Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.3907

Deep Learning Applied to Imbalanced Malware Datasets Classification

Marcelo Palma Salas [**Universidade de Campinas (UNICAMP)** | ***marcelopalma@ic.unicamp.br***]
Paulo Lício de Geus [**Universidade de Campinas (UNICAMP)** | ***pgeus@unicamp.br***]

Institute of Computing, Universidade de Campinas (UNICAMP), Av. Albert Einstein, 1251 - Cidade Universitária, Campinas - SP, 13083-852, Brazil.

Received: 07 December 2023 • **Accepted:** 29 May 2024 • **Published:** 16 September 2024

Abstract In the current day, the evolution and exponential proliferation of malware involve modifications and camouflage of their structure through techniques like obfuscation, polymorphism, metamorphism, and encryption. With the advancements in deep learning, methods such as convolutional neural networks (CNN) have emerged as potent tools for deciphering intricate patterns within this malicious software. The present research uses the capacity of CNN to learn the global structure of the code converted to an RGB or grayscale image and decipher the patterns present in the malware datasets generated from these images. The study explores fine-tuning techniques, including bicubic interpolation, ReduceLROnPlateau, and class weight estimation, in order to generalize the model and reduce the risk of overfitting for malware that uses evasion techniques against classification. Taking advantage of transfer learning and the MobileNet architecture, we created a MobileNet fine-tuning (FT) model. The application of this new model in four datasets, including Microsoft Big 2015, Malimg, MaleVis, and a new Fusion dataset, achieved 98.71%, 99.08%, 96.04%, and 98.04% accuracy, respectively, which underscores the robustness of the proposed model. The Fusion dataset is a combination of the first three datasets, consisting of a set of 32,601 known malware image files representing a mix of 59 different families. Despite the success, the study reveals performance deterioration with an increase in the number of malware families, highlighting the need for further exploration into the limits of CNNs in malware classification.

Keywords: malware, classification, CNN, MobileNet, interpolation, Big 2015, Malimg, MaleVis, Fusion, dataset

1 Introduction

In the current day, the evolution and exponential growth of malware have posed significant challenges for cybersecurity researchers [AV-TEST GmbH, 2023]. Attackers modify and camouflage the structure of this malicious software through the use of obfuscation, polymorphism, metamorphism, and encryption techniques. Hence, conventional techniques such as signature-based and heuristic-based analysis become less effective, prompting a need for more sophisticated analysis methods capable of deciphering the patterns and characteristics of malware.

With the progress of deep learning, techniques such as Convolutional Neural Networks [LeCun *et al*., 2015] (CNN[1]) have proven to be powerful tools for deciphering complex patterns in large image datasets. As a CNN processes an image, it generates a set of feature maps that indicate where the features searched by each filter have been detected.

In Nataraj *et al*. [2011], the authors introduced a method to transform binary code from malware families into image format under an approach based on texture features of images. Leveraging advancements in deep learning, this approach utilizes Convolutional Neural Networks (CNNs) to classify malware by converting binary code into RGB or grayscale images. This research relies on the CNNs' ability to learn the overall structure of the code and extract patterns from the datasets generated from these images.

The approach to convert binary code into an image was applied in Palma Salas *et al*. [2023], which achieved 98.41% accuracy and 0.08078 log loss in identifying and categorizing malware samples. Building upon Palma Salas *et al*. [2023], the present research further explores the utilization of hyperparameters and other techniques to enhance the efficiency in model training execution time and the effectiveness of malware classification performance metrics using CNNs across three benchmarking datasets and a new dataset of malware families.

The application of interpolation and resampling techniques allowed us to standardize the size and resolution of malware images to match the input requirements of the MobileNet architecture. Additionally, these techniques ensured that all images in the dataset had uniform dimensions, reducing training time for the CNN and improving classification results by ensuring that each image was represented more consistently.

Furthermore, we implemented ReduceLROnPlateau to automatically adjust the learning rate during model training based on validation performance. This technique enabled more careful training and helped the model converge towards a global or local optimum.

Malware analysis datasets are often imbalanced, with many more samples of one class compared to others. This

[1] A CNN is a type of neural network characterized by its ability to perform convolutional operations, i.e., mathematical operations applied to images, with the main objective of learning filters that detect specific patterns in images.

research analyzed class imbalance across three well-known datasets: the Microsoft Big 2015 datasets [Ronen *et al*., 2018], the Malimg dataset [Nataraj *et al*., 2011], and the MaleVis dataset [Bozkir *et al*., 2019]. Additionally, we introduced the Fusion dataset, which combines the first three datasets and consists of 59 malware families with 32,601 samples. The Fusion Dataset not only provides a diverse and extensive representation of malware variants but also serves as a benchmark for evaluating the robustness of CNN classification architectures.

One approach to handling class imbalance is the use of the class weight technique. We employed this technique to correct the imbalance bias in malware classification across the four datasets.

The application of fine-tuning through techniques such as ReduceLROnPlateau, estimating class weights, and k-fold cross-validation on the MobileNet architecture resulted in a model called MobileNet Fine Tuning (MobileNet FT), which achieved 98.71%, 99.08%, 96.04%, and 98.04% accuracy on the Microsoft Big 2015, Malimg, MaleVis, and Fusion datasets, respectively.

Despite the advances provided by CNNs in deciphering patterns within the classification of malware families and the use of fine-tuning techniques, we observed a decline in performance metrics (e.g., accuracy and logloss) due to the increased number of malware classes or families. This deterioration can potentially be mitigated by applying fine-tuning techniques to MobileNet or another architecture, but it does not guarantee that the model will effectively support a greater number of malware families.

The remainder of the article is organized as follows: Section 2 provides an overview of the state of the art in malware classification using convolutional neural networks; Section 3 covers CNNs, transfer learning, MobileNet architecture, fine-tuning in CNNs, and image resizing techniques. The description of the improvements in resizing and bicubic interpolation of datasets, along with the new MobileNet model fine-tuning, is presented in Section 4. Section 5 presents the results and analysis of the experiments conducted in this research. The impact of the increase in malware family classification units on the performance metrics is discussed in Section 6. Section 7 outlines the limitations of the present research. Finally, the conclusions of the research, including its main contributions and future work, are presented in Section 8.

2 Related work

The dynamic nature of the cybersecurity landscape requires continuous re-evaluation of CNN-based classification models to adapt to the emergence of new and larger malware families. According to Sun *et al*. [2017], the success of deep learning in vision can be attributed to: (a) models with high capacity; (b) increased computational power; and (c) the availability of large-scale labeled data. They also found that performance on vision tasks increases logarithmically based on the volume of training data, and one can improve performance on many vision tasks by just training a better base model. Additionally, the authors in Sun *et al*. [2017] encourage the community to continue studying the behavior of models in the face of large amounts of information.

The current state of the art in research related to malware classification primarily revolves around the use of convolutional neural network (CNN) algorithms and techniques. Typically, most studies in this domain utilize three widely recognized datasets—Microsoft Big 2015, Malimg, and MaleVis—to assess the effectiveness of their proposed.

