EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

Synthetic Malware Using Deep Variational Autoencoders and Generative Adversarial Networks

Aaron Choi[1], Albert Giang[1], Sajit Jumani[1], David Luong[1], Fabio Di Troia[1,*]

[1]San Jose State University, One Washington Sq, 95192, San Jose, CA, U.S.A.

Abstract

The effectiveness of detecting malicious files heavily relies on the quality of the training dataset, particularly its size and authenticity. However, the lack of high-quality training data remains one of the biggest challenges in achieving widespread adoption of malware detection by trained machine and deep learning models. In response to this challenge, researchers have made initial strides by employing generative techniques to create synthetic malware samples. This work utilizes deep variational autoencoders (VAE) and generative adversarial networks (GAN) to produce malware samples as opcode sequences. The generated malware opcodes are then distinguished from authentic opcode samples using machine and deep learning techniques as validation methods. The primary objective of this study was to compare synthetic malware generated using VAE and GAN technologies. The results showed that neither approach could create synthetic malware that could deceive machine learning classification. However, the WGAN-GP algorithm showed more promise by requiring a higher number of synthetic malware samples in the train set to effectively be detected, proving it a better approach in synthetic malware generation.

Received on 15 February 2024; accepted on 30 June 2024; published on 09 July 2024

Keywords: Malware, Synthetic Malware, GAN, VAE

doi:10.4108/eetiot.6566

1. Introduction

The threat of malware attacks poses a significant and constantly evolving challenge to information security. As our society becomes more digitally connected, malware has become increasingly sophisticated and able to exploit various vulnerabilities [1]. Malicious software, or malware, is designed to harm computers, systems, and networks, encompassing a range of types such as viruses, worms, trojans, spyware, ransomware, adware, fileless malware, bots, rootkits, mobile malware, and wiper malware [2]. These malicious programs can cause disruptions in multiple areas, including businesses, supply chains, infrastructure, government systems, financial systems, personal computers, mobility systems, and more. To combat these threats, researchers are actively working on leveraging machine learning and deep learning techniques to classify and detect malware [3, 4]. Additionally, there has been a focus on generating synthetic malware to augment training datasets [5]. The effectiveness of detecting malicious files greatly depends on the quality of the training dataset, particularly in terms of sample size and authenticity [6]. However, the scarcity of high-quality training data remains a significant challenge for the widespread adoption of malware detection by trained machine learning and deep learning models. Fortunately, recent advancements in generative techniques have shown promising results in creating synthetic malware samples that closely resemble authentic malware and can deceive machine learning detection models in some cases.

In this study, malware samples were generated as opcode sequences using a variational autoencoder (VAE) with inputs from multiple original malware families. The objective was to distinguish synthetic malicious code from actual malicious code samples using machine learning and deep learning techniques as validation methods.

*Corresponding author. Email: fabio.ditroia@sjsu.edu

2. Related Work

Synthetic malware generation is a growing field that uses both machine learning and deep learning generation techniques to augment training datasets for malware classification and detection algorithms. The primary driver behind selecting the appropriate synthetic malware generator is based on which malware features will be extracted and how they will be represented. The following are all examples of features that can be extracted from malware: API call sequences [7], images [8, 9], opcode sequences [10], [11], assembly code and dynamic-link library import [12], control flow graph [13], [14], portable executable header data [15], network activities [16], malicious domain name server [17], and more [18].

After extracting the desired features there are several options for how to represent the data. One popular example is converting the binary malware file into an 8-bit grayscale image which can then be replicated three times to create an RGB corollary [8, 19]. Another approach is to use opcode sequences or opcodes, which are mnemonic representations of machine code that symbolize assembly instructions [10]. Opcodes can be converted to an integer vector, n-grams, or word embeddings with Word2Vec [5]. Oftentimes, combinations of approaches complement each other to provide a comprehensive view of the malware signature and behavior [20]. For our research, we chose to work with opcode sequences and implement integer embedding.

Generating synthetic malware can be accomplished through various methods. Burks et al. [19] have shown that both GANs and VAEs can generate high-quality synthetic benign and malware images, validated by single-digit percentage improvements in the accuracy of deep learning classifiers when differentiating between benign and malicious files. Lu and Li [21] achieved similar results when using a deep convolutional GAN (DCGAN) to boost the training datasets. When working with malware opcode sequences, WGAN with a gradient penalty (WGAN-GP) produces some of the most realistic synthetic malware, performing better than Hidden Markov Model (HMM), regular GAN, and WGAN. WGAN- GP was able to reduce the average classifier accuracy by approximately 20% compared to HMM and WGAN [5]. Comparing the performance of a deeper VAE against WGAN-GP for generating synthetic malware opcode sequences had not been explored and formed the basis of our study. Our work drew inspiration from the concepts presented in [5] for our GAN application, and we incorporated metrics and implementation strategies from that source. Furthermore, we attempted to improve upon the performance of WGAN-GP by using a Dense VAE and CNN-based VAE (CNN VAE) to generate synthetic malware opcodes. Bae and Lee [22] implemented a relatively shallow VAE to augment the training dataset of opcodes (both benign and malicious files). Their VAE had a 128-dimensional embedding layer followed by the encoder and decoder which each contained one hidden layer and one gated recurrent unit (GRU) layer. This resulted in an improvement in accuracy of 0.98% for their long short-term memory (LSTM) classifier. This indicated that their VAE was able to produce adequately realistic benign opcode and malware opcode sequences. However, it did not address the potential performance improvement of adding depth to the VAE, nor did it quantify how realistic the synthetic malware was compared to the authentic malware.

