CAD-FSL: Code-Aware Data Generation based Few-Shot Learning for Efficient Malware Detection

Sreenitha Kasarapu
George Mason University
FairFax, Virginia, USA
skasarap@gmu.edu

Sanket Shukla
George Mason University
FairFax, Virginia, USA
sshukla4@gmu.edu

Rakibul Hassan
George Mason University
FairFax, Virginia, USA
rhassa4@gmu.edu

Avesta Sasan
University of California Davis
Davis, California, USA
asasan@ucdavis.edu

Houman Homayoun
University of California Davis
Davis, California, USA
hhomayoun@ucdavis.edu

Sai Manoj PD
George Mason University
FairFax, Virginia, USA
spudukot@gmu.edu

ABSTRACT

One of the pivotal security threats for embedded computing systems is malicious software *a.k.a* malware. With efficiency and efficacy, Machine Learning (ML) has been widely adopted for malware detection in recent times. Despite being efficient, the existing techniques require updating the ML model frequently with newer benign and malware samples for training and modeling an efficient malware detector. Furthermore, such constraints limit the detection of emerging malware samples due to the lack of sufficient malware samples required for efficient training. To address such concerns, we introduce a code-aware data generation-based few-shot learning technique. CAD-FSL generates multiple mutated samples of the limitedly seen malware for efficient malware detection. Loss minimization ensures that the generated samples closely mimic the limitedly seen malware, restore malware functionality and mitigate the impractical samples. Such developed synthetic malware is incorporated into the training set to formulate the model that can efficiently detect the emerging malware despite having limited (few-shot) exposure. The experimental results demonstrate that with the proposed "Code-Aware Data Generation" technique, we detect malware with 90% accuracy, which is approximately 9% higher while training classifiers with only limitedly available training data. Also, our proposed technique has a better weighted F-1 score, about 10-15% improvement is observed compared to other similar works.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**;

KEYWORDS

Malware Detection, Few-shot Learning, Random Obfuscation, Stealthy Malware, Computer Vision

GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9322-5/22/06…$15.00
https://doi.org/10.1145/3526241.3530825

ACM Reference Format:
Sreenitha Kasarapu, Sanket Shukla, Rakibul Hassan, Avesta Sasan, Houman Homayoun, and Sai Manoj PD. 2022. CAD-FSL: Code-Aware Data Generation based Few-Shot Learning for Efficient Malware Detection. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530825

1 INTRODUCTION

With the technical developments in hardware architecture and embedded systems, IoT applications have procured enormous interest in the past few decades [5]. Currently, manufacturers are experiencing an immense desire to automate user applications and produce interactive software systems such as smart homes, digital monitoring, and smart grid. Most of these connect to the internet over a network for communicating between devices. It is important for these devices to communicate because many devices have medical and activity tracking abilities. These systems handle vast amounts of user data daily and are vulnerable to security threats [4] due to malware. Malware is malicious software developed to infect a system to explore and steal information such as passwords, financial data, and manipulate the stored data without the user's consent. There have been more than 5 billion malware attacks worldwide, only in the year, 2020 [12].

Despite the advanced anti-malware software, malware attacks increase because of the newer emerging-malware signatures each year. Adversaries generate millions of new signatures of malware each year [1] to steal valuable information for financial benefit and stay undetectable. The tremendous rise in the volume of malware attacks poses a severe threat to hardware security [11]. Thus it is vital to detect the malware, as it can exploit confidential user information, leading to a substandard user experience. Realizing the threat caused by malware in terms of stolen information, access to sensitive information, and billions of revenue loss, severe measures are being taken to abate malware escalation.

Static and dynamic analysis [9] are employed for malware detection. Static analysis [9] is performed in a non-runtime environment by examining the internal structure of malware binaries and not by actually executing the binary executable files. In dynamic analysis, the binary applications are inspected of malware, traces by executing in a harmless, isolated environment [9, 19]. Unlike static analysis, dynamic analysis is a functionality test. Static analysis serves as a quick testing but not efficient and dynamic analysis can't detect hidden malware.

As the malware classes evolve over time, so do the detection methods. Works on Malware Detection techniques using a variety of Machine Learning (ML) and Federated Learning techniques [21] show better detection accuracy than the static and dynamic analysis methods [18]. Among the ML-based malware detection techniques, the CNN-based image classification technique [10] is more robust and efficient due to its prime ability to learn image features. However, one of the main challenges with adopting such a technique is the requirement of enough samples for training. With the exponential increase in the generation of newer malware families each year, it is complex to obtain a sufficient number of malware samples for each new malware class.

Furthermore, malware developers implement code obfuscation, metamorphism, and polymorphism [6, 24] to mutate malware binary executables. Code obfuscation is a technique where specific parts of the code in malware binary files are encapsulated, masking the malware's behavioral patterns without affecting its functionality. Adversaries use this to evade detection and prolong their presence in the embedded systems to exploit their security. A new strategy in masking malware's identity is stealthy malware [22], where malware is incorporated into benign applications using random obfuscation techniques. Malware attackers also sneak malware into benign application files to hide its vulnerability. This stealthy malware does not exhibit many malware characteristics. Some stealthy malware can take a long time before revealing any malicious activity and adversely impact system security. It is highly impossible to collect an extensive amount of such data for training.

In this work, we address all the issues mentioned above. We propose a technique that can effectively detect complex malware with only limited data. We propose a code-aware data generation-based few-shot technique that can generate mutated training samples and capture the features of actual samples. The generated images mimic the limitedly seen malware functionality and resolve the demand for comprehensive data acquisition.

The novel contributions of this work can be outlined in a three-fold manner:

- Introduces a code-aware data generation architecture for increasing the training dataset.
- Loss minimization is employed to ensure, that the generated data, captures the code patterns of limitedly seen complex malware and its functionality is preserved.
- Few-shot learning is used to classify complex stealthy malware and code obfuscated malware efficiently.

2 RELATED WORK

Static analysis [9] on malware data is performed by comparing the opcode sequences of binary executable files, control flow graphs, and code patterns. The main drawback of static analysis is, it cant detect malware when adversaries add junk of unrelated functionalities, which decreases the malware similarity score [17]. Malware detection using dynamic analysis is performed based on detecting system calls or HPC [19]. But they are not efficient in detecting hidden malware code blocks and are computationally expensive.

Later [18] introduced a technique for malware detection using image processing where binary applications are converted into grayscale images. The generated images have identical patterns because of the executable file structural distributions. The paper used the K-Nearest Neighbour ML algorithm for the classification of malware images. Other approaches [13] include image visualization and classification using machine learning algorithms such as SVM. However, these approaches don't address the problem of classifying newer complex malware that is code obfuscated, polymorphed, etc. Neural networks such as ANNs are used extensively to solve the problem [16], as neurons can capture the features of the images more accurately than other machine learning algorithms. But, the fully connected layers of artificial neural networks tend to exhaust computational resources. In [10] authors used Convolutional neural networks, as they are popular for their ability to efficiently handle image data through feature extraction by Convolutional 2D layers and using Maxpooling 2D layers to down sample the input parameters, thus, reducing the computational resources. The drawback here is, they need to be trained with a balanced dataset to perform classification efficiently, but with an increase in malware families, collecting each type of malware for training is challenging.

We, therefore, need an efficient model which can address all the concerns mentioned above. Also, the model must efficiently classify code obfuscated and stealthy malware without the need for a vast training dataset. In this paper, we address this by developing a code-aware generator that can generate realistic images. These generated images can replicate features of various malware families, solving the problem of training the CNN for efficient malware detection with limited samples.