In Hemalatha *et al*. [2021], the authors proposed the utilization of a reweighted class-balanced loss function in the final classification layer of their model, which is based on DenseNet architecture. This approach aimed to achieve performance improvements in classifying malware. The method yields an accuracy of 98.46% for the Microsoft BIG 2015 dataset, 98.23% for the Malimg dataset, 98.21% for the MaleVis dataset, and 89.48% for the unseen Malicia dataset. Similarly, Wang *et al*. [2021] employed a set of layers as the Depthwise Efficient Attention Module (DEAM) alongside a DenseNet architecture, utilizing grayscale images transformed from malware. Their model achieved an accuracy of 99.3% for malware detection on their dataset and 98.5% and 97.3% on the Malimg dataset and BIG 2015 dataset, respectively.

Gibert *et al*. [2019] propose a deep learning approach for malware classification into families based on a set of patterns extracted from their visualization as images using the benchmark Microsoft Malware Classification Challenge dataset (BIG 2015) and the Malimg dataset. The results obtained in both approaches, 98.28% and 97.50%, respectively, demonstrated the effectiveness of CNNs to classified malign software. This offers promising prospects for the improvement of detection techniques and the classification of computer threats.

The authors in Roseline *et al*. [2020] proposed a system based on a layered ensemble approach that mimics the key characteristics of deep learning techniques, does not require hyperparameter tuning or backpropagation, and works with reduced model complexity. The model obtained 98.65%, 97.2%, and 97.43% for the Malimg, BIG 2015, and MaleVis malware datasets, respectively.

The study in Aslan and Yilmaz [2021] introduces a novel hybrid architecture that integrates two widely-used pre-trained network models, namely AlexNet and ResNet-152. The proposed method combines these renowned pre-trained network models to form a hybrid model. Initially, features are extracted using these pre-trained networks, followed by the training phase of a deep neural network architecture using a supervised learning approach. The model achieved accuracies of 97.78%, 94.88%, and 96.6% for the Malimg, BIG 2015, and MaleVis datasets, respectively.

In Shaik *et al*. [2023], the authors used three datasets classified by sample imbalance by malware families. Applying six transfer learning architectures, they achieved 97% for the imbalanced Malimg dataset, 95% for the intermediate imbalance Blended dataset (combination of datasets Malimg and MaleVis with 50 malware families), and 95% also for the best balanced MaleVis dataset. From the comparative analysis, the authors observed that the greater the class imbalance in the dataset, the greater the variance in the performance of different models and the number of epochs required for con-

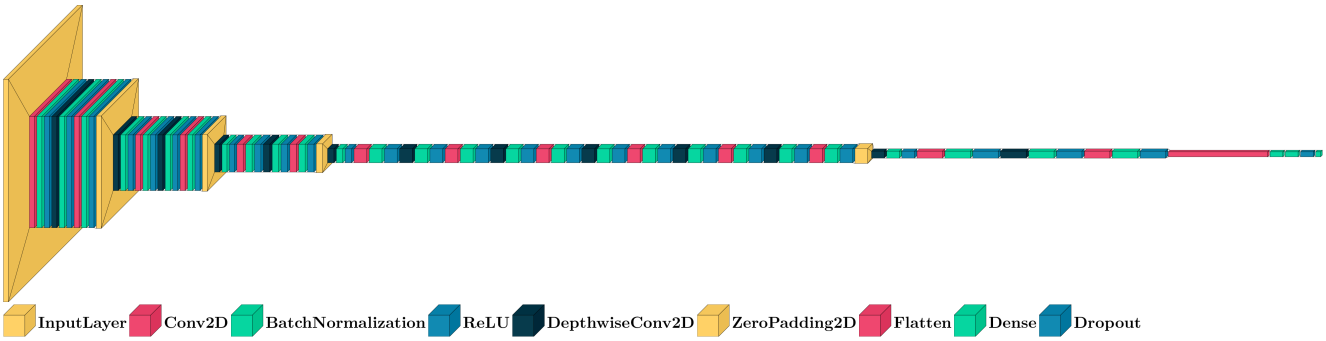

Figure 1. An example of convolutional neural network using MobileNet layers.

vergence.

In Kalash *et al*. [2018], the authors propose a deep learning framework for malware classification through a CNN-based architecture to classify malware samples. They converted malware binaries to grayscale images. Experiments on two challenging malware classification datasets, Malimg and Microsoft (Big 2015) malware, achieve 98.52% and 99.97% accuracy on the Malimg and Microsoft datasets, respectively.

3 Background

3.1 Convolutional Neural Networks

Convolutional Neural Networks [LeCun *et al*., 1998] (CNN) are a type of deep learning architecture originally designed to process and analyze image/video data. This architecture has proven to be very effective in object recognition, feature detection, segmentation, and image classification tasks, among others. The key innovation of CNNs lies in their ability to automatically learn hierarchical representations of features from raw pixel values, enabling them to capture spatial hierarchies and patterns in visual data. The CNN consist of many layers, as shown in Fig. 1. Each serving a specific purpose. Here are the main types of layers in a typical CNN [LeCun *et al*., 2015]:

- Activation (ReLU) Layer. The activation layer introduces non-linearity using the Rectified Linear Unit (ReLU) function, replacing negative values with zero ($f(x) = \max(0, x)$) to help the model learn complex relationships in the data.
- Pooling (Subsampling) Layer. This layer reduces the spatial dimensions of feature maps, decreasing parameters and computational load. Common techniques include max pooling and average pooling.
- Fully Connected (Dense) Layer. This layer connects every neuron in one layer to every neuron in the next layer, enabling the network to learn global patterns and make predictions.
- Flatten Layer. This layer reshapes the output from the preceding layer into a one-dimensional vector, preparing it for input to the fully connected layers.
- Normalization Layers. This layer, like Batch Normalization, enhance training stability and speed by normalizing layer inputs. Batch Normalization reduces internal covariate shift by applying the transformation $y_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \cdot \gamma + \beta$, where μ and σ^2 are the mean and variance, ϵ is a small constant, and γ and β are learnable parameters.
- Dropout Layer. This layer is a regularization technique that randomly sets a fraction of neurons to zero during training (x is input, p is dropout probability). Remaining values are scaled by $1/(1-p)$ to maintain the expected value, preventing overfitting.
- Softmax Layer. This layer, used in the output layer for multi-class classification, converts raw scores ($z = [z_1, z_2, ..., z_k]$) into probabilities ($p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$). The class with the highest probability is predicted as the final output.

3.2 Transfer Learning

This technique [LeCun *et al*., 2015] consists of using the knowledge acquired in one task to improve performance in another related task. In the context of CNN, transfer learning involves taking a network pretrained on a large dataset and applying it to a different task using previously learned weights and features.

Pre-trained CNNs, such as VGG, ResNet, Inception, and MobileNet, among others, have been trained on massive datasets, such as ImageNet [Russakovsky *et al*., 2015], which contains millions of images of different classes. These networks have learned to extract relevant visual features from images, allowing them to capture complex and subtle patterns.