3. Background

This section provides information about the authentic malware dataset, and the parameters used for the generative models and for the machine learning classifiers.

3.1. Malware Datasets

We utilized malware samples obtained from the Malicia Project and VirusShare group, which were compiled from online sources. We selected datasets from five distinct malware families, namely WinWebsec, Zbot, Renos, VBInject, and OnLineGames, to train our deep learning models. Additionally, we created a sixth dataset by combining the opcode samples from all five families. This dataset aimed to provide a more challenging training set for the deep learning generators and explore their ability to generate synthetic malware across multiple families.

To preprocess the malware samples, we disassembled them from their executable format into raw assembly code. We then eliminated unnecessary data, retaining only the opcodes. To reduce noise within the data, we constructed an indexed dictionary containing the top N most frequently occurring opcodes in each malware family. This dictionary was used to map the opcode datasets to integer representations. Creating a dictionary file for every unique opcode would be computationally intensive, so we chose the top N most frequent opcodes for each malware family. The value of N for each family was determined based on a parametric study conducted in [5], where the highest Area Under the ROC Curve (AUC) models were selected, and their corresponding N values were kept constant for training the other neural networks. For the mixed dataset, the unique opcode count was determined by performing a mathematical union of the dictionary files from the other five malware families. During the neural network training, any opcode not present in the dictionary file was assigned

an integer value of N+1. Furthermore, within each family's dataset, samples with opcode sequences shorter than 600 were discarded, while samples with opcode sequences longer than 600 were truncated to 600. This resulted in training datasets consisting of 600x1 tensors. The value 600 was chosen for consistency with the research outcome in [5]. Table 1 provides details on the number of unique opcodes and total samples per malware family after data processing.

Table 1. Training Data Parameters

Malware Family	Unique Opcodes	# of Samples
OnLineGames	21	1507
Renos	21	1566
VBInject	24	2429
WinWebSec	21	3932
Zbot	19	1951
Mixed	40	11335

These processed datasets served as the training data for the generative models. On the other hand, the machine learning and deep learning malware classifiers utilized a larger dataset for both training and testing purposes. The classifiers employed a balanced dataset comprising a 1:1 ratio of synthetically generated malware and authentic malware. Each classifier was provided with six balanced datasets, corresponding to the five malware families plus the mixed family, associated with each of the three generative models (WGAN-GP, CNN VAE, Dense VAE).

3.2. Generative Models

Wasserstein Generative Adversarial Network with Gradient Penalty. The fundamental structure of a GAN comprises generator layers and discriminator layers. The generator layers are responsible for generating new data samples that resemble the training data inputs. The initial generator layers consist of alternating convolution and batch normalization layers. The convolution layer transforms input noise into a higher dimensional space, while the batch normalization layer enhances the stability and convergence of the convolutional layer outputs by subtracting the batch mean and dividing it by the batch standard deviation. After passing through three sets of convolution and batch normalization layers, the data flows into a flatten layer and then a dense layer. Subsequently, the data undergoes a reshape layer, which converts the higher dimensional input data into an output matching the shape of the opcode training dataset (600x1). The GAN discriminator receives either synthetic data produced by the GAN generator or authentic data from the training dataset. This data is processed through a sequence of convolutional layers that extract higher dimensional features. Unlike the generator, batch normalization layers are omitted from the discriminator design because the gradients of each individual input are of interest. The final flatten and dense layers transform the higher dimensional data from the convolutional layers into a lower dimensional format suitable for the activation function, typically a sigmoid or Rectified Linear Unit (ReLU) function.

The aforementioned descriptions outline the core architecture of a traditional GAN. To create a WGAN, a deviation is introduced at the output of the discriminator. Instead of a binary classification decision, the output of the traditional GAN discriminator is replaced with a linear function that produces an uncapped output. This allows the model to calculate the distance between the outputs and the original data, referred to as the Wasserstein distance. The Wasserstein distance measures the dissimilarity between the synthetic and authentic data distributions. The role of the discriminator model shifts to that of a critic model, as it no longer produces a binary classification but a distance that it aims to minimize during training to generate better malware samples.

To further enhance the features of the WGAN, the WGAN-GP architecture introduces a gradient penalty to the Wasserstein loss. This penalty penalizes the discriminator when its gradient norm deviates from a value of one. This modification encourages the critic to have a smooth decision boundary and improves the overall stability and convergence of the model.

CNN-Based Variational Autoencoder. Two types of VAE architectures were implemented: a one-dimensional CNN-based VAE (CNN VAE) and a fully-connected dense VAE (Dense VAE). The encoder takes authentic malware as a 600x1 tensor input. This tensor is passed through four one-dimensional convolutional layers. Each convolutional layer has an increasing number of filters (64, 128, 256, 512) with a depth of three channels. This is followed by a flatten layer and a 16-node dense layer, which branches into two parallel nodes: z-mean and z-variance. The z-sampling layer uses the z-mean and z-variance nodes to generate the latent space variable outputs at the end of the encoder. A Gaussian distribution is employed to sample the latent variables, which are then passed to the decoder.

The CNN VAE decoder architecture, receives the 2 latent variables from the encoder and feeds them into a dense layer with 12,800 nodes. The number of nodes in this dense layer matches the flattened shape after the final convolutional layer in the encoder. This dense layer is followed by five transposed convolutional layers with three-channel filters. The filter count per layer mirrors the encoder, gradually decreasing (512, 256, 128, 64, 1) to ultimately produce a 600x1 tensor, similar to the encoder input.

