A Method for Packed (and Unpacked) Malware Detection by Means of Convolutional Neural Networks

Giovanni Ciaramella[1,2][a] Fabio Martinelli[3][b],
Antonella Santone[4][c] and Francesco Mercaldo[4,2][d]

[1] *IMT School for Advanced Studies Lucca, Lucca, Italy*
[2] *Institute for Informatics and Telematics, National Research Council of Italy (CNR), Pisa, Italy*
[3] *Institute for High Performance Computing and Networking, National Research Council of Italy (CNR), Rende, Italy*
[4] *University of Molise, Campobasso, Italy*

Keywords: Malware, Packed Malware, Obfuscation, Deep Learning, Security, Testing.

Abstract: The current signature-based mechanism implemented by free and commercial antimalware requires the presence of the signature of the malicious sample to provide protection, i.e., to detect malicious behavior. This is why malware writers are developing techniques that can change the syntax of the code but leave the semantics unchanged, i.e., the malware business logic. Among these techniques is the so-called packed malware, i.e., malware with binary code modified by packers, software aimed to pack software, compress it, and package it with a stub. It is a program capable of decompressing and executing it in memory. In this way, malware detected by antimalware is not even detected in the packed version. In this paper, we propose a technique to detect packed malware by exploiting convolutional neural networks. In a nutshell, the proposed method performs static analysis, i.e., it does not require running the application to detect the malicious samples: we start from the application's binary code exploited to generate an image that represents the input for a set of deep learning classifiers. The classifiers aim to discern an application under analysis between trusted or (packed) malicious. In the experimental analysis, we consider three different packers (i.e., mpress, BEP, and gzexe) to generate packed malware, thus demonstrating the ability of the proposed method to detect packed and unpacked malware with interesting performances.

1 INTRODUCTION

The main aim of malware, contraction for malicious and software, is to perpetrate damages on the victim devices (i.e., computers but also smartphones) with particular regard to the information gathering, i.e., the ability to retrieve sensitive information and send this information to the attacker. Malware writers are constantly focused on developing new techniques to evade the signature-based detection provided by current free and commercial antimalware: one of the last trends is represented by the so-called packed malware. The term packed malware refers to malicious software that has been compressed or obfuscated using a technique known as ”packing.” This process involves compressing the code of the malware into a smaller, often encrypted format, making it harder for antimalware programs to detect or analyze. Packing can also involve embedding the malware inside other legitimate software or files, disguising its true nature. In the following, we provide several details related to the packer's working mechanisms:

- Compression and Encryption: The malware's original code is compressed or encrypted. When the packed malware is executed, it decompresses or decrypts itself in memory to carry out its malicious actions.
- Bypassing Detection: Antivirus software often uses signature-based detection, looking for patterns in code to identify malware. Packing changes the malware's appearance, making it look different from known threats.

[a] https://orcid.org/0009-0002-9512-0621
[b] https://orcid.org/0000-0002-6721-9395
[c] https://orcid.org/0000-0002-2634-4456
[d] https://orcid.org/0000-0002-9425-1657

DOI: 10.5220/0013210400003979
In *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, pages 557-564
ISBN: 978-989-758-760-3; ISSN: 2184-7711

- Multiple Layers of Obfuscation: Malware authors sometimes use multiple layers of packing, making it even more challenging to analyze. Security tools must first unpack or decrypt each layer before inspecting the malware's actual code.

Packed malware complicates both static and dynamic analysis static analysis (examining code without running it) is more complex because the code is not in its original form, while dynamic analysis (running the malware in a sandbox environment to observe its behavior) can also be thwarted, as some packed malware checks for signs of being in a virtual environment and behaves differently to avoid detection (Qiang et al., 2022).

Starting from these considerations, we propose a method to detect packed and unpacked malware in this paper. The proposed method relies on a Convolutional Neural Network (CNN) (He et al., 2024). We propose a representation of an application as an image, and we consider training a set of CNN to understand the best performance for classifying (unpacked) malware and trusted samples. In the evaluation, we consider a set of packed malware to comprehend whether the trained model can detect packed malware (not exploited in the training phase).

Thus, the paper proceeds as follows: in the section we review the current state of the art related to (packed) malware detection, in Section 3 we present the method we designed for the detection of packed and unpacked malware; the experimental analysis is discussed in Section 4 and, finally, in the last section conclusion and future research lines are drawn.

