Classifying Packed Malware with Convolutional Neural Networks

A ScoreCAM and Occlusion Analysis of Necessary Features

Tristan Rietveldt

Supervisor(s): Tom Viering, Akash Amalan

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Tristan Rietveldt
Final project course: CSE3000 Research Project
Thesis committee: Tom Viering, Akash Amalan, Georgios Smaragdakis

Abstract

Accurate and scalable detection has become a challenge in cybersecurity due to an exponential increase in the volume of new malware and its increasing complexity. A promising solution could be using convolutional neural networks to classify malware binaries interpreted as grayscale images. However, malware is frequently packed, which makes classification more difficult. We studied the impact of these packers on classification and what underlying features the models rely on. We trained independent ResNet-18 classifiers and evaluated them using two explainable AI methods: ScoreCAM and occlusion. The experimental setup analysed a synthetic dataset of 19,735 samples subjected to eleven different packers. We have four main results: (1) models perform poorly on large files packed with UPX, (2) models generalise well across structure-preserving packers, but (3) fail on transformations that alter the entire structural layout. Furthermore, (4) models rely more on unaffected binary sections when the main code section is obfuscated. Thus, convolutional neural networks remain highly dependent on file structure. Future research should investigate the efficacy of different explainable AI methods and the effects of resizing malware images.

1 Introduction

Accurate and scalable detection has become a challenge in cybersecurity due to an exponential increase in the volume of new malware and its increasing complexity [1]. Both traditional static analysis, which examines a binary without running it, and dynamic analysis, which inspects its behaviour during execution, have drawbacks. Static analysis often struggles against obfuscation, whereas dynamic analysis remains slow and difficult to scale [2, 3, 4].

To solve this today, research has recognised automated machine learning approaches as a possible solution [2]. Within this domain, the interpretation of malware binaries as grayscale images has been a promising development. By mapping each byte to a pixel intensity, similar malware variants display recognisable, shared textures [5]. Researchers have successfully applied deep Convolutional Neural Networks (CNNs) to these image representations, eliminating the need for manual feature engineering and achieving near-perfect accuracies on unpacked datasets [3, 6, 7, 8, 9, 10, 11].

However, these high accuracies can be misleading due to simplistic and biased datasets. Also, this approach struggles in practice because malware is frequently packed (compressed, encrypted, and/or obfuscated). Packing drastically alters the binary's structural layout and destroys the visual patterns CNNs rely upon. Consequently, image-based CNNs might suffer from "packer dependency," where they overfit to simplistic packer-specific visual artefacts rather than learning genuine malicious semantics [12, 13]. While Explainable AI (xAI) techniques have been used to interpret models trained on unpacked malware, a considerable research gap remains regarding how packers affect what these CNNs learn and focus on.

To address this gap, we use an empirical approach with xAI methods to systematically evaluate how eleven different packers affect CNN representations. Specifically, we explore the following research question: "What is the impact of packing on CNN malware family classification, and what do xAI methods (ScoreCAM and occlusion) reveal about the underlying features the models rely on?"

To comprehensively answer this question, we define the following sub-questions:

1. How does packing degrade the per-family classification performance?
2. To what extent do models cross-generalise across different packers?
3. Which regions of the malware image are highlighted as highly activating by ScoreCAM for different packers?
4. To what extent do regions identified by ScoreCAM drive CNN predictions?

The rest of this paper is structured as follows. Section 2 provides background on malware image representations and reviews related work on cross-packer generalisation and explainable AI. Section 3 details our experimental methodology, including the synthetic dataset, packers, and CNN evaluation pipeline. Section 4 presents our results on classification performance, cross-packer generalisation, and the causal importance of binary sections. Section 5 discusses these findings in the context of previous research and addresses our study's limitations. Section 6 concludes the paper and outlines future work. Finally, Section 7 reflects on the ethics of this research.

2 Background and Related Work

This section establishes the context for our research. It first provides the necessary background on interpreting malware as images for classification (Section 2.1), before reviewing the existing literature on cross-packer generalisation and the use of explainable AI in this domain (Section 2.2).

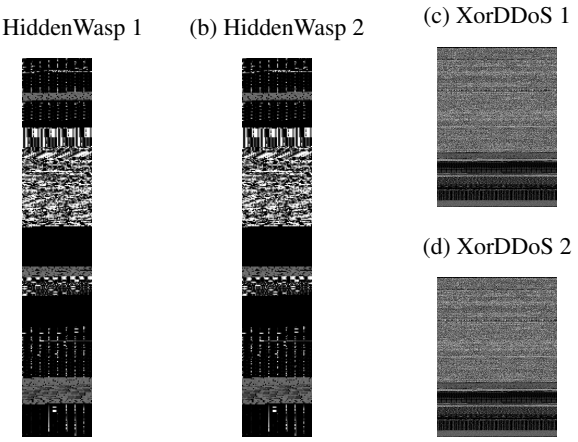
2.1 Background

As comprehensively reviewed by Gibert et al. [2], traditional machine learning approaches for malware detection and classification rely heavily on manual feature extraction. Features such as strings, n-grams, entropy, and control flow were extracted and subsequently classified using algorithms like k-Nearest Neighbours (k-NN) or Random Forests.

Malware Images

A major shift in static feature extraction occurred when Nataraj et al. [5] proposed interpreting malware binaries as uncompressed grayscale images. They argued that malware variants within the same family retain broad structural textures and asserted that this global textual representation provides inherent resilience against obfuscation techniques such as packing. This is visualised in Figure 1, which shows similarities between samples of two malware families.

Figure 1: Visual comparison of malware binaries converted to images. The first two images demonstrate texture similarity between two distinct samples of HiddenWasp, which are distinct from two similar samples of XorDDoS displayed on the right.



However, a major limitation of this early work was its continued reliance on manually extracted GIST texture features for k-NN classification.

To address the limitations of manual feature engineering, recent state-of-the-art research has increasingly adopted Convolutional Neural Networks (CNNs) to automatically learn spatial features directly from malware images [3]. These CNN-based methods report near-perfect accuracies on unpacked datasets [3, 6, 7]. Whether these results hold for packed malware remains debated, as discussed in Section 2.2.

In this paper, we build upon this malware-as-image paradigm to understand *how* and *why* CNN performance changes under different packing transformations through a combination of cross-packer evaluation and explainable AI methods.

2.2 Related Work

Cross-Packer Generalisation

Whether CNN classifiers generalise across different packers remains an open debate. On one side, studies such as Vasan et al. [7] and Alkhateeb et al. [14] support Nataraj's original assertion that can generalise successfully to unknown packers. They report that deep learning models maintain high accuracy (95–99%) when evaluated on packed datasets. However, recent empirical evaluations cast doubt on these claims. Yu et al. [15] and Gibert et al. [13] warn that the near-perfect accuracies reported in earlier studies are often misleading, stemming from overfitting to simplistic, biased datasets rather than learning malicious features. Kalapala and Zhang [12] demonstrated that CNN classifiers suffer from strong "packer dependency": instead of identifying malicious characteristics, models tend to learn packer-specific visual artefacts, leading to near-random performance when confronted with unseen packers. Furthermore, Gibert et al. [13] found that greyscale image-based detectors exhibit the worst performance against packing compared to other static detection methods. These robust studies suggest that the optimistic results reported by earlier work may not hold under rigorous evaluation.

Neither side has systematically evaluated *which types* of packing transformations affect cross-packer generalisation and why. In this paper, we contribute to this discussion by training independent CNN models for eleven distinct packers spanning compression, encryption, and obfuscation, and cross-evaluating them to assess how each transformation type affects generalisation.