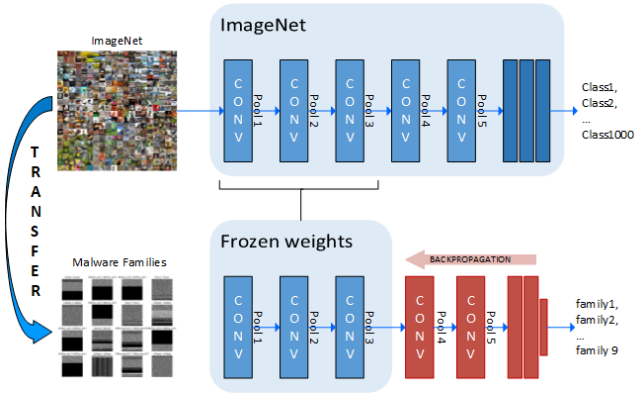

Figure 2. The use of transfer learning with CNN for malware classification

Using transfer learning with CNN for malware classification, as illustrated in Fig.2, proves advantageous due to the

insufficient data available in certain datasets. For example, in the Microsoft Big 2015 dataset, the Simda family accounts for only 0.4% (42 malware samples), while the Kelihos_ver3 family represents 27.1% (2942 samples of the total). The imbalance issue of the Big 2015 Dataset is evident in Fig.3.

Other benefits of using transfer learning include the ability to generalize features, which allows the model to recognize patterns across different datasets and tasks more effectively. Additionally, transfer learning can significantly reduce training time by leveraging pre-trained models and learned features, saving computational resources. Moreover, it helps prevent overfitting by transferring knowledge from a larger dataset to a smaller one, thus improving the model's generalization capabilities.

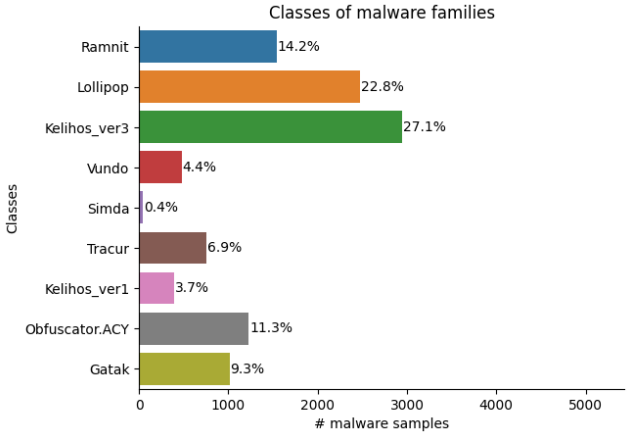

Figure 3. Microsoft Big 2015 Dataset with sample imbalance in nine malware families.

3.3 MobileNet Architecture

Introduced by Howard *et al*. [2017], this architecture was oriented to address the need for efficient deep learning models that could run on devices with limited computational power, such as mobile phones or embedded systems. This architecture is characterized by its use of depthwise separable convolutions, which significantly reduces the number of parameters and computations compared to traditional convolutional layers.

This architecture is based on factorizing traditional convolutions into two types of layers: a first depthwise convolutional layer and a 1x1 pointwise convolutional layer. This division allows us to reduce the computational cost and the size of the model between eight and nine times greater than the computational cost of both depthwise and poinwise layers [Srudeep, 2020].

The depth-wise separable convolution is composed of two layers: the depth-wise convolution and the point-wise convolution. Basically, the first layer is used to filter the input channels, and the second layer is used to combine them to create a new feature.

In simpler terms, depth-wise convolution helps MobileNet efficiently capture features within each color channel (RGB) separately, helping to detect specific characteristics on each channel, and point-wise convolution helps combine these features in a computationally efficient way. This design, as seen in Table 1, allows MobileNet to achieve a good trade-off between model size, computational efficiency, and accuracy, making it well-suited for real-time applications on resource-constrained devices, such as mobile phones and IoT devices.

Table 1. MobileNet body architecture [Howard *et al*., 2017].

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

3.4 Fine-Tuning in CNN

Fine-tuning [Patterson and Gibson, 2017] is a commonly used technique in convolutional neural networks (CNNs) to adapt a pretrained model to a new dataset or specific task. Here are some techniques that can be employed during the fine-tuning process in CNNs:

1. Unfreezing Specific Layers: When applying fine-tuning, some layers of the pretrained network are typically unfrozen, especially layers closer to the output, while deeper layers that capture more general features are kept frozen. This allows the unfrozen layers to adjust to the specific features of the new dataset, while the frozen layers retain the general features learned during pretraining.
2. Adjusting Learning Rate: During fine-tuning, it's common to adjust the learning rate for the unfrozen layers. A lower learning rate is typically used for the pretrained layers and a higher learning rate for the new layers being trained. This helps prevent drastic changes in the weights of the pretrained layers and allows the new layers to adapt more quickly to the new dataset.
3. Data Augmentation: Data augmentation can be applied during fine-tuning to generate additional variations in the training data and prevent overfitting. This can include techniques such as rotation, horizontal flipping, random cropping, and brightness adjustment. Data augmentation helps increase the diversity of the training data and improves the model's ability to generalize to new images.
4. Regularization: Regularization is used to prevent overfitting during fine-tuning. This can include techniques

such as dropout (randomly dropping neurons during training), L1 or L2 regularization (penalizing large weights), and batch normalization (normalizing activation values for each layer). Regularization helps improve the model's generalization and reduces the risk of overfitting.

5. Staged Training: In some cases, it may be useful to train the model in stages during fine-tuning. For example, initially only a few new layers may be unfrozen and trained, then more layers added as training progresses. This can help prevent abrupt changes in the weights of the pretrained layers and facilitate model convergence.

Fine-tuning in CNNs involves adapting a pretrained model to a new dataset or specific task by selectively modifying the model's layers, adjusting the learning rate, applying regularization techniques, and using data augmentation, among other techniques. These strategies help improve the model's performance on the new task without losing the knowledge learned during pretraining.

3.5 Image Resizing

Image resizing is a fundamental operation in image processing and computer vision, playing a crucial role in many applications such as deep learning [Talebi and Milanfar, 2021]. The process involves altering the dimensions of an image, either to fit a specific display size, reduce computational complexity, or preprocess data for machine learning models [Fadnavis, 2014]. Two key concepts central to image resizing are interpolation and resampling.

3.5.1 Interpolation

This method [Fadnavis, 2014] is used to estimate pixel values at non-integer coordinates when resizing an image. It involves determining the color or intensity of a pixel by considering its neighboring pixels.

In Python, the Pillow library (PIL fork) is widely used for image processing tasks, including resizing. The resize() function in Pillow supports various resampling filter algorithms, such as nearest, bilinear, bicubic, lanczos, and box. The choice of resampling method can significantly impact the quality of the resized image, as seen in Fig. 4.

