis. Based on the above-mentioned possible problems, we provide research on these issues. Siamese Neural Networks (SNNs) have been implemented for a variety of tasks, including signature verification, face recognition, and in recent years, for detecting malware.

In the context of malware detection, SNNs are used to learn and identify the similarity between benign and malicious code. While they present innovative approaches to pattern recognition within malware analysis, there are several limitations to their existing implementations. Siamese Neural Networks (SNNs) are utilized in malware detection primarily for their ability to identify similarities between data points, making them effective in comparing new software samples with known malware. They are particularly advantageous in few-shot learning scenarios and handling imbalanced datasets common in cybersecurity. However, challenges include their complex architecture and training requirements, the necessity for carefully prepared data pairs, potential generalization issues, dependency on the quality of training data, and computational intensity. Despite these limitations, SNNs hold significant potential for enhancing malware detection through advanced similarity analysis.

1.1 Research Questions

Along with the research described before, we try to solve and overcome the problems involved in the research.

- Q1, Evaluating Few-Shot Learning for Malware Detection: This study investigates the challenges and effectiveness of using few-shot learning, especially Siamese Neural Networks, in identifying both existing and new malware attacks with limited real-life data samples. The focus is on assessing whether few-shot learning is a practical approach in real-world scenarios for malware detection.
- Q2, Combatting Obfuscation: How can we effectively mitigate the adverse effects of obfuscation techniques on the performance of semantic features, aiming to enhance the precision and reliability of our results?
- Q3, How can we effectively utilize few-shot learning models to quantify the risks using the Euclidean distance of malicious attacks and use this information to build a more predictive and resilient malware triage system?
- Q4, Exploring Malware Relationships through Semantic Embedding: When we analyze different malware samples using semantic embedding-based entity-relationship sets, is there a recognizable relationship among them, and can this identified relationship shed light on the diverse behavioural patterns exhibited by different malware?

1.2 Research Goals

The main goal of this research is to make the combination of artificial intelligence and cybersecurity into a real-world application while some factors that can interfere with the accuracy of its detection are taken into account, such as obfuscation technology.

Therefore, we define the following three sub-goals corresponding to the research questions above.

- To examine the practicality of employing Siamese Neural Network with a limited set of malware examples. This involves assessing the performance of a Siamese neural network trained with specific network components, exploring the impact of batch normalization layers, various loss functions, and determining optimal parameters for data augmentation.
- To Assess the impact of two major factors on the model's malware detection capabilities: the robustness of the features and the model's capacity to distinguish significant features amidst noise. To do this, we will analyze grayscale images and entropy patterns derived from the static features of binary malware independently. Additionally, we will investigate the integration of few-shot learning, autoencoders, and weighted values on malware classes to enhance the model's resistance to noise.
- To establish a balance between artificial intelligence and semi-automation to accurately assess the risks associated with known vulnerabilities. This is crucial for utilizing data to make informed decisions during threat hunting activities. A pivotal aspect of this process is employing classification systems that can identify malicious content based solely on its semantic features.
- To explore the semantic relationships between malicious functions using a few-shot learning-based artificial intelligence model. The goal is to identify a reliable mechanism capable of handling obfuscated malware samples, contributing to the development of robust security analyses and mitigation strategies.

1.3 Thesis Organization

As Fig.1.1 shows the organization of this is as follows, the chapters in the flow-chart demonstrate a cohesive progression in the realm of malware detection using Siamese Neural Networks. Chapter 2 lays the groundwork with a multi-loss Siamese Neural Network for basic malware detection. Building upon this, Chapter 3 introduces task-aware meta-learning to tackle the more complex challenge of classifying obfuscated malware. The focus then shifts in Chapter 4 to few-shot meta-learning, emphasizing ransomware classification using entropy features, catering to scenarios with limited data. Chapter 5 advances the narrative by exploring a malware triage system based on a task memory and meta-transfer learning framework, enhancing threat prioritization and categorization. The culmination is seen in Chapter 6, which presents a relation-aware Siamese

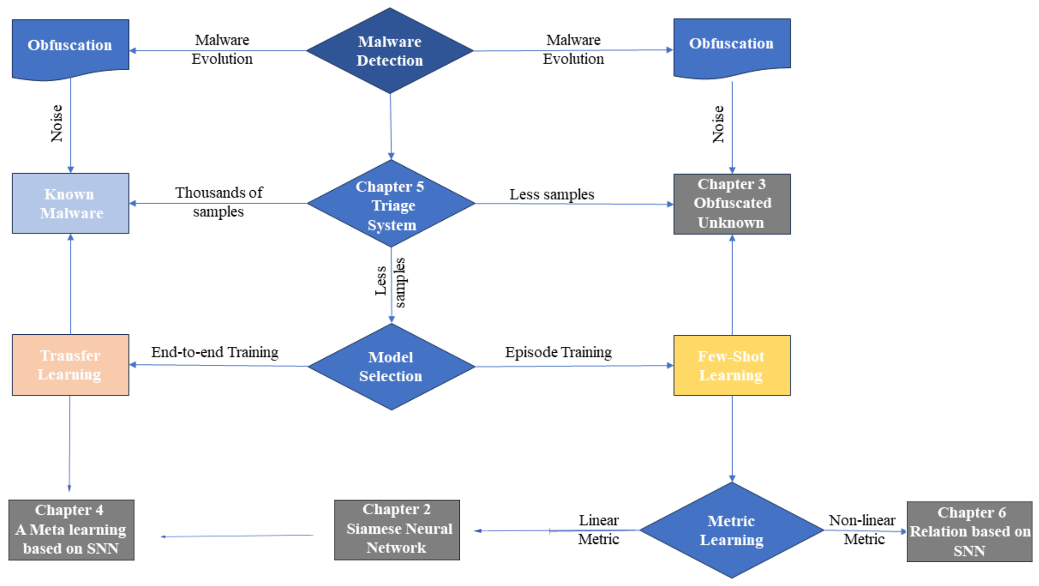


Figure 1.1: The flow-chart outlines an evolutionary path in malware detection, progressing from Siamese Neural Networks for basic detection to advanced few-shot, relation-aware models for classifying emerging malware in data-scarce environments.

Denoising Autoencoder, specifically designed for the few-shot classification of new and emerging malware. This structured progression encapsulates the evolution from basic detection techniques to advanced, specialized models for efficient and effective malware identification and classification.

In Chapter 2, we propose a Multi-Loss Siamese Neural Network with a Batch Normalization Layer that can work with fewer samples while providing high detection accuracy. Our model utilizes the Siamese Neural Network to detect new variants of malware that are trained with only a few samples. This model is equipped with batch normalization and multiple loss functions to address the overfitting issue, due to the use of small samples, which can create the vanishing gradient problem as a result of binary cross-entropy loss, and feature embedding space to improve the detection accuracy. In addition, we illustrate a way to convert raw binary files into malware grayscale images, to work with the popular Siamese Neural Network by generating the positive and negative pairs for training.

In Chapter 3, we demonstrate a Task-Aware Meta Learning-based Siamese Neural Network resilient against the presence of malware variants affected by such control flow obfuscation techniques. Using the average entropy features of each malware family as inputs in addition to the image features, our model generates the parameters for the feature layers to more accurately adjust the feature embedding for different malware families each of which has obfuscated malware variants. Our experimental results, validated with N-way on N-shot learning, show that our model is highly effective in classification accuracy compared to other similar methods.

In Chapter 4, we propose an approach Siamese Neural Network constructed with a pre-trained network (e.g. VGG-16), which is only trained with positive samples that boost its own latent. In addition, an entropy pattern is directly obtained from binary ransomware files to retain more fine-grained features associated with different ransomware signatures. These entropy features are used further to train and optimize our model. Our experimental results show that our proposed model is highly effective in providing a weighted F1-score exceeding the rate $>$ 86%.

In Chapter 5, we propose a ransomware triage approach that can rapidly classify and prioritize different ransomware classes. Our Siamese Neural Network (SNN) based approach utilizes a pre-trained ResNet18 network in a meta-learning fashion to reduce the biases in weight and parameter calculations typically associated with a machine learning model trained with a limited number of training samples. In addition, we offer a new triage strategy based on the normalized and regularized weight ratios that evaluate the level of similarity matching across ransomware, classes to identify any risky and unknown ransomware (e.g., zero-day attacks) so that a rapid further analysis can be conducted

In Chapter 6, Malicious samples are sometimes unknown (zero-day attack), as well as updated frequently, leaving the current few- shot learning methods struggling to deal with the situation caused by this feature change, as it could mine a the true relationship between the malware samples is necessary for building a model in a real scenario while avoiding the effects of obfuscation techniques. We developed a robust model that could mine a the true relationship between the malware samples is necessary for building a model in a real scenario while avoiding the effects of obfuscation techniques, such as no operation obfuscation.

In Chapter 7, we conclude this thesis and introduce some future research directions.

Chapter 2

Multi-loss Siamese Neural Network with Batch Normalization Layer for Malware Detection

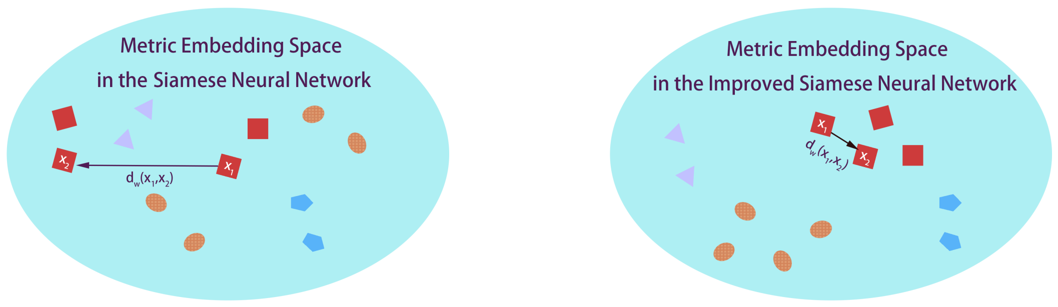
2.1 Summary

Malware detection is an essential task in cyber security. With the tremendous growth of malicious attacks in recent years, accurately detecting malware that is not previously seen before becomes more and more challenging. The current deep learning-based approaches for malware detection typically involve supplying huge amounts of samples (e.g, thousands and millions) that are often labelled based on the existing malware families in the training set thus their capability to detect new unseen malware (such as a zero-day attack) is limited. To address this issue we propose a new one-shot model called "Multi-Loss Siamese Neural Network with Batch Normalization Layer" that can work well with fewer samples while providing high detection accuracy. Our model utilizes the Siamese Neural Network to detect new variants of malware that are trained with a few samples. Our model is equipped with batch normalization and multiple loss functions to address the overfitting issue due to the use of small samples that can create a vanishing gradient problem as a result of binary cross-entropy loss in the embedding space to improve the detection accuracy. In addition, we illustrate a way to convert raw binary files into malware grayscale images that can work on the Siamese Neural Network by generating positive and negative pairs for training. Our experimental results show that our model outperforms existing similar methods.

2.2 Introduction

Traditional machine learning algorithms allow for the identification of malware that is potentially harmful. However, employing traditional machine learning algorithms requires training the machine learning model with thousands and millions of samples that are often labelled with existing malware samples [31]. In a real-life scenario, it is often difficult to obtain that kind of a large number of malware samples with proper labels, especially if they are new variants of malware and have not been seen before (e.g., zero-day attack).

Malware features are generally extracted by using two different types of analysis techniques: static and dynamic respectively. Signature-based static features are extracted by disassembling the code and analyzing the execution trace to identify any malicious patterns. On the other hand, dynamic features are extracted from executing the code in a virtual environment and by generating a behavioural report based on the execution trace. The behavioural report is analyzed to capture the behaviour of malicious code.



(a) Feature embedding space

(b) Improved embedding space

Figure 2.1: Improving the similarity distance in an embedding space

Dynamic analysis is typically regarded to have more advantages than static analysis because it provides better information for detecting malware. This includes dynamic code loading and system calls. Despite the advantages, executing the malicious code in a virtual environment comes with several challenges. For example, it may not trigger the conditions that are critical in detecting malware in a real environment. Moreover, it generally demands more significant time, resources and professional knowledge. For example, the dynamic analysis may need to activate the C&C server [169]. Hence, the requirement for a simpler way to extract dynamic features has been raised.

The analysis of raw byte sequence has been regarded to be most promising[79; 116;

78] thus a new method based on the raw executable technique known as ”malware visualization” has been proposed. Nataraj et al. [105] utilizes a technique to visualize malware binaries resulted in the dynamic analysis as grey-scale images then classify them based on similar image layout and texture. Though promising, another issue involved with effective malware detection is with the size of malware samples especially if the deep learning-based technique is utilized. In the majority of cases, a deep learning model typically requires tens of thousands (or even millions) of data points to train to achieve accurate classification, such as seen in Malconv[116]. Unfortunately, it is often difficult to find such a large number of samples for training. To address this issue, various data augmentation techniques has been proposed to increase the number of samples as well as diversity (e.g., different malware families)[157].

Extending further from the data argumentation, a new type of neural network model called N-shot learning (also called one-shot or few-shot) has been proposed. N-shot models can make full use of prior information to improve performance even if they are operated on more constrained input samples. As the name implies, N-shot learning aims at building a task from one or very few training examples. These latest techniques have been applied in malware detection with some degree of success. Li et al.[87] proposed a methodology for detecting malware with a binary classifier. Though claimed to produce better accuracy with a smaller sample, their proposal does not take account into the details of malware classes and the intra-class features. Kalash et al.[69] and Pascanu et al. [112] proposed a network for classifying malware based on malware samples only without getting benign samples involved in detection. As these existing methods do not contain the comparison among different samples involved (e.g., cross-referencing among the same or different families of samples, the distance between the pairs generated from the same family), the binary cross-entropy loss tends to show results in slow convergence and unstable performance [13]. Furthermore, these models tend to ignore the embedding space of inter-class where similar pairs may exist. By ignoring such, the similar pairs, which are supposed to share similar labels and contribute to a loss term via the same label, may end up having a large distance in the feature embedding space, as depicted in Figure 1 (a). This can cause poor performance in detection accuracy.

To address these issues associated with the existing methods, we propose a Multi-Loss Siamese Neural Network with Batch-Normalization. With added Batch-Normalization and the combination of two loss functions, hence the name multi-loss, our model provides a better similarity distance measure for similar malware pairs, as depicted in Figure 1 (b), which attributes to achieving high detection accuracy. The main contributions of our proposal are as follows:

- We introduce a strategy to convert a binary file, such as the Andro-Dumpsys dataset, into image files that can feed into an N-shot based neutral network model such as a Siamese Neural Network.

- Our proposed model is tuned such a way that it can work well on a small amount of training datasets. This is done by appropriately setting the data augmentation parameters to increase the size of the sample inputs and by adding batch normalization to avoid overfitting that could have caused due to the use of small datasets.
- In addition, the multiple loss function is used to calculate more effective similarity distance measure (i.e., reduce the similarity distance) for each positive pair that belongs to the same class. This is attributed towards achieving high detection accuracy.
- Our experimental results illustrate that our proposed model achieves higher accuracy than the baseline methods in one-shot classification tasks in malware detection.

2.3 Related Work

In this section, we discuss the state of the arts in the use of deep learning techniques for malware detection. In one of the earliest adoptions for a deep learning technique in malware analysis, Kalash et al. [69] used a Convolutional Neural Network (CNN) to detect malicious code and showed detection results on the grey images of the Malimg Dataset[105]. From there, Wang et al. [159] employed multiple CNNs to detect Android malware including the architecture for the use of the activation function based on Rectified Linear Unit (ReLU) to increase sparseness and dropout to prevent overfitting. In many cases, deep learning researchers generally use the combined convolutional and pooling layers with the full-connection layer to enhance feature extraction capability. This approach is further defined to combine CNN and deep auto-encoder to learn a more flexible model and reduce the training time. A Recurrent Neural Network (RNN) [29] is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence which allows for exhibiting temporal dynamic behaviour. Unlike other feed-forward-based neural network models, RNNs can use their internal states, kept in the memory, to process sequences of inputs. This makes it more applicable to tasks such as sentence classification [59] and speech recognition [43]. Pascanu et al. [112] employed an RNN technique to detect malware by constructing an API call language model. Their proposal used the hidden state of the model which encodes the history of previous events as the fixed-length feature vector that is given to loss function of logistic regression. Xiao et al. [165] considered that there exists some semantic information in system call sequences, much like the structure of natural language, by treating one system call sequence as a sentence in the language and then constructing a classifier based on the Long Short-Term Memory (LSTM) language model. In this model, two LSTM models are trained by the system call sequences from malware and those from benign samples respectively. According to the LSTM models, two similarity scores are computed. Finally, the classifier determines whether the application under analysis is malicious or trusted by examining the score. Tobiyama et al. [147] investigated the application of Deep Neural Networks (DNN) to classify malware processes. In their proposal, the authors trained a Recurrent Neural Network (RNN) to extract the features of malware behaviour followed by further training the Convolutional Neural Network (CNN) to classify the feature images which are generated by the extracted features from the trained RNN.

2.3.1 Siamese Neural Network for malware detection

2.4 Our model

In this section, we introduce our model with the overview, the main components, and the roles and impacts of the added layers and functions.

2.4.1 Malware Images

In our approach, we take malware detection as a visualization task by converting malware binary code into malware images followed by running a classification task using a deep learning approach. To feed into our proposed model, the malware binary codes need to be converted into an appropriate input format as seen in Figure 2.2.

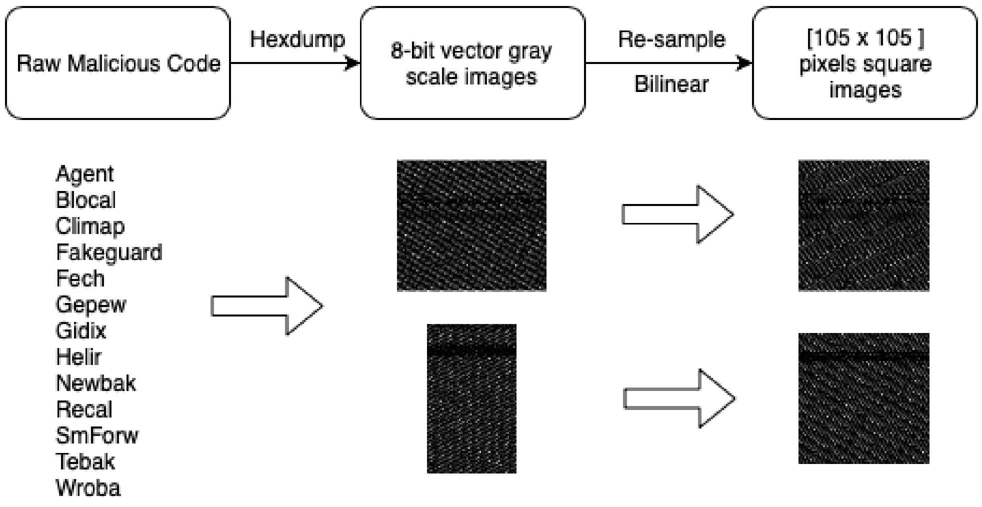


Figure 2.2: Input image conversion

Firstly, the binary malware code is read as a vector of 8 bit unsigned integers. This 1D vector of 8 bit unsigned integers is converted into a 2D vector with the fixed width for various file sizes (see Table 1) and the height corresponding to the actual size of the original file, as proposed by Nataraj et al. [105]. Further, we convert the 2D vector to form the gray images in the range [0, 255]. At this stage, the converted gray images are in various dimension (e.g., vary in size with different height and width). These

Table 2.1: Image width for various file sizes

File size	Image width	File size	Image width
<100 KB	256	4M∼8M	1280
100KB∼500KB	512	8M∼16M	1536
500 KB∼1M	768	16M∼32M	1792
2M∼4M	1024	>32M	2048

various image dimensions would result in the fully connected layer in a neural network causing errors due to inconsistency in the dimensions across different input samples. To avoid such problems, we use a bi-linear interpolation method [96] to make the images uniform to 105×105.

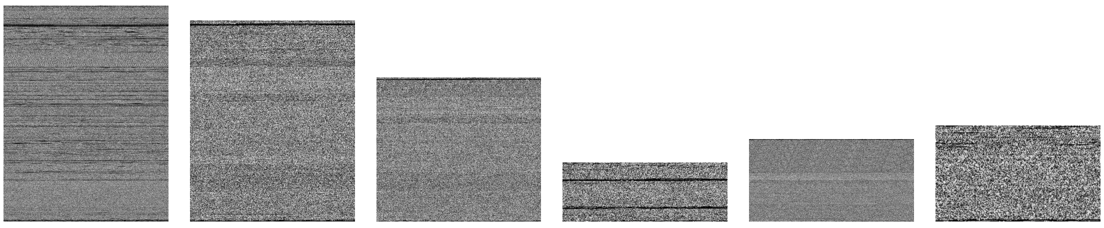

(a) fakebank.M (b) smforw.H (c) misosms.A (d) smsspy.AZ (e) fakeinst.IT (f) smforw.N

Figure 2.3: Improving the similarity distance in an embedding space

In addition, the malware images come from different malware families. The understanding of the origin of the malware family can be captured by examining the texture of malware images. For example, the texture of two malware images from the same family would be similar but slightly different in texture (as seen in Figure 2.3 (b) and (e)). The similarity in the image texture is because many variants of malware in the same family are usually developed based on the original malware and share many similar characteristics (i.e., malware signatures). Figure 2.3 illustrates the different textures of different malware families.

2.4.2 The Overview

We utilize a Siamese Neural Network (SNN) as an underlying deep learning model. Based on SNN, our proposed model is mainly composed of two main parts; the shared neural network and multi-loss structure respectively. The shared neural network is further composed of a convolutional neural network, a batch normalization layer, and a pooling layer whereas the multi-loss structure is composed of the two loss functions and fully connected layers. This general overview of our approach is shown in Figure 2.4. Our proposed model goes through three phases; malware image pre-processing, training, and testing respectively.

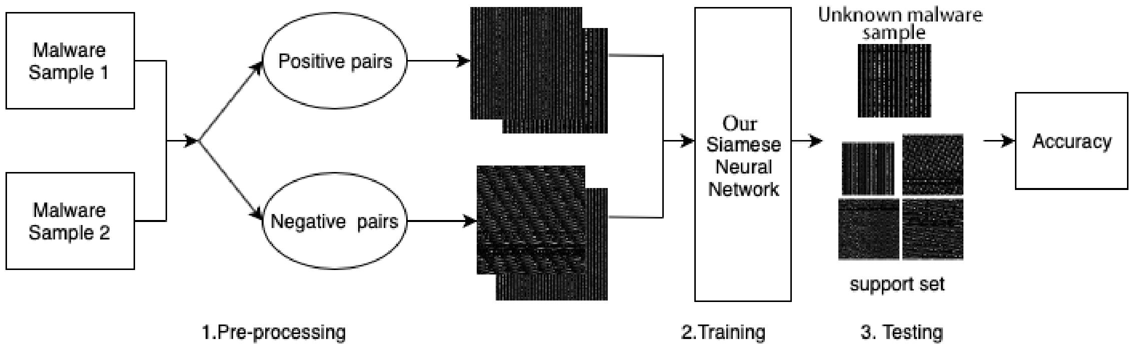


Figure 2.4: Th overview of our model

- The pre-processing phase: the input images are prepared from the binary format (of Android malware) into gray photos (as seen in Section 3.1). The images are divided into two groups: positive pairs and negative pairs and feed into two separate CNN neural networks (i.e., branches of the Siamese Neural network).

- The training phase: we create the pairs of images concatenated with the samples selected from the same or different classes. The dimension of N-image batch size is n, x, and y. Overall the batches are separated into two aspects equally and labeled as 1 or 0 respectively, where N is the batch size, n is the randomly selected sample in the categories, x is the width of the image and y is the height of the image.

To put it more formally, our approach can be written as follows. Note that we define each malware image as E_i, where the dimension is 150 $\times$ 150 pixels. Table 2.2 lists the notation and their descriptions.

Inputted matrix We use $< E_i^+, E_j^+ >$ and $< E_i^-, E_j^- >$ to denote the positive pairs and negative pairs respectively. The form of the inputted matrix E_i are concatenated as:

$$E_i = [E_i^n; E_i^x; E_i^y : E_i^z] \tag{2.1}$$

where E_i^n is the number of sample, and E_i^x is the width pixels, E_i^y is the height pixels. the E_i^z is the RGB channels for the image. For instance, if the image is RGB the E_i^z will

Table 2.2: Notations

Notation	Description
E_i^n	the number of sample
E_i^x	the width of image
E_i^y	the height of image
E_i^z	the channel of image
x_i	the feature vectors of image
y_i	the label of sample
d_w^i	the distance feature of a pairs of images E$_i^+ and E_i^-$
y_d	the label of pairs of images
F_w	the convolutional filter with parameters w
λ	the hyper-parameters

be 3. As images used in our model are in grayscale, the value of E_i^z is set to 1.

Network Structure Our model has a pair of convolutional networks which share the parameters such as the weights $W \in R^d$. The shared weights can substantially lower the degrees of parameters for the optimization to avoid overfitting as the weight are shared with other neurons within and across two convolutional networks. The pair of convolutional networks is inputted with a unit matrix which can be represented as:

$$F_w(E_i) =< F_w(E_i^+), F_w(E_i^-) > \tag{2.2}$$

where the F_w is the feature representation of the inputted matrix of E_i which is generated by the model. Generally, this model g_w: $R^n \mapsto R^d$ is parameterized by weights w;

Multi-loss Training Given a Siamese Neural Network structure g_w and the inputted matrix E_t, the aim of our model is to predict the probability of malware with the multi-loss function which jointly optimizes identification loss and detection loss. This is defined as follows:

$$L(d_w, y_d, x_i, y_i) = L_1(d_w, y_d) + \lambda L_2(x_i, y_i) \tag{2.3}$$

where λ is a hyper-parameter to weigh the relative importance of each loss.

2.4.3 Main Components

The main components of our Siamese neural network based model include: 1) the batch normalization layer is added for normalizing the output of the previous activation; 2) multi-loss structure is added for improving the feature embedding space, as seen in Figure 2.5.

Algorithm 1: Pseudo-code of our proposed algorithm

```
Pairs generator:$E = G(x_1, x_2, ..., x_n)$;
Initialization weights and biases as [77];
function Euclid $d_w(x_1, x_2)$;
Binary_crossentropy $L_1$; Softmax $L_2$;
Input  : Training set $E_i$, support set $E_s$, Target label $L_t$, Hyper-parameters $\lambda$
Output: [Predicted PairLabel]
1) Training Stage with forward and backprogation ;
while not reach to iterations do
    $F_w(x_1)$ = One branch of Siamese with batch normalization layer ;
    $F_w(x_2)$ = One branch of Siamese with batch normalization layer ;
    $d_w(x_1, x_2) = F_w(x_1) - F_w(x_2)$;
    Loss of PairLabel = $L_1(d_w)$;
    Loss of ClassLabel = $L_2((F_w(x_1))$;
    Loss of ClassLabel = $L_2((F_w(x_2))$;
    Total loss = $L_1 + \lambda L_2$
2) Testing Stage;
while not reach to iterations do
    if $E_s[index].Predict\_binarylabel = L_t[index]$ then
        return $correct + 1$
Accuracy = $100 \times correct/iterations$
```

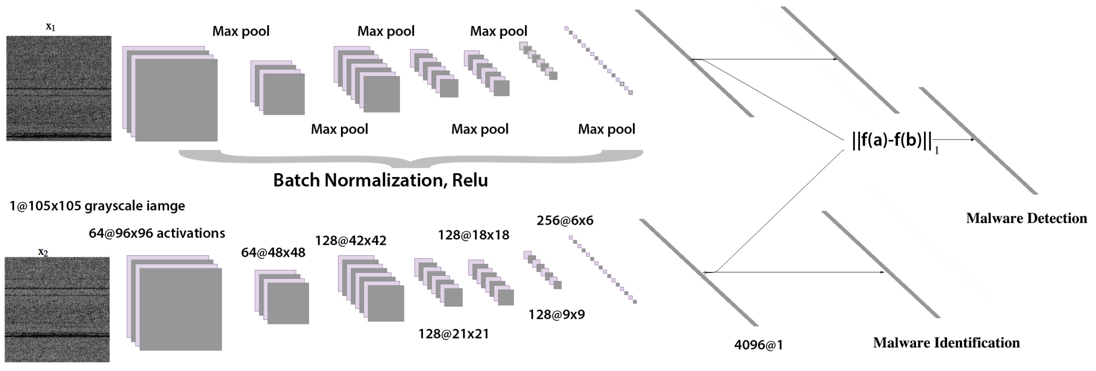


Figure 2.5: The overall network structure

In a Siamese Neural Network g_w, there are typically two symmetrical branches which share the same learned weights. A pair of images is entered as a representation of certain features into each branch. Within a branch, the pair of images goes through a series of convolutional layers, pooling layers, and fully connected layers. Generally, the top layer in each branch uses the softmax activation function to classify the pairs of the image to see whether the pair belongs to the same category.

Mathematically, the similarity between a pair of images (x_1, x_2) within Euclidean distance (ED) can be computed by a Siamese neural network (SNN) which can be described as:

$$d_w(x_1, x_2) = \|F_w(x_1) - F_w(x_2)\|_2 \tag{2.4}$$

In this equation, if x_1 and x_2 are similar, the $d_w(x_1, x_2)$ will be close to zero, otherwise they are dissimilar. However, the previous research [58] only employed the logistic loss for malware detection, as shown in Eq.(2.5), to measure the similarity between the inputted images:

$$L(w) = \sum_{i=1}^{p}(1 - y_i)f_p(d_w^i) + y_i f_q(d_w^i) \tag{2.5}$$

where y_i is the label for the input pair of images. If the images (x_1, x_2) are similar, the $y_i = 1$, otherwise, $y_i = 0$ denote that they are dissimilar.

2.4.4 Batch Normalization(BN) Layer

However, in a typical SNN, the vanishing gradient problem would occur mainly because the distribution of training data points has changed or shifted as the distribution gradually approaches the upper and lower limits of the interval of the nonlinear function value. This happens since the parameters of the preceding layers change at the training stage. Accordingly, the distribution changes at the current layer such that the current layer needs to constantly re-adjust to new distributions. This problem is especially severe for deep networks because small changes in shallower hidden layers would be amplified as they propagate within the network resulting in significant shifts in deeper hidden layers. Our model adds Batch Normalization (BN) [63] to reduce these unwanted shifts to speed up the training and to produce a more reliable model. With this additional layer, the network can use a higher learning rate without vanishing or exploding gradients. Furthermore, it also regularizes the network such that it is easier to generalize and avoids the dropout to mitigate overfitting. The network also becomes more robust to different initialization schemes and learning rates. As in [63], the BN framework is considered primarily for convolutional neural networks. Both the input and output of a BN layer are four-dimensional tensors, which are denoted as $I_{b,c,x,y}$ and $O_{b,c,x,y}$, respectively. These dimensions correspond to examples within a batch b, channel c, and two spatial dimensions x, y respectively. For input images, the channels correspond to the RGB channels. BN applies the same normalization for all activation in a given channel in such a way:

$$O_{b,c,x,y} \leftarrow \gamma_c \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c \qquad \forall b, c, x, y \tag{2.6}$$

where μ_c denotes $\frac{1}{|\omega|}\sum_{b,x,y} I_{b,c,x,y}$ from all input activations in the channel c, where ω means all activations in the channel c across all features b in the entire mini-batch and spatial x, y locations. Subsequently, BN divides the centered activation by the standard deviation μ_c (plus ϵ for numerical stability) which is calculated analogously. During testing, the running average of the mean and variances are used. Normalization is followed by a channel-wise affine transformation that is parameterized through γ_c, β_c which are learned during the training.

2.4.5 Multi-Loss Structure

In our model, we introduce two loss functions: the softmax loss (as the identification loss function) and logistic loss (as the detection loss function), respectively. The softmax loss is used for optimizing the feature embedding space which is enhanced by the logistic loss. The logistic loss is used for calculating the similarity distance between the pair of images. The combined use of two loss functions allows the sharing of information better across different tasks thus contributing to achieve better detection accuracy. In addition, the sharing of the information by these two multi-loss functions also allows our model to suffer less from overfitting when compared to other similar models that use only a single loss function.

Identification Loss Generally, the identification loss plays the same role as the general classification task. In our proposal, we employ softmax loss to enlarge the inter-class distinction to optimize the feature embedding space to force the feature representation corresponding to the data point of the same class. This can be written as follows:

$$L_1 = -\frac{1}{M}\sum_{i=1}^{M} log \frac{e^{W_{y_i}^T x_i} + b_{y_i}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}} \tag{2.7}$$

where M is the number of batch size, y_i denotes the label and x_i which denotes the deep feature of malware. $W_j \in \mathbb{R}^{d\times n}$ is the identification of the fully connected layer and $b \in \mathbb{R}^n$ is the bias term. Finally, the Eq.(2.7) is the part of two losses which are optimized jointly during the training.

Detection Loss In our model, the malware samples belonging to the same class are considered to be positive pairs and are labelled with 1. In contrast, a pair consisting of a malware sample and a benign sample, or two malware samples from two different classes (i.e., from different malware families) are considered to be negative pairs and are labelled with 0. The distance of two images in the positive pairs is expected to be closest when they are matched with each other. With this consideration, the most suitable loss function for our model is the logistic regression. Thus, the logistic regression for our model can be given as follows:

$$L_2 = -\frac{1}{M}\sum_{i=1}^{M}[y_d^i f_p(d_w^{(i)}) + (1 - y_d^i) f_q(d_w^{(i)})] \tag{2.8}$$

where y_d is the label of image pairs, d_w is the Euclidean distance (ED) between two images. It should note that the most similar image is supposed to be the closest in the feature embedding space.

Multi-Loss Function Our model is trained by softmax loss and logistic loss and finally, both of them are converged. The identification loss of a softmax function, which shares the fully connected layer with detection loss, considers the feature of the shared fully connected layer as the input. Each pair of feature vectors independently generates a loss term and its specific embedding space. It is possible that the manifold feature is changed by the behaviour of malware code because the same malware family not sharing the specific class label, is only marked by the indicator, y_d of a pair. Furthermore, the features of all branches will be fed into their final loss to be optimized. During the training stage, the combined loss function is formulated as follows:

$$L = L_1 + \lambda L_2 \tag{2.9}$$

where λ is a hyper-parameter to balance the weight of two loss functions, which is set to 0.4. From Eq.(2.9), we can derive that each loss function is responsible for performing a specific classification task, and this loss function assists the prediction of the one-vs-one classifier with its discriminative features.

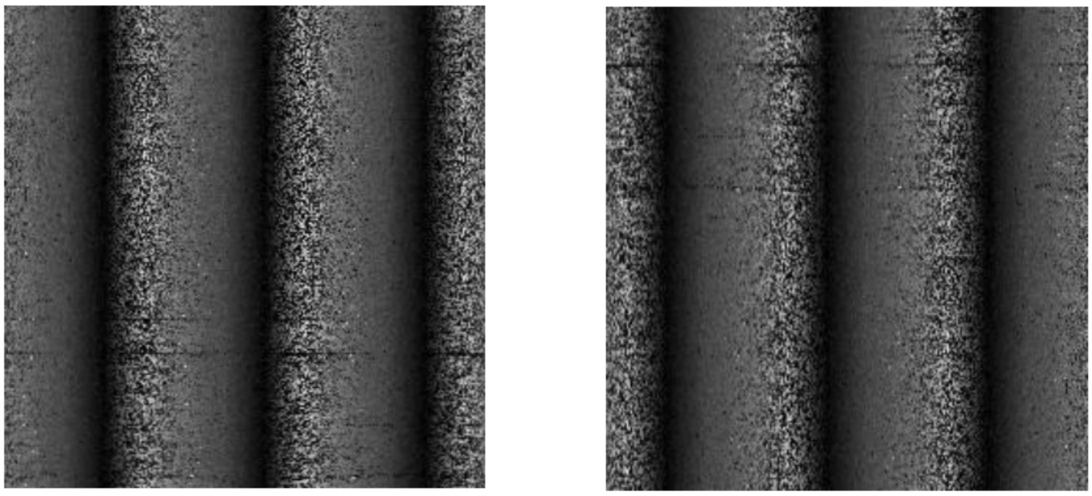
2.5 Experiments and Results

In this section, we evaluate the performance of our proposal. The experimental results have been obtained by running a set of experiments on the desktop with the 32GM RAM, Nvidia Quadro P2000(5GB), and Xeon W-2133 CPU@3.6GHz.

2.5.1 Dataset and Setup

In this experiment, we employ the dataset from our collaborator [65] that has been widely used. The dataset consists of 906 malicious binary files from 13 malware families. The number of malware samples from different malware classes varied. Almost half of the classes have no more than 25 malware samples, and some have only one as they are new malware detected lately (e.g., Blocal and Newbak). We increased the data size for every malware class to have at least 30 samples with the manual data argumentation technique [31]. A separate dataset containing 1776 benign samples was randomly used as input pairs in the support set.

Training on our model was conducted per mini-batch. Our model randomly selects image pairs as half-positive pairs and half-negative pairs. We randomly select anchor image pairs as mini-batch to feed into the model. Each mini-batch contains the image pairs that are randomly selected. The initial learning rate was set to 0.00002. The maximum number of iterations was 15000, and the batch size was 35. After training, we save the weights of each neuron. The details of the training parameters are shown in Table 2.3.

