

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

Jinting Zhu*, Julian Jang-Jaccard*, Ian Welch†, Harith AI-Sahaf†, Seyit Camtepe‡ and Aeryn Dunmore*

*Cybersecurity Lab, Massey University, New Zealand

†Email: jzhu3@massey.ac.nz

†School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

‡CSIRO Data61, Australia

Abstract—When malware employs an unseen zero-day exploit, traditional security measures such as vulnerability scanners and antivirus software can fail to detect them. This is because these tools rely on known patches and signatures, which do not exist for new zero-day attacks. Furthermore, existing machine learning methods, which are trained on specific and occasionally outdated malware samples, may struggle to adapt to features in new malware. To address this issue, there is a need for a more robust machine learning model that can identify relationships between malware samples without being trained on a particular malware feature set. This is particularly crucial in the field of cybersecurity, where the number of malware samples is limited and obfuscation techniques are widely used. Current approaches using stacked autoencoders aim to remove the noise introduced by obfuscation techniques through reconstruction of the input. However, this approach ignores the semantic relationships between features across different malware samples. To overcome this limitation, we propose a novel Siamese Neural Network (SNN) that uses relation-aware embeddings to calculate more accurate similarity probabilities based on semantic details of different malware samples. In addition, by using entropy images as inputs, our model can extract better structural information and subtle differences in malware signatures, even in the presence of obfuscation techniques. Evaluations on two large malware sample sets using the N-shot and N-way methods show that our proposed model is highly effective in predicting previously unseen malware, even in the presence of obfuscation techniques.

I. 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.

Malware detection refers to identifying whether a given piece of software or a file is malicious or benign. In the context of zero-day attacks, malware detection faces significant challenges because these attacks use unknown vulnerabilities and novel techniques that are not recognized by traditional detection systems. Detection methods often rely on known signatures, patterns, or behaviours that have been previously identified and documented. However, in the case of zero-day

attacks, which exploit the unknown or unaddressed vulnerability is referred to as a zero-day vulnerability or zero-day threat, as these signatures or patterns are not available, making detection difficult.

Given the popularity of Artificial Intelligence (AI) techniques, their application in detecting zero-day or unknown malware has become increasingly sophisticated, particularly with the integration of few-shot learning mechanisms. Few-shot learning, a subset of machine learning, enables AI systems to recognize new patterns or anomalies with minimal data examples. This is especially crucial for identifying zero-day threats, as these types of malware are often not well-represented in large datasets. By using few-shot learning, AI algorithms can quickly adapt to and recognize new, previously unseen forms of malware with very few examples, significantly enhancing the speed and efficiency of threat detection. This approach is a game-changer in cybersecurity, offering robust defenses against emerging and rapidly evolving digital threats.

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 [1] and entropy graphs [2], [3]. 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 [4]. 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, [5] and [6] proposed enhancing the feature representation methods through an autoencoder. But they were still reported to be vulnerable to many types of unseen samples [7], [8].

Due to the cost of dynamic feature analysis, more practitioners have defaulted to static feature analysis as it provides higher levels of efficiency [9]. 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 [10]. 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 [11]. 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; [Entropy is employed to identify anomalies, unusual patterns, or irregular behaviours in data, often indicative of malicious activity. Malware and cyber-attacks typically manifest through such atypical data patterns, which are effectively discernible via entropy-based analysis. Additionally, this method demonstrates resilience against noise introduced by malware developers, enhancing its efficacy in cybersecurity applications](#)

Malware developers use obfuscation techniques to reduce the likelihood of detection [12]. 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 [13], [14], we can uncover groups of correlated malware with common features [15], [16], [17]. Sharma et al. [11] found that the underlying physical processes behind converting malware binaries to images are highly non-linear [11], and this correlation exists between malware instances. Other works tend to focus on either the correlation between the malware samples [18] 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. More-

over, 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 II. Section III introduces the preliminary materials in the obfuscation technique. Section IV provides the method for constructing our model and the details of our network architecture. In Section V, we provide an ablation experiment to clarify our model's ability and provide multiple analyses for hyperparameters. Finally, this paper concludes in Section VI, which provides a summary, conclusion, and potential directions for future work.

II. RELATED WORK

1) *Relational exploring for malware detection:* Reveal-Droid [19] 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. [20] 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 behavior exists universally in this field. Zheng et al. [21] applied a meta-learning approach to a classification task of encrypted traffic and to classify unseen categories based on a few labeled samples. Han et al. [22] 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. [23] 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. [24] 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.

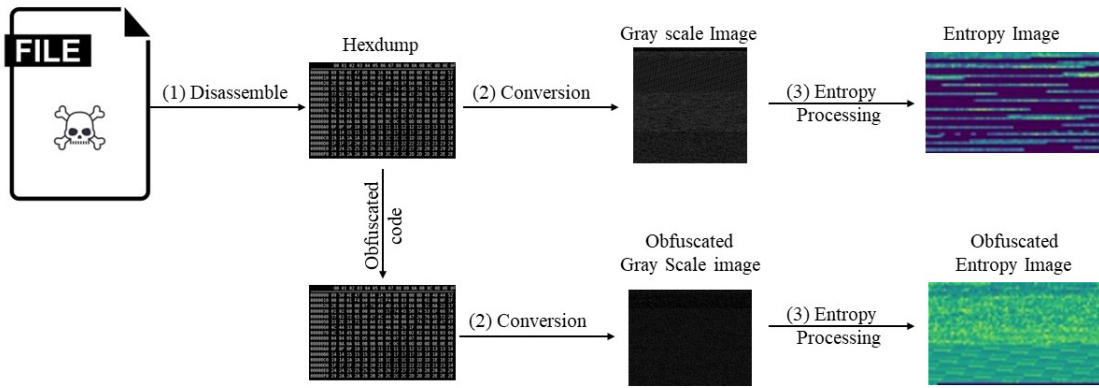


Fig. 1: Feature Processing for Entropy Image

2) *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 [25], [26], [27], [28], [29], though the hybrid features [30], [31], [32] 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 [33], [34]. This approach is similar to the research proposed by Kang et al. [35], which captured similar information through the bytecode frequency. Additionally to the use of n-gram features for the static analysis, Nataraj et al. [36] 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. [37] 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. [38] 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