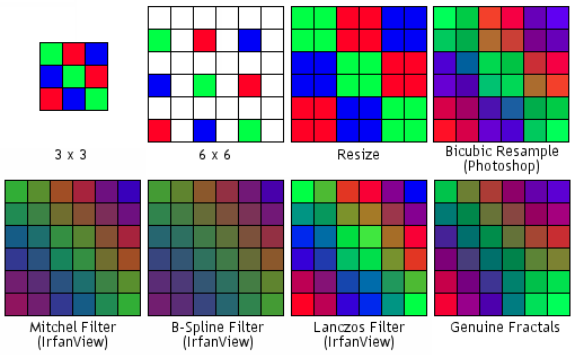

Figure 4. The resampling method can significantly impact the quality of the resized image [HobbyMaker, 2023]

Below, we examine some characteristics of resize filters [Fadnavis, 2014]:

- Nearest-Neighbor Interpolation. The simplest method is to assign the value of the nearest pixel to the target location. In the Pillow Library, we can use the Resampling.NEAREST function.
- Bilinear Interpolation. It considers the weighted average of the four nearest pixels. This method produces smoother results compared to nearest-neighbor interpolation. In the Pillow Library, we can use the Resampling.BILINEAR function[2].
- Bicubic Interpolation. A more sophisticated method that considers a 4x4 neighborhood and uses a cubic polynomial to interpolate pixel values. Bicubic interpolation is known for producing high-quality results but requires more computation. In the Pillow Library, we can use the Resampling.BICUBIC function.
- Lanczos Interpolation. Based on the sync function, this resize filter provides a high-quality interpolation method that mitigates the blurring effect seen in other methods. It is particularly useful for preserving fine details during resizing. In the Pillow Library, we can use the Resampling.LANCZOS, and this filter can only be used with the resize() and thumbnail() methods.

3.5.2 Resampling

As defined by Fadnavis [2014], is the process of adjusting the sampling rate of an image, which entails changing the number of pixels along its dimensions. This procedure is essential in image resizing, as it directly influences the spatial resolution and overall size of the image. Resampling becomes necessary when aligning the image with a specific display size or when preparing data for CNN models with fixed input dimensions, such as the MobileNet architecture, which requires a 224x224 pixel image.

In this research, we use downsampling methods that involve reducing the number of pixels in the image. Common downsampling techniques include averaging or selecting specific pixels from the original image. Other commonly used methods are upsampling and anti-aliasing.

3.5.3 RGB and Grayscale Images

The choice of interpolation and resampling method can depend on the type of image being resized. For RGB (color) images, maintaining color fidelity and avoiding color artifacts are essential. Bilinear and bicubic interpolation methods are commonly used. Lanczos interpolation can be particularly beneficial when preserving fine color details is crucial, but it has a higher computational cost.

For grayscale images, the considerations may differ slightly. Bilinear interpolation is a reasonable choice for general resizing tasks. However, if preserving subtle intensity variations is important, bicubic or Lanczos interpolation may be preferred.

[2]TensorFlow uses the default bilinear interpolation method in the tf.image.resize function to resize images before being used for training CNN models.

In the context of malware classification using a convolutional neural network, resizing allows the adaptation of image-based deep learning techniques to effectively capture features from binary to image representations of malware samples. It facilitates the development of models that can automatically learn and differentiate between many malware families, contributing to the detection of cyber threats.

4 The Proposed Method

This section describes the methods and techniques used to improve the MobileNet model in order to reduce its execution time and improve its quality metrics in the classification of malware families.

4.1 Experiment settings

The runtime environment was a ASUS nv580vd with an Intel® Core™ i7 7700HQ 2,8GHz processor, 16 GB SDRAM, NVIDIA GeForce GTX 1050, 4GB GDDR5 VRAM, Ubuntu 20.04 LTS (64bit). The PIL 9.4.0 library was used for the conversion of binary files to the PNG format using resampling filter algorithms like bicubic interpolation. The libraries to implementation the convolutional network models were TensorFlow, Keras 2.12.0 over Python 3.8.10, Pandas 1.5.3, and Numpy 1.23.5.

4.2 Dataset Description

In recent years, the field of malware classification has witnessed significant advancements driven by the integration of deep learning techniques. However, one persistent challenge in this domain is the availability of large-scale malware families dataset.

To evaluate our improved MobileNet model, we utilized four well-known datasets: the Microsoft Malware Classification Challenge dataset, also known as Big 2015 [Ronen *et al*., 2018], the Malimg (MalIMG) dataset [Nataraj *et al*., 2011], and the MaleVis dataset [Bozkir *et al*., 2019]. Additionally, we introduced a new dataset named Fusion, which combines the first three datasets. The Fusion dataset comprises samples from 59 malware families, and we applied resampling with Bicubic interpolation to each image to create more generalized models. Below, we provide detailed characteristics of each dataset.

4.2.1 The Microsoft Big 2015 Dataset

The well-known Microsoft Malware Classification Challenge dataset [Ronen *et al*., 2018], or Big 2015, is an imbalanced dataset, as shown in Fig.3, consisting of almost 200GB. It comprises a set of 10,868 known malware byte files representing a mix of 9 different families, as depicted in Table2. Each malicious file has a 20-character hash value for unique identification and a class label (1 to 9) representing the family names.

The malicious fileset is made up of raw data with a hexadecimal representation of the file's binary content, without the header, i.e., to ensure sterility. The dataset can be downloaded from the competition website[3].

Table 2. Type and number of malware samples in the Microsoft Big 2015 dataset.

Family Name	# Train Samp.	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownl.
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	Obfuscated mal.
Gatak	1013	Backdoor

4.2.2 The Malimg Dataset

Also known as MalIMG [Nataraj *et al*., 2011], it comprises a set of 9,339 known malware image files representing a mix of 25 different families, as illustrated in Fig. 5. The dataset is almost 1.11 GB in size. After downloading the malimg_dataset.zip file, you will find the malimg_paper_dataset_imgs directory, which consists of 25 sub directories representing malware families converted into images in PNG format. Each malicious image file has a 32-character hash value for unique identification. The images range from 64 to 1024 pixels wide and 208 to 5334 pixels high, with an average size of 0.12 MB per PNG file. The dataset can be downloaded from the Google Drive website[4].

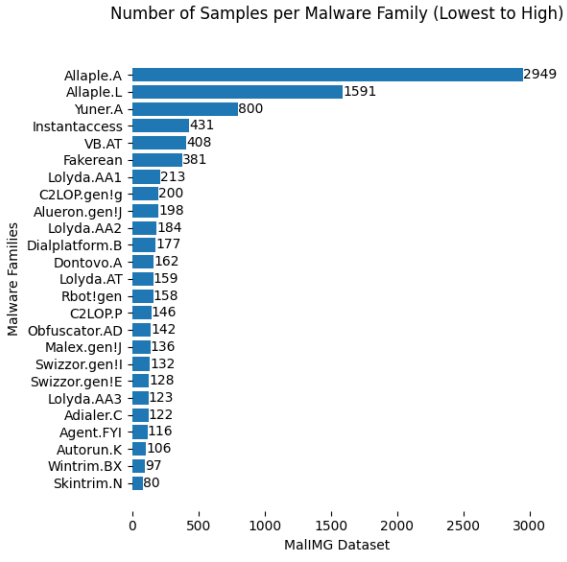

Figure 5. The Malimg dataset has 25 malware families in descending order by number of samples per family.