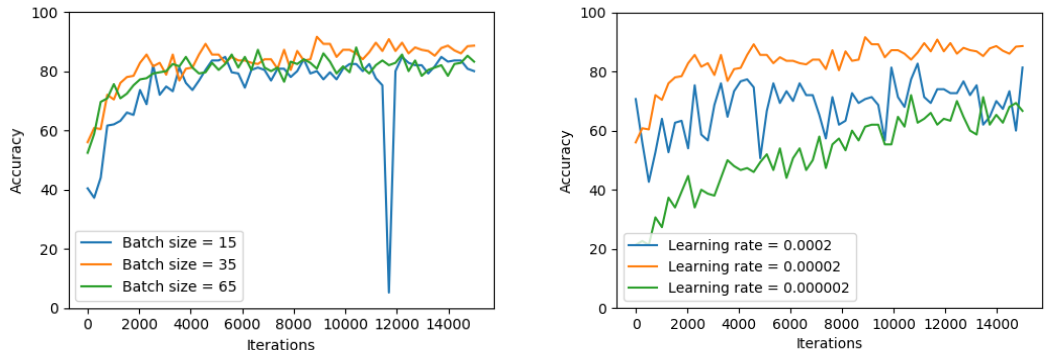

(a) Agent(1).a: the original image

(b) Agent(1).b: an augmented image

Figure 2.6: The original image vs. an augmented image

Table 2.3: Training parameters

Parameters	Values	Description
Mini-Batch	35	The number of training examples in one forward/backward pass
Learning rate	0.00002	Learning rate is used in the training of neural networks- range between 0.0 and 1.0.
N-iterations	15000	Total numbers of iterations in the training process
N-way	3 to 15	To compare a test image with N different images from different classes
N-tasks	250	The number of one-shot tasks to validate on



(a) Batch sizes and impacts (b) Learning rates and impacts

Figure 2.7: The original image vs. an augmented image

2.5.2 N-way One-shot Accuracy

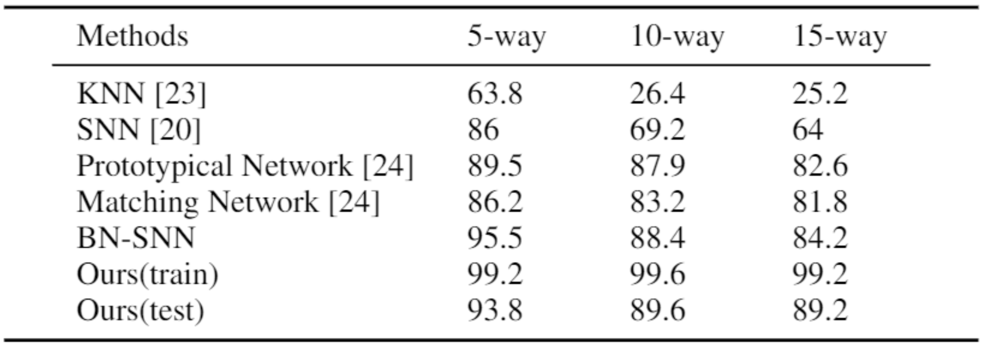
The testing was conducted m times of N-way one-shot learning tasks where Q times of the correct prediction that contributes towards accuracy was calculated by the following formula, and according to the different N-Way to divide the training set and the test set, and then the accuracy has been calculated averagely:

$$Accuracy = (100 * Q/m)\% \tag{2.10}$$

To evaluate the N-way one-shot learning at each test state, an anchor image from one class of test was chosen followed by randomly selecting N classes of images in order to create a support set $X = \{x_i\}_{i=1}^{N}$, where for x_1, $\forall x \in X$, the class of the selected images was the same as the anchor image of $\hat{x}$ where other images in the support set were from different classes. The similarity score between $\hat{x}$ and other image was calculated as follows:

- The task is labelled as a correct prediction if the similarity score of the feature vector x_1, which can be represented as $S = \{s_i\}_{i=1}^{N}$, was the maximum of S.
- Otherwise, it is regarded as an incorrect prediction.

As we can see from Figure 2.8, our proposal worked efficiently by producing the best accuracy compared to other similar models [58; 62] and similar work [150]. Our proposed model follows the procedure of the [58], which rescaled into the 105 × 105 pixels, and keeping the original aspect ratio and filling the background with black. With this setup, We produced the best detection accuracy – that is, above 99% during the training phase while around 90% during the testing phase. It must be noted that our proposed model did not have any large decline in the detection rate, as observed in other similar approaches when N-way pairs were increased. It also has shown that the mechanism regarding the image width followed by the is appropriate [105].



Methods	5-way	10-way	15-way
KNN [23]	63.8	26.4	25.2
SNN [20]	86	69.2	64
Prototypical Network [24]	89.5	87.9	82.6
Matching Network [24]	86.2	83.2	81.8
BN-SNN	95.5	88.4	84.2
Ours(train)	99.2	99.6	99.2
Ours(test)	93.8	89.6	89.2

Figure 2.8: Accuracy of N-way methods for different models

2.5.3 Distance Measure Effectiveness

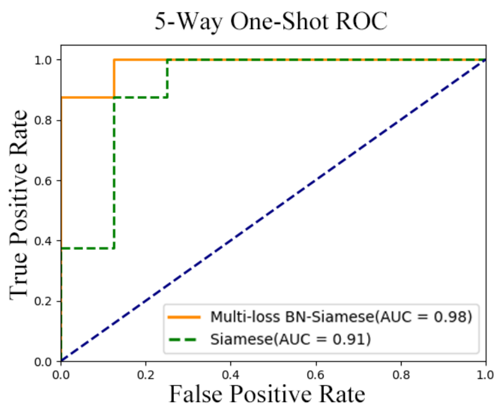
We also experimented with analyzing the effectiveness of our algorithm on distance measures using the Receiver Operating Characteristic (ROC) curve. We denote $f_0(p)$ as the probability density function of predictions $p(x)$ from our algorithm of negative pairs that are labelled as 0 and $f_1(p)$ are the probability from positive pairs that are labelled 1. The true positive rate (TPR) and false positive rate (FPR) for a given discrimination threshold p are the integrals of the tails of these distributions.

$$\begin{aligned} FPR(P^*) &:= \int_{p^*}^{1} f_0(p)dp \\ TPR(P^*) &:= \int_{p^*}^{1} f_1(p)dp \end{aligned} \tag{2.11}$$

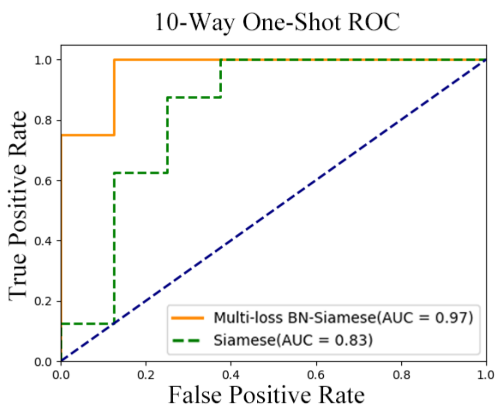
The ROC curve is the function TPR(FPR) and thus the area under the curve (AUC) is:

$$AUC = \int_0^1 TPR(FPR)D(FPR) \tag{2.12}$$

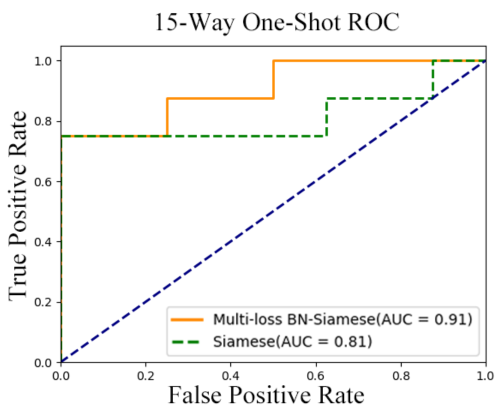
from the equation, we can see that the AUC (Area Under the Curve) is the probability that a randomly chosen point from class 0 ranks below a randomly chosen point from class 1. If the classifiers perform well, the AUC is to be closer to 1. We benchmarked our model against the popular original implementation of the Siamese Neural Network[178]. As Figure 2.9 shows, the result is on a set of points in the true positive rate - false positive rate plane, which is the curve for our data set. When the probability value of AUC is close to 0.5, it means that an algorithm randomly guessing whether a given sample is malware or benign. We respectively achieved the AUC equal to 0.98,



(a) ROC of 5-way One-shot



(b) ROC of 10-way One-shot



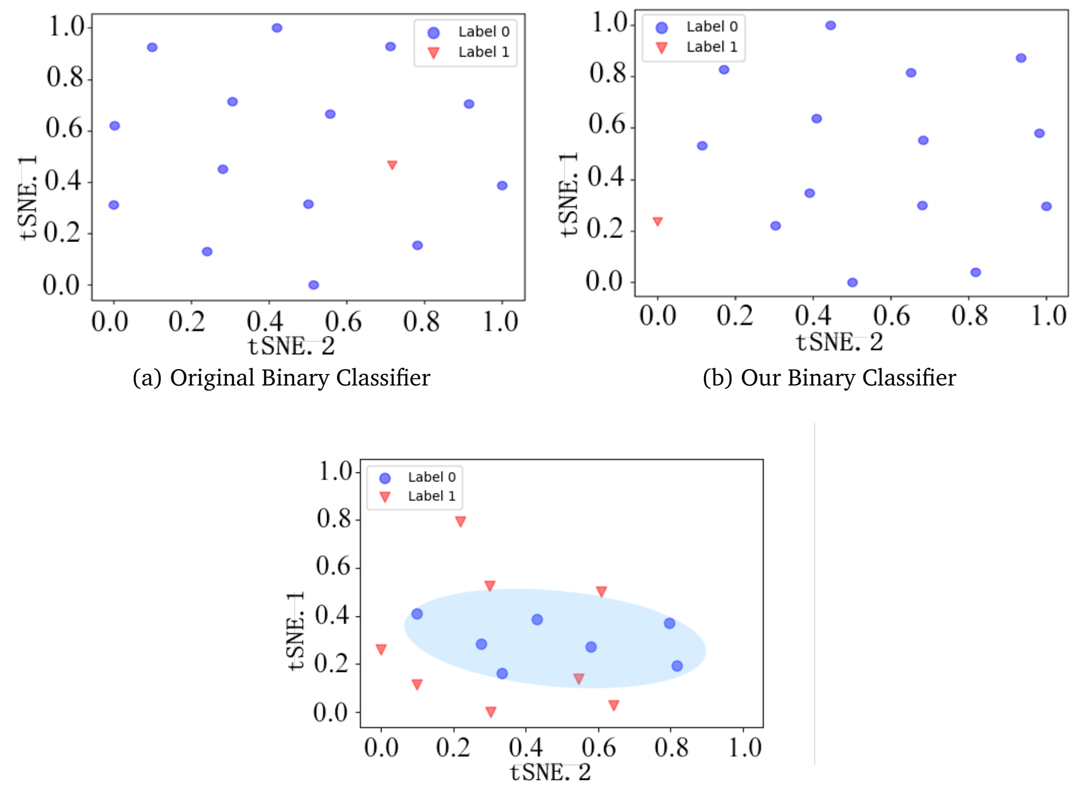
(c) ROC of 15-way One-shot

Figure 2.9: ROC curves under the N-way One-shot

0.97, and 0.91 respectively under the 5-way, 10-way, and 15-way. It is noted that our model always performs better as the AUC area of our proposal is larger against other benchmarks.

2.5.4 Visualizing Nearest Neighbors with t-SNE

Figure 2.10 displays the feature vectors of 4096 dimensions extracted from our model. With the PCA initialization, the t-distributed Stochastic Neighbor Embedding (t-SNE) technique projects these feature vectors into two dimensional for visualization. For this visualization, we only show the positions of 15 image pairs, including 14 negative pairs (blue circle point) and 1 positive pair (red triangle point). Compared with the original classifier, as shown in Figure 2.10 (a), the positive pairs are isolated by other negative pairs well in Figure 2.10 (b). We also do multiple positive pairs to visualize this model's performance. In Figure 2.10 (c), the blue area covers all seven negative pairs and only contains one wrong positive pair.



(a) Original Binary Classifier

(b) Our Binary Classifier

(c) Few-Shot Binary Classifier

Figure 2.10: Feature Visualization Using t-SNE

2.6 Conclusion and Limitations

In this paper, we propose a new neural network model based on one-shot learning for malware detection. This model effectively solves the problem that the traditional model cannot detect unknown malware, and further optimizes the feature space so that positive samples from the same class have a local distance greater than samples of different malware classes.

The experiments showed that our algorithm performed better than other baseline methods, such as Siamese Neural Network and KNN. For further study, we plan to make use of other metric learning applied to our model to improve recognition accuracy. We also have a plan to add the Spatial Pyramid Pooling layer into the SNN since it allows the convolutional network to take arbitrary sizes of images to avoid the information loss of malicious code. By adding this layer, the arbitrary size of feature maps can be adjusted via the spatial pooling regions to be appropriately proportional to the size of the output matched with the fully connected layers.

However, there are a number of limitations of our current proposed model. Our proposed model, for example, could not correctly classify a polymorphic malware (i.e., disguised as a new malware by changing or moving some parts of the code elsewhere) but would recognize it as a new malware. The validity of the synthetic data produced by the data argumentation strategy we proposed has not been fully examined to ensure whether the synthetic data captures the characteristics of real malware. In addition, our current model can be vulnerable to misclassification against adversarial attacks if the original data is modified, especially when the original malware sample is in the raw binary file format.

Chapter 3

Task-Aware Meta Learning-based Siamese Neural Network for Classifying Control Flow Obfuscated Malware

3.1 Summary

Malware authors apply different techniques of control flow obfuscation to create new malware variants to avoid detection. Existing Siamese Neural Network (SNN) based malware detection methods fail to correctly classify different malware families when such obfuscated malware samples are present in the training dataset resulting in high false-positive rates. To address this issue, we propose a novel Task-Aware Meta Learning-based Siamese Neural Network resilient against the presence of malware variants affected by such control flow obfuscation techniques. Using the average entropy features of each malware family as inputs in addition to the image features, our model generates the parameters for the feature layers to more accurately adjust the feature embedding for different malware families each of which has obfuscated malware variants. In addition, our proposed method can classify malware classes even if there are only one or a few training samples available. Our model utilizes meta-learning with the extracted features of a pre-trained network (e.g., VGG-16) to avoid the bias typically associated with a model trained with a limited number of training samples. Our proposed approach is highly effective in recognizing unique malware signatures, thus correctly classifying malware samples that belong to the same malware family even in the presence of obfuscated malware variants. Our experimental results, validated with N-way on N-shot learning, show that our model is highly effective in classification accuracy

exceeding the rate $>91\%$ compared to other similar methods.

3.2 Introduction

Neural networks are typically trained on existing data, which may not include the latest obfuscation techniques. When new obfuscation methods are developed, the neural network may not recognize them, leading to a failure to generalize and identify novel threats. Traditional neural network models may rely on handcrafted features or automatically derived features that obfuscation can easily alter. If the obfuscation changes the features in a way that the neural network cannot understand, it may not classify the malware correctly.

The malware producers are ever more motivated to create new variants of malware to gain profits from unauthorized information stealth. According to the malware detection agency AV Test[1], there are 100 million new variants of malware generated from January to October 2020 which translates as roughly three thousand new malware daily. Research is needed to develop neural network architectures and training methods that are robust against unseen obfuscation techniques. This includes exploring transfer learning, few-shot learning, or meta-learning strategies that allow models to adapt quickly to new patterns with minimal data.

Especially, we have witnessed the fast growth of mobile-based malware. The NTSC report published in 2020[2] reported that 27% of organizations globally were impacted by malware attacks sent via Android mobile devices. In recent times, we have seen malware producers employ techniques such as obfuscation [36; 28; 9] and repackaging [136; 86; 184], mostly through the change of static features [186; 140; 61] to avoid detection. Realizing the trend in the growth of mobile-based malware attacks, there have been numerous Artificial Intelligence (AI)-based defence techniques proposed [153; 92; 138; 95; 58; 132]. [139; 9; 68], which aim to exploit obfuscation-invariant features based on the static analysis. As the static analysis does not provide insight into how the malware behaves during execution. This means it can miss dynamically loaded or executed components, which are often used in sophisticated malware and static analysis can struggle to decipher these obfuscations, as it requires analyzing the code in its unexecuted form.

We argue that there are two large issues to be addressed in the existing state-of-the-art of AI-based mobile malware attack defence. The first issue is the majority of existing research tends to focus on learning from common semantic information of the generic feature of malware families and building feature embeddings [117; 47; 130].

[1] https://www.av-test.org/en/statistics/malware/
[2] https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf

Here the feature embedding represents the features contained in a malware binary sample to provide an important clue as to whether the malware image the feature embedding is created from is malicious or not. If malicious, what type of malware family it belong to assess and build the right set of response strategies. These existing works often treat the fraction of the code changed by the obfuscation and repackaging as a type of noise [48] and thus tend to ignore the effect of the modification. This is in largely because the code changed by the obfuscation and repackaging techniques show a similar appearance when malware visualization techniques are applied [2; 108; 104]. Using the common semantic information as data input points to feed into a deep neural network cannot capture the unique characteristics of each malware family signature thus they will not be able to accurately classify many variants driven from the same malware family [69; 99; 153; 173], especially if an obfuscation technique is applied.

The second issue with the existing approaches is the demand for large data inputs to find more relevant correlations across the features. They are unable to detect and classify the malware families trained with a limited number of samples (e.g., newly emerging variants of malware) [17].

To address these two important issues, we propose a novel task-aware meta learning-based Siamese Neural Network capable of detecting obfuscated malware variants belonging to the same malware family even if only a small number of training samples are available.

The contributions of our proposed model are following:

- Our proposed model can learn a unique malware signature across malware families using the meta-learning even if obfuscated malware samples exist in each malware family.
- Each CNN branch of our proposed SNN model is equipped with two sub-networks: a task-aware meta learner network and an image network, respectively. The task-aware meta learner network generates task-specific weights using the entropy graphs that capture the unique signatures of different malware families. This weight parameter is combined with the shared weights of two CNNs for the feature layers so that feature embedding at each CNN is accurately adjusted for different malware families.
- Our task-aware meta learner network combines entropy attributes with image-based features for malware detection and classification. The combined features are fed to the pre-trained VGG-16 network. By utilizing the VGG-16 network as a part of the meta-learning process, the weight generator generates the weights that accurately capture the combined features. This can avoid the potential issue of introducing bias in weight generation when the training sample size is limited.

- Our proposed model uses two different types of loss functions that can more accurately compute the similarity scores for the features within the feature embedding of each CNN (i.e., inter-class variance) and across the feature embeddings of two CNNs (i.e., intra-class variance). For the embedding loss to compute the inter-class variance, we add a secondary embedding loss alongside the binary loss to improve the bias that is introduced by a limited size of training samples. For the hybrid loss to compute the intra-class variance, the center loss is added alongside the constractive loss to enable positive pairs and negative pairs to form more distinct clusters across the pairs of images processed by two CNNs.

- Our extensive experiments with N-way on N-shot learning on the Andro-dumpy dataset show that our proposed model is highly effective to recognise the presence of a unique malware signature thus correctly classifying malware samples that belong to the same malware family despite the presence of obfuscation techniques. Our classification accuracy is at >91% exceeding the performance of similar methods.

We organize the rest of the paper as follows. We examine the related work in Section 3.3. We provide the preliminary that describes how a control flow obfuscated malware variants are created and addresses the issues as to why the generic SNN approach would not work to detect such obfuscated malware variants in Section 3.4. We provide the details of our proposed model along with the details of the main components, their roles and responsibilities, and an overview of the algorithm involved in Section 3.5. In Section 3.6, we describe the details of the dataset, feature extraction, and the experimental results with analysis. Finally, we provide a conclusion of our work including the limitation of our proposal and future work directions in Section 3.7.

3.3 Related work

In this section, we review two lines of research relevant to our study. These include few-shot learning-based malware detection and feature embedding applied for malware detection.

3.3.1 Few-shot Learning for Malware Detection

One-shot learning is a method of utilizing prior knowledge to learn a generic feature with a few image samples. This method has been widely applied in several applications for example in the field of image classification and recognition, speech recognition, and using the Siamese Neural Network and Prototypical Network. Sun et al. [141] proposed a static method to analyze the assembly language operation code through the visualization of malicious code. More recently, Hsiao et al. [58] developed an end-to-end framework based on a Siamese Neural Network to detect malware. Moustakidis et al. [102] attempted the transformation of raw data using fuzzy class memberships which then feed into the Siamese Neural Network to defend against the intrusion attack. Although these researchers have achieved competitive results utilizing different feature extraction techniques, they are limited to learning a semantic feature embedding to distribute tasks more effectively. However, the description of the semantic information of raw binary files they used was not clear and it was difficult to see how these were used for feature embedding. It appeared that a model learned for a generic feature from rare categories was unrealistic to capture the common attribute of malicious code. Tang et al. [146] proposed a high-level malware class feature with a meta-learner and evaluated that their proposal was effective at detecting the malware with distinct features. However, this set of works does not consider the distance between the positive pairs and negative pairs therefore the problem with the precise capture of the distance across intra-class variance still exists. Many existing state-of-the-arts proposed by these existing few-shot learning models tend to employ the contrastive loss which we believe does not contribute towards shrinking the intra-class variance. To resolve this issue, the positive and negative pairs must be effectively separated by the hyperplane. Based on this concept, we propose a hybrid loss function with a centre loss combined with a constractive function to improve the positive and negative pairs interval as well as the inter-class variance.

3.3.2 Feature Embedding for Malware Detection

Feature embedding techniques have been widely used in many applications, such as face and speech recognition because they can be purposely designed for a specific function and to capture the critical semantic information which may appear at any position of an image (or sentence). Based on this concept, [177; 107; 57; 113; 141; 24] adopted it into malware detection. Though it has proven that the embedded N-grams of opcodes [88] and graph embedding [177; 57; 187] can efficiently capture unique malicious components, however, conducting the opcodes and graph embedding is time-consuming and their generalizability limited (e.g., the graph embedding tends only work well in learning the static features). Using these techniques, a unique behaviour presented in a malware family that is critical to capture the variant of that malware family often tends to be regarded as noises while only common known behaviours of malware families are captured by the CNN model, for example, the work by Microsoft collaborated with Intel demonstrates the setting of the practical value of the image-based transfer learning approach for static malware classification [24]. To address this problem, Sun et al [141] proposed a model which learns the unique information specific to a malicious code. However, it did not achieve a competitive result when only a small number of training datasets was available. Tran et al. [150] evaluated the performance of malware detection for zero-day attack malware using the matching and prototypical network. The Matching Network uses the softmax over the cosine distance with two embedding functions and the memory caches the common pattern of malware feature representation. Although these works focus on the semantic feature embedding of malicious code, most of them are limited to working well on many training samples. These existing models tend to be effective at capturing malware samples when the common attribute of malicious code is distinct but often do not consider when there are slight differences in features of the malware as is the case of many obfuscated malware variants.

3.4 Preliminary

Table 3.1 illustrates the notations we use throughout this paper.

Table 3.1: Notations

Notation	Description
x_i	the malware image feature of ith samples
y_t^i	the class t of ith samples
θ_t	feature layers' parameters
W^i	i-th feature layer in $\mathcal{F}$'s weights
W_{sr}^i	the shared parameters for all malware families
W_{ts}^i	the task-specific parameters for each malware family
c_i	the center point of each class
L_e	the embedding loss
$L_{b,c}$	binary cross entropy loss with center loss
d_w^i	the distance feature of a pairs of images
y_d	the label of pairs of images
F_w	the convolutional filter with parameters w
β	the hyper-parameters

3.4.1 Control Flow Obfuscation

Malware obfuscation is a technique that is applied by malware authors to create new malware variants to avoid detection without creating a completely brand-new malware signature. Among many different obfuscation techniques, we focus on control flow obfuscation which involves creating a new malware variant by reordering the control flow of functional logic from the original malware program [36]. This type of obfuscation technique makes the complied malicious code appear to be different from the existing malware signature and thus can easily avoid detection. There are three different types of control flow obfuscation our model can detect.

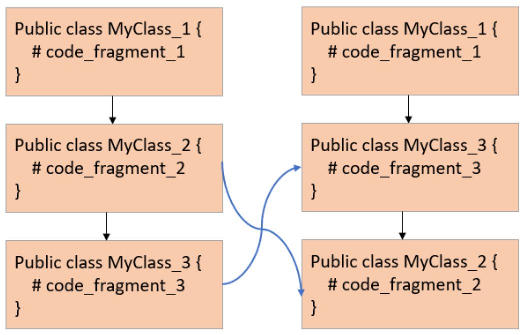


Figure 3.1: Function Logic Shuffling

Function Logic Shuffling This technique alters the control flow path of a malware program by shuffling the order of function calls without affecting the semantics (i.e., purpose) of the original malware program. Though the functionality between the original malware and obfuscated version stays the same, because of the change of appearance in the compiled code, they appear to be different malware families, when in fact, they belong to the same malware family. An example is shown in Fig. 3.1 where the order of function logic MyClass_2 and MyClass_3 is changed.

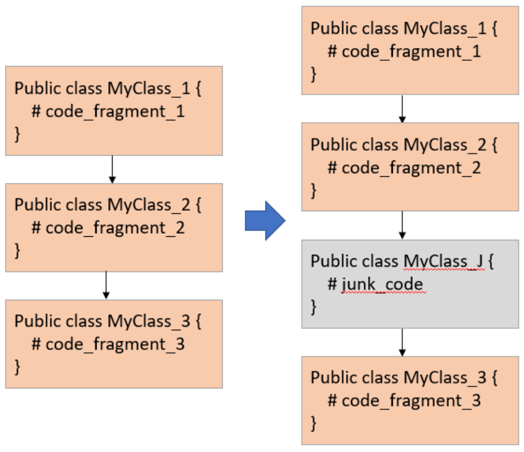


Figure 3.2: Junk Code Insertion

Junk Code Insertion In this technique, the malware author inserts many (junk) code that never gets executed, either after unconditional jumps, calls that never return, or conditional jumps with conditions that would never be met. The main goal o this technique is to change the control flow path to avoid detection or to waste the time for the reverse engineer analyzing useless code. An example of a junk code insertion is shown in Fig. 3.2 where a junk code MyClass_J is added in between two normal function calls.

Function Splitting With this technique, the malware author applies the function splitting method where a function code is fragmented into several pieces each of which piece is inserted across the control flow. This technique splits the function into n code fragments or merges pieces of unrelated codes to make the changes in the compiled code. An example of a function splitting is shown Fig. 3.3 where two splits from MyClass_2 are generated and randomly added among other function calls.

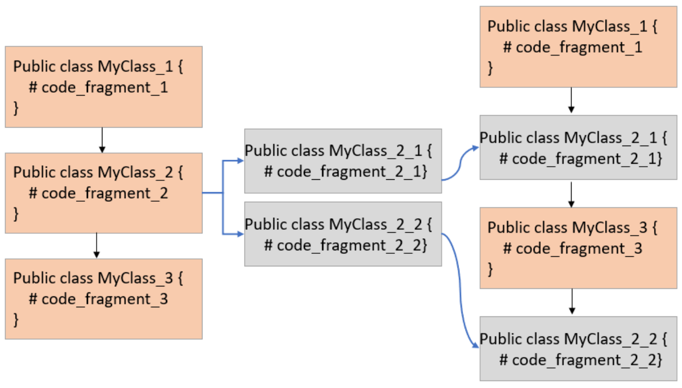

Public class MyClass_1 { # code_fragment_1 }
Public class MyClass_2 { # code_fragment_2 }
Public class MyClass_3 { # code_fragment_3 }
Public class MyClass_2_1 { # code_fragment_2_1}
Public class MyClass_2_2 { # code_fragment_2_2}
Public class MyClass_1 { # code_fragment_1 }
Public class MyClass_2_1 { # code_fragment_2_1}
Public class MyClass_3 { # code_fragment_3 }
Public class MyClass_2_2 { # code_fragment_2_2}

Figure 3.3: Function Splitting

Though the functionality between the original malware and obfuscated versions (e.g., malware variants) stays the same, a malware code applied with the control flow obfuscation can easily avoid detection from many anti-virus programs [36]. To address this, we propose a model resilient against the presence of many variants of malware created as a result of applying the control flow obfuscation technique. Our proposed method utilizes the information gain calculated through the entropy features associated with each malware variant. In our proposal, the entropy features measure the amount of uncertainty of a given probability distribution of a malware program that is not affected by the order of functional logic of the malware program.

subsectionGeneric Approach and Issues

In the last few years, a few-shot learning technology, such as Siamese Neural Network (SNN), which uses only a few training samples to get better predictions has emerged. SNN contains two identical subnetworks (usually Convolutional Neural networks) hence the name Siamese. The two CNNs have the same configuration with the same weights, $\boldsymbol{W} \in \mathbf{R}^d$, where $\boldsymbol{W}$ depicts the model's parameters while $\mathbf{R}^d$ depicts the distance feature in a low dimensional space. The updating of the hyperparameters is mirrored across both CNNs which is used to find the similarity of the inputs by comparing its feature vectors.

Each parallel CNN is designed to produce an embedding (i.e., reduced dimensional representation) of the input. These embeddings can then be used to optimize a loss function during the training phase and to generate a similarity score during the testing phase.

This architecture is effectively different from traditional neural networks that learn to predict multiple classes from a large volume of datasets. This however poses a problem because they need to be retrained and updated on the whole dataset when a new class is added and removed to the dataset. SNN, on the other hand, learns a similarity function by training it to test if the two images are the same. This new architecture enables it to classify new classes of data without training the network again. This allows SNN to be more robust to class imbalance as a few images per class is sufficient for SNN to recognize those images in the future.

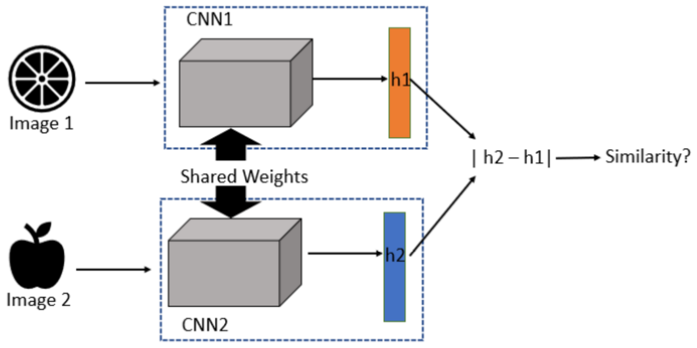


Figure 3.4: The overview of Siamese Neural Network

Fig. 3.4 illustrates the working of a generic SNN. The goal of the above SNN is to determine if two image samples belong to the same class or not. This is achieved through the use of two parallel CNNs (CNN1 and CNN2 in the above figure) trained on the two image samples (image1 and image 2 in the above figure). Each image is fed through one branch of the SNN which generates a d-dimensional feature embedding (h1 and h2 in the above figure) for the image. It is these feature embeddings that are used to optimize a loss function rather than the images themselves. A supervised cross-entropy function is used in SNN for the binary classification to determine whether two images are similar or dissimilar by computing [h2-h1] and processing it by a sigmoid function.

Mathematically, the similarity between a pair of images (x_1, x_2) within Euclidean distance (ED) is computed in SNN using the Equation:

$$d_w(x_1, x_2) = \|F_w(x_1) - F_w(x_2)\|_2 \tag{3.1}$$

where the F_w indicates the feature representation for the inputted feature matrix. Generally, the model g_w: $\mathbf{R}^n$ to $\mathbf{R}^d$ is parameterized by the weights $\boldsymbol{w}$ and its loss function is calculated using the Equation:

$$L_b = -\frac{1}{N}\sum_{i=1}^{N}[y_d^i f_p(d_w^{(i)}) + (1 - y_d^i) f_q(d_w^{(i)})] \tag{3.2}$$

where y_d denotes the label of image pairs. d_w represents the Euclidean distance (ED) between two images at the i-th pair. Note that the most similar images are supposed to be the closest in the feature embedding space. Though this approach would work well for finding similarities/dissimilarities across distinct image objects, this would work not well for obfuscated malware samples.

Recall that an obfuscated malware, say x_1, changes some part of the original malware code, x_2. When these two are converted as feature representation, say $F_w(x_1)$ and $F_w(x_2)$, the feature values in the feature representation would look very different – this is how obfuscated malware avoids detection by anti-virus software. Inadvertently, the different values in the feature representation make the distance across obfuscated malware images very different from one another (i.e., $d_w(x_1, x_2)$ is large). Eventually, when a similarity score is computed and compared, using the loss (i.e., L_b) based on the distance calculation (i.e., $d_w(x_1, x_2)$), they would appear to be different malware families, though, in fact, they all belong to the same malware family.

3.5 Task-Aware Meta Learning-based Siamese Neural Network

We introduce our task-aware meta learning-based SNN model that provides a novel feature embedding better suited for control flow obfuscated malware classification. We start with the overview of our proposed model, the details of the CNN architecture that is used by our model, followed by how task-specific weights are calculated using factorization. Finally, we discuss the details of the loss functions our model use to address the challenges with the weight generation with the limited number of training samples.

3.5.1 Our Model

As shown in Fig. 3.5, our model utilizes a pre-trained network and two identical CNN networks. We use a pre-trained network (VGG-16) to compute more accurate weights for entropy features. Similarly, each CNN takes image features to calculate weight for

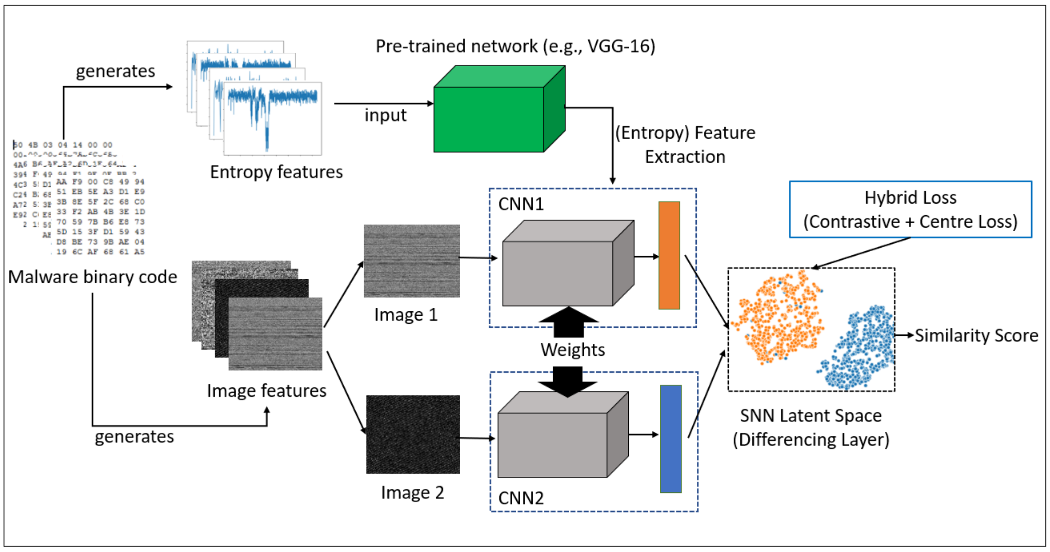

Pre-trained network (e.g., VGG-16)
generates
input
Entropy features
(Entropy) Feature Extraction
Malware binary code
CNN1
Hybrid Loss (Contrastive + Centre Loss)
Image 1
Weights
Similarity Score
Image features
SNN Latent Space (Differencing Layer)
generates
Image 2
CNN2

Figure 3.5: Overview of our proposed model

image features and to generate a feature embedding using the task-specific weights and shared weights of both entropy and image features. The feature embeddings produced by two CNNs are used by the SNN to calculate the similarity score across intra-class variants using a new hybrid loss function.

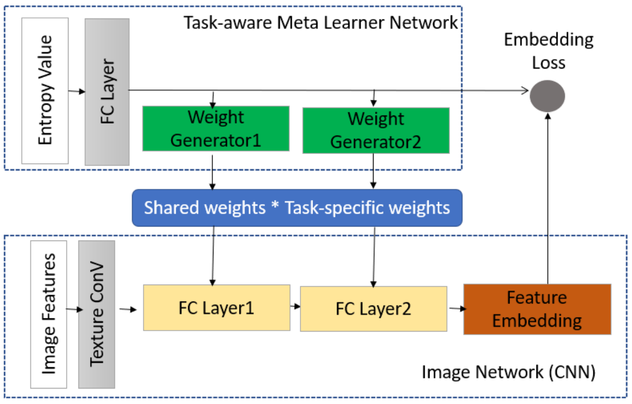


Figure 3.6: CNN Architecture of our proposed model

Within each CNN, there are two sub-networks: a task-aware meta learner network and an image network, as shown in Fig. 3.6. The task-aware meta learner network starts by taking a task specification (e.g., entropy feature vector) and generates the weights. At the same time, the image network (e.g., this is a typical CNN branch of an SNN) starts by taking the image feature and convoluting it until a fully connected layer is produced. The last fully connected layers use the weights generated by the task-aware meta learner network along with the shared weights to produce a task-aware feature embedding space. Embedding loss for inter-class variance is calculated for back-propagation until CNN is fully trained for all input images.

Mathematically, it can be written as the following Equation:

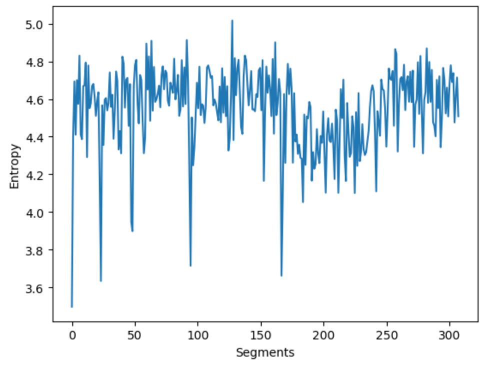
$$y = \mathcal{F}(x; \theta = \{\mathcal{G}(t), \theta_f\}) \tag{3.3}$$

where the SNN $\mathcal{F}$ takes malware images $\mathbf{x}$ as inputs and produce a task-aware feature embedding that is used by the SNN to predict the similarity $\mathbf{y} \in \{1, 0\}$ between an image pair inputted to two CNNs. Each CNN is parameterized by the weights θ which is composed of generated parameters from $\mathcal{T}$ and the share parameters θ_s in the SNN $\mathcal{F}$ that is shared across all malware families. The task-aware meta learner network $\mathcal{T}$ creates a set of weight generators $g^i, i = 1...k$ to generate parameters for k feature layers in $\mathcal{F}$ conditioned on e_t. The overall approach of our proposed model can also be summarised using the following Algorithm 1.