3 PROBLEM FORMULATION

The motivation for the proposed malware detection is the limited available malware data and the need to train a classification model for efficient malware detection with this limited data. So in this work, we address the problem of constructing an efficient model for malware-based with limited exposure to data. The problem is constructed as follows:

$$D = \{B + M + M_O + M_{ST}\} \tag{1}$$

Given a dataset D containing n samples $(D_1, D_2, .., D_n)$ comprising of four classes benign B, malware M, complex random obfuscated malware M_O and stealthy malware M_{ST} as shown in equation 4. Dataset D has the required number of samples n to train a Machine Learning model for efficient malware detection. But, the complex data available for training is only x number of samples and the limited sample dataset is represented as D_l^x. The x value is less than or equal to ∇% of samples n as shown in equation 2. ∇ is a numerical value and ∇% n are the limited number of samples available for training.

$$D_l^x \subset D^n; \ \forall \ \{x <= \nabla\% n\} \tag{2}$$

Despite the limited complex data available for training an ML model, one should devise a technique that can efficiently classify between these limitedly seen malware data and benign samples.

$$\mathbf{C}{:}(D_l) \Rightarrow (B, M, M_O, M_{ST}) \tag{3}$$

As shown in equation 3, given a dataset D_l containing limited number of complex malware and benign samples, a classifier **C:** must be built which can efficiently classify between limitedly seen classes, such as, benign B, traditional malware M, complex random obfuscated malware M_O and stealthy malware M_{ST}.

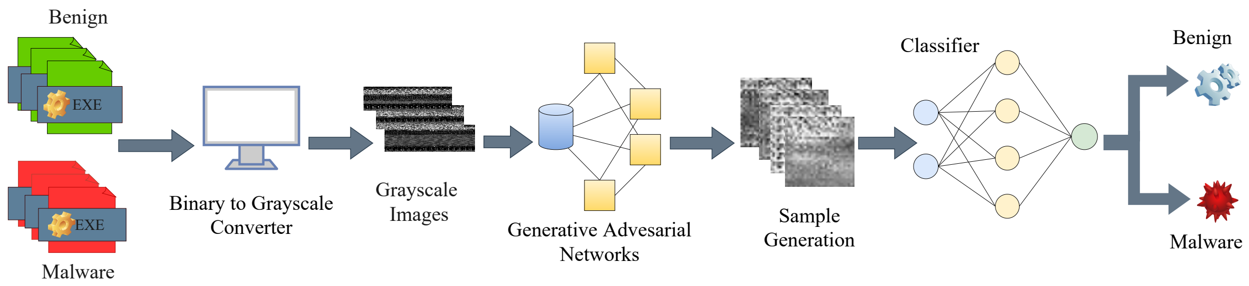

Figure 1: Overview of Proposed Code-Aware Data Generation based Few-Shot Learning Technique

4 PROPOSED TECHNIQUE

4.1 Overview of the Proposed Technique

The overview architecture of the proposed technique is illustrated in the Figure 1, the solution initiates by collecting benign and malware binary files. Among these, malware binaries are a severe threat to the system. These binary files are converted to grayscale images. Stealthy malware is generated using code obfuscated techniques from the available benign and malware samples, while obfuscated malware is generated using the random obfuscation technique. These files are processed as an input to the proposed technique. More images samples of this limitedly seen malware, are generated by training Code-aware Generative Adversarial Networks (GANs). The generated synthetic samples help the few-shot learning for efficient malware detection .

Algorithm 1 Pseudo-Code for proposed technique

```
1: Input: $(B_{exe})$ (Benign application files), $(M_{exe})$ (Malware application files), $B$ (Benign grayscale images), $M$ (Malware grayscale images), $M_O$ (Random obfuscated malware), $M_{ST}$ (Stealthy malware), $D_w^n$ (Whole Dataset with n samples)
2:   $\{(B_{exe}),(M_{exe})\}^i; \forall\ i \leftarrow (1,2,3,...n)$
3:   $\{B, M\} \Rightarrow Input\_images$
4:   stealthy_generator$(B_{exe}, M_{exe}) \Rightarrow M_{ST}$ {// Algorithm 2}
5:   random_obfuscator$(M) \Rightarrow M_O$ {// Algorithm 3}
6:   $D_w^n = \{B + M + M_O + M_{ST}\} \Rightarrow D_l^x$
7:   CAD_generator$(D_l) \Rightarrow D_{mu}$ {// Algorithm 4}
8:   $Proposed = \{D_l \cup D_{mu}\}$
9: Output: Classifier$(Few-shot) \Rightarrow (B, M, M_O, M_{ST})$
```

Algorithm 1 describes the pseudo-code for the proposed technique. The binary application files exe are converted into grayscale images for training the CNN model. The benign, and malware application files are represented as B_{exe} and M_{exe}, containing n number of samples each. Random Obfuscated malware M_O is generated using malware M image samples, which are passed through the random obfuscation generator represented as **random_obfuscator**(M). Stealthy malware M_{ST} samples are generated using the combination of benign and malware binary samples, which are passed through the stealthy generator represented as **stealthy_generator**(B_{exe}, M_{exe}). The benign and malware samples are combined iteratively to generate M_{ST} samples.

The dataset D_w is the entire pool of samples comprising benign B, malware M, obfuscated malware M_O, and stealthy malware M_{ST} classes each containing n number of samples. We create limited data samples D_l^x from the dataset D_w. The number of samples in D_l is represented by x, whose value is given as ∇% of the original samples n, randomly picked from each class of the dataset D_w. By utilizing dataset D_l, generative adversarial networks are trained to generate code-aware mutated fake images, represented with the function **CAD_generator**(D_l^x). The mutated images are stored in dataset D_{mu} and limited data version dataset D_l are combined to create proposed dataset Few-shot with k samples. Multi-class classification is performed on datasets Proposed using the CNN classifier which is the output of algorithm 1.

4.2 Data Generation

The steps shown in Figure 1 are followed to generate input data for classification. The generator takes the binary applications file as input and converts them into a raw binary bitstream. This binary bitstream is then converted to an 8-bit vector. Each 8-bit vector containing the binary values is taken as a byte, representing different image pixels. As the output image is grayscale, 0 represents white, and 255 represents black color in the grayscale image, the rest of the pixels are intermediate shades of gray.

4.2.1 Stealthy Malware Generation. Stealthy malware is created by obfuscating malware into benign. It hides the malware functionality in the benign processes, as it has both malware and benign elements. A code obfuscator is built to generate stealthy malware using input M_{exe} and B_{exe} files.

Algorithm 2 Stealthy Malware Generation Algorithm

```
1: Input: $(B_{exe})$ (Benign application files), $(M_{exe})$ (Malware application files)
2:   define $stealthy\_generator(B_{exe}, M_{exe})$:
3:     for $i \leftarrow range(len(n))$: do
4:       for $j \leftarrow range(len(n))$: do
5:         $(B_{exe})^i \oplus (M_{exe})^j \Rightarrow M_{ST}$
6:       end for
7:     end for
8:     return $M_{ST}$
9: Output: $M_{ST}$ (Stealthy Malware)
```

Algorithm 2 represents the steps to be followed to generate stealthy malware. The random obfuscation function represented as $(B_{exe}) \oplus (M_{exe})$ takes in the benign and malware binary files as input, breaks the code blocks in both files, converts certain parts of the code unreadable and shuffles the code structures from both files to create a single binary file. Code obfuscation changes the code structure of binary files, but it still retains the malware functionality. Identifying code obfuscated malware is complex as it contains the majority of traces of benign features. The generator is iterated to cover all the possible combinations of i and j between n number of benign and malware files, to give output stealthy malware M_{ST}.

4.2.2 *Random Obfuscated Malware Generation.*

Random obfuscation is a technique that can mask the malware functionality in a sample. We generate random obfuscated malware samples as a part of the dataset. The process of randomly obfuscating a malware image is iterative, where in each iteration, a random obfuscator considers a different input image, divides it into segments, and randomly re-arranges it. However, it is essential to note that the random obfuscation retains the malicious functionality of the obfuscated malware image.

Algorithm 3 Random Obfuscated Malware Generation

```
 1: Input: $M = (M_1, M_2, M_3, ....M_n), \forall\ n \leftarrow (1, 2, 3, ...n)$,
    $w, h \leftarrow M \cdot size$ (Size of the malware image)
 2:   define $random\_obfuscator(M)$:
 3:     for $M \leftarrow range(0, n)$: do
 4:       for $i \leftarrow range(0, h, height)$: do
 5:         for $j \leftarrow range(0, w, width)$: do
 6:           $box \leftarrow (j, i, j + width, i + height)$
 7:           $list \leftarrow M \cdot crop(box)$ ;
 8:           $list \leftarrow rand(list)$
 9:           $M_O \leftarrow vstack(list)$
10:         end for
11:       end for
12:     end for
13:     return $M_O$
14: Output: $M_O$ (Random Obfuscated Malware)
```

The algorithm 3, describes the steps involved in randomly obfuscating malware samples. Malware images are taken as input represented as M. As the width w of these images is fixed to 256 and height h is variable, to find the height of the input images $M \cdot size$ function is used. The images are segmented vertically, with the height of the segments fixed to 10, and the width is constant to 256. The malware M images are cropped using the $crop()$ function, based on box coordinates, creating segments each of size $(256, 10)$. Using these coordinates, a box function is constructed and iterated to cover the whole image. The segments from M images are then added to a list as represented in the algorithm. The segments are shuffled using the $rand()$ function and randomly reconstructed by vertically stacking the segments using $vstack()$ function, to create randomly obfuscated malware images M_O as output. Nevertheless, it still has all of its components and unchanged functionality with the added benefit of its ability to mask malware components in itself and make it hard to detect.

4.2.3 *Code-Aware Data Generation.*

Code-aware data generation is a novel approach to generate mutated data for malware detection. Code-aware data generation is a technique where pseudo data is generated using malware binary code. The malware binaries are converted into images, these images capture the code patterns, representing the malware functionalities, and are then given as input to generate more samples. The generated data interpret the malware behavior, making them malware code-aware samples. The generated images are loss-controlled, so they are good at capturing limited malware data features. We use generative adversarial networks (GANs) to perform this task. GANs consist of two neural networks known as generator and discriminator. Generators consider a random uniform distribution as a reference to generate new data points. The generator works to generate fake images which are similar to the input data. The discriminator's work is to classify between the fake images produced from the generator and the real images. The generator and discriminator try to control their loss function. The generator learns to generate images much similar to the real images the discriminator learns to classify them better. With enough training steps, the generator generates realistic images and can be used as a data generator.

Algorithm 4 Code-Aware Data Generation Algorithm

```
 1: Input: $D_l$ (Dataset with limited data version), $B$ (Benign
    grayscale images), $M$ (Malware grayscale images), $M_O$ (Random
    obfuscated malware), $M_{ST}$ (Generated Stealthy malware),
    $D_l = \{B + M + M_O + M_{ST}\}$
 2:   define CAD_generator(X):
 3:     for $X \leftarrow D_l$: do
 4:       for $epoch \leftarrow range(1000)$: do
 5:         $G\_model = define\_generator()$
 6:         $D\_model = define\_discriminator()$
 7:         $noise \leftarrow vector(256, None)$
 8:         $X\_fake \leftarrow G\_model(noise)$
 9:       end for
10:       $D_{mu} \leftarrow G\_model \cdot predict(vector)$
11:       end for
12:       return $D_{mu}$
13: Output: $D_{mu}$ (Generated dataset with mutated samples)
```

The algorithm 4 takes in the limited version dataset D_l as input. For each class in the dataset, the CAD_generator(X) function trains a generator and a discriminator. We train our GAN for 1000 epochs, enough times to minimize the loss and generate images similar to training data. As represented in the algorithm, the generator model is described as *G_model*, and the discriminator model is described as *D_model*. They are convolutional neural networks, where, *G_model* is trained to generate an image when a latent space is given as input. As represented in the algorithm, when a latent noise generated by function $vector()$ of size $(256, None)$ is given as input, it generates an image of size $(32, 32)$. The *D_model* tries to classify the generated fake image X_fake. A loss function is generated for *D_model* and *G_model*. To decrease the gradient loss, the generator learns to generate better fake images X_fake, and the discriminator keeps on learning to classify them. After 1000 epochs, the generator model learns enough to be able to generate realistic fake images. So vectors of latent spaces are created to generate mutated data by using the $model \cdot predict()$ function, they are represented using dataset D_{mu}.

$$D_{mu}(X) \sim D_w(X) \tag{4}$$

The samples in the generated synthetic dataset represented as $D_{mu}(X)$ have high correlation with real samples $D_w(X)$.

4.3 Classification

Convolutional Neural networks are used for the classification task. CNNs are a type of neural network built to mimic the human visual cortex. The ability of CNN to extracts features from an image has made it prominent to be used in visual recognition tasks such as image classification or object detection.

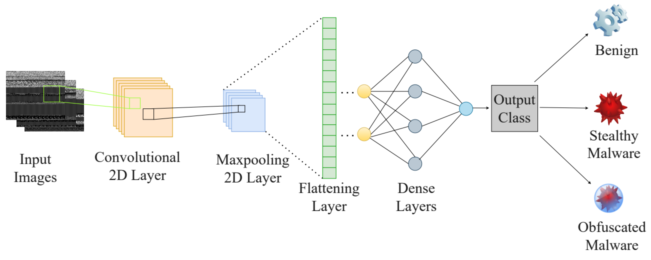

Figure 2: Classifier trained for Malware Detection

The dataset contains limited samples from four classes benign, malware, stealthy malware, and code obfuscated malware. The CNN model is trained with these limited samples and synthetic data generated from trojan, backdoor, worm, virus malware families, as shown in Figure 2. Feature extraction is done on the images using the convolutional 2D layers of the CNN architecture with (3 x 3) filters, followed by the max-pooling 2D layers. The data is then flattened to pass it to the dense layers. All the layers of the CNN architecture are built using a *'relu'* activation function, except the dense output layer, which is equipped with a *'softmax'* activation to produce probabilities for each class. The class with highest probability is classified to be the output class. The performance of the CNN model on different datasets and comparisons with other ML models are analyzed in Section 5.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

The proposed methodology is implemented on an Intel core Nvidia GeForce GTX 1650 GPU with 16GB RAM. We have obtained malware applications from VirusTotal [2] with 12500 malware samples that encompass 5 malware classes: backdoor, rootkit, trojan, virus, and worm. We collected about 13700 benign application files, which are harmless to work with. From malware families backdoor, trojan, virus, and worm, 4800 random obfuscated malware samples are generated. About 6000 stealthy malware samples are generated. The limited version of whole dataset D_w with less than 30% of data samples, is represented as D_l. Few-shot is the case where we train the CNN model with dataset D_l and the synthetic data generated using Code-Aware Data generation technique. We test the CNN on rootkit data, to understand how the proposed techniques generalizes on new data.

5.2 Simulation Results

CNNs are trained using the three different datasets D_w, D_l, Few-shot, and the classification metrics are compared with other Machine learning models such as AlexNet, ResNet-50, MobileNet, and VGG-16. As shown in table 1 we evaluate and compare the test accuracy, precision, recall, and F1-score of these ML models. We observed that the CNN model performs best compared to all the other classifiers in all three data setups. CNN attains the highest accuracy of 94.53% when it is trained with the whole dataset D_w, followed by pre-trained Alexnet and VGG-16 models in the whole data setup. It is observed that the Resnet-50 model has the least performance in all the data setups. We can observe an accuracy drop of about 10% in all the models when trained with only 30% of the initial data, i.e., dataset limited D_l. The model with the highest accuracy, when trained with the D_l dataset is CNN with 82.32%, which is low for a neural network and depicts the inefficiency in classification. When ML models are trained with the Few-shot dataset

Table 1: Performance comparison of proposed model with other datasets on different ML algorithms

Model	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
AlexNet	D_w	91.84	91.62	91.58	91.70
	D_l	76.48	76.45	76.46	76.45
	Few-shot	86.15	86.23	86.28	86.32
ResNet-50	D_w	56.79	56.82	57.00	56.82
	D_l	42.25	42.00	42.25	42.25
	Few-shot	45.73	45.74	45.78	44.93
MobileNet	D_w	87.58	87.32	87.45	87.41
	D_l	78.05	78.03	78.05	78.03
	Few-shot	82.25	82.05	82.2	82.15
VGG-16	D_w	91.25	91.10	91.30	91.20
	D_l	74.60	74.30	74.40	74.40
	Few-shot	84.05	84.00	83.00	83.00
CNN	D_w	94.52	94.18	94.40	94.15
	D_l	82.32	82.03	82.21	82.18
	Few-shot	**89.35**	**89.35**	**89.35**	**89.35**
DNN [7]	Few-shot	-	76.6	74.2	75.3
RNN [7]	Few-shot	-	77.8	75.4	76.1
VGG-16 [25]	Few-shot	-	77.1	74.6	75.2
Inception V3 [8]	Few-shot	-	78.6	74.8	76.3
Xception [15]	Few-shot	-	77.8	73.4	76.5
SNN [3]	Few-shot	-	80.3	82.9	81.8

we can see a subsequential increase in classification metrics. We can observe a performance increase in the CNN model, with its accuracy increased to 89.35%. More than 7% of accuracy boosting can be observed compared to D_l, in all models except ResNet-50 with only 3% accuracy boost. As shown in graph 3, models trained on Few-shot are performing better than models trained on D_l and managed to suffer only 5% of accuracy drop compared to models trained on D_w. This is efficient because they too have only 30% of real samples as dataset D_l. The mutated samples in dataset Few-shot are accounting for the performance improvement compared to the dataset D_l. Thus, making the proposed dataset Few-shot a better option, in case of limitedly available data rather than directly using limited data D_l to train the classification model. Compared to existing few-shot learning techniques such as [7, 8, 15, 25], the F-1 score is improved by 10% in the proposed CAD-FSL technique. Compared to the current state-of-the-art ransomware classification, based on few-shot learning (for an imbalanced dataset) [3], our proposed technique has improved the F-1 score by about 8%.

Table 2, has the quantitative analysis between existing works and proposed works. Due to the complex nature of training data, there are only limited examples of these data classifications. We

Table 2: Quantitative measure with existing works

Samples	Existing works		Proposed	
	Input data	Accuracy	Input data	Accuracy
Malware	14733	97.89%	700	92.01%
Ofuscated	1172	96.31 %	340	89.14%
Stealthy	1000	90.80%	400	85.23%

considered about 30% or fewer training samples of which existing works were used and achieved results with a slight accuracy decay. The existing work for stealthy malware [20], achieved 90.08% accuracy, we can suffer only a 5% decrease in that. The existing work for malware [23], has an accuracy of 97.89% and we have a 6% decrease in that. The existing work for obfuscated malware [14], has an accuracy of 96.31% and we have a 6% decrease in that. Considering the number of samples used for training, the proposed technique serves as the best option when only a limited amount of data is available.

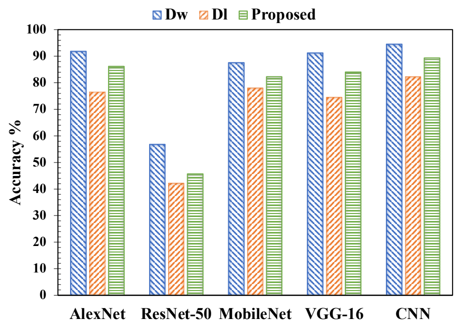

Figure 3: Performance analysis of various ML algorithms

5.3 ASIC Implementation of Proposed Technique for different Classifiers

We conducted a comprehensive hardware implementation of the classifiers embedded into the proposed technique on ASIC. All the experiments are implemented on a Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit, 28 nm SoC running at 1.5 GHz. The power, area, and energy values are reported at 100 MHz. We used Design Compiler Graphical by Synopsys to obtain the area for the models. Power consumption is obtained using Synopsys Primetime PX. The post-layout area, power, and energy are summarized in Table 3. Among all the classifiers, AlexNet consumes the highest power, energy, and area on-chip (Table 3). The post-layout energy numbers were almost 2 × higher than the post-synthesis results. This increase in energy is mainly because of metal routing resulting in layout parasitics. As the tool uses different routing optimizations, the power, area, and energy values keep changing with the classifiers' composition and architecture.

Table 3: Post synthesis hardware results for proposed framework with different ML classifiers (@100MHz)

Classifier	Power (mW)	Energy (mJ)	Area (mm^2)
AlexNet	72.45	5.22	4.5
ResNet-50	68.64	3.75	3.55
MobileNet	64.63	3.79	3.81
CNN	46.44	2.29	2.45
VGG-16	56.46	3.21	3.26

6 CONCLUSION

With the proposed code-aware data generation technique we were able to employ few-shot learning to efficiently classify, limitedly seen malware data. From the results presented, it is evident that the model trained on data, generated through the proposed technique outperforms, the model trained on limited available data, with approximately 9% more accuracy. We also furnished the ASIC implementations of different classifiers trained using the proposed technique. Thus, instead of training the ML models with limited available data, the proposed code-aware data generation technique should be employed.

REFERENCES

[1] 2021. Malware statistics Trends REPORT: AV-TEST. https://www.av-test.org/en/statistics/malware/
[2] 2021. Virustotal package. https://www.rdocumentation.org/packages/virustotal/versions/0.2.1
[3] 2022. A few-shot meta-learning based siamese neural network using entropy features for ransomware classification. *Computers and Security* 117 (2022), 102691.
[4] Omaiyma Abbas and et al. 2016. Big Data Issues and Challenges.
[5] Trio Adiono. 2014. Challenges and opportunities in designing internet of things. *2014 The 1st International Conference on Information Technology, Computer, and Electrical Engineering* (2014). https://doi.org/10.1109/icitacee.2014.7065704
[6] Babak Bashari Rad and et.al. 2012. Camouflage In Malware: From Encryption To Metamorphism. *IJCSNS* (2012).
[7] Manoj Basnet and et.al. 2021. Ransomware Detection Using Deep Learning in the SCADA System of Electric Vehicle Charging Station. In *PES Innovative Smart Grid Technologies Conference-Latin America (ISGT Latin America).*
[8] Li Chen. 2018. Deep transfer learning for static malware classification. *arXiv preprint arXiv:1812.07606* (2018).
[9] Anusha Damodaran et al. 2015. A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques* (2015).
[10] Daniel Gibert and et.al. 2019. Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques* (2019).
[11] M. S. Jalali and et al. 2019. The Internet of Things Promises New Benefits and Risks: A Systematic Analysis of Adoption Dynamics of IoT Products. *IEEE Security Privacy* (2019).
[12] Joseph Johnson. 2021. Number of malware attacks per year 2020. https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/
[13] Kesav Kancherla and et.al. 2013. Image visualization based malware detection. In *Computational Intelligence in Cyber Security (CICS).*
[14] Nitesh Kumar and et.al. 2019. Malware Classification using Early Stage Behavioral Analysis. In *Asia Joint Conference on Information Security (AsiaJCIS).*
[15] Wai Weng Lo and et.al. 2019. An xception convolutional neural network for malware classification with transfer learning. In *Int. Conf. on New Technologies, Mobility and Security (NTMS).*
[16] Aziz Makandar and Anita Patrot. 2017. Malware class recognition using image processing techniques. In *Int. Conf. on Data Management, Analytics and Innovation (ICDMAI).*
[17] Andreas Moser and et.al. 2007. Limits of Static Analysis for Malware Detection. In *Annual Computer Security Applications Conference (ACSAC 2007).*
[18] L. Nataraj and et al. 2011. Malware Images: Visualization and Automatic Classification. In *Int. Symposium on Visualization for Cyber Security.*
[19] Christian Rossow and et.al. 2012. Prudent practices for designing malware experiments: Status quo and outlook. *Symposium on Security and Privacy* (2012).
[20] Sanket Shukla and et.al. 2019. Stealthy Malware Detection using RNN-Based Automated Localized Feature Extraction and Classifier. In *International Conference on Tools with Artificial Intelligence (ICTAI).*
[21] Sanket Shukla and et.al. 2021. On-device Malware Detection using Performance-Aware and Robust Collaborative Learning. In *Design Automation Conference.*
[22] Salvatore J. Stolfo and et.al. 2007. Towards Stealthy Malware Detection. In *Malware Detection.*
[23] Danish Vasan and et.al. 2020. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture.
[24] Ilsun You and et.al. 2010. Malware Obfuscation Techniques: A Brief Survey. In *Int. Conf. on Broadband, Wireless Comp., Comm. and Applications.*
[25] Songqing Yue. 2017. Imbalanced malware images classification: a CNN based approach. *arXiv preprint arXiv:1708.08042* (2017).