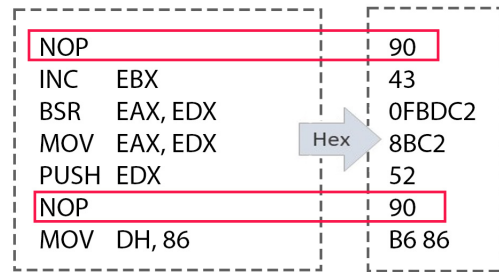


Fig. 2: Obfuscated Code of No Operation (NOP) Insertion

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 [39], [40] and potentially leading to failure [41] in the virtual environment.

3) *Denosing autoencoder for malware detection:* One method of unsupervised learning within the research consists of an autoencoder architecture [42], [43], [44], which can learn the reconstructed features from noised samples. Alahmadi et al. [45] 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. [46] 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. [5] 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.

III. OBFUSCATION MATERIALS

A. 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. [47] 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 [48], [47]. 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: $M_{app} = m_1, m_{mod_2}, m_3, m_{add_1}, m_{add_2}, \dots, m_n$, 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 [49]. To visualize this procedure, Fig. 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.

B. Obfuscation of No Operation (NOP) Insertion

No-op instruction (NOP) obfuscation, as shown in Fig 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 while preserving the semantic information [48], [50]. It is possible to evade malware detection solutions employing opcode sequences as malware signatures,

through repeats of randomly inserting the bytecode of NOP multiple times at ambiguous positions in the hexdump form of a malware sample.

IV. METHODOLOGY

Our approach involves a few-shot classification task using a Siamese denoising autoencoder against junk code obfuscation. 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. 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, as shown in Fig 4. 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.

A. The Entropy Feature Conversion

Entropy feature conversion aims to measure the uncertainty distribution of the information. Moreover, Vidya et al. [51] 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 [52], [53], 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 [29]. 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. 3 shows. The entropy images are comprised of global and local entropy information that can be calculated with the Shannon entropy equation below:

$$Ent = - \sum_i \sum_j M(i, j) \log M(i, j) \quad (1)$$

where M is the probability of an occurrence of a byte value. The Shannon entropy equation obtains the minimum value of 0 when all bytecode of the malware have no changes. Alternatively, the maximum entropy value of byte value 8 is obtained when all the bytecode is different. If specific information of bytecode occurs with high probabilities, the entropy value will be smaller. To visualize the entropy pattern that reflects the pattern of entropy information and the intrinsic

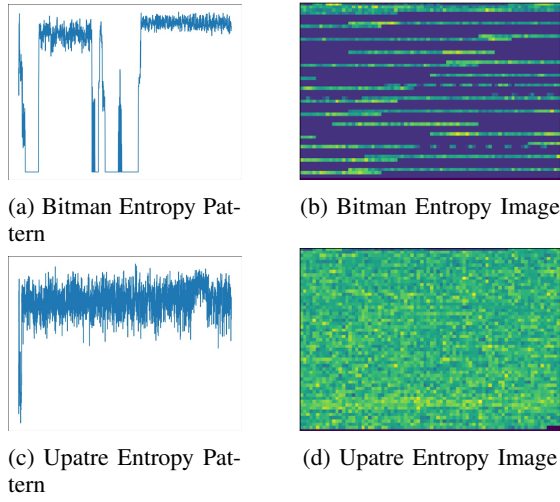


Fig. 3: Entropy pattern corresponding to entropy image

connection of each sequence of raw bytecode, we built a grey-scale matrix through Equation 2, and the entropy values are then concatenated into a stream of values that can form an entropy sequence.

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

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 .

B. Denoising AutoEncoder

In cybersecurity research, denoising encoders have been widely used in recent years [42], [54], [55]. 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 [56], occluded faces [57], 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:

$$z_i^{(m,v)} = a(W_{ae}^{m,v} z_i^{m-1,v} + b_{ae}^{(m,v)}) \quad (3)$$

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 $X^{(p)} = [x_1^p, x_2^p, \dots, x_3^p], \in \mathbb{R}^{d_p \times n}$ for one of the sub-networks.

Meanwhile, the corresponding reconstructed representation is denoted as:

$$Z^{(M,v)} = [z_1^{(M,v)}, z_2^{(M,v)}, \dots, z_3^{(M,v)}] \quad (4)$$

where $z_i^{(M,v)}$ is the reconstructed representation for the i th 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):

$$\mathcal{L}_{mse} = \frac{1}{2n} \sum_{i=1}^n \|X^{(p)} - Z^{(M,v)}\|_F^2 \quad (5)$$

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

C. Siamese-based Denoising AutoEncoder

Our model contains a Siamese neural network (SNN) that is commonly used in contrastive learning. The SNN model is useful on 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-network 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:

$$d_\varphi(z_i, z_j) = \|g_\phi(z_i^{(\frac{M}{2},v)}) - g_\phi(z_j^{(\frac{M}{2},v)})\|_F^2 \quad (6)$$

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:

$$\mathcal{L} \sum_{i=1}^p L(\varphi, (z_i^{(\frac{M}{2},v)}, z_j^{(\frac{M}{2},v)})^i) \quad (7)$$

$$L(\varphi) = (1 - y_i)L_s(d_\varphi) + y_i L_d(d_\varphi) \quad (8)$$

The classification ability depends on the performance of d_φ calculated by Eq. 8, which linearly represents the Euclidean distance.