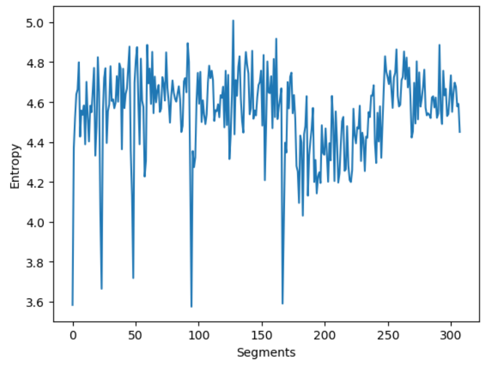
```
Algorithm 2: Pseudo-code of our proposed algorithm
Binary Cross-entropy with center loss:$L_{b,c}$;
Additional supervision loss :$L_e$;
Input  : Entropy Graph feature $F_{ent}$, Texture feature $F_t$, support set $s$, query
         set $q$, Pair label $y_d^i$, Sample label $y_t^i$, Hyper-parameters $\beta$ , Initialized
         Centers $c_i$
Output: [Predicted similarity]
Training stage:
1) Initializing the parameters for our proposed models and the task-specific
   weights $W^i$ for the weight generators at the task-aware meta learner network
   $g^i$ using the weight factorization Eq.3.5.
2) To input the malware texture features $F_t$ and the 4096 features of Entropy
   Value $F_{ent}$ extracted by the pre-trained network (e.g., VGG-16). Note that
   these are integrated with the support set $s$ of our proposed model.
3) And then, the weighted features from Eq.5 are fed into the embedding loss
   according to the one-hot label generated from the target label.
4) Calculating the Euclidean Distance (ED) of features in the two branches of
   SNN through the hybrid loss function.
5) Back-propagation and update parameters by Adam optimizer
Testing Stage:
while not reach to iterations do
    Extract features of samples in the query set $q$ and feed them do the one-shot
      accuracy
Accuracy = 100×$correct/iterations$
```

3.5.2 Task-aware Meta Learner

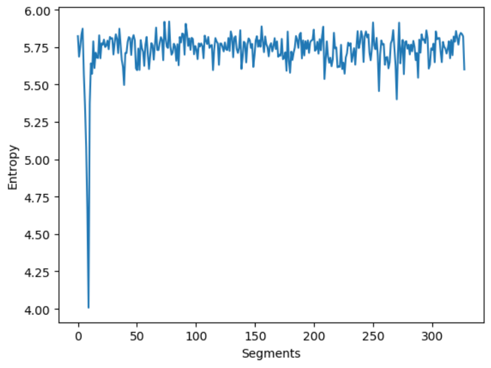
Our task-aware meta learner provides two important functionalities. One is generating optimized task-specific weights using the entropy values extracted from a pre-trained deep learning model (e.g., VGG-16). Another function it provides is to work with the image network to compute the new weights based on the shared weights and the task-specific weights so that the embedding loss is accurately calculated to capture the relative distance across inter-class variance (e.g., the features of the image). These functions are necessary since some malware samples (e.g., zero-day attack samples) usually are much smaller than the number of images required for training a SNN model



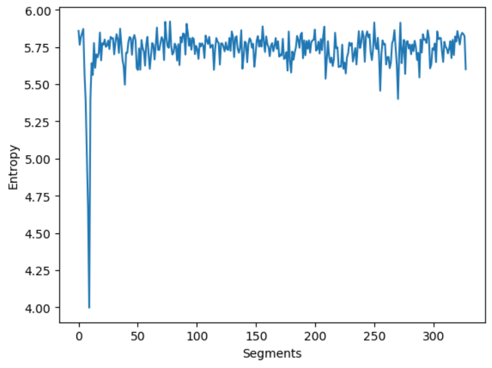
(a) Wroba.wm class



(b) Wroba.wm class



(c) Agent.ad class



(d) Agent.ad class

Figure 3.7: Examples of entropy graphs of two malware families

Using the entropy values, our meta learner recognizes a specific malware signature present in the entropy so that later it uses this knowledge to find whether some malware samples are derived from the same malware family or not (e.g., obfuscated malware). The entropy values are extracted from the VGG-16 when entropy graphs are inputted.

We use entropy graphs to recognize a unique malware signature belonging to each malware family. To illustrate the use of an entropy graph as a task specification, we show four samples of malware images as shown in Fig.3.7. Fig.3.7 (a) and Fig.3.7 (b) are two obfuscated malware samples from the same Wroba.wm family. Similarly, Fig.3.7 (c) and Fig.3.7 (d) of the name Agent.ad are from the same Agent family. One can see that the entropy graphs within the same family share a similar pattern while there are visible differences in the entropy graphs between two different malware families.

Our task-ware meta learner utilizes the entropy values extracted from the entropy graph to train our proposed model to recognize if an image pair is similar or dissimilar (i.e., belong to the same malware family or not) – see Algorithm 3. To obtain an entropy graph, a malware binary file is read as a stream of bytes and separated into a few segments. The frequency of unique byte value is counted and computes the entropy using Shannon's formula as follows:

$$Ent = -\sum_i \sum_j M(i,j) log M(i,j) \tag{3.4}$$

where M is the probability of an occurrence of a byte value. The entropy obtains the minimum value of 0 when all the byte values in a binary file are the same while the maximum entropy value of 8 is obtained when all the byte values are different. The entropy values are then represented as a stream of values which can be reshaped as an entropy graph. The entropy graph is then converted as a feature vector inputted through the convolutional extractor of a pre-trained network (e.g., VGG-16 [131]). The summary of the steps involved in the entropy graph is described in Algorithm 3.

Algorithm 3: Pseudo-code of entropy graph

```
Input  : $f$: malware binary file; $l$: segment length; $n$: the number of files
Output: entropy graph matrix $m$
while not reach to $n$ do
    1. read $l$ bytes from $f$, and defined as segment $s$;
    2. for $i = 0$ to $255$ do
    3.     compute the probability $p_i$ of $i$ appearing in $s$;
    4.     compute the Shannon entropy
Generate entropy graph $m$
```

3.5.3 Weight Generator via Factorization

In the generic SNN approach, the feature extractor only uses the image feature. This approach however is no longer effective in the detection of obfuscated malware where multiple obfuscated malware samples contain almost identical image features.

The problem is further complicated when only a few samples (i.e., less than 5 malware samples) exist (i.e., not enough malware feature information to use for classification as there are very small variations of malware samples to be collected from a small number of malware samples). To address this issue, we present a new novel weight generation scheme based on the work presented by [49]. In our proposed model, the weight generator $G(.,.|\phi)$ gets as input the entropy vectors W_{ave} of a class in addition to the image vectors $\mathbf{Z}' = \{z_i'\}_{i=1}^{N'}$ of the N' training samples of the corresponding class. This results a weight vector $w' = G(Z', W_{ave})$.

In our model, the weight generator scheme is incorporated in the Fully-Connected (FC) layer to solve the non-linear issue that exists in the relationship between the entropy feature and malware image. By integrating the weight generator at the FC layer, the weights of the features extracted before the FC layers can be integrated better into calculating new and more optimized weights for the whole model. We generate weights by creating a weight combination. The weight combination produces the composite features that encode the non-linear connection in the feature space. This is done by multiplying the entropy features and image features together such that the composite features learn a feature embedding resistance to different obfuscated malware variations. Note that the dimension of the weight generator g^i on the FC layer must be matched with the dimension of the weight size of the ith feature layers in F, so that the weights $W_i \in \mathbf{R}^{m \times n}$ can be decomposed using the following Equation:

$$W^i = W^i_{sr} \cdot W^i_{ts} \tag{3.5}$$

where $W^i_{sr} \in \mathbf{R}^{m \times n}$ is the shared parameters for all malware family $\{t_1, ...t_N\}$ and $W^i_{ts} \in \mathbf{R}^n$ is the task-specific parameter for each malware family. With such factorization, the weight generators only need to generate the malware-specific parameters for each malware family in the lower dimension and learn one set of parameters in the high dimension shared across all malware families.

3.5.4 Loss Function

Our proposed model uses two different types of loss functions. Embedding loss is used by our task-aware meta learner network to compute a loss across the inter-class variance (e.g., the features in the feature embedding space of a CNN branch) while a hybrid loss is used by the differencing layer of the SNN to compute the similarities across inter-class variants between an image pair.

Embedding Loss for Meta Learner

The feature representation of the entropy graph of a malware class can be easily influenced by binary loss. This is because the use of binary loss can only give the probability of the distribution of distances between positive and negative image pairs. It cannot estimate the probabilities of distances between positive and negative image pairs across different malware variations therefore not being able to correctly classify similar pairs of images across obfuscated malware samples (i.e., cannot learn a discriminative feature during the training procedure). To address this issue, we add a secondary cross-entropy loss not only to learn the discriminative feature but also the effect of overfitting caused by contrastive loss. This embedding loss is defined using the following Equation.

$$L_e = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} log[\frac{exp(F(x_i;\theta_t)) \cdot y_t^i}{\sum_{j=1}^{T} exp(F(x_i;\theta_t))}] \tag{3.6}$$

where x^i represents the ith sample in the dataset of the size N. $y_t \in$\{0, 1\}^{t} indicates the one-hot encoding applied to the input based on the labels. T indicates the number of tasks during training (e.g., either in the whole dataset or in the minibatch).

Hybrid Loss for Our SNN

To calculate the similarity score for our proposed model, we propose a hybrid loss function comprised of a center loss and a constractive loss. The center loss proposed by Wen et. al. [163] was a supplement loss function to the softmax loss for the classification task. It can learn to find a sample that can act as the center image of each class and try to shorten the distance across the training samples of similar features by moving them to be closer to the center of the sample as much as possible. This center loss can be calculated as follows:

$$L_c = \frac{1}{2N}\sum_{N}^{i=1} \left\| d_w^{(i)} - c_i \right\|_2^2 \tag{3.7}$$

where c_i denotes the center of class i while $d_w^{(i)}$ denoting the features of the Euclidean distance. The objective function stands for the squared Euclidean distance. Intuitively, the center loss encourages instances of the same classes to be closer to a learned class center.

However, this approach doesn't address the issue of moving apart from the training samples of dissimilar features. To address this issue, we propose the hybrid loss function integrated with the pairwise center to better project the latent task embedding $e_t = T(t)$ into a joint embedding space that contains both the negative and positive center points.

We adopt a metric learning approach where the corresponding learned feature is closer to the joint feature embedding for positive inputs of a given image pair while the corresponding learned feature is far away from the joint feature embedding for negative inputs of a given image pair.

$$\min L = \beta \min L_e + L_{b,c} \tag{3.8}$$

where β is the hyperparameter to balance the two terms. In our study, we set it at 0.8.

Our experimental results illustrate that the β value at 0.8 provides the best performance when it was tested on one-shot N-way learning. This result also confirms that the hybrid multi-loss function can reduce the distance between the same classes and enlarge the distance between the different classes. Additionally, it does not change the attribute of the feature in the feature space, so the optimization of this layer will never negatively affect the deeper network layers. This hybrid loss function can also compute classification accuracy with a learned distance threshold on distances.

3.6 Experiments

In this section, we describe the details of datasets we used for experiments, model configuration, and the results of our experiments. The results were obtained by running the experiments on the desktop with the 32GM RAM, Nvidia Geforce RTX 2070(8GB), and Intel(R) Core(TM) i7-9700 CPU @ 3.00 GHz.

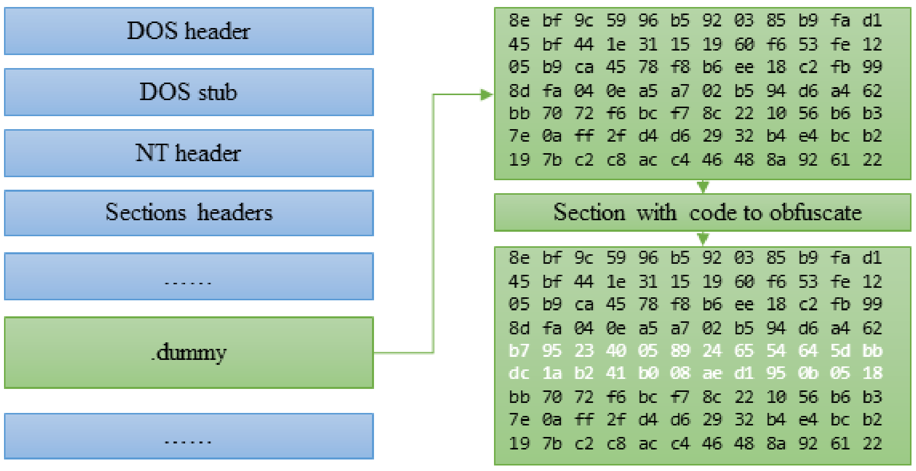
3.6.1 Andro-dumpsys Dataset

Table 3.2: Andro-dumpy Dataset.

No.	Family	Number of variants (+3 Synthetic)	Number of samples (+ 6 Synthetic)
1	Agent	39 (42)	150 (156)
2	Blocal	1 (4)	1 (7)
3	Climap	1 (4)	5 (11)
4	Fakeguard	1 (4)	10 (16)
5	Fech	1 (4)	3 (9)
6	Gepew	4 (7)	112 (118)
7	Gidix	6 (9)	108 (114)
8	Helir	1 (4)	15 (21)
9	Newbak	1 (4)	1 (7)
10	Recal	2 (5)	25 (31)
11	SmForw	23 (26)	166 (172)
12	Tebak	10 (13)	93 (99)
13	Wroba	23 (26)	108 (114)

We use the Andro-dumpsy dataset obtained from [164] that has been widely used for malware detection. The original dataset consists of 906 malicious binary files from 13 malware families. As illustrated in Table 3.2, the number of the malware variants and the total number of samples from different malware families varied. Almost half of the malware family had no more than 25 malware samples while some only had one sample as they were most likely the new malware detected lately (e.g., Blocal and Newbak). In addition to the original dataset, we also generated three additional synthetic malware variants each of which is applied with different control flow obfuscation techniques described earlier: function logic shuffling, junk code insertion, and function splitting, respectively. Two samples from each additional malware variant, a total of 6 additional samples, for each malware family, were added to the original dataset.

Fig. 3.8 illustrates a snippet of how obfuscated malware is created by applying a junk code insertion. In this example, we created a dummy array that acts as a junk code and added in between two function calls from the original malware code. We applied a similar approach for the other two types of control flow obfuscation.



```
8e bf 9c 59 96 b5 92 03 85 b9 fa d1
45 bf 44 1e 31 15 19 60 f6 53 fe 12
05 b9 ca 45 78 f8 b6 ee 18 c2 fb 99
8d fa 04 0e a5 a7 02 b5 94 d6 a4 62
bb 70 72 f6 bc f7 8c 22 10 56 b6 b3
7e 0a ff 2f d4 d6 29 32 b4 e4 bc b2
19 7b c2 c8 ac c4 46 48 8a 92 61 22
```

```
8e bf 9c 59 96 b5 92 03 85 b9 fa d1
45 bf 44 1e 31 15 19 60 f6 53 fe 12
05 b9 ca 45 78 f8 b6 ee 18 c2 fb 99
8d fa 04 0e a5 a7 02 b5 94 d6 a4 62
b7 95 23 40 05 89 24 65 54 64 5d bb
dc 1a b2 41 b0 08 ae d1 95 0b 05 18
bb 70 72 f6 bc f7 8c 22 10 56 b6 b3
7e 0a ff 2f d4 d6 29 32 b4 e4 bc b2
19 7b c2 c8 ac c4 46 48 8a 92 61 22
```

Figure 3.8: Example of Inserting Junk code

We also increased the image sample size to have at least 30 samples for every malware family using a data argumentation technique (e.g., applying random transformations such as image rotations, re-scaling, and clipping the images horizontally). The details of the argumentation parameters are as shown in Table 3.3.

Image Feature

We use the same technique proposed by [191] to produce image features. To produce image features, we first read the binary malware as a vector of 8-bit unsigned integers which are then converted into 2D vectors. We use the fixed width while the height

Table 3.3: Data augmentation parameters

Methods	Values	Description
rescale	1./255	Resizing an image by a given scaling factor.
zca_epsilon	1e-06	Epsilon for ZCA whitening.
fill_mode	wrap	Points outside the boundaries of the input are filled according to the given mode.
rotation_range	0.1	Setting degree of range for random rotations.
height_shift_range	0.5	Setting range for random vertical shifts.
horizontal_flip	True	Randomly flips inputs horizontally.

is decided according to the size of the original file. Finally, the 2D vector is coveted into the gray images using the color range [0, 255]. Note that the gray images at this stage are different dimensions according to the varying heights and widths in size which biases could occur in the fully connected layer. To address this issue, we use the bilinear interpolation method, as suggested by [96], to produce our image feature in uniform to the size of 105×105.

Entropy Feature

To obtain the entropy feature of each malware family, we take each byte of the malware binary file as a random variable and count the frequency of each value (00h-FFh). More concretely, the byte reads from the binary file are divided into several segments. For each segment, we first calculate the frequency of each byte value $p_j (1 \leq j \leq m)$ and secondly, the entropy y_i of the segment is calculated. The entropy values are then represented as a stream of values which can be reshaped as an entropy graph with the size of $254 \times 254 \times 1$. These entropy graphs are then converted as a 4096-dimensional feature vector inputted through the convolutional extractor of the VGG-16 architecture [131].

3.6.2 Model Configurations

The task-aware meta learner network T(t) is a two-layer FC network with a hidden unit size of 512 except for the top layer which is 4096 for input. The weight generator g^i is a single FC layer with the output dimension the same as the output dimension of the corresponding feature layer in $\mathcal{F}$. We add a Relu function to the output of g^i. In the case where the processed malware images are used as inputs, and the convolutional part is configured as the 4-layer convolutional network (excluding pooling layers) following the structure as [77]. Besides, the Relu function and batch normalization are added after by the convolution layer and FC layer. The total number of parameters in our proposed model is 40 million. The overview of our network configurations is described in Fig. 3.9.

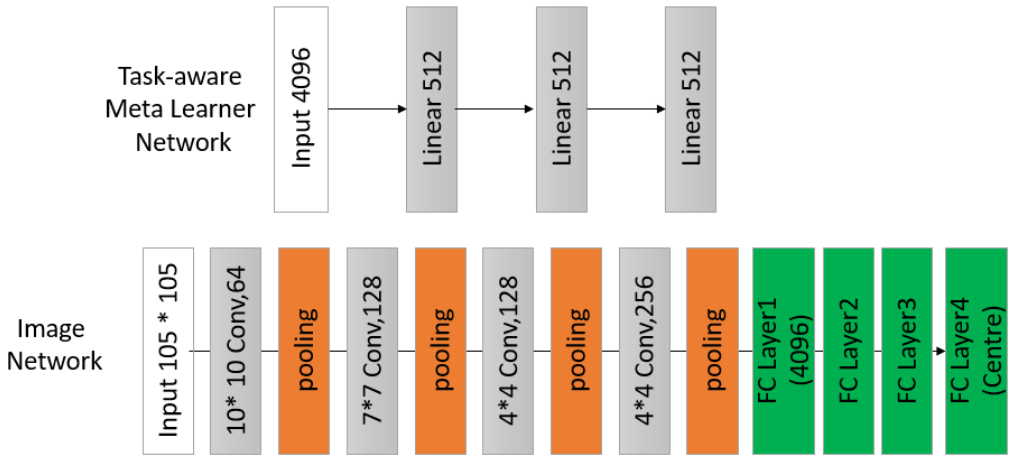


Figure 3.9: Network Configurations

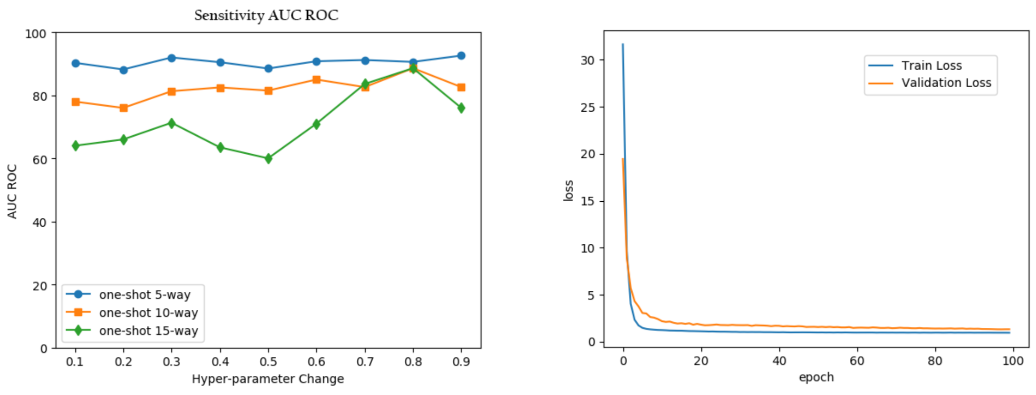
3.6.3 Results

We set the batch size to 32 and used Adam as the optimizer with the initial learning rate of 10^{-4} for the image network and the weight generators, and 10^{-5} for the task embedding network. The network is trained with 50 epochs, which run for approximately 2 hours. As Fig. 3.10 illustrates, both train and validation loss stabilizes after 50 epochs confirming the training is done by this stage. The testing process conducts M times of N-way on N-shot learning tasks, where Q times of correct predictions contribute to the accuracy calculated by the following formula:

$$Accuracy = (100 * Q/m)\% \tag{3.9}$$

N-way matching accuracy

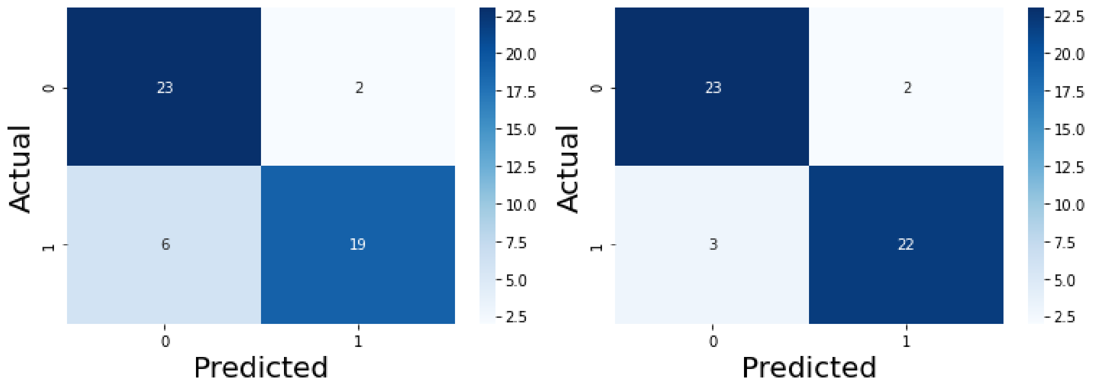
The evaluation of N-way learning at each test state is carried out for one-shot and 5-shot. For N-way one-shot learning, we choose an anchor image from one class of test, and then randomly selects N classes of images to form the support set $X = \{x_i\}_{i=1}^{N}$, where x_1, $\forall x \in X$, where selected image's class is the same as the anchor image $\hat{x}$, the other images in support sets are from different classes. The similarity score between $\hat{x}$ and other images is calculated through our model. To be specific, if the similarity score



(a) One-shot N-way Convergence (b) Loss during Training on Epochs

Figure 3.10: Traing Performance

of the feature vector of x_1, which can be represented as $S = \{s_i\}_{i=1}^{N}$, that score is the maximum of S the task can be labeled as a correct prediction. Otherwise, it is regarded as an incorrect prediction. For N-way 5-shot learning, we randomly select N unseen classes and six instances, in which five instances of each class are randomly selected as the support set, $X = \{x_1, ..., x_i\}_{i=5}^{N}$ and the remaining instances of each class form the query set. Its prediction procedure is the same as the test in the one-shot.

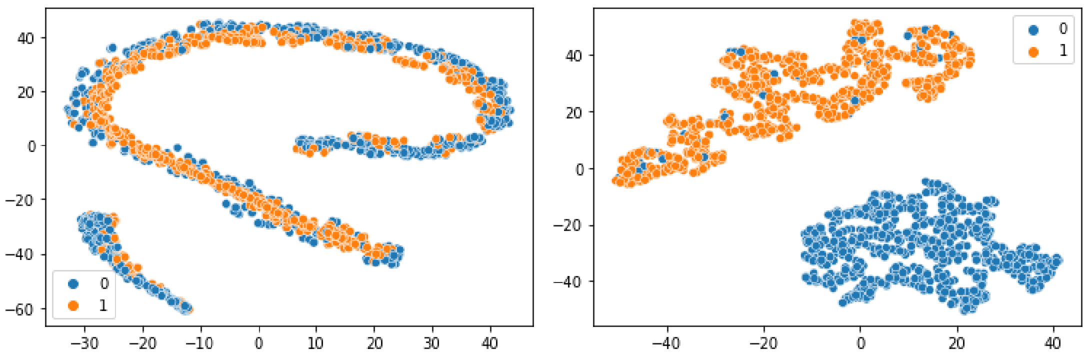


(a) N-way One-shot (b) N-way Five-shot

Figure 3.11: Matching Accuracy

The matching accuracies of N-way accuracy for the one-shot and 5-shot are illustrated in Fig.3.11. We randomly used 50 pairs of images, 25 containing positive image pairs and 25 containing negative image pairs, to test the effectiveness of our proposed model. As shown in the N-way one-shot result in Fig.3.11 (a), 19 out of 25 positive image pairs were matched correctly where there were 6 true negatives (i.e., 2 positive pairs not matched correctly). Similarly, 23 of 25 negative pairs matched correctly while there were 2 false positives (i.e., 2 negative pairs matched incorrectly). For the N-way 5-shot results (shown in Fig.3.11 (b)), the accuracy of matching is higher as almost 22 out 25 pairs matched correctly for both positive and negative pairs, and there were 3 incorrectly matched results.

Embedding space projection



(a) Epoch 1 (b) Epoch 90

Figure 3.12: Feature Embedding Before vs. After Training

Fig.3.12 shows the projection of the embedding space using the 2- dimensional Principal Component Analysis (PCA) technique, where each orange point dictates the distance of a positive pair while each blue point dictates the distance of a negative pair. Fig.3.12 (a) is the embedding space before training while Fig.3.12 (b) projects the embedding space after training. After training, we can clearly see two distinct clusters - one around the distance calculated for all positive pairs and the other for negative pairs. This confirms that our proposed model has learned well to distinguish the similarities between positive pairs and negative pairs and separate them far apart.

Benchmark against similar methods

Table. 3.4 shows the result of benchmarking our proposed model against the current state-of-the-art, especially along with the Matching network and Prototypical network, as well as the original Siamese Network. Our model surpassed the performance of the Matching Network and Prototypical Network by 2.4% and 1.8% on the 1-shot learning on the 5-way. The difference between our results of 1-shot and 5-shot was 3.1%, 1.9%, 1.4% on the 5-way, 10-way, 15-way respectively. Our proposed 5-shot result outperformed all three exiting models by 1.4%, 5.9%, 5.4% respectively on the 5-way, 10-way, and 15-way.

Table 3.4: Comparison of classification performance of different few-shot learning approaches for Andro-dumpy datasets

1-shot				
Ref.	Method	5-way	10-way	15-way
[155]	Matching Network	85.7	84.3	76.8
[134]	Prototypical Network	86.3	82.9	81.2
[77]	Siamese Network	82	69.2	64
	Task-aware SNN	88.1	86.6	82.1
5-shot				
Ref.	Method	5-way	10-way	15-way
[155]	Matching Network	89.8	86.2	78.4
[134]	Prototypical Network	85.3	82.6	82.6
[77]	Siamese Network	85.8	72.6	69.9
	Task-aware SNN	91.2	88.5	83.5

Distance measure effectiveness

We also examine the effectiveness of our proposed model in distance measurement by using the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. The AUC-ROC curve is commonly composed of two performance measures, the true-positive (FPR) rate and the false-positive rate (FPR) rate, respectively. The equations related to these two performance measures of the ROC curve are shown as follows:

$$\begin{aligned} FPR(P^*) &:= \int_{p^*}^{1} f_0(p)dp \\ TPR(P^*) &:= \int_{p^*}^{1} f_1(p)dp \end{aligned} \tag{3.10}$$

where the $f_0(p)$ is denoted by the probability of a density function for the predictions $p(x)$ produced by our proposed model. The negative pairs are labelled as 0, and $f_1(p)$ is the probability from the positive pair that is labelled as 1. The given discrimination threshold P are the integrals of the tails of these distributions according to the true-positive rate and false-positive rate. Based on the two parameters TPR and FPR, the AUC-ROC curve is defined as follows:

$$AUC - ROC = \int_{0}^{1} TPR(FPR)D(FPR) \tag{3.11}$$

where the AUC measures the entire two-dimensional area underneath the entire ROC curve (i.e., integral calculus) from (0,0) to (1,1). For example, a model whose predictions are 100% wrong has an AUC of 0.0 while a model whose predictions are 100% correct has an AUC of 1.0. Using this concept, we demonstrate the result predicted by the saved weights of 1-shot and 5-shot respectively of the learned model on the test set including 5-way, 10-way, and 15-way. These AUC-ROC curves are shown in Fig 3.13.

As it shows, our result is on a set of points in the true positive rate - false positive rate plane. The results achieve the AUC-ROC equals 0.92, 0.91, and 0.80 respectively under the 5-way, 10-way, and 15-way on the one-shot learning. We further conducted the AUC-ROC on the 5-shot learning. Our proposed model also obtained better performance than the generic SNN with 95.6, 90.7, and 86.8 % at 5-way, 10-way, and 15-way respectively. As expected, the accuracy of both 1-shot and 5-shot drops as the number of N-way increases with higher intra-class variance.

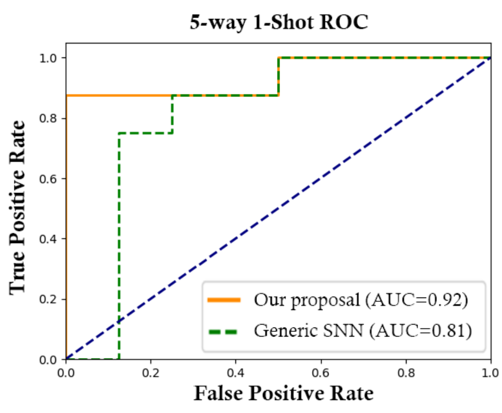
Note that our model always performs better as shown in these graphs as the AUC-ROC areas (i.e., the areas up to the blue line) of our proposed is larger compared to the generic SNN.

3.7 Conclusion

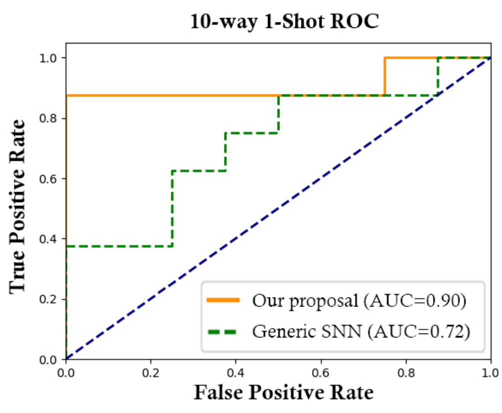
We proposed a novel task-aware meta-learning based Siamese Neural Network to accurately classify different malware families even in the presence of obfuscated malware variants. Each branch of CNN used by our model has an additional network called “task-aware meta learner network” that can generate task-specific weights using the entropy graphs obtained from malware binary code. By combining the weight-specific parameters with the shared parameters, each CNN in our proposed model produces the fully connected feature layers so that the feature embedding at each CNN is accurately adjusted for different malware families despite there being obfuscated malware variants in each malware family. However, the detection environment for malware under the packaging of obfuscation technology is complex, and our proposed model cannot accurately detect malware with multiple obfuscation packaging environments.

In addition, our proposed model can provide accurate similarity scores even if it is trained with a limited number of samples. Our model also uses a pre-trained VGG-16 network in a meta-learning fashion to computer-accurate weight factors for entropy features. This meta-learning approach essentially solves the issues that are associated with creating potential bias due to not having enough training samples.

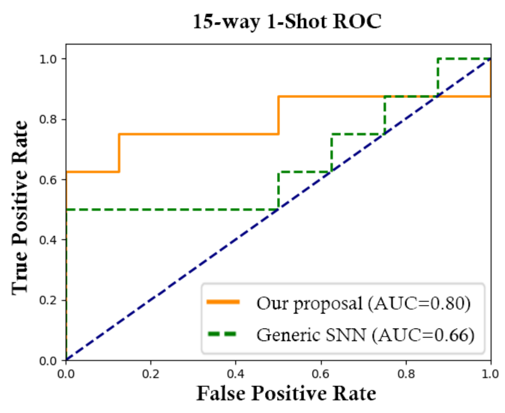
Our model also offers two different types of innovative loss functions that can more accurately compute the similarity scores within a CNN and the feature embeddings used by two CNNs.



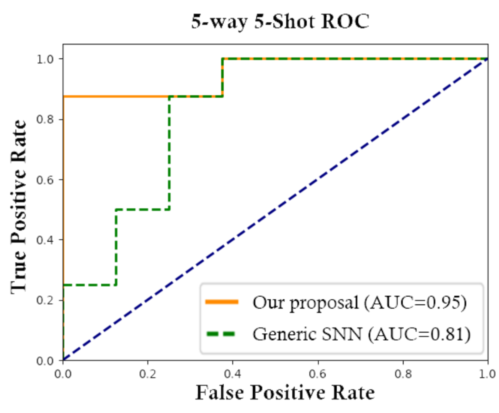
(a) ROC of 5-way One-shot



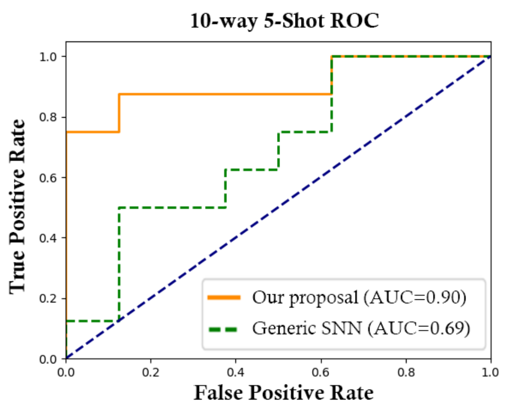
(b) ROC of 10-way One-shot



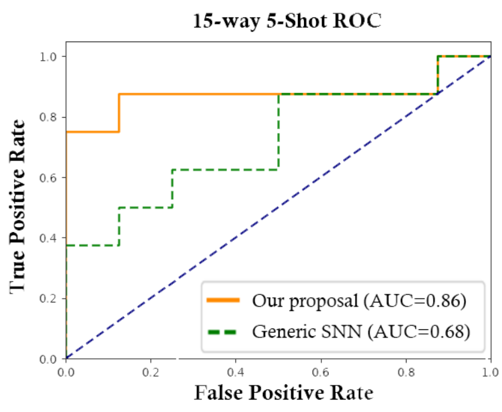
(c) ROC of 15-way One-shot



(d) ROC of 5-way 5-shot



(e) ROC of 10-way 5-shot



(f) ROC of 15-way 5-shot

Figure 3.13: AUC-ROC curves under the N-way N-shot

Our experimental results show that our proposed model is highly effective in recognizing the presence of unique malware signatures thus able to correctly classify obfuscated malware variants that belong to the same malware family.

We are planning to apply different types of malware samples (e.g., DDoS attack [161] and ransomware families [188; 189; 98; 97]) and other data samples (e.g., finding similar abnormalities in medical x-ray images [40]) to test the generalizability of our model.

Chapter 4

A Few-Shot Meta-Learning based Siamese Neural Network using Entropy Features for Ransomware Classification

4.1 Summary

Ransomware defence solutions that can quickly detect and classify different ransomware classes to formulate rapid response plans have been in high demand in recent years. Though the applicability of adopting deep learning techniques to provide automation and self-learning provision has been proven in many application domains, the lack of data available for ransomware (and other malware) samples has been raised as a barrier to developing effective deep learning-based solutions. To address this concern, we propose a few-shot meta-learning-based Siamese Neural Network that not only detects ransomware attacks but is able to classify them into different classes. Our proposed model utilizes the entropy feature directly obtained from ransomware binary files to retain more fine-grained features associated with different ransomware signatures. These entropy features are used further to train and optimize our model using a pre-trained network (e.g., VGG-16) in a meta-learning fashion. This approach generates more accurate weight factors, compared to feature images are used, to avoid the bias typically associated with a model trained with a limited number of training samples. Our experimental results show that our proposed model is highly effective in providing a weighted F1-score exceeding the rate >86% compared to other similar methods.

4.2 Introduction

Ransomware is a type of malware designed to attack the victim's system and denies access to the victim's sensitive data until a ransom is paid [34]. With the popularity of bitcoin which hides the true identity of the account used by attackers along with the prevalence of encryption techniques, this type of malware has proliferated recently, causing millions of dollars in losses for businesses and consumers.

There are two approaches for performing ransomware (and other malware) detection: static and dynamic-based analysis. The dynamic analysis-based detection method can provide a high detection rate [3; 1], but requires the execution of malicious code. However, this may cause a serious scalability issue in processing the large number of previously unseen binaries that come across the desks of malware analysts. In addition, they are not able to extract significant behavioural patterns of the ransomware due to anti-emulation techniques such as fingerprinting or time bombs [120]. Furthermore, different ransomware families vary significantly thus make difficult to profile their behaviour pattern [34; 15]. Static analysis overcomes some of the dynamic analysis detection issues by detecting ransomware attacks prior to their execution. This is done by analyzing bytes or instruction sequences to carry out the malicious activity (i.e., the signature of ransomware in the code). For this reason, we utilize static code analysis to obtain unique features of ransomware despite the obvious advantages present in the dynamic analysis.

With the popularity of Artificial Intelligence (AI) techniques, many (shallow) machine learning and deep learning methods have been proposed. We argue that there are two significant drawbacks to the existing state-of-the-art. The first issue is the majority of existing research tends to focus on learning from the generic features of malware images from different malware families and building feature profiles [106; 74; 23; 91; 175]. Here the feature profiles represent the (static) features contained in a malware sample to provide an important clue as to whether the image sample is malicious or not.

We also argue that there are issues with using malware grayscale images as feature profiles. For example, the obfuscation and repackaging used to introduce new ransomware variants is treated as a form of noise when creating the grayscale image. This often makes it impossible to accurately classify many variants that come from the same malware family [99; 153; 173]. Secondly, malware grayscale images of executables are of different sizes and shapes. In order to use them as input to the deep learning model, they need to be downsampled to the same shape, resulting in a loss of information compared to the original file. Finally, the computational cost [54] of training a model based on grayscale images is higher compared to entropy-based features. Moreover, only using the common static information (e.g., program byte sequences, instructions in the form of disassembled opcodes, functional calls) as data input points to feed into a deep neural network cannot capture unique characteristics of each malware family signature. This shortcoming limits the use of static information not able to accurately classify many variants driven from the same malware family [99; 153; 173].

The second issue with the existing approaches is the demand for large data inputs to find more relevant correlations across the features [187]. They are unable to detect and classify the malware families trained with a limited number of samples [17; 191; 188].

In this research work, we propose a few-shot learning based method Siamese Neural Network (SNN) that is capable of not only detecting different ransomware families but also accurately classifying ransomware variants belonging to the same ransomware family even in the presence of only a small number of training samples available.

The contributions of our proposed model are the following:

- Our proposed model can learn unique ransomware signatures from different ransomware classes even if the number of ransomware samples for each class is very small.

- Different from the existing models, which typically use image features, we use entropy features directly obtained from each ransomware binary file as inputs to our model. This prevents information loss typically associated with image feature conversion. The use of entropy features thus significantly contributes towards accurately distinguishing different ransomware signatures represented in different ransomware classes, addressing the first issue we discussed earlier.

- To solve the second issue associated with the existing approaches, we use a pre-trained VGG-16 network as a part of the meta-learning process to generate weights that more accurately capture the characteristics of each ransomware sample. This not only contributes to improving the classification accuracy but also avoids the potential bias associated with the use of a limited number of training samples in a deep learning model.

- Our proposed model uses a combination of two centre losses and a softmax loss to accurately capture the similarity between the ransomware samples belonging to the same class (i.e., intra-class variance) and the di-similarity across the ransomware samples across different ransomware classes (i.e., inter-class variance).

- Our experiments, tested on a total of 1,046 ransomware samples with 11 different classes, show that the proposed model is highly effective for correctly classifying different ransomware samples by achieving the classification with the weighted F1-score exceeding 86%, which outperformed other existing benchmark methods.

The rest of the paper is organized as follows. Section 4.3 presents related works for ransomware detection and classification. Section 4.4 provides the details of the proposed model. Section 4.5, describes the experimental setup and evaluation metrics. In Section 4.6, the experimental results are presented and discussed. Lastly, Section 4.7 draws a conclusion from our proposed method and describe the future work that is planned.

4.3 Related work

4.3.1 Feature analysis for malware detection

Dynamic analysis avoids the problem of having to preprocess samples using an unpacker. However, it has several limitations. First, it requires either partial or full execution of the malware itself using an emulator or sandbox. Second, the malware sample may use anti-virtualization techniques that detect and prevent its execution in a sandbox or have particular conditions before running that are not triggered in a sandbox environment. Third, there is no guarantee that you will achieve sufficient code coverage for accurate classification. Static analysis based upon reverse engineering also suffers from costs associated with dealing with obfuscation or performing data or control flow techniques. Therefore, researchers have looked for effective but efficient static analysis techniques that do not require reverse engineering such as disassembly of the malware samples. Nataraj et al. 2011 [106] proposed a less expensive but effective static analysis technique based on image processing methods. Their central insight was to treat binary programs like images and use a visual similarity to avoid either execution or disassembly of the malware. Their preliminary work showed promising results, and other researchers have subsequently expanded upon their ideas. More recently Kim et al. [74] proposed a binary classification system that takes a hybrid approach that extracts features from the portable executable (PE) file as well as using both black and white and colour images created from the file's binary representation. The features derived from the PE file header and sections include entropy and packers, where the packers are classified using YARA rules. Decision tree, random forest, and gradient boosting algorithms are used to classify the malware based on the features from the PE file.

4.3.2 Shallow machine learning for malware detection

Several machine learning approaches have been attempted to automate the learning of different malware signatures. SigMal [76] extracts features from the PE structure of malware executable. SigMal transforms these features into a digital image. Signatures are extracted from images and investigated with similarity measures using the KNN technique. They claimed above 99% precision evaluated on the samples they collected. Baldwin and Dehghantanha [11] used static analysis to extract opcode characteristics as input features for a support vector machine (SVM) classifier to classify five class ransomware families with 96.5% accuracy. In the same vein, Zhang et al. [180] transformed raw opcode sequences into N-gram-based input features for a random forest algorithm to achieve 91.43% accuracy. The downside of these techniques is they need to use reverse engineering with two main problems - scalability to convert binary representations into opcodes and anti-disassembly techniques used to generate incorrect opcode sequences.

4.3.3 Deep Learning for Malware Detection

To capitalize image-based features proposed by [106], various end-to-end deep learning solutions based on convolutional neural network (CNN) architecture are proposed are in literature [32; 30]. Furthermore, these works were extended to have a capability of CNN under limited input samples available for training with transfer learning such as Chen [23] who proposed the training of their model with Inception neural network's pre-trained weights. In this work, low-level features were borrowed from the Inception model and applied that transferred knowledge to malware classification. In the same vein, Lo et al. [91] employed pre-trained weights from the Xception neural network with malware classification. Likewise, Yue [175] proposed weighted softmax loss for the CNN model to classify malware in an unbalanced dataset along with VGG pre-trained weights to fine-tune the CNN model.

The static analysis offers early detection of ransomware but sometimes ransomware uses obfuscation and polymorphism techniques that avoid signature detection through static analysis [98]. To address this, various hybrid solution is proposed that combines static and dynamic techniques such as system resource usages and statistics with opcode frequency [41; 129]. A CNN algorithm was used to classify the malware based on the images. A final decision algorithm uses majority voting to come to a final classification as malicious or benign. This approach achieved promising accuracy but requires an extra step of preprocessing the PE file itself whereas we focus on a single image and can deal with the scarcity of training data. Zhu et al. [191] proposed an SNN-based model to classify different Android malware types. They use a batch normalization layer and multiple loss functions to address the overfitting issue associated with a model trained with a limited number of samples. These existing models tend to be effective at capturing malware samples when the common attribute of malicious code is distinct but often do not consider when there are slight differences in features of the malware as is the case of many obfuscated malware variants.

4.4 Materials and methodology

Siamese Neural Network (SNN) has been applied in many applications such as computer vision and natural language processing. Its core function involves estimating the similarity between two images represented by feature embedding space and generating a similarity score. A generic SNN is comprised of two sub-networks, as shown in Fig 4.1, where the weights and hyperparameter settings are shared. Both sub-networks take one image belonging to the same class (i.e., positive pairs) as input and output the features it learned. These features are projected in the feature embedding of the fully connected layer while a loss function is used to predict whether the images processed by two sub-networks belong to the same class or not.

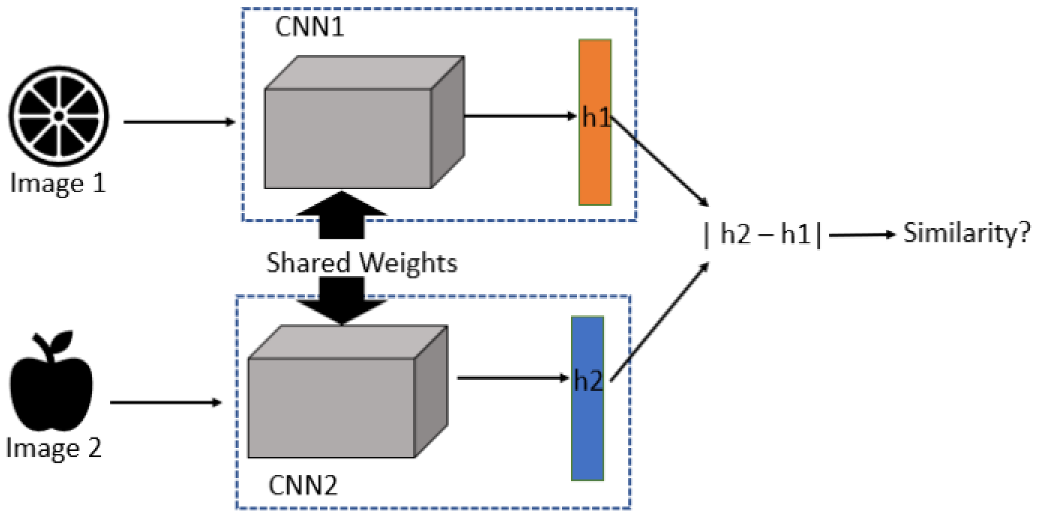
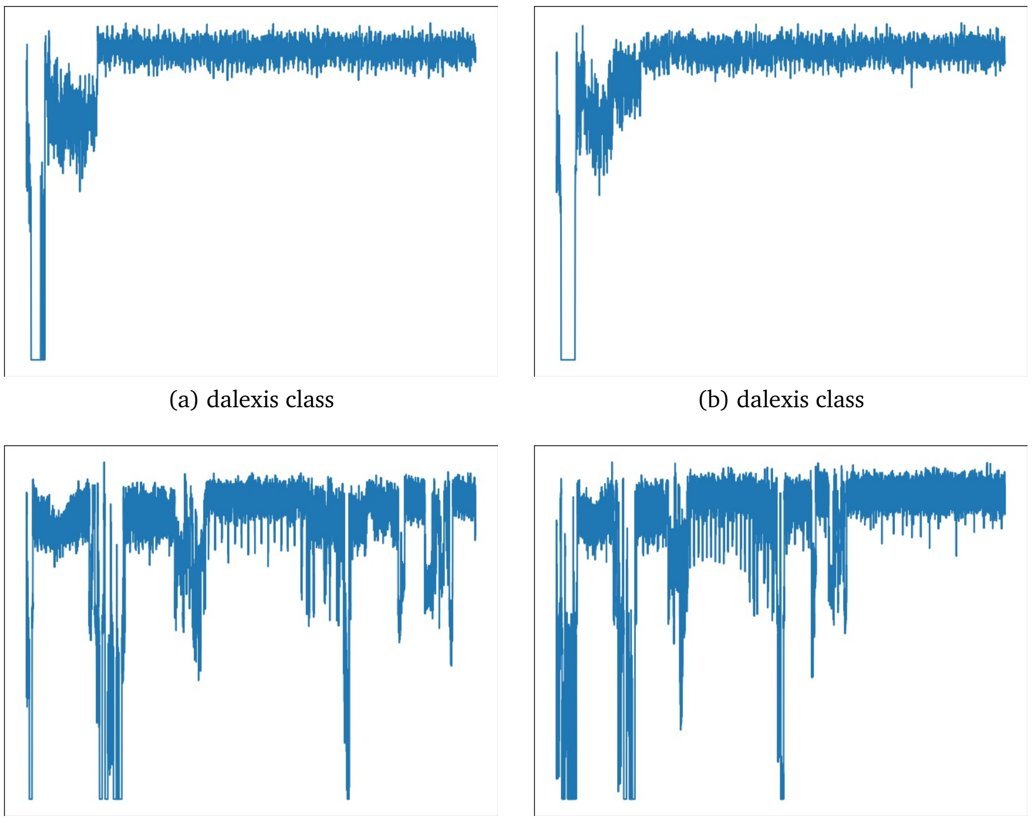


Figure 4.1: The overview of Siamese Neural Network

It can also predict if the different features learned are complementary to different instances within the same class. For example, it can correctly predict that different ransomware instances still belong to the same ransomware family as they share a unique ransomware signature.

4.4.1 Entropy Feature

The main innovation of our proposed model is using entropy features with SNN to train and optimize the model in a meta-learning fashion that utilizes already trained machine learning algorithms (e.g., VGG-16). The entropy features we use are directly computed from binary ransomware files therefore they retain unique characteristics involved in each ransomware signature much better, compared to image features [106] typically used in the existing state-of-the-art. This has a clear advantage over using image features as inputs which tends to lose precise information through conversion processes in order to produce the size of images that works better for a specific model. By using the entropy feature, any computations our model goes through to find feature correlations across different ransomware samples become more robust compared to when the image features were used. This is because image features can be easily influenced by white noise and complex texture features while entropy features are less sensitive to the spatial distribution of the small changes made to the generic code thus able to recognize any arbitrary changes better (i.e., obfuscation methods applied to avoid detection) [53; 191; 188]. Thus, the entropy feature resulted in more accurate classification and is especially useful to find ransomware variants derived from the generic code of the same ransomware family.


(a) dalexis class (b) dalexis class

(c) WannaCry class (d) WannaCry class

Figure 4.2: Examples of entropy graphs of two malware families.

To illustrate the usefulness of using entropy features, Figure 4.2 shows the entropy graphs, containing entropy values from entropy features, obtained from two different ransomware families each of which has two variants. Figures 4.2(a) and 4.2(b) are two ransomware variants from the same ransomware family "dalexia" while Figures 4.2(c) and 4.2(d) are two ransomware variants from the ransomware family "WannaCry". There is a significant difference in the shape of the entropy graphs between two families thus providing very unique feature information when trained. Among the variants of the same ransomware family, there is a slight difference in the entropy graphs but the general pattern shown in each entropy graph is similar to each other thus our model recognizes them as the instances belonging to the same ransomware family.

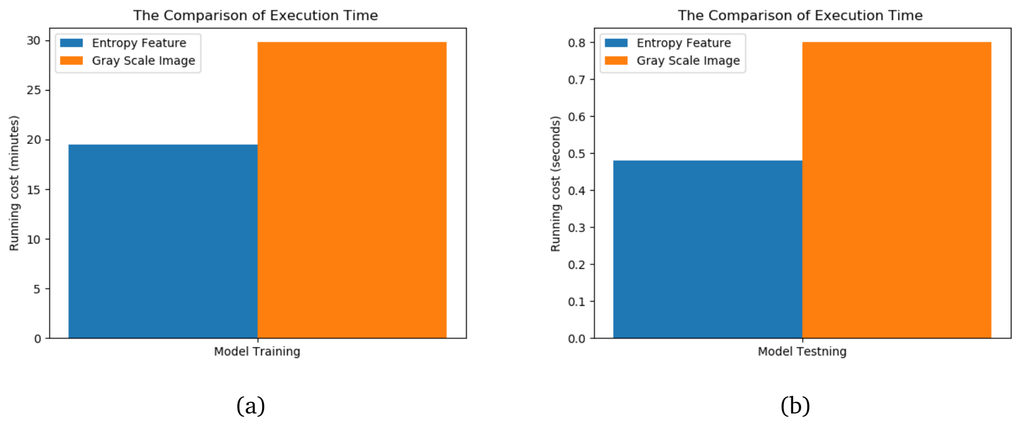


Figure 4.3: The comparison of execution time with or without entropy feature.

We tested the computational cost by comparing the training time of our proposed model with entropy features and with grayscale images. As shown in Figure 4.3, (a) the proposed model based on entropy features takes about 10 minutes less time to train than the model based on grayscale images. (b) The execution time of the model based on entropy features for a new test example is less than 2 seconds compared to the model based on grayscale images.

Our model utilizes the entropy values extracted from the entropy graph to train our proposed model to find different ransomware classes. The overall algorithm involved to produce an entropy graph from a ransomware binary file is depicted in Algorithm 4.

```
Algorithm 4: Generating Entropy Graph
  Input  : $f$: malware binary file; $l$: segment length; $n$: the number of files
  Output: entropy graph matrix $m$
  while not reach to $n$ do
      1. read $l$ bytes from $f$, and defined as segment $s$;
      2. for $i = 0$ to $255$ do
      3.     compute the probability $p_i$ of $i$ appearing in $s$;
      4.     compute the Shannon entropy
  Generate entropy graph $m$
```

To construct an entropy graph, we first read a ransomware binary file as a stream of bytes. The bytes are made into multiple segments each of which is comprised of 200 bytes. We further count the frequency of unique byte value followed by computing the entropy using Shannon's formula as follows:

$$Ent = -\sum_{i}\sum_{j} M(i,j) log M(i,j) \tag{4.1}$$

where M is the probability of an occurrence of a byte value.

The entropy obtains the minimum value of 0 when all the byte values in a binary file of ransomware are the same while the maximum entropy value of byte value 8 is obtained when all the byte values are different. The entropy values are then concatenated as a stream of values which can be formed as an entropy graph.

4.4.2 Our proposed Model

The overview of our proposed model is shown in Figure 4.4. The entropy values from each entropy graph are fed into each sub-network of SNN. At each sub-network, we use a pre-trained VGG-16 whose weights and parameters were trained on ImageNet and use it in a meta-learning fashion (i.e., the pre-trained model assists the training of our proposed model).

The VGG-16 architecture we use has 5 blocks each of which comprises several convolution layers and a pooling layer. The first two blocks contain 2 convolution layers of the receptive size of 3 $\times$ 3 with ReLU activation functions. The first block contains the convolution layer with 64 filters while the second block contains the convolution layer with 128 filters. The next three blocks contain 3 convolution layers, again with the receptive size of 3 $\times$ 3 with ReLU activation functions. The convolution layer in the third block contains 256 filters while the last two blocks (i.e., fourth and fifth) contain 512 filters at each convolution layer. The convolution stride is fixed at 1 pixel as well as the padding size at 1 pixel. We use max-pooling at each block over a 2 $\times$ 2 pixel window with a stride of 2 pixels.

This pre-trained VGG-16 architecture is further trained and optimized with the entropy values inputted as 2-dimensional vectors of a fixed size 224 $\times$ 224. By first utilizing weights and parameters well trained with the image samples and further fine-tuning the parameters of our proposed model with entropy features. This way can avoid potential bias associated with training a deep learning model with a limited number of samples. Note that each sub-network takes one entropy graph as input from the same ransomware family.

The five blocks of VGG-16 architecture are followed by two fully connected layers with a fattening layer in-between. The first fully connected layer has 1024 neurons and the last fully connected layer serves as the output layer and has 512 neurons. We apply a centre loss function to the final output layer to compute a loss across the inter-class variance (e.g., the combination features with two sub-networks). Finally, a softmax loss is used for categorical classification to distinguish different ransomware families.

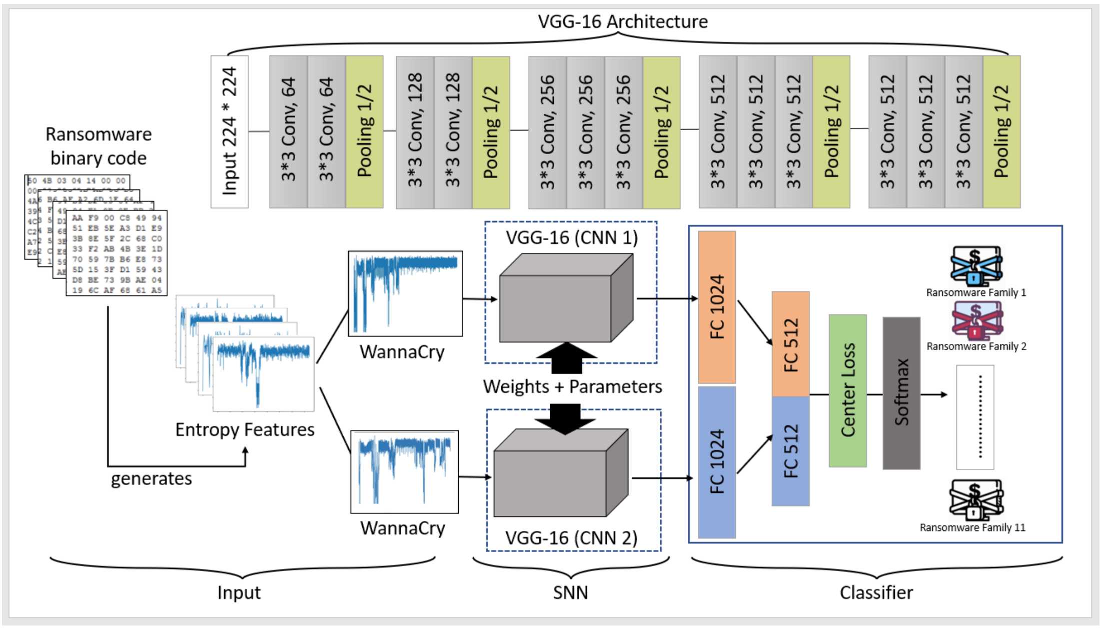


Figure 4.4: The overview of our proposed SNN using Entropy features

Algorithm 5 illustrates the pseudocode involved in our proposed model in terms of training and testing stages.

```
Algorithm 5: Pseudocode for Our Proposed Algorithm
Input  : Entropy Graph Feature $x_i$, Sample label $y_t^i$, Hyper-parameters $\beta$ ,
         Initialized Centers $c_{y_t}$ and Class Weights $w_{y_t}$
Output: Class Probability of Each Class
Dataloader = SelectPostivePairs($y_t^i$)
for $x_1$, $x_2$ in Dataloader:
    $z_t^1, z_t^2$ = EncoderNetworks($x_t^1, x_t^2$)
    $z$ = WeightedCenterLayer([$z_t^1, z_t^2$], $c_i$ ,$w_i$)
    $s_z$ = SoftMaxLayer($z$)
    Loss = CrossEntropyLoss($s_z$) + 0.3 $\times$ CenterLoss($z$)
    Loss.backward()
    Optimizer.step()
```

4.4.3 Classification

A softmax loss function alone cannot recognize intra-class variance (i.e., differences of features within a class) and inter-class variance (i.e., differences of features across classes). To overcome this problem, we first use a center loss function proposed by Wen et al. [163] to make the distance of the features belonging to different classes further away while the distance of the features belonging to the same class be closer.

$$L_s = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} log\left[\frac{exp\left(F\left(x_i;\theta_t\right)\right)\cdot y_t^i}{\sum_{j=1}^{T} exp\left(F\left(x_i;\theta_t\right)\right)}\right] \tag{4.2}$$

where x_i represents the ith sample in the dataset of the size N. $y_t \in \{0,1\}^t$ indicates the one-hot encoding applied to the input based on the labels. T indicates the number of tasks during training (e.g., either in the entire dataset or in the mini-batch).

Additionally, we also use the second term of center loss to create clusters for different ransomware families. The distance across the training samples belonging to the same class is shortened around each cluster while the distance across the training samples of different classes is separated. However, it also implies that in the course of learning a center cluster and the feature x_i, they may have a similar direction in the feature embedding during the gradually optimizing step. To mitigate the redundancy in the learning procedure, we assign different weights on each class cluster, which is formally defined as follows.

$$L_c = \frac{1}{2N}\sum_{N}^{i=1} \|x_i - w_{y_t} c_{y_t}\|_2^2 \tag{4.3}$$

where c_{y_t} denotes the center of a class i while x_i denotes the features of the Euclidean distance and w_i indicates the class weights [75] on each center cluster.

The objective function stands for the squared Euclidean distance. Observed from Eq (4.3), the center loss with different weight parameters encourages instances of the same class to be closer to a learned class center by different gradient descent directions of each class.

The classification probability is calculated by the use of two terms of center loss and the softmax loss function, as shown below.

$$\begin{aligned} Classification = &-\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} log\left[\frac{exp\left(F\left(x_i;\theta_t\right)\right)\cdot y_t^i}{\sum_{j=1}^{T} exp\left(F\left(x_i;\theta_t\right)\right)}\right] \\ &+ \alpha\frac{1}{2N}\sum_{N}^{i=1} \|x_i - w_{y_t} c_{y_t}\|_2^2 \end{aligned} \tag{4.4}$$

where α is the hyperparameter to balance the two terms.

4.5 Experiment Setup and Evaluation Metrics

This section describes the dataset, the experiment setup, and the performance metrics used to evaluate the proposed approach.

4.5.1 Dataset Description

In order to assess the effectiveness of the proposed method, the following benchmark dataset is used in this study.

The binaries of this dataset's instances were obtained from ViruseShare. The dataset comprises 11 families/classes of ransomware, each of which consists of a varying number of instances as listed in Table 4.1. Clearly, the dataset is highly imbalanced, which reflects reality, as some classes, e.g., Petya and Dalexis, are largely outnumbered by the other classes, e.g., Zerber, as shown in the third column of Table 4.1.

Table 4.1: Details of the ransomware dataset.

Family	Instances	Ratio (%)
Bitman	99	9.45
Cerber	91	8.68
Dalexis	9	0.86
Gandcrab	100	9.54
Locky	96	9.16
Petya	6	0.57
Teslacrypt	91	8.68
Upatre	18	1.72
Virlock	162	15.46
Wannacry	178	16.98
Zerber	198	18.89

4.5.2 Experimental Setup

Table 4.2 summarizes the system configuration. This study was carried out using a 3.6 GHz 8-core Intel Core i7 processor with 32 GB memory on Windows 10 operating system. The proposed approach is developed using Python programming language with several statistical and visualization packages such as Sckit-learn, Numpy, Pandas, Pytorch, and Matplotlib.

4.5.3 Evaluation Metrics

The proposed method is compared and evaluated using Accuracy, Precision, Recall, F1-score, and Area under the receiver operating characteristics (ROC) curve. In this work, we have used the macro and micro average of Recall, Precision, and F1-score for multi-class classification. All the above metrics can be obtained using the confusion matrix

Table 4.2: Implementation environment specification.

Unit	**Description**
Processor	3.6 GHz 8-core Inter Core i7
RAM	32 GB
GPU	GeForce GTX 1080 Ti
Operating System	Windows 10
Packages	Pytorch, Sckit-Learn, Numpy, Pandas, Pyriemannian and Matplotlib

(CM) which is described in Table 4.3.

Table 4.3: Illustration of confusion matrix.

		Predicted	
		Class$_{pos}$	Class$_{neg}$
Actual	Class$_{pos}$	True Positive (*TP*)	False Positive (*FP*)
	Class$_{neg}$	False Negative (*FN*)	True Negative (*TN*)

In Table 4.3, True positive (TP) means the amount of class$_{pos}$ data predicted actual belong to class$_{pos}$, True negative (TN) is amount of class$_{neg}$ data predicted is actually class$_{neg}$, False positive (FP) indicates data predicted class$_{pos}$ is actual belong to class$_{neg}$ and False negative (TN) is data predicted as class$_{neg}$ but actually belong to class$_{pos}$. Based on the aforementioned terms, the evaluation metrics are calculated as follows. For a balanced test dataset, higher accuracy indicates the model is well learned, but for unbalanced test dataset scenarios relying on accuracy can give the wrong illusion about the model's performance. Thus for the unbalanced datasets, recall and F1-score metrics give a more intuitive explanation of the model's performance.

Recall (also known as true positive rate) estimates the ratio of the correctly predicted samples of the class to the overall number of instances of the same class. It can be computed using Eq 4.5. A higher Recall value indicates the good performance of a model.

$$Recall = \frac{TP}{TP + FN} \tag{4.5}$$

Precision measures the quality of the correct predictions. Mathematically, it is the ratio of correctly predicted samples to the number of all the predicted samples for that particular class as shown in Eq 4.6. Precision is usually paired with Recall to evaluate the performance of a model. Sometimes pairs can appear contradictory thus comprehensive measure F1-score is considered for unbalanced test data-sets.

$$Precision = \frac{TP}{TP + FP} \tag{4.6}$$

F1-score computes the trade-off between precision and recall. Mathematically, it is the harmonic mean of precision and recall, which is computed as

$$F1 - score = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right) \tag{4.7}$$

The area under the curve (AUC) computes the area under the receiver operating characteristics (ROC) curve which is plotted based on the trade-off between the true positive rate on the y-axis and the false positive rate on the x-axis across different thresholds [50]. Mathematically, AUC is computed as

$$AUC_{ROC} = \int_0^1 \frac{TP}{TP + FN} d \frac{FP}{TN + FP} \tag{4.8}$$

In the case of multi-class and unbalanced test data classification, the performance of models is usually evaluated using weighted and micro-averaging of recall, precision, and F1-score. We have employed weighted averaging which in simple terms is the weighted mean of recall, precision, and F1-scores with weights equal to class probability while summing up the individual recall, precision, or F1-scores in each class respectively, as shown in Eq (4.9)–(4.11).

$$R_{weighted} = \sum_{i=1}^{m} \frac{n_i}{\sum_{j=1}^{m} n_j} R_i \tag{4.9}$$

$$P_{weighted} = \sum_{i=1}^{m} \frac{n_i}{\sum_{j=1}^{m} n_j} P_i \tag{4.10}$$

$$F_{weighted} = \sum_{i=1}^{m} \frac{n_i}{\sum_{j=1}^{m} n_j} F_i \tag{4.11}$$

4.6 Experimental Results

We evaluated our proposed model on ransomware dataset summarized in Table 4.1, and benchmarked our model with both the lightweight deep learning models (e.g., deep neural network (DNN) and recurrent neural network (RNN)) and the heavyweight deep learning models (e.g., InceptionV3, Resnet50, and VGG16) used widely in malware classification tasks that are trained with the limited number of samples.

In order to have an unbiased comparison, all the models are trained under bootstrap sampling with 30 repetitions using 80% ransomware data and the remaining 20% is used for evaluating their performances.

Table 4.4 shows a performance comparison between our proposed approach and other state-of-the-art approaches used as a baseline to benchmark our results. All models in Table 4.4 were trained with entropy features generated from binary executables.

As it can be seen in Table 4.4, our approach exceeds DNN by14.5%, RNN by13.3%, VGG16 by 14.1%, Inception V3 by 13.9%, and Xception by 15.3% in terms of weighted average recall metric respectively. In all the metrics our approach outperformed in comparison to other baseline models.

Table 4.4: performance of different models

Ref.	Method	Recall	Precision	F1-score	Auc-Roc
[12]	DNN	74.2	76.6	75.3	95.8
[12]	RNN	75.4	77.8	76.1	96.1
[175]	VGG16	74.6	77.1	75.2	94.2
[23]	Inception V3	74.8	78.6	76.3	96.1
[91]	Xception	73.4	77.8	76.5	96.3
	Our model (entropy features)	88.7	85.3	86.2	98.2
	Our model (grayscale images)	82.9	80.3	81.8	94.5

Figure 4.5 shows performance variations in terms of the F1-score metric for different models under 30 repetitions. We can see that 80% of the training data size performance variation (with 20% test set) is very minimal. We were able to achieve a mean F1-score of more than 88% with the lowest at 86% and the highest around 94%. Other models in some repetitions are able to achieve above 80% weighted F1-score but the average is around 75%.

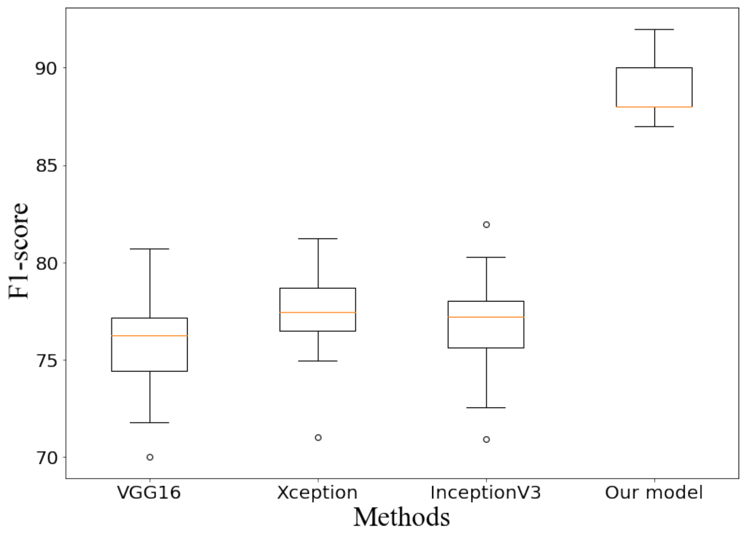


Figure 4.5: Boxplot showing performance variation of different methods under bootstrap sampling with 80% training and 20% test data

We also compared computation cost in terms of training time for each model based on entropy features in Table 4.4. Figure 4.6 shows the training time of our proposed model compared to all the other similar models. The training time for our proposed model is higher because our model uses more trainable parameters.

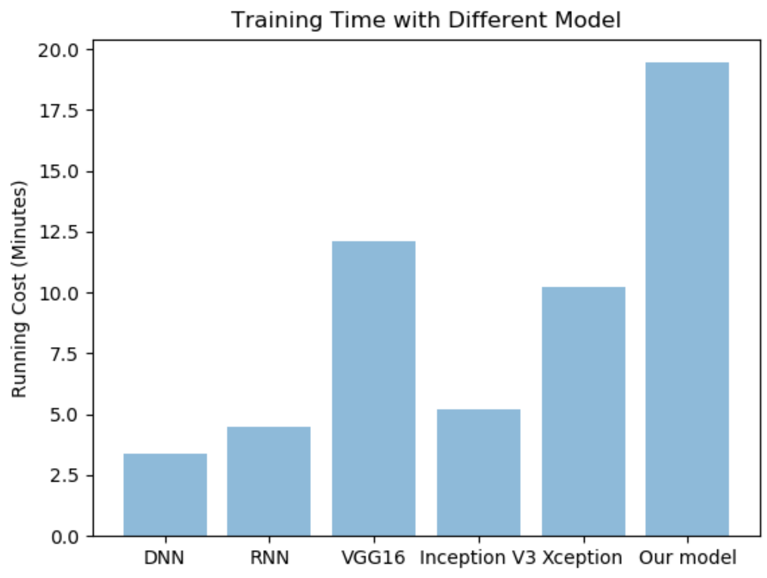


Figure 4.6: The training time compared with DNN, RNN, VGG16(Fine-tune), Inception V3(Fine-tune), Xception(Fine-tune)

Table 4.5 lists hyper-parameter settings for baseline and our proposed approach. For instance, all the models were trained with 50 epochs.

Table 4.5: Hyper-Parameter setting for different model

Training Parameters	Other models	Our model
Epoch Number	50	50
Batch Size	32	24
Optimizer	Adam	Adam
Learning rate	1e-4	1e-4
Train/Test	80%/20%	80%/20%
Re-scaling	1/255	1/255
Augmentation	for 3 classes	for 3 classes
Loss function	Crossentropy	Crossentropy Center loss

As Fig 4.7 illustrates, both train and validation loss stabilizes during 50 epochs confirming the training is done by this stage. As we can see by classifying the input samples into 11 ransomware classes, our model has achieved an F1-score of more than 86%. As observed from Table 4.1, the sample number of dalexis, petya, and uptatre is less compared with other classes, which results in an imbalanced problem in the training stage. To solve this issue, we employ the data augmentation technique [191] for these 3 classes and achieved better results.

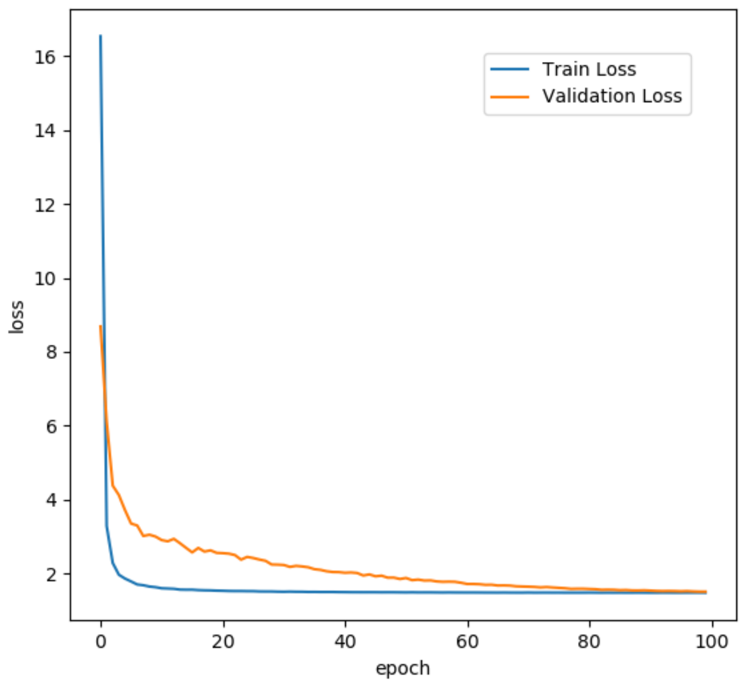


Figure 4.7: Loss during Training on Epochs

Fig 4.8 shows the area under the curve for different ransomware classes in receiver operating characteristics (ROC). The micro and macro average of all the classes is around 0.98 which indicates our model has good separability between different classes. Furthermore, we can see different clusters formed under our model after training.

Fig 4.9 shows different clusters of ransomware classes in our data that are visualized using the t-sne dimensionality reduction technique. Firstly, the t-sne plot in Fig 4.9 indicates that entropy is a very informative feature as we can see different clusters of ransomware attacks at the beginning of our model's training. Secondly, some of the ransomware attack classes are easily distinguishable even at the first epoch of training such as locky, wannacry, uptatre, zerber and virlock, etc. At the end of the training, we can see our model has learned feature embedding to categorize ransomware samples based on their similarity with the cluster. This can also be seen in our proposed approach's results shown by the confusion matrix for one run in bootstrap sampling in Fig 4.10.

As we can see in Fig 4.10, our model is able to distinguish all the classes of ransomware except Bitman and Teslacrypt. This is because these two classes have very similar signatures that can be seen in the t-sne plot after 50 epochs in Figure 4.9. Both classes clusters are very close to each other with some samples very difficult to differentiate,

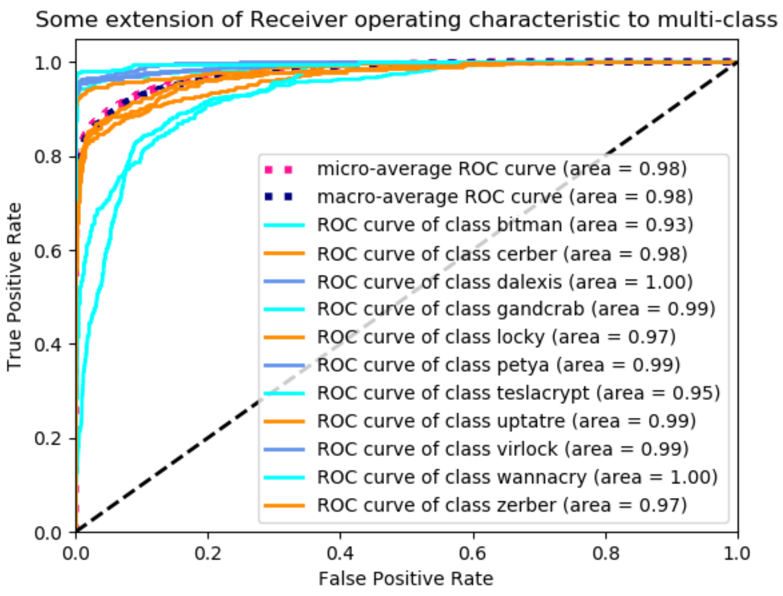
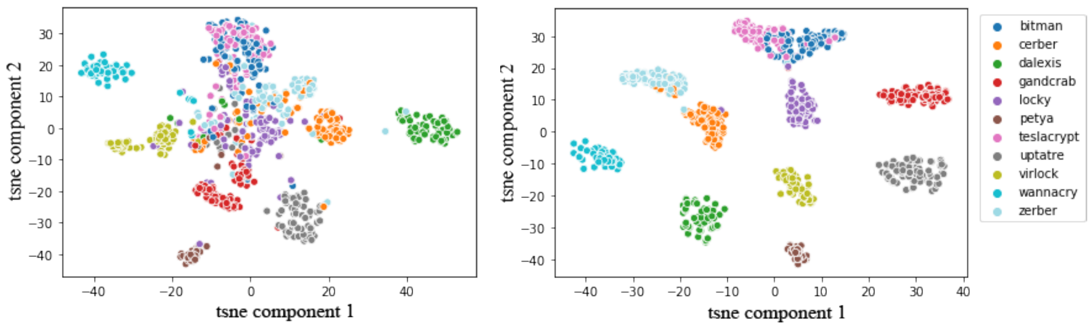


Figure 4.8: Area under curve performance our proposed approach



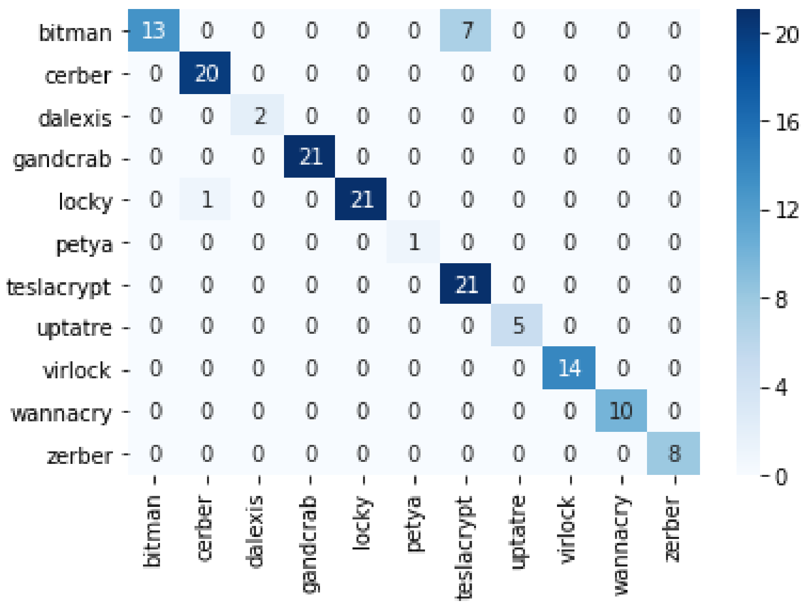
(a) Epoch 1

(b) Epoch 50

Figure 4.9: T-SNE visualization of our data before and after training

that is why our model performed poorly for these two classes. We further conduct another experiment to explore the performance of our model with different ratios of training size, as shown in Figure 4.11.

We systematically reduce the training data size from 90% to 10%. This experiment was repeated 30 times using the bootstrapping technique (sampling with replacement) and the mean F1-score on test data based on training data size is reported using Figure 4.11. As we can see, our model is able to achieve a weighted F1-score greater than 80% at just 50% training size whereas other state-of-the-art methods required a very large amount of data (approx. 90% of data for training) to reach more than 70% weighted F1-score. Finally, from all these experiments we can confirm our model performs well



	bitman	cerber	dalexis	gandcrab	locky	petya	teslacrypt	uptatre	virlock	wannacry	zerber
bitman	13	0	0	0	0	0	7	0	0	0	0
cerber	0	20	0	0	0	0	0	0	0	0	0
dalexis	0	0	2	0	0	0	0	0	0	0	0
gandcrab	0	0	0	21	0	0	0	0	0	0	0
locky	0	1	0	0	21	0	0	0	0	0	0
petya	0	0	0	0	0	1	0	0	0	0	0
teslacrypt	0	0	0	0	0	0	21	0	0	0	0
uptatre	0	0	0	0	0	0	0	5	0	0	0
virlock	0	0	0	0	0	0	0	0	14	0	0
wannacry	0	0	0	0	0	0	0	0	0	10	0
zerber	0	0	0	0	0	0	0	0	0	0	8

Figure 4.10: Confusion Matrix

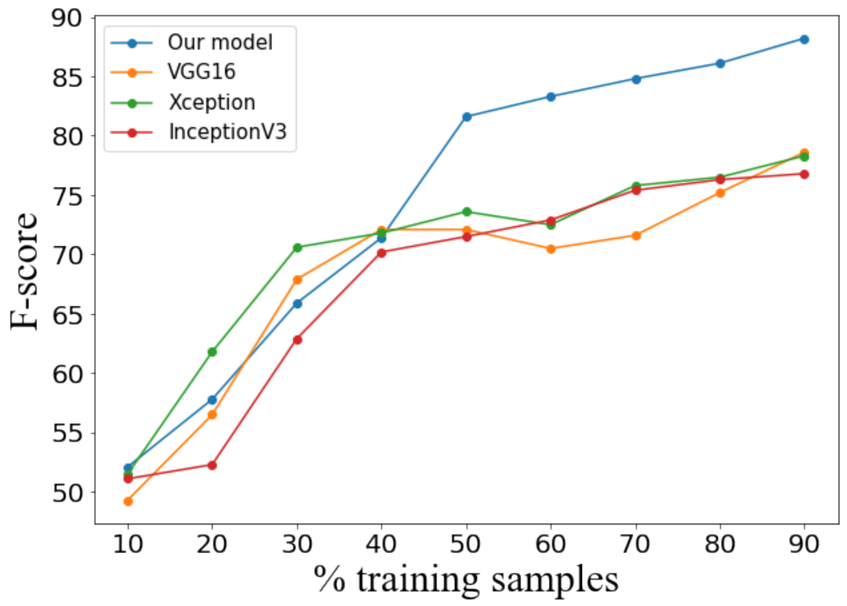


Figure 4.11: Performance with the different ratio of % training samples

under a small amount of training data.

Although the proposed model performs well overall, it has several limitations. First of all, our model is still unable to classify the Bitman ransomware family or the Teslacrypt ransomware family. The reason for this can be seen in Fig 4.9 where after training, the clusters of Bitman and Teslacrypt families still overlap. Without a clear cluster around these two families, our model tends to produce mi-classification results, especially across the malware instances belonging to these two families. Second of all, our proposed model has not been tested for zero-day attacks but only provides results for the classification of known attacks that are included in the training samples. Finally, our model takes a longer training time compared to other methods due to the higher number of parameters that need to be trained.

4.7 Conclusions

We proposed a few-shot learning method based on the Siamese Neural Network model which can rapidly detect and classify different ransomware attack classes even if only a few ransomware samples are available for each class. In our proposed approach, we use entropy features in addition to image features to capture more accurately the unique characteristics associated with each malware signature represented in the feature embedding space of the model. A pre-trained network such as VGG-16 is utilized to generate more accurate weights of different ransomware samples in a meta-learning fashion to avoid the bias typically associated with a model trained with a limited number of training samples. The experimental results tested on the ransomware samples containing 11 different ransomware classes show a very high classification with a weighted F1-score exceeding 86%.

We plan to extend our work to other application domains to evaluate the generalizability and practicability of our proposed solution. These include but are not limited to the rapid classification of other types of malware [98; 162; 167; 97]. We also plan to extend our work to a practical market-scale triage solution to screen different malware rapidly.

Chapter 5

Malware Triage Approach using a Task Memory based on Meta-Transfer Learning Framework

5.1 Summary

Triage is focused on quickly evaluating and prioritizing incidents or alerts, while malware classification is concerned with thoroughly analyzing and categorizing malware samples. Triage is highly time-sensitive, aiming to make quick decisions to address the most critical issues first. Malware classification can be more time-consuming, Triage deals with all kinds of security incidents, not just those involving malware. Malware classification specifically deals with analyzing malicious software. The outcome of triage is a prioritized list of incidents that need attention, while the outcome of malware classification is a detailed understanding of a malware sample's characteristics and behaviour

It is not cost-effective to defend all systems equally in a complex cyber environment. Instead, prioritizing the defence of critical functionality and the most vulnerable systems is desirable. Threat intelligence is crucial for guiding Security Operations Center (SOC) analysts' focus toward specific system activity and provides the primary contextual foundation for interpreting security alerts. This paper explores novel approaches for improving incident response triage operations, including dealing with attacks and zero-day malware. This solution for rapid prioritization of different malware has been raised to formulate fast response plans to minimize socioeconomic damage from the massive growth of malware attacks in recent years, it can also be extended to other incident responses. We proposed a malware triage approach that can rapidly classify and prioritize different malware classes to address this concern. We utilized a pre-trained ResNet18 network based on the Siamese Neural Network (SNN) to reduce the biases in weights and parameters. Furthermore, our approach incorporates external task memory to retain the task information of previously encountered examples. This helps to transfer experience to new samples and reduces computational costs, without requiring backpropagation on external memory. Evaluation results indicate that the classification aspect of our proposed method surpasses other similar classification techniques in terms of performance. This new triage strategy based on task memory with meta-learning evaluates the level of similarity matching across malware classes to identify any risky and unknown malware (e.g., zero-day attacks) so that a defence of those that support critical functionality can be conducted.

5.2 Introduction

In security operations, threat intelligence is crucial in directing SOC analysts toward specific system activity and provides the background context for interpreting security alerts. Lower-threat incidents should be assigned to lower-level analysts to optimize human resources, while senior analysts should focus on more serious threats. Additionally, automated resolution should be the preferred approach for the most dangerous malicious attacks, with manual intervention reserved only for rare cases such as unknown malware. Triage in the context of cybersecurity is a highly time-sensitive process designed for the swift evaluation and prioritization of incidents or alerts. Its primary goal is to make rapid decisions that allow for the immediate addressing of the most urgent issues first. Unlike malware classification, which is more intricate and can be a time-consuming task involving the detailed analysis and categorization of malware samples, triage encompasses a broader range of security incidents, extending beyond just those pertaining to malware. The main objective of triage is to produce a prioritized list of security events that require urgent attention, whereas malware classification aims to achieve a comprehensive understanding of a malware sample's distinct characteristics and behaviours.

According to the report [1] by the European Union Agency for Cybersecurity in 2022, which mentioned that malware and attacks against availability ranked the highest during the reporting period, malware accounted for the rank most elevated threat. Moreover, malware refers to a malicious program designed to block access to a computer system until money is paid, hence the name malware. Compared to other types of malware, the emergence and rapid increase of malware has been due to Bitcoin and encryption technology. The invention of Bitcoin has provided an anonymous payment

[1]https://www.enisa.europa.eu/publications/enisa-threat-landscape-2022

channel for criminals demanding malware. The wide use of solid encryption techniques in many applications also allowed the creation of malware where decryption is impossible without knowing the cryptographic key. The rapid increase in malware has been reported causing the loss of millions of dollars for businesses and individuals. Primarily, resourceful threat actors have utilized 0-day exploits to achieve their operational and strategic goals. The more organizations increase the maturity of their defences and cybersecurity programs, the more they increase the cost for adversaries, driving them to develop and/or by zero-day exploits since in-depth defence strategies reduce the availability of exploitable vulnerabilities. However, statistic-based methods such as clustering [187], entropy analysis [97; 98], similarity analysis [64], information flow analysis [100], and examining manifest file [118] have been proposed to recognize malware quickly. However, the heavy reliance on manual analysis and tool support became either too expensive or impossible due to the growing volume of malware attacks to millions within a brief period.

The proliferation of machine learning techniques has allowed the reduction of manual intervention and has offered more automation-based machine analysis to rapidly recognize different types of malware (and other malware) to reduce a significantly increasing loss of money and productivity [191; 161; 167; 90]. Semi-automated approaches using random forest [85; 84] were proposed to detect malware rapidly. For example, a KNN-based [76] and decision tree-based [7] classifiers to triage a large number of malware samples rapidly have been proposed in recent years.

However, many hurdles remain in providing effective machine learning-based triage solutions. One big obstacle to developing machine learning-based triage solutions is the availability of malware samples to train machine learning models to learn about the features involved in malware codes. Unfortunately, current machine learning techniques demand an extensive dataset (e.g., hundreds and thousands) to train a machine learning technique to recognize important/relevant features, detect different malware signatures, or find correlations across various elements. In this paper, we quantize the threat rate according to the Euclidean distance through two points in the feature embedding, where feature embedding is created by the concatenation of features optimized through the external memory and episode training. With the external memory and the episode training, which aims to provide serves as an auxiliary storage where the model can store and retrieve information that is not immediately present in its parameters while preventing the information from being forgotten during the episode training. We use “classification” to classify malware samples into specific known profiles of existing malware classes. Extending from the classification, we use the word “triage” to refer to the overall assessment process, classifying malware samples into known malware classes and identifying new types of malware samples, for example, risky and unknown malware.

The other obstacle is feature representation when the quality of parts feeding to machine learning models heavily decides the quality of outcomes (e.g., detection accuracy). In many cases, malware authors can still easily avoid detection by applying obfuscation techniques to change feature representations easily even after a machine learning model is trained with a set feature representation. To address these issues, we proposed a novel malware triage approach, and the main contributions of our proposed approach follow.

- We have created a novel meta-transfer learning framework for an incident response that leverages external information to optimize the learning process, enhancing the overall efficiency of the triage system. Our work marks the first attempt to utilize task-driven meta-transfer learning for malware triage, demonstrating the potential of this approach to strengthen cybersecurity operations.
- We employed a first-order approach in Model-Agnostic Meta-Learning (MAML) [42] along with fine-tuning ResNet18 to write the support items to the memory module. This simplifies and accelerates episode training without the need for backpropagation.
- Our method employed a weighted ratio that relies on similarity scores to triage malware classes that are considered risky or unknown, such as those that exhibit features typical of malware but do not match known malware profiles exactly. This prioritizes potentially zero-day attacks for further analysis, enabling the development of appropriate response strategies.
- We conducted extensive experiments on two public datasets for few-shot learning to demonstrate that our method can effectively leverage limited data and achieve competitive results compared with the benchmark performance.

The rest of the paper is organized as follows. Section 5.3 presents related works. Section 5.4 provides the details of the proposed approach. Section 5.5 describes the experiments' details and discusses the proposed approach's results, discussion, and limitations. Finally, Section 5.7 draws a conclusion and describes the planned future work.

5.3 Related work

5.3.1 Triage Approach

Triage aims to assess the severity of the malware based on its impact on the victim. The literature offers a variety of methods to speed up the severity of classification. The aim is to automatically investigate malware samples and pass them to the proper channels (e.g., cybersecurity professionals) for further analysis. A semi-automatic malware analysis architecture proposed in [85] replaces multi-class classification with a group of one-class classifiers to decrease the runtime. A random forest classifier is used in [84] with static malware features. Static features can be extracted fast without running the malware code and help perform quick triage. However, a weakness random forest algorithm as a classifier, this approach tends to overfit when there is malware noise (e.g., obfuscated code) intentionally introduced by malware authors [6].

BitShred [64] is a system developed for large-scale malware clustering and similarity analysis. BitShred uses feature hashing to reduce feature space dimensionality and uncover family relations by investigating correlated features with Jaccard similarity. SigMal [76] extracts features from the malware executable headers. SigMal transforms these features into a digital image. Signatures are extracted from images and investigated with similarity measures using the KNN algorithm. However, the KNN algorithm is susceptible to class imbalance, as pointed out by [83]. TriDroid [7] extracts coarse-grained features (e.g., permissions) to allocate an investigated Android app into a three-class queue and uses fine-grained static features with three-class classifiers to confirm the queue assignments with high accuracy.

TRIFLOW [100] uses information flow in Android apps to characterize behaviours of the risky apps for triage purposes. DROIDSEARCH [118] is a search engine with triage capability that triages based upon information such as ratings, downloads, or information from the manifest file. Mobile Application Security Triage (MAST) [21] uses statistical methods called Multiple Correspondence Analysis (MCA), to find correlations in qualitative data obtained from the market. Rather than inspecting malware itself, tweet messages are investigated in [114] using Deep Neural Networks (DNN) for malware severity classification.

In the previous malware triage work, the proposed models have not achieved either satisfactory accuracy or were extensively tested on different attack classes. This leaves the triage system vulnerable only to classifying known malware samples with a high false-positive rate but also unable to identify unknown samples. The significance of the triage system is to support the automated operation of the system with less human intervention or cyber security analysts to recognize the urgency of the malicious attack to formulate more informed response strategies. Therefore, it is essential for a triage system to be equipped not only to accurately classify them, but in some special cases, it identify risky unknown malware for further analysis.

5.3.2 Transfer-Learning based meta Learning

The training strategy based on Transfer-learning differs from the episodic training strategy in meta-learning. Instead, the conventional techniques are able to be applied in a pre-train model with a large amount of data from the base classes. And then, the pre-trained model is adapted to recognize the base classes and novel classes. [174; 143; 14; 142; 135]. Yuan et al.[174] proposed an offline adaptive learning algorithm that is able to be learned through the meta-training part and the fine-tuning part. The meta-training part aims to optimize the task procedure, and fine-tuning part adjusts the pre-trained parameters with the limited data in the new application task. Sun et al. [143] learned a base learner that could be adjusted to the new task with a few labeled samples. In order to improve the efficacy of performance, they further introduced the hard task meta-batch scheme as a learning curriculum. Soh et al. [135] exploit both external and internal instances to present a Meta-Transfer Learning for the zero-shot task.

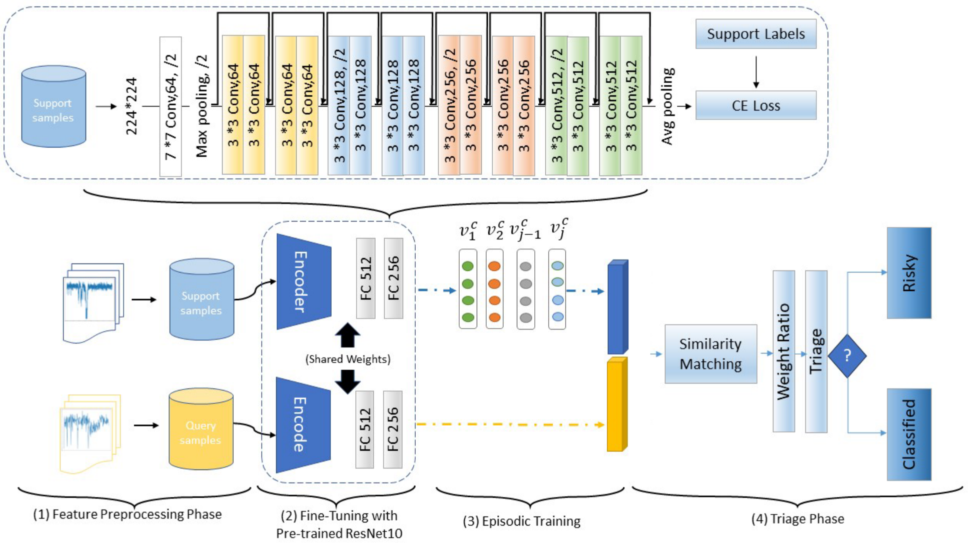


Figure 5.1: Episodic Training for our Triage System

5.4 Methodology

The overview of our proposed model is shown in Fig. 5.1. Our model consists of three phases, (1) feature preprocessing phase, (2) meta learning phase, and (3) triage phase, respectively. The main goal of the feature preprocessing phase is to obtain entropy features from the bytecode of malware and preprocess them to feed to the model of Resnet

18 for fine-tuning. In the meta-learning phase, our model constructs the feature embedding space of the input by utilizing two identical ResNet18 networks. We load a pre-trained version of the ResNet18 network trained on more than a million images from the ImageNet database [2]. This pre-trained ResNet18 is used in a meta-learning fashion, where the memory features [18] are used for the feature generalization ability in the context of malware detection [71; 111]. The triage phase provides an operation service that classifies different malware classes and identifies risky and unknown malware based on the weight ratio.

5.4.1 Problem Formulation

We proceed to introduce the problem setup and notations of meta-transfer learning, which is a novel learning method that helps deep neural networks converge faster while reducing the probability of overfitting when using a few labelled training data only. To train a meta-learning model, which can be divided into few-shot learning or zero-shot learning. In this paper, we construct the few-shot tasks based on the episode training, which is defined as an N-way K-shot classification problem. In few-shot classification, a small support set S and a query set Q are associated with the Meta-training and Meta-testing phase. During the optimizing stage with the loss function, a parameter θ is learned from episode training data . The number of episodic optimizations equals the number of updating times. During the testing stage, an optimized meta-learning space measures the correlation between unseen samples in the support S and query set Q.

5.4.2 Feature Preprocessing Phase

Entropy is often used as the measure of changes in information content. More changes to the original information content produce higher entropy values, while fewer changes to the original information are associated with lower entropy values. As discussed in [190; 188], using entropy values as feature representation has several advantages compared to using grayscale image features. The feature values associated with grayscale features can be easily fooled if malware authors apply certain obfuscation techniques. For instance, control flow obfuscation techniques [156; 8; 27] can easily alter the control flow path of a malware program. For example, it inserts a junk code or shuffles the order of function calls which does not affect the semantics of the original malware program. This simple technique can effectively cause the appearance of grayscale images to be different from each other. This can easily lead to misclassification results where a classification algorithm puts them into different malware families when in

[2]ImageNet. http://www.image-net.org

fact, they belong to the same malware family. In addition, the computation cost associated with grayscale image features is usually much higher due to the higher cost involved in processing texture features of grayscale images [190; 188]. To avoid these weaknesses associated with grayscale features, we instead use entropy features that are more resilient to changes in the malware binary code and reduce computation costs.

We first read a malware binary file as a stream of bytes to construct entropy features. The bytes are made into multiple segments, each comprising 200 bytes. We further counted the frequency of unique byte values in the range of pixel values between 0 and 255, followed by computing the entropy using Shannon's formula as seen in the following Eq. (5.1).

$$H(p) = -\sum_{i} p_i log p_i \tag{5.1}$$

where p_i is the probability of an occurrence of a byte value. The entropy obtains the minimum value of 0 when all the byte values in a binary file are the same, while the maximum entropy value of 8 is obtained when all the byte values are different. The entropy values are then concatenated as a stream of values and are used to generate an image-based entropy graph. The entropy graph, an image of size 224*224, is fed as entropy features as input to our model. Generating the entropy features from a malware binary file is depicted in Algorithm 6.

Algorithm 6: Generating Entropy Features

```
Input  : $f$: malware binary file; $l$: segment length; $n$: the number of files
Output: entropy pattern image $m$
while not reach to $n$ do
    1. read $l$ bytes from $f$, and define as a segment $s_i$;
    2. for $j = 0$ to $255$ do
        2.1 compute the probability $p_j$ of $j$ appearing in $s$ using the Shannon
        entropy function
    3. an entropy graph image $m$ generated from whole segments in a binary
    file
```

5.4.3 Resnet 18 with Fine Tuning for Feature Extraction

Figure. 5.1 shows a typical parameter-level fine-tune (FT) operation, which is in the meta-optimization phase. At the first stage, we fine-tune ResNet18 with the support set to adjust the feature embedding, and then it is prepared to be applied in the backbone network to generate feature embedding for the meta-learning phase (i.e., the entropy feature representing a malware byte code from the same family). The structure of the ResNet18 network follows the paper except for the fully connection layers (FC), in which the first fully connected layer has 512 neurons and the last fully connected layer serves as the output layer and has 256 neurons. The last fully connected layer from each ResNet18 is combined together to create a single fully connected layer with 512 neurons to compute a loss function across the features processed by two ResNet18 networks. The inputs are fed by the entropy graph that is represented as an image form of 2-dimensional vectors of a fixed size 224 $\times$ 224, the fine-tuned ResNet18 is further optimized to obtain updated weights and parameters. The four blocks of each ResNet18 are followed by two fully connected layers with a fattening layer in between.

.

5.4.4 Meta-Transfer learning

The first order for optimization follows the MAML algorithm, which aims [42] to narrow the gap from the distribution of unseen samples. Specifically, the weights of feature extractor Θ are fine-tuned with other datasets, and the original classifier θ is replaced with an FC layer we designed, as the data set for fine-tuning and then optimised them by Adam optimizer as follows:

$$[\Theta;\theta] =: [\Theta;\theta] - a\nabla\mathcal{L}_{\mathcal{D}}([\Theta;\theta]) \tag{5.2}$$

where $\mathcal{L}$ denotes the following empirical loss,

When the fine-tuned model is obtained, we begin the training process for meta transfer-learning outlined with a set of weights optimized for the last Residual Network block. This makes the last residual network block that can easily adapt weights to new examples, thus ensuring effective handling of novel inputs. Despite being trained solely on the malware source domain, the Malimage dataset has been observed to facilitate the acquisition of transferable features during training, as noted in [143]. Moreover, transferring statistics is used in this paper to calibrate the distribution of a limited number of training samples, which can be challenging as the learned model is prone to overfitting due to the biased and limited number of samples. The specific training process is outlined in Algorithm 7.

During the meta-learning phase, the classifier θ will be discarded, because subsequent few-shot tasks contain different classification objectives, such as the N-way classification. To further optimize the meta operations scaling and shifting through meta-batching training. Given a task $\mathcal{T}$, the loss of $\mathcal{L}^{tr}$ is used to optimize the current base-learner (classifier) θ by gradient descent.

$$\theta_i = \theta - \alpha\nabla_\theta\mathcal{L}^{tr}_{\mathcal{T}_i}(\theta) \tag{5.3}$$

where α is the task-level learning rate, the loss of $\mathcal{T}$ is used to optimize the current base-learning (classifier) $\theta^{'}$ by gradient descent:

5.4.5 Task Memory for Adapted Semantic Feature

We make an external memory module [181] that aims to store the previous experience from the fine-tuning stage with true label y_m. Each of the columns represents as average features of one class from other support sets. Memory columns are given by, which could be queried by finding the top k neighbours between a feature and memory columns. Meanwhile, the feature in the support and query set could absorb the task-specific information and therefore able to adopt the current task.

Algorithm 7: Learning in Meta-Training Phase

```
Input  : Restnet18 (Pre) ; Support Set $\mathcal{S}$ and Query Set $\mathcal{Q}$ in Base Set
Output: updated $\theta_i$
Require: Model parameter $\theta_N$ trained for base learner;
Support sample in task $\mathcal{T}_i$;
Query batch in task $\mathcal{T}_i$
while not reach in episodes of $\mathcal{T}_{train}$) do
    Randomly sample task $\mathcal{T}_i$
    $x_s, x_q$ = Feature extractor $\Theta$
    Evaluate $\mathcal{L}_D(([\Theta; \theta^{(meta)}])) = d(h_s, h_q)$ by Eq. 2;
    Optimize the $\Theta$ and $\theta^{(meta)}$ by Eq.10 and Adam
```

Algorithm 8: Prediction in Meta-Testing Phase

```
Input  : Feature Extractor Learned: $\Theta$; Classifier $\theta^{meta}$; A new task Support $\mathcal{S}$
         and Query set $\mathcal{Q}$ in Novel Set
Output: Predicted Label of $N$-way $y_i$
Require: Task Memory: $v_m = \frac{1}{N_c}\sum_{i=1}^{N_c} f(x_i^j; \Theta)$
while not reach to episodes in $\mathcal{T}_{test}$ do
    accuracy = $p(h_{s,q}, v_m; \theta^{(meta)})$
avg_acc = running_acc / test_episode
```

Inspired by the concept we mentioned above, our aim is to create a semantic embedding invariance referring to the prototypical network [134] and in which the feature vector $\mathbf{v_i}$ can be constructed to a memory module called task-memory. We define the entries in $\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, ..., \mathbf{v_k}\}$, in which each entry $\mathbf{v_i}$ is an average feature performance computed via all instances in one class of the support set, and which could be represented as:

$$v_j = \frac{1}{n}\sum_{j=1}^{n_i} f(x_i^j; \Theta) \tag{5.4}$$

where $f(\cdot)$ is a feature embedding constructed with Θ and input feature $\mathbf{x_i^j}$, where $\mathbf{v_j}$ is the memory feature calculated from the support set in the base set and $\mathbf{x_i}$ is an instance of total number n_i in the feature embedding of the base set. Gao et al. [46] argued that a given instance may have a higher variance from the mean feature calculated by all of the rest features or some of them have been wrongly labelled. Moreover, the small number of sample tend to result in a large variance so that a straightly average performance may not estimate a true distribution over all instances. To enhance the ability of feature representation, we make use of the extent of memory to compute the relation between the support and query samples, which correlation score $a_{j,i}$ could be computed based on the softmax function.

$$Sim(f(\mathbf{x_i^j};\Theta),\mathbf{v_j}) = \frac{f(\mathbf{x_i^j};\Theta)\cdot v_j}{||f(\mathbf{x_i^j};\Theta)||||\mathbf{v_j}||} \tag{5.5}$$

which is used to compute a weight, a_j, computed by a softmax function. A memory vector could be retrieved by this weight and weighted average on the task-specific information of the support set and query set according to the following equation:

$$a_j = \frac{exp(Sim(f(\mathbf{x_i^j};\Theta),\mathbf{v_j})}{\sum_{k=1}^{n_i} exp(Sim(f(\mathbf{x_i^j};\Theta),\mathbf{v_k}))} \tag{5.6}$$

where *tanh* is selected as an activation function before the softmax function to produce weights a_j for the inner product of the two feature vectors of instances. Then, we aggregate the sequentially arriving semantic feature $\mathbf{x_i^j}$ regardless of whose gradient and then average all the instances embedded in the support set for each relation. After reshaping the feature maps, we feed them into the memory module to obtain the corresponding feature vectors. To determine the memory size m, we set it equal to the size of the support set, and which feature vectors contained with task information could be represented as with corresponding weights:

$$\mathbf{v_m} = \sum_{j=1}^{n_i} a_j \mathbf{v_j^i} \tag{5.7}$$

where $\mathbf{v_m}$ is the adaptive prototype for a class i and a_j indicates the class weights on each memory cluster. so that a high similar score given by the relation is obtained of higher weights to the instance in the support set.

$$\mathbf{h} = \tau\mathbf{v_m} + (1-\tau)f(f(\mathbf{x_i^j};\Theta)) \tag{5.8}$$

where τ is the hyper-parameter determined by cross-validation. To construct training episodes, a random selection of classes is made from the training set, followed by selecting a subset of examples within each class to form the support set.

$$p_j^e = \frac{exp(-d(\mathbf{h_{s_i}},\mathbf{h_{q_i}})))}{exp(-d(\sum_{c^i\in V^c} exp(\mathbf{h_{s_i}},\mathbf{h_{q_i}}))} \tag{5.9}$$

where $d(\cdot, \cdot)$ is the Euclidean distance function for relation vectors according to the strategy of Snell et al. [134] and it also generates a distribution over the specific class with the cross entropy loss function.

5.4.6 Triage Phase

Our triage approach is similar to one used in the hospital system. Like patients whose symptoms match the known profiles of illnesses, any (test) malware samples whose features match the known malware classes are classified immediately to formulate a rapid response strategy. In some patient diagnoses whose symptoms do not clearly match one known profile but several profiles of illnesses, they could be sent to a designated specialist for further examinations. Our triage also recognizes this type of malware sample whose feature demonstrates that it is malware but does not match a specific malware family. These cases are classified as risky and unknown malware so that further analysis is carried out rapidly to establish if this is a new variant (e.g., zero-day attack). Towards this approach, our model first uses similarity matching to search if a malware sample under the examination (i.e., test sample) exhibits features similar to any known malware classes. The level of similarity matching to different malware classes is computed as a weight ratio. By evaluating the weight ratio, we can triage a malware sample to if this can be classified as specific malware or should be classified as risky unknown malware.

Similarity Matching

In a general similarity searching approach, a query record is compared against a stored database of records. The main goal here is to retrieve a set of database records that are similar to the query record. For example, if there is a picture of a dog as a query record, a similarity search should give a list of pictures with dogs in them.

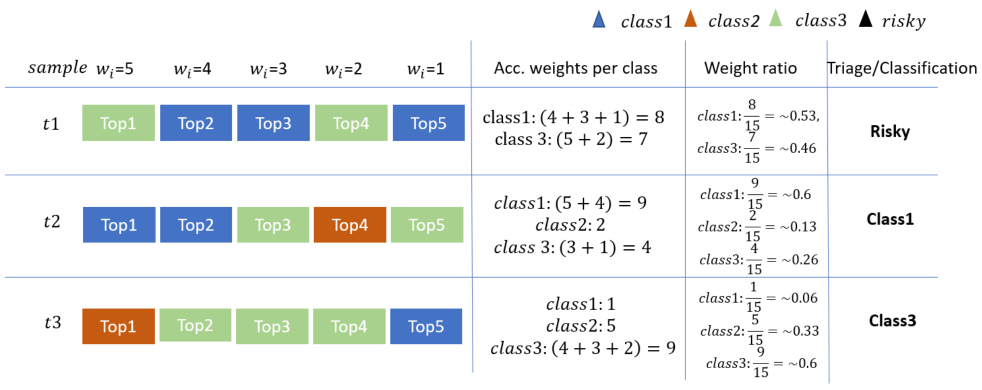
In the context of the triage approach, the database corresponds to a collection of vectors of malware, which is a collection of feature embedding trained for a pair of training samples from a malware class. Similarity searching in our context refers to searching for similar feature embeddings. We employ the FAISS algorithm [3], Facebook's library for faster similarity searching even for very large datasets, to calculate the similarity based on cosine distance. The details of the cosine distance we use are:

In our model, we use the FAISS algorithm [52; 93] to return the top K most similar feature embeddings of malware samples whose cosine distance between the feature embedding of a test sample and the feature embeddings of the training sample are minima. Here, a similarity score indicates a different relationship between two samples

[3]https://github.com/facebookresearch/faiss/blob/main/INSTALL.md

(i.e., a test sample and a class representative from the trained malware classes). The minimum value of 0 indicates two samples are completely dissimilar. The maximum value of 1 indicates two samples are completely similar. The value close to 0 indicates the characteristics of orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.

Weight Ratio and Classification


🟦 *class1* 🟧 *class2* 🟩 *class3* ▲ *risky*

sample	w_i=5	w_i=4	w_i=3	w_i=2	w_i=1	Acc. weights per class	Weight ratio	Triage/Classification
*t*1	🟩 Top1	🟦 Top2	🟦 Top3	🟩 Top4	🟦 Top5	class1: (4 + 3 + 1) = 8 class 3: (5 + 2) = 7	$class1: \frac{8}{15} = \sim 0.53,$ $class3: \frac{7}{15} = \sim 0.46$	**Risky**

*t*2	🟦 Top1	🟦 Top2	🟩 Top3	🟧 Top4	🟩 Top5	*class*1: (5 + 4) = 9
						*class*2: 2
						class 3: (3 + 1) = 4
							$class1: \frac{9}{15} = \sim 0.6$
							$class2: \frac{2}{15} = \sim 0.13$
							$class3: \frac{4}{15} = \sim 0.26$
								Class1

*t*3	🟧 Top1	🟩 Top2	🟩 Top3	🟩 Top4	🟦 Top5	*class*1: 1
						*class*2: 5
						*class*3: (4 + 3 + 2) = 9
							$class1: \frac{1}{15} = \sim 0.06$
							$class2: \frac{5}{15} = \sim 0.33$
							$class3: \frac{9}{15} = \sim 0.6$
								Class3

Figure 5.2: Weight Ratio and Triage

Based on the top K search results and their weights, a weight ratio is computed and the final tirage/classification is done. Mathematically, weight w_i to the ith nearest neighbor for $i = 1, ..., K$ and classifies the test sample x as the class that is assigned the most weight w_i, as follows,

$$W_{nor} = argmax \sum_{i=1}^{K} w_i I_{\{y_i = g\}} \tag{5.10}$$

where w_i denotes the weights and I is the total weights in the same class.

We normalize and regularize the weight ratio in a style that satisfies a linear interpolation with the maximum entropy (LiME) objective by linearly combining each weight [151]. This allows the combined weights to be better balanced and the test sample, x, is best approximated based on the training samples through the Equation 5.11. The LiME objective is described as follows:

$$\begin{aligned} \min_{d} = ||\sum_{i=1}^{k} d_i x_i - f(t)||_2^2 - \lambda ||\sum_{i=1}^{k} d_i||_2^2 \\ subject\ \ to \sum_{i=1}^{k} d_i = 1, d_i \geq 0, i = 1, ..., k \end{aligned} \tag{5.11}$$

where $x_i \to \mathbf{R}$ is ith training sample, the d_i is the basis atoms for the feature vector, x_i, and $f(t) \to \mathbf{R}$ is the feature vector of test sample, t, λ is a regularization parameter.

Fig. 5.2 illustrates how the weight ratios are computed to decide a triage/classification outcome. Let's take a test sample t_1 whose top 5 similar results were from the malware class 3 and malware class 1. The weights from the malware class 3 were 5 (had the highest similarity score based on FAISS similarity matching) in the top 1 position and 2 (had the 2nd lowest similarity score), respectively. This results in the total accumulated weight for the malware class 3 = 7. The weight ratio is calculated by dividing the total accumulated weights for a class by the total weights of search results which results in a weight ratio of approximately 0.53 for class 1 and 0.46 for class 3. Let's presume that the weight ratio rate we want to decide to classify a given sample is set at the threshold = 0.6. Neither the weight ratio of class 1 nor class 3 satisfies this threshold. In this case, our model will put the test sample in the "risky and unknown" class and the sample does not exhibit any clear features that can be classified into a known malware class despite it exhibiting some common features from known malware classes. In contrast, take a test sample t where the weight ratio for class 1 is equal to (or greater than) the threshold. In this case, the t is classified as belonging to the malware class1.

5.5 Experimental Results

This section describes the details of our experiments including the system environment, dataset, and performance metrics we used. We also discuss the experimental results with discussion.

5.5.1 Dataset

Table 5.1: Details of the malware dataset

Class Name	**Instances**	**Ratio** (%)
Bitman	99	9.45
Cerber	91	8.68
Dalexis	9	0.86
Gandcrab	100	9.54
Locky	96	9.16
Petya	6	0.57
Teslacrypt	91	8.68
Upatre	18	1.72
Virlock	162	15.46
Wannacry	178	16.98
Zerber	198	18.89

Table 5.2: Details of the Malimage dataset

Class Name	**Family**	**Instances**
Worm	Allaple.L	1591
Worm	Allaple.A	2949
Worm	Yuner.A	800
PWS	Lolyda.AA 1	213
PWS	Lolyda.AA 2	184
PWS	Lolyda.AA 3	123
Trojan	C2Lop.P	146
Trojan	C2Lop.gen!g	200
Dialer	Instantaccess	431
TDownloader	Swizzot.gen!I	132
TDownloader	Swizzor.gen!E	128
Worm	VB.AT	408
Rogue	Fakerean	381
Trojan	Alueron.gen!J	198
Trojan	Malex.gen!J	136
PWS	Lolyda.AT	159
Dialer	Adialer.C	125
TDownlaoder	Wintrim.BX	97
Dilaer	Dialplatform.B	177
TDownlaoder	Dontovo.A	162
TDownlaoder	Obfuscator.AD	142
Backdoor	Agent.FYI	116
Worm:AutoIT	Autorun.K	106
Backdoor	Rbot!gen	158
Trojan	Skintrim.N	80

We created a dataset containing malware binaries from ViruseShare [4]. The dataset comprises a total number of 1,048 samples from 11 families/classes of malware, each of which consists of a varying number of samples as listed in Table 5.1. The distribution of data in the VUW dataset accurately reflects real-world situations, with some classes, such as Petya and Dalexis, being outnumbered by other classes like Zerber, as indicated in the third column of Table 5.1.

In contrast, the Malimage dataset, as described in [106] and listed in Table II, consists of 9,314 instances from 25 different malware families. The width and height of each malware image differ between families, but Nataraj et al. [106] standardized the width based on the file size and bytecode sequence. The number of samples in the Malimage dataset exceeds that of the VUW dataset. Additionally, the Malimage dataset features distinct image textures among the various malware families.

5.5.2 Experimental Environment

This study was carried out using a 3.6 GHz 8-core Intel Core i7 processor with 32 GB memory on Windows 10 operating system. The proposed approach is developed

[4]VirusShare. https://virusshare.com/

Table 5.3: System configuration

Unit	**Description**
Processor	3.6 GHz 8-core Inter Core i7
RAM	32 GB
GPU	GeForce GTX 2070
Operating System	Windows 10
Packages	Pytorch, Sckit-Learn, Numpy, Pandas

using Python programming language with several statistical and visualization packages such as Sckit-learn, Numpy, Pandas, and Pytorch. Table 5.3 summarizes the system configuration for our environment.

Table 5.4 illustrates the best hyperparameters we used for our proposed model, as well as the 1-shot for 19 batch size; 5-shot for 15 batch size. The classes in Table I exhibit a significant variation, with nearly half of them containing no more than 25 malware samples. In fact, some classes had only one sample, which could be attributed to the recent detection of new malware types such as Blocal and Newbak. To address this issue, we employed a data augmentation technique that increased the image sample size for each malware class to a minimum of 30 samples. This technique involved applying random transformations to the images, such as rotations and re-scaling, with rotations set at angles of 90, 180, and 270 degrees. Additionally, we ensured that the mean value and standard deviation were set to 0.52206 and 0.08426, respectively. In order to evaluate the performance of our models, we conducted measurements of 1-shot and 5-shot accuracy across 2-way and 5-way classifications using random datasets drawn from the complete collection of training and test sets in 20,000 distinct episodes. It should be noted that the size of the malware images remained constant for each model throughout the evaluation process and half for training and the rest half for testing

Table 5.4: Training parameters

Hyperparameters	**Values**	**Descriptions**
Learning rate	0.001	Learning speed (within range 0.0 and 1.0)
Batch size	64	No. of samples in one fwd/bwd pass
Epoch	50	No. of one fwd/bwd pass of all samples
Memory size	k	size of support set

5.5.3 Classification Performance

Vuw Dataset Results Table 5 demonstrates the superior few-shot classification performance of our proposed model with Task-memory and ResNet18(pre), achieving an

Table 5.5: Accuracy scores (%) tested on two dataset

Methods	**2-way 1-shot**	**2-way 5-shot**	**5-way 1-shot**	**5-way 5-shot**
		VUW dataset		
Siamese [134]	71.6 ± 1.6%	73.2 ± 2.0%	60.5 ± 1.4%	63.1 ± 1.9%
Relation [144]	55.8 ± 1.8%	55.2 ± 2.6%	42.6 ± 3.1%	43.2 ± 2.1%
Prototypical (4CONV) [134]	79.3±2.0%	84.2 ± 1.8%	67.8 ± 2.0%	69.2 ± 1.9%
Matching [99]	74.1±2.2%	62.2±1.7%	63.4 ± 2.1%	64.4 ± 2.5%
Ours(Resnet18 (Pre))	83.1±1.5%	86.1 ± 1.7 %	73.5 ± 1.6%	75.8± 1.9%
Ours	84.8 ± 1.8%	87.5 ± 1.8 %	75.7 ± 2.1%	77.3 ± 2.3%
		Malimage dataset		
Siamese [134]	79.6 ± 2.0%	83.0 ± 1.9%	71.3 ± 1.7%	73.2 ± 2.1%
Relation [144]	62.4 ± 2.5%	67.2 ± 2.7%	55.7 ± 3.2%	57.1 ± 2.7%
Prototypical (4CONV) [134]	94.6 ± 1.8%	95.5 ± 1.5%	91.1 ± 2.0%	95.9 ± 1.8%
Matching [99]	86.9 ± 2.1%	88.3 ± 1.8%	80.2 ± 2.4%	81.6 ± 2.2%
Ours (Resnet 18 (Pre))	95.9 ± 2.3%	96.4 ± 1.7%	92.2 ± 1.9%	98.1 ± 1.9%
Ours	96.8 ± 1.6%	97.2 ± 1.8%	94.3 ± 2.2%	98.7 ± 1.7%

accuracy of 84.8% for (2-way, 1-shot). Additionally, our model's accuracy of 87.5% for (2-way, 5-shot) is comparable to the benchmark results of 79.3% and 84.2% reported by Prototypical [134]. However, it is worth noting that Prototypical's use of ResNet18 without task memory resulted in a 1.7% and 1.4% decrease compared to our model in 2-way, respectively. Furthermore, it is important to highlight that models utilizing ResNet18(pre) outperformed those using 4CONV models by a significant margin. This is evidenced by the fact that our best result is 10.8% higher than the best 1-shot result of 5.5% reported in [134] using 4 CONV models.

Malimage Dataset Results The proposed method achieves the highest accuracy, with 94.3% and 98.7% on 5-way 1-shot and 5-way 5-shot, respectively, as shown in Table 5. This represents a significant improvement over previous methods. Our method's success is largely attributed to the novel feature extractor, which generates task-memory feature maps that adopt the previous information expression.

Our approach stands out compared to benchmarks such as Matching and Prototypical Nets, which typically use shallow networks to extract image features. In addition, unlike typical triage methods [85], our approach employs meta-transfer learning, resulting in significantly higher accuracy rates for 5-way 1-shot and 5-way 5-shot classification. Our feature extraction network also outperforms others, with the best-performing result highlighted. By combining meta-transfer learning and task memory, our model can learn a more practical feature embedding and achieve greater convergence stability, leading to even higher accuracies, particularly in 2-way and 5-way classification tasks.

5.6 Triage Analysis

we could utilize the similarity compared with other features of samples from other families and put the unknown malware into the risky pool. In other words, this can classify the unknown and known samples with rates into the risky pool. We further examined the sensitivity of the threshold used for classification and accuracy, which is shown in Table 5.6. This is based on the similarity score obtained by running the KNN algorithm used in the FAISS similarity search with k size = 20. Here the threshold is the value associated with the normalized and regulized weight ratio.

We take the weight ratio with higher confidence into account. Concretely speaking, when we used the highest threshold = 0.50, slightly more than half of the test samples = 57.8%, were fully classified into known malware classes while slightly less than half were classified as risky unknown malware. With such a high threshold rate, the accuracy rate was also very high because at this stage the similar score contributing to

Table 5.6: VUW Dataset Triage vs Accuracy with Threshold

Threshold	Classified	Risk Pool	Accuracy
$W_{nor} = 0.50$	57.8%	42.2%	93.0 %
$W_{nor} = 0.45$	59.2%	40.8 %	90.9%
$W_{nor} = 0.40$	61.2%	38.8 %	78.4%
$W_{nor} = 0.30$	85.1%	14.9 %	65.5%

weight ratio calculation has to be very high (i.e., two samples being compared need to have a very high correlation). As expected, as the threshold value decreased, more test samples were classified into known malware classes as it require a lower level of similarity score computed between a test sample and one classified during training. The accuracy rate dropped orthogonally compared to the decrease of the threshold value. At the lowest threshold value = 0.30, 85.1 % of the test samples were classified into known malware classes while 14.9% of the test samples went into the risky and unknown malware class. The lowest accuracy rate of 65.5% was achieved at the lowest threshold rate.

5.6.1 Discussion and Limitations

In this study, we proposed a model that can be used as a triage application based on the classification of malware. Instead of using image features, we utilize entropy features that are more robust against noises (e.g., changes made by obfuscation technique) and less computationally complex. We use pre-trained two ResNet18 networks in a meta-learning fashion to obtain accurate weights and other parameters when our training sample size is limited. Instead of a feature embedding created based on a single input, our model constructs a feature embedding based on two positive inputs from the twin ResNet18. This results in our feature embedding containing relevant features to detect a certain malware class. Note that we only use positive inputs (i.e., two samples from the same malware family) to train our model to find more common features exhibited within the same malware family. This approach is different from other proposals where negative inputs (i.e., two samples from different malware families) are used [25; 160; 126]. Our experimental results confirm that the use of positive inputs contributes to improving detection accuracy compared to the models using a mix of negative inputs. The triage part of our model utilizes weight ratio to compute a better classification result by taking into account the similarity scores produced by the feature vectors of test samples compared to the feature vectors of the whole training samples. This allows our triage system to classify known malware classes with high accuracy while being able to classify risky unknown malware. These risky unknown malware samples can be given a high priority for further analysis.

At the moment, our model simply classifies any samples that cannot be mapped into any known malware classes as the risky and unknown malware class as these samples exhibit the features of malware. However, it may require further deep analysis to investigate the true nature and severity of these samples (e.g., these may well be zero-day attacks or maybe disguised through more complex obfuscation techniques based on the known malware classes). We observed that there could be a potential bias when the feature vectors from unknown samples are made into feature embeddings trained with known samples. To reduce this bias, we may require further analysis of the nature of unknown samples before feeding them to our model.

Our model classifies any unseen test samples into the risky and unknown pool. However, it is possible that some of these samples may exhibit some features that appear in malware but it can be benign. Our model can be trained with some benign samples to understand the features associated with these types of samples to more clearly classify whether it is completely benign or risky malware. By using entropy features, our model is more resilient to producing misclassification when obfuscated malware is included in training samples. Though the resilience against the control flow obfuscation technique has been evidenced in [188], the influence of other types of obfuscation techniques requires further investigation.

5.7 Conclusion

We proposed a malware triage approach that can rapidly classify different malware classes even in the presence of unknown classes. Our Siamese Neural Network (SNN) based approach utilizes a pre-trained ResNet18 in a meta-learning fashion to generate more accurate feature embedding and overcomes the biases in weight and parameter calculation typically associated with a model trained with training samples. Instead of image features typically used as inputs to machine learning-based malware detection and triage applications, our approach uses entropy features directly obtained from the malware binary files. Our evaluations confirm that the use of entropy features provides a better feature representation and contributes towards improving triage accuracy. The experimental results tested on various malware samples show a very high classification accuracy exceeding 88%. In addition, we offer a new triage strategy that can recognize risky and unknown malware which exhibits the feature commonly seen in other known malware but exact matching profiles cannot be found. These types of malware can be easily prioritized for further analysis to formulate an appropriate response strategy faster before any significant damages emerge (e.g., the loss of ransom payments or reduced productivity).

We plan to extend our work to assign more sophisticated weights for the matching feature vectors for Top-k nearest neighbours as discussed in [37; 89; 170] and also include benign samples to compare their features with existing malware samples.

Chapter 6

Relation-aware based Siamese Denoising Autoencoder for Malware Few-shot Classification

6.1 Summary

Central to the chapter is the proposal of a novel AI-based technique incorporating few-shot learning and a Siamese Neural Network (SNN) with denoising AutoEncoders. This method leverages entropy-based features for an enhanced understanding and detection of malware. The entropy-based features offer a more detailed capture of malware signatures, aiding in distinguishing between various malware classes and uncovering similarities in obfuscated malware.

A key emphasis of the paper is on the non-linear relationships inherent in malware data, which can recognize interactions between different features of a program or a system that might indicate the presence of malware. These interactions can be subtle and not immediately apparent through linear analysis. Non-linear networks can be more robust to noise and irrelevant data, focusing on the most significant indicators of malware. This reduces false positives and improves the overall reliability of the detection system and it is generally better suited to find patterns in such complex datasets compared to linear models.

The integration of AutoEncoders aids in enhancing the effectiveness of the SNN, enabling it to learn robust feature representations for each malware sample. It also plays a vital role in reducing noise and learning data distribution, which is particularly beneficial in countering malware obfuscation techniques.This is instrumental in accurately measuring similarities between different malware samples, crucial for the detection of new or unknown malware variants, including zero-day threats.

6.2 Introduction

The security of nations and the privacy of individuals are highly dependent on the safety and dependability of electronic devices. The biggest risks in this sector often stem from unknown threats, where malware is a major contributor. To address these threats, cyber security professionals must have the ability to accurately and quickly detect potential hazards and identify ongoing attacks. This need has led to the development of advanced deep learning-based detection methods, which are essential for defence in the constantly evolving malware landscape.

Among them, many deep learning methods based on static and dynamic features have been proposed. Motivated by promising results in the use of AI, various feature-based detection models have been proposed, including using malware grayscale images [81] and entropy graphs [190; 20]. Although these static features have improved AI models in the detection of many known malware classes, they are vulnerable to new malware which is without compiled signatures [119]. In addition, these existing methods are also vulnerable to slight changes in malware images, which also results in a decrease in the detection accuracy, most likely due to applying obfuscation techniques. To mitigate the influences of the obfuscation, [122] and [82] proposed enhancing the feature representation methods through an AutoEncoder. But they were still reported to be vulnerable to many types of unseen samples [176; 94].

Due to the cost of dynamic feature analysis, more practitioners have defaulted to static feature analysis as it provides higher levels of efficiency[124]. However, one of the disadvantages of the static feature analysis is that it is susceptible to inaccuracies due to polymorphic and metamorphic obfuscation techniques. Even with the utilization of AI tools in recent years, static feature analysis is still vulnerable to weak representations of embedded spatial features and thus seems to fail to capture core malware signatures [68]. One of the weaknesses of static feature analysis is that a single feature is not always enough to represent the complex interrelationships among malware. The limited abilities of single representations, such as the feature learned from the spatial hierarchies through a back-propagation algorithm, can not completely express the complex relationships across malware data because there is a non-linear relationship across the malicious code [127]. Therefore, a model that can capture the relation of malware through feature embedding is needed. The artificial white noise can also influence the accuracy of a given detection model in a probability distribution of zero value in the grey image, such as no operation (NOP) obfuscation;

Malware developers use obfuscation techniques to reduce the likelihood of detection [192]. To help mitigate this potential threat, we posit that a robust feature relation can greatly help the task of malware detection, as malware functions that exist in a non-relation-aware context operate independently of the relationships or interactions among system components. We set out to examine relation-aware few-shot learning with robust features and to capture the underlying image regularity in malware samples in which a specific pattern links to the corresponding malicious function. By using feature relations [60; 149], we can uncover groups of correlated malware with common features [166; 73; 185]. Sharma et al. [127] found that the underlying physical processes behind converting malware binaries to images are highly non-linear [127], and this correlation exists between malware instances. Other works tend to focus on either the correlation between the malware samples [19] or only on an individual robust feature for the malware samples. In this study, we provide a novel insight into how detection mechanisms are constructed using a relation-aware Siamese embedding in the context of few-shot learning to classify unseen, obfuscated malware samples. The major contributions of this paper are:

- Our proposed solution involves utilizing entropy-based features to train our model, rather than relying solely on traditional malware image features. The use of entropy-based features allows for a more comprehensive capture of the distinctiveness and structural information of the malware. This leads to improved accuracy when identifying different malware signatures and discovering similarities in obfuscated malware. This innovative approach significantly contributes to the effective differentiation of malware classes.
- We propose the implementation of a cutting-edge Siamese Neural Network (SNN) combined with denoising AutoEncoders. The utilization of SNN enables our machine learning model to efficiently classify malware classes, even when limited samples are available. Moreover, the integration of denoising AutoEncoders with a relation-aware module in each branch of the SNN enables us to effectively capture the semantic differences and the complex nonlinear relationships between features, in a pairwise malware comparison.
- Instead of relying on traditional distance scores, our model incorporates relation-aware embeddings to output more precise similarity probabilities between various malware samples. This innovative approach enhances the accuracy of our learning model and enables it to distinguish between different malware signatures more effectively.
- Our proposed model has undergone thorough evaluation and analysis using two substantial sets of malware samples and the N-shot and N-way methods. The results demonstrate that our model is highly efficient in identifying zero-day malware attacks, even those modified by obfuscation techniques.

The next section begins with related work in Section 6.3. Section 6.4 introduces the preliminary materials in the obfuscation technique. Section 6.5 provides the method for constructing our model and the details of our network architecture. In Section 6.6, we provide an ablation experiment to clarify our model's ability and provide multiple analyses for hyperparameters. Finally, this paper concludes in Section 6.7, which provides a summary, conclusion, and potential directions for future work.

6.3 Related Work

Relational exploring for malware detection

RevealDroid [145] simultaneously adopted multiple types of features to train their detection model for Android malware. This method declares that it supports the resistance of four obfuscation techniques: API reflection obfuscation, class name obfuscation, array encryption, and string encryption. However, with the limited capability of the model as trained and tested on the seen classes, it is difficult to apply in a real scenario where unknown attacks are common. Xu et al. [168] explored the structural and semantic relations in Android applications through the entity feature combined with matrices and meta-paths. Moreover, the imbalanced property of malicious behaviour exists universally in this field. Zheng et al. [183] applied a meta-learning approach to a classification task of encrypted traffic and to classify unseen categories based on a few labelled samples. Han et al. [55] analyze the underlying correlation given by the explainable framework between the dynamic and static API call sequences of malware in order to construct the hybrid feature vector space. Nikolopoulos et al. [109] constructed the System-call Dependency Graphs obtained through the dynamic taint analysis over the execution of a program that exploited the valuable structural characteristics of the augmented graphs. Mpanti et al. [103] generated the graph given by degrees and the vertex-weights by utilizing the functionality of system calls to extend the representation of malicious behaviors. Above all, generalization on unseen malicious attacks without depending overly on data size has proven challenging.

Malware classification with the feature extraction

The cybersecurity community has explored automated malware behavior analysis in the last decade. Many detection mechanisms have been proposed to prevent attacks on individual and national data. The cognitive mechanism to understand the malware characteristics in the semantic information involves the dynamic and static features [56; 115; 110; 101; 158], though the hybrid features [148; 33; 171] are also considered in some cases.

N-grams have been used as features in several works and are one of the most common feature types for static analysis. Byte n-grams are particularly attractive since they require no knowledge of the file format and do not require any dynamic analysis. In this manner, one could potentially learn information from both the headers and the binary code sections of an executable [117; 67]. This approach is similar to the research proposed by Kang et al. [70], which captured similar information through the bytecode frequency. Additionally to the use of n-gram features for the static analysis, Nataraj et al. [105] first proposed a technique to extract pixel features from the grayscale images translated from the raw bytecode PE files of malware. Motivated by these grayscale features, Bakour et al. [10] proposed a hybridized ensemble approach using both local and global features as a voter to make a decision in an ensemble voting classifier. Although they take the overfitting problems related to the imbalanced dataset into account, in addition to the small number of malware samples, they ignore the effect on performance caused by the polymorphic obfuscation. In such a situation, the texture of the malware images could be intentionally changed. Without this consideration, performance in existing malware detection research can achieve high accuracy on some datasets, but it cannot effectively detect obfuscated variants of malware, and as such the effectiveness of these methods drops dramatically.

Jeon et al. [66] also proposed a hybrid scheme for malware detection that extracts the static features of the opcode sequence in the static feature classification step. To overcome the shortages caused by static in the obfuscated malware, they also take the dynamic features into account in the next step and execute the files in a nested virtual environment. Although the improved approach enables classifying the obfuscated malware based on hybrid features, most of the existing cybersecurity assessment tools act on real systems, incurring high costs and risk [128; 45] and potentially leading to failure [44] in the virtual environment.

Denoising AutoEncoder for malware detection

One method of unsupervised learning within the research consists of an AutoEncoder architecture [35; 159; 72], which can learn the reconstructed features from noised samples. Alahmadi et al. [4] aim to extract meaningful features with the use of stacked convolutional denoising AutoEncoders (CAE) because the obfuscation and the complex nature of these malicious scripts causes bias in feature selection. Sandra et al. [123] regarded the adversarial examples as noise that assists the malware samples in evading detection based on a deep learning model. To improve malware defenses, they aim to eliminate most noise in malware images through the denoising AutoEncoder. In addition, Salman et al. [122] proposed an unsupervised deep learning-based model based on denoising AutoEncoders that de-anonymizes the mutated traffic to detect a malicious attack using obfuscation techniques which, when applied to the traffic's statistical characteristics, cause a misclassification.

6.4 Obfuscation Techniques

6.4.1 Bytecode-based Junk code for Obfuscation

Junk code obfuscation is a technique which makes executable files impossible to decompile into source code, or impedes the ability to understand a decompiled program, even while the original semantic function is preserved. Jien-Tsai et al. [22] proposed the technique to intentionally introduce syntactic and semantic errors, called junk code, into a decompiled program such that the program would have to be debugged manually, resulting in a bytecode, signature-based anti-virus generating an error or failing in the task of malware detection.

Obfuscation techniques can be applied to benign software to prevent the benign application from being implanted with malicious functions or intentionally modified for other interests. Alternatively, the obfuscation technique can also be exploited into malware to defend against static analysis methods and top-rated anti-malware products. Malware authors employ polymorphic and metamorphic obfuscation techniques, such as No Operation (NOP) insertion, altering the control flow into the source function to evade anti-malware solutions which use the opcode sequence as the malware signature [39; 22]. Under this situation, a variant may be included with an arbitrary number of additions, modifications, or deletions to the code by inserting malicious functions, so as to maintain the original semantic information. Specifically, a malware variant is represented as: $M_{app} = m_1, m_{mod_2}, m_3, m_{add_1}, m_{add_2}, ..., m_n$, where m_{add_1}, m_{add_2} are inserted codes and m_{mod_2} is the corresponding code modification of m_2 while still maintaining the intended functionality. Evolutionary possibilities of malware variants are more pronounced in opcode or hexdump [137]. To visualize this procedure, Fig. 6.1 illustrates the connection between machine instructions, opcodes, and the hexdump. The hexdump is usually a common representation for a disassembled analysis, also known as the static representation of an executable file (or data in general), related to machine instructions or termed opcodes. It is a technique that aims to analyze several types of files, including execution files, shared libraries, object files, etc. The easiest way to obfuscate malware samples is via the bytecode, which is considered as an interpreter executes. It can also be compiled into machine code (opcode) for the target platform. We implemented the operation on the bytecode with the hexdump of malware samples in the datasets to create our synthetic dataset used in this paper.

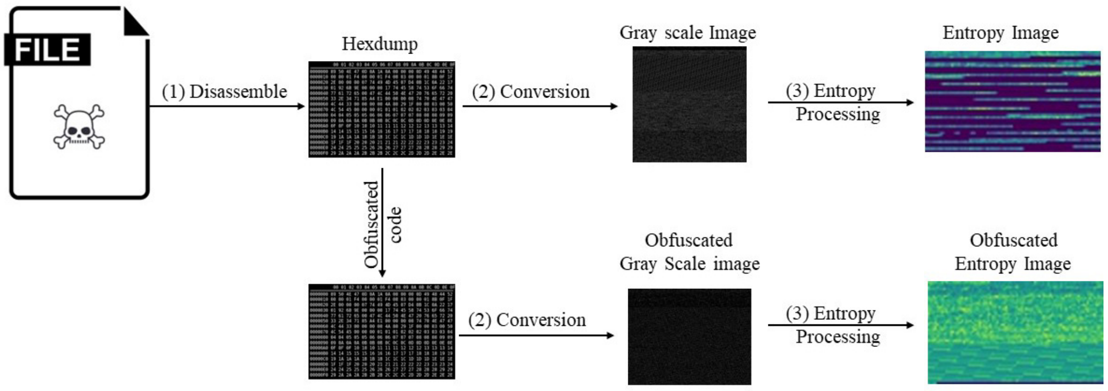


Figure 6.1: Feature Processing for Entropy Image

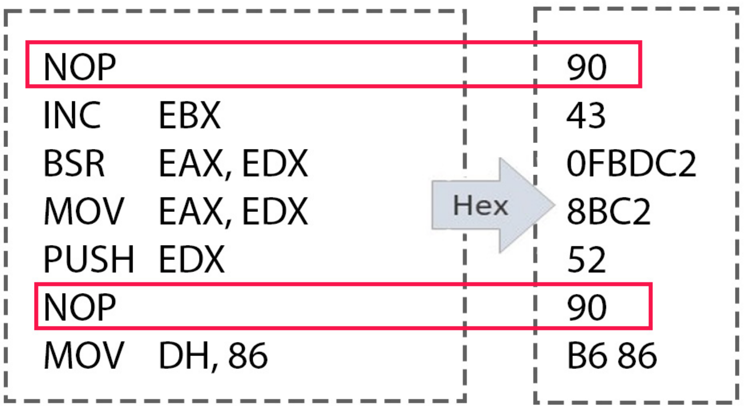


Figure 6.2: Obfuscated Code of No Operation (NOP) Insertion

6.4.2 Obfuscation of No Operation (NOP) Insertion

No-op instruction (NOP) obfuscation, as shown in Fig 6.2, is used to waste the CPU execution cycle, where NOP is an assembly language instruction that does nothing. An obfuscated malware sample synthesizes the original android samples with NOP instruction, which is randomly added into the disassembled methods preserving the semantics [39; 172]. It is possible to evade anti-malware solutions employing opcode sequences as malware signatures, which repeats randomly inserting the bytecode of NOP multiple times at an ambiguous position in the hexdump form of a malware sample.

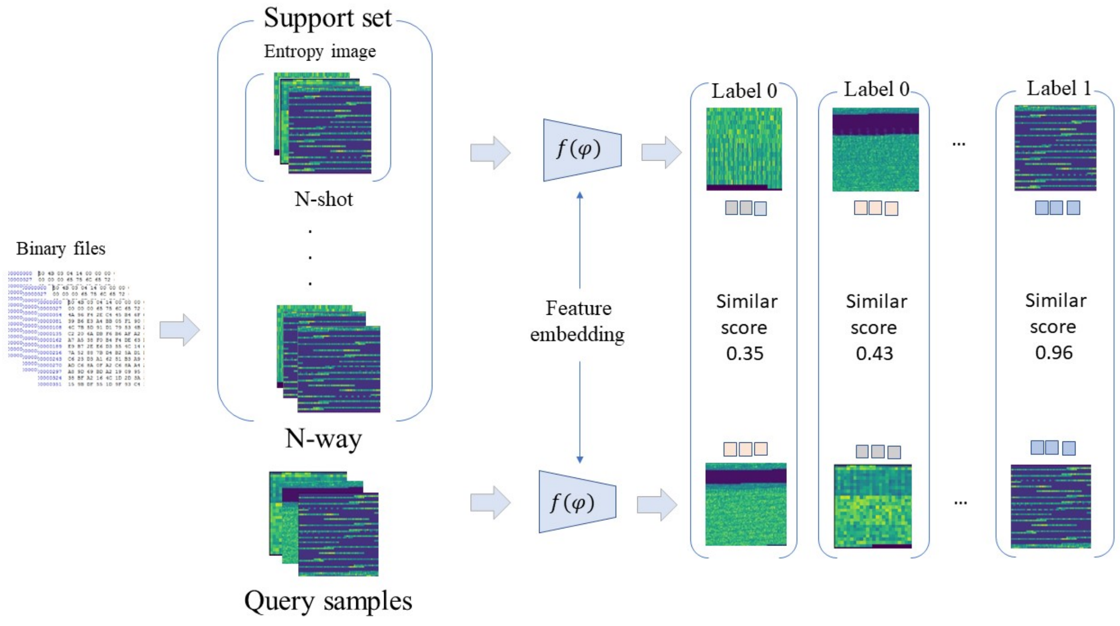


Figure 6.3: The diagram of Few-shot learning

6.5 Methodology

Our approach involves a few-shot classification task using a Siamese denoising AutoEncoder against junk code obfuscation. Fig. 6.3 shows that outlines a machine learning-based malware detection process using few-shot learning. Initially, binary files are converted into entropy images, which are used to create a support set categorized into N-shot and N-way groups for training. Query samples, also in the form of entropy images, are then processed through feature embedding to extract pertinent features. These samples are subsequently classified and assigned similarity scores based on how closely they match the features of the support set, facilitating the identification of potential malware.

The details of the proposed scheme are presented in this section. The evolution of malware samples often has a certain inheritance correlation with the key functions of previous versions, or the key functions are packaged with multiple obfuscation techniques to evade detection. [5; 121]. The core of its detection is to mine such a property that it will be able to detect new versions that exist or evolve in the future. A generic few-shot learning method establishes a metric embedding trained through the support and query samples. In this work, the relationships are established by a Siamese relation-aware feature embedding. With the embedding learned through the fully connected layer (FC) or the convolutional layer (CNN), an input feature can be projected onto the embedding to calculate the relation score between the support sample and the query sample. A relation-aware embedding trained in this manner can predict samples in the untrained class through the query samples.

6.5.1 The Entropy Feature Conversion

Entropy feature conversion aims to measure the uncertainty distribution of the information. Moreover, Vidya et al. [154] argue that the entropy feature has better feature representation since it exhibits higher uniqueness and entropy values are incorporated from local regions to add extra information content to these images. In the context of the malicious code, we could build the malicious pattern with characteristics of entropy information calculated from the binary pattern. In [48; 16], the authors divided bytecodes into blocks of fixed size, and then computed the Shannon entropy for each block. We attempt to further their achievements in image recognition using the entropy value converted from a binary pattern [158]. We assume that the entropy information can be converted to an entropy image. This approach preserves the connected information of each entropy sequence in the context of the raw bytecode data, as Fig. 6.4 shows. The entropy images are comprised of global and local entropy information that can be calculated with the Shannon entropy equation below:

$$Ent = -\sum_{i}\sum_{j} M(i,j) log M(i,j) \tag{6.1}$$

where M is the probability of an occurrence of a byte value. The entropy obtains the minimum value of 0 when all the byte codes of malware have no changes. Alternatively, the maximum entropy value of byte value 8 is obtained when all the byte codes are different. On the contrary, if specific information of bytecode occurs with high probabilities, the entropy value will be smaller.

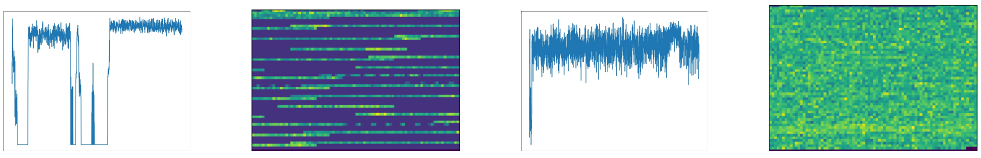

(a) Bitman Entropy Pattern (b) Bitman Entropy Image (c) Upatre Entropy Pattern (d) Upatre Entropy Image

Figure 6.4: Entropy pattern corresponding to entropy image

To visualize the entropy pattern that reflects the pattern of entropy information and the intrinsic connection of each sequence of raw bytecode, we built a grey-scale matrix through Eq. 6.2, and the entropy values are then concatenated into a stream of values that can form an entropy sequence.

$$P_{(i,j)} = 2^{ent} - 1 \tag{6.2}$$

where ent denotes the entropy value of the bytecode block (i, j) and $P_{(i,j)}$ means that the gray pixel value is in the range of [0, 255]. The entropy images are concatenated with the full entropy sequences sequentially to generate an 105×105 entropy image. However, the sizes of bytecodes are different, and we fix the width value to resize the image size to 105×105.

6.5.2 Denoising AutoEncoder

In cybersecurity research, denoising encoders have been widely used in recent years [35; 125; 38]. This can be attributed to the denoising AutoEncoder (DAE) network's aim to reconstruct the data to its original characteristics or its uncorrupted version, without the noise. This noise can be considered to be Gaussian noise, music [182], occluded faces [51], or even junk code.

A typical denoising AutoEncoder can be stacked using multiple convolutional layers consisting of encoders ($E^{\frac{M}{2}}$) and decoders ($D^{\frac{M}{2}}$) in varied depths. Specifically, the first $M/2$ hidden layers encode the input as a new representation, and the last $M/2$ layer decodes the representation in the latent embedding to reconstruct the input. We can define this equation as:

$$z_i^{(m,v)} = a(W_{ae}^{m,v} z_i^{m-1,v} + b_{ae}^{(m,v)}) \tag{6.3}$$

where $z_i^{(m,v)} \in \mathbb{R}$ and $d_{m,v}$ is the number of nodes, $\{W_{ae}^{m,v}, b_{ae}^{(m,v)}\}$ is the parameter set for all layers with $M + 1$ being the number of layers of our network while $a(\cdot)$ is a nonlinear activation function. The feature matrix is $X^{(p)} = [x_1^p, x_2^p, , ..., x_3^p], \in \mathbb{R}^{d_p \times n}$ for one of the sub-networks. Meanwhile, the corresponding reconstructed representation is denoted as:

$$Z^{(M,v)} = [z_1^{(M,v)}, z_2^{(M,v)}, ..., z_3^{(M,v)}] \tag{6.4}$$

where $z_i^{(M,v)}$ is the reconstructed representation for the ith sample in one sub-network. Thus, the low-dimension representation $Z^{(\frac{M}{2},v)}$ is obtained by the following reconstruction loss with mean square error (MSE):

$$\mathcal{L}_{mse} = \frac{1}{2n} \sum_{i=1}^{n} \|X^{(p)} - Z^{(M,v))}\|_F^2 \tag{6.5}$$

where $X^{(p)}$ and $Z^{(M,v))}$ is the noise sample inputted and the original samples, respectively.

6.5.3 Siamese-based Denoising AutoEncoder

Our model contains a Siamese neural network (SNN) that is commonly used in contrastive learning. The SNN model is useful for pairwise identity verification, and it takes as input two images in order to identify a meaningful distance between the representations of those two images. Each of the sub-networks is parameterized by the shared weights and bias $\{W_{ae}^{m,v}, b_{ae}^{(m,v)}\}$, performed on both input images whether or not they are the same.

With the connection part of encoders ($E^{\frac{M}{2}}$) and decoders ($D^{\frac{M}{2}}$), we extracted the features out of the latent embedding optimized by the siamese denoising AutoEncoder. A relation-aware function then calculates the correlation between the two input images. This correlation of the pair of images for the latent features in the last fully connected layers within the encoder ($E^{\frac{M}{2}}$) can be denoted as:

$$d_{\varphi}(z_i, z_j) = \|g_{\phi}(z_i^{(\frac{M}{2},v)}) - g_{\phi}(z_j^{(\frac{M}{2},v)})\|_F^2 \tag{6.6}$$

where notation $d_{\varphi}(z_i, z_j)$ is represented as d_{φ} and d_{φ} denotes whether the z_i and z_j are similar. When this is similar the d_{φ} is close to one, or zero otherwise. The contrastive loss to calculate their relationship is computed by:

$$\mathcal{L}\sum_{i=1}^{p} L(\varphi, (z_i^{(\frac{M}{2},v)}, z_j^{(\frac{M}{2},v)})^i) \tag{6.7}$$

$$L(\varphi) = (1 - y_i)L_s(d_{\varphi}) + y_i L_d(d_{\varphi}) \tag{6.8}$$

The classification ability depends on the performance of d_{φ} calculated by Eq. 6.8, which linearly represents the Euclidean distance.

6.5.4 Siamese in Relation-Aware Embedding for Malware Classification

Strategies such as [152; 80] only take the independent malware signature into account, or rarely consider the non-linear relationship [58; 133] in the contextual difference between all malware pairs. Moreover, malware classification often struggles in data-poor problems where the underlying structure is characterized by organized but complex relations. This is especially true for the interaction of functions of coding between malware samples.

We propose a relation-aware Siamese denoising AutoEncoder that enhances the conventional SNN with relational semantic information. We jointly learn new representations in the non-linear relationships for $\mathcal{C}$ that are assumed to represent the concatenation of feature maps in depth that can explore a learnable rather than fixed metric, or non-linear rather than linear classifier [144; 179; 26].

Modeling the relations between the instance pairs has been shown to improve the results of classification, and these latent features can be explored by this relation module to determine if they are similar using the relation classifier.

$$r_{i,j} = \mathcal{C}(f_\varphi(z_i^{(\frac{M}{2},v))}, f_\varphi(z_j^{(\frac{M}{2},v))}), \quad i = 1, 2, ..., C \tag{6.9}$$

where the projected features, $z_i^{(\frac{M}{2},v)}$, $z_j^{(\frac{M}{2},v))}$ of the support sample and query sample, are aimed to be combined with each other. These are then fed through the sub branch of the relation-aware siamese neural network f_φ.

$$\mathcal{L}_r = \sum_{i=1}^{i}\sum_{j=1}^{j}(r_{i,j} - y_{i,j})^2 \tag{6.10}$$

Given the set of latent features $\mathbf{z} = \{z_1, z_2, ..., z_n\}$ from the latent embedding for a malware sample and $y_{i,j}$ denotes the one-hot encoding for the ground-truth label of malware family. Finally, the objective function for few-shot learning is:

$$\mathcal{L} = \mathcal{L}_r + \lambda\mathcal{L}_{mse} \tag{6.11}$$

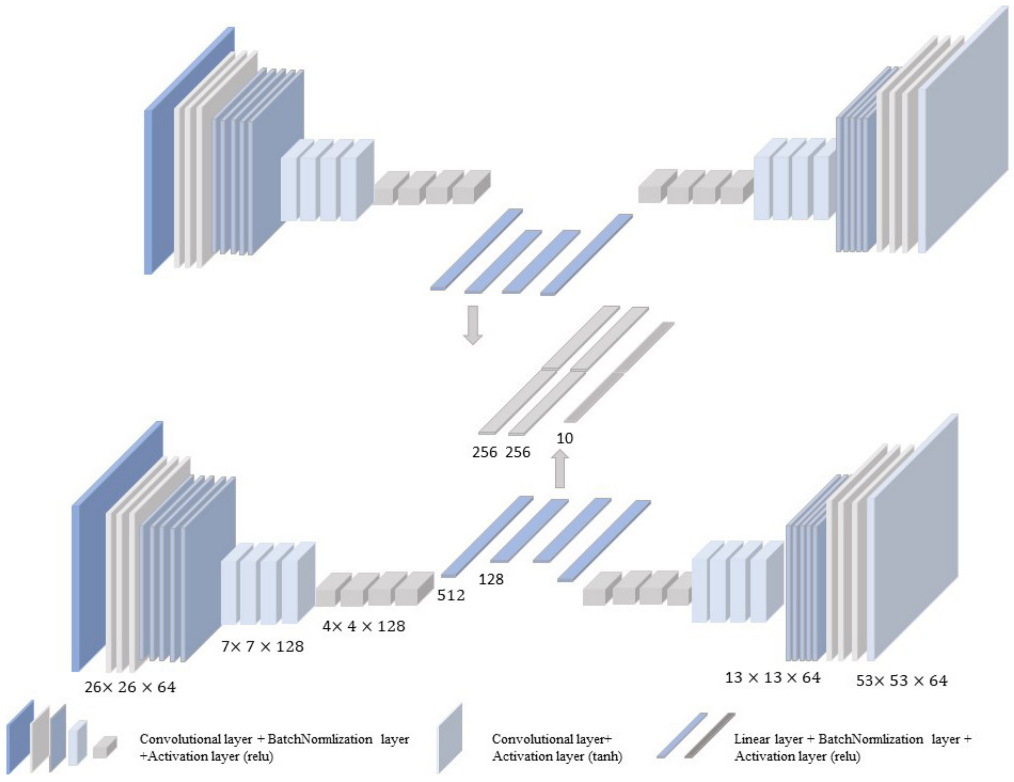
where the value of hyperparameter λ settled as 0.7.

6.5.5 Network Architecture

In this section, we describe the specific parameters in the architecture of our model. First, we apply convolutional layers at the front of the CAE. One advantage of adding a convolutional network is that it is highly invariant to translation, scaling, tilting or other forms of deformation. Furthermore, the fully-connected layers at the front part are one of the major causes of increasing the number of parameters. We put convolutional layers first and stack the fully connected layers on top of the convolutional layers for ease of the reduction to the target dimension d.

The block diagram of the proposed image compression based in Fig. 5 shows the architecture a/1 layouts, stacking up to 4 convolutional layers. The only pre-processing steps before the CAE network consist of normalizing the entropy pattern to $[1, 1]$ by calculating the value of mean and standard deviation in 0.52206 and 0.08426 respectively in the VUW dataset. The size of the input is denoted as H $\times$ W $\times$ C, where C represents the number of colour components. We considered C $=$ 1 for the entropy images since we converted them to gray-scale images.

The analysis transform and synthesis transform have symmetric networks, apart from using convolutional and deconvolutional filters, respectively. In network design, we also consider the term of the receptive field of the topmost feature layer, because if



256 256
10
128
512
4× 4 × 128
7× 7 × 128
26× 26 × 64
13 × 13 × 64
53× 53 × 64
Convolutional layer + BatchNormlization layer +Activation layer (relu)
Convolutional layer+ Activation layer (tanh)
Linear layer + BatchNormlization layer + Activation layer (relu)

Figure 6.5: The diagram of our model architecture

the receptive field is too large, it means that there are too many layers, and the risk of network overfitting will increase. The high-level complex features cannot be learned, resulting in poor network learning ability. The whole model architecture is included in the encoder part, decoder part, and relation part that is calculated for the classification task, as shown in Fig. 6.5. The encoder part is comprised of 4 convolution layers and each layer is followed by a batch normalization layer and Relu activation layer. Specifically, all the convolutional layers (CNN) are comprised of filter size 3 $\times$ 3 and stride 2, the difference is the out channels in 64, 64, 128, and 128 for the encoder part, while the order of CNN layers in the decoder part is opposite to the encoder part. It is noted that a max pool layer is added after the first convolution layer. Then we construct three linear layers to improve the classification performance with the extracted features through the encoder part. The extracted features have a robust ability against the NOP obfuscation as it has been trained with an AutoEncoder module. We further compare the performance with the different dimensions of the linear layer and we claim that the output dimension of 256 for the first linear layer has better performance than the dimension of 128. Finally, the sigmoid activation function is taken as the outcome function as relation label 1 or 0.

Algorithm 9: Pseudocode for Our Proposed Algorithm

```
Input  : Folder_root : f, Class_num : cc, Num_per_class : np, Batch_num : bn,
         obfuscated samples $x_b^1, x_b^2$ and original samples $x_o^1, x_o^2$, Reconstruction
         loss : RC, Relaton loss : RL
Output: Relation score for each pair
Dataloader = FewShotTask(f, cc, np, bn)
for episoded in range(EPISODE)
    $(x_b^1, x_b^2)$ = Dataloader.iter()
    $en_1, en_2$ = Siamese_EncoderNetwork($x_b^1, x_b^2$)
    $dn_1, dn_2$ = Siamese_DecoderNetwork($en_1, en_2$)
     Pairs $(r_n)$ = Feature_concatenation(($en_1$,$en_2$))
    Score $(s_z)$ = Siamese_RelationNetwork(($r_n$))
    $Loss_1$ = RC ([$dn_1, en_1$],[$dn_2, en_2$])
    $Loss_2$ = RL ($s_z$)
    $Loss$ = 0.7*$Loss_1$ + $Loss_2$
    $Loss$.backward()
    Optimizer.step()
```

6.6 Experiments

We evaluated the performance of our proposal model on two malware datasets according to the few-shot learning mechanism. Subsequently, the ablation experiments are commenced to analyze the effect of various parameters, such as learning rate, batch size, layer dimension and hyperparameters. The experiments were conducted on equipment consisting of 32GM RAM, Nvidia GeForce RTX 2070(8GB), and Intel i7-9700 CPU@ 3.00GHz.

6.6.1 Description of dataset

Two datasets were utilized for conducting the experiments:

VUW dataset This dataset was collected from VirusShare[1], which is a repository of malware samples provided to security researchers, incident responders, forensic analysts, and for the curious. With this resource, we created a total number of 1,048 samples from 11 families/classes of ransomware, each of which consists of a varying number of samples, as listed in Table 6.1. This dataset intuitively reflects the distribution of the data in real situations, as some classes, e.g., Petya and Dalexis, are largely outnumbered by the other classes, e.g., Zerber, as shown in the third column of Table 6.1.

[1]VirusShare. https://virusshare.com/

Malimage dataset The details of the second malware dataset, Malimage, published in [105] are listed in Table 6.2. There are 9314 instances spread over 25 families in which the width and height of each malware image differ from other families. Nataraj et al. [105] fixed the width with a certain length according to the file size and the byte-code sequence can be organized in order. Observed in Table 6.2, the number of samples is relatively larger than the VUW dataset. The malware samples have distinctive image textures across the different malware families.

Table 6.1: Details of the VUW ransomware dataset

Family name	Instances	Ratio (%)
Bitman	99	9.45
Cerber	91	8.68
Dalexis	9	0.86
Gandcrab	100	9.54
Locky	96	9.16
Petya	6	0.57
Teslacrypt	91	8.68
Upatre	18	1.72
Virlock	162	15.46
Wannacry	178	16.98
Zerber	198	18.89

Table 6.2: Details of the Malimage dataset

Class name	Family	Instances
Worm	Allaple.L	1591
Worm	Allaple.A	2949
Worm	Yuner.A	800
PWS	Lolyda.AA 1	213
PWS	Lolyda.AA 2	184
PWS	Lolyda.AA 3	123
Trojan	C2Lop.P	146
Trojan	C2Lop.gen!g	200
Dialer	Instantaccess	431
TDownloader	Swizzot.gen!I	132
TDownloader	Swizzor.gen!E	128
Worm	VB.AT	408
Rogue	Fakerean	381
Trojan	Alueron.gen!J	198
Trojan	Malex.gen!J	136
PWS	Lolyda.AT	159
Dialer	Adialer.C	125
TDownlaoder	Wintrim.BX	97
Dilaer	Dialplatform.B	177
TDownlaoder	Dontovo.A	162
TDownlaoder	Obfuscator.AD	142
Backdoor	Agent.FYI	116
Worm:AutoIT	Autorun.K	106
Backdoor	Rbot!gen	158
Trojan	Skintrim.N	80

6.6.2 Dataset setup and augmentation

The number of malware samples from different malware classes in Table 6.1 varied. Almost half of the classes had no more than 25 malware samples, while some only had one, as they were most likely new malware detected recently (e.g., Blocal and Newbak). We increased the image sample size to have at least 30 samples for every malware class using a data argumentation technique (e.g., applying random transformations such as image rotations and re-scaling), in which the rotations are set up at degrees of 90, 180, and 270. At the same time the mean value and standard deviation are set to 0.52206 and 0.08426, as shown in Table 6.3.

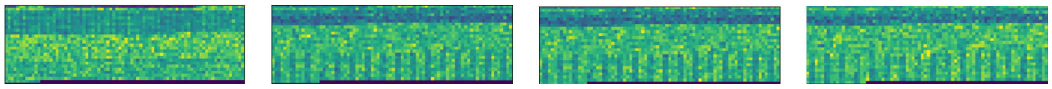

(a) Original Entropy Image (b) Frequency 200 with Nop (c) Frequency 400 with Nop (d) Frequency 600 with Nop

Figure 6.6: The appearance comparison between original images and obfuscated images

Table 6.3: Model learning parameters and their values

Methods	Values	Description
rescale	1./255	Resizing an image by a given scaling factor.
learning rate	1e-02	Epsilon for ZCA whitening.
batch size	19/10-15	1-shot for 19 batch size; 5-shot for 15 batch size
rotation_degree	90/180/270	Setting degree of range for random rotations.
mean	0.52206	the average value in VUW
std	0.08426	the standard deviations in VUW
λ	0.7	the hyperparamter for the loss function

Furthermore, to imitate the obfuscation techniques, we performed NOP insertion on the original image at frequencies 200 times each. Fig. 6.6 shows the visual appearance difference in the original malware image, which also intuitively differs from the obfuscated entropy image. From this change, we can also conclude that static signatures can be easily tampered with intentionally by malware developers. However, the appearance of entropy images at frequencies of 200, 400, and 600 does not show the obviously visual difference between them. Thus, we conducted experiments on the 200-frequency dataset to evaluate the performance of our model.

Compared methods

We use two typical methods as a method of comparison, ProtoNet [134] and RelationNet [144] are known for their typical few-shot learning approaches. We trained them from scratch without obfuscating the dataset. We measured 1-shot and 5-shot accuracies in 2-way and 5-way on random datasets drawn from the total set of training and test sets in each of 20,000 episodes. Correspondingly, the malware image size is fixed for each desired model.

6.6.3 Training Details

We adopt the episode-based training strategy, which takes the support set and query set into account during each training episode. This could be noted as C-way K-shot training. For example, the 5-way 1-shot contains one labelled sample for each unique class in 5-way. For K-shot where $K > 1$, we element-wise average over the embedding module outputs of all samples from each training class to form this class' feature map. And then, the class-level average pooling is concatenated with the query image feature. The number of query images is dependent on the batch size in each training episode. For example, the 2-way 5-shot contains 19 query images, and the 5-way 5-shot has 15 query images. The corresponding relationship between the support samples, query samples and batch size could be expressed by the following equation,

$$\begin{aligned} S =& \{s^{(1)}, ..., s^{(c)}, ..., s^{(C)}\} \subset \mathbb{C}^{train}, s^{(c)} = K \\ Q =& \{q^{(1)}, ..., q^{(c)}, ..., q^{(C)}\} \subset \mathbb{B}^{train}, q^{(c)} = N \end{aligned} \tag{6.12}$$

where c is the class index and K is the number of samples in class $s^{(c)}$; thus the training samples are constructed with NK samples in each episode and corresponds with the N-way K-shot problem. Specifically, when we expand them, we could get the relationship below:

$$\begin{aligned} r_{ij} :& \{(s^{(1)}, q^{(1)}), (s^{(1)}, q^{(1)}), (s^{(n)}, q^{(1)})\}, ... \\ & \{(s^{(1)}, q^{(c)}), (s^{(c)}, q^{(c)}), (s^{(n)}, q^{(c)})\}, ... \\ & \{(s^{(1)}, q^{(n)}), (s^{(c)}, q^{(n)}), (s^{(n)}, q^{(n)})\} \end{aligned} \tag{6.13}$$

observed from this, the $(s^{(1)}, q^{(1)})$, $(s^{(c)}, q^{(c)})$, $(s^{(n)}, q^{(n)})$ are labelled as 1, others are labelled as 0.

The training sets and testing sets are randomly split into the 9/6 classes and 2/5 classes respectively. The Malimages dataset is split into 13 classes for training and 12 classes for testing. We resized the entropy images inputted to 105 $\times$ 105 and the results were conducted randomly 10 times, in one of which the whole model ran 20000 epochs in the training stage and then was tested over 2000 epochs. There are some values preset that were followed by [144], in which the rotation for the sample was set to 90, 180, and 270 degrees respectively. The batch size of 19 and 15 were for the 1-shot and 5-shot respectively. Our model shared the initialized learning rate with a value of 0.02. Meanwhile, the hyperparameter for λ in the loss function is 0.7.

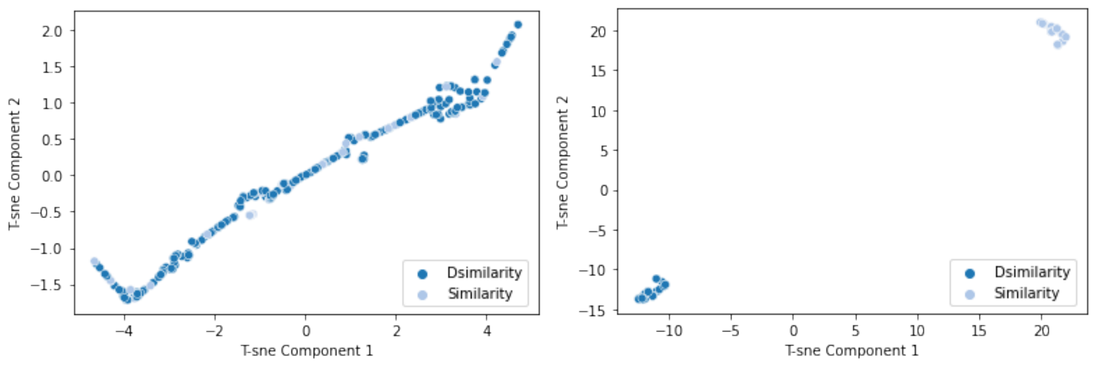
6.6.4 Experimental Results on VUW Dataset

As seen in Table 6.4, we evaluated our model on two different features, the grayscale feature, and the entropy image we developed. Our model achieved the highest accuracy with 83.1% and 81.7% on 2-way 1-shot and 2-way 5-shot, respectively. As a comparison, the prototypical also has greater improvement with the entropy image on the 2-way and 5-way result than with the grayscale feature, but the relation network clearly underperformed. The reason that our model and prototypical had such great improvements in the overall performance is that the representation of entropy images which exhibit higher uniqueness and entropy values is incorporated into local regions with higher information content to these images. It is worth noting that instead of taking the sum pooling for the feature concatenation as the relation network does, we took the mean pooling for the few-shot learning. The details will be discussed in the Section F.

In the following comparison, we only compared our performance with the prototypical network under the obfuscated entropy image, because the learning mechanism of the relation network under-performed in the results. In the obfuscated dataset, our model still showed a significant success on the 2-way results, which at 81.7% and 85.2% are higher than the prototypical network by 1.4% and 0.4%, respectively. However, the result on the 5-way drops significantly due to the large intra-class variance. Therefore, it is more difficult to construct a feature representation distinguished with a high inter-class variance, which also resulted in a greatly decreased 5-way result. In addition, we present a t-SNE visualisation of the feature embeddings generated using our model. Fig. 6.7 (a) illustrates the overall data points added with NOP obfuscation in the support set and query set, projected into the raw embedding, in which data points are optimized by episode training. The relation between the pair of samples is recognized from Fig. 6.7 (b). Concretely, two clusters are represented as similarity and dissimilarity, respectively, which perform the characteristics of inter and intra.

6.6.5 Experimental Results On Malimage Dataset

On the Malimage dataset, it is worth noting that the intra-class variance has a significant decrease compared with the VUW dataset. Therefore, the feature representation



(a) Initialization Stage (b) Testing Stage

Figure 6.7: t-distributed stochastic neighbour embedding (t-SNE) visualisation of embeddings generated using our model

Table 6.4: Accuracy scores (%) tested on two dataset

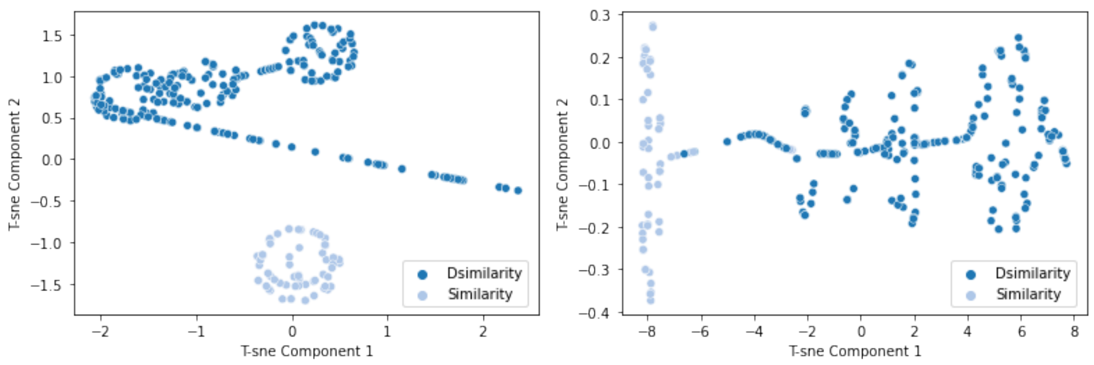
Model	2-way 1-shot	2-way 5-shot	5-way 1-shot	5-way 5-shot
	VUW dataset			
Relation [144]	55.8 ±1.8%	55.2±2.6%	42.6±3.1%	43.2±2.1%
Prototypical [134]	81.1±2.2%	86.4±1.9%	**69.3**±2.2%	**70.2**±2.0%
Prototypical [134] (Gray)	74.2±2.6%	82.6±2.5%	51.3±2.9%	53.7±2.1%
Our model (No augmentation)	80.1±2.9%	83.1±2.2%	60.2±3.4%	64.3±2.8%
Our model	**83.1**±3.1%	**87.2**±2.3%	63.2 ±2.8%	68.1±3.1%
Prototypical (Obfuscation)	80.3±2.7%	84.8±3.1%	**65.2**±2.7%	**68.3**±2.0%
Our model (Obfuscation)	**81.7**±2.9%	**85.2**±2.5%	53.2±2.7%	57.3±2.1%
	Malimage dataset			
Relation [144]	63.2±2.8%	68.1±3.1%	56.8± 3.4%	58.2±3.1%
Prototypical [134]	95.6±1.9%	96.7±1.8%	**92.3**±2.3%	**96.9**±2.1%
Prototypical [134] (Gray)	94.3±2.1%	94.9± 2.3%	90.1±2.5%	94.8±1.9%
Our model (No augmentation)	93.2 ±1.8%	95.2±1.7%	80.1 ±2.8%	87.1±1.7%
Our model	**96.1**±0.7%	**97.3**±1.4%	83.8 ±2.1%	90.2±1.8%
Prototypical (Obfuscation)	91.9±1.7%	94.7±1.8%	**85.4**±1.9%	85.7±1.8%
Our model (Obfuscation)	**94.3**±1.2%	**95.7**±1.1%	82.9±2.2%	**90.3**±2.1%

can have robust performance with the condition that variance is low. It can be seen that our model still achieves the best performance on 2-way results with 96.1% and 97.3% on 2-way 1-shot and 2-way 5-shot respectively on the non-obfuscated dataset. The performance on the obfuscated dataset is almost identical as on the non-obfuscated dataset, which results in a drop of only 1.8% and 1.6% on 2-way 1-shot and 2-way 5-shot respectively. This is contrasted with the prototypical network, which decreased 3.7% and 2.0%. The same situation happens with another result on the 5-way performance, which results on the obfuscated dataset a drop of only 0.9% on the 5-way 1-shot. The performance on the 5-way 5-shot only has a slight difference in the result, whereas the prototypical network decreased by 6.9% and 11.2% on the 5-way 1 shot and 5-way 5-shot on the obfuscated dataset. From these results, our model has a certain effect on the anti-obfuscation technology. By using the framework of the Siamese AutoEncoder, our model achieved promising results with few-shot training, which further proves the efficiency of generalization methods in real applications.

6.6.6 Ablation Study

Pooling methods for feature selection

Within the domain of feature extraction and dimensionality reduction, the application of different pooling strategies can yield varied topological structures in the transformed feature space. The initial scatter plot Fig. 6.8(a), derived from an average pooling strategy, presents a diffuse constellation of data points across the feature space delineated by the first two components of t-SNE. This dispersion indicates that average pooling, which typically mitigates the influence of outliers by computing the mean feature value within a local region, has not resulted in pronounced separability among the data points. Consequently, this may suggest either an intrinsic homogeneity within the dataset or that the averaging process has overly smoothed distinctive patterns. Contrastingly, the second scatter plot Fig. 6.8(b) emerges from a sum pooling approach, where the aggregation of features is achieved through summation. Remarkably, this plot exhibits distinct, well-defined clusters, implying that the cumulative effect of features has enhanced the discrimination capability within the dataset. Thus, in this instance, sum pooling appears to have bolstered the contrast between different data instances, rendering a feature space where clusters are more easily distinguishable, which could be particularly advantageous for classification tasks requiring clear demarcation between different categories.



(a) Average pooling (b) Sum pooling

Figure 6.8: t-SNE visualisation of embeddings generated using our model

Hyperparameter λ with the dimension of the linear layer

We first analyze how the performance of our model is affected by the hyperparameter λ and then combine the dimensions of the linear layer to observe the changes in the results, as shown in Table 6.5. It has been seen that the few-shot detection results are sensitive to the dimensions of the linear layer and the hyperparameter λ. A 256-dimension of linear layer and $\lambda = 0.7$ achieve better performance than other values, compared with the $\lambda = 0.3$. Secondly, our average feature calculated from feature concatenation guides the relation-aware to construct an objective embedding that distinguishes the characteristics of intra-class and inter-class and reduces the rate of misclassifications.

We analyzed the trend of different dimension combinations in the linear layer and the value of λ for the performance. It has been seen from Table 6.5, as the value of λ increases when the dimension of linear layer fixed, the accuracy rate increases gradually until the value of λ reaches 0.9. When the value of λ is fixed, the dimension of 256 performs better than the result with the dimension of 128.

Table 6.5: The performance on Malimage dataset with hyperparameter λ for 2-way 5-shot accuracy

	$\lambda = 0.3$	$\lambda = 0.5$	$\lambda = 0.7$	$\lambda = 0.9$
linear layer				
128	80.2	81.7	91.1	85.0
256	82.7	86.4	95.7	86.1

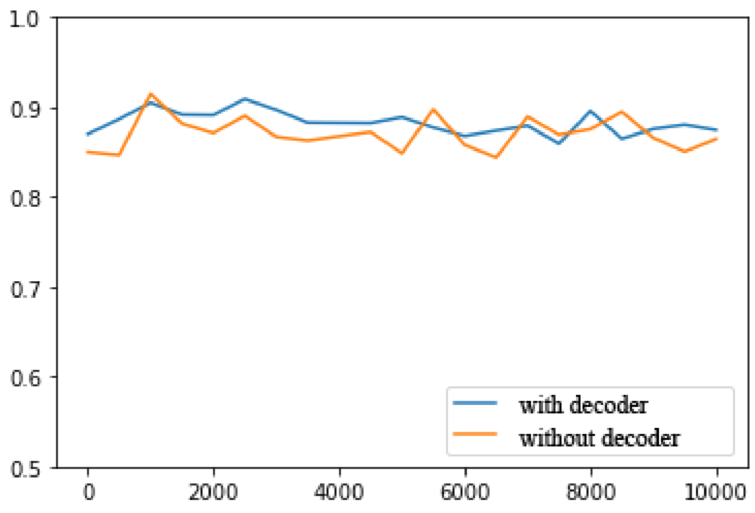


Figure 6.9: The comparison with or without the decoder part with/without decorder

Impact of decoder part

To explore the presence of the AutoEncoder in the schema, we use only part of the encoder in our model to learn the embedding representation while removing the decoder part. The encoder's purpose is to reduce the influence of the reconstruction loss function, as this is designed to create a robust feature against the NOP obfuscation. Observed from Fig. 6.9, the model's performance fluctuates between 0.842 and 0.921 without decoding, which is slightly wider than the fluctuation range of 0.852 to 0.903 observed with decoding. Additionally, the average performance of the model with a decoder, at 0.882, is higher than the 0.862 average achieved without a decoder, particularly noticeable in the episodes from 2000 to 6000.

6.7 Conclusion

In this study, we introduce a new approach for detecting zero-day malware attacks that have been disguised using obfuscation techniques like junk code and no-operation code insertions. Our proposed solution, a Siamese Neural Network (SNN) combined with denoising AutoEncoder, addresses the challenge of identifying unseen malware signatures.

We take into account the relationships between features in different malware samples during the training process, leveraging entropy-based features to better capture the unique and structural information of each malware sample, even in the presence of obfuscation. Instead of relying on traditional distance scores, we use a relation-aware embedding method based on probabilities to accurately capture the semantic differences between malware samples and classify them.

Evaluations were conducted on two widely used malware datasets, Malimage and VUW, to demonstrate the effectiveness of our proposed model in predicting unseen malware classes, even in the presence of obfuscation techniques.

In the future, we aim to broaden the scope of our research by incorporating a more extensive range of malware samples. This will allow us to uncover a more diverse range of correlations and further evaluate the adaptability and versatility of our proposed model. Additionally, we plan to incorporate various other obfuscation techniques such as code hiding using Packers/XOR/Base64, Register reassignment, and Code Transposition/Subroutine Reordering, among others. This will further enhance our model's ability to predict zero-day malware attacks effectively.

Chapter 7

Conclusions and Future Directions

The objective of this thesis is to enhance the accuracy of malware detection using few-shot learning techniques. Traditional deep learning methods lack the ability to detect unknown software while existing proposals rarely focus on detecting unknown malware with few-shot learning. To address this gap, we first utilized a Siamese neural network with batch normalization layers to identify shortcomings in existing proposals. Subsequently, we defined, investigated, and improved few-shot learning for the detection of unknown malware.

In Chapter 2 in response to the challenge posed by having a limited set of malware samples, we developed a new one-shot model named "Multi-Loss Siamese Neural Network with Batch Normalization Layer." This innovative approach aligns with our first research goal as it specifically addresses the need for high detection accuracy in situations where data is scarce. Our model leverages the capabilities of the Siamese Neural Network to effectively detect new variants of malware, even when trained with just a few samples. This directly contributes to our goal of assessing the performance of a Siamese neural network under conditions of limited data availability. We have equipped our model with batch normalization and multiple loss functions to mitigate the overfitting issue, which is a common challenge when working with small sample sizes. The integration of these components helps to prevent vanishing gradient problems that could arise due to the use of binary cross-entropy loss in the embedding space. This enhancement is crucial for improving detection accuracy, aligning seamlessly with our objective of determining optimal parameters and network components for effective malware detection with a Siamese Neural Network. By proposing and implementing the "Multi-Loss Siamese Neural Network with Batch Normalization Layer," we have made significant strides towards achieving our first research goal. Our model not only demonstrates the feasibility of using a Siamese Neural Network with a limited dataset but also showcases how specific network modifications and enhancements can lead to high detection accuracy in such challenging conditions.

In summary, the work done directly addresses the research goal, providing valuable insights and practical solutions for employing Siamese Neural Networks in malware detection, even with scarce data resources.

In Chapter 3, In our approach, we incorporated the average entropy features of each malware family as additional inputs alongside image features. This strategy is in direct response to our second research goal, aiming to enhance the model's feature robustness and its ability to differentiate important features even when obfuscated malware variants are present. By using these average entropy features, our model generates parameters specifically tailored for the feature layers, resulting in a more precise adjustment of the feature embedding for different malware families. This demonstrates our commitment to addressing the challenges posed by obfuscated malware variants, ensuring that our model can accurately identify and differentiate between them. The inclusion of average entropy features aids in mitigating the noise and enhancing the model's detection capabilities, particularly in the presence of obfuscated malware variants. This aligns with our research goal of evaluating the robustness of features and the model's capacity to discern significant features amidst noise, ultimately contributing to the development of a more effective and resilient malware detection system. Through the integration of average entropy features and the subsequent enhancement of feature embedding for various malware families, we have made significant progress in achieving our second research goal. Our model's improved ability to adjust feature embedding in the presence of obfuscated malware variants highlights the effectiveness of our approach, ensuring the robustness of features and enhanced detection capabilities amidst noise.

In summary, the work done directly addresses the second research goal, demonstrating a tangible improvement in the model's malware detection capabilities, particularly in challenging scenarios involving obfuscated malware variants.

In Chapter 3, The utilization of entropy features in our model has significantly bolstered its ability to find robust feature correlations across different ransomware samples. Compared to image features, entropy features have demonstrated a greater resilience against the influence of white noise and complex texture features. This enhancement directly aligns with our second research goal, as it contributes to improving the feature robustness of the model. Entropy features' reduced sensitivity to the spatial distribution of small changes in the genetic code makes them more adept at recognizing arbitrary alterations, including obfuscation methods used to evade detection. This capability is crucial for the model's capacity to distinguish significant features amidst noise, ensuring accurate malware detection even in the presence of obfuscation techniques. The additional context provided highlights the challenges posed by relying solely on image features, as they are easily influenced by external factors such as white noise. By incorporating entropy features, our model overcomes these challenges, showcasing a marked improvement in feature robustness and noise resistance. This directly contributes to achieving our research goal, highlighting the effectiveness of our approach in enhancing the malware detection capabilities of the model.

In Chapter 4, our proposed model learns a unique malware signature across families using meta-learning, even in the presence of obfuscated malware samples. Each CNN branch of our SNN model has two sub-networks: a task-aware meta-learner network and an image network. The task-aware meta-learner network generates task-specific weights using entropy graphs that capture the unique signatures of different malware families. This weight parameter is combined with the shared weights of the two CNNs for accurate feature embedding adjustment. Our model combines entropy attributes with image-based features for malware detection and classification and uses a pre-trained VGG-16 network as part of the meta-learning process to avoid potential bias when training sample sizes are limited. We utilize two different types of loss functions to compute similarity scores within and across the feature embeddings of the two CNNs: embedding loss to compute inter-class variance, with a secondary embedding loss added to reduce bias from limited training samples, and hybrid loss to compute intra-class variance, with center loss added to enable positive and negative pairs to form more distinct clusters. In Chapter 5, we present a novel meta-transfer learning framework for incident response, which optimizes the learning process by leveraging external information to enhance the overall efficiency of the triage system. This is the first attempt to utilize task-driven meta-transfer learning for malware triage, showcasing the potential of this approach to strengthen cybersecurity operations. To achieve this, we employ a first-order approach in Model-Agnostic Meta-Learning (MAML) and fine-tune ResNet18 to write support items to the memory module. This simplifies and accelerates episode training without the need for backpropagation. Moreover, we utilize a weighted ratio based on similarity scores to triage malware classes that are considered risky or unknown. This prioritizes potentially zero-day attacks for further analysis, enabling the development of appropriate response strategies. The incorporation of entropy features and the resultant improvement in feature robustness and noise resistance underscore the success of our approach in addressing the challenges outlined in the second research goal. Our model has demonstrated an enhanced ability to accurately detect malware, even in the face of obfuscation techniques, affirming the effectiveness of our chosen features and methodologies.

In summary, by integrating entropy features and demonstrating their effectiveness in improving feature robustness and noise resistance, our work directly addresses and fulfills the objectives set out in the second research goal, contributing to the development of a more resilient and effective malware detection system.

In chapter 5, Our proposed solution, a malware triage approach, addresses the need for quick and efficient prioritization of various malware types. This is particularly relevant given the recent surge in malware attacks and the subsequent socioeconomic damage. The capability to rapidly classify and prioritize different malware classes aligns with our third research goal, aiming to minimize the impact of such attacks through fast response plans.

The versatility of our approach allows for its application beyond malware prioritization, extending to other incident response scenarios. This broad applicability further demonstrates our commitment to achieving a balance between artificial intelligence and semi-automation, ensuring that the risks associated with known vulnerabilities are accurately assessed and addressed in a timely manner.

Our malware triage approach directly contributes to the fulfilment of the third research goal. By facilitating rapid classification and prioritization of different malware classes, we enhance the capacity for prompt response, minimizing socioeconomic damage and bolstering cybersecurity measures. This achievement underscores the effectiveness of our approach and its alignment with the objectives set out in our research goals.

In summary, the work done in proposing and implementing a malware triage approach directly addresses the challenges outlined in the third research goal. Our model not only enhances the ability to respond swiftly to malware attacks but also demonstrates adaptability to other incident response scenarios, contributing to a more robust and efficient cybersecurity landscape.

In Chapter 6, The frequent updates and unknown nature of malicious samples pose a significant challenge to current few-shot learning methods, particularly when obfuscation techniques are employed to evade detection. This issue directly aligns with our fourth research goal as it highlights the need for a robust model capable of understanding the semantic relationships between malware samples, even when obfuscation is present. We recognized that establishing a pattern and creating a relation-aware embedding in malware detection cannot be achieved with just a branch of a stacked autoencoder. This insight is crucial as it points to the limitations of existing methods and sets the stage for our proposed solution.To overcome these challenges, we propose a model constructed with a Siamese Neural Network (SNN), specifically designed to provide a relation-aware embedding. This model is intended to be resilient against obfuscation techniques, ensuring effective malware detection even when adversaries employ evasion tactics.

Our proposed Siamese Neural Network-based model addresses the critical challenges posed by unknown and frequently updated malicious samples, as well as obfuscation techniques. By focusing on relation-aware embedding, our approach directly contributes to fulfilling the fourth research goal, ensuring that the model can effectively understand semantic relationships between malicious functions and remain robust in real-world scenarios.

In summary, the work done directly addresses the challenges outlined in the fourth research goal, demonstrating a significant advancement in the field of malware detection. Our proposed Siamese Neural Network-based model enhances the model's ability to understand semantic relationships between malware samples, providing robustness against obfuscation techniques and ensuring effectiveness in practical applications.

Meanwhile Chapters 2–5 mainly have focused in the feature representation of malware , the importance of function against obfuscation technology encouraged us to develop and adapt a few-shot learning model in a realistic environment. In Chapter 6, we address the challenge of obfuscation technology by developing and adapting a few-shot learning model in a realistic environment, with a focus on exploring new features to train our model beyond traditional malware image features. We propose the use of entropy-based features, which provide a more comprehensive capture of the distinctiveness and structural information of the malware and significantly contribute to the effective differentiation of malware classes. To achieve this, we implement a state-of-the-art Siamese Neural Network (SNN) combined with denoising autoencoders. The SNN enables efficient classification of malware classes, even with limited samples, while the integration of denoising autoencoders with a relation-aware module in each branch of the SNN effectively captures the semantic differences and complex nonlinear relationships between features in a pairwise malware comparison. Our model incorporates relation-aware embeddings to output more precise similarity probabilities between various malware samples, enhancing the accuracy of our learning model and enabling it to distinguish between different malware signatures more effectively than traditional distance scores. Thus, we will explore the signatures with the sub-graph relationship in the meta-learning context in the near future.

Bibliography

[1] Abbasi, M. S., Al-Sahaf, H., and Welch, I. Particle swarm optimization: A wrapper-based feature selection method for ransomware detection and classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 12104 LNCS* (4 2020), 181–196.

[2] Akarsh, S., Poornachandran, P., Menon, V. K., and Soman, K. A detailed investigation and analysis of deep learning architectures and visualization techniques for malware family identification. In *Cybersecurity and Secure Information Systems*. Springer, 2019, pp. 241–286.

[3] Al-Sahaf, H., and Welch, I. A genetic programming approach to feature selection and construction for ransomware, phishing and spam detection.

[4] Alahmadi, A., Alkhraan, N., and BinSaeedan, W. Mpsautodetect: A malicious powershell script detection model based on stacked denoising autoencoder. *Computers & Security 116* (2022), 102658.

[5] Alenezi, M. N., Alabdulrazzaq, H., Alshaher, A. A., and Alkharang, M. M. Evolution of malware threats and techniques: A review. *International journal of communication networks and information security 12*, 3 (2020), 326–337.

[6] Alsouda, Y., Pllana, S., and Kurti, A. Iot-based urban noise identification using machine learning: performance of svm, knn, bagging, and random forest. In *Proceedings of the international conference on omni-layer intelligent systems* (2019), pp. 62–67.

[7] Amira, A., Derhab, A., Karbab, E., Nouali, O., and Khan, F. Tridroid: a triage and classification framework for fast detection of mobile threats in android markets. *Journal of Ambient Intelligence and Humanized Computing 12* (02 2021).

[8] Azar, K. Z., Kamali, H. M., Roshanisefat, S., Homayoun, H., Sotiriou, C. P., and Sasan, A. Data flow obfuscation: A new paradigm for obfuscating circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 29*, 4 (2021), 643–656.

[9] Bacci, A., Bartoli, A., Martinelli, F., Medvet, E., and Mercaldo, F. Detection of obfuscation techniques in android applications. In *Proceedings of the 13th International Conference on Availability, Reliability and Security* (2018), pp. 1–9.

[10] Bakour, K., and Ünver, H. M. Visdroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. *Neural Computing and Applications 33*, 8 (2021), 3133–3153.

[11] Baldwin, J., and Dehghantanha, A. *Leveraging Support Vector Machine for Opcode Density Based Detection of Crypto-Ransomware*. Springer International Publishing, Cham, 2018, pp. 107–136.

[12] Basnet, M., Poudyal, S., Ali, M., Dasgupta, D., et al. Ransomware detection using deep learning in the scada system of electric vehicle charging station. *arXiv preprint arXiv:2104.07409* (2021).

[13] Berrada, L., Zisserman, A., and Kumar, M. P. Smooth loss functions for deep top-k classification. *arXiv preprint arXiv:1802.07595* (2018).

[14] Bronskill, J., Massiceti, D., Patacchiola, M., Hofmann, K., Nowozin, S., and Turner, R. Memory efficient meta-learning with large images. *Advances in Neural Information Processing Systems 34* (2021), 24327–24339.

[15] Camp, L. J., Grobler, M., Jang-Jaccard, J., Probst, C., Renaud, K., and Watters, P. Measuring human resilience in the face of the global epidemiology of cyber attacks. In *Proceedings of the 52nd Hawaii International Conference on System Sciences* (2019).

[16] Canfora, G., Mercaldo, F., and Visaggio, C. A. An hmm and structural entropy based detector for android malware: An empirical study. *Computers & Security 61* (2016), 1–18.

[17] Cao, J., Su, Z., Yu, L., Chang, D., Li, X., and Ma, Z. Softmax cross entropy loss with unbiased decision boundary for image classification. In *2018 Chinese Automation Congress (CAC)* (2018), IEEE, pp. 2028–2032.

[18] Carrier, T. Detecting obfuscated malware using memory feature engineering.

[19] CHAI, Y., DU, L., QIU, J., YIN, L., AND TIAN, Z. Dynamic prototype network based on sample adaptation for few-shot malware detection. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[20] CHAI, Y., QIU, J., YIN, L., ZHANG, L., GUPTA, B. B., AND TIAN, Z. From data and model levels: Improve the performance of few-shot malware classification. *IEEE Transactions on Network and Service Management* (2022).

[21] CHAKRADEO, S., REAVES, B., TRAYNOR, P., AND ENCK, W. Mast: Triage for market-scale mobile malware analysis. In *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks* (2013), pp. 13–24.

[22] CHAN, J.-T., AND YANG, W. Advanced obfuscation techniques for java bytecode. *Journal of systems and software 71*, 1-2 (2004), 1–10.

[23] CHEN, L. Deep transfer learning for static malware classification. *CoRR abs/1812.07606* (2018).

[24] CHEN, L., SAHITA, R., PARIKH, J., AND MARINO, M. Stamina: Scalable deep learning approach for malware classification. *Intel Labs Whitepaper, https://www. intel. com/content/www/us/en/artificial-intelligence/documents/stamina-deep-learningfor-malware-protection-whitepaper. html* (2020).

[25] CHEN, X., AND HE, K. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 15750–15758.

[26] CHENG, S., ZHONG, B., LI, G., LIU, X., TANG, Z., LI, X., AND WANG, J. Learning to filter: Siamese relation network for robust tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 4421–4431.

[27] CHOW, S., GU, Y., JOHNSON, H., AND ZAKHAROV, V. A. An approach to the obfuscation of control-flow of sequential computer programs. In *International Conference on Information Security* (2001), Springer, pp. 144–155.

[28] CHUA, M., AND BALACHANDRAN, V. Effectiveness of android obfuscation on evading anti-malware. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy* (2018), pp. 143–145.

[29] CONNOR, J. T., MARTIN, R. D., AND ATLAS, L. E. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks 5*, 2 (1994), 240–254.

[30] CUI, Z., DU, L., WANG, P., CAI, X., AND ZHANG, W. Malicious code detection based on cnns and multi-objective algorithm. *Journal of Parallel and Distributed Computing 129* (2019), 50–58.

[31] CUI, Z., XUE, F., CAI, X., CAO, Y., WANG, G.-G., AND CHEN, J. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics 14*, 7 (2018), 3187–3196.

[32] CUI, Z., XUE, F., CAI, X., CAO, Y., WANG, G.-G., AND CHEN, J. Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics 14*, 7 (2018), 3187–3196.

[33] DAMODARAN, A., TROIA, F. D., VISAGGIO, C. A., AUSTIN, T. H., AND STAMP, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques 13*, 1 (2017), 1–12.

[34] DAVIES, S. R., MACFARLANE, R., AND BUCHANAN, W. J. Differential area analysis for ransomware attack detection within mixed file datasets. *Computers & Security 108* (2021), 102377.

[35] DE PAOLA, A., FAVALORO, S., GAGLIO, S., RE, G. L., AND MORANA, M. Malware detection through low-level features and stacked denoising autoencoders. In *ITASEC* (2018).

[36] DONG, S., LI, M., DIAO, W., LIU, X., LIU, J., LI, Z., XU, F., CHEN, K., WANG, X., AND ZHANG, K. Understanding android obfuscation techniques: A large-scale investigation in the wild. In *International Conference on Security and Privacy in Communication Systems* (2018), Springer, pp. 172–192.

[37] DORFER, M., SCHLÜTER, J., VALL, A., KORZENIOWSKI, F., AND WIDMER, G. End-to-end cross-modality retrieval with cca projections and pairwise ranking loss. *International Journal of Multimedia Information Retrieval 7*, 2 (2018), 117–128.

[38] D'ANGELO, G., FICCO, M., AND PALMIERI, F. Malware detection in mobile environments based on autoencoders and api-images. *Journal of Parallel and Distributed Computing 137* (2020), 26–33.

[39] FARUKI, P., BHARMAL, A., LAXMI, V., GAUR, M. S., CONTI, M., AND RAJARAJAN, M. Evaluation of android anti-malware techniques against dalvik bytecode obfuscation. In *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications* (2014), IEEE, pp. 414–421.

[40] Feng, S., Liu, Q., Patel, A., Bazai, S. U., Jin, C.-K., Kim, J. S., Sarrafzadeh, M., Azzollini, D., Yeoh, J., Kim, E., et al. Automated pneumothorax triaging in chest x-rays in the new zealand population using deep-learning algorithms. *Journal of Medical Imaging and Radiation Oncology* (2022).

[41] Ferrante, A., Malek, M., Martinelli, F., Mercaldo, F., and Milosevic, J. Extinguishing ransomware - a hybrid approach to android ransomware detection. In *Foundations and Practice of Security* (Cham, 2018), A. Imine, J. M. Fernandez, J.-Y. Marion, L. Logrippo, and J. Garcia-Alfaro, Eds., Springer International Publishing, pp. 242–258.

[42] Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning* (2017), PMLR, pp. 1126–1135.

[43] Fujimoto, M., and Kawai, H. Comparative evaluations of various factored deep convolutional rnn architectures for noise robust speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), IEEE, pp. 4829–4833.

[44] Furfaro, A., Argento, L., Parise, A., and Piccolo, A. Using virtual environments for the assessment of cybersecurity issues in iot scenarios. *Simulation Modelling Practice and Theory 73* (2017), 43–54.

[45] Furfaro, A., Piccolo, A., Parise, A., Argento, L., and Sacca, D. A cloud-based platform for the emulation of complex cybersecurity scenarios. *Future Generation Computer Systems 89* (2018), 791–803.

[46] Gao, T., Han, X., Liu, Z., and Sun, M. Hybrid attention-based prototypical networks for noisy few-shot relation classification. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 6407–6414.

[47] Gibert, D., Mateu, C., and Planes, J. A hierarchical convolutional neural network for malware classification. In *2019 International Joint Conference on Neural Networks (IJCNN)* (2019), IEEE, pp. 1–8.

[48] Gibert, D., Mateu, C., Planes, J., and Vicens, R. Classification of malware by using structural entropy on convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2018), vol. 32.

[49] Gidaris, S., and Komodakis, N. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 4367–4375.

[50] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[51] GÖRGEL, P., AND SIMSEK, A. Face recognition via deep stacked denoising sparse autoencoders (dsdsa). *Applied Mathematics and Computation 355* (2019), 325–342.

[52] GUO, G., WANG, H., BELL, D., BI, Y., AND GREER, K. Knn model-based approach in classification. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"* (2003), Springer, pp. 986–996.

[53] HAMID, T., AL-JUMEILY, D., AND MUSTAFINA, J. Evaluation of the dynamic cybersecurity risk using the entropy weight method. In *Technology for Smart Futures*. Springer, 2018, pp. 271–287.

[54] HAN, K. S., LIM, J. H., KANG, B., AND IM, E. G. Malware analysis using visualized images and entropy graphs. *International Journal of Information Security 14*, 1 (2015), 1–14.

[55] HAN, W., XUE, J., WANG, Y., HUANG, L., KONG, Z., AND MAO, L. Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *computers & security 83* (2019), 208–233.

[56] HAN, W., XUE, J., WANG, Y., LIU, Z., AND KONG, Z. Malinsight: A systematic profiling based malware detection framework. *Journal of Network and Computer Applications 125* (2019), 236–250.

[57] HASHEMI, H., AZMOODEH, A., HAMZEH, A., AND HASHEMI, S. Graph embedding as a new approach for unknown malware detection. *Journal of Computer Virology and Hacking Techniques 13*, 3 (2017), 153–166.

[58] HSIAO, S.-C., KAO, D.-Y., LIU, Z.-Y., AND TSO, R. Malware image classification using one-shot learning with siamese networks. *Procedia Computer Science 159* (2019), 1863–1871.

[59] HSU, S. T., MOON, C., JONES, P., AND SAMATOVA, N. A hybrid cnn-rnn alignment model for phrase-aware sentence classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (2017), pp. 443–449.

[60] HU, H., GU, J., ZHANG, Z., DAI, J., AND WEI, Y. Relation networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 3588–3597.

[61] Hu, X., Griffin, K. E., and Bhatkar, S. B. Encoding machine code instructions for static feature based malware clustering, Sept. 2 2014. US Patent 8,826,439.

[62] Hwang, W.-J., and Wen, K.-W. Fast knn classification algorithm based on partial distance search. *Electronics letters 34*, 21 (1998), 2062–2063.

[63] Ioffe, S., and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[64] Jang, J., Brumley, D., and Venkataraman, S. Bitshred: Feature hashing malware for scalable triage and semantic analysis. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2011), CCS '11, Association for Computing Machinery, p. 309–320.

[65] Jang, J.-w., Kang, H., Woo, J., Mohaisen, A., and Kim, H. K. Andro-dumpsys: anti-malware system based on the similarity of malware creator and malware centric information. *computers & security 58* (2016), 125–138.

[66] Jeon, J., Jeong, B., Baek, S., and Jeong, Y.-S. Hybrid malware detection based on bi-lstm and spp-net for smart iot. *IEEE Transactions on Industrial Informatics 18*, 7 (2021), 4830–4837.

[67] Johnson, S., Gowtham, R., and Nair, A. R. Ensemble model ransomware classification: A static analysis-based approach. In *Inventive Computation and Information Technologies*. Springer, 2022, pp. 153–167.

[68] Jusoh, R., Firdaus, A., Anwar, S., Osman, M. Z., Darmawan, M. F., and Ab Razak, M. F. Malware detection using static analysis in android: a review of feco (features, classification, and obfuscation). *PeerJ Computer Science 7* (2021), e522.

[69] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., and Iqbal, F. Malware classification with deep convolutional neural networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (2018), IEEE, pp. 1–5.

[70] Kang, B., Kang, B., Kim, J., and Im, E. G. Android malware classification method: Dalvik bytecode frequency analysis. In *Proceedings of the 2013 research in adaptive and convergent systems*. 2013, pp. 349–350.

[71] Khan, R. U., Zhang, X., and Kumar, R. Analysis of resnet and googlenet models for malware detection. *Journal of Computer Virology and Hacking Techniques 15* (2019), 29–37.

[72] KIM, J.-Y., BU, S.-J., AND CHO, S.-B. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. *Information Sciences 460* (2018), 83–102.

[73] KIM, J.-Y., AND CHO, S.-B. Obfuscated malware detection using deep generative model based on global/local features. *Computers & Security 112* (2022), 102501.

[74] KIM, S., YEOM, S., OH, H., SHIN, D., AND SHIN, D. Automatic malicious code classification system through static analysis using machine learning. *Symmetry 13* (2020).

[75] KING, G., AND ZENG, L. Logistic regression in rare events data. *Political analysis 9*, 2 (2001), 137–163.

[76] KIRAT, D., NATARAJ, L., VIGNA, G., AND MANJUNATH, B. S. Sigmal: A static signal processing based malware triage. In *Proceedings of the 29th Annual Computer Security Applications Conference* (New York, NY, USA, 2013), ACSAC '13, Association for Computing Machinery, p. 89–98.

[77] KOCH, G., ZEMEL, R., AND SALAKHUTDINOV, R. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop* (2015), vol. 2, Lille.

[78] KOLOSNJAJI, B., DEMONTIS, A., BIGGIO, B., MAIORCA, D., GIACINTO, G., ECKERT, C., AND ROLI, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In *2018 26th European Signal Processing Conference (EUSIPCO)* (2018), IEEE, pp. 533–537.

[79] KRČÁL, M., ŠVEC, O., BÁLEK, M., AND JAŠEK, O. Deep convolutional malware classifiers can learn from raw executables and labels only.

[80] KUMAR, S., ET AL. Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. *Future Generation Computer Systems 125* (2021), 334–351.

[81] KUMAR, S., AND JANET, B. Dtmic: Deep transfer learning for malware image classification. *Journal of Information Security and Applications 64* (2022), 103063.

[82] KUMAR, S., MEENA, S., KHOSLA, S., AND PARIHAR, A. S. Ae-dcnn: Autoencoder enhanced deep convolutional neural network for malware classification. In *2021 International Conference on Intelligent Technologies (CONIT)* (2021), IEEE, pp. 1–5.

[83] KURNIAWATI, Y. E., PERMANASARI, A. E., AND FAUZIATI, S. Adaptive synthetic-nominal (adasyn-n) and adaptive synthetic-knn (adasyn-knn) for multiclass imbalance learning on laboratory test data. In *2018 4th International Conference on Science and Technology (ICST)* (2018), IEEE, pp. 1–6.

[84] LAURENZA, G., ANIELLO, L., LAZZERETTI, R., AND BALDONI, R. Malware triage based on static features and public apt reports. In *Cyber Security Cryptography and Machine Learning* (Cham, 2017), S. Dolev and S. Lodha, Eds., Springer International Publishing, pp. 288–305.

[85] LAURENZA, G., LAZZERETTI, R., AND MAZZOTTI, L. Malware triage for early identification of advanced persistent threat activities. *Digital Threats: Research and Practice 1*, 3 (aug 2020).

[86] LEE, Y., WOO, S., LEE, J., SONG, Y., MOON, H., AND LEE, D. H. Enhanced android app-repackaging attack on in-vehicle network. *Wireless Communications and Mobile Computing 2019* (2019).

[87] LI, J., SUN, L., YAN, Q., LI, Z., SRISA-AN, W., AND YE, H. Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics 14*, 7 (2018), 3216–3225.

[88] LI, X., QIU, K., QIAN, C., AND ZHAO, G. An adversarial machine learning method based on opcode n-grams feature in malware detection. In *2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC)* (2020), IEEE, pp. 380–387.

[89] LI, Y., CHAI, X., AND CHEN, X. End-to-end learning for action quality assessment. In *Pacific Rim Conference on Multimedia* (2018), Springer, pp. 125–134.

[90] LIU, T., SABRINA, F., JANG-JACCARD, J., XU, W., AND WEI, Y. Artificial intelligence-enabled ddos detection for blockchain-based smart transport systems. *Sensors 22*, 1 (2021), 32.

[91] LO, W. W., YANG, X., AND WANG, Y. An xception convolutional neural network for malware classification with transfer learning. In *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (2019), pp. 1–5.

[92] LUO, J.-S., AND LO, D. C.-T. Binary malware image classification using machine learning with local binary pattern. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 4664–4667.

[93] MA, Y., AND ZHAO, X. Pod: A parallel outlier detection algorithm using weighted knn. *IEEE Access 9* (2021), 81765–81777.

[94] MAHDAVIFAR, S., ALHADIDI, D., AND GHORBANI, A. A. Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. *Journal of network and systems management 30* (2022), 1–34.

[95] MAKANDAR, A., AND PATROT, A. Trojan malware image pattern classification. In *Proceedings of International Conference on Cognition and Recognition* (2018), Springer, pp. 253–262.

[96] MALVAR, H. S., HE, L.-W., AND CUTLER, R. High-quality linear interpolation for demosaicing of bayer-patterned color images. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing* (2004), vol. 3, IEEE, pp. iii–485.

[97] MCINTOSH, T., JANG-JACCARD, J., WATTERS, P., AND SUSNJAK, T. The inadequacy of entropy-based ransomware detection. In *International Conference on Neural Information Processing* (2019), Springer, pp. 181–189.

[98] MCINTOSH, T. R., JANG-JACCARD, J., AND WATTERS, P. A. Large scale behavioral analysis of ransomware attacks. In *International Conference on Neural Information Processing* (2018), Springer, pp. 217–229.

[99] MILOSEVIC, N., DEHGHANTANHA, A., AND CHOO, K.-K. R. Machine learning aided android malware classification. *Computers & Electrical Engineering 61* (2017), 266–274.

[100] MIRZAEI, O., SUAREZ-TANGIL, G., TAPIADOR, J., AND DE FUENTES, J. M. Triflow: Triaging android applications using speculative information flows. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (New York, NY, USA, 2017), ASIA CCS '17, Association for Computing Machinery, p. 640–651.

[101] MORATO, D., BERRUETA, E., MAGAÑA, E., AND IZAL, M. Ransomware early detection by the analysis of file sharing traffic. *Journal of Network and Computer Applications 124* (2018), 14–32.

[102] MOUSTAKIDIS, S., AND KARLSSON, P. A novel feature extraction methodology using siamese convolutional neural networks for intrusion detection. *Cybersecurity 3*, 1 (2020), 1–13.

[103] MPANTI, A., NIKOLOPOULOS, S. D., AND POLENAKIS, I. A graph-based model for malicious software detection exploiting domination relations between system-call groups. In *Proceedings of the 19th International Conference on Computer Systems and Technologies* (2018), pp. 20–26.

[104] NAEEM, H., ULLAH, F., NAEEM, M. R., KHALID, S., VASAN, D., JABBAR, S., AND SAEED, S. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Networks 105* (2020), 102154.

[105] NATARAJ, L., KARTHIKEYAN, S., JACOB, G., AND MANJUNATH, B. S. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (2011), pp. 1–7.

[106] NATARAJ, L., YEGNESWARAN, V., PORRAS, P., AND ZHANG, J. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. *Proceedings of the 4th ACM workshop on Security and artificial intelligence - AISec '11* (2011).

[107] NG, C. K., JIANG, F., ZHANG, L. Y., AND ZHOU, W. Static malware clustering using enhanced deep embedding method. *Concurrency and Computation: Practice and Experience 31*, 19 (2019), e5234.

[108] NI, S., QIAN, Q., AND ZHANG, R. Malware identification using visualization images and deep learning. *Computers & Security 77* (2018), 871–885.

[109] NIKOLOPOULOS, S. D., AND POLENAKIS, I. Behavior-based detection and classification of malicious software utilizing structural characteristics of group sequence graphs. *Journal of Computer Virology and Hacking Techniques* (2022), 1–24.

[110] NOOR, M., ABBAS, H., AND SHAHID, W. B. Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis. *Journal of Network and Computer Applications 103* (2018), 249–261.

[111] PANT, D., AND BISTA, R. Image-based malware classification using deep convolutional neural network and transfer learning. In *Proceedings of the 3rd International Conference on Advanced Information Science and System* (2021), pp. 1–6.

[112] PASCANU, R., STOKES, J. W., SANOSSIAN, H., MARINESCU, M., AND THOMAS, A. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015), IEEE, pp. 1916–1920.

[113] PEKTAŞ, A., AND ACARMAN, T. Deep learning for effective android malware detection using api call graph embeddings. *Soft Computing 24*, 2 (2020), 1027–1043.

[114] R., V., Alazab, M., Jolfaei, A., K.P., S., and Poornachandran, P. Ransomware triage using deep learning: Twitter as a case study. In *2019 Cybersecurity and Cyberforensics Conference (CCC)* (2019), pp. 67–73.

[115] Rabbani, M., Wang, Y. L., Khoshkangini, R., Jelodar, H., Zhao, R., and Hu, P. A hybrid machine learning approach for malicious behaviour detection and recognition in cloud computing. *Journal of Network and Computer Applications 151* (2020), 102507.

[116] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. K. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[117] Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., and Nicholas, C. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques 14*, 1 (2018), 1–20.

[118] Rasthofer, S., Arzt, S., Kolhagen, M., Pfretzschner, B., Huber, S., Bodden, E., and Richter, P. Droidsearch: A tool for scaling android app triage to real-world app stores. In *2015 Science and Information Conference (SAI)* (2015), pp. 247–256.

[119] Ravi, V., Pham, T. D., and Alazab, M. Attention-based multidimensional deep learning approach for cross-architecture iomt malware detection and classification in healthcare cyber-physical systems. *IEEE Transactions on Computational Social Systems* (2022), 1–10.

[120] Rhode, M., Burnap, P., and Jones, K. Early-stage malware prediction using recurrent neural networks. *Computers & Security 77* (2018), 578–594.

[121] Sahay, S. K., Sharma, A., and Rathore, H. Evolution of malware and its detection techniques. In *Information and Communication Technology for Sustainable Development: Proceedings of ICT4SD 2018* (2020), Springer, pp. 139–150.

[122] Salman, O., Elhajj, I. H., Kayssi, A., and Chehab, A. Denoising adversarial autoencoder for obfuscated traffic detection and recovery. In *International conference on machine learning for networking* (2019), Springer, pp. 99–116.

[123] Sandra, K., and Lee, S.-H. Bm3d and deep image prior based denoising for the defense against adversarial attacks on malware detection networks. *International journal of advanced smart convergence 10*, 3 (2021), 163–171.

[124] SAXE, J., AND BERLIN, K. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th international conference on malicious and unwanted software (MALWARE)* (2015), IEEE, pp. 11–20.

[125] SHAMSUDDIN, M. R. B., ALI, F. H. H. M., AND ABIDIN, M. S. B. Z. Transforming malware behavioural dataset for deep denoising autoencoders. In *IOP Conference Series: Materials Science and Engineering* (2020), vol. 769, IOP Publishing, p. 012071.

[126] SHAO, H., ZHONG, D., DU, X., DU, S., AND VELDHUIS, R. N. Few-shot learning for palmprint recognition via meta-siamese network. *IEEE Transactions on Instrumentation and Measurement 70* (2021), 1–12.

[127] SHARMA, P., AND RAGLIN, A. Efficacy of nonlinear manifold learning in malware image pattern analysis. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2018), IEEE, pp. 1095–1102.

[128] SHARMA, S., KHANNA, K., AND AHLAWAT, P. Survey for detection and analysis of android malware (s) through artificial intelligence techniques. In *Cyber Security and Digital Forensics*. Springer, 2022, pp. 321–337.

[129] SHAUKAT, S. K., AND RIBEIRO, V. J. Ransomwall: A layered defense system against cryptographic ransomware attacks using machine learning. In *2018 10th International Conference on Communication Systems Networks (COMSNETS)* (2018), pp. 356–363.

[130] SHEN, J., CAO, X., LI, Y., AND XU, D. Feature adaptation and augmentation for cross-scene hyperspectral image classification. *IEEE Geoscience and Remote Sensing Letters 15*, 4 (2018), 622–626.

[131] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[132] SINGH, A., DUTTA, D., AND SAHA, A. Migan: malware image synthesis using gans. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 10033–10034.

[133] SISON, M. O. T. Calculating distances between windows malware using siamese neural network embeddings.

[134] SNELL, J., SWERSKY, K., AND ZEMEL, R. Prototypical networks for few-shot learning. *Advances in neural information processing systems 30* (2017).

[135] SOH, J. W., CHO, S., AND CHO, N. I. Meta-transfer learning for zero-shot super-resolution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 3516–3525.

[136] SONG, L., TANG, Z., LI, Z., GONG, X., CHEN, X., FANG, D., AND WANG, Z. Appis: Protect android apps against runtime repackaging attacks. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)* (2017), IEEE, pp. 25–32.

[137] STIBOR, T. A study of detecting computer viruses in real-infected files in the n-gram representation with machine learning methods. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems* (2010), Springer, pp. 509–519.

[138] SU, J., VASCONCELLOS, D. V., PRASAD, S., SGANDURRA, D., FENG, Y., AND SAKURAI, K. Lightweight classification of iot malware based on image recognition. In *2018 IEEE 42Nd annual computer software and applications conference (COMPSAC)* (2018), vol. 2, IEEE, pp. 664–669.

[139] SUAREZ-TANGIL, G., DASH, S. K., AHMADI, M., KINDER, J., GIACINTO, G., AND CAVALLARO, L. Droidsieve: Fast and accurate classification of obfuscated android malware. In *Proceedings of the seventh ACM on conference on data and application security and privacy* (2017), pp. 309–320.

[140] SUN, B., LI, Q., GUO, Y., WEN, Q., LIN, X., AND LIU, W. Malware family classification method based on static feature extraction. In *2017 3rd IEEE International Conference on Computer and Communications (ICCC)* (2017), IEEE, pp. 507–513.

[141] SUN, G., AND QIAN, Q. Deep learning and visualization for identifying malware families. *IEEE Transactions on Dependable and Secure Computing* (2018).

[142] SUN, Q., LIU, Y., CHEN, Z., CHUA, T.-S., AND SCHIELE, B. Meta-transfer learning through hard tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[143] SUN, Q., LIU, Y., CHUA, T.-S., AND SCHIELE, B. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 403–412.

[144] SUNG, F., YANG, Y., ZHANG, L., XIANG, T., TORR, P. H., AND HOSPEDALES, T. M. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 1199–1208.

[145] TANG, J., LI, R., JIANG, Y., GU, X., AND LI, Y. Android malware obfuscation variants detection method based on multi-granularity opcode features. *Future Generation Computer Systems 129* (2022), 141–151.

[146] TANG, Z., WANG, P., AND WANG, J. Convprotonet: Deep prototype induction towards better class representation for few-shot malware classification. *Applied Sciences 10*, 8 (2020), 2847.

[147] TOBIYAMA, S., YAMAGUCHI, Y., SHIMADA, H., IKUSE, T., AND YAGI, T. Malware detection with deep neural network using process behavior. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)* (2016), vol. 2, IEEE, pp. 577–582.

[148] TONG, F., AND YAN, Z. A hybrid approach of mobile malware detection in android. *Journal of Parallel and Distributed computing 103* (2017), 22–31.

[149] TORABI, S., DIB, M., BOU-HARB, E., ASSI, C., AND DEBBABI, M. A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships. *IEEE Networking Letters 3*, 3 (2021), 161–165.

[150] TRAN, T. K., SATO, H., AND KUBO, M. Image-based unknown malware classification with few-shot learning models. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)* (2019), IEEE, pp. 401–407.

[151] VAN NGUYEN, H., PATEL, V. M., NASRABADI, N. M., AND CHELLAPPA, R. Kernel dictionary learning. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2012), IEEE, pp. 2021–2024.

[152] VASAN, D., ALAZAB, M., WASSAN, S., NAEEM, H., SAFAEI, B., AND ZHENG, Q. Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks 171* (2020), 107138.

[153] VASAN, D., ALAZAB, M., WASSAN, S., SAFAEI, B., AND ZHENG, Q. Image-based malware classification using ensemble of cnn architectures (imcec). *Computers & Security 92* (2020), 101748.

[154] VIDYA, B. S., AND CHANDRA, E. Entropy based local binary pattern (elbp) feature extraction technique of multimodal biometrics as defence mechanism for cloud storage. *Alexandria Engineering Journal 58*, 1 (2019), 103–114.

[155] VINYALS, O., BLUNDELL, C., LILLICRAP, T., WIERSTRA, D., ET AL. Matching networks for one shot learning. In *Advances in neural information processing systems* (2016), pp. 3630–3638.

[156] WANG, L., LI, Y., ZHANG, H., HAN, Q., AND CHEN, L. An efficient control-flow based obfuscator for micropython bytecode. In *2021 7th International Symposium on System and Software Reliability (ISSSR)* (2021), IEEE, pp. 54–63.

[157] WANG, Q., GUO, W., ZHANG, K., ORORBIA, A. G., XING, X., LIU, X., AND GILES, C. L. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), pp. 1145–1153.

[158] WANG, S., CHEN, Z., YAN, Q., YANG, B., PENG, L., AND JIA, Z. A mobile malware detection method using behavior features in network traffic. *Journal of Network and Computer Applications 133* (2019), 15–25.

[159] WANG, W., ZHAO, M., AND WANG, J. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing 10*, 8 (2019), 3035–3043.

[160] WANG, Z., PENG, C., ZHANG, Y., WANG, N., AND LUO, L. Fully convolutional siamese networks based change detection for optical aerial images with focal contrastive loss. *Neurocomputing 457* (2021), 155–167.

[161] WEI, Y., JANG-JACCARD, J., SABRINA, F., SINGH, A., XU, W., AND CAMTEPE, S. Ae-mlp: A hybrid deep learning approach for ddos detection and classification. *IEEE Access 9* (2021), 146810–146821.

[162] WEI, Y., JANG-JACCARD, J., SABRINA, F., SINGH, A., XU, W., AND CAMTEPE, S. Ae-mlp: A hybrid deep learning approach for ddos detection and classification. *IEEE Access to appear* (2021).

[163] WEN, Y., ZHANG, K., LI, Z., AND QIAO, Y. A discriminative feature learning approach for deep face recognition. In *European conference on computer vision* (2016), Springer, pp. 499–515.

[164] WOOK JANG, J., KANG, H., WOO, J., MOHAISEN, A., AND KIM, H. K. Andro-dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information. *Computers Security 58* (2016), 125 – 138.

[165] XIAO, X., ZHANG, S., MERCALDO, F., HU, G., AND SANGAIAH, A. K. Android malware detection based on system call sequences and lstm. *Multimedia Tools and Applications 78*, 4 (2019), 3979–3999.

[166] XU, L., ZHANG, D., JAYASENA, N., AND CAVAZOS, J. Hadm: Hybrid analysis for detection of malware. In *Proceedings of SAI Intelligent Systems Conference* (2016), Springer, pp. 702–724.

[167] XU, W., JANG-JACCARD, J., SINGH, A., WEI, Y., AND SABRINA, F. Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset. *IEEE Access* (2021).

[168] XU, Z., LI, M., HEI, Y., LI, P., AND LIU, J. A malicious android malware detection system based on implicit relationship mining. In *2021 8th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2021 7th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)* (2021), IEEE, pp. 59–64.

[169] XU, Z., NAPPA, A., BAYKOV, R., YANG, G., CABALLERO, J., AND GU, G. Autoprobe: Towards automatic active malicious server probing using dynamic binary analysis. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 179–190.

[170] YATES, A., ARORA, S., ZHANG, X., YANG, W., JOSE, K. M., AND LIN, J. Capreolus: A toolkit for end-to-end neural ad hoc retrieval. In *Proceedings of the 13th International Conference on Web Search and Data Mining* (2020), pp. 861–864.

[171] YOO, S., KIM, S., KIM, S., AND KANG, B. B. Ai-hydra: Advanced hybrid approach using random forest and deep learning for malware classification. *Information Sciences 546* (2021), 420–435.

[172] YOU, G., KIM, G., CHO, S.-J., AND HAN, H. A comparative study on optimization, obfuscation, and deobfuscation tools in android. *J. Internet Serv. Inf. Secur. 11*, 1 (2021), 2–15.

[173] YUAN, B., WANG, J., LIU, D., GUO, W., WU, P., AND BAO, X. Byte-level malware classification based on markov images and deep learning. *Computers & Security 92* (2020), 101740.

[174] YUAN, Y., ZHENG, G., WONG, K.-K., OTTERSTEN, B., AND LUO, Z.-Q. Transfer learning and meta learning-based fast downlink beamforming adaptation. *IEEE Transactions on Wireless Communications 20*, 3 (2020), 1742–1755.

[175] YUE, S. Imbalanced malware images classification: a CNN based approach. *CoRR abs/1708.08042* (2017).

[176] ZAHOORA, U., RAJARAJAN, M., PAN, Z., AND KHAN, A. Zero-day ransomware attack detection using deep contractive autoencoder and voting based ensemble classifier. *Applied Intelligence 52*, 12 (2022), 13941–13960.

[177] ZHANG, B., XIAO, W., XIAO, X., SANGAIAH, A. K., ZHANG, W., AND ZHANG, J. Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes. *Future Generation Computer Systems 110* (2020), 708–720.

[178] ZHANG, C., LIU, W., MA, H., AND FU, H. Siamese neural network based gait recognition for human identification. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2016), IEEE, pp. 2832–2836.

[179] ZHANG, D., ZHENG, Z., LI, M., HE, X., WANG, T., CHEN, L., JIA, R., AND LIN, F. Reinforced similarity learning: Siamese relation networks for robust object tracking. In *Proceedings of the 28th ACM International Conference on Multimedia* (2020), pp. 294–303.

[180] ZHANG, H., XIAO, X., MERCALDO, F., NI, S., MARTINELLI, F., AND SANGAIAH, A. K. Classification of ransomware families with machine learning based onn-gram of opcodes. *Future Generation Computer Systems 90* (2019), 211–221.

[181] ZHANG, L., ZUO, L., DU, Y., AND ZHEN, X. Learning to adapt with memory for probabilistic few-shot learning. *IEEE Transactions on Circuits and Systems for Video Technology 31*, 11 (2021), 4283–4292.

[182] ZHAO, M., WANG, D., ZHANG, Z., AND ZHANG, X. Music removal by convolutional denoising autoencoder in speech recognition. In *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)* (2015), IEEE, pp. 338–341.

[183] ZHENG, W., GOU, C., YAN, L., AND MO, S. Learning to classify: A flow-based relation network for encrypted traffic classification. In *Proceedings of The Web Conference 2020* (2020), pp. 13–22.

[184] ZHENG, X., PAN, L., AND YILMAZ, E. Security analysis of modern mission critical android mobile applications. In *Proceedings of the Australasian Computer Science Week Multiconference* (2017), pp. 1–9.

[185] ZHOU, X., LIANG, W., SHIMIZU, S., MA, J., AND JIN, Q. Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems. *IEEE Transactions on Industrial Informatics 17*, 8 (2020), 5790–5798.

[186] ZHU, H.-J., YOU, Z.-H., ZHU, Z.-X., SHI, W.-L., CHEN, X., AND CHENG, L. Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing 272* (2018), 638–646.

[187] ZHU, J., JANG-JACCARD, J., LIU, T., AND ZHOU, J. Joint spectral clustering based on optimal graph and feature selection. *Neural Processing Letters 53*, 1 (2021), 257–273.

[188] ZHU, J., JANG-JACCARD, J., SINGH, A., WATTERS, P. A., AND CAMTEPE, S. Task-aware meta learning-based siamese neural network for classifying obfuscated malware. *arXiv preprint arXiv:2110.13409* (2021).

[189] ZHU, J., JANG-JACCARD, J., SINGH, A., WELCH, I., AI-SAHAF, H., AND CAMTEPE, S. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *arXiv preprint arXiv:2112.00668* (2021).

[190] ZHU, J., JANG-JACCARD, J., SINGH, A., WELCH, I., HARITH, A.-S., AND CAMTEPE, S. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *Computers & Security 117* (2022), 102691.

[191] ZHU, J., JANG-JACCARD, J., AND WATTERS, P. A. Multi-loss siamese neural network with batch normalization layer for malware detection. *IEEE Access 8* (2020), 171542–171550.

[192] ZHU, Z., YOU, X., CHEN, C. P., TAO, D., OU, W., JIANG, X., AND ZOU, J. An adaptive hybrid pattern for noise-robust texture analysis. *Pattern Recognition 48*, 8 (2015), 2592–2608.

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Jinting Zhu
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Zhu, J., Jang-Jaccard, J., & Watters, P. A. (2020). Multi-loss siamese neural network with batch normalization layer for malware detection. IEEE access, 8, 171542-171550.	
In which Chapter is the Manuscript /Published work:	Chapter 2
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	85
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Concept Designing, Conducting the experiments, Manuscript Written	
For manuscripts intended for publication please indicate target journal:	
Candidate's Signature:	Jinting Zhu Digitally signed by Jinting Zhu Date: 2023.08.21 08:16:28 +12'00'
Date:	21/08/2023
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2023.08.22 09:50:25 +12'00'
Date:	22/08/2023

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Jinting Zhu
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Zhu, J., Jang-Jaccard, J., Singh, A., Watters, P. A., & Camtepe, S. (2023). Task-Aware Meta Learning-Based Siamese Neural Network for Classifying Control Flow Obfuscated Malware. Future Internet, 15(6), 214.	
In which Chapter is the Manuscript /Published work:	Chapter 3
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Concept Designing, Conducting the experiments, Manuscript Written	
For manuscripts intended for publication please indicate target journal:	
Candidate's Signature:	Jinting Zhu Digitally signed by Jinting Zhu Date: 2023.08.21 08:16:28 +12'00'
Date:	21/08/2023
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2023.08.22 09:51:19 +12'00'
Date:	22/08/2023

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION
DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Jinting Zhu
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Zhu, J., Jang-Jaccard, J., Singh, A., Welch, I., Harith, A. S., & Camtepe, S. (2022). A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. Computers & Security, 117, 102691.	
In which Chapter is the Manuscript /Published work:	Chapter 4
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	70
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Concept Designing, Conducting the experiments, Manuscript Written	
For manuscripts intended for publication please indicate target journal:	
Candidate's Signature:	Jinting Zhu Digitally signed by Jinting Zhu Date: 2023.08.21 08:16:28 +12'00'
Date:	21/08/2023
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2023.08.22 09:51:49 +12'00'
Date:	22/08/2023

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Jinting Zhu
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Zhu, J., Jang-Jaccard, J., Welch, I., Al-Sahaf, H., & Camtepe, S. A Ransomware Triage Approach using a Task Memory based on Meta-Transfer Learning Framework.	
In which Chapter is the Manuscript /Published work:	Chapter 5
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	90
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Concept Designing, Conducting the experiments, Manuscript Written	
For manuscripts intended for publication please indicate target journal:	
Computers & Security	
Candidate's Signature:	Jinting Zhu Digitally signed by Jinting Zhu Date: 2023.08.21 08:16:28 +12'00'
Date:	21/08/2023
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2023.08.22 09:52:22 +12'00'
Date:	22/08/2023

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)

MASSEY UNIVERSITY
GRADUATE RESEARCH SCHOOL

STATEMENT OF CONTRIBUTION DOCTORATE WITH PUBLICATIONS/MANUSCRIPTS

We, the candidate and the candidate's Primary Supervisor, certify that all co-authors have consented to their work being included in the thesis and they have accepted the candidate's contribution as indicated below in the *Statement of Originality*.

Name of candidate:	Jinting Zhu
Name/title of Primary Supervisor:	Associate Professor Julian Jang-Jaccard
Name of Research Output and full reference:	
Zhu, J., Jang-Jaccard, J., Welch, I., Al-Sahaf, H., & Camtepe, S. Relation-Aware Based Siamese Denoising Autoencoder for Malware Few-Shot Classification	
In which Chapter is the Manuscript /Published work:	Chapter 6
Please indicate:	
• The percentage of the manuscript/Published Work that was contributed by the candidate:	85
and	
• Describe the contribution that the candidate has made to the Manuscript/Published Work:	
Concept Designing, Conducting the experiments, Manuscript Written	
For manuscripts intended for publication please indicate target journal:	
Expert systems with Applications	
Candidate's Signature:	Jinting Zhu Digitally signed by Jinting Zhu Date: 2023.08.21 08:16:28 +12'00'
Date:	21/08/2023
Primary Supervisor's Signature:	Julian Jang-Jaccard Digitally signed by Julian Jang-Jaccard Date: 2023.08.22 09:52:43 +12'00'
Date:	22/08/2023

(This form should appear at the end of each thesis chapter/section/appendix submitted as a manuscript/ publication or collected as an appendix at the end of the thesis)