Explainable AI

Explainable AI (xAI) techniques have been adopted to interpret these image-based models. However, there is a contradiction in their results. Sarı and Acı [9] and Kinkead et al. [8] used SHAP and LIME to show that CNNs trained on unpacked malware images learn semantically meaningful structural patterns. Specifically, Alshomrani et al. [10] and Ganapathiyappan et al. [11] showed that early CNN layers capture localised, fine-grained visual patterns (e.g., specific opcode clusters or boundary transitions), while deeper layers cluster these into abstract regions. On the other hand, as previously discussed, Kalapala and Zhang [12] and Gibert et al. [13] argue that models overfit to superficial visual artefacts rather than genuine malicious semantics. This contradiction likely stems from the introduction of packing: while CNNs may successfully learn meaningful structural semantics on unpacked baselines, packing drastically alters these structures, forcing the model to rely on packer-specific artefacts instead. However, xAI insights are largely limited to unpacked malware. While Mahmud Sujon et al. [16] explored interpretability for packed files, their work was limited to entropy-based feature engineering with traditional machine learning models, such as Random Forests and Decision Trees, instead of CNNs. A notable exception is Brosolo et al. [17], who investigated CNN robustness against UPX packing. Using HiResCAM, SHAP, and Occlusion Maps, they observed that the CNN maintains accuracy by shifting its focus from the obfuscated topmost sections to the unobfuscated overlay.

Our research expands upon the work of Brosolo et al. [17], as well as that of Kalapala and Zhang [12] and Gibert et al. [13]. Where prior xAI work focused exclusively on unpacked malware or a single packer (UPX), we apply ScoreCAM and occlusion analysis across eleven packing configurations to systematically investigate how different packing transformations affect the regions CNNs rely upon for classification.

3 Methodology

This section outlines the methodology used to evaluate how packers affect malware classification with CNNs and how explainable AI (XAI) methods can interpret these models' predictions. We first provide a high-level overview of our pipeline, the synthetic dataset, and the packers applied. We then describe the conversion of binaries into image representations, the CNN architecture used for classification, and the interpretation methods (ScoreCAM and Occlusion). Finally, in Section 3.8, we detail our experimental design, including the specific software environment, preprocessing, and training details.

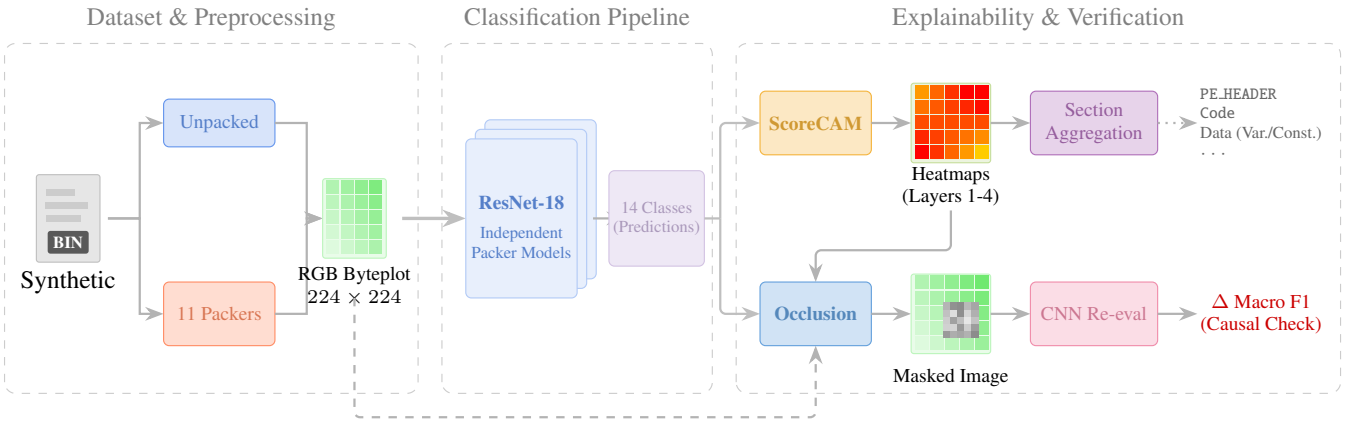


Figure 2: **Experimental Methodology.** *Left:* Synthetic executables (packed and unpacked) are converted into byteplots and resized to 224 × 224. *Middle:* Independent ResNet-18 classifiers are trained to classify into 14 classes. *Right:* ScoreCAM heatmaps extract spatial attention, mapped to logical binary sections, while Occlusion validates causality by assessing performance drop under targeted masking. Adapted from the TikZ source code of Amalan et al. [18]. Licensed under CC BY 4.0.

3.1 Pipeline

Figure 2 shows a graphical representation of the pipeline. We used a pre-constructed dataset of synthetic malware samples, their packed variants, and corresponding image representations. Using the provided visual representations, independent CNN classifiers (ResNet-18) were trained: one baseline model on unpacked data and separate models for each packer. Once trained, the models were interpreted using ScoreCAM to generate activation maps highlighting the most important image regions. Finally, an occlusion verification step was performed to empirically validate whether these highlighted regions causally drive the model's predictions.

3.2 Dataset and Preprocessing

The dataset used is synthetically generated and neutered, primarily to avoid the risk of unintended infection by real malware. The samples in the dataset still represent the original malware; however, their harmful code has been removed. The dataset was provided by Akash Amalan.

The synthetic dataset consists of 19,735 samples, out of which 10,010 are unpacked, and 9,725 are packed. These samples are distributed across 14 classes across malware and benign binaries, of which 8 in PE format and 6 in ELF format. Table 1 shows the specific families, the number of unpacked samples per family, and their median size with standard deviation. Table 7 in the Appendix (section A) shows the dataset distribution in detail.

Note that all debug sections were stripped from the binaries during dataset preparation. Initial exploratory experiments revealed that the CNN classifiers would disproportionately focus their attention on these debug sections rather than on the structural or malicious features of the executables. Real malware often lacks these debug sections.

3.3 Packers

In our experiments, the synthetic binaries were packed using eleven packers in total: `Mangle`, `MPress`, three `UPX` packers (default, brute, and LZMA), four `xor` packers (static/rolling application combined with a fixed/random key), `zip`, and `zlib`. Alongside complex real-world packers like `UPX`, `MPRESS`, and `Mangle`, the dataset contains simplified packers, specifically *xor* encryption and *zip/zlib* compression. These custom packers are less complex than commercial packers because they only have one effect (either encryption or compression). This makes it easier to draw clear conclusions about how these packing types affect the model.

Not every packer was universally applicable. The number of samples per family and packer was divided as evenly as possible; however, some packers could not be applied to certain samples due to format (PE/ELF) and other requirements. Importantly, `Mangle` and `MPress` could only be run on PE executables. The distribution of samples is available in Table 7 in the Appendix (section A).

Table 1: Dataset distribution per family, showing the number of (unpacked) samples and the median binary size with standard deviation.

Family	Samples	Median Size
PE Format		
Adware	715	22 KB ± 1 KB
BenignPE	715	20 KB ± 2 KB
Botnet	715	58 KB ± 1 KB
Ransomware	715	20 KB ± 1 KB
Rootkit	715	14 KB ± 1 KB
Spyware	715	22 KB ± 1 KB
Trojan	715	19 KB ± 1 KB
Worm	715	20 KB ± 1 KB
ELF Format		
BenignELF	715	38 KB ± 2 KB
Gafgyt	715	81 KB ± 22 KB
HiddenWasp	715	14 KB ± 2 KB
Mirai	715	81 KB ± 22 KB
Tsunami	715	66 KB ± 2 KB
XorDDoS	715	6 KB ± 26 KB

3.4 Image Representation

Malware images were generated using the approach recommended in Nataraj et al. [5]: Images were given a fixed width based on their file size (see Table 2).

Table 2: Image widths for images generated from malware binaries. Adapted from Nataraj et al. [5].

Filesize	Image Width
0-10 kB	32 px
10-30 kB	64 px
30-60 kB	128 px
60-100 kB	256 px
100-200 kB	384 px
200-500 kB	512 px
500-1000 kB	768 px
1000+ kB	1024 px

Bytes were visualised in the grayscale image, with each byte (0-255) mapping to a pixel (brightness 0-255), preserving the ordering. If the number of bytes in the input binary did not evenly divide by the image width, then the binary was padded with zeroes to fill the final row.

3.5 Classification Model

We used ResNet-18 as our Convolutional Neural Network (CNN) architecture [19] without any pre-trained weights. ResNet-18 is a good baseline for distinguishing malware families: Alshomrani et al. [10] have achieved a baseline F1 score of 0.976 on the Microsoft Malware dataset, and [4] have achieved 99.64% precision on the Malimg and MaleVis datasets using extensive feature engineering. We used a model with only 18 layers to avoid the type of overfitting described by Yu et al. [15]: the model learns signatures rather than malware features. Following their recommendation, we also used dropout to prevent overfitting. Finally, the output layer has 14 nodes, one for each of the 12 malware families in the dataset, and two for the benign families. In case there were fewer classes (for example, for PE only packers), we kept the 14 output nodes to keep the training and evaluation pipeline simple.

We trained independent, identical ResNet-18 models for each packer, as well as one baseline model trained on unpacked samples. This experimental design isolates the impact of different packing algorithms. By training separate models, the CNNs were specialised to the structural artefacts produced by a single packer (or lack thereof). We then cross-evaluated these models to test for generalisation across different packers and quantify the extent to which classification performance degrades under each specific packer.

3.6 Visual Explanations with ScoreCAM

This subsection details our ScoreCAM-based interpretation of the trained CNNs. It covers our motivation for selecting it over GradCAM, the generation of attention heatmaps, and how we map attention back to the binary's sections for structural analysis.

Motivation

As highlighted by recent findings in malware classification [20, 21], gradient-based methods can suffer from issues like unstable or shattered gradients and false confidence in deep networks, highlighting artefacts rather than semantically meaningful regions in the malware image. Instead, we used ScoreCAM [22]. Unlike Grad-CAM, ScoreCAM is a gradient-free approach that derives weights for activation maps through forward-pass ablation: measuring the change in model confidence when the input image is masked by the activation map itself. By relying on raw class scores rather than gradients to weight the activation maps, ScoreCAM reduces this dependency and provides more robust visual explanations of the model's decision-making process.

Heatmap Generation

We ran ScoreCAM on each test sample in the dataset (for each of the five folds) to generate heatmaps of CNN attention. In particular, attention maps were extracted from the last layer of each convolution block. Standard practice typically focuses only on the final convolutional layer (layer 4). However, this layer inherently only captures global textures due to its spatial coarseness (7×7). We hypothesised that earlier layers capture fine-grained, localised structural patterns, such as PE headers and specific code sections, before clustering them into abstract textures. Thus, we investigated earlier layers as well.

Section-Level Aggregation

Since the heatmaps generated were 224×224 (ResNet-18 input size), they were resized to the malware byteplot dimensions using bilinear interpolation. After resizing, the attention values were mapped to byteplot pixels. These attention values were mapped back to the binary and aggregated for each file section (e.g., `.text`, `.rdata`, `.bss`) parsed from the PE/ELF headers. Mean, median, minimum, maximum, and standard deviation were saved per section, and grouped by layer, packer and family. For PE binaries, the PE header was treated as a section because it contains structural information about the executable, such as DLL imports and exports.

Since the sections present in a binary are highly file-dependent and lack consistency (especially for ELF), we could not aggregate attention based on exact section names alone. Therefore, we categorised sections into five broad categories: (1) *PE_HEADER* (PE only), (2) *Code*, (3) *Data (Var. / Const.)*, (4) *Linking & Memory*, and (5) *Runtime & Exceptions*. We averaged the attention scores across these logical groups, respecting section lengths. After all sections had been aggregated, the mean and standard deviation of the aggregated values were calculated. The section mappings used for this aggregation are detailed in Table 10 in the Appendix (section A).

3.7 Occlusion

While ScoreCAM highlights the regions the model pays attention to, it does not guarantee that those regions are necessary for correct classification. To verify the causal impact of these highlighted features, we used occlusion: we masked out regions of the malware image that were identified as highly activating by ScoreCAM. We then compared the

classification performance on these masked images against a control set in which random sections of the same size were occluded. Masked regions were filled with random noise to disrupt the structural integrity of that section. For binaries packed with *zip* and *zlib*, we used a targeted occlusion size that corresponded only to the unpadded compressed data, ignoring the zero-padding. A drop in the macro F1-score when the ScoreCAM-identified regions are masked, relative to the random control masks, indicates that the CNN's predictions are driven by these specific regions.

3.8 Experimental Design

Reproducibility

A fixed random seed was used throughout all experiments. The software packages and libraries used for the experiments, model instantiation, ScoreCAM generation, and data processing are detailed in Table 12 in Appendix A. The Jupyter notebooks and Docker environment used for the experiments are available at https://github.com/tristan-tr/malware-thesis-packed-features.

Data Preparation & Training Details

Before training, binary images were converted to RGB images with dimensions 224×224 because ResNet-18 requires this. Images were resized using bilinear interpolation and converted to RGB by duplicating the grayscale channel. After converting these images to tensors, they were normalised using the default ImageNet mean $[0.485, 0.456, 0.406]$ and standard deviation $[0.229, 0.224, 0.225]$, as done in earlier work by Alshomrani et al. [10]. During training, random horizontal flips were performed to reduce artefacts of fragmentation due to image row boundaries. This is discussed as a limitation in section 5.

All experiments were performed using stratified 5-fold cross-validation using a 60-20-20 train-validation-test split. This split was done using SHA hashes of the unpacked binaries to avoid test leakage for cross-packer evaluation. The network was trained to minimise the `CrossEntropyLoss`. To mitigate overfitting and preserve the most generalisable state of the network, we used checkpointing, in which performance at each epoch was evaluated on the validation set. Only the model weights with the lowest validation loss over the epochs were saved. All hyperparameters used in our experiments are detailed in Table 11 in Appendix A.

Since the dataset was not entirely balanced, we used a macro-averaged F1 score to evaluate classification performance across the different packers. This balances precision and recall, and accounts for any class imbalances within folds or packer subsets.

4 Results

This section presents the outcomes of our experiments. First, we detail classification performance across packers. Then, we clarify the extent to which models generalise across different packers. Afterwards, we identify important binary sections via ScoreCAM. Finally, we validate the causal importance of these sections with occlusion.

Table 3: Macro F1 scores (5-fold cross validation) of CNNs trained solely on samples packed with the specified packer

Packer	Macro F1-score
None	1.000 ± 0.000
Mangle	1.000 ± 0.000
MPRESS	0.997 ± 0.006
UPX	$\mathbf{0.947 \pm 0.011}$
UPX (brute)	$\mathbf{0.863 \pm 0.023}$
UPX (lzma)	$\mathbf{0.796 \pm 0.014}$
XOR (fixed)	0.995 ± 0.006
XOR (random)	0.992 ± 0.009
XOR (rolling fixed)	0.994 ± 0.007
XOR (rolling random)	0.995 ± 0.006
zip	0.998 ± 0.003
zlib	1.000 ± 0.000

Table 4: Per-family macro F1 scores (5-fold cross validation) for CNNs trained on UPX packers. *Low sample size (n = 17).
The confusion matrix for Gafgyt, Mirai, Tsunami and XorDDoS is available in Table 13 in Appendix A.

	UPX	UPX (brute)	UPX (lzma)
Gafgyt	0.75 ± 0.06	0.39 ± 0.10	0.19 ± 0.21
Mirai	0.75 ± 0.09	0.37 ± 0.21	0.50 ± 0.13
Tsunami	0.88 ± 0.07	0.66 ± 0.06	0.65 ± 0.08
XorDDoS	0.92 ± 0.09	0.80 ± 0.10	$*0.00 \pm 0.00$
Other	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00

4.1 Classification Scores Across Packers

We evaluated how different packers degrade the per-family classification performance compared to the unpacked baseline. Table 3 shows the Macro F1-scores of the CNN classifier for each packing method.