D. Siamese in Relation-Aware Embedding for Malware Classification

Strategies such as [58], [59] only take the independent malware signature into account, or rarely consider the non-linear relationship [60], [61] 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.

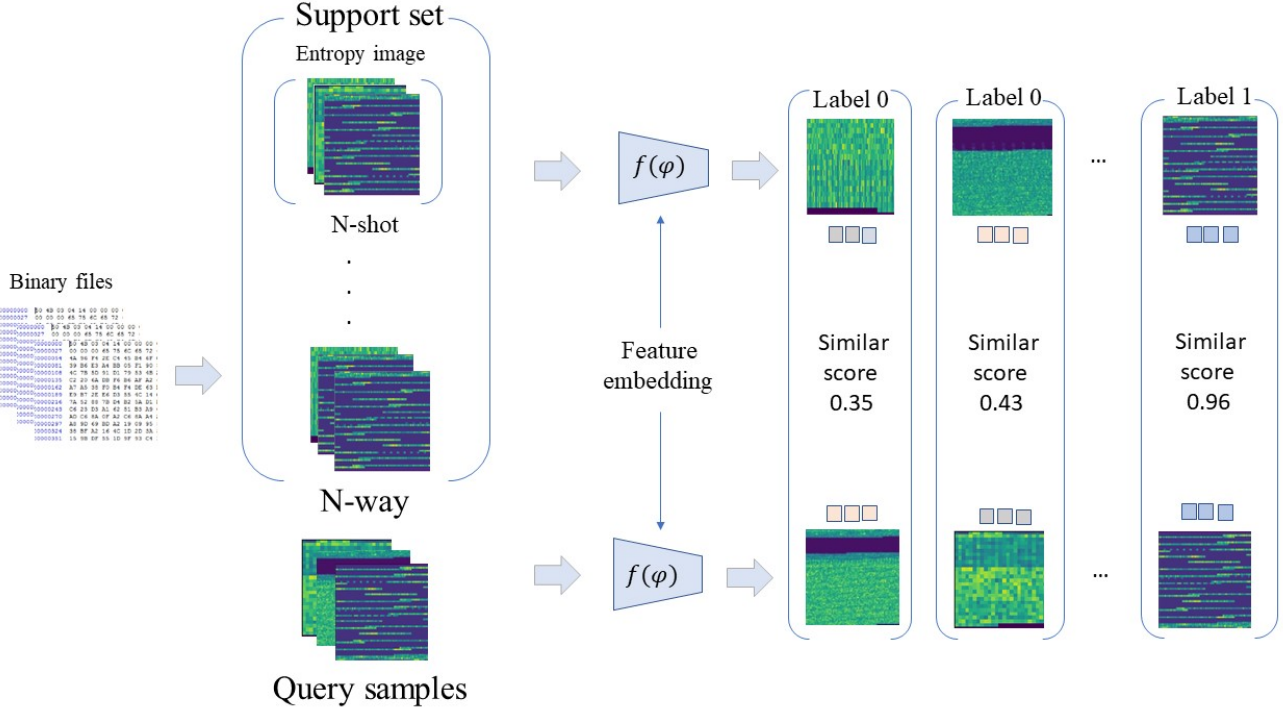


Fig. 4: The diagram of Few-shot learning

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 [62], [63], [64].

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.

$$r_{i,j} = \mathcal{C}(f_{\varphi}(z_i^{(\frac{M}{2},v)}), f_{\varphi}(z_j^{(\frac{M}{2},v)})), \quad i = 1, 2, \dots, C \quad (9)$$

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} .

$$\mathcal{L}_r = \sum_{i=1}^i \sum_{j=1}^j (r_{i,j} - y_{i,j})^2 \quad (10)$$

Given the set of latent features $\mathbf{z} = \{z_1, z_2, \dots, z_n\}$ 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:

$$\mathcal{L} = \mathcal{L}_r + \lambda \mathcal{L}_{mse} \quad (11)$$

where the value of hyperparameter λ settled as 0.7.

E. 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 for increasing the number of parameters. We put convolutional layers first and stack the fully-connected layers on top of convolutional layers for ease of the reduction to the target dimension \mathbb{R}^d .

The block diagram of the proposed image compression based in Fig. 5 shows the architecture of the layers, stacking up to 4 convolutional and batch-normalization layers. The only preprocessing step before the CAE network consists of normalizing the entropy pattern to $[-1, 1]$ by calculating the value of mean and standard deviation of 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 due to the conversion to grey scale images.

In term of network design, a siamese neural network is more robust to class imbalance [65] as it focuses on learning embeddings (in the deeper layer) that place the same classes/concepts close together in order to learn semantic similarity. We have also considered the term of the receptive field of the topmost feature layer in each sub-branch, because if the receptive field

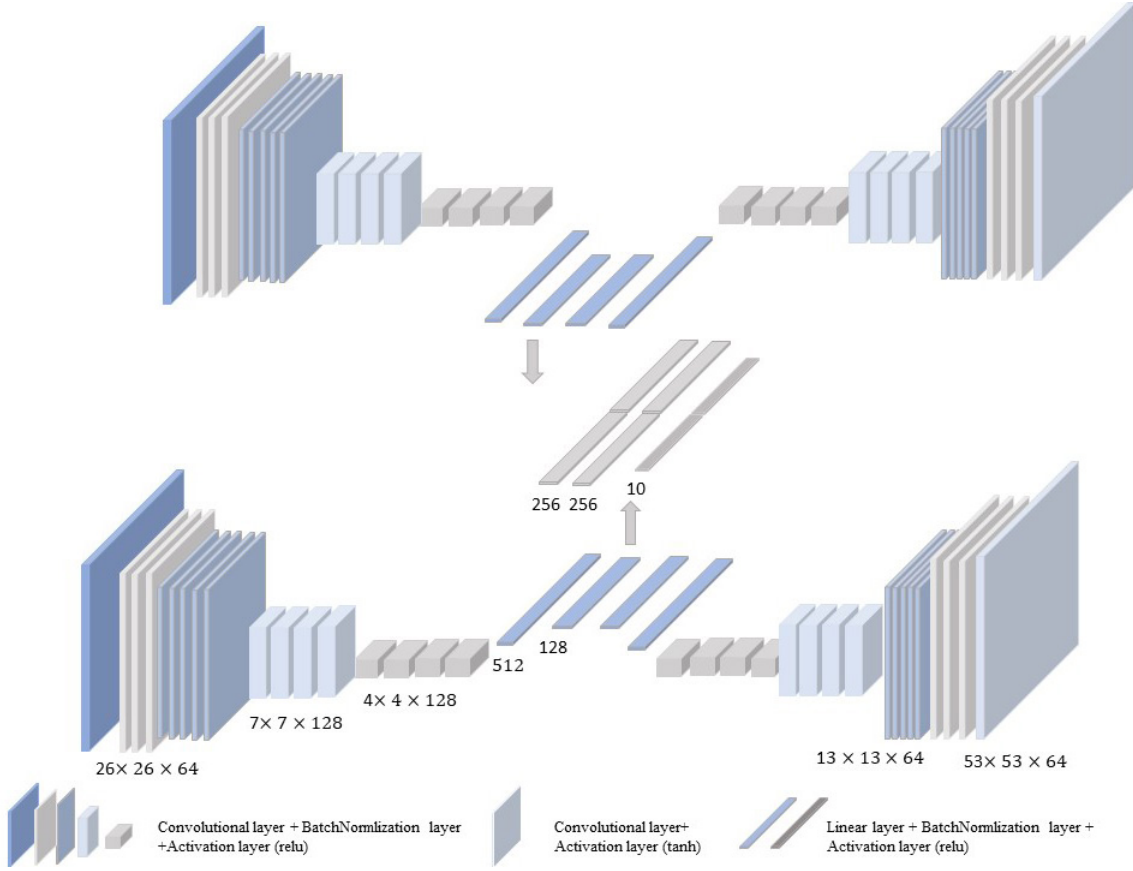


Fig. 5: The diagram of our model architecture

is too large it means that there are too many layers, and the risk of network overfitting will increase. Meanwhile, the entire network is difficult to converge.

Algorithm 1

By considering the Siamese theory of conception, the encoder part of our model is designed with 4 convolution layers in which each layer is followed by a batch normalization layer and relu activation layer. All convolutional filters have size of 3×3 and use a stride of 2 for the encoder part. The order of the CNN layers in the decoder part is opposite to the encoder part. A max pool layer is added after the first convolution layer. Two linear layers are used to improve the classification performance with the extracted features through the encoder part, and it is the key module for relational embedding. The extracted features have are robust against the NOP obfuscation as they have been training with autoencoder module. We further compare the performance with the different dimensions of the linear layer and find that the output dimension of 256 for the first linear layer has better performance than that with a dimension of 128. Finally, the sigmoid activation function is taken as the outcome function for relation label 1 or 0.

V. EXPERIMENTS

We evaluated the performance of our proposed model on two malware datasets according to the few-shot learning mechanism. We also conducted the ablation experiments to analyze

Algorithm 1: 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

- 1 Dataloader = FewShotTask(f, cc, np, bn)
- 2 for episoded in range(EPISODE)
- 3 $(x_b^1, x_b^2) = \text{Dataloader.iter}()$
- 4 $en_1, en_2 = \text{Siamese_EncoderNetwork}(x_b^1, x_b^2)$
- 5 $dn_1, dn_2 = \text{Siamese_DecoderNetwork}(en_1, en_2)$
- 6 Pairs $(r_n) = \text{Feature_concatenation}((en_1, en_2))$
- 7 Score $(s_z) = \text{Siamese_RelationNetwork}((r_n))$
- 8 $Loss_1 = RC([dn_1, en_1], [dn_2, en_2])$
- 9 $Loss_2 = RL(s_z)$
- 10 $Loss = 0.7 * Loss_1 + Loss_2$
- 11 $Loss.backward()$
- 12 $Optimizer.step()$

the effect of various parameters, such as pooling methods, layer dimension, and hyperparameters. The experiments were conducted on equipment consisting of 32GB RAM, Nvidia Geforce RTX 2070(8GB), and Intel i7-9700 CPU@ 3.00GHz.

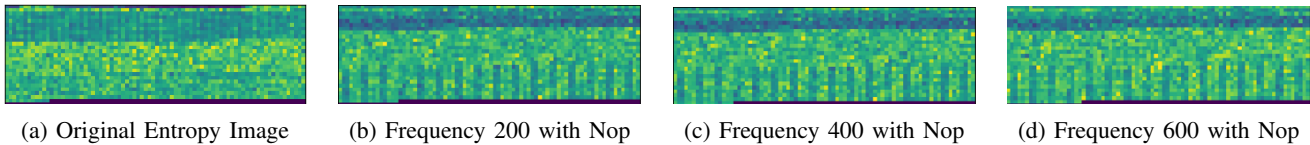


Fig. 6: The appearance comparison between original images and obfuscated images

A. Description of dataset

a) *VUW dataset*: This dataset utilized VirusShare¹ to collect malware samples. With the VUW dataset, we collected a total number of 1,048 samples from 11 families/classes of ransomware, each consisting of a varying number of examples, as listed in Table I. 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 I.

b) *Malimage dataset*: The details of the second malware dataset, Malimage, published in [36] are listed in Table II. There are 9,314 instances spread over 25 families in which the width and height of each malware image differ between families. Nataraj et al. [36] fixed the width to a certain length according to the file size and the bytecode sequence. Observed in the Table II, the number of samples is greater than the VUW dataset. The malware samples have distinctive image textures across the different malware families.

TABLE I: 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

B. Dataset setup and augmentation

The number of malware samples from different malware classes in Table I vary significantly. Almost half of the classes had no more than 25 malware samples, while some only had one, likely because they were new malware samples detected recently (e.g., Blocal and Newbak). We increased the image sample size to allow for at least 30 samples for every malware class using a data augmentation 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 were set to 0.52206 and 0.08426, as shown in Table III.

We performed NOP insertion on the original image at frequencies 200 times each. Observed from Fig 6, the same ransomware families have shown a similar entropy pattern. Fig. 6

¹VirusShare. <https://virusshare.com/>

TABLE II: 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

TABLE III: 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 hyperparameter for the loss function

shows the visual difference in the obfuscated malware images, which also intuitively differs from the original entropy image. From this change, we can conclude that static signatures can be easily tampered by using a simple obfuscation technique such as NOP. However, the appearance of entropy images at frequencies with 200, 400, 600 does not show obvious visual differences between them. We conducted experiments on the 200-frequency dataset to evaluate the performance of our model.

We measured 1-shot and 5-shot accuracy 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. The malware image size is fixed for each model.

TABLE IV: 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 [62]	55.8 ± 1.8%	55.2 ± 2.6%	42.6 ± 3.1%	43.2 ± 2.1%
Prototypical [66]	81.1 ± 2.2%	86.4 ± 1.9%	69.3 ± 2.2%	70.2 ± 2.0%
Prototypical [66] (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 [62]	63.2 ± 2.8%	68.1 ± 3.1%	56.8 ± 3.4%	58.2 ± 3.1%
Prototypical [66]	95.6 ± 1.9%	96.7 ± 1.8%	92.3 ± 2.3%	96.9 ± 2.1%
Prototypical [66] (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%

C. Training Details

We adopt an 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 labeled sample for each unique class in 5-way. For K -shot where $K > 1$, we calculate the element-wise average over the embedding module outputs of all samples from each training class to form this class’ feature map. The class-level average pooling is then concatenated with the query image feature. The number of query images is dependent on the batch size in each training episode. For example, 2-way 5-shot contains 19 query images, the 5-way 5-shot has 15 query images. The corresponding relationship between the support samples, query samples, and batch size can be expressed by the following equation,

$$\begin{aligned}
 S &= \{s^{(1)}, \dots, s^{(c)}, \dots, s^{(C)}\} \subset \mathbb{C}^{train}, s^{(c)} = K \\
 Q &= \{q^{(1)}, \dots, q^{(c)}, \dots, q^{(C)}\} \subset \mathbb{B}^{train}, q^{(c)} = N
 \end{aligned}
 \tag{12}$$

where c is the class index and K is the number of samples in class $s^{(c)}$; thus the training samples are constructed with $N \times K$ samples in each episode. Specifically, when we expand them, we can get the relationship below as:

$$\begin{aligned}
 r_{ij} &: \{(s^{(1)}, q^{(1)}), (s^{(1)}, q^{(1)}), (s^{(n)}, q^{(1)})\}, \dots \\
 &\{(s^{(1)}, q^{(c)}), (s^{(c)}, q^{(c)}), (s^{(n)}, q^{(c)})\}, \dots \\
 &\{(s^{(1)}, q^{(n)}), (s^{(c)}, q^{(n)}), (s^{(n)}, q^{(n)})\}
 \end{aligned}
 \tag{13}$$

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 set 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 to 105×105 and the results were gathered randomly 10 times. In one set the whole model ran 20000 epochs in the training stage and then was tested over 2000 epochs. The batch size of 19 and 15 were

for the 1-shot and 5-shot respectively. Our model shared the initialized learning rate with the value of 0.02. Meanwhile, the hyperparameter for λ in the loss function was 0.7.

D. Experimental Results on VUW Dataset

As seen in Table IV, we evaluated our model on two different features, the grayscale feature, and the entropy image. 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 prototype also has greater improvement with the entropy image on the 2-way and 5-way result than with the grayscale feature, which result increased by 7.1%. The reason that our model and prototype had better performance than other models is that the representation of entropy images which exhibits higher uniqueness and entropy values incorporated into local regions with higher information content. It is worth noting that instead of taking the sum pooling for the feature concatenation like a relation network [62] does, we took the mean pooling for the few-shot learning. We further compared our performance with the prototype network under the obfuscated entropy image, and our model still showed a significant success on the 2-way results, which at 81.7% and 85.2% are higher than the prototype network by 1.4% and 0.4%, respectively.

E. Experimental Results On Malimage Dataset

It is worth noting that the intra-class variance across all samples within the Malimage dataset significantly decreased when compared with the VUW dataset. The feature representation can give better performance on the condition that there is a low level of intra-class variance. Our model still achieves the best performance on the 2-way result 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 to 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 prototype network, which decreased 3.7% and 2.0%. The same situation

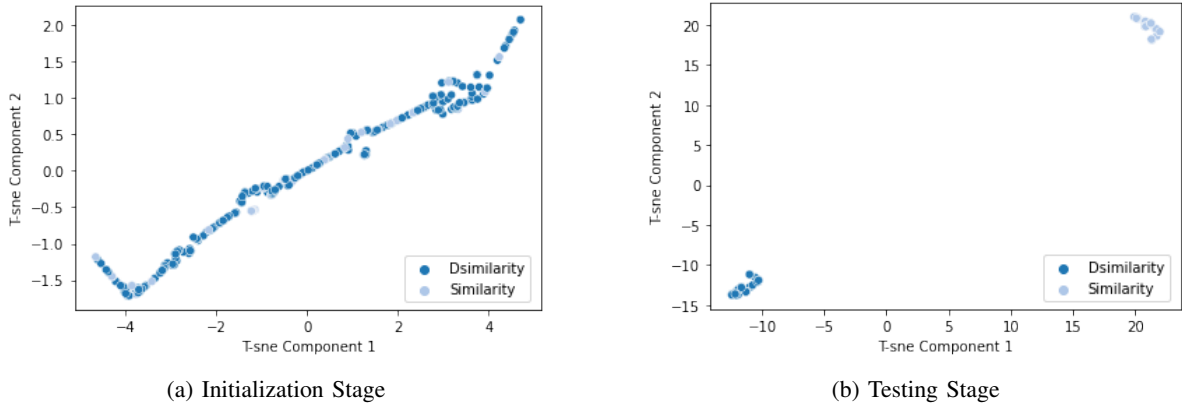


Fig. 7: t-distributed stochastic neighbor embedding (t-SNE) visualisation of embeddings generated using our model

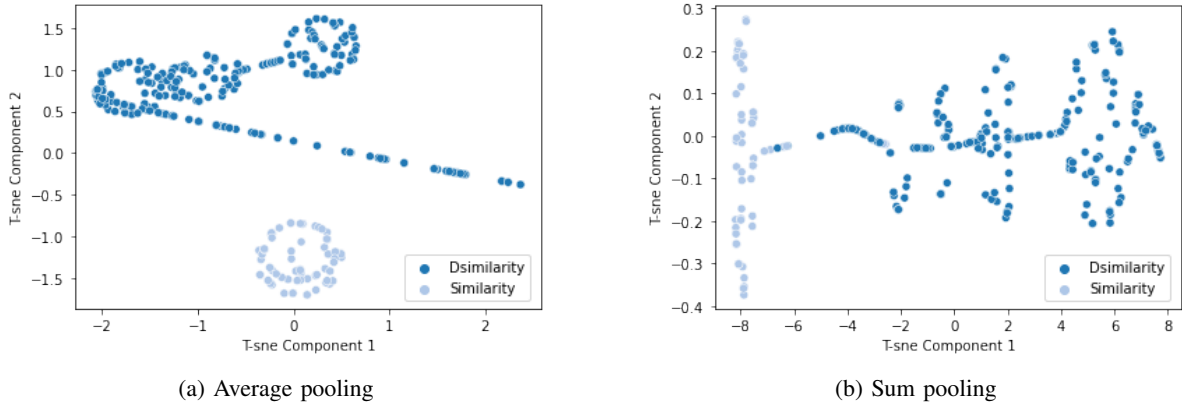


Fig. 8: t-distributed stochastic neighbor embedding (t-SNE) visualization of embeddings generated using our model

happens on the 5-way performance, which resulted in the obfuscated dataset offering a drop of only 0.9% on 5-way 1-shot. The performance on 5-way 5-shot only has a slight difference in result, whereas the prototype network decreased by 6.9% and 11.2% on the 5-way 1 shot and 5-way 5-shot on the obfuscated dataset. From observation of subtle differences in the results, we can prove that our model has a positive effect on the anti-obfuscation technology.

F. Ablation Study

1) *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 we combine the dimensions of the linear layer in order to observe the changes in the results, as shown in Table V. 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 $\lambda = 0.3$. We analyzed the trend of different dimension combinations in the linear layer and the value of λ for the performance. It can be seen from the Table V, that as the value of λ increases while the dimension of the 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 with the dimension of 128.

TABLE V: 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

2) *Pooling methods for feature selection:* We present a t-SNE visualization of the feature embeddings generated using our model. Based on different pooling methods used, Fig. 7 (a) illustrates the overall data points added with NOP obfuscation in the support set and query set, as 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. 7 (b). The two clusters are represented as similarity and dissimilarity, respectively, which offer the inter- and intra-characteristics. The model can benefit from the average calculation in the feature embedding since these averaging calculations are less deviated from the center of the object, as shown in Fig .8. This figure clearly demonstrates that the data points in sum pooling are more dispersed, while

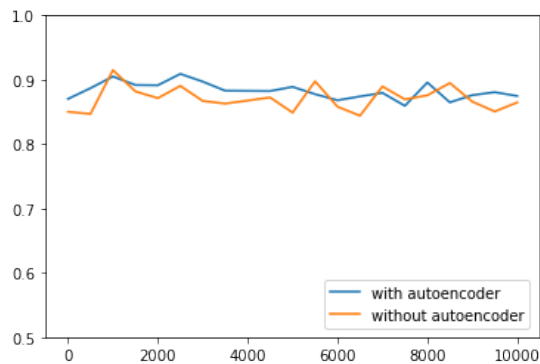


Fig. 9: The comparison with or without autoencoder

the data points in average pooling are more structured.

3) *Impact of autoencoder:* 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 9, the fluctuation range of the results obtained by the model without decoding is slightly larger than that with decoding while the average performance is decreased as well. The results without the decoder part are below those with the decoder part during the episodes ranging from 2000 to 6000. Moreover, its average result throughout the training phase is also slightly higher than that of the model without an autoencoder.

VI. 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 autoencoders, 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.

ACKNOWLEDGMENT

This research is 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 the grant number MAUX1912.

REFERENCES

- [1] S. Kumar and B. Janet, “Dtmic: Deep transfer learning for malware image classification,” *Journal of Information Security and Applications*, vol. 64, p. 103063, 2022.
- [2] J. Zhu, J. Jang-Jaccard, A. Singh, I. Welch, A.-S. Harith, and S. Camtepe, “A few-shot meta-learning based siamese neural network using entropy features for ransomware classification,” *Computers & Security*, vol. 117, p. 102691, 2022.
- [3] Y. Chai, J. Qiu, L. Yin, L. Zhang, B. B. Gupta, and Z. Tian, “From data and model levels: Improve the performance of few-shot malware classification,” *IEEE Transactions on Network and Service Management*, 2022.
- [4] V. Ravi, T. D. Pham, and M. Alazab, “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*, pp. 1–10, 2022.
- [5] O. Salman, I. H. Elhadj, A. Kayssi, and A. Chehab, “Denoising adversarial autoencoder for obfuscated traffic detection and recovery,” in *International conference on machine learning for networking*. Springer, 2019, pp. 99–116.
- [6] S. Kumar, S. Meena, S. Khosla, and A. S. Parihar, “Ae-dcnn: Autoencoder enhanced deep convolutional neural network for malware classification,” in *2021 International Conference on Intelligent Technologies (CONIT)*. IEEE, 2021, pp. 1–5.
- [7] U. Zahoor, M. Rajarajan, Z. Pan, and A. Khan, “Zero-day ransomware attack detection using deep contractive autoencoder and voting based ensemble classifier,” *Applied Intelligence*, vol. 52, no. 12, pp. 13941–13960, 2022.
- [8] S. Mahdavi, D. Alhadidi, and A. A. Ghorbani, “Effective and efficient hybrid android malware classification using pseudo-label stacked autoencoder,” *Journal of network and systems management*, vol. 30, pp. 1–34, 2022.
- [9] J. Saxe and K. Berlin, “Deep neural network based malware detection using two dimensional binary program features,” in *2015 10th international conference on malicious and unwanted software (MALWARE)*. IEEE, 2015, pp. 11–20.
- [10] R. Jusoh, A. Firdaus, S. Anwar, M. Z. Osman, M. F. Darmawan, and M. F. Ab Razak, “Malware detection using static analysis in android: a review of feco (features, classification, and obfuscation),” *PeerJ Computer Science*, vol. 7, p. e522, 2021.
- [11] P. Sharma and A. Raglin, “Efficacy of nonlinear manifold learning in malware image pattern analysis,” in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 1095–1102.
- [12] Z. Zhu, X. You, C. P. Chen, D. Tao, W. Ou, X. Jiang, and J. Zou, “An adaptive hybrid pattern for noise-robust texture analysis,” *Pattern Recognition*, vol. 48, no. 8, pp. 2592–2608, 2015.
- [13] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, “Relation networks for object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3588–3597.
- [14] S. Torabi, M. Dib, E. Bou-Harb, C. Assi, and M. Debbabi, “A strings-based similarity analysis approach for characterizing iot malware and inferring their underlying relationships,” *IEEE Networking Letters*, vol. 3, no. 3, pp. 161–165, 2021.
- [15] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, “Hadm: Hybrid analysis for detection of malware,” in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2016, pp. 702–724.

- [16] J.-Y. Kim and S.-B. Cho, "Obfuscated malware detection using deep generative model based on global/local features," *Computers & Security*, vol. 112, p. 102501, 2022.
- [17] X. Zhou, W. Liang, S. Shimizu, J. Ma, and Q. Jin, "Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5790–5798, 2020.
- [18] Y. Chai, L. Du, J. Qiu, L. Yin, and Z. Tian, "Dynamic prototype network based on sample adaptation for few-shot malware detection," *IEEE Transactions on Knowledge and Data Engineering*, 2022.
- [19] J. Tang, R. Li, Y. Jiang, X. Gu, and Y. Li, "Android malware obfuscation variants detection method based on multi-granularity opcode features," *Future Generation Computer Systems*, vol. 129, pp. 141–151, 2022.
- [20] Z. Xu, M. Li, Y. Hei, P. Li, and J. Liu, "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)*. IEEE, 2021, pp. 59–64.
- [21] W. Zheng, C. Gou, L. Yan, and S. Mo, "Learning to classify: A flow-based relation network for encrypted traffic classification," in *Proceedings of The Web Conference 2020*, 2020, pp. 13–22.
- [22] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *computers & security*, vol. 83, pp. 208–233, 2019.
- [23] S. D. Nikolopoulos and I. Polenakis, "Behavior-based detection and classification of malicious software utilizing structural characteristics of group sequence graphs," *Journal of Computer Virology and Hacking Techniques*, pp. 1–24, 2022.
- [24] A. Mpanti, S. D. Nikolopoulos, and I. Polenakis, "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.
- [25] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "Malinsight: A systematic profiling based malware detection framework," *Journal of Network and Computer Applications*, vol. 125, pp. 236–250, 2019.
- [26] M. Rabbani, Y. L. Wang, R. Khoshkangini, H. Jelodar, R. Zhao, and P. Hu, "A hybrid machine learning approach for malicious behaviour detection and recognition in cloud computing," *Journal of Network and Computer Applications*, vol. 151, p. 102507, 2020.
- [27] M. Noor, H. Abbas, and W. B. Shahid, "Countering cyber threats for industrial applications: An automated approach for malware evasion detection and analysis," *Journal of Network and Computer Applications*, vol. 103, pp. 249–261, 2018.
- [28] D. Morato, E. Berrueta, E. Magaña, and M. Izal, "Ransomware early detection by the analysis of file sharing traffic," *Journal of Network and Computer Applications*, vol. 124, pp. 14–32, 2018.
- [29] S. Wang, Z. Chen, Q. Yan, B. Yang, L. Peng, and Z. Jia, "A mobile malware detection method using behavior features in network traffic," *Journal of Network and Computer Applications*, vol. 133, pp. 15–25, 2019.
- [30] F. Tong and Z. Yan, "A hybrid approach of mobile malware detection in android," *Journal of Parallel and Distributed computing*, vol. 103, pp. 22–31, 2017.
- [31] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 1–12, 2017.
- [32] S. Yoo, S. Kim, S. Kim, and B. B. Kang, "Ai-hydra: Advanced hybrid approach using random forest and deep learning for malware classification," *Information Sciences*, vol. 546, pp. 420–435, 2021.
- [33] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, 2018.
- [34] S. Johnson, R. Gowtham, and A. R. Nair, "Ensemble model ransomware classification: A static analysis-based approach," in *Inventive Computation and Information Technologies*. Springer, 2022, pp. 153–167.
- [35] B. Kang, B. Kang, J. Kim, and E. G. Im, "Android malware classification method: Dalvik bytecode frequency analysis," in *Proceedings of the 2013 research in adaptive and convergent systems*, 2013, pp. 349–350.
- [36] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.
- [37] K. Bakour and H. M. Ünver, "Visdroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques," *Neural Computing and Applications*, vol. 33, no. 8, pp. 3133–3153, 2021.
- [38] J. Jeon, B. Jeong, S. Baek, and Y.-S. Jeong, "Hybrid malware detection based on bi-lstm and spp-net for smart iot," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 7, pp. 4830–4837, 2021.
- [39] S. Sharma, K. Khanna, and P. Ahlawat, "Survey for detection and analysis of android malware (s) through artificial intelligence techniques," in *Cyber Security and Digital Forensics*. Springer, 2022, pp. 321–337.
- [40] A. Furfaro, A. Piccolo, A. Parise, L. Argento, and D. Sacca, "A cloud-based platform for the emulation of complex cybersecurity scenarios," *Future Generation Computer Systems*, vol. 89, pp. 791–803, 2018.
- [41] A. Furfaro, L. Argento, A. Parise, and A. Piccolo, "Using virtual environments for the assessment of cybersecurity issues in iot scenarios," *Simulation Modelling Practice and Theory*, vol. 73, pp. 43–54, 2017.
- [42] A. De Paola, S. Favaloro, S. Gaglio, G. L. Re, and M. Morana, "Malware detection through low-level features and stacked denoising autoencoders," in *ITASEC*, 2018.
- [43] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.
- [44] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460, pp. 83–102, 2018.
- [45] A. Alahmadi, N. Alkhraan, and W. BinSaeedan, "Mpsautodetect: A malicious powershell script detection model based on stacked denoising auto-encoder," *Computers & Security*, vol. 116, p. 102658, 2022.
- [46] K. Sandra and S.-H. Lee, "Bm3d and deep image prior based denoising for the defense against adversarial attacks on malware detection networks," *International journal of advanced smart convergence*, vol. 10, no. 3, pp. 163–171, 2021.
- [47] J.-T. Chan and W. Yang, "Advanced obfuscation techniques for java bytecode," *Journal of systems and software*, vol. 71, no. 1-2, pp. 1–10, 2004.
- [48] P. Faruki, A. Bharmal, V. Laxmi, M. S. Gaur, M. Conti, and M. Rajarajan, "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*. IEEE, 2014, pp. 414–421.
- [49] T. Stibor, "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*. Springer, 2010, pp. 509–519.
- [50] G. You, G. Kim, S.-j. Cho, and H. Han, "A comparative study on optimization, obfuscation, and deobfuscation tools in android," *J. Internet Serv. Inf. Secur.*, vol. 11, no. 1, pp. 2–15, 2021.
- [51] B. S. Vidya and E. Chandra, "Entropy based local binary pattern (elbp) feature extraction technique of multimodal biometrics as defence mechanism for cloud storage," *Alexandria Engineering Journal*, vol. 58, no. 1, pp. 103–114, 2019.
- [52] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
- [53] G. Canfora, F. Mercaldo, and C. A. Visaggio, "An hmm and structural entropy based detector for android malware: An empirical study," *Computers & Security*, vol. 61, pp. 1–18, 2016.
- [54] M. R. B. Shamsuddin, F. H. H. M. Ali, and M. S. B. Z. Abidin, "Transforming malware behavioural dataset for deep denoising autoencoders," in *IOP Conference Series: Materials Science and Engineering*, vol. 769, no. 1. IOP Publishing, 2020, p. 012071.
- [55] G. D'Angelo, M. Ficco, and F. Palmieri, "Malware detection in mobile environments based on autoencoders and api-images," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 26–33, 2020.
- [56] M. Zhao, D. Wang, Z. Zhang, and X. Zhang, "Music removal by convolutional denoising autoencoder in speech recognition," in *2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. IEEE, 2015, pp. 338–341.

-
- [57] P. Görgel and A. Simsek, "Face recognition via deep stacked denoising sparse autoencoders (dsdsa)," *Applied Mathematics and Computation*, vol. 355, 2019.
- [58] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, 2020.
- [59] S. Kumar *et al.*, "Mcf-t-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things," *Future Generation Computer Systems*, vol. 125, pp. 334–351, 2021.
- [60] S.-C. Hsiao, D.-Y. Kao, Z.-Y. Liu, and R. Tso, "Malware image classification using one-shot learning with siamese networks," *Procedia Computer Science*, vol. 159, pp. 1863–1871, 2019.
- [61] M. O. T. Sison, "Calculating distances between windows malware using siamese neural network embeddings," 2021.
- [62] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "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.
- [63] D. Zhang, Z. Zheng, M. Li, X. He, T. Wang, L. Chen, R. Jia, and F. Lin, "Reinforced similarity learning: Siamese relation networks for robust object tracking," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 294–303.
- [64] S. Cheng, B. Zhong, G. Li, X. Liu, Z. Tang, X. Li, and J. Wang, "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.
- [65] P. Bedi, N. Gupta, and V. Jindal, "Siam-ids: Handling class imbalance problem in intrusion detection systems using siamese neural network," *Procedia Computer Science*, vol. 171, pp. 780–789, 2020.
- [66] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," *Advances in neural information processing systems*, vol. 30, 2017.