The malware classes include Allaple.L (worm), Allaple.A (worm), Yuner.A (worm), Lolyda.AA 1 (PWS), Lolyda.AA 2 (PWS), Lolyda.AA 3 (PWS), C2Lop.P (trojan), C2Lop.gen!g (trojan), Instantaccess (dialer), Swizzot.gen!I (TDownloader), Swizzor.gen!E (TDownloader),

[3] https://www.kaggle.com/competitions/malware-classification/

[4] https://drive.google.com/file/d/1M83VzyIQj_kuE9XzhClGK5TZWh1T_pr-/view

VB.AT (worm), Fakerean (rogue), Alueron.gen!J (trojan), Malex.gen!J (trojan), Lolyda.AT (PWS), Adialer.C (Dialer), Wintrim.BX (TDownloader), Dialplatform.B (Dialer), Dontovo.A (TDownloader), Obfuscator.AD (TDownloader), Agent.FYI (backdoor), Autorun.K (worm:autoIT), Rbot!gen (backdoor), Skintrim.N (trojan).

4.2.3 The MaleVis Dataset

The MaleVis dataset [Bozkir *et al*., 2019] consist of a set of 12,394 known malware images files representing a mix of 25 different families, as shown in Fig. 6. Once the MaleVis_train_val_224x224.zip file has been downloaded, we will find the train and val directories, each consisting of 25 sub directories plus one additional directory. Here, one class [Bozkir *et al*., 2019] represents the legitimate samples, while the remaining 25 classes correspond to different malware families. Each malicious image file has a 40-character hash value that is unique for identification purposes. The images have an average size of 0.12 MB per PNG file. The dataset can be downloaded from the project website[5].

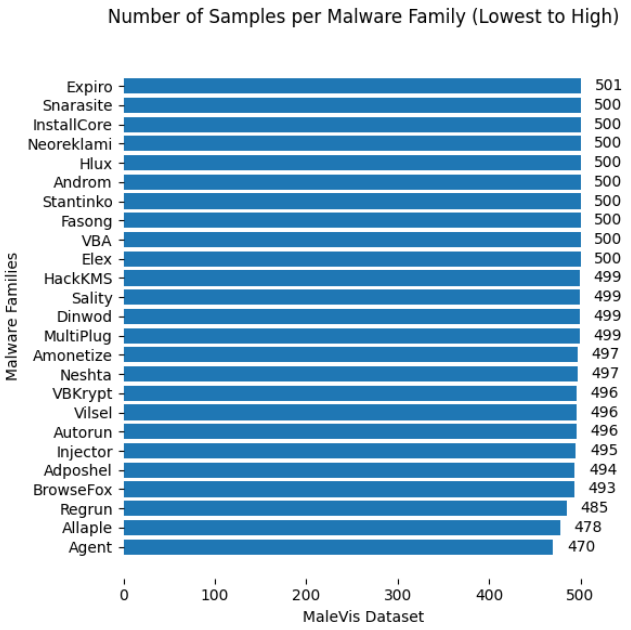

Figure 6. The MaleVis dataset has 25 malware families in descending order by number of samples per family.

Malware classes contain Adposhel (Adware), Agent-fyi (Trojan), Allaple.A (worm), Amonetize (adware), Androm (backdoor), AutoRun-PU (worm),BrowseFox (adware), Dinwod!rfn (trojan), Elex (trojan), Expiro-H (virus), Fasong (worm), HackKMS.A (trojan), Hlux!IK (worm), Injector (trojan), InstallCore.C (adware), MultiPlug (adware), Neoreklami (adware), Neshta (virus), Regrun.A (trojan), Sality (virus), Snarasite.D!tr (trojan), Stantinko (backdoor), Hilium.A (virus) VBKrypt (trojan), Vilsel (trojan).

4.2.4 The Fusion Dataset

This dataset combines the three datasets (Microsoft Big 2015, Malimg, and MaleVis), resulting in a new Fusion dataset comprising a set of 32,601 known malware image files representing a mix of 59 different families, as illustrated in Table 3. The three datasets were subjected to a procedure involving the resizing and interpolation of each image to a standard size of 224x224 pixels in PNG format, utilizing bicubic interpolation through the PIL library. This process is detailed in Section 4.4.

Table 3. Type and number of malware samples in the Fusion dataset.

Nro	Family Name	# Samples	Type
1	Adialer.C	122	Dialer
2	Adposhel	494	Adware
3	Agent-fyi	470	Trojan
4	Agent.FYI	116	Backdoor
5	Allaple	478	Worm
6	Allaple.A	2949	Worm
7	Allaple.L	1591	Worm
8	Alueron.gen!J	198	Trojan
9	Amonetize	497	Adware
10	Androm	500	Backdoor
11	AutoRun-PU	496	Worm
12	AutoIT Auto.K	106	Worm
13	BrowseFox	493	Adware
14	C2Lop.gen!g	200	Trojan
15	C2Lop.P	146	Trojan
16	Dialplatform.B	177	Trojan
17	Dinwod!rfn	499	Trojan
18	Dontovo.A	162	TDownloader
19	Elex	500	Trojan
20	Expiro-H	501	Virus
21	Fakerean	381	Rogue
22	Fasong	500	Worm
23	Gatak	1013	Backdoor
24	HackKMS.A	499	Trojan
25	Hlux!IK	500	Worm
26	Injector	495	Trojan
27	InstallCore	500	Virus
28	Instantaccess	431	Dialer
29	Kelihos_ver1	398	Backdoor
30	Kelihos_ver3	2942	Backdoor
31	Lollipop	2478	Adware
32	Lolyda.AA 1	213	PWS
33	Lolyda.AA 2	184	PWS
34	Lolyda.AA 3	123	PWS
35	Lolyda.AT	159	PWS
36	Malex.gen!J	136	Trojan
37	MultiPlug	499	Adware
38	Neoreklami	500	Adware
39	Neshta	497	Virus
40	Obfuscator.ACY	1228	Obfuscated mal
41	Obfuscator.AD	142	TDownloader
42	Ramnit	1541	Worm
43	Rbot!gen	158	Backdoor
44	Regrun.A	485	Trojan
45	Sality	499	Virus
46	Simda	42	Backdoor
47	Skintrim.N	80	Trojan

Continued in the next column

[5]https://web.cs.hacettepe.edu.tr/ selman/MaleVis/

Nro	Family Name	# Samples	Type
48	Snarasite.D!tr	500	Trojan
49	Stantinko	500	Backdoor
50	Swizzor.gen!E	128	TDownloader
51	Swizzot.gen!I	132	TDownloader
52	Tracur	751	TrojanDownl
54	Worm VBA	500	Worm
53	VB.AT	408	Worm
55	VBKrypt	496	Trojan
56	Vilsel	496	Trojan
57	Vundo	475	Trojan
58	Wintrim.BX	97	TrojanDownl
59	Worm Yuner.A	800	Worm