The baseline model, evaluated on unpacked samples (*none*), achieves a perfect Macro F1-score of 1.000 ± 0.000. Most packers, including encryption (*xor* variants), compression (*zip*, *zlib*, *mpress*), and other obfuscators (*mangle*), largely do not degrade the model's accuracy, with Macro F1-scores remaining above $\approx$0.992. However, a large performance drop is observed for samples packed with UPX and its variants. Standard *upx* reduces the Macro F1-score to 0.947 ± 0.011, while *upx brute* and *upx lzma* further decrease it to 0.863 ± 0.023 and 0.796 ± 0.014, respectively. This indicates that UPX packing partly obscures the visual patterns the CNN relies on for classification, particularly at higher compression levels such as LZMA.

Surprisingly, zip/zlib and xor variants did not degrade classification performance. Upon further investigation, we found that the zip/zlib and xor packers used in our dataset only compress/encrypt the code section, leaving other sections intact. Thus, the models retain enough unaffected structural information to classify the binaries accurately.

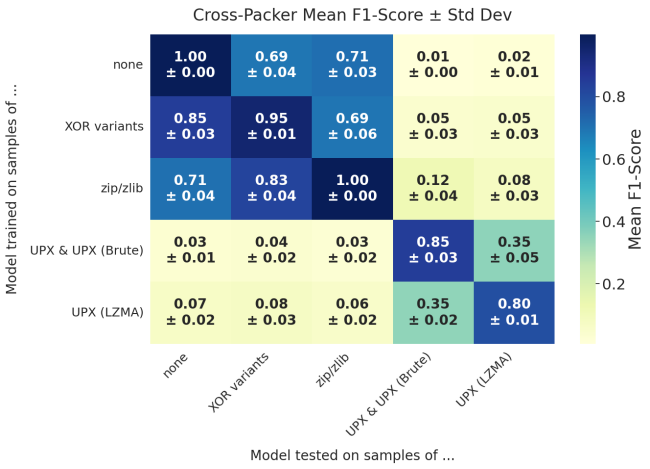
To understand why UPX variants suffer such a severe decline, we break down the performance by malware family in Table 4. The degradation is entirely concentrated within the ELF Linux botnet families: Gafgyt, Mirai, Tsunami, and

XorDDoS. For the "Other" families (which include all PE formats and benign samples), the classifier maintains a perfect score. However, under *UPX lzma*, the F1-score for Gafgyt and XorDDoS drops significantly. This decline indicates that the CNN struggles to distinguish these specific families when packed with UPX. This may result from the substantial file sizes associated with these families or from their shared botnet characteristics.

4.2 Cross-Packer Generalisation

To better understand the features learned by the CNNs, we evaluated the generalisation of models trained on one packer when tested on samples packed with a different packer. Here, F1-scores are reported over only the shared families between packers. The results of these cross-packer evaluations are presented as a heatmap in Figure 3, where packers are grouped for brevity. Furthermore, *mangle* and *mpress* have been omitted from this short figure because they demonstrated no cross-generalisation with other packers. The full results are available in Table 8 in the Appendix (section A).

Figure 3: F1-scores of models trained on samples packed with the left packer and evaluated on samples packed with the bottom packer, with packers grouped for brevity; the full results are available in Table 8 in the Appendix (section A)



The results indicate that models do not broadly generalise based on packing transformation family (e.g., compression, encryption, or obfuscation).

First, models trained on the unpacked baseline (*none*) maintain strong performance on samples packed with *xor* variants (≈0.69 F1-score) and the combined *zip/zlib* group (≈0.68 F1-score). Conversely, models trained on *xor* and *zip/zlib* also perform well when tested on unpacked binaries, achieving F1-scores between ≈0.85 and ≈0.71, respectively.

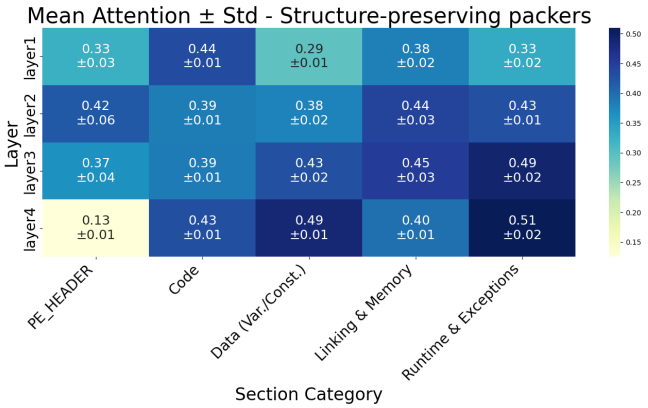
Interestingly, *xor* and *zip/zlib* models also generalise effectively to each other. For instance, models trained on *zip/zlib* achieve an average F1-score of ≈0.83 on *xor* variants, and models trained on *xor* variants achieve an average F1-score of ≈0.69 on the *zip/zlib* group. This cross-generalisation can be explained by our earlier finding that *zip/zlib* and *xor* only compress/encrypt the code section, forming the basis for our later occlusion analysis. In contrast, packers that alter the structural layout of the file, such as *mangle*, *mpress*, and the *upx* variants, break this generalisation. Models trained on unpacked binaries or *xor* and *zip/zlib* packers fail to classify samples packed with these methods (F1 $\leq$ 0.15).

However, there is some generalisation among the *upx* variants. *UPX* and *UPX (Brute)* models generalise well to each other with an F1-score of ≈0.85. Models trained on *UPX* and *UPX (Brute)* generalise to a lesser extent to *UPX (LZMA)* (F1-score of ≈0.35). Based on these findings, *xor* variants, *zip/zlib*, and *none* can generalise to each other, *upx* variants can generalise to each other, and *Mangle* and *MPRESS* models do not generalise.

4.3 Section Attention

To test whether the network's attention shifts from fine-grained structural patterns to abstract textures across layers, we ran ScoreCAM on all samples in our dataset, for all packers and all layers. Figure 4 shows the resulting heatmap of CNN attention across different layers for samples packed with structure-preserving packers (*xor* variants, *zip/zlib*).

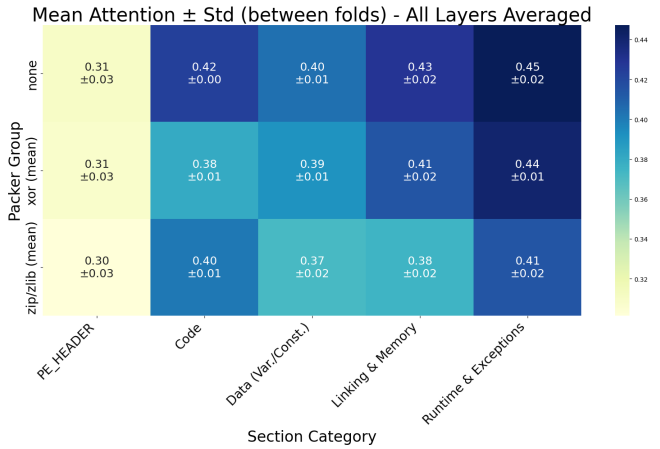
Figure 4: Heatmap of CNN attention for samples packed with structure-preserving packers



Analysis of samples packed with structure-preserving packers (*xor* variants, *zip/zlib*), shown in Figure 4, showed that different CNN layers pay attention to different sections of the binary. Notable patterns are that attention steadily rises over layers 1-4 in the *Data (Var./Const.)* and *Runtime & Exceptions* sections. *Code* and *Linking & Memory* attention stays roughly the same across the layers. *PE_HEADER* attention drops heavily in layer 4. However, it should be noted that this section is small compared to the other sections and is therefore very sensitive to interpolation. These results show that the CNN uses earlier layers to capture specific fine-grained information, such as discriminative headers.

Figure 5 shows a heatmap of CNN attention for unpacked, xor, and zip/zlib samples. As shown in the figure, CNN attention to the code section does not decrease by a large margin when this section is encrypted/compressed. This might suggest that the CNN does not maintain its accuracy by focusing on unobfuscated sections. However, this is complicated by occlusion analysis in Section 4.4.

Figure 5: Heatmap of CNN attention for unpacked samples and samples packed with structure-preserving packers (xor and zip/zlib)



4.4 Causal Importance via Occlusion

To verify whether the regions highlighted by ScoreCAM are necessary for classification, we performed an occlusion analysis. Before detailing the results, we summarise our three key findings:

- **Minimal reliance on the Code section:** Across most packers, the CNN relies significantly less on the Code section than other structural sections for classification.
- **Increased reliance on other sections during obfuscation:** When the Code section is obfuscated, the model's reliance on unobfuscated structural sections (like the Data section) increases to compensate.
- **Learning of fixed encryption keys:** The CNN is capable of extracting meaningful classification features from encrypted Code sections when a fixed encryption key is used, but fails to do so when keys are randomised.

In our occlusion analysis, we masked specific structural sections and compared the resulting macro F1-scores against those obtained when masking a randomly selected region of the same size. A larger performance degradation from section occlusion compared to random occlusion indicates that the model relies on the target section. The results for the unpacked baseline and structure-preserving packers are presented in Table 5. The unaggregated target and control F1-scores are available in Table 9 in the Appendix (section A).

Minimal reliance on the Code section

The models rely less on the *Code* section than on other sections. Across all packers, masking the Code section resulted in a smaller performance drop than masking control regions. Because the Code section is generally the largest segment of the binary, a random control mask of equivalent size is highly likely to overlap with smaller, load-bearing regions (such as the Data section). Under the targeted occlusion of the unpadded compressed data for *zip/zlib*, the F1-score remained at ≈0.9991, as shown in the Appendix (section A). This indicates that while ScoreCAM highlighted the obfuscated code section, the model does not use this compressed data for classification.

Table 5: F1-scores of models evaluated on samples with sections occluded and random control regions occluded.

Packer Group Occluded	Unpacked	XOR (mean)	zip/zlib (mean)
PE_HEADER	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Random	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
Code	**0.67 ± 0.04**	**0.95 ± 0.01**	**1.00 ± 0.00**
Random	**0.39 ± 0.03**	**0.73 ± 0.02**	**0.87 ± 0.02**
Data	**0.96 ± 0.02**	**0.82 ± 0.03**	**0.86 ± 0.05**
Random	**0.98 ± 0.01**	**0.97 ± 0.01**	**0.92 ± 0.01**
Linking & Memory	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
Random	0.99 ± 0.00	0.96 ± 0.01	0.93 ± 0.04
Runtime & Exceptions	1.00 ± 0.00	0.98 ± 0.00	0.99 ± 0.01
Random	1.00 ± 0.00	0.99 ± 0.00	0.97 ± 0.01

Table 6: Occlusion F1-Scores for Individual XOR Variants

Occluded Packer	Code	Random (Code size)	Data	Random (Data size)
Unpacked	0.67 ± 0.04	0.39 ± 0.03	0.96 ± 0.02	0.98 ± 0.01
XOR (fixed)	0.95 ± 0.02	0.76 ± 0.03	0.89 ± 0.03	0.99 ± 0.01
XOR (rolling fixed)	0.88 ± 0.02	0.73 ± 0.04	0.90 ± 0.06	0.99 ± 0.01
XOR (random)	0.98 ± 0.02	0.72 ± 0.02	0.75 ± 0.08	0.95 ± 0.02
XOR (rolling random)	0.98 ± 0.01	0.71 ± 0.03	0.75 ± 0.04	0.94 ± 0.01

Increased reliance on other sections

Comparing the occlusion impact across different packers shows a shift in feature reliance. For the unpacked baseline, the model is relatively robust to occlusion of the Data section, maintaining an F1-score of ≈0.96. However, when the Code section is obfuscated through *xor* or *zip/zlib*, the performance drop upon occluding the Data section is larger. This suggests an increased reliance on unobfuscated sections when the code section is obfuscated.

Learning of fixed encryption keys

Furthermore, analysing the individual *xor* variants provides empirical support that the CNN learns fixed encryption keys. Table 6 shows the macro F1-scores when different sections are occluded for each of the *xor* variants. When the *Code* section is occluded (target occlusion), the F1-scores for the fixed-key variants (*xor_fixed* and *xor_rolling_fixed*) drop to ≈0.95 and ≈0.88, respectively. In contrast, the random-key variants (*xor_random* and *xor_rolling_random*) remain largely unaffected (both ≈0.98). This shift can also be seen in the F1-scores for *Data* section occlusion. Models trained on random keys suffer a large performance drop (to ≈0.75) when it is occluded, compared to a smaller drop (≈0.90) for fixed-key models. These findings suggest that the CNN learns fixed encryption keys: The models extract information from the encrypted code section only when the training and test samples are encrypted with a fixed key, as opposed to a random key.

5 Discussion

Key Findings

The primary objective of this study was to evaluate how packers impact the classification performance of Convolutional Neural Networks (CNNs) on malware image representations, and to use Explainable AI (xAI) methods to uncover the underlying features driving these models' predictions. Our findings demonstrate that while CNNs can achieve high classification accuracy when trained on specifically packed datasets, they struggle with UPX variants. This performance degradation was heavily concentrated within ELF Linux botnet families, which are large binaries and similar to each other. Furthermore, our cross-packer evaluation highlighted that structural preservation is necessary for cross-packer generalisation; models can successfully generalise across different packers (such as between the unpacked baseline, *xor*, and *zip/zlib*) provided the overarching binary structure remains similar. Finally, through occlusion analysis, we discovered that CNNs maintain high accuracy on these structurally similar, partially obfuscated binaries by shifting their causal reliance from the obfuscated code sections to unaffected structural areas, like the Data section.

Relation to Prior Work

These results bring clarity to the conflicting claims in existing literature regarding CNN resilience to packing. Our findings challenge studies by Vasan et al. [7] and Alkhateeb et al. [14], which assert high model resilience to unseen packers. Instead, our research aligns with the warnings of Kalapala and Zhang [12] and Gibert et al. [13], demonstrating that image-based CNNs exhibit severe "packer dependency" when faced with transformations that completely alter binary structure (e.g., UPX, MPRESS, Mangle). Our models trained on specific packers failed to generalise to dissimilar structural layouts. Thus, models trained on a single packer are vulnerable to out-of-distribution samples (i.e., packers that result in a different binary structure). Additionally, we extend the interpretability work of Brosolo et al. [17] by systematically evaluating multiple packing types. Where they observed a focus shift for UPX, we demonstrate that for structure-preserving transformations like *xor* and *zip/zlib*, the CNN maintains high accuracy by explicitly shifting causal reliance from the obfuscated *.text* sections to the unaffected Data section.

Unexpected Results

An unexpected finding was that *zip/zlib* compression and *xor* encryption did not degrade overall classification performance because the packers only targeted the code section. However, it was surprising that ScoreCAM heatmaps still indicated high attention to these obfuscated code sections. Our subsequent occlusion analysis revealed that despite this high attention, the CNN did not causally rely on the compressed code or on *xor* variants with random keys for its predictions, instead depending heavily on the unaffected Data sections. This discrepancy confirms that high attention (as indicated by ScoreCAM) does not necessarily equate to causality. The CNN may simply highlight high-entropy compressed regions as a distinct visual texture, even if that texture does not drive the final classification.

Limitations

Despite these insights, there are several limitations to our study. First, the use of a synthetically generated dataset likely does not perfectly represent the diversity and complexity of real malware, meaning the samples might be too structurally homogeneous. Second, our image generation and preprocessing pipeline introduced potential artefacts. Resizing malware images of varying dimensions to a fixed 224×224 tensor, and subsequently scaling the resulting attention heatmaps back up, relies heavily on interpolation. This distortion is larger for bigger files, such as the ELF botnets that failed under UPX, which are squashed on both axes. This interpolation makes attention values for small sections, like the `PE_HEADER`, unreliable. Then, it has to be noted that *MPRESS* is a PE-only packer, so its results are skewed; UPX also scores well on the smaller PE files. Third, during training, random horizontal flips were performed in an attempt to reduce image row boundary fragmentation. However, using random horizontal flips was a suboptimal transformation; flipping the byte order destroys the sequential integrity of the binary data, and applying small offset shifts would have been a more sound augmentation strategy. Unfortunately, due to time constraints, we could not redo the experiments without horizontal flips. Finally, our occlusion methodology using random noise can induce bias. Models trained on *zip/zlib* are inherently accustomed to high-entropy data in the code section and might be more robust to noise-based occlusion, whereas other occlusion methods (like setting bytes to zero) would introduce different structural biases.

6 Conclusions and Future Work

In conclusion, we demonstrate that image-based CNN classifiers can achieve exceptional accuracy on unpacked malware, but their robustness against packed binaries is fragile and highly dependent on structural preservation. Transformations that alter the entire file layout break cross-packer generalisation and drastically degrade performance. Our combination of ScoreCAM and occlusion analysis reveals that CNNs circumvent localised obfuscation by shifting their causal reliance onto unaffected structural elements, and calls into question the efficacy of ScoreCAM for explaining deep learning models in cybersecurity.

Future work could investigate which binary regions are important for classification in real-world packers, such as Themida. Another potential next step is to investigate the discrepancy between ScoreCAM attention and occlusion results; why does the CNN pay attention to compressed code sections if they are not necessary? This also raises questions on the efficacy of ScoreCAM for the interpretation of malware classification models, and invites investigation into the efficacy of different xAI methods. Furthermore, future studies should test the effects of resizing on classification.

7 Responsible Research

This research is guided by the TU Delft Code of Conduct [23] and its core values of Diversity, Integrity, Respect, Engagement, Courage, and Trust (DIRECT). The following subsections outline how these values apply to our work. We first

discuss replicability and reproducibility, which relate to *Trust* and *Integrity*. We then address potential dataset bias and general research *Integrity*. Next, we cover the broader impact of this research, which connects to *Engagement* and *Courage*. Finally, we document our use of generative AI, which also relates to academic *Integrity*.

7.1 Replicability & Reproducibility

This research did not involve data related to human subjects. However, there are still considerable risks for sharing the dataset. Malware binaries can be misused, and even malware images can be converted back to the binaries themselves. Given the risk of harm, we decided not to upload the malware dataset. If one wants to reproduce this research, malware image datasets are available for researchers to apply to.[1]

For reproducibility, the code used in this research is available at https://github.com/tristan-tr/malware-thesis-packed-features. All software used in this research is available free of charge, and explicit version numbers are provided in this paper to ensure interoperability. An extensive methodology overview is provided in this paper, in case the original code is no longer available, or a researcher wants to independently reproduce this research.

7.2 Bias

Research outcomes can be influenced by various forms of bias, such as unintentional p-hacking or biases present in the dataset. To reduce research bias, we followed the scientific method and formulated hypotheses prior to each experiment. Because the dataset was supplied by the research team, we had no control over its composition. As it is synthetically generated, the dataset is probably biased; for instance, samples may be more alike than real-world malware typically is. Nonetheless, we explicitly address and acknowledge this potential dataset bias in the discussion section (Section 5).

7.3 Research Integrity

All sources used in all parts of this research have been cited, including software used. To comply with licensing agreements, we adapted a figure from Nataraj et al. [5] rather than copying it. This has been properly cited. We made all other figures ourselves. The raw results from the experiments are reported in the Appendix (Section A). The results were evaluated using 5-fold cross-validation to ensure their validity. All results are reported with standard deviations over the 5 folds.

7.4 Broader Research Impact

Cybersecurity research, in general, carries considerable risk of misuse. Any findings can be used by adversaries to bypass the specific automated detection described here. For example, adversaries could pad their binaries with data that fools image-based classifiers. To that end, it is important to note that machine learning for malware detection/classification should not be fully trusted. However, the main problem nowadays is the volume of new malware. Having robust automated classifiers will likely help flag at least some malware, which can alleviate the load on experts. We recommend not relying too heavily on machine-learning-based classifiers. They can be easily evaded and should only be used as a tool. Instead, we should still aim to rely on trusted and proven methods, such as signature detection.

[1] Such as VirusShare (https://virusshare.com/)

7.5 Generative AI Usage

Generative AI was used in this project with six main goals: (1) to rephrase text in this paper, (2) to suggest outlines and section structures (3) as formatting assistance for LaTeX, (4) to create boilerplate code, (5) to improve pipeline performance, and (6) to generate scripts for data visualisation. This was done using `Grammarly` and `Gemini 3.1 Pro`. Example prompts for each goal are:

1. **Goal 1: Rephrasing**: "Rephrase this paragraph to use simpler language. This should be understandable for anyone with a Computer Science bachelor. I have attached my full research paper for extra context. See the paragraph here: $\cdots$"
2. **Goal 2: Text structuring**: "How should I structure my introduction for a scientific paper? Specifically, how should I transition from the introduction to the methodology?"
3. **Goal 3: Formatting assistance**: "This table is too large for a two-column research paper. Make it fit in a single column. See the table here: $\cdots$"
4. **Goal 4: Boilerplate code**: "See my other notebooks. Create a setup code block for an occlusion pipeline, similarly to my other notebooks."
5. **Goal 5: Improving performance**: "See the pipeline in train_models.ipynb; This only uses 20% of my GPU. Make it use all of my GPU so that it runs quicker."
6. **Goal 6: Data visualisation**: "Create a heatmap of the table in the last cell of scorecam_differences_across_packers.ipynb; This should show the data formatted to 2 decimal places and include the standard deviation. Make sure all text is legible for a report (so use large font sizes)."

All produced text and code by generative AI was checked afterwards for any inaccuracies or hallucinations to ensure academic integrity.

References

[1] G. Varshney, S. Varshney, A. Suman, K. Chouhan, and P. Suman, "Machine learning based malware detection system," in *2023 3rd International Conference on Advancement in Electronics Communication Engineering (AECE)*, Nov. 2023, pp. 559–563. DOI: 10.1109/AECE59614.2023.10428565. [Online]. Available: https://ieeexplore.ieee.org/document/10428565.

[2] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," en, *Journal of Network and Computer Applications*, vol. 153, p. 102 526, Mar. 2020, ISSN: 10848045. DOI: 10.1016/j.jnca.2019.102526.

[3] A. Bensaoud, J. Kalita, and M. Bensaoud, "A survey of malware detection using deep learning," *Machine Learning with Applications*, vol. 16, p. 100 546, 2024, ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2024.100546.

[4] J. Wilkie, H. Hindy, I. Andonovic, C. Tachtatzis, and R. Atkinson, "Signal-based malware classification using 1d cnns," en, *Cybersecurity*, vol. 9, no. 1, p. 36, Mar. 2026, ISSN: 2523-3246. DOI: 10.1186/s42400-025-00454-6.

[5] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," en, in *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, Pittsburgh Pennsylvania USA: ACM, 2011, pp. 1–7, ISBN: 978-1-4503-0679-9. DOI: 10.1145/2016904.2016908. [Online]. Available: https://dl.acm.org/doi/10.1145/2016904.2016908.

[6] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," en, *Computers Security*, vol. 77, pp. 871–885, Aug. 2018, ISSN: 01674048. DOI: 10.1016/j.cose.2018.04.005.

[7] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of cnn architectures (imcec)," en, *Computers Security*, vol. 92, p. 101 748, May 2020, ISSN: 01674048. DOI: 10.1016/j.cose.2020.101748.

[8] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, "Towards explainable cnns for android malware detection," in *Procedia Computer Science*, ser. The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops, vol. 184, Jan. 2021, pp. 959–965. DOI: 10.1016/j.procs.2021.03.118.

[9] N. V. Sarı and M. Acı, "A hybrid cnn-gru model with xai-driven interpretability using lime and shap for static analysis in malware detection," en, *PeerJ Computer Science*, vol. 11, e3258, Oct. 2025, ISSN: 2376-5992. DOI: 10.7717/peerj-cs.3258.

[10] M. Alshomrani, A. Albeshri, A. A. Alsulami, and B. Alturki, "An explainable hybrid cnn–transformer architecture for visual malware classification," en, *Sensors*, vol. 25, no. 15, p. 4581, Jan. 2025, ISSN: 1424-8220. DOI: 10.3390/s25154581.

[11] K. Ganapathiyappan, H. G. Mohamed, A. Yadav, G. A. Chinnaswamy, A. U. Rehman, and H. Hamam, "X-malnet: A cnn-based malware detection model with visual and structural interpretability," English, *Computers, Materials, Continua*, vol. 86, no. 2, pp. 1–18, 2026, ISSN: 1546-2218. DOI: 10.32604/cmc.2025.069951.

[12] J. S. Kalapala and L. Zhang, "Packing induced bias in deep learning malware classifiers: A systematic experimental study," in *2026 IEEE 5th International Conference on AI in Cybersecurity (ICAIC)*, Feb. 2026, pp. 1–6. DOI: 10.1109/ICAIC67076.2026.11395794. [Online]. Available: https://ieeexplore.ieee.org/document/11395794/.

[13] D. Gibert, N. Totosis, C. Patsakis, Q. Le, and G. Zizzo, "Assessing the impact of packing on static machine learning-based malware detection and classification systems," *Computers Security*, vol. 156, p. 104 495, 2025, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2025.104495. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482500183X.

[14] E. Alkhateeb, A. Ghorbani, and A. H. Lashkari, "Packed malware detection using grayscale binary-to-image representations," en, *arXiv preprint arXiv:2512.15414*, no. arXiv:2512.15414, Dec. 2025, arXiv:2512.15414 [cs]. DOI: 10.48550/arXiv.2512.15414. [Online]. Available: http://arxiv.org/abs/2512.15414.

[15] Y. Yu et al., "Semantic lossless encoded image representation for malware classification," en, *Scientific Reports*, vol. 15, no. 1, p. 7997, Mar. 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-88130-0.

[16] K. Mahmud Sujon, R. Binti Hassan, M. Abdullah-Al-Wadud, and J. Uddin, "Optistack: A hybrid ensemble learning and xai-based approach for malware detection in compressed files," *IEEE Access*, vol. 13, pp. 104 992–105 026, 2025, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3579880.

[17] M. Brosolo, V. Puthuvath, and M. Conti, "The road less traveled: Investigating robustness and explainability in cnn malware detection," en, *arXiv preprint arXiv:2503.01391*, no. arXiv:2503.01391, Mar. 2025, arXiv:2503.01391 [cs.CR]. DOI: 10.48550/arXiv.2503.01391. [Online]. Available: http://arxiv.org/abs/2503.01391.

[18] A. Amalan, G. Smaragdakis, and T. J. Viering, *Maltree: Tracing malware evolution from embeddings at scale*, 2026. arXiv: 2606.06570 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2606.06570.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," en, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[20] G. Ciaramella, F. Martinelli, F. Mercaldo, and A. Santone, "Exploring quantum machine learning for explainable malware detection," en, in *2023 International Joint Conference on Neural Networks (IJCNN)*, Gold Coast, Australia: IEEE, 2023, pp. 1–6, ISBN: 978-1-6654-8867-9. DOI: 10.1109/IJCNN54540.2023.10191964. [Online]. Available: https://ieeexplore.ieee.org/document/10191964/.

[21] G. Ciaramella, F. Mercaldo, and A. Santone, "Dynamic analysis for explainable fine-grained android malware detection," in *Security and Trust Management*, F. Martinelli and R. Rios, Eds., Cham: Springer

Nature Switzerland, 2025, pp. 110–127, ISBN: 978-3-031-76371-7.

[22] H. Wang et al., "Score-cam: Score-weighted visual explanations for convolutional neural networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 111–119. DOI: 10.1109/CVPRW50498.2020.00020. [Online]. Available: https://openaccess.thecvf.com/content_CVPRW_2020/html/w1/Wang_Score-CAM_Score-Weighted_Visual_Explanations_for_Convolutional_Neural_Networks_CVPRW_2020_paper.html.

[23] S. Roeser and S. Copeland, *TU Delft Code of Conduct: Why What Who How*, English. Netherlands: Delft University of Technology, 2020. DOI: 10.4233/uuid:704e72b8-6b14-4cf1-a931-9c0f93c50152.

[24] B. E. Granger and F. Pérez, "Jupyter: Thinking and storytelling with code and data," *Computing in Science Engineering*, vol. 23, no. 2, pp. 7–14, 2021. DOI: 10.1109/MCSE.2021.3059263.

[25] J. Ansel et al., "PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*, ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. [Online]. Available: https://docs.pytorch.org/assets/pytorch2-2.pdf.

[26] T. maintainers and contributors, *TorchVision: PyTorch's Computer Vision library*, Nov. 2016. [Online]. Available: https://github.com/pytorch/vision.

[27] J. Gildenblat and contributors, *Pytorch library for cam methods*, https://github.com/jacobgil/pytorch-grad-cam, 2021.

[28] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.

[29] The pandas development team, *pandas-dev/pandas: Pandas*. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://github.com/pandas-dev/pandas.

[30] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[31] A. Clark and Contributors, *Pillow (PIL Fork)*, 2010. [Online]. Available: https://github.com/python-pillow/Pillow.

A Appendix

Table 10: Mapping of individual PE and ELF sections to broader analytical categories.

Category	**Included Sections**
PE_HEADER	`PE_HEADER`
Code	`.text`, `.init`, `.fini`
Data (Var./Const.)	`.data`, `.bss`, `.rdata`, `.tls`, `.rodata`, `.sdata`, `.data.rel.ro`
Linking & Memory	`.idata`, `.edata`, `.reloc`, `.dynamic`, `.dynsym`, `.dynstr`, `.hash`, `.gnu.hash`, `.got`, `.got.plt`, `.plt`, `.plt.got`, `.rel.dyn`, `.rela.dyn`, `.rela.plt`, `.rld_map`, `.shstrtab`
Runtime & Exceptions	`.CRT`, `.eh_fram` (PE), `.eh_frame` (ELF), `.init_array`, `.fini_array`, `.eh_frame_hdr`, `.note.gnu.property`, `.ARM.exidx`, `.ARM.attributes`, `.MIPS.abiflags`, `.reginfo`, `.pdr`

Hyperparameter	**Value**
Batch Size	64
Optimiser	Adam
Learning Rate	1×10^{-4}
Learning Decay	None
Number of Epochs	30
Dropout	0.2
Regularisation	None

Table 11: Model hyperparameters used during training.

Package	**Version**	**Reference**
Python	3.11	
Jupyter	1.1.1	[24]
PyTorch	2.12.0	[25]
Torchvision	0.27	[26]
pytorch-grad-cam	1.5.5	[27]
NumPy	2.4.6	[28]
pandas	3.0.3	[29]
OpenCV Headless	4.13.0.92	[30]
PIL	12.2.0	[31]

Table 12: Software packages used in this research.

Table 13: Confusion Matrix for CNNs trained on UPX packers (summed over 5 cross-validation folds)

Pred True	Gafgyt	Mirai	Tsunami	XorDDoS
Gafgyt	102	71	66	0
Mirai	59	130	58	0
Tsunami	5	4	241	0
XorDDoS	3	1	49	124

Table 7: Dataset distribution

family packer	Adware	BenignELF	BenignPE	Botnet	Gafgyt	HiddenWasp	Mirai	Ransomware	Rootkit	Spyware	Trojan	Tsunami	Worm	XorDDoS	Total
mangle	56	0	60	74	0	0	0	54	59	75	66	0	76	0	520
mpress	73	0	65	45	0	0	0	65	67	69	83	0	53	0	520
none	715	715	715	715	715	715	715	715	715	715	715	715	715	715	10010
upx	72	60	81	55	80	0	89	73	59	54	72	85	54	83	917
UPX (brute)	61	83	69	60	77	0	76	66	64	66	68	84	66	78	918
UPX (LZMA)	61	86	70	70	82	0	82	65	66	78	52	81	58	17	868
XOR (Static Fixed)	62	78	75	69	69	80	81	60	72	65	52	84	65	85	997
XOR (Static Random)	59	87	66	60	76	72	76	65	71	66	53	88	80	78	997
XOR (Rolling Fixed)	78	77	54	71	74	80	86	68	68	54	68	77	59	83	997
XOR (Rolling Random)	52	84	67	62	85	84	77	60	71	64	71	69	73	78	997
zip	70	81	59	82	83	89	82	58	51	64	64	73	72	69	997
zlib	71	79	49	67	89	86	66	81	67	60	66	74	59	83	997
Total	1430	1430	1430	1430	1430	1206	1430	1430	1430	1430	1430	1430	1430	1369	19735

Table 8: Cross-Packer Mean F1 scores and Standard Deviation

test packer train packer	mangle	mpress	none	upx	upx brute	upx lzma	xor fixed	xor random	xor rolling fixed	xor rolling random	zip	zlib
mangle	1.000 ± 0.000	0.016 ± 0.009	0.033 ± 0.015	0.077 ± 0.019	0.074 ± 0.009	0.088 ± 0.014	0.015 ± 0.010	0.017 ± 0.011	0.022 ± 0.024	0.012 ± 0.008	0.016 ± 0.006	0.015 ± 0.006
mpress	0.067 ± 0.038	0.997 ± 0.005	0.047 ± 0.019	0.038 ± 0.022	0.043 ± 0.020	0.053 ± 0.031	0.039 ± 0.010	0.045 ± 0.010	0.045 ± 0.003	0.040 ± 0.004	0.013 ± 0.010	0.013 ± 0.010
none	0.100 ± 0.061	0.097 ± 0.017	1.000 ± 0.000	0.002 ± 0.003	0.014 ± 0.003	0.018 ± 0.010	0.646 ± 0.039	0.744 ± 0.044	0.703 ± 0.042	0.682 ± 0.034	0.713 ± 0.030	0.713 ± 0.032
upx	0.208 ± 0.046	0.000 ± 0.000	0.047 ± 0.020	0.947 ± 0.010	0.749 ± 0.044	0.344 ± 0.054	0.043 ± 0.028	0.037 ± 0.014	0.053 ± 0.021	0.037 ± 0.012	0.032 ± 0.037	0.032 ± 0.016
upx brute	0.135 ± 0.039	0.000 ± 0.000	0.016 ± 0.009	0.827 ± 0.027	0.863 ± 0.020	0.362 ± 0.039	0.032 ± 0.024	0.046 ± 0.013	0.032 ± 0.014	0.044 ± 0.017	0.028 ± 0.017	0.033 ± 0.011
upx lzma	0.112 ± 0.078	0.054 ± 0.071	0.068 ± 0.021	0.360 ± 0.033	0.345 ± 0.014	0.796 ± 0.013	0.056 ± 0.029	0.080 ± 0.023	0.083 ± 0.035	0.082 ± 0.028	0.059 ± 0.021	0.059 ± 0.027
xor fixed	0.097 ± 0.071	0.132 ± 0.019	0.670 ± 0.059	0.061 ± 0.023	0.089 ± 0.047	0.084 ± 0.054	0.995 ± 0.005	0.903 ± 0.014	0.906 ± 0.025	0.930 ± 0.019	0.625 ± 0.048	0.637 ± 0.043
xor random	0.044 ± 0.035	0.159 ± 0.045	0.975 ± 0.009	0.049 ± 0.023	0.023 ± 0.016	0.033 ± 0.010	0.990 ± 0.012	0.992 ± 0.008	0.994 ± 0.008	0.988 ± 0.006	0.705 ± 0.094	0.722 ± 0.092
xor rolling fixed	0.046 ± 0.042	0.149 ± 0.013	0.812 ± 0.057	0.051 ± 0.023	0.034 ± 0.029	0.050 ± 0.042	0.796 ± 0.057	0.875 ± 0.026	0.994 ± 0.006	0.909 ± 0.021	0.666 ± 0.063	0.673 ± 0.056
xor rolling random	0.035 ± 0.023	0.127 ± 0.017	0.923 ± 0.014	0.031 ± 0.025	0.031 ± 0.025	0.030 ± 0.020	0.987 ± 0.004	0.979 ± 0.010	0.985 ± 0.008	0.995 ± 0.005	0.757 ± 0.042	0.766 ± 0.036
zip	0.035 ± 0.026	0.076 ± 0.057	0.705 ± 0.029	0.090 ± 0.041	0.113 ± 0.023	0.076 ± 0.031	0.780 ± 0.043	0.826 ± 0.034	0.840 ± 0.030	0.851 ± 0.039	0.998 ± 0.003	0.998 ± 0.003
zlib	0.037 ± 0.023	0.148 ± 0.040	0.713 ± 0.047	0.120 ± 0.046	0.166 ± 0.049	0.087 ± 0.037	0.757 ± 0.031	0.832 ± 0.031	0.869 ± 0.043	0.866 ± 0.034	1.000 ± 0.000	1.000 ± 0.000

Table 9: Full occlusion results

category packer	Code	Code (Random)	Data (Var./Const.)	Data (Var./Const.) (Random)	Linking & Memory	Linking & Memory (Random)	PE_HEADER	PE_HEADER (Random)	Runtime & Exceptions	Runtime & Exceptions (Random)
none	0.6721 ± 0.0396	0.3918 ± 0.0321	0.9564 ± 0.0169	0.9845 ± 0.0079	0.9863 ± 0.0081	0.9875 ± 0.0049	0.9997 ± 0.0004	0.9997 ± 0.0004	0.9990 ± 0.0008	0.9977 ± 0.0016
xor_fixed	0.9542 ± 0.0157	0.7644 ± 0.0300	0.8926 ± 0.0322	0.9887 ± 0.0058	0.9934 ± 0.0071	0.9682 ± 0.0112	0.9990 ± 0.0022	0.9990 ± 0.0022	0.9932 ± 0.0101	0.9954 ± 0.0046
xor_random	0.9752 ± 0.0248	0.7171 ± 0.0188	0.7538 ± 0.0751	0.9540 ± 0.0163	0.9715 ± 0.0165	0.9502 ± 0.0228	0.9963 ± 0.0083	0.9963 ± 0.0083	0.9671 ± 0.0245	0.9782 ± 0.0076
xor_rolling_fixed	0.8787 ± 0.0164	0.7329 ± 0.0446	0.9010 ± 0.0569	0.9863 ± 0.0056	0.9910 ± 0.0063	0.9730 ± 0.0048	0.9963 ± 0.0040	0.9951 ± 0.0050	0.9946 ± 0.0076	0.9916 ± 0.0029
xor_rolling_random	0.9758 ± 0.0121	0.7050 ± 0.0311	0.7520 ± 0.0443	0.9437 ± 0.0059	0.9890 ± 0.0122	0.9362 ± 0.0262	0.9991 ± 0.0020	0.9980 ± 0.0028	0.9660 ± 0.0325	0.9802 ± 0.0098
zip	0.9983 ± 0.0039	0.8718 ± 0.0365	0.8324 ± 0.0936	0.9203 ± 0.0176	0.9856 ± 0.0277	0.9269 ± 0.0556	0.9983 ± 0.0039	0.9958 ± 0.0059	0.9974 ± 0.0039	0.9623 ± 0.0103
zlib	1.0000 ± 0.0000	0.8652 ± 0.0232	0.8930 ± 0.0270	0.9249 ± 0.0278	0.9989 ± 0.0026	0.9341 ± 0.0298	1.0000 ± 0.0000	0.9990 ± 0.0023	0.9898 ± 0.0143	0.9782 ± 0.0191