2 RELATED WORK

Over the years, due to the increase in cybersecurity, malicious users started introducing several methodologies to curb malware detectors. This Section provides a literature review of several methods that identify packed and unpacked malware.

Authors in (Devi and Nandi, 2012) proposed a method to identify packed and unpacked malware in the Windows environment. In detail, they created two datasets of 4,075 executable applications, where 2,954 were malicious programs and 1,121 were benign executables. The authors applied the UPX packer to the first dataset, while the second dataset used unpacked malware. Moreover, using Weka, they applied classification algorithms for both datasets. Differently from them, we proposed a methodology to identify packed and unpacked malware leveraging Deep Learning by applying several Convolutional Neural Networks belonging to the state of the art. To do that, we converted all datasets created (one composed of unpacked executable applications and three using different packing methods) into images, reaching accuracy values in most of the cases higher than 0.980.

Biondi *et al.* in (Biondi et al., 2019) proposed a method to identify packed malware leveraging three different classification Machine learning algorithms such as Native Bayes, Decision Tree, and Random Forest Extra Trees. In detail, researchers employed a dataset of 280,000 samples on which they applied two packing techniques (UPX and TheMida), extracting many features. Moreover, they also perform classification using unpacked samples. In our proposed method, we employed a methodology based on Deep Learning. We applied three packing algorithms on each malware sample, obtaining four datasets (one composed of unpacked malware). In the following step, all Windows executable applications were converted by a script into images and submitted to ten different Convolutional Neural Networks belonging to the literature. Authors in (Rabadi and Teo, 2020) proposed a method to detect malware Windows field using Machine Learning. In detail, they composed a dataset of benign and malicious executable applications and extracted some features to train and test models using two different methods. In the first method, each API call and the list of its arguments are presented as one feature. In the second method, each API call and each element of its arguments are considered as one feature. Consequently, they achieved remarkable results in terms of accuracy. In our proposed approach, we employed Deep Learning to train and test models. Moreover, instead of using API calls, we converted the entire application into an image to create our dataset.

In (Ciaramella et al., 2024), authors proposed a method to classify ransomware, general malware, and trusted applications belonging to the Windows domain. In detail, they employed several Deep Learning architectures, obtaining the best result in terms of accuracy using the VGG16 network. Moreover, on the best model, they also applied the Grad-CAM algorithm to identify which area of images turns out to be crucial for a certain prediction. Unlike them, we employed a dataset of general malware and trusted PE files, for which we applied three different packing methods. Moreover, we also trained and tested models using unpacked malware. Using all created datasets, we reached good results in terms of accuracy.

3 THE METHOD

In this section we present the proposed method for the detection of packed and unpacked malware. Figures 1 and 2 respectively present the training and the testing step related to the proposed method.

Figure 1 shows the workflow of the proposed malware detection method consists of the following steps:

- **Malware Applications.** This step involves collecting applications known to be malware. These samples serve as malicious samples during the training process.
- **Trusted Applications.** This step involves gathering safe or trusted applications. These samples act as legitimate samples in training.
- **Application Label.** Each application is labeled as either ”Malware” or ”Trusted” based on its nature, which will be used as the target variable in the training process.
- **Application Images.** The collected applications are converted into images. To convert a binary to an image, we treat the sequence of bytes representing the binary as the bytes of a gray-scale PNG image. Depending on the binary size, we consider a predefined width of 256 and a variable length. We developed a script to encode any binary file into a lossless PNG (Mercaldo and Santone, 2020).
- **Deep Learning Network.** A deep learning network is then trained on these labeled images to learn features that distinguish malware from trusted applications. We consider several CNNs in this task: ALEX_NET, LE_NET, STANDARD CNN, MobileNet, DenseNet, EfficientNet, ResNet50, VGG16, VGG19 and Inception.
- **Deep Learning Model.** After training, the deep learning model can predict whether a new application is malware or trusted based on the learned features. This model can be deployed for real-time malware detection.

Figure 2 shows the testing step of the proposed method.

As shown from Figure 2, the testing step of proposed malware detection method includes the following steps:

- **Application Packer:** A tool used to create packed versions of applications, often by compressing or encrypting their code. This is used to generate packed applications, which can obfuscate the behavior of the software and make detection more challenging.
- **Packed Application:** Applications processed through the packer. Due to their obfuscation techniques, these packed applications are often harder to analyze and detect as malware.
- **Unseen Malware:** This step involves testing the detection method with new malware samples not part of the training set, representing real-world malware scenarios.
- **Unseen Trusted Application:** Similarly, new trusted applications are used during testing to ensure the model's ability to distinguish between malware and safe software.
- **Application Images:** The unseen malware, trusted applications, and packed applications are transformed into images, which serve as inputs for the deep learning model.
- **Deep Learning Model:** The trained deep learning model from the training phase analyzes the application images and predicts their nature as either malware or trusted.
- **Malware/Trusted Prediction:** The model predicts each application, determining whether it is malware or trusted. This output helps evaluate the model's performance on unseen and packed samples.

4 EXPERIMENTAL ANALYSIS

To collect real-world samples to evaluate the proposed method, we consider the following repositories: the first one is the Dike dataset[1], a freely available collection of trusted and malicious Portable Executable (PE) and Object Linking and Embedding (OLE) files. The malware belonging to this dataset represents the unpacked malware and tries a wide spectrum of malware categories, i.e., generic trojan, ransomware, worm, backdoor, spyware, rootkit, encrypter, and downloader.

Moreover, to evaluate the robustness of the proposed method concerning packed malware, we have to generate packed variants, so we consider three different packers *i.e., mpress, BEP* and *gzexe*. With the aim to cover the spectrum of different packers, we experiment with three packers: we consider mpress, as modern and efficient Windows executable compressor with added benefits of space saving and making reverse engineering more difficult. From the other side BEP is an older, less commonly used packer that compresses executables, but has been largely replaced by more modern tools. We take into account also gzexe,

[1]https://github.com/iosifache/DikeDataset

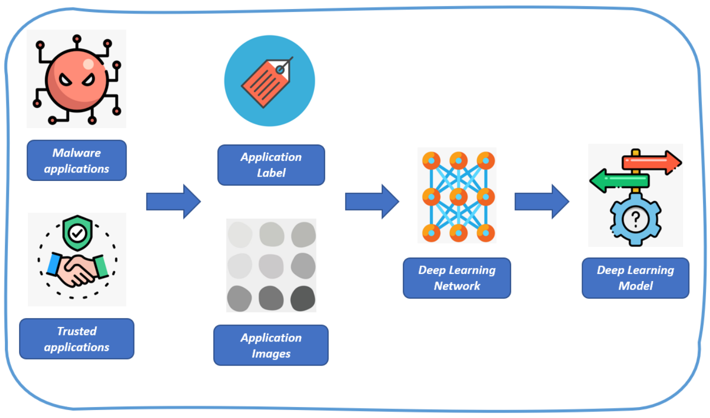

Figure 1: The training phase of the proposed method.

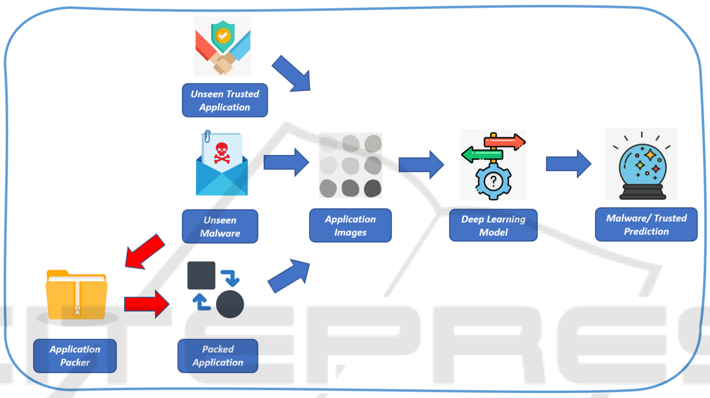

Figure 2: The testing phase of the proposed method.

a simple Linux-based tool that compresses executables using the gzip algorithm for space savings on Unix-like systems.

Thus, the full dataset is composed by following 5000 applications:

- 1000 (unpacked) malware samples;
- 1000 packed malware samples obtained with the mpress packer:
- 1000 packed malware samples obtained with the BEP packer;
- 1000 packed malware samples obtained with the gzexe packer;
- 1000 trusted samples.

We train the deep learning models with the (unpacked) and trusted samples. In the evaluation step, we consider following evaluations:

- with the unpacked malware and the trusted samples (i.e., the O dataset);
- with the packed malware (obtained with the mpress packed) and the trusted samples (i.e., the P1 dataset);
- with the packed malware (obtained with the BEP packed) and the trusted samples (i.e., the P2 dataset);
- with the packed malware (obtained with the gzexe packed) and the trusted samples (i.e., the P3 dataset).

We conduct following experiments to evaluate the effectiveness of the proposed method for the detection of packed and unpacked malware.

Table 1 shows the details related to the hyperparameters we exploited for the considered DL models, in particular we consider the Image size, the learning rate, the epochs and the batch size considered in training.

As shown from Table 1, several well-known architectures are used, including AlexNet, LeNet, Standard CNN, MobileNet, DenseNet, EfficientNet, ResNet50, VGG16, VGG19, and Inception.

From the image size point of view, most models use an input image size of 100x3 (likely representing a height and width of 100 pixels and 3 color channels). However, certain models require larger input sizes due to their architectural design, in fact

Table 1: Hyper-parameters setting.

Model	*Image size*	*Learning rate*	*Epochs*	*Batch size*
ALEX_NET	100x3	0.0001	25	16
LE_NET	100x3	0.0001	25	16
STANDARD CNN	100x3	0.0001	25	16
MobileNet	224x3	0.0001	25	16
DenseNet	224x3	0.0001	25	16
EfficientNet	224x3	0.0001	25	16
ResNet50	100x3	0.0001	25	16
VGG16	100x3	0.0001	25	16
VGG19	100x3	0.0001	25	16
Inception	299x3	0.0001	25	16

MobileNet, DenseNet, EfficientNet use images sized 224x3, while Inception requires an even larger input size of 299x3.

Relating to the Learning Rate, a consistent learning rate of 0.0001 is used across all models. This low learning rate suggests careful training to ensure stability and gradual convergence.

All models are trained for 25 epochs, providing a uniform duration of training across different architectures.

The batch size is fixed at 16 for all models. This suggests that the computational resources available may favor smaller batches, allowing gradient updates after processing small sets of samples.

The table outlines a consistent set of hyperparameters (learning rate, epochs, and batch size) across all models, with variations in the image size depending on the architecture's input requirements. This setup allows for a controlled comparison of model performance under similar training conditions, focusing on the impact of different network architectures.

Table 2 shows the results related to the training and validation of the models involted in the experimental analysis.

Table 2 shows the performance results related to the training and validation performance of the model involved in the experimental analysis. The table provides insight into each model's ability to generalize from the training data to the validation set.

First of all, we note that EfficientNet and ResNet50 achieved the highest training accuracy, precision, and recall (0.994), indicating their robustness during training. However, EfficientNet's validation performance drops significantly (0.498 across accuracy, precision, and recall), suggesting potential overfitting. The MobileNet, DenseNet, and Inception networks showed strong validation performance (accuracy, precision, and recall values close to or above 0.9), suggesting better generalization compared to models like EfficientNet. ALEX_NET and STANDARD CNN both performed well in terms of validation metrics, with STANDARD CNN slightly outperforming ALEX_NET (0.923 vs. 0.879). LeNet, VGG16, and VGG19 performed poorly, with validation accuracy, precision, and recall stuck around 0.5, implying these models struggled to learn effectively from the dataset.

- ALEX_NET: Shows a moderate drop in validation performance compared to training (validation accuracy of 0.879). This could indicate some overfitting, though the gap is not extreme.
- LeNet: Both training and validation metrics are around 0.5, indicating that the model failed to learn useful features, likely due to its simpler architecture being insufficient for the task.
- STANDARD CNN: Exhibits strong performance in both training and validation. This model generalizes well and has a low validation loss (0.295).
- MobileNet: Strong validation performance with a high accuracy of 0.939. The model balances good generalization with relatively low training loss.
- DenseNet: Performs very well on both training and validation data, with validation accuracy at 0.945, indicating effective feature extraction and generalization.
- EfficientNet: While this model performs exceptionally well during training, its validation metrics (0.498) suggest significant overfitting, as it fails to generalize.
- ResNet50: Very balanced, achieving excellent results in both training and validation, with validation accuracy at 0.926.
- VGG16 and VGG19: Both models fail to train effectively, as shown by the high training and validation loss values. Their architecture may not be well-suited for this dataset.
- Inception: Achieves good performance across all metrics, with particularly low validation loss (0.197) and high accuracy (0.943), indicating good generalization.

Models like ResNet50, DenseNet, MobileNet, and Inception are strong candidates due to their balance of low loss and high validation metrics, suggesting better generalization. Simpler models like LeNet and the VGG variants struggled to capture relevant features, and EfficientNet overfitted despite its high complexity.

Thus, from Table 2 emerges that the models obtaining a validation accuracy greater than 0.90 are STANDARD CNN, MobileNet, DenseNet, ResNet50, and Inception: for this reason, these models are considered in the testing step.

Table 3 presents the results of the experimental analysis involving four datasets (0, P1, P2, and P3) and five models (i.e., STANDARD CNN, MobileNet, DenseNet, ResNet50, and Inception).

Table 2: Training and validation performances related to loss, accuracy, precision, recall.

Model	*train_loss*	*train_acc*	*train_prec*	*train_rec*	*val_loss*	*val_acc*	*val_prec*	*val_rec*
ALEX_NET	0.040	0.985	0.985	0.985	0.506	0.879	0.879	0.879
LE_NET	0.693	0.508	0.508	0.508	0.693	0.500	0.500	0.500
STANDARD CNN	0.033	0.988	0.988	, 0.988	0.295	0.923	0.923	0.923
MobileNet	0.037	0.991	0.991	0.991	0.413	0.939	0.939	0.939
DenseNet	0.062	0.984	0.984	0.984	0.237	0.945	0.945	0.945
EfficientNet	0.019	0.994	0.994	0.994	2.682	0.498	0.498	0.498
ResNet50	0.021	0.994	0.994	0.994	0.405	0.926	0.926	0.926
VGG16	0.693	0.496	0.496	0.496	0.693	0.499	0.499	0.499
VGG19	0.693	0.491	0.491	0.491	0.693	0.500	0.500	0.500
Inception	0.061	0.988	0.988	0.988	0.197	0.943	0.943	0.943

Table 3: The results of the experimental analysis.

Model	*Dataset*	*Loss*	*Accuracy*	*Precision*	*Recall*	*F-Measure*	*AUC*
STANDARD CNN	O	0.337	0.921	0.921	0.921	0.921	0.963
	P1	0.180	0.954	0.954	0.954	0.954	0.982
	P2	0.156	0.960	0.960	0.960	0.960	0.984
	P3	0.174	0.956	0.956	0.956	0.956	0.983
MobileNet	O	0.369	0.949	0.949	0.949	0.949	0.970
	P1	0.140	0.974	0.974	0.974	0.974	0.987
	P2	0.167	0.973	0.973	0.973	0.973	0.990
	P3	0.151	0.974	0.974	0.974	0.974	0.988
DenseNet	O	0.214	0.947	0.947	0.947	0.947	0.979
	P1	0.133	0.966	0.966	0.966	0.966	0.988
	P2	0.122	0.968	0.968	0.968	0.968	0.989
	P3	0.120	0.968	0.968	0.968	0.968	0.989
ResNet50	O	0.442	0.920	0.920	0.920	0.920	0.960
	P1	0.248	0.950	0.950	0.950	0.950	0.979
	P2	0.234	0.952	0.952	0.952	0.952	0.980
	P3	0.246	0.950	0.950	0.950	0.950	0.979
Inception	O	0.193	0.941	0.941	0.941	0.941	0.976
	P1	0.175	0.946	0.946	0.946	0.946	0.982
	P2	0.157	0.952	0.952	0.952	0.952	0.987
	P3	0.165	0.950	0.950	0.950	0.950	0.982

- Standard CNN
  - Dataset 0: The Standard CNN model performs well, with a high accuracy of 0.921 and an AUC of 0.963, indicating that the model is effective at distinguishing between malware and trusted samples. However, the loss is relatively higher at 0.337 compared to datasets P1, P2, and P3.
  - Datasets P1, P2, P3: The performance improves consistently across the datasets, with accuracy peaking at 0.960 for P2 and slightly dropping to 0.956 in P3. The loss also decreases, indicating that the model learns better with more complex or larger datasets. The F-measure and AUC remain consistently high, suggesting that the model performs well across these different datasets.
  - The Standard CNN performs robustly across all datasets, showing particularly strong generalization on P2 and P3, with AUC values consistently above 0.98, indicating good discrimination ability.
- MobileNet
  - Dataset 0: MobileNet starts with a high accuracy of 0.949, a lower loss of 0.369 compared to Standard CNN, and an AUC of 0.970. The model shows a very strong balance between precision and recall, both at 0.949.
  - Datasets P1, P2, P3: Performance improves significantly, with accuracy reaching 0.974 for P1 and P3, and the loss dropping to 0.140 for P1. The AUC remains consistently high, around 0.987-0.990, and the F-measure sug-

gests a high degree of consistency between precision and recall.
  - MobileNet performs exceptionally well, improving across larger or more complex datasets. Its ability to generalize is highlighted by its minimal fluctuation in accuracy, precision, and recall. The AUC remains high, confirming its reliability for classification.

- DenseNet
  - Dataset 0: DenseNet shows strong performance with a loss of 0.214 and accuracy of 0.947, but slightly lower than MobileNet for this dataset. Precision, recall, and F-measure are all consistent at 0.947, and the AUC is high at 0.979.
  - Datasets P1, P2, P3: As the datasets change, DenseNet consistently improves, with its loss reducing to 0.120 for P3 and accuracy peaking at 0.968 across P2 and P3. Precision, recall, and F-measure remain stable at 0.968, and the AUC remains around 0.989.
  - DenseNet shows strong and stable performance, with an especially low loss on the P2 and P3 datasets. It has excellent precision, recall, and AUC, making it a reliable model for malware detection.
- ResNet50
  - Dataset 0: ResNet50 starts with the highest loss among the models (0.442) and relatively lower accuracy (0.920). Its precision, recall, and F-measure are all consistent at 0.920, and the AUC is 0.960, indicating moderate discrimination ability.
  - Datasets P1, P2, P3: ResNet50 improves in accuracy and other metrics with P1, P2, and P3, reaching a maximum accuracy of 0.952 for P2. However, its loss remains relatively high compared to other models (0.246 for P3). The AUC remains stable around 0.979-0.980, indicating that the model can still discriminate well between malware and trusted samples.
  - ResNet50 performs solidly on the larger datasets but struggles with higher loss and slightly lower accuracy compared to DenseNet and MobileNet. It does, however, maintain good AUC values across all datasets.
- Inception
  - Dataset 0: Inception shows competitive performance, with an accuracy of 0.941 and loss of 0.193. Its precision, recall, and F-measure are all consistent at 0.941, and the AUC is 0.976, suggesting good performance.
  - Datasets P1, P2, P3: As with the other models, Inception's performance improves with the more complex datasets, reaching an accuracy of 0.952 for P2 and a low loss of 0.157. The AUC peaks at 0.987 for P2, indicating strong discriminatory power.
  - Inception performs well, especially on dataset P2, with high precision, recall, and AUC values. It strikes a balance between performance metrics and generalization capability, though it slightly lags behind MobileNet and DenseNet on the larger datasets.

We note improved performance on the P1, P2, and P3 datasets; all models show better performance on the P1, P2, and P3 compared to dataset 0. This suggests that the complexity or size of these datasets helps the models learn more effectively, resulting in higher accuracy, precision, recall, and lower loss. MobileNet and DenseNet Outperform: MobileNet and DenseNet consistently outperform the other models regarding accuracy, precision, recall, and F-measure, particularly on datasets P1 through P3. Both models maintain low loss values and high AUC, making them strong candidates for the task. ResNet50, while still performing well in terms of accuracy and AUC, suffers from higher loss values, indicating that it struggles to fit the data as effectively as the other models. Inception shows balanced performance across all datasets, with competitive metrics, though slightly behind MobileNet and DenseNet in accuracy and loss. MobileNet and DenseNet stand out for their ability to generalize and maintain high performance across all datasets. Both models show minimal fluctuation in accuracy and have high AUC, indicating their effectiveness in malware classification. The performance improvements across the datasets suggest that the models benefit from more complex datasets (i.e., P1, P2, P3). While the ResNet50 and Inception models perform adequately, their higher loss values and slightly lower accuracy suggest they may not generalize as well as MobileNet and DenseNet, especially in handling more complex data.

These findings indicate that MobileNet and DenseNet would be the most reliable models for malware detection. However, considering that the MobileNet models obtain slightly higher accuracy if compared with the DenseNet one, we consider the MobileNet model the best model for the detection of packed malware. As a matter of fact, the DenseNet model obtains an accuracy equal to 0.949, 0.974, 0.973, and 0,974, while the DenseNet reaches the following accuracy, i.e., 0.947, 0.966, 0.968, and 0.968.

5 CONCLUSION AND FUTURE WORK

Current signature-based mechanisms used by free and commercial antimalware solutions rely on having a known signature of a malicious sample to detect and block its activity. As a result, malware authors have developed techniques that modify the syntax of the malware's code while preserving its underlying behavior or logic. One such method involves creating packed malware, where packers—software tools that compress and package an application along with a decompression stub—alter the binary code of the malware. This stub decompresses the packed code in memory and executes it, allowing a previously detected malware to evade detection in its packed form.

In this paper, we introduce a technique that utilizes convolutional neural networks (CNNs) to detect packed malware. Our approach involves static analysis, meaning it does not require the execution of the application to identify malicious samples. Starting with the binary code of an application, we transform it into an image that serves as input to a series of deep learning classifiers. These classifiers aim to determine whether the application under analysis is trusted or (packed) malicious.

From the experimental analysis, it emerges that the MobileNet and the DenseNet models show minimal fluctuation in accuracy and have high AUC, indicating their effectiveness in malware classification.

In future work, we plan to consider prediction explainability, with the aim of understanding which parts of the images related to malware are symptoms of the model prediction. Moreover, we will also consider the possibility of detecting ransomware with the proposed model.

ACKNOWLEDGMENT

This work has been partially supported by EU DUCA, EU CyberSecPro, SYNAPSE, PTR 22-24 P2.01 (Cybersecurity) and SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the EU - NextGenerationEU projects, by MUR - REASONING: foRmal mEthods for computAtional analySis for diagnOsis and progNosis in imagING - PRIN, e-DAI (Digital ecosystem for integrated analysis of heterogeneous health data related to high-impact diseases: innovative model of care and research), Health Operational Plan, FSC 2014-2020, PRIN-MUR-Ministry of Health, the National Plan for NRRP Complementary Investments D^3 4 Health: Digital Driven Diagnostics, prognostics and therapeutics for sustainable Health care, Progetto MolisCTe, Ministero delle Imprese e del Made in Italy, Italy, CUP: D33B22000060001, FORESEEN: FORmal mEthodS for attack dEtEction in autonomous driviNg systems CUP N.P2022WYAEW and ALOHA: a framework for monitoring the physical and psychological health status of the Worker through Object detection and federated machine learning, Call for Collaborative Research BRiC -2024, INAIL.

REFERENCES

Biondi, F., Enescu, M. A., Given-Wilson, T., Legay, A., Noureddine, L., and Verma, V. (2019). Effective, efficient, and robust packing detection and classification. *Computers & Security*, 85:436–451.

Ciaramella, G., Iadarola, G., Martinelli, F., Mercaldo, F., and Santone, A. (2024). Explainable ransomware detection with deep learning techniques. *Journal of Computer Virology and Hacking Techniques*, 20(2):317–330.

Devi, D. and Nandi, S. (2012). Detection of packed malware. In *Proceedings of the First International Conference on Security of Internet of Things*, pages 22–26.

He, H., Yang, H., Mercaldo, F., Santone, A., and Huang, P. (2024). Isolation forest-voting fusion-multioutput: A stroke risk classification method based on the multidimensional output of abnormal sample detection. *Computer Methods and Programs in Biomedicine*, page 108255.

Mercaldo, F. and Santone, A. (2020). Deep learning for image-based mobile malware detection. *Journal of Computer Virology and Hacking Techniques*, 16(2):157–171.

Qiang, W., Yang, L., and Jin, H. (2022). Efficient and robust malware detection based on control flow traces using deep neural networks. *Computers & Security*, 122:102871.

Rabadi, D. and Teo, S. G. (2020). Advanced windows methods on malware detection and classification. In *Proceedings of the 36th Annual Computer Security Applications Conference*, pages 54–68.