During development, various architecture parameters were explored. Two and three convolutional layer architectures were tested for both the encoder and decoder. Increasing the number of convolutional layers up to four showed improvements in training error, but beyond four layers, the error began to plateau. The quantity of latent variables was also explored, ranging from 2 to 4, 8, 16, and 32. However, no significant improvement in training performance was observed beyond 2 latent variables. Peak filter counts ranging from 128 to 1024 were evaluated as well. Increasing the peak filter count up to 512 resulted in improved training performance, but further increasing it to 1024 did not yield noticeable changes.

A Leaky ReLU activation function with an alpha of 0.01 was used for both the convolutional and dense layers. The default alpha of 0.3, as well as 0.2 and 0.1, were also considered, but lowering the alpha value reduced the training error. Hence, progressively smaller alpha values were explored until 0.01 was determined to be optimal. The dense layers originally employed a sigmoid activation function, but training performance significantly improved with Leaky ReLU.

During training, an Adam optimizer with a learning rate of 0.001 was utilized. Learning rates of 0.002 and 0.003 resulted in NaN errors during training, so the learning rate of 0.001 was maintained. The CNN VAE was trained using stochastic gradient descent. Batch gradient descent was also experimented with, but no improvements in training speed or loss were observed.

Dense Variational Autoencoder. Similar to the CNN VAE, the encoder takes a 600x1 tensor as input. In the Dense VAE, the encoder consists of five fully connected layers with node counts of 256, 128, 64, 32, and 16, respectively. The 16-node layer is followed by two parallel nodes: z-mean and z-variance. The z-sampling layer utilizes the z-mean and z-variance nodes to generate the latent space variables as outputs at the end of the encoder.

The decoder receives the two latent variables as input and proceeds with five fully connected layers, containing 16, 32, 64, 128, and 256 nodes, respectively.

During the development process, various architecture parameters were explored. Increasing the number of fully connected layers and the number of nodes per layer in both the encoder and decoder led to a reduction in the overall reconstruction loss. The ReLU activation function was applied to each dense layer in both the encoder and decoder. The AdamW optimizer was utilized with a learning rate of 0.001. Once trained, the Dense VAE was capable of generating synthetic samples solely by utilizing the latent variables as inputs to the decoder.

To generate synthetic malware using the trained VAEs, the latent space was randomly sampled and passed through the trained decoder. The resulting outputs served as the synthetic data, which could then be fed into machine learning and deep learning classifiers.

3.3. Machine Learning Classifiers

Random Forest. To maintain consistency with the previous research [5], we employed a Random Forest (RF) classifier as one of the machine learning classifiers to distinguish between synthetic and authentic malware. Following the approach in [5], we used 50 decision trees with a depth of 5 for each tree as the parameters for the RF classifier.

Support Vector Machine. In line with our goal of comparing our results to the previous research, we selected a Support Vector Machine (SVM) as the second machine learning classifier for differentiating between synthetic and authentic malware. Similar to the methodology employed in [5], we maintained the SVM parameters. Specifically, we implemented the SVM with a regularization parameter of five and utilized the radial basis function (RBF) kernel.

k-Nearest Neighbors. To ensure consistency with the previous findings [5], we included a k-Nearest Neighbors (KNN) classifier as the third machine learning classifier to distinguish between synthetic and authentic malware. The number of neighbors for the KNN classifier was set to one, and the power parameter for the Minkowski metric, which determines the distance calculation, was set to two for Euclidean distance.

Long Short-Term Memory Recurrent Neural Network Classifiers. To leverage the capability of processing long sequences of data, we built and utilized three different LSTM classifiers: a standard LSTM, a bi-directional LSTM, and a CNN-based LSTM (CNN LSTM).

The standard LSTM architecture was implemented as a sequential model consisting of an embedding input layer, an LSTM middle layer, and a dense output layer. In the embedding layer, each opcode was mapped to a 32-length real-valued vector. The LSTM layer, with 100 memory units, evaluated the similarity between opcode sequences by measuring their proximity in vector space. The dense output layer, with a single neuron and sigmoid activation function, computed the probability of classifying the malware as authentic or synthetic.

For the bi-directional LSTM architecture , the network was trained in both the forward and backward directions. Similar to the standard LSTM, the bi-directional LSTM included an embedding layer, but the middle layer was replaced with a bi-directional layer. The input data was passed to both the forward and backward layers within the bi-directional layer, and their outputs were fed into the activation layer. Dropout

layers were introduced between the embedding and bi-directional LSTM layers, as well as between the bi-directional LSTM and dense output layers, to mitigate overfitting. A dropout rate of 0.2 was used for each dropout layer, including the input and recurrent connections.

To capture invariant features of the malware, a CNN LSTM architecture was constructed by integrating a one-dimensional convolutional layer between the embedding and LSTM layers of the standard LSTM classifier. In this architecture, the convolutional layer had a filter size of 32, a kernel size of three, and a ReLU activation function. A MaxPooling layer with a size of two was added after the convolutional layer to reduce the feature map size.

During training, we employed the binary cross-entropy loss function, commonly used for binary classification problems. The Adam algorithm, a combination of AdaGrad and RMSProp, was selected as the optimization algorithm for efficient stochastic gradient descent during backpropagation [23].

4. Methodology and Implementation

There were five sequential steps involved in our implementation process. First, we built the three generator models, that is, WGAN-GP, CNN VAE, and Dense VAE. Second, we trained the generators on authentic malware samples. Third, we generated synthetic malware samples using the trained generators. Fourth, we classified the malware samples using three machine learning classifiers and three deep learning classifiers to differentiate between synthetic and authentic malware. Finally, we determined the minimum number of training samples required for the classifiers to achieve at least 95% accuracy in detecting the synthetic malware.