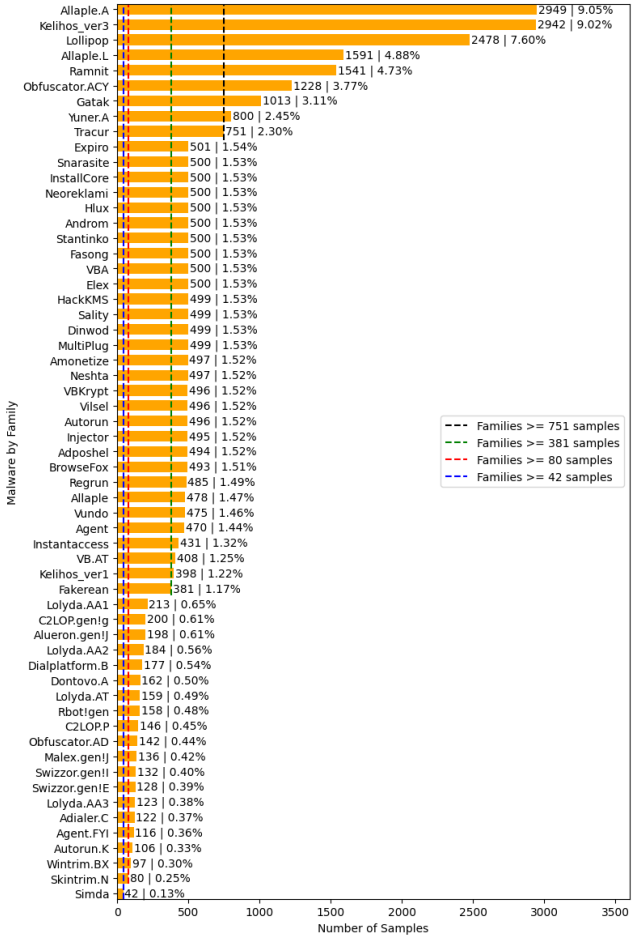

Figure 7. The Fusion dataset consisting of 59 malware families with unbalanced samples in descending order by number of samples per family.

Once the all_families directory has been created, we will find 59 sub directories or malware families that contain image files in PNG format. Each malicious image file has the same name as the original datasets. All images have 224x224 pixels and an average size of 0.09 MB per PNG file, reducing the malware image datasets in comparison with the original datasets. The dataset can be downloaded from the Kaggle datasets website[6].

As we can see in Fig. 7, there is an imbalance of samples in the 59 malware families, where the Allaple.A family has 2,949 samples (9.05%), while Simda has only 42 samples (0.13%).

[6]https://www.kaggle.com/datasets/marcesalas/fusion-dataset-59-malware-families-in-png-format

4.3 Evaluation Metrics

The evaluation or performance metrics used in CNN malware classification allow us to provide a quantitative measure of the performance and quality of the models developed in this task. Below, we describe the metrics used in the present research.

4.3.1 Accuracy

This basic metric measures the proportion of malware classified correctly by the model out of the total number of examples. It is calculated using the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

Where TP is the number of malware samples correctly classified into their respective families, TN is the number of non-malware samples correctly classified as non-malware, FP is the number of non-malware samples incorrectly classified as malware, and FN is the number of malware samples incorrectly classified as non-malware.

4.3.2 Logloss

Also called log loss, is a metric that measures the quality of the classification probabilities generated by a model. It is calculated using the following equation:

$$Logloss = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)) \tag{2}$$

Where N is the total number of examples, (y_i) is the actual label of example (i) (0 or 1), and (p_i) is the predicted probability of example (i) belonging to the positive class.

4.3.3 Precision

The precision is calculated as the proportion of samples correctly classified in a given family with respect to the total number of samples classified in that family. In the context of malware classification, it is calculated as follows in equation (3):

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

Where TP represents the number of samples correctly classified in their respective families and FP represents the number of samples that were incorrectly classified.

4.3.4 Recall

Also known as sensitivity, is calculated as the proportion of correctly classified samples in a certain family with respect to the total number of real samples in that family. In this context, it is calculated as follows in equation (4):

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

Where TP represents the number of samples correctly classified in its respective family and FN represents the number of samples that were incorrectly classified as belonging to other families.

4.3.5 F1-score

It provides a balanced measure of model performance by taking both precision and recall into account. Its formula is as follows in equation (5).

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5}$$

4.4 Visualizing Malware as an Image

This area was introduced by Nataraj *et al*. [2011] who performed the conversion from a byte file to an image, interpreting each byte as a pixel. In this research, the PIL[7] (Python Imaging Library) repository was used to convert byte files into an image in PNG format, as shown in Fig. 8, which scales the values of an image to a matrix from 0 to 255, converting to the 'uint8' type. This conversion allows creating images in four formats: grayscale, grayscale resize 224x224 with bicubic interpolation, RGB, and RGB resize 224x224 with bicubic interpolation.

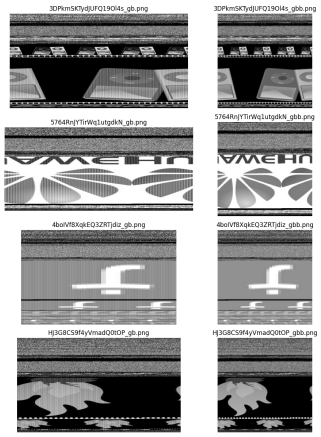

Figure 8. Conversion of malware files to images without filters (left) and with resize and bicubic interpolation (right) in PNG format.

Initially, the bytes files are processed, verifying that the files end in '.bytes' to start processing. Converts hexadecimal data to integers, then the saveimg function to perform additional processing and save resulting images. The pseudo-code is shown in Algorithm 1.

Next, the saveimg function takes an array and a name as input. Then performs a series of operations to resize the array and create an image using the Image library. The resulting image can be subjected to interpolation e.g. bicubic, and resized to 224x224 pixels (same image format used in MobileNet architecture). Finally, the image is saved in PNG format with a name based on the input name, removing the last 6 characters and adding the '.png' extension. The pseudo-code of saveimg function is shown in Algorithm 2.

The process of resizing and interpolating images to a standard size of 224x224 pixels before incorporating them

[7]https://pillow.readthedocs.io/en/stable/

Algorithm 1 Main Loop

```
bytes_files ←′ /byteFiles/′
for x in os.listdir(bytes_files) do
    if str.endswith(x, '.bytes') then
        with open(os.path.join(bytes_files, x), 'r') as f:
        array ← []
        for line in f do
            xx ← line.split()
            if len(xx) ! = 17 then
                continue
            end if
            array.append([int(i, 16) if i ≠′ ??′ else 0 for i in xx[1 :]])
        end for
        saveimg(np.array(array), x)
        del array
    end if
end for
```

Algorithm 2 Pseudo-code for the saveimg function

```
function saveimg(array, name)
    if array.shape[1] ≠ 16 then
        assert(False)
    end if
    b ← int((array.shape[0] × 16)^0.5)
    b ← 2^(int(log(b)/ log(2))+1)
    a ← int(array.shape[0] × 16/b)
    array ← array[: a × b//16, :]
    array ← np.reshape(array, (a, b))
    im ← Image.fromarray(np.uint8(array), mode='L')
        ▷ Apply bicubic interpolation and resize to (224, 224)
    im ← im.resize((224, 224), Image.BICUBIC)
    im.save(′/image/′ + name[: −6] +′ .png′, "PNG")
end function
```

into the training of convolutional networks, such as TensorFlow [Keras, 2023], is highlighted in Fig. 8. This technique enhances:

- Software efficiency. Avoid using the default (nearest) filter of ImageDataGenerator [Keras, 2023] from the TensorFlow Keras library, improving runtime efficiency and efficacy by avoiding using a poor filter to resize malware images.
- Hardware efficiency. By using a standard image input size, specialized hardware such as a GPU can be optimized to work with large volumes of data that have equal dimensions, resulting in more efficient processing and improved performance.
- Memory saving. By having all images the same size, you can load them in batches more efficiently since you don't need to reserve memory for different image sizes.
- Single interpolation. By performing interpolation and resizing beforehand, you reduce the computational load on each iteration rather than repeatedly during training.
- Improves generalization. It facilitates the training and comparison of patterns learned from features that are consistently presented across images.
- Constant input size. Many CNN architectures are designed to accept fixed-size images as input. For example, MobileNet uses 224x224 pixels.

5 Experimental Result and Analysis

This section describes the procedure used to enhance the MobileNet model from the article by Palma Salas *et al*. [2023] for malware classification, applied to four datasets described in Subsection 4.2.

5.1 Conversion of Malicious Binary Files to PNG Images

Based on the procedure outlined in Subsection 4.4, four styles of PNG images[8] were generated to obtain evaluation metrics and assess the performance of MobileNet on each style of PNG image. The following describes each of the styles:

1. Grayscale. Converting byte files into grayscale PNG format. This style generates images without using a standard size.
2. Grayscale W/ bicubic. Conversion of byte files into grayscale PNG format by applying 224x224 pixel resizing with bicubic interpolation.
3. RGB. Conversion of byte files into RGB PNG format. This style generates images without using a standard size.
4. RGB W/ bicubic. Conversion of byte files into RGB PNG format by applying 224x224 pixel resizing with bicubic interpolation.

[8]PNG was chosen because it is a graphic format based on a lossless compression algorithm for bitmaps that is not subject to patents.

Each style returns a dataset composed of 10,868 PNG images, making a total of 43,472 images stored in four directories. Following the MobileNet model implementation in Palma Salas *et al*. [2023], the datasets were divided into 70% for training, 15% for validation, and 15% for testing. The results presented in Table 4 demonstrate that using grayscale images in PNG format, resized to 224x224 pixels using bicubic interpolation, enhances performance metrics: precision improves to 98.7116%, log loss decreases to 0.0614, and F1-Score increases to 97%, improving the results obtained in Palma Salas *et al*. [2023].

Table 4. Results of the procedure to convert binary files to images.

MobileNet FT	Accuracy	Logl.	F1-Score
MobileNet [Palma Salas *et al*., 2023]	98.41%	0.081	96%
Grayscale	98.2822%	0.066	95.07%
Grayscale W/ Bicubic	**98.7116%**	**0.061**	**97%**
RGB	97.4233%	0.122	93%
RGB W/ bicubic	97.7914%	0.076	94%

5.2 Fine-tuning MobileNet Model

We use the base MobileNet model implemented in Palma Salas *et al*. [2023], with the training parameters described in Table 5. Also, **the code can be accessed at** `https://github.com/ecram/malware_classification_cnn`.

Table 5. Training parameters for Malware Classifications models implemented in Palma Salas *et al*. [2023].

Parameter	Value
Learning Rate	0.0001
Epochs	50
Optimizer	Adam
Loss Function	Sparse categorical cross entropy
Fold Cross-Validation	$n_splits = 5$
ReduceLROnPlateau	factor=0.1, patience=3
compute$_class_weight$	class$_weight =' balanced'$

The implementation of fine-tuning techniques described in Section 3.4 allows us to improve the performance metrics of the MobileNet model developed in Palma Salas *et al*. [2023] under the following premises:

- ReduceLROnPlateau[9] (callback function) was used to dynamically adjust the learning rate during training of the MobileNet model. A factor of 0.1 was used; it is reduced to one tenth of its current value. Patience is equal to 3, which implies that three epochs must pass without improvement to reduce the learning rate.
- Class weight estimation was used to address the issue of malware family imbalance. The weight for each class is computed by dividing the total number of samples by the product of the number of classes and the number of samples in each class. This technique assigns higher

[9]ReduceLROnPlateau improves training stability and adjusts model convergence, reducing training oscillations.

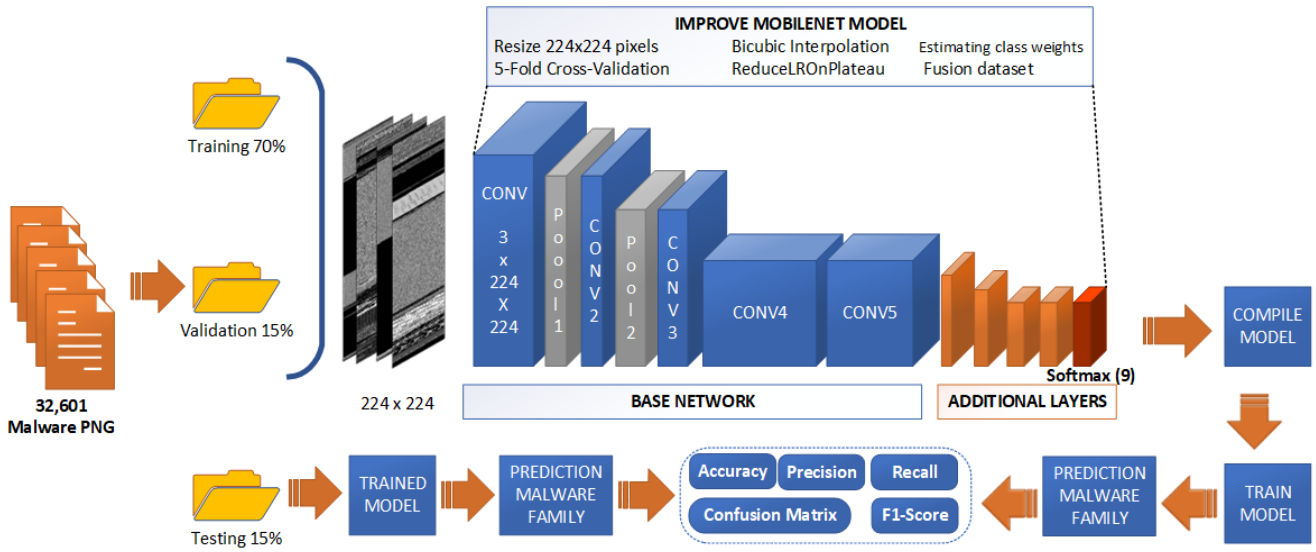

Figure 9. Implementation of transfer learning models for malware classification.

weights to less represented classes, enabling the classifier to pay more attention to them and thus enhance the overall performance of the classifier and fine-tune performance metrics.

The customized MobileNet Fine-Tuning (MobileNet FT) model is described below, as shown in Fig. 9:

- Initially, the pre-trained MobileNet models were loaded with the ImageNet model weights, excluding the top classification layer to add a new classification layer specific for each malware dataset.
- The convolutional layers of the pretrained model were frozen.
- The output of the pretrained model is taken, and a series of custom layers are applied to it to perform the specific malware dataset. The layers used include:
  - Flatten
  - Dense(512, ReLU activation)
  - BatchNormalization
  - Dropout
  - Dense(# of malware families, softmax) [10]
- For the compilation, we use the Adam optimizer with a low learning rate (0.0001) and a loss function in "categorical_crossentropy", as a shown in Table 5.
- For the training of the model, 50 epochs were sufficient per stage of training and validation because in most cases, the model stabilized at around 20 epochs and required 30 epochs to further improve its results.
- Finally, the model was evaluated using the test dataset through the performance metrics described in the subsection 4.3.

We also implemented a 5-fold cross-validation technique to evaluate both the variability of the data and the reliability of the MobileNet architecture, ensuring the robustness and generalization capability of the machine learning model.

Below, we present the results obtained through 5-fold cross-validation, which provide an average of the values for each malware dataset achieved by applying the MobileNet FT model.

5.3 MobileNet fine-tuning model Applied to Big 2015 Dataset

The model introduced in Section 4 surpasses other models outlined in Palma Salas *et al*. [2023], as illustrated in Table 6, achieving higher accuracy (98.86%), lower logloss (0.05966), and an improved F1-Score (98.86%). These results enhance the effectiveness of the fine-tuning method and the implemented adjustments.

The execution time (runtime) of approximately 2 hours and 56 minutes is also remarkably efficient compared to some other models that require more training time, such as Xception, which requires 9 hours and 42 minutes to reach 98.22% accuracy, or VGG19, which requires 38 hours and 10 minutes to get 98.10% accuracy. This is a consequence of the resizing and bicubic interpolation in the dataset before entering training.

This model has a high computational performance, which proves that it is not necessary to use complex architectures with many and full connect layers, such as VGG-16 and VGG-19, to obtain results close to 99%, with a shorter and more efficient execution time.

The comparison with the eight known CNN architectures used in Palma Salas *et al*. [2023] suggests that our MobileNet fine-tuning model is a lightweight architecture for environments with limited computational resources and shorter training time (Wall Time[11] training model) compared to the rest of the models used, as can be seen in Table 6.

As can be seen in Fig. 10, the model classifies almost perfectly. 5 of the 9 malware families were classified correctly (approximately 100%). It can be observed that MobileNet FT fails to classify the Simda Backdoor family correctly (71%)

[10] the output layer consists of a Dense layer with a number of units equal to the number of malware families in the dataset, with a softmax activation function applied to produce probability scores for each class.

[11] Wall Time is the total time taken to execute the entire model to 50 epochs, including the model training process and aspects of execution, such as data loading, model training, and evaluation time.

Table 6. Comparison of results of our proposal with research in Palma Salas *et al*. [2023] for malware families classification.

Models	Accur.	Logl.	F1-Sco.	Wall Time
MobN FT	**98.86%**	**0.060**	**98.86%**	**2h 56m**
MobileNet	98.41%	0.081	96%	3h 19m
Xception	98.22%	0.081	97%	9h 42m
MobNV2	98.22%	0.093	94%	3h 6m
VGG19	98.10%	0.078	95%	38h10m
DenseN169	97.98%	0.092	96%	9h 46m
VGG16	97.79%	0.086	95%	32h13m
InceptionV3	97.11%	0.153	94%	4h 15m
ResNet50	96.01%	0.153	94%	9h 17m

due to a lack of samples. This family presents an imbalance with only 42 samples.

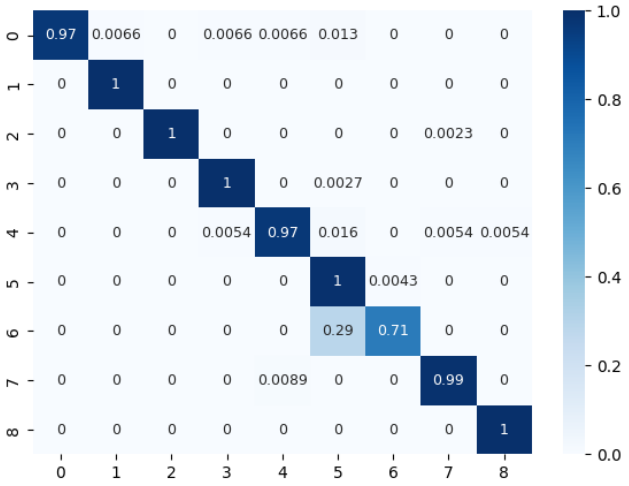

Figure 10. The confusion matrix for the Microsoft Big 2015 dataset obtained by the MobileNet FT model.

The results described in Table 7 show that the MobileNet FT model proposal has satisfactory results regarding accuracy (98.86%), precision (98.87%) and F1-Score (98.86%), as depicted in Fig. 11, showing perfect predictions of the image's family. Compared to the model developed in Palma Salas *et al*. [2023], this model shows improvement, being lighter and more effective in performance metrics when applied to an imbalanced dataset. By limiting the training to 50 epochs, we mitigate the risk of overfitting, as observed in other studies Rezende *et al*. [2017] (2000 epochs), Rezende *et al*. [2018] (100 epochs), which achieved excellent results but tended to overfit with prolonged training epochs.

As we can see in Table 7, the MobileNet FT model obtains the second highest accuracy (98.86%), surpassing almost all the other models listed in the table. Regarding the F1-Score, the model has reached 98.86%, obtaining a balance between precision and recall. The choice of the 5-fold cross-validation scheme contributes to the robustness of the results and the generalization of the model, performing a careful evaluation of the performance metrics. The comparison cannot be performed in runtime with other models because this information is not available in the cited papers.

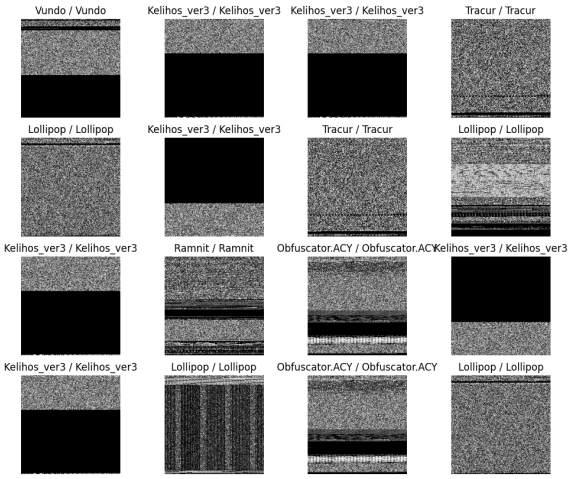

Figure 11. Malware image classification with MobileNet FT.

5.4 MobileNet fine-tuning model Applied to Malimg Dataset

The renowned Malimg benchmark dataset was used [Nataraj *et al*., 2011] applying the MobileNet FT model, achieving an accuracy of 99.07% in the test set, which indicates that the model classifies the 25 malware families and generalizes quite well the dataset. The log loss is 0.041 on the test set, demonstrating high confidence in its predictions.

The precision (99.06%), recall (99.07%), and F1-Score (99.06%) metrics are all greater than 99%, indicating good performance in terms of the model's ability to classify both positive and negative classes.

The confusion matrix, