

Strengthening Deep-Learning-Based Malware Detection Models Against Adversarial Attacks



Rohit Pai, Mahipal Purohit, and Preetida Vinayakray-Jani

1 Introduction

In 2019, 2 billion computers were present globally, including servers, desktops, and laptops [15]. It means that there are many attack surfaces, and it is of utmost importance to protect these devices. One of the many ways in which attackers infiltrate a system is by using malware. Malware is any software intentionally designed to cause damage to a computer, server, client, or computer network [1]. Malware is one of the biggest threats to computer security [2] in this age, and we must find effective ways to detect and tackle it.

Two major approaches have been used in the past to detect malware: static analysis and dynamic analysis. Static analysis involves using metadata, unique data, or bits of code to identify malware and create signatures and heuristics. Dynamic analysis involves executing the malware in a sandbox environment such as a virtual machine to observe its behavior and determine its malignity. These traditional methods could not keep up with the growth of malware and their variants, and alternative faster methods were needed.

Machine learning and deep learning algorithms were explored as an alternative to traditional approaches as they gave exceptional results in detecting and predicting hidden patterns. Deep learning was preferred to standard machine learning due to its ability to learn complex features and deliver high-quality results. Classical deep learning models gave exceptional results in malware detection but failed against variants of the existing malware families and were susceptible to well-crafted adversarial samples.

R. Pai (✉) · M. Purohit · P. Vinayakray-Jani
Sardar Patel Institute of Technology, Mumbai, India
e-mail: rohit.pai@spit.ac.in; mahipal.purohit@spit.ac.in; preeti.vinayakray@spit.ac.in

Generative adversarial network or GAN is the perfect solution to the problem faced by classical deep learning models. Initially proposed by Goodfellow et al. [14] in 2014, GANs can be described as a machine learning system where two neural networks, a generator, and a discriminator, compete. In the end, the generator becomes capable of generating samples that are close to real-life samples, and the discriminator becomes a great classifier. Due to adversarial training, GANs prove to be very effective against variants and obfuscated malware [4, 5]. The existing GAN-based solutions are limited due to being trained on smaller and older datasets, and there is need for newer models.

We take inspiration from the architecture proposed by Kim et al. [4] and tune it to create a novel system that is trained on a state-of-the-art dataset [10]. Utilizing adversarial training of GANs and a variety of malicious samples allows our model to detect a wide range of malware and be viable for a long time. Our main contribution is devising a robust system that identifies 11 types of malware within a malicious sample (i.e., multilabel classification) with a high degree of certainty and a low response time.

2 Background

The main objective of the survey was to find the extent of research done in the malware detection domain across academia and the industry, compare them, discuss their pros and cons, and find solutions to the existing problems. Papers [11] and [12] are survey papers compiling the existing results and techniques in academia till date. Papers [3–5] represent the state-of-the-art techniques in this domain. Paper [7] provides insights into how malware detection models can be attacked and how one can protect and improve them. Lastly, paper [9] represents the progress of the industry in malware detection.

The survey done by D. Gilbert et al [11], provides a systematic review of machine learning and deep learning techniques used for malware detection and classification. The authors have studied 67 papers that use various static, dynamic, and hybrid approaches for malware analysis. They present the issues and challenges with each type of technique and provide several research gaps. First, class imbalance in the existing datasets was identified as a major gap. Second, there were no benchmark real-world datasets available for malware detection to train the machine learning models. There are services that provide malware binaries freely, but obtaining benign samples is a hassle. Additionally, classifying a file as benign or malicious and classifying a malicious file to its family are time-consuming processes, even for a security expert. Furthermore, there is discrepancy between each dataset's labeling approach that makes it impossible to meaningfully compare the accuracy across different works. Third, malware tends to evolve over time, and new variants and families appear periodically. The machine learning models need to be periodically retrained over time to keep up with this pattern. Lastly, malware authors make the feature representation of a malicious file very similar to that of a benign

file. Recently created classifiers could be easily fooled by well-crafted adversarial samples, and there is a need for adversarial training of the models.

R. Komatwar et al. [12] begin by surveying various categories of malware and classify them into 3 types: malware by platform, malware by fiction, and malware by stubs. A notable point is that across all categories, the attackers use various packing techniques to hide the presence of malware. The authors proceed to analyze the existing static and dynamic techniques of malware detection including various machine learning approaches such as K-means, decision tree, ANN, neuro-fuzzy networks, etc. They also provide a comprehensive analysis of the existing malware image creation and classification techniques. They emphasize on the need to classify malware as images. They state that existing techniques have loopholes that the hackers can exploit, and there is a need for new, complementary, and orthogonal techniques to defeat them.

Z. Cui et al. [3] propose a CNN-based approach to detect malware. Their model is trained using a dataset of over 9342 grayscale images. To avoid the problem of overfitting, they use image augmentation techniques such as rotation, width and height shift, rescale, shear, etc. The authors created models using images of sizes 24×24 , 48×48 , 96×96 , and 192×192 and compared their results. The authors found out that as the image size increases, the performance of the model becomes better, but the training time also increases. The authors settled on 96×96 as the final image size due to an equal trade-off between performance and time. The authors also make use of the Bat algorithm to handle class imbalance over multiple families of malware. Their model classifies 25 malware families with an accuracy of 94.5%. The authors demonstrate that the model achieved better accuracy and speed when compared to other malware detection models. This model lacks due to being trained on a limited number of samples and having no adversarial training.

J.-Y. Kim et al. [4] propose a new GAN-based model called tDCGAN, capable of classifying malware and detecting zero-day attacks within 9 malware families. The authors first convert the malicious executable files into images of size 63×135 . A deep autoencoder (DAE) is trained on the dataset, and it learns to pick up important features from the image and tries to reconstruct the same image from the latent representation. The decoder of the DAE is used as the generator of the GAN as it stabilizes the learning process of the GAN. The GAN is then trained, and the discriminator of the GAN is used for malware detection. This system achieves an average classification accuracy of 95.74%. The authors test the resistance of the model against zero-day attacks by adding noise to the existing malware and testing them against the system. The authors compare the performance of their system vs. other machine learning and deep learning techniques and conclude that their system is better even in case of zero-day attacks because of the inherent adversarial training. The main drawbacks of this chapter are the limited number of malware families and the inability to classify samples as benign or malicious.

LSC-GAN proposed by J. Kim and S. Cho [5] is capable of detecting and classifying malware within 9 families. It achieves an average classification accuracy of 96.97%. The drawbacks of this model are that it requires the input malware

images of the same size and that the malicious examples generated by the GAN for training may not be malicious in practice.

R. Podschwadt et al. [7] studied 12 techniques of crafting adversarial examples and 4 defensive techniques that may withstand such attacks on different datasets. The authors found that generating adversarial samples and attacking models are far easier tasks than defending them. The paper demonstrated that adversarial training was the most effective and efficient defense for image-based classifiers. The paper stated a need for approaches tailored toward classifiers based on binary data.

Kaspersky is one of the leading anti-virus vendors in the industry. In one of their white papers [9], they give an overview of how machine learning is used for malware detection. They highlight the salient challenges in building a machine-learning-based malware detection model. They give us an insight into the malware detection techniques used by Kaspersky.

After a thorough literature survey, we observed the need to focus on classifying and detecting malware as images [12] and defending such models by using adversarial training [7]. This need was somewhat satisfied by the models proposed in papers [4] and [5], but these papers lack due to being trained on much smaller and older datasets. There is a need for models trained on newer, larger, and diverse malware datasets containing real-world malicious samples [11] to be viable in the future. There was a stark difference in the quality of models between the anti-malware industry [9] and academia due to the lack of high-quality and large datasets. This gap significantly narrowed when SOPHOS released the SOREL-20M dataset. This gave researchers like us the access to 10 million malicious samples to create drastically better models. We apply the learnings gained through the literature survey on this dataset and propose our system to enhance their work.

The remaining chapter is organized as follows. Section 3 describes the design of our proposed system. In Sect. 4, the technical details and relevant theory of the dataset and the components of our system are presented. The results obtained from training, testing, and validation of the malware classification system are presented in Sect. 5. The user facing component of this system is also shown here. Section 6 summarizes the proposed work and the principal findings of our research. The promising research directions and further improvements in this project are also presented here.

3 Proposed Design

Figure 1 shows the overview of our proposed malware classification model. It consists of 4 main components: creation of malware images, autoencoder, GAN, and malware classifier.

The process begins by creating malware images. The binary malware executable files are collected from the SOREL- 20M dataset repository, and we convert them into images using a custom algorithm. We proceed to train the autoencoder using these images. The autoencoder converts these images into a latent representation of

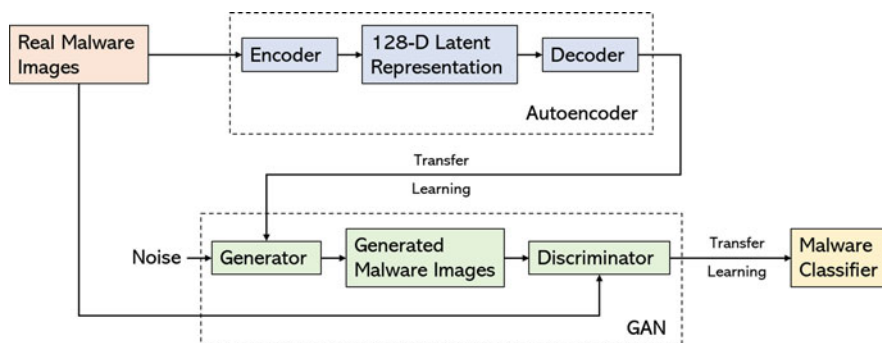


Fig. 1 Proposed model

128 dimensions and converts them back into images. The weights of the decoder are then transferred to the generator. This process provides stability to the training process of the GAN and helps it converge. The GAN model is now trained, and the discriminator of the GAN model is taken out and used for the malware classification. We proceed to describe in detail each phase of the model.

3.1 Creation of Malware Images

As stated earlier, we use SOREL-20M as the dataset for our model. We obtained the binary executable malware files from the dataset's online repository. These files are stored in zlib compressed format to avoid accidental execution of malware. We began by decompressing these files and obtained binary strings (consisting of 0s and 1s) of the malicious code. These strings were converted into grayscale images by forming groups of 8 bits (1 byte) and converting each group to its decimal representation (an integer from 0 to 255). This integer represents the pixel value in the images. Since the length of each binary string may vary, we cannot convert them to images as it is. We need a standardized width or height for the images and pad the missing bits to the strings. We choose to fix the width and vary the height to ensure that the horizontal features of image are preserved. The width of the image is calculated using Table 1. Note that the file size and the length of the binary string are one and the same.

Once the width of the image is obtained, the number of padded bits is obtained using the following equation:

$$padding\ bits = image\ width - (file\ size \% image\ width).$$

After these calculations, we pad the bits to the binary string and create a grayscale image with the calculated width. Since we use convolutional layers in the subsequent phases, we need all the malware images to be of the same size. After

Table 1 Image width for different file sizes

File size	Image width	File size	Image width
≤ 10 KB	32	200 KB \sim 500 KB	512
10 KB \sim 30 KB	64	500 KB \sim 1000 KB	768
30 KB \sim 60 KB	128	1000 KB \sim 2000 KB	1024
60 KB \sim 100 KB	256	≥ 2000 KB	2048
100 KB \sim 200 KB	384		

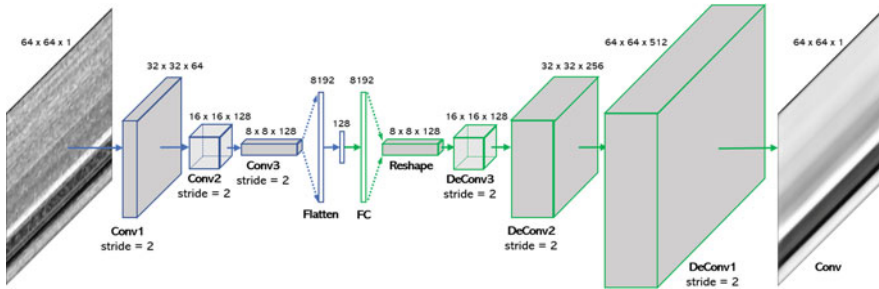


Fig. 2 Architecture of autoencoder

working with several image sizes, we concluded that smaller image sizes provided less precision, recall, and F1 score, while larger image sizes took more time to train. We finally chose 64×64 as our final image size since it was best trade-off between the training time and the evaluation metrics. We resized all the images using cubic and area interpolation. Cubic interpolation was used when image width was less than 64 and area interpolation was used in all other cases. The resized images are then fed to the autoencoder, GAN, and malware classifier for training, testing, and validation.

3.2 Autoencoder

Autoencoders are a type of neural network used to compress the raw data (usually images) and reconstruct it from the compressed form [16]. It consists of two parts: an encoder and a decoder. The encoder compresses the input data to the latent representation by retaining only the most important features, and the decoder tries to reconstruct the original data from this latent representation. Once fully trained, the decoder has the ability to independently generate instances of the original dataset from the latent representation. The architecture of the proposed system’s autoencoder is shown in Fig. 2.

As done previously by J.-Y. Kim et al. [4], we leverage this ability of the autoencoder to create a decoder capable of generating malware images from a latent representation of 128 dimensions. We transfer the weights of this decoder to the

generator of the GAN. This gives it a great starting point and helps stabilize the training process of the GAN.

3.3 GAN

A GAN consists of two parts: the generative network and the discriminative network, called generator and discriminator, respectively. These two halves train by competing with each other. The task of generator is to generate images that are very close to the dataset provided and fool the discriminator. It creates fake data, and this data along with the actual data from the dataset is fed into the discriminator. The task of the discriminator is to correctly distinguish between the real and fake data. This is very similar to a zero-sum game [17].

We identified the need of using adversarial training on machine learning models in our literature survey. We found that GANs inherently make use of adversarial training and that using them would make our models more robust and resistant to adversarial attacks. This feature is lacking in any other deep learning algorithm and motivated us to use GANs. Hence, we use the GAN in our proposed system to perform adversarial training on the discriminator.

The generator generates fake images using noise vector of size 128 dimensions. Since we previously performed transfer learning on the generator, the generator is capable of producing images that are very close to real malware images and may even resemble variants of malware families. The discriminator is trained using these generated and real malware images. After sufficient epochs, the discriminator is able to recognize these types of malware images with ease. Transferring these weights to the malware classifier helps it in classifying malware present in the dataset as well as unseen variants of malware that may occur in the future. Figures 3 and 4 show the architecture of our system's generator and discriminator.

3.4 Malware Classifier

The malware classifier is the final module of the proposed system and performs the classification of the malicious samples to their respective types of malware. Figure 5 shows the general structure of the malware classifier model in our system.

We transfer the weights obtained from the discriminator to the malware classifier. The last layer of the discriminator is replaced by a new fully connected dense layer of size 11, as seen in the figure. This allows the classifier to predict the 11 types of malware and perform multilabel classification. This classifier is much better than any CNN counterpart due to the use of adversarial training.

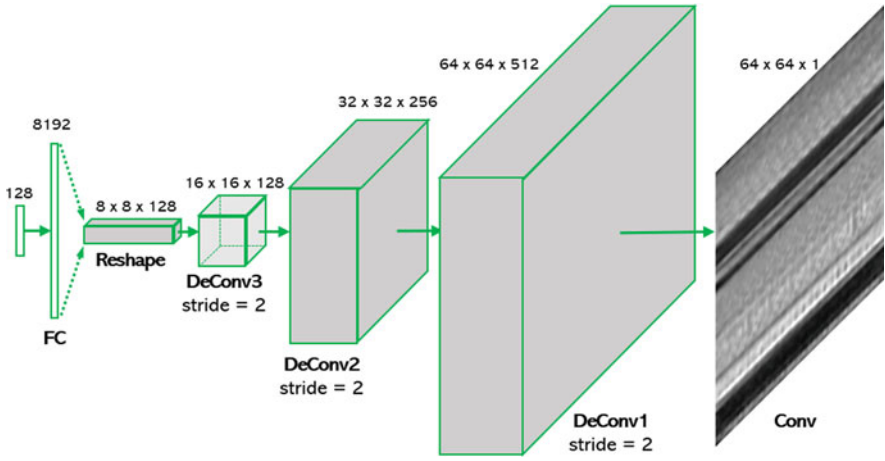


Fig. 3 Architecture of generator

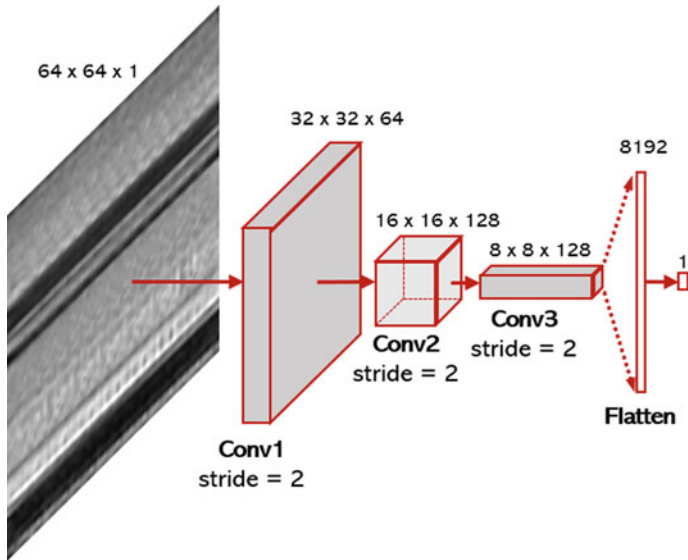


Fig. 4 Architecture of discriminator

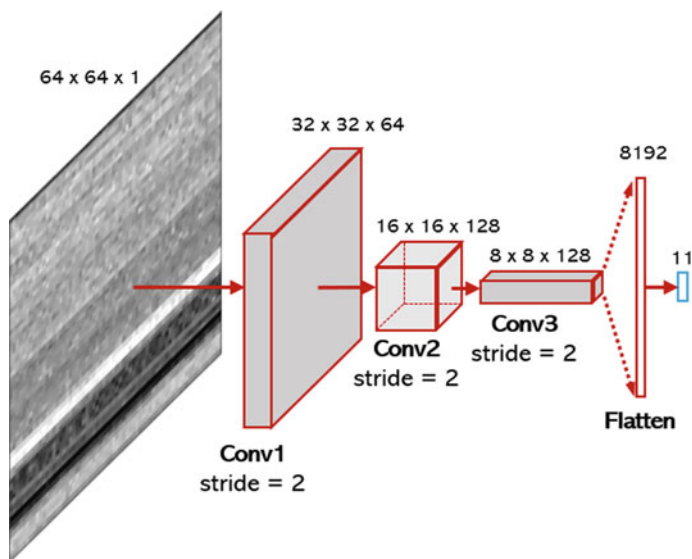


Fig. 5 Architecture of malware classifier

4 Experiments

4.1 Dataset

The SOREL-20M dataset consists of 10 million malicious binary executable files for Windows operating system. It is one of its kind since it provides access to such a large quantity of data for academic research. The dataset provides a database file consisting of the *hash value*, *first seen time*, and 11 columns for *adware*, *flooder*, *ransomware*, *dropper*, *spyware*, *packed*, *crypto_miner*, *file_infector*, *installer*, *worm*, and *downloader* for each malicious file. The columns of the types of malware in these records show the certainty with which each file belongs to a particular type. The higher the number, the higher the certainty. We converted each non-zero value in these columns to a 1 and used it as the labels. Since this dataset contains files with more than one type of malware in it, we chose to perform multilabel classification.

It was infeasible for us to use the entire dataset at this moment since its size is about 8 terabytes. Hence, we chose to train our model on 2 subsets of the dataset: random 10,000 files and the first 100,000 files by first seen time of the malware. The count of each label in the first 100,000 files is shown in Fig. 6. We set aside the first 76.6% of the files as the training set, the next 9.7% as the testing set, and the remaining 13.7% as the validation set [10]. Some of the sample types of files (containing 1 or more types of malware in it) are shown in Table 2 along with their images and its frequency within our subset of 100,000 files. Files containing the malware Spyware, File Infector, and Worm were the most frequent in this subset,

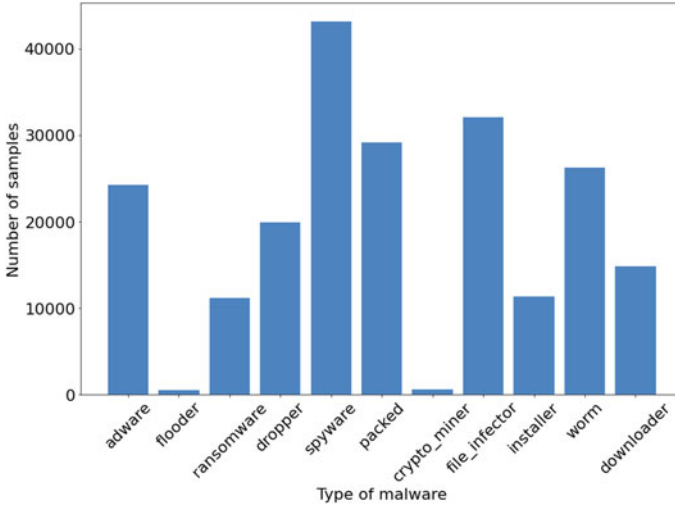


Fig. 6 Label counts for the first 100,000 files

















































while those containing solely Cryptominer or Flooder were the least frequent. We can see from Table 2 that even though files have the same type of malware, their images are very different.

4.2 Experiment Details

All of our experiments were performed using free cloud resources on Kaggle, Google Colab, and Google Cloud. We used the available GPUs to drastically speed up our training times.

We used the Keras library to create all of our models. Convolutional layers form the basis of all of our models since they are key in extracting the important features from the malware images. LeakyReLU was used as the activation function for all convolutional layers except the output layer. LeakyReLU was used instead of ReLU to avoid the dying ReLU problem. Sigmoid function is used as the activation function for the output layer since we are performing multilabel classification. Dropout layer is used after each convolutional layer for regularization and stable learning. Batch normalization layers are added after the dropout layers in the encoder, discriminator, and malware classifier to avoid overfitting. Deconvolution layers were used to rescale and obtain back the images of actual size. We used a stride instead of pooling layers to perform downsampling while training the models:

Table 2 Sample types of files with their frequency and images

Type of malware	No. of data (no. of test data)	Sample images of each type			
Spyware, File Infector, and Worm	11,257 (1086)				
Adware	8228 (814)				
Ransomware, Packed, and File Infector	6566 (641)				
Spyware and File Infector	6187 (573)				
Adware and Installer	3719 (364)				
Dropper, Spyware, and Packed	3150 (310)				
Spyware and Worm	2430 (234)				
File Infector	2219 (218)				
Packed	2073 (196)				
Adware, Installer, and Downloader	2046 (226)				
Flooder and Packed	135 (7)				
Cryptominer	20 (3)				

– Autoencoder

- Encoder: $4 \times 4\text{Conv}@64 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow \text{Flatten}$ to size 8192 \rightarrow Fully connected layer of size 128
- Decoder: Fully connected layer of size 8192 $\rightarrow 4 \times 4\text{DeConv}@128 \rightarrow 4 \times 4\text{DeConv}@256 \rightarrow 4 \times 4\text{DeConv}@512 \rightarrow 5 \times 5\text{Conv}@1$

– GAN

- Generator: Same as autoencoder’s decoder
- Discriminator: $4 \times 4\text{Conv}@64 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow$
Flatten to size 8192 \rightarrow Fully connected layer of size 1

– Malware Classifier

- Decoder: $4 \times 4\text{Conv}@64 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow 4 \times 4\text{Conv}@128 \rightarrow$ Flatten
to size 8192 \rightarrow Fully connected layer of size 11

The batch size was set at 32 for all the models. We used the Adam optimizer for training the models with $\text{beta}_1 = 0.5$ and loss function set as Binary Crossentropy. The learning rate was set to 10^{-4} for the autoencoder, the generator, and the discriminator. The autoencoder is trained for 150 epochs, while the GAN is trained for 200 epochs. In case of the malware classifier, the training is done in 2 phases. First, all the layers except the dense layer of size 11 are frozen. The model is then trained for 500 epochs with learning rate set to 10^{-3} . Second, the remaining layers are unfrozen, and the model is trained again for 400 epochs at a much lower learning rate of 10^{-5} to fine-tune it. We note that fine-tuning improved the results of the malware classifier.

5 System Evaluation and Results

We trained our proposed model on random 10,000 samples and the first 100,000 samples by first seen time of the malware. We also compared the performance of our model with a CNN model trained on the same random 10,000 samples. Referring to similar works on multilabel classification [18, 19], we chose the average per-class precision (CP), recall (CR), F1 score (CF1), false-positive rate (CFPR), the average overall precision (OP), recall (OR), F1 score (OF1), false-positive rate (OFPR) as the metrics to evaluate the performance of our classifier.

Table 3 shows detailed class-wise metrics for the model trained on 100,000 samples for reference. Spyware and File Infector show the best results as they are most dominating types in the dataset. Flooder and Cryptominer perform badly due to the lack of samples in this subset, as seen in Fig. 6. This behavior may change when the number of samples is increased.

Table 4 shows the average and overall metrics of all the models. Our proposed models perform much better than the standard CNN model. This justifies the usage of the autoencoder and the GAN in our system.

The metrics for both the proposed models show that our model is able to perform multilabel classification on the given dataset with a relatively high precision, recall, and F1 score and low false-positive rate. The proposed model trained on 100,000 samples performs better than the model trained on 10,000 samples in all metrics except CR and CF1. This is due to drop in precision and recall of the Flooder and

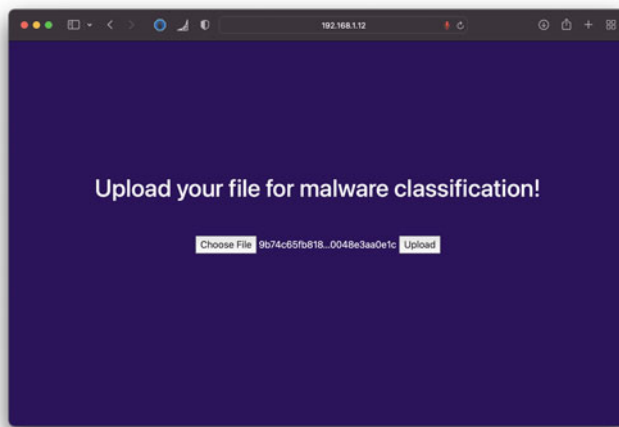
Table 3 Metrics of the model created using 100,000 samples

	Adware	Flooder	Ransomware	Dropper	Spyware	Packed	Crypto Miner	File Infector	Installer	Worm	Downloader
Precision	0.8474	0.4872	0.8620	0.8043	0.9019	0.8311	0.2292	0.8966	0.7753	0.8873	0.7585
Recall	0.8267	0.4043	0.8677	0.7696	0.8736	0.8459	0.1746	0.8894	0.7873	0.8852	0.7391
F1 score	0.837	0.4419	0.8648	0.7866	0.8875	0.8385	0.1982	0.8930	0.7812	0.8863	0.7487
False-Positive rate	0.064	0.0025	0.0218	0.0601	0.1080	0.092	0.0047	0.0658	0.0368	0.0522	0.0524

Table 4 Comparing the results of various models

	CP	CR	CF1	CFPR	OP	OR	OF1	OFPR
CNN—10k Samples	0.7141	0.6808	0.6962	0.0730	0.7961	0.7703	0.7830	0.0613
Proposed model—10k samples	0.7368	0.7627	0.7489	0.0654	0.8165	0.7984	0.8074	0.0558
Proposed Model—100k samples	0.7528	0.7330	0.7422	0.0509	0.8410	0.8531	0.8470	0.0449

The bold values indicate the best value in a given metric amongst the 3 models

**Fig. 7** Landing page of the Web portal

Crypto types of malware, caused by the low count. Hence, this affects only the average metrics and not the overall metrics.

We also deployed the proposed model trained on 100k samples on a website where end users can upload any binary executable file and obtain detailed results on the probability of the file belonging to each type of malware. We created this lightweight web application using Flask. Figure 7 shows the landing page of our website.

By clicking on upload file option, the user can upload a file from their own machine to the website's server. The website converts the file into a 64×64 image using the algorithm stated in Sect. 3.1 and predicts the types of malware present in the file using the malware classification model.

We obtained results for some samples from the test dataset. Here, we discuss the results for one such sample. Figure 8 shows the actual image of the malware on the left side, before resizing it to 64×64 , and the images on the right side are the classification results that are obtained.

The results show 11 probabilities of a malware belonging to a specific type. As can be seen from the figure, the classifier predicts with 100% probability that the file belongs to File Infector and Spyware types and does not belong to any other type with 100% probability, which is the ideal result.

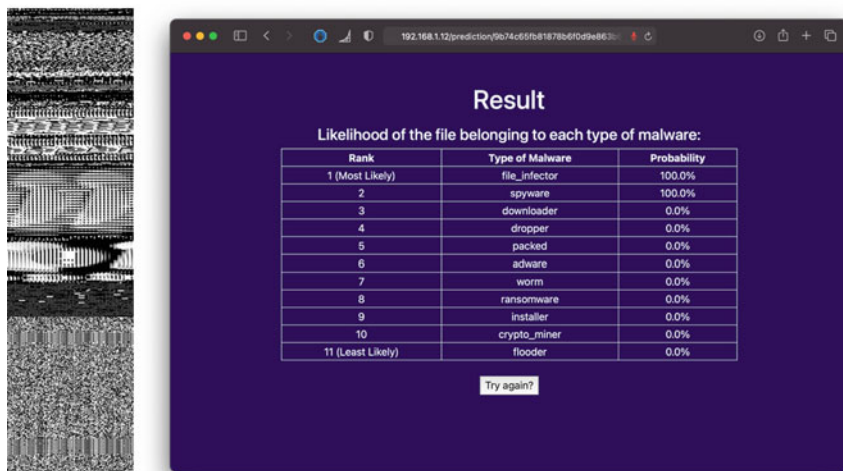


Fig. 8 Results of classification of a malware belonging to Spyware and File Infector types

The website has been designed to be easy-to-use and hassle-free, allowing ordinary users to check the malignity of any file quickly without taking the risk of executing the file. We observe that the main bottleneck of the website is while uploading the file to the server and once a file is uploaded, the system can produce results rapidly.

6 Conclusion and Future Scope

After a thorough literature survey, we observed that the latest malware detection methods have various machine learning and deep learning models at their core. Most of these models have the common issue of failing against specially crafted adversarial samples and variants of the existing malware families. Moreover, the existing models were trained on older and much smaller datasets. There was a need for models that could withstand such attacks while providing a low false-positive rate, accurate results, scalability, and a fast response time.

Our proposed system is trained on a massive dataset with various types and families of malware and utilizes the power of autoencoders and the adversarial training of GANs to solve this problem. It is designed to identify all types of malware present within a malicious sample with a high degree of certainty. Our best model achieves an overall precision of 84.1%, an overall recall of 85.31%, an overall F1 score of 84.7%, and a false-positive rate of 4.49%, outperforming conventional neural network models. It is deployed on a website, allowing the end users to upload any executable file to the website and check its malignity. Due to the use of GANs,

the model will withstand future variants within the malware families and be viable for a long time in the future.

In the future, we will explore the effect of larger image sizes such as 192×192 and 384×384 on the model. Further, we used a subset and not the entire SOREL-20M dataset, which limited our model. We plan to explore the effect of training a model with such a large number of samples. We also need to manually test our model against well-known adversarial attacks to strengthen it. Lastly we intend to expand our model by integrating benign samples into the training process. This would ensure that the model becomes a full-fledged malware detection system and can distinguish between malicious and benign files.

References

1. Wikipedia Contributors. “Malware”, in Wikipedia, The Free Encyclopedia. Accessed: Jul. 13, 2021. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=Malware&oldid=1033368871>
2. “What Happens If Your Computer Is Infected by Malware?”, Consolidated Technologies, Inc. Accessed: Jun. 15, 2021. [Online]. Available: <https://consoltech.com/blog/what-happens-if-your-computer-is-infected-by-malware>
3. Z. Cui, F. Xue, X. Cai, Y. Cao, G. Wang and J. Chen, “Detection of Malicious Code Variants Based on Deep Learning”, IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3187–3196, July 2018, <https://doi.org/10.1109/TII.2018.2822680>.
4. 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. 460–461, pp. 83–102, Sep. 2018, <https://doi.org/10.1016/j.ins.2018.04.092>.
5. J.-Y. Kim and S.-B. Cho, “Detecting Intrusive Malware with a Hybrid Generative Deep Learning Model”, in Intelligent Data Engineering and Automated Learning - IDEAL 2018, Springer International Publishing, 2018, pp. 499–507. https://doi.org/10.1007/978-3-030-03493-1_52.
6. J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, “Black-box adversarial attacks against deep learning based malware binaries detection with GAN,” in the 24th European Conference on Artificial Intelligence (ECAI 2020), pp. 2536–2542, <https://doi.org/10.3233/FAIA200388>.
7. R. Podschwadt and H. Takabi, “Effectiveness of Adversarial Examples and Defenses for Malware Classification,” arXiv:1909.04778 [cs], Sep. 2019.
8. H. Li, S. Zhou, W. Yuan, J. Li and H. Leung, “Adversarial-Example Attacks Toward Android Malware Detection System,” in IEEE Systems Journal, vol. 14, no. 1, pp. 653–656, March 2020, <https://doi.org/10.1109/JSYST.2019.2906120>.
9. “Machine Learning Methods for Malware Detection,” Kaspersky Lab, Moscow, Russia. Accessed: Jun. 14, 2021. [Online]. Available: <https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf>
10. R. Harang and E. M. Ruddy, “SOREL-20M: A Large Scale Benchmark Dataset For Malicious PE Detection,” arXiv:2012.07634v1 [cs.CR], Dec. 2020.
11. D. Gibert, C. Mateu, and J. Planes, “The rise of machine learning for detection and classification of malware: Research developments, trends and challenges,” Journal of Network and Computer Applications, vol. 153, p. 102526, Mar. 2020, <https://doi.org/10.1016/j.jnca.2019.102526>
12. R. Komatwar and M. Kokare, “A Survey on Malware Detection and Classification,” Journal of Applied Security Research, vol. 16, no. 3, pp. 390–420, Aug. 2020, <https://doi.org/10.1080/19361610.2020.1796162>

13. “Internet Security Threat Report”. NortonLifeLock Inc., Tempe, Arizona, U.S. Accessed: Jun. 15, 2021. [Online]. Available: <https://docs.broadcom.com/doc/istr-22-2017-en>
14. I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al., “Generative Adversarial Networks”, arXiv:1406.2661 [stat.ML], Jun. 2014.
15. “How Many Computers Are There in the World?”, Aug. 9, 2019. Accessed: Aug. 20, 2021. [Online]. Available: <https://www.scmo.net/faq/2019/8/9/how-many-computers-is-there-in-the-world>
16. Wikipedia Contributors. “Autoencoder”, in Wikipedia, The Free Encyclopedia. Accessed: Nov. 24, 2021. [Online]. Available: <https://en.wikipedia.org/wiki/Autoencoder>
17. Wikipedia Contributors. “Generative adversarial network”, in Wikipedia, The Free Encyclopedia. Accessed: Nov. 24, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Generative_adversarial_network
18. A. Shalaginov and K. Franke, “A deep neuro-fuzzy method for multi-label malware classification and fuzzy rules extraction,” 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1–8, <https://doi.org/10.1109/SSCI.2017.8280788>.
19. J. Lanchantin, T. Wang, V. Ordonez, et al., “General Multi-label Image Classification with Transformers”, arXiv:2011.14027 [cs.CV], Nov. 2020.