The primary goal of the classification process was to assess the quality and believability of the synthetic malware. A lower classification accuracy indicated that the synthetic malware was harder to distinguish from authentic malware, implying higher quality. The inclusion of deep learning classifiers aimed to explore potential performance improvements over machine learning classifiers and provide alternative classifier architectures if the machine learning classifiers struggled with the malware datasets.

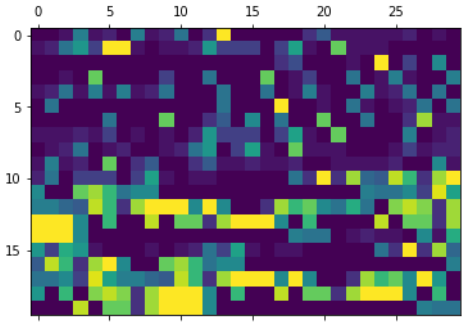
For the WGAN-GP generator, we used the same parameters as in [5]. The Adam optimizer with a learning rate of 0.0001, beta1 of 0.5, and beta2 of 0.9 was employed. However, we trained the WGAN-GP on 10,000 epochs instead of 100,000 mini-batches to ensure that the generator model encountered every available malware sample at each training step. The number of samples seen by the generator using epochs is roughly equivalent to the mini-batch system, as each epoch's size is at least ten times that of a single mini-batch. The generated malware from WGAN-GP was then compared to the malware samples generated by the CNN and Dense VAE models.

When training the CNN VAE generator, the majority of the error reduction occurred within 30 to 50 epochs. Training up to 75, 100, 150, and 200 epochs resulted in only marginal improvements in training performance. Therefore, we selected 100 epochs as the nominal value to capture some of the marginal improvements along the plateau. Additionally, the WinWebSec family was used to iterate quickly through various hyperparameters due to its relatively larger dataset compared to the other families.

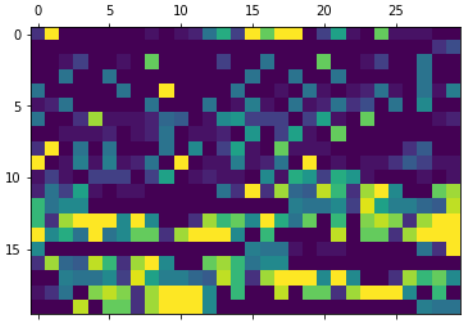
For the Dense VAE generator, we explored eight different model optimizers: Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSprop), Adadelta, AdaGrad, Adam, AdamW, Nadam, and Follow The Regularized Leader (FTRL). Each optimizer produced a unique distribution of the trained latent space. SGD created a latent space with samples clustered in parallel lines, while Nadam resulted in a large z-variance (greater than 40) and fewer extreme outliers. Most other optimizers produced a z-mean and z-variance between -5 to 5. After comparing the impact of all the optimizers on Dense VAE training performance for the WinWebSec family at 30 epochs, we found that AdamW achieved the lowest training loss and generated a latent space that contributed to the most realistic synthetic malware.

After training the three generative models on authentic malware, they were utilized to generate synthetic malware samples. The WGAN-GP generator accomplished this by converting random noise into synthetic malware samples using the trained generator. The CNN VAE and Dense VAE generators generated synthetic malware by randomly sampling from the learned latent spaces using a Gaussian distribution and feeding the samples into the decoder. All three generators produced sequences of real-valued numbers ranging from zero to one. These values were then scaled up by multiplying them by the number of unique opcodes for their respective malware family. The scaled-up values were rounded to obtain synthetic integer mappings for the opcode sequences.

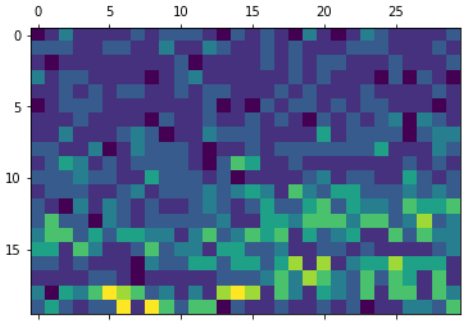
Figure 1 provides a visual representation of authentic and synthetic malware. In (a), malware from the WinWebSec family is represented, while (b), (c), and (d) display visual representations of synthetic malware generated by the WGAN-GP, Dense VAE, and CNN VAE, respectively. Each visualization represents a 600x1 one-dimensional malware sample as a 20x30 two-dimensional representation. Each pixel value corresponds to an integer mapping of an opcode. These visualizations facilitate the observation of the varying resemblance of each synthetic malware sample to the



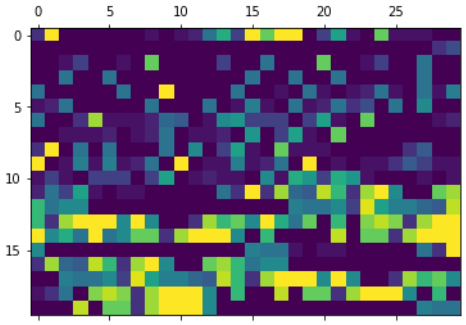
(a) Authentic Malware.



(b) Synthetic Malware (WGAN-GP).



(c) Synthetic Malware (Dense VAE).



(d) Synthetic Malware (CNN-VAE).

Figure 1. Visual representation of authentic and synthetic malware (WinWebSec family).

authentic sample, characterized by a darker top half and a lighter lower half.

These visualizations were instrumental in exploring the sensitivity of the latent space sampling and its impact on the generated synthetic malware. Figure 1 (c), for instance, was produced by randomly sampling from the trained latent space of the Dense VAE using a Gaussian distribution. While it bears some resemblance to the authentic malware sample in 1 (a), other synthetic samples were visually distinct. Further investigation revealed that these distinguishable synthetic samples consisted of very few opcodes within the top N for a given family, sometimes as few as 5 out of 600. This sampling approach failed to consistently produce high-quality synthetic malware with the Dense VAE as the sampling noise propagated into the generated samples. Moreover, these noisy synthetic samples exhibited overlap between different clusters of encoded malware samples.

To mitigate unwanted noise when sampling in undesired locations, K-means clustering was introduced to the latent space. Twenty clusters were created, and the cluster with the smallest sum of squared Euclidean distances (SSE) was selected as the new sampling space. Instead of sampling from the entire latent space, a Gaussian distribution was employed to randomly sample from within the selected cluster. The mean of the Gaussian distribution was set as the center of the selected cluster. Applying this approach to generators trained on the combined dataset of all five malware families did not yield any benefits. However, when clustering was applied to generators trained on individual malware families, there was an approximate 10% reduction in machine learning classifier performance when classifying between 100 synthetic and 100 authentic samples. This clustering approach helped avoid overlap between samples from different clusters and produced samples that closely aligned with the encoded authentic samples in the latent space.

The classification process involved utilizing both machine learning and deep learning classifiers. Deep learning models were particularly useful for classification tasks and provided a meaningful comparison to machine learning models in case the machine learning classifiers performed poorly. To address any issues related to classification imbalance, the number of synthetic samples was matched with the number of authentic malware samples.

During the training of the classifiers, it was observed that the deep learning classifiers required significantly more time compared to the machine learning classifiers. To expedite the training process, an early stopping criterion was implemented for the LSTMs. If there was no improvement in the loss by at least 0.01 after 10 training epochs, the training was stopped, and the model parameters were selected from the epoch with the best loss value.

The machine learning and deep learning classifiers were trained using an 80/20 train/test split on all five datasets comprising both authentic and synthetic malware generated by WGAN-GP, Dense VAE, and CNN VAE. To ensure consistency and evaluate the performance of classifying unseen malware, the results were averaged using K-fold cross-validation. The accuracy, precision, and recall scores of all six classifiers were compared across all three generators for each malware family, as well as a mixed malware family. In the case of well-trained and highly accurate classifiers, lower classification scores indicated higher quality synthetic malware, as the classifier struggled to differentiate between synthetic and authentic malware.

The final experiment aimed to determine the minimum number of mixed family malware samples required by each classifier to achieve a classification accuracy of at least 95% for the three generators. To optimize efficiency, a binary and jump search routine was applied to the classification process.

5. Analysis and Results

Figure 2 illustrates the training loss of WGAN-GP on the WinWebSec malware family. Although the model was trained for 10,000 epochs, it appeared that the generator loss had stabilized around 1,500 epochs. This loss value was calculated using the Wasserstein distance, which measures the dissimilarity between the generated malware samples and the authentic malware samples.

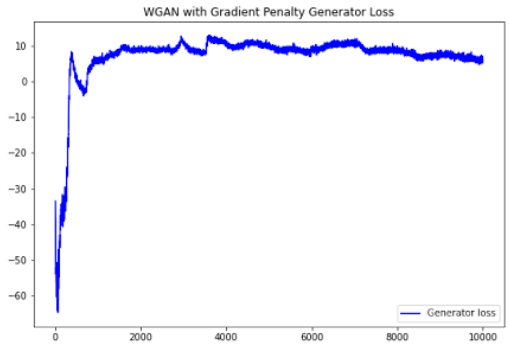
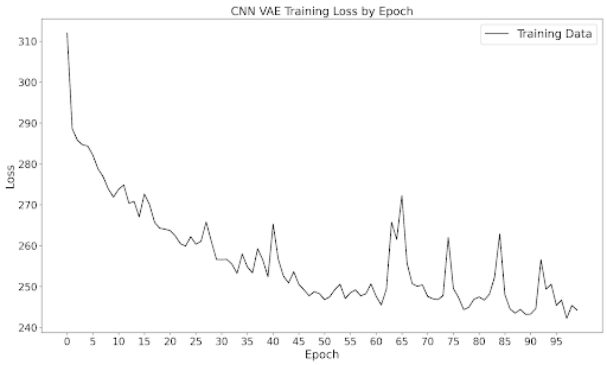
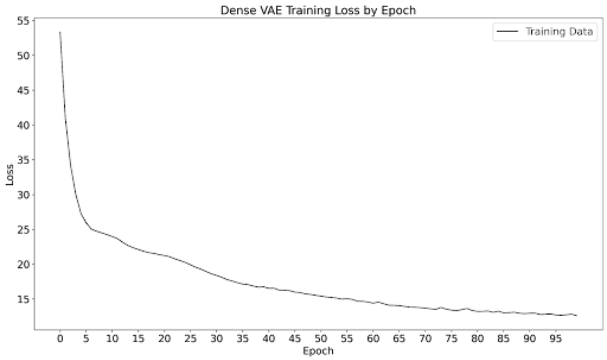


Figure 2. Training loss of WGAN-GP for WinWebSec.

Figure 3 displays the training loss of the CNN VAE (a) and Dense VAE (b), also on the WinWebSec dataset. Both models were trained for 100 epochs, with the loss values gradually decreasing during the training process. The loss function of the VAE consists of a reconstruction loss and a KL divergence loss. In the case of the CNN and Dense VAEs, the reconstruction loss was computed using binary cross entropy. The KL divergence loss quantifies the difference between



(a) Training Loss of CNN VAE for WinWebSec.



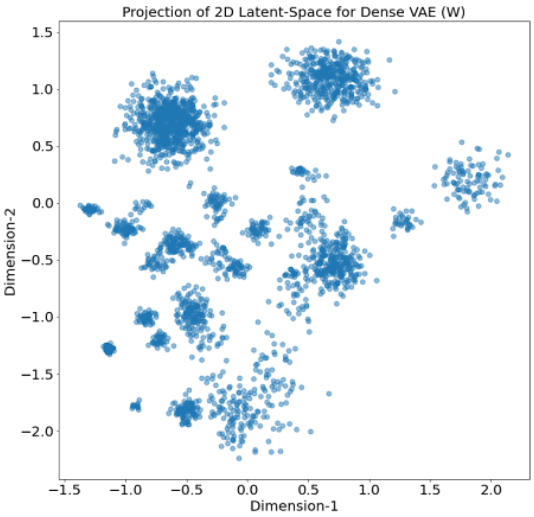
(b) Training loss of Dense VAE for WinWebSec.

Figure 3. Training loss for WinWebSec.

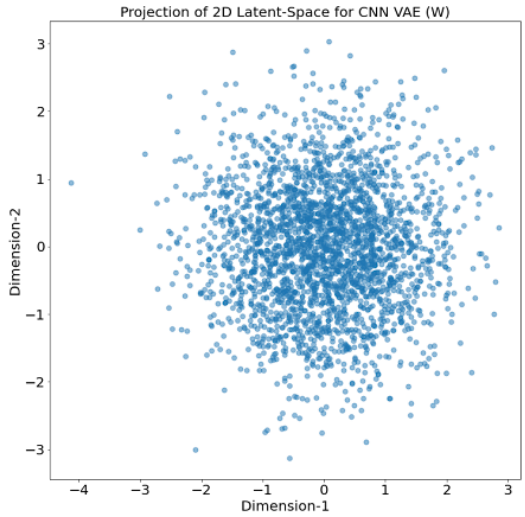
the distribution of the latent space and a Gaussian distribution. Both VAEs exhibited relatively low KL divergence loss compared to the reconstruction loss. It is worth noting that the loss value reached a higher level of convergence for the CNN VAE compared to the Dense VAE, indicating that the Dense VAE may be more effective in generating synthetic malware samples that closely resemble authentic malware.

Figure 4 depicts the sampled points in the latent spaces of the VAEs for the WinWebSec malware family. Notably, the Dense VAE's latent space exhibited the formation of multiple clusters (a), whereas the CNN VAE's latent space displayed a single, larger cluster resulting from the sampling process (b).

Although both the Dense VAE and CNN VAE utilize the reconstruction error and KL divergence, the CNN VAE provides a more regular latent space. As demonstrated in Figure 4, the encoded samples in the CNN VAE are closer to each other compared to the Dense VAE. This suggests that the decoder of the CNN VAE is capable of generating synthetic samples that are more realistic when compared to the irregular latent space of the Dense VAE. Generating samples from the latent space would result in more nonsensical samples. The random selection of samples using a normal distribution would produce fewer meaningful



(a) Latent space visualization of Dense VAE for WinWebSec.



(b) Latent space visualization of CNN VAE for WinWebSec.

Figure 4. Latent space visualization for WinWebSec.

data points. This is because the mean of the encoded data is not evenly distributed or close to zero.

Tables 2, 3, and 4 present the classification results obtained by the three machine learning classifiers for the WGAN-GP, Dense VAE, and CNN VAE generators, respectively. The classifiers evaluated a balanced dataset consisting of 50% synthetic and 50% authentic malware samples from all five malware families, including the mixed family. All three classifiers achieved high precision, recall, and accuracy across all malware families for each generator, with SVM and RF outperforming KNN. K-fold validation with K = 5 was employed by each classifier to ensure consistent results. These findings suggest that the synthetic malware generated by all three generators does not closely resemble authentic malware. However, this result is influenced by the large number of training samples provided to the three malware classifiers. If the number of samples is reduced, the machine learning classifiers would approach the performance achieved in [5] when comparing the WGAN-GP instance.

Table 2. Classification Results for WGAN-GP Synthetic Malware

WGAN-GP Malware	Classifier	Performance		
		Precision	Recall	Accuracy
OnLineGames (3014 Samples)	SVM[a]	1.00	1.00	1.00
	RF[b]	1.00	0.96	0.98
	KNN[c]	1.00	0.95	0.98
Renos (3132 Samples)	SVM	1.00	0.99	0.99
	RF	0.90	0.98	0.93
	KNN	1.00	0.87	0.94
VBInject (4858 Samples)	SVM	1.00	1.00	1.00
	RF	0.98	0.95	0.97
	KNN	1.00	0.66	0.83
WinWebSec (7864 Samples)	SVM	1.00	1.00	1.00
	RF	0.99	0.98	0.99
	KNN	1.00	0.97	0.98
Zbot (3802 Samples)	SVM	1.00	0.99	1.00
	RF	1.00	0.97	0.99
	KNN	1.00	0.98	0.99
Mixed (22670 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	0.89	0.94
	KNN	0.52	1.00	0.54

[a] SVM: Support Vector Machines
[b] RF: Random Forest
[c] KNN: k-Nearest Neighbor

The lower performance of the KNN classifier can be attributed to the inherent limitations of the classifier itself, rather than the synthetic malware appearing realistic. This conclusion is supported by the accurate classification achieved by the other machine learning classifiers on the same malware samples. The performance of deep learning LSTM classifiers is not presented here since at least one of the machine learning classifiers performed exceptionally well. Deep learning classifiers are not necessary for this particular evaluation, as they would surpass the performance of the machine learning classifiers while requiring longer training time.

Given the high classification performance achieved by the machine learning classifiers on sufficiently

Table 3. Classification Results for Dense VAE Synthetic Malware

Dense VAE Malware	Classifier	Performance		
		Precision	Recall	Accuracy
OnLineGames (3014 Samples)	SVM[a]	1.00	1.00	1.00
	RF[b]	1.00	1.00	1.00
	KNN[c]	1.00	0.81	0.88
Renos (3132 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.77	0.85
VBInject (4858 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.63	0.71
WinWebSec (7864 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.99	0.99
Zbot (3802 Samples)	SVM	1.00	0.94	0.97
	RF	1.00	1.00	1.00
	KNN	1.00	0.82	0.89
Mixed (22670 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.82	0.89

[a] SVM: Support Vector Machines
[b] RF: Random Forest
[c] KNN: k-Nearest Neighbor

Table 4. Classification Results for CNN VAE Synthetic Malware

CNN VAE Malware	Classifier	Performance		
		Precision	Recall	Accuracy
OnLineGames (3014 Samples)	SVM[a]	1.00	1.00	1.00
	RF[b]	1.00	1.00	1.00
	KNN[c]	1.00	0.82	0.89
Renos (3132 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.80	0.87
VBInject (4858 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.63	0.71
WinWebSec (7864 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	1.00	1.00
	KNN	1.00	0.99	0.99
Zbot (3802 Samples)	SVM	0.91	1.00	0.95
	RF	0.98	0.98	0.98
	KNN	1.00	0.94	0.97
Mixed (22670 Samples)	SVM	1.00	1.00	1.00
	RF	1.00	0.96	0.98
	KNN	1.00	0.82	0.89

[a] SVM: Support Vector Machines
[b] RF: Random Forest
[c] KNN: k-Nearest Neighbor

large datasets from the three generators, a second experiment was conducted to compare the synthetic malware. The objective was to determine the minimum number of training samples required by each classifier to achieve a target classification accuracy of at least 95%. Table V displays the minimum number of samples needed to achieve 95% accuracy using a mixed dataset composed of both synthetic and authentic malware from all five malware families. A lower number of samples indicates that synthetic malware is more easily distinguishable from authentic malware. In other words, the classifier can differentiate between authentic and synthetic malware using fewer samples because the synthetic malware appears artificial. When comparing the minimum samples required for SVM and RF to achieve 95% accuracy, it was found that WGAN-GP necessitated more samples than the CNN VAE or Dense VAE. This suggests that WGAN-GP produced synthetic malware that was more realistic. Similarly, the CNN VAE generated slightly better synthetic malware compared to the Dense VAE.

It is worth noting that SVM, RF, and KNN required significantly more samples than the LSTM classifiers across all three generators. As mentioned earlier and demonstrated in these experiments, the deep learning classifiers achieved superior classification performance compared to the three machine learning classifiers, requiring only 20 samples to achieve 95% accuracy. This substantial performance difference underscores the strength of LSTM classifiers over machine learning classifiers. However, if there are limitations in computational capacity and the number of training samples, and it is known that the synthetic malware is not generated by a GAN, then SVM and RF machine learning classifiers would still offer satisfactory classification performance.

6. Conclusions

The aim of this study was to assess the capability of Dense VAE and CNN VAE in generating realistic synthetic malware compared to WGAN-GP across multiple malware families. The evaluation was conducted using both machine learning and deep learning classifiers, with the expectation that lower classifier performance would indicate more realistic synthetic malware.

In the standard classification experiments, the trained machine learning classifiers were able to easily differentiate between synthetic and authentic malware when an adequate number of samples were provided. This was evident from the high classification scores obtained for the synthetic malware generated by all

Table 5. Minimum Number of Mixed Malware Samples Needed for 95% Classification Accuracy

Generative Model	Classifier	Number of Samples
WGAN-GP[a]	SVM[b]	772
	Random Forest	236
	KNN[c]	22,670
	LSTM[d]	20
	BiLSTM[e]	20
	CNN-LSTM[f]	20
CNN-VAE[g]	SVM	44
	Random Forest	58
	KNN	22,670
	LSTM	20
	BiLSTM	20
	CNN-LSTM	20
Dense VAE	SVM	20
	Random Forest	48
	KNN	22,670
	LSTM	20
	BiLSTM	20
	CNN-LSTM	20

[a] WGAN-GP:Wasserstein Generative Adversarial Networks with Gradient Penalty
[b] SVM: Support Vector Machines
[c] KNN: k-Nearest Neighbor
[d] LSTM: Long Short-Term Memory Recurrent Neural Network
[e] BiLSTM: Bidirectional LSTM
[f] CNN-LSTM: Convolutional Neural Network LSTM
[g] CNN-VAE: Convolutional Neural Network Variational Autoencoder

three generators. SVM and RF emerged as the strongest machine learning classifiers, consistently achieving accuracy scores above 90% for the synthetic malware produced by WGAN-GP, Dense VAE, and CNN VAE. On the other hand, KNN consistently underperformed compared to SVM and RF, likely due to the inherent limitations of the classifier itself. Since SVM and RF classifiers demonstrated such high classification performance in distinguishing synthetic and authentic malware, there was no need to employ the three LSTM deep learning classifiers in the initial experiment.

The high classification scores obtained for all three generators posed a challenge in determining which generator produced more realistic malware. Therefore, instead of focusing on classification scores, we examined the minimum number of samples required to achieve 95% classification accuracy. As the LSTM classifiers proved to be too accurate for the second experiment, and KNN exhibited insufficient accuracy, the results from SVM and RF provided better performance comparisons among the three generators' synthetic malware. It became evident that the synthetic malware generated by WGAN-GP necessitated a significantly larger number of training samples for SVM and RF to achieve 95% classification accuracy compared to the malware samples from the two VAEs. This observation suggested that WGAN-GP produced synthetic malware that more closely resembled authentic malware. Among the two VAEs, the CNN VAE slightly outperformed the Dense VAE in generating synthetic malware. This finding was unexpected considering that the Dense VAE had a significantly lower training loss than the CNN VAE, and the validation loss of the Dense VAE was also low, indicating no significant overfitting concerns.

In conclusion, GANs like WGAN-GP demonstrate more potential in generating synthetic malware that closely resembles authentic malware compared to VAEs. However, this improved performance comes at a higher computational cost. Furthermore, while machine learning classifiers such as SVM currently suffice, as synthetic malware becomes more realistic, the use of deep learning classifiers like LSTM is likely to become more valuable in evaluating the quality of synthetic malware.

Future work could involve replicating the current implementation while employing alternative representations for the malware sample features, or a hybrid approach thereof. The present study primarily concentrated on model training utilizing the initial 600 opcodes from the files. Despite the adequacy demonstrated by this quantity as indicated in [5], exploring the precise number of opcodes required to enhance the generative models' performance could offer valuable insights. Finally, there is potential for a more in-depth exploration of additional VAE and GAN architectures.

References

[1] Cisco. What is Malware? - Definition and Examples;. Accessed Jun 26, 2024. https://www.cisco.com/c/en/us/products/security/advanced-malware-protection/what-is-malware.html.

[2] Baker K. 12 Types of Malware + Examples That You Should Know;. Accessed Jun 26, 2024. https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/.

[3] Ucci D, Aniello L, Baldoni R. Survey of Machine Learning Techniques for Malware Analysis. Computers Security. 2019 Mar;81:123-47.

[4] Aslan Ö, Samet R. A Comprehensive Review on Malware Detection Approaches. IEEE Access. 2020;8:6249-71.

[5] Trehan H, Di Troia F. Fake Malware Generation Using HMM and GAN. Journal Name. 2022;02:3-21.

[6] Illes D. On the impact of dataset size and class imbalance in evaluating machine-learning-based windows malware detection techniques. arXiv preprint arXiv:220606256. 2022.

[7] Vemparala S, Di Troia F, Corrado VA, Austin TH, Stamo M. Malware detection using dynamic birthmarks. In:

Proceedings of the 2016 ACM on international workshop on security and privacy analytics; 2016. p. 41-6.

[8] Yajamanam S, Selvin VRS, Di Troia F, Stamp M. Deep Learning versus Gist Descriptors for Image-based Malware Classification. In: Icissp; 2018. p. 553-61.

[9] Iadarola G, Martinelli F, Mercaldo F, Santone A, et al. Image-based Malware Family Detection: An Assessment between Feature Extraction and Classification Techniques. In: IoTBDS; 2020. p. 499-506.

[10] Santos I, Brezo F, Nieves J, Penya YK, Sanz B, Laorden C, et al. Idea: Opcode-sequence-based malware detection. In: Engineering Secure Software and Systems: Second International Symposium, ESSoS 2010, Pisa, Italy, February 3-4, 2010. Proceedings 2. Springer; 2010. p. 35-43.

[11] Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection. information Sciences. 2013;231:64-82.

[12] Gittins Z, Soltys M. Malware persistence mechanisms. Procedia Computer Science. 2020;176:88-97.

[13] Cesare S, Xiang Y. Classification of malware using structured control flow. In: Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing-Volume 107. Citeseer; 2010. p. 61-70.

[14] Yan J, Yan G, Jin D. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In: 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE; 2019. p. 52-63.

[15] Kumar A, Kuppusamy K, Aghila G. A learning model to detect maliciousness of portable executable using integrated feature set. Journal of King Saud University-Computer and Information Sciences. 2019;31(2):252-65.

[16] Morales JA, Al-Bataineh A, Xu S, Sandhu R. Analyzing and exploiting network behaviors of malware. In: Security and Privacy in Communication Networks: 6th Iternational ICST Conference, SecureComm 2010, Singapore, September 7-9, 2010. Proceedings 6. Springer; 2010. p. 20-34.

[17] Messabi KA, Aldwairi M, Yousif AA, Thoban A, Belqasmi F. Malware detection using dns records and domain name features. In: Proceedings of the 2nd International Conference on Future Networks and Distributed Systems; 2018. p. 1-7.

[18] Maniriho P, Mahmood AN, Chowdhury MJM. A Survey of Recent Advances in Deep Learning Models for Detecting Malware in Desktop and Mobile Platforms. arXiv [csCR]. 2022.

[19] Burks R, Islam KA, Lu Y, Li J. Data Augmentation with Generative Models for Improved Malware Detection: A Comparative Study. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON); 2019. p. 0660-5.

[20] Ahmadi M, Giacinto G, Ulyanov D, Semenov S, Trofimov M. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. CoRR. 2015;abs/1511.04317.

[21] Lu Y, Li J. Generative Adversarial Network for Improving Deep Learning Based Malware Classification. In: 2019 Winter Simulation Conference (WSC); 2019. p. 584-93.

[22] Bae J, Lee C. Easy Data Augmentation for Improved Malware Detection: A Comparative Study. In: 2021 IEEE International Conference on Big Data and Smart Computing (BigComp); 2021. p. 214-8.

[23] Saxena S. Understanding Embedding Layers in Keras;. Accessed Jun 26, 2024. https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce.