future internet

MDPI

Article

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

Jinting Zhu [1,*], Julian Jang-Jaccard [1], Amardeep Singh [1], Paul A. Watters [2,*] and Seyit Camtepe [3]

[1] Cybersecurity Lab, Massey University, Auckland 0632, New Zealand
[2] Cyberstronomy Pty Ltd., Melbourne 3086, Australia
[3] Data61, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Epping 1710, Australia
* Correspondence: jzhu3@massey.ac.nz (J.Z.); dr.paul.watters@gmail.com (P.A.W.)

Abstract: Malware authors apply different techniques of control flow obfuscation, in order 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 few-shot-learning-based Siamese Neural Network that is 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 few-shot 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 by N-way on N-shot learning, show that our model is highly effective in classification accuracy, exceeding a rate >91%, compared to other similar methods.

Keywords: Siamese neural network; meta-learning; malware classification; code obfuscation; few-shot learning

Citation: Zhu, J.; Jang-Jaccard, J.; Singh, A.; Watters, P.A.; Camtepe, S. Task-Aware Meta Learning-Based Siamese Neural Network for Classifying Control Flow Obfuscated Malware. *Future Internet* **2023**, *15*, 214. https://doi.org/10.3390/fi15060214

Academic Editors: Weizhi Meng, Christian D. Jensen and Claude Chaudet

Received: 12 March 2023
Revised: 7 June 2023
Accepted: 12 June 2023
Published: 14 June 2023

1. Introduction

Malware producers are ever more motivated to create new variants of malware, in order to gain profits from unauthorized information stealth. According to the malware detection agency AV Test (https://www.av-test.org/en/statistics/malware/ (accessed on 1 July 2021)), 100 million new variants of malware were generated from January to October 2020, which translates as roughly three thousand new malware daily.

In particular, we have witnessed the fast growth of mobile-based malware. An NTSC report published in 2020 (https://www.ntsc.org/assets/pdfs/cyber-security-report-2020.pdf (accessed on 15 June 2021)) reported that 27% of organizations globally have been impacted by malware attacks sent via Android mobile devices. In recent times, we have seen malware producers employ techniques such as obfuscation [1–3] and repackaging [4–6], mostly through the change of static features [7–9], to avoid detection. In response to the trend in the growth of mobile-based malware attacks, numerous Artificial Intelligence (AI)-based defense techniques have been proposed [10–15].

We argue that there are two main issues to be addressed in the existing state-of-the-art of AI-based mobile malware attack defense.

The first issue is that most of the existing research tends to focus on learning from common semantic information about the generic features of malware families, and building

feature embeddings [16–18]. In this context, 'feature embedding' means the features contained in a malware binary sample, which provide important clues as to whether the malware image—generated from the hexdump utility that displays the contents of binary files in hexadecimal, from which feature embedding is created—is malicious or not: if malicious, what type of malware family it belongs to is assessed, to 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 [19], and thus tend to ignore the effect of the modification: this is largely because the code changed by the obfuscation and repackaging techniques displays a similar appearance when malware visualization techniques are applied [20–22]. Using common semantic information as data input points to be fed 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 arising from the same malware family [10,23–25], especially if an obfuscation technique is applied.

The second issue with the existing approaches is the demand for large data input, with which to find more relevant correlations across the features: such input is unable to detect and classify malware families trained with a limited number of samples (e.g., newly emerging variants of malware) [26].

To address these two important issues, we propose a novel task-aware few-shot-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 as follows:

- Our task-aware meta-learner network combines entropy attributes with image-based features for malware detection and classification. By utilizing the VGG-16 network as part of the meta-learning process, the weight generator assigns the weights of the combined features, which avoids the potential issue of introducing bias when the training sample size is limited;
- For the hybrid loss to compute the intra-class variance, the center loss is added alongside the constructive loss, to enable positive pairs and negative pairs to form more distinct clusters across the pairs of images processed by two CNNs;
- The results of our extensive experiments show that our proposed model is highly effective in recognizing the presence of a unique malware signature, despite the presence of obfuscation techniques, and that its accuracy exceeds the performance of similar methods.

We organized the rest of the paper as follows. We examine the related work in Section 2. We describe how control-flow-obfuscated malware variants are created, and we address why the generic SNN approach cannot detect such obfuscated malware variants in Section 3. 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 4. In Section 5, we describe the details of the dataset, feature extraction, and the experimental results with analysis. Finally, we provide the conclusion to our work, including the limitations of our proposal, and future work directions, in Section 6.

2. Related Work

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

2.1. Entropy Feature in Feature Selection

Entropy value serves as a metric for feature selection, quantifying the information each feature contributes to the target variable or class. Selecting features with the highest information gain can enhance model performance and reduce overfitting. Huang et al. [27] utilized the entropy function defined on sparse signal x to recover such signals: minimizing it with an appropriate p-value yielded sparse solutions and improved signal recovery

rates. Additionally, Finlayson et al. [28] demonstrated that quadratic entropy values, being smoother and typically having a single minimum, offer the most efficient approach. This method aids full-color shadow removal, by comparing edges in the original and invariant images, followed by re-integration. Moreover, Allahverdyan et al. [29,30] suggest that entropy minimization can lead to basic forms of intelligent behavior. Specifically, Kolouri et al. [29] employed entropy minimization to bolster classifier confidence, while entropy regularization ensured that predictions remained close to unseen attributes in zero-shot tasks.

2.2. Meta-Learning in Cyber Threat Intelligence

Machine learning has made significant strides in intrusion detection, network protection, anomaly detection, and identifying known threats; however, traditional machine learning struggles with accurately recognizing unknown attacks. Meta-learning, encompassing "zero-shot", "one-shot", and "few-shot" learning, addresses this by detecting new malicious samples, using limited or no samples, relying on existing knowledge correlations. The goal is to quickly develop accurate and generalizable models with minimal data training.

Yang et al. [31] outlined the application of meta-learning in network security, highlighting its rapid development and promising potential. Zoppi et al. [32] suggested adopting meta-learners, which create ensembles of base-learners, to reduce misclassifications common in supervised learning. Base-learner results partially form meta-data, which, along with other features, is provided to the meta-layer for the overall classification. Despite ensemble learning's high accuracy, it suffers from complexity and reduced efficiency during model training. In a comprehensive study, Zoppi et al. [33] compared meta-learning methods to other approaches for detecting unknown attacks, evaluating major classifier families using the same methodology. Although their research did not cover "few-shot learning" methods, it highlighted the potential of meta-learning in detecting unknown attacks, such as zero-day exploits. Zhou et al. [34] proposed a Siamese CNN encoding network, designed to measure input sample distances based on optimized feature representations. This intelligent detection algorithm, applied to anomaly detection in large-scale industrial CPS data with few labeled samples, showcases the efficiency of an end-to-end training method based on "few-shot learning".

2.3. 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. [35] proposed a static method of analyzing assembly language operation code through the visualization of malicious code. More recently, Hsiao et al. [14] developed an end-to-end framework based on a Siamese neural network, to detect malware. Moustakidis et al. [36] attempted the transformation of raw data using fuzzy class memberships, which then fed into the Siamese neural network, to defend against the intrusion attack. While these researchers achieved competitive results, utilizing different feature extraction techniques, they were limited to learning semantic feature embedding, to distribute tasks more effectively; however, the description of the semantic information of the raw binary files that they used was not clear, and it was difficult to see how these were used for feature embedding. It appeared to be unrealistic to expect a model learned for a generic feature from rare categories to capture the common attribute of malicious code. Tang et al. [37] proposed a high-level malware class feature with a meta-learner, and evaluated that their proposal was effective at detecting malware with distinct features: however, their research did not consider the distance between the positive pairs and the negative pairs; therefore, the problem with the precise capture of the distance across intra-class variance still existed. Many existing state-of-the-arts proposed by these existing few-shot learning models tend to employ 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 center loss combined with a constructive function, to improve the positive and negative pairs interval and the inter-class variance.

2.4. 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 these principles, Sun et al. [35,38–42] adopted feature embedding techniques into malware detection. It has been proven that the embedded N-grams of opcodes [43] abbreviated from operation code—also known as instruction machine code—and graph embedding [38,40,44] can efficiently capture unique malicious components; however, conducting the opcodes and graph embedding is time-consuming, and their generalizability limited (e.g., graph embedding tends only to work well in learning the static features). Using these techniques, a unique behavior presented in a malware family, that is critical to capturing the variant of that malware family, often tends to be regarded as noises, while only the common known behaviors of malware families are captured by the CNN model: for example, work by Microsoft in collaboration with Intel demonstrated the setting of the practical value of the image-based transfer learning approach for static malware classification [42]. To address this problem, Sun et al. [35] proposed a model that learned the unique information specific to a malicious code; however, it did not achieve a competitive result when only a small number of training datasets were available. Tran et al. [45] evaluated the performance of malware detection for zero-day attack malware, using matching and prototypical networks. The matching network used the softmax over the cosine distance, with two embedding functions, and the memory cached the common pattern of malware feature representation. While these works focused on the semantic feature embedding of malicious code, most of them were limited to working well on many training samples: these existing models tended to be effective at capturing malware samples when the common attribute of malicious code was distinct, but often did not consider when there were slight differences in features of the malware, as is the case with many obfuscated malware variants.

3. Preliminary

3.1. Control Flow Obfuscation

Malware obfuscation is a technique that is applied by malware authors to create new malware variants, in order to avoid detection without creating a completely brand-new malware signature. Among the many different obfuscation techniques, we focused on control flow obfuscation, which involves creating a new malware variant by reordering the control flow of functional logic from the original malware program [1]. This type of obfuscation technique makes the compiled malicious code appear to be different from the existing malware signature so that it can easily avoid detection. Our model can detect three different types of control flow obfuscation.

3.1.1. 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: while the functionality between the original malware and the obfuscated version remains the same, the changing of appearance in the compiled code can result in the appearance of the malware image changing, and detection accuracy decreasing. An example is shown in Figure 1, where the order of function logic MyClass_2 and MyClass_3 is changed.

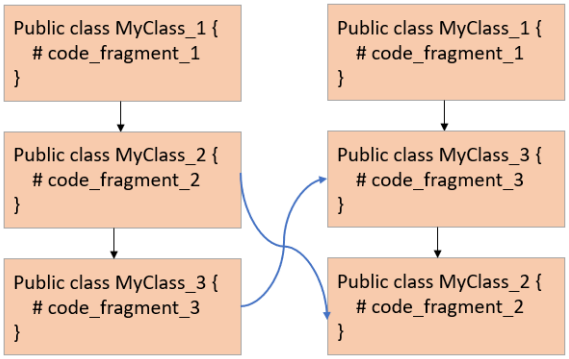


Figure 1. Function Logic Shuffling.

3.1.2. Junk Code Insertion

In this technique, the malware author inserts much (junk) code that never gets executed, whether after unconditional jumps, calls that never return, or conditional jumps with conditions that will never be met. The main goal of this technique is to change the control flow path, in order to avoid detection or to waste time for the reverse engineer analyzing useless code. An example of a junk code insertion is shown in Figure 2, where a junk code, MyClass_J, is added in between two normal function calls.

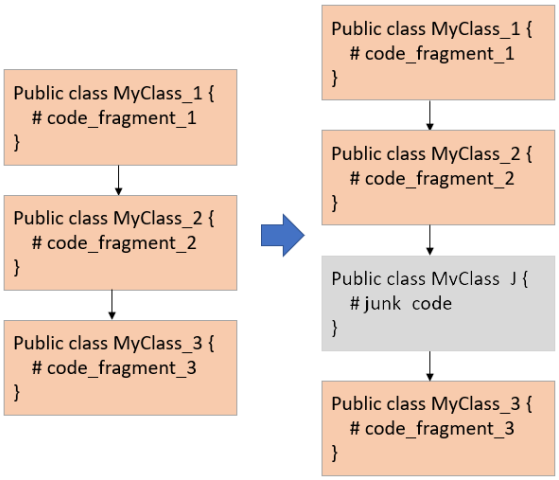


Figure 2. Junk Code Insertion.

3.1.3. 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 is inserted into 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, which also results in the malware image appearance changing. An example of a function splitting is shown in Figure 3, where two splits from MyClass_2 are generated, and randomly added among other function calls.

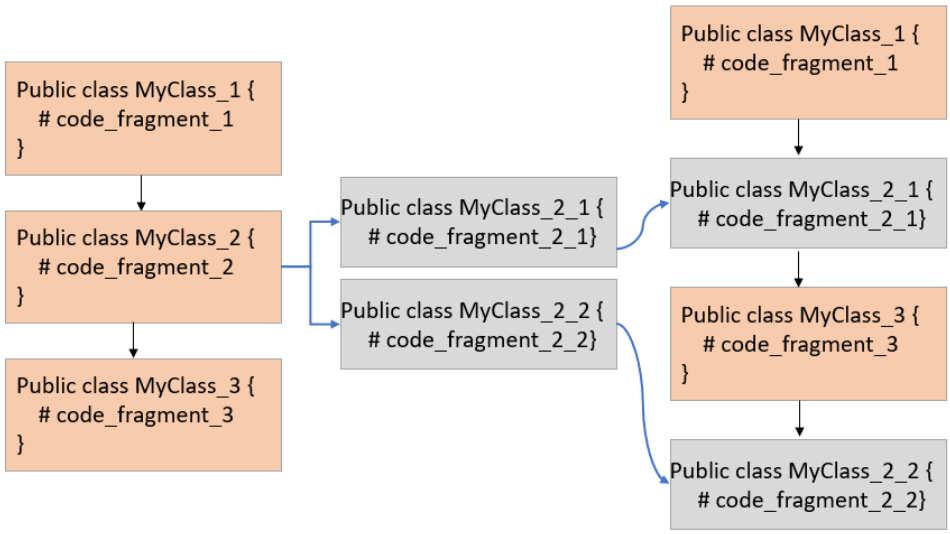


Figure 3. Function Splitting.

Though the functionality between the original malware and obfuscated versions (e.g., malware variants) stays the same, a malware code applied with control flow obfuscation can easily avoid detection from many anti-virus programs [1]. To address this issue, 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.

3.2. Generic Approach and Issues

In the last few years, few-shot learning technology, for example, the Siamese neural network (SNN), which uses only a few training samples to get better predictions has emerged. An 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 embedding to calculate two samples inputted from each subnetwork, respectively, and the value of the distance indicates whether they are closed to each other, as the value increases or decreases in the Euclidean space. The updating of the hyperparameters is mirrored across both CNNs, and 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., a 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.

The architecture of Spiking Neural Networks (SNNs) contrasts significantly with traditional neural networks, which rely on extensive datasets to learn the prediction of multiple classes. The latter's requirement for total retraining and updating with each addition or removal of a class presents a challenge. SNNs instead learn through a similarity function, testing whether two images are identical. This innovative architecture empowers them to classify new data classes without the need for additional network training. Furthermore, SNNs demonstrate greater robustness against class imbalance, as a small number of images per class is enough to enable future recognition. The corresponding notations are listed as Table 1.

Table 1. Notations in task-aware meta learning-based Siamese neural network.

Notation	Description
f_p	The log probability calculated by the $d_w^{(i)}$
f_q	The log probability calculated by the $1 - d_w^{(i)}$
$\mathbf{x_i}$	The malware image feature of ith samples
y_t^i	The class t of ith samples
θ_t	The feature layers' parameters
$\mathbf{W^i}$	I-th feature layer in $\mathcal{F}$'s weights
$\mathbf{W_{sr}^i}$	The shared parameters for all malware families
$\mathbf{W_{ts}^i}$	The task-specific parameters for each malware family
$\mathbf{c_i}$	The center point of each class
L_e	The embedding loss
$L_{b,c}$	The binary cross entropy loss with center loss
$\mathbf{d_w^i}$	The distance feature of a pair of images
y_d	The label of pairs of images
$\mathbf{F_w}$	The convolutional filter with parameters w
β	The hyper-parameter

Figure 4 illustrates the working of a generic SNN, its goal being 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 Figure 4) trained on the two image samples (Image 1 and Image 2 in Figure 4). Each image is fed through one branch of the SNN, which generates a d-dimensional feature embedding (h_1 and h_2 in Figure 4) 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 the SNN for the binary classification, to determine whether two images are similar or dissimilar, by computing $[h_2 - h_1]$ and processing it by a sigmoid function.

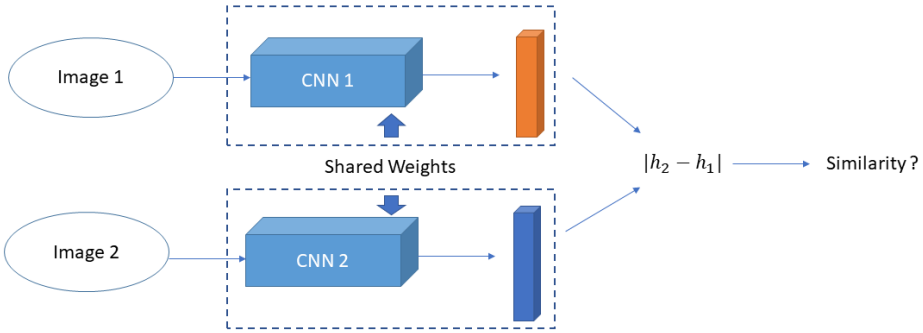


Figure 4. Overview of a Siamese neural network.

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

$$\mathbf{d_w}(\mathbf{x_1}, \mathbf{x_2}) = \|F_w(\mathbf{x_1}) - F_w(\mathbf{x_2})\|_2, \tag{1}$$

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 following equation:

$$L_b = -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{y_d^i} f_p(\mathbf{d_w^{(i)}}) + (1 - \mathbf{y_d^i}) f_q(\mathbf{d_w^{(i)}})], \tag{2}$$

where $\mathbf{y_d} = \{\mathbf{y_d^1}, \mathbf{y_d^2}, \ldots, \mathbf{y_d^i}\} \in \{0, 1\}$ denotes the ground truth label of image pairs $(\mathbf{x_i}, \mathbf{x_j})$, and 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, it would not work well for obfuscated malware samples.

Recall that an obfuscated malware—for example, $\mathbf{x_1}$—changes some part of the original malware code, x_2: when these two are converted as a feature representation—for example, $F_w(\mathbf{x_1})$ and $F_w(\mathbf{x_2})$—the feature values in the feature representation will look very different, which 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., $d_w(\mathbf{x_1}, \mathbf{x_2})$ 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., $d_w(\mathbf{x_1}, \mathbf{x_2})$), they appear to be different malware families—though, in fact, they all belong to the same malware family.

4. Task-Aware Meta Learning-Based Siamese Neural Network

We now introduce our task-aware meta-learning-based SNN model, which provides a novel feature embedding better-suited to control-flow-obfuscated malware classification. We start with the overview of our proposed model, and 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 uses to address the challenges of weight generation with a limited number of training samples.

4.1. Our Model

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

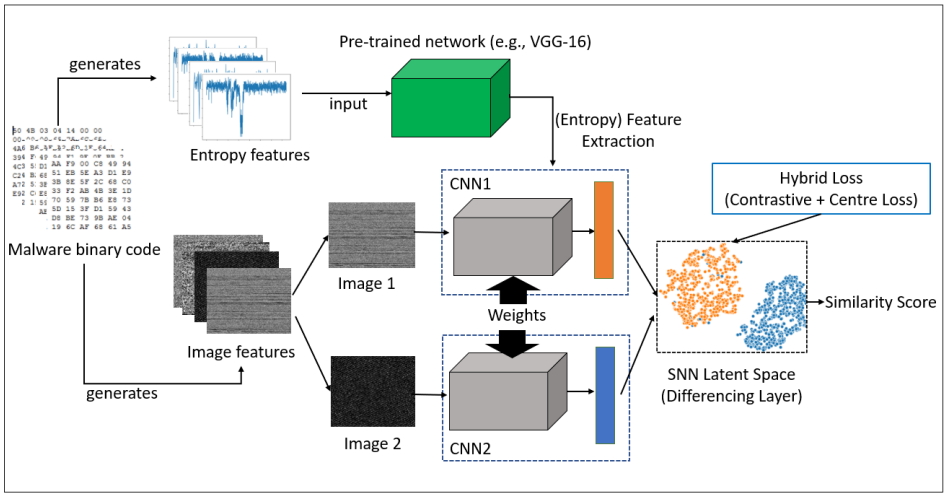


Figure 5. Overview of our proposed model.

Within each CNN, there are two sub-networks: a task-aware meta-learner network and an image network, as shown in Figure 6. The task-aware meta-learner network starts by taking a task specification (e.g., an entropy feature vector), and generates the weights. At the same time, the image network (e.g., 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 the CNN is fully trained for all input images.

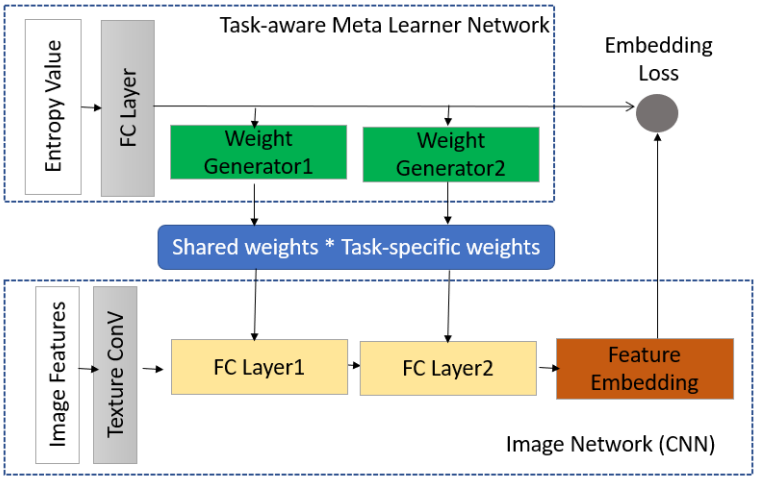


Figure 6. CNN architecture of our proposed model.

Mathematically, this can be written as the following equation:

$$y = \mathcal{F}(x; \theta = \{\mathcal{G}(t), \theta_f\}), \tag{3}$$

where the SNN $\mathcal{F}$ takes malware images $\mathbf{x}$ as inputs, and produces 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 are 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 $g^i, i = 1 \ldots k$, to generate parameters for k feature layers in $\mathcal{F}$, conditioned on e_t. The overall approach of our proposed model can also be summarized using Algorithm 1:

Algorithm 1: Pseudo-code of our proposed algorithm.

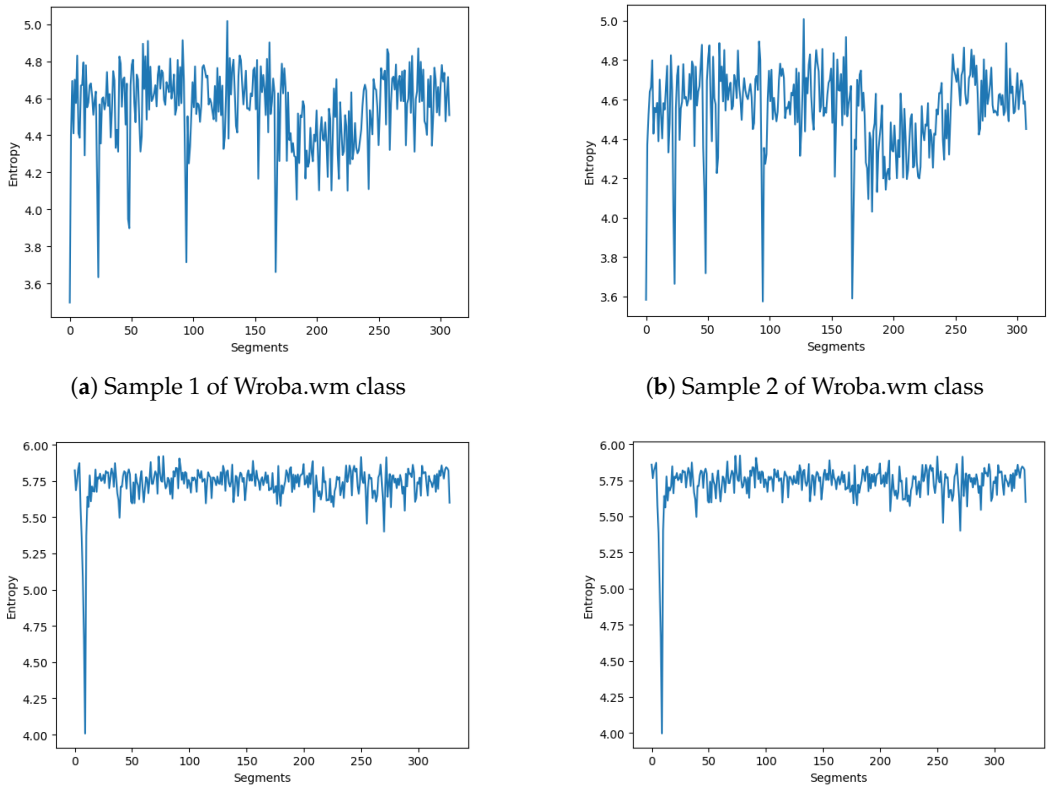
Binary cross-entropy with center loss:$L_{b,c}$;
Additional supervision loss :L_e;
Input :Entropy graph feature F_{ent}, texture feature F_t, support set s, query set q, pair label y_d^i, sample label $\mathbf{y_t^i}$, hyper-parameters β , initialized centers c_i
Output:[Predicted similarity]
Training stage:
(1) Initializing the parameters for our proposed models, and the task-specific weights W^i for the weight generators in the task-aware meta-learner network g^i, using the weight factorization Equation (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) The weighted features from Equation (5) are fed into the embedding loss, according to the one-shot 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 the one-shot accuracy.
Accuracy = $100 \times correct/iterations$

4.2. 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); the other function 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, in order to capture the relative distance across inter-class variance (e.g., the features of the image). These functions are necessary, because some malware samples (e.g., zero-day attack samples) are usually much smaller than the number of images required for training an SNN model.

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, four samples of malware images are shown in Figure 7. Figure 7a,b are two obfuscated malware samples from the same Wroba.wm family. Similarly, Figure 7c,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.



(**a**) Sample 1 of Wroba.wm class

(**b**) Sample 2 of Wroba.wm class

(**c**) Sample 1 of Agent.ad class

(**d**) Sample 2 of Agent.ad class

Figure 7. Examples of entropy graphs of two malware families.

Our task-aware 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 2. To obtain an entropy graph, a malware binary file is read as a stream of bytes, and is separated into a few segments.

The frequency of unique byte value is counted, and computes the entropy, using Shannon's formula, as follows:

$$Ent = -\sum_{i}\sum_{j} M(i,j) log M(i,j), \tag{4}$$

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 that 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 [46]). The summary of the steps involved in the entropy graph is described in Algorithm 2:

Algorithm 2: 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

4.3. 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 five malware samples) exist (i.e., not enough malware feature information to use for classification, as very few variations of malware samples can 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 [47]. In our proposed model, the weight generator $G(.,.|\phi)$ receives, as input, the entropy vectors $\mathbf{W_{ave}}$ of a class, in addition to the image vectors $\mathbf{Z'} = \{\mathbf{z_i'}\}_{i=1}^{N'}$ of the N' training samples of the corresponding class: this results in a weight vector $\mathbf{w'} = G(\mathbf{Z'}, \mathbf{W_{ave}})$.

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 the malware image. By integrating the weight generator into the FC layer, the weights of the features extracted before the FC layer can then 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 $\mathbf{W_i} \in \mathbf{R}^{m \times n}$ can be decomposed using the following equation:

$$\mathbf{W^i} = \mathbf{W^i_{sr}} \cdot \mathbf{W^i_{ts}}, \tag{5}$$

where $\mathbf{W^i_{sr}} \in \mathbf{R}^{m \times n}$ is the average entropy feature vector for each malware family $\{t_1, \ldots t_N\}$, and $\mathbf{W^i_{ts}} \in \mathbf{R}^n$ is the task-specific image feature vector for each malware. 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.

4.4. 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.

4.4.1. 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, and cannot estimate the probabilities of distances between positive and negative image pairs across different malware variations, thus not being able to correctly classify similar pairs of images across obfuscated malware samples (i.e., not being able to learn a discriminative feature during the training procedure). To address this issue, we added a secondary cross-entropy loss, not only to learn the discriminative feature, but also to address the effect of overfitting caused by contrastive loss. This embedding loss is defined using the following equation:

$$L_e = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T} log[\frac{exp(F(\mathbf{x_i};\theta_t))\cdot \mathbf{y_t^i}}{\sum_{j=1}^{T} exp(F(\mathbf{x_i};\theta_t))}], \tag{6}$$

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

4.4.2. 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 constructive loss. The center loss proposed by Wen et al. [48] is a supplement loss function to the softmax loss for the classification task, which 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 as close to the center of the sample as possible. This center loss can be calculated as follows:

$$L_c = \frac{1}{2N}\sum_{N}^{i=1}\left\|\mathbf{d_w^{(i)}} - \mathbf{c_i}\right\|_2^2, \tag{7}$$

where c_i denotes the center of class i, while $d_w^{(i)}$ denotes 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.

This approach, however, does not 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 have adopted 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:

$$\min L = \beta \min L_e + L_{b,c}, \tag{8}$$

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

5. Experiments

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

5.1. Andro-Dumpsys Dataset

We used the Andro-Dumpsys dataset obtained from [49], which has been widely used for malware detection. The original dataset consists of 906 malicious binary files from 13 malware families. As illustrated in Table 2, the number of malware variants and the total number of samples from different malware families varied. Almost half of the malware families had no more than 25 malware samples, while some only had 1 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 was applied using the 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.

Table 2. Andro-Dumpsys Dataset with synthesis samples.

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)

Figure 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, which we added in between two function calls from the original malware code. We applied a similar approach to the other two types of control flow obfuscation.

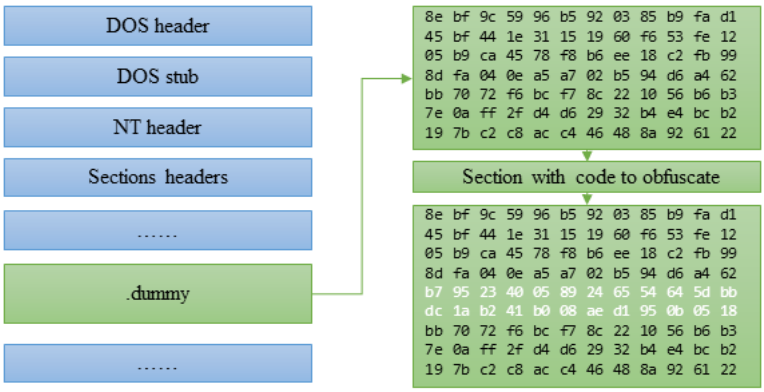


Figure 8. Example of inserting a junk code.

We also increased the image sample size, so as to have at least 30 samples for every malware family, using a data augmentation technique (e.g., applying random transformations, such as image rotations, re-scaling, and clipping the images horizontally). The details of the augmentation parameters are shown in Table 3. In particular, ZCA whitening is an image preprocessing method that leads to transformation of the data, such that the covariance matrix is the identity matrix, leading to decorrelated features, while the fill_mode argument with "wrap" simply replaces the empty area, and follows the filling scheme.

Table 3. Parameters of data augmentation.

Parameters	Values	Description
rescale	1./255	Resizing an image by a given scaling factor.
zca_epsilon	$1e^{-6}$	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.

5.1.1. Image Feature

We used the same technique proposed by [50] to produce image features. To produce image features, we first read the binary malware as a vector of 8-bit unsigned integers, which were then converted into 2D vectors. We used the fixed width, while the height was decided according to the size of the original file. Finally, the 2D vector was converted into the gray images, using the color range [0, 255]. Note that the gray images at this stage were different dimensions according to the varying heights and widths in which size biases could occur in the fully connected layer: to address this issue, we used the bilinear interpolation method, as suggested by [51], to produce our image feature in uniform to the size of 105×105.

5.1.2. Entropy Feature

To obtain the entropy feature of each malware family, we took each byte of the malware binary file as a random variable, and counted the frequency of each value (00h-FFh). More concretely, the byte reads from the binary file were divided into several segments. For each segment, we first calculated the frequency of each byte value $p_j (1 \leq j \leq m)$, and then we calculated the entropy y_i of the segment. The entropy values were then represented as a stream of values that could be reshaped as an entropy graph, with the size of $254 \times 254 \times 1$. These entropy graphs were then converted as a 4096-dimensional feature vector inputted through the convolutional extractor of the VGG-16 architecture [46].

5.2. Model Configurations

The task-aware meta-learner network $\mathcal{T}$(t) was a two-layer FC network with a hidden unit size of 512, except for the top layer, which was 4096 for input. The weight generator g^i was a single FC layer with the output dimension the same as the output dimension of the corresponding feature layer in $\mathcal{F}$. We added a ReLU function to the output of g^i in cases where processed malware images were used as inputs, and the convolutional part was configured as the 4-layer convolutional network (excluding pooling layers), following the same structure as [52]. In addition, the ReLU function and batch normalization were added afterwards by the convolution layer and the FC layer. The total number of parameters in our proposed model is 40 million. The overview of our network configurations is described in Figure 9.

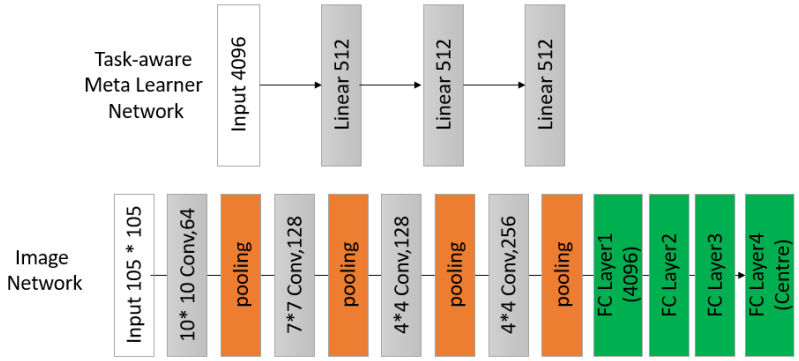


Figure 9. Network Configurations.

5.3. Results

We set the batch size to 32, and we used Adam as the optimizer, with an initial learning rate of 10^{-4} for the image network and the weight generators, and 10^{-5} for the task embedding network. The network was trained with 50 epochs, which ran for approximately 2 h. As Figure 10 illustrates, both train and validation loss stabilized after 50 epochs, confirming that the training had been done by this stage.

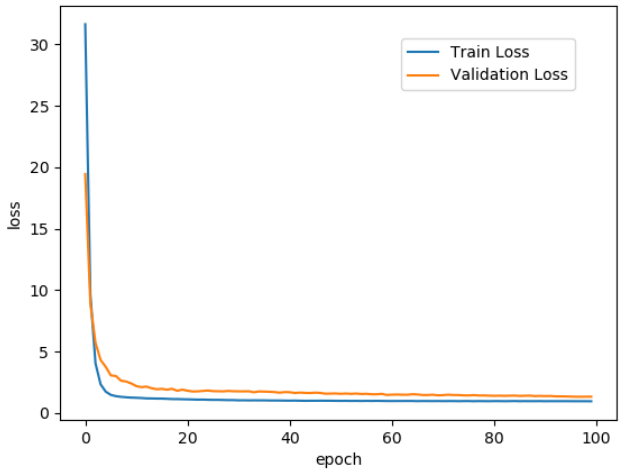


Figure 10. Loss during training on epochs.

The testing process conducted M times of N-way on N-shot learning tasks, where Q times of correct predictions contributed to the accuracy, calculated by the following formula:

$$Accuracy = (100 * Q/m)\%. \quad (9)$$

5.3.1. *N*-Way Matching Accuracy

The evaluation of N-way learning at each test state was carried out for one-shot and five-shot. For N-way one-shot learning, we chose an anchor image from one class of test, and then randomly selected N classes of images to form the support set $X = \{x_i\}_{i=1}^{N}$, where $x_1, \forall x \in X$; the selected image's class was the same as the anchor image $\hat{x}$, and the other images in the support sets were from different classes. The similarity score between $\hat{x}$ and other images was calculated using our model. Specifically, if the similarity score of the feature vector of x_1, which could be represented as $S = \{s_i\}_{i=1}^{N}$, was the maximum of S, then the task could be labeled as a correct prediction; otherwise, it was regarded as an incorrect prediction. For N-way five-shot learning, we randomly selected N unseen classes and six instances, in which five instances of each class were randomly selected as the support set, $X = \{x_1, \ldots, x_i\}_{i=5}^{N}$, and the remaining instances of each class formed the query set: its prediction procedure was the same as the test in the one-shot.

The matching accuracy of N-way accuracy for the one-shot and five-shot are illustrated in Figure 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 Figure 11a, 19 out of 25 positive image pairs were matched correctly, while 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 five-shot results (shown in Figure 11b), the accuracy of matching was higher, as almost 22 out 25 pairs matched correctly for both positive and negative pairs, and there were 3 incorrectly matched results.

Figure 12 shows the projection of the embeddings space, using the two-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. Figure 12a shows the embedding space before training, while Figure 12b projects the embedding space after training. After training, we could clearly see two distinct clusters—one around the distance calculated for all positive pairs, and the other for negative pairs: this confirmed that our proposed model had learned well, so as to distinguish the similarities among positive pairs and negative pairs and separate them far apart.

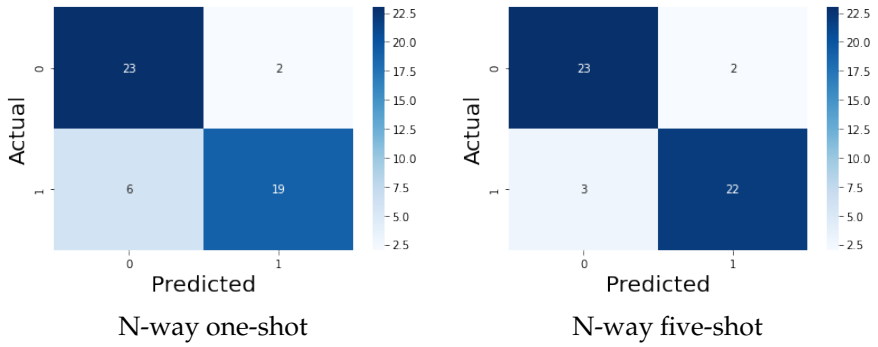
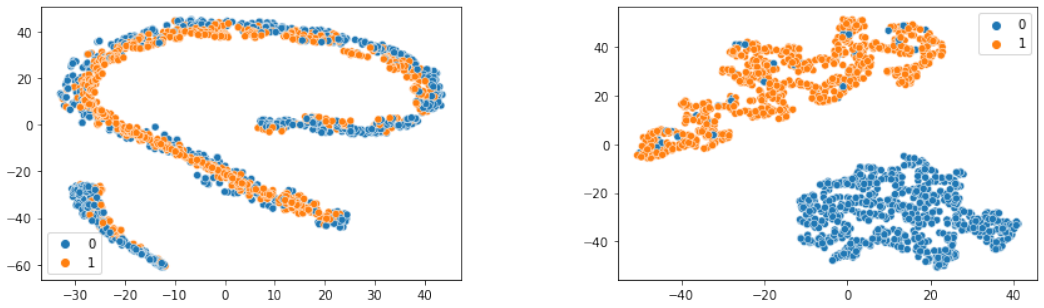


Figure 11. Matching Accuracy.



(**a**) Initial Stage (**b**) Trained Stage

Figure 12. PCA visualization for classification performance.

5.3.2. Benchmark against Similar Methods

Table 4 shows the result of benchmarking our proposed model against the current state-of-the-art, especially 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 one-shot learning on the 5-way. The difference between our results of one-shot and five-shot was 3.1%, 1.9%, and 1.4% on the 5-way, 10-way, and 15-way, respectively. Our proposed five-shot result outperformed all three exiting models by 1.4%, 5.9%, and 5.4% on the 5-way, 10-way, and 15-way, respectively.

Table 4. Comparison of classification performance of different few-shot learning approaches for Andro-Dumpsys datasets.

1-shot				
Ref	**Method**	**5-way**	**10-way**	**15-way**
[53]	Matching network	85 ± 2.2%	84 ± 2.4%	76 ± 2.6%
[54]	Prototypical network	86 ± 1.7%	82 ± 1.7%	81 ± 1.9%
[52]	Siamese network	82 ± 2.5%	69 ± 2.3%	64 ± 2.6%
	Task-aware SNN	88 ± 2.2%	86 ± 2.2%	82 ± 2.4%
5-shot				
Ref	**Method**	**5-way**	**10-way**	**15-way**
[53]	Matching network	89 ± 2.1%	86± 2.1%	78± 2.3%
[54]	Prototypical network	89 ± 1.2%	85± 1.4%	82 ± 1.5%
[52]	Siamese network	85 ± 2.0%	72 ± 2.2%	69 ± 2.7%
	Task-aware SNN	91± 1.8%	88± 2.1%	83 ± 2.1%

5.3.3. Distance Measure Effectiveness

We also examined the effectiveness of our proposed model in distance measurement, by using the AUC (area under the curve) ROC (receiver operating characteristic) curve. The AUC–ROC curve is commonly composed of two performance measures: the true-positive (FPR) rate and the false-positive rate (FPR) rate. The equations related to these two performance measures of the AUC–ROC curve are shown as follows:

$$\begin{aligned} FPR(P^*) &:= \int_{p^*}^{1} f_0(p)dp, \\ TPR(P^*) &:= \int_{p^*}^{1} f_1(p)dp. \end{aligned} \tag{10}$$

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 labeled as 0, and $f_1(p)$ is the probability from the positive pair that is labeled as 1. The given discrimination threshold P is 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:

$$AUC - ROC = \int_{0}^{1} TPR(FPR)D(FPR). \tag{11}$$

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 demonstrated the result predicted by the saved weights of one-shot and five-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 Figures 13 and 14, which illustrate that the β value at 0.8 provided 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 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.

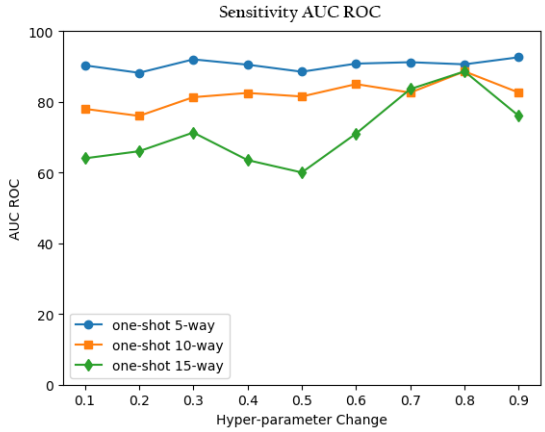
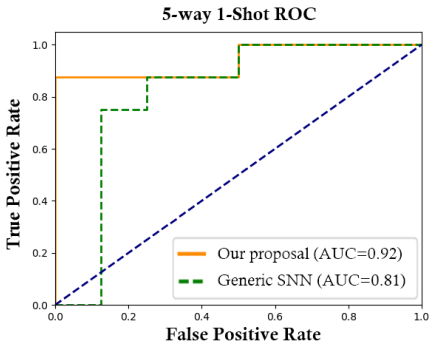
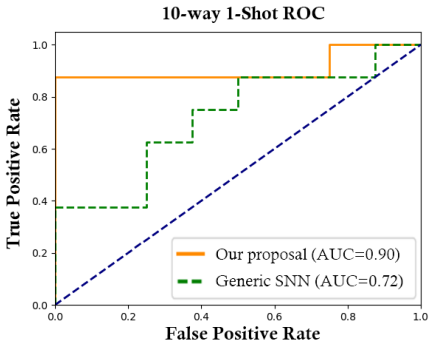


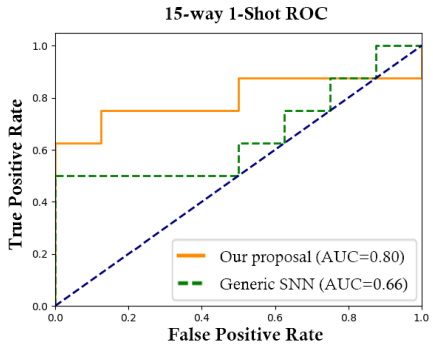
Figure 13. One-shot performance with hyperparameter changing.



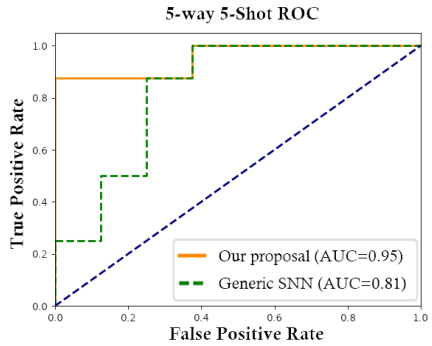
(**a**) ROC of 5-way one-shot



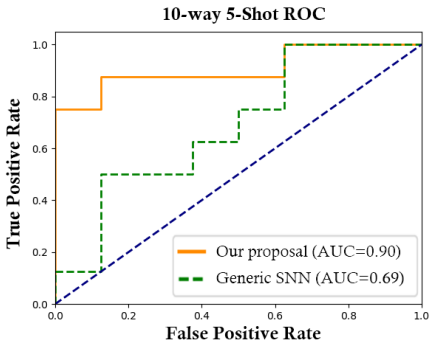
(**b**) ROC of 10-way one-shot



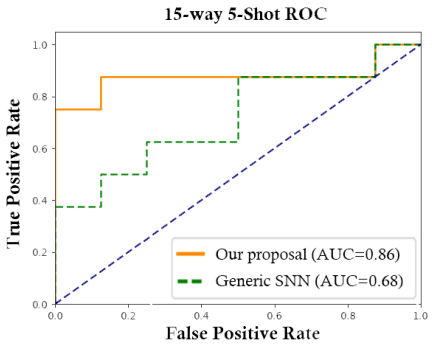
(**c**) ROC of 15-way one-shot



(**d**) ROC of 5-way five-shot



(**e**) ROC of 10-way five-shot



(**f**) ROC of 15-way five-shot

Figure 14. AUC–ROC curves under the N-way N-shot.

As shown, our result is on a set of points in the true positive rate–false positive rate plane. The results achieved an AUC–ROC equal to 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 five-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 one-shot and five-shot dropped as the number of N-way increased with higher intra-class variance.

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

6. Conclusions

We propose 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 the CNNs used by our model has an additional network called the "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 fully connected feature layers, so that the feature embedding in each CNN is accurately adjusted for different malware families, despite there being obfuscated malware variants in each malware family.

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 compute 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.

Our experimental results show that our proposed model is highly effective in recognizing the presence of unique malware signatures, and is 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 [55] and ransomware families [56–59]) and other data samples (e.g., finding similar abnormalities in medical X-ray images [60]), to test the generalizability of our model.

Author Contributions: Conceptualization, J.Z., J.J.-J. and A.S.; methodology, J.Z., J.J.-J. and A.S.; validation, J.Z., J.J.-J. and A.S.; formal analysis, J.Z. and A.S.; investigation, J.Z., P.A.W. and S.C.; writing—original draft preparation, J.Z. and J.J.-J.; writing—review and editing, J.Z., J.J.-J., P.A.W. and S.C.; visualization, J.Z. and A.S.; supervision, J.J.-J. and P.A.W.; funding acquisition, J.J.-J. and P.A.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Ministry of Business, Innovation and Employment: MAUX1912.

Data Availability Statement: Restrictions apply to the availability of these data. Data were obtained with the permission of its author of Andro-Dumpsys.

Acknowledgments: This research was supported by the Cyber Security Research Programme—Artificial Intelligence for Automating Response to Threats, from the Ministry of Business, Innovation, and Employment (MBIE) of New Zealand, as a part of the Catalyst Strategy Funds under grant number MAUX1912.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dong, S.; Li, M.; Diao, W.; Liu, X.; Liu, J.; Li, Z.; Xu, F.; Chen, K.; Wang, X.; Zhang, K. Understanding android obfuscation techniques: A large-scale investigation in the wild. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Singapore, 8–10 August 2018; Springer: Cham, Switzerland, 2018; pp. 172–192.
2. Chua, M.; Balachandran, V. Effectiveness of android obfuscation on evading anti-malware. In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, 19–21 March 2018; pp. 143–145.
3. Bacci, A.; Bartoli, A.; Martinelli, F.; Medvet, E.; Mercaldo, F. Detection of obfuscation techniques in Android applications. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–9.
4. Song, L.; Tang, Z.; Li, Z.; Gong, X.; Chen, X.; Fang, D.; Wang, Z. Appis: Protect android apps against runtime repackaging attacks. In Proceedings of the 2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS), Shenzhen, China, 15–17 December 2017; pp. 25–32.
5. Lee, Y.; Woo, S.; Lee, J.; Song, Y.; Moon, H.; Lee, D.H. Enhanced Android app-repackaging attack on in-vehicle network. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 5650245. [CrossRef]
6. Zheng, X.; Pan, L.; Yilmaz, E. Security analysis of modern mission critical android mobile applications. In Proceedings of the ACSW 2017: Australasian Computer Science Week 2017, Geelong, Australia, 30 January–3 February 2017; pp. 1–9.
7. Zhu, H.J.; You, Z.H.; Zhu, Z.X.; Shi, W.L.; Chen, X.; Cheng, L. DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* **2018**, *272*, 638–646. [CrossRef]
8. Sun, B.; Li, Q.; Guo, Y.; Wen, Q.; Lin, X.; Liu, W. Malware family classification method based on static feature extraction. In Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 13–16 December 2017; pp. 507–513.
9. Hu, X.; Griffin, K.E.; Bhatkar, S.B. Encoding Machine Code Instructions for Static Feature Based Malware Clustering. U.S. Patent 8,826,439, 2 September 2014.
10. Vasan, D.; Alazab, M.; Wassan, S.; Safaei, B.; Zheng, Q. Image-based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* **2020**, *92*, 101748. [CrossRef]
11. Luo, J.S.; Lo, D.C.T. Binary malware image classification using machine learning with local binary pattern. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 4664–4667.
12. Su, J.; Vasconcellos, D.V.; Prasad, S.; Sgandurra, D.; Feng, Y.; Sakurai, K. Lightweight classification of IoT malware based on image recognition. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 2, pp. 664–669.
13. Makandar, A.; Patrot, A. Trojan malware image pattern classification. In Proceedings of the International Conference on Cognition and Recognition, London, UK, 27–30 June 2016; Springer: Singapore, 2018; pp. 253–262.
14. Hsiao, S.C.; Kao, D.Y.; Liu, Z.Y.; Tso, R. Malware image classification using one-shot learning with Siamese networks. *Procedia Comput. Sci.* **2019**, *159*, 1863–1871. [CrossRef]
15. Singh, A.; Dutta, D.; Saha, A. Migan: Malware image synthesis using gans. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 10033–10034.
16. Raff, E.; Zak, R.; Cox, R.; Sylvester, J.; Yacci, P.; Ward, R.; Tracy, A.; McLean, M.; Nicholas, C. An investigation of byte n-gram features for malware classification. *J. Comput. Virol. Hacking Tech.* **2018**, *14*, 1–20. [CrossRef]
17. Gibert, D.; Mateu, C.; Planes, J. A hierarchical convolutional neural network for malware classification. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
18. Shen, J.; Cao, X.; Li, Y.; Xu, D. Feature adaptation and augmentation for cross-scene hyperspectral image classification. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 622–626. [CrossRef]
19. Gibert, D.; Mateu, C.; Planes, J.; Vicens, R. Classification of malware by using structural entropy on convolutional neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–3 February 2018; Volume 32.
20. Akarsh, S.; Poornachandran, P.; Menon, V.K.; 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: Cham, Switzerland, 2019; pp. 241–286.
21. Ni, S.; Qian, Q.; Zhang, R. Malware identification using visualization images and deep learning. *Comput. Secur.* **2018**, *77*, 871–885. [CrossRef]
22. Naeem, H.; Ullah, F.; Naeem, M.R.; Khalid, S.; Vasan, D.; Jabbar, S.; Saeed, S. Malware detection in industrial internet of things based on hybrid image visualization and deep learning model. *Ad Hoc Netw.* **2020**, *105*, 102154. [CrossRef]
23. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.; Wang, Y.; Iqbal, F. Malware classification with deep convolutional neural networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5.
24. Milosevic, N.; Dehghantanha, A.; Choo, K.K.R. Machine learning aided Android malware classification. *Comput. Electr. Eng.* **2017**, *61*, 266–274. [CrossRef]
25. Yuan, B.; Wang, J.; Liu, D.; Guo, W.; Wu, P.; Bao, X. Byte-level malware classification based on markov images and deep learning. *Comput. Secur.* **2020**, *92*, 101740. [CrossRef]

26. Cao, J.; Su, Z.; Yu, L.; Chang, D.; Li, X.; Ma, Z. Softmax cross entropy loss with unbiased decision boundary for image classification. In Proceedings of the 2018 Chinese Automation Congress (CAC), Xi'an, China, 30 November–2 December; pp. 2028–2032.
27. Huang, S.; Tran, D.N.; Tran, T.D. Sparse signal recovery based on nonconvex entropy minimization. In Proceedings of the 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 25–28 September 2016; pp. 3867–3871.
28. Finlayson, G.D.; Drew, M.S.; Lu, C. Entropy minimization for shadow removal. *Int. J. Comput. Vis.* **2009**, *85*, 35–57. [CrossRef]
29. Kolouri, S.; Rostami, M.; Owechko, Y.; Kim, K. Joint dictionaries for zero-shot learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–3 February 2018; Volume 32.
30. Allahverdyan, A.E.; Galstyan, A.; Abbas, A.E.; Struzik, Z.R. Adaptive decision making via entropy minimization. *Int. J. Approx. Reason.* **2018**, *103*, 270–287. [CrossRef]
31. Yang, A.; Lu, C.; Li, J.; Huang, X.; Ji, T.; Li, X.; Sheng, Y. Application of meta-learning in cyberspace security: A survey. *Digit. Commun. Netw.* **2023**, *9*, 67–78. [CrossRef]
32. Zoppi, T.; Gharib, M.; Atif, M.; Bondavalli, A. Meta-learning to improve unsupervised intrusion detection in cyber-physical systems. *ACM Trans. Cyber-Phys. Syst. (TCPS)* **2021**, *5*, 1–27. [CrossRef]
33. Zoppi, T.; Ceccarelli, A.; Puccetti, T.; Bondavalli, A. Which Algorithm can Detect Unknown Attacks? Comparison of Supervised, Unsupervised and Meta-Learning Algorithms for Intrusion Detection. *Comput. Secur.* **2023**, *127*, 103107. [CrossRef]
34. Zhou, X.; Liang, W.; Shimizu, S.; Ma, J.; Jin, Q. Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems. *IEEE Trans. Ind. Inform.* **2020**, *17*, 5790–5798. [CrossRef]
35. Sun, G.; Qian, Q. Deep learning and visualization for identifying malware families. *IEEE Trans. Dependable Secur. Comput.* **2018**, *18*, 283–295. [CrossRef]
36. Moustakidis, S.; Karlsson, P. A novel feature extraction methodology using Siamese convolutional neural networks for intrusion detection. *Cybersecurity* **2020**, *3*, 1–13. [CrossRef]
37. Tang, Z.; Wang, P.; Wang, J. ConvProtoNet: Deep Prototype Induction towards Better Class Representation for Few-Shot Malware Classification. *Appl. Sci.* **2020**, *10*, 2847. [CrossRef]
38. Zhang, B.; Xiao, W.; Xiao, X.; Sangaiah, A.K.; Zhang, W.; Zhang, J. Ransomware classification using patch-based CNN and self-attention network on embedded N-grams of opcodes. *Future Gener. Comput. Syst.* **2020**, *110*, 708–720. [CrossRef]
39. Ng, C.K.; Jiang, F.; Zhang, L.Y.; Zhou, W. Static malware clustering using enhanced deep embedding method. *Concurr. Comput. Pract. Exp.* **2019**, *31*, e5234. [CrossRef]
40. Hashemi, H.; Azmoodeh, A.; Hamzeh, A.; Hashemi, S. Graph embedding as a new approach for unknown malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 153–166. [CrossRef]
41. Pektaş, A.; Acarman, T. Deep learning for effective Android malware detection using API call graph embeddings. *Soft Comput.* **2020**, *24*, 1027–1043. [CrossRef]
42. Chen, L.; Sahita, R.; Parikh, J.; Marino, M. STAMINA: Scalable Deep Learning Approach for Malware Classification. Intel Labs Whitepaper. 2020. Available online: https://www.intel.com/content/www/us/en/artificial-intelligence/documents/stamina-deep-learningfor-malware-protection-whitepaper.html (accessed on 20 June 2021).
43. Li, X.; Qiu, K.; Qian, C.; Zhao, G. An Adversarial Machine Learning Method Based on OpCode N-grams Feature in Malware Detection. In Proceedings of the 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC), Hong Kong, China, 27–29 July 2020; pp. 380–387.
44. Zhu, J.; Jang-Jaccard, J.; Liu, T.; Zhou, J. Joint Spectral Clustering based on Optimal Graph and Feature Selection. *Neural Process. Lett.* **2021**, *53*, 257–273. [CrossRef]
45. Tran, T.K.; Sato, H.; Kubo, M. Image-Based Unknown Malware Classification with Few-Shot Learning Models. In Proceedings of the 2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW), Nagasaki, Japan, 26–29 November 2019; pp. 401–407.
46. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
47. Gidaris, S.; Komodakis, N. Dynamic few-shot visual learning without forgetting. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 4367–4375.
48. Wen, Y.; Zhang, K.; Li, Z.; Qiao, Y. A discriminative feature learning approach for deep face recognition. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Cham, Switzerland, 2016, pp. 499–515.
49. Jang, J.-w.; Kang, H.; Woo, J.; Mohaisen, A.; Kim, H.K. Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information. *Comput. Secur.* **2016**, *58*, 125–138. [CrossRef]
50. Zhu, J.; Jang-Jaccard, J.; Watters, P.A. Multi-Loss Siamese Neural Network with Batch Normalization Layer for Malware Detection. *IEEE Access* **2020**, *8*, 171542–171550. [CrossRef]
51. Malvar, H.S.; He, L.W.; Cutler, R. High-quality linear interpolation for demosaicing of bayer-patterned color images. In Proceedings of the 2004 IEEE International Conference on Acoustics, Speech, and Signal Processing, Montreal, QC, Canada, 17–21 May 2004; Volume 3, pp. iii–485.
52. Koch, G.; Zemel, R.; Salakhutdinov, R. Siamese neural networks for one-shot image recognition. In Proceedings of the ICML Deep Learning Workshop, Lille, France, 6–11 July 2015; Volume 2.
53. Vinyals, O.; Blundell, C.; Lillicrap, T.; Kavukcuoglu, K.; Wierstra, D. Matching networks for one shot learning. *arXiv* **2016**, arXiv:1606.04080.

54. Snell, J.; Swersky, K.; Zemel, R.S. Prototypical networks for few-shot learning. *arXiv* **2017**, arXiv:1703.05175.
55. Wei, Y.; Jang-Jaccard, J.; Sabrina, F.; Singh, A.; Xu, W.; Camtepe, S. Ae-mlp: A hybrid deep learning approach for ddos detection and classification. *IEEE Access* **2021**, *9*, 146810–146821. [CrossRef]
56. Zhu, J.; Jang-Jaccard, J.; Singh, A.; Watters, P.A.; Camtepe, S. Task-aware meta learning-based siamese neural network for classifying obfuscated malware. *arXiv* **2021**, arXiv:2110.13409.
57. Zhu, J.; Jang-Jaccard, J.; Singh, A.; Welch, I.; AI-Sahaf, H.; Camtepe, S. A Few-Shot Meta-Learning based Siamese Neural Network using Entropy Features for Ransomware Classification. *arXiv* **2021**, arXiv:2112.00668.
58. McIntosh, T.R.; Jang-Jaccard, J.; Watters, P.A. Large scale behavioral analysis of ransomware attacks. In Proceedings of the International Conference on Neural Information Processing, Siem Reap, Cambodia, 13–16 December 2018; Springer: Cham, Switzerland, 2018; pp. 217–229.
59. McIntosh, T.; Jang-Jaccard, J.; Watters, P.; Susnjak, T. The inadequacy of entropy-based ransomware detection. In Proceedings of the International Conference on Neural Information Processing, Sydney, Australia, 12–15 December 2019; Springer: Cham, Switzerland, 2019; pp. 181–189.
60. 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. *J. Med. Imaging Radiat. Oncol.* **2022**, *6*, 1035–1043. [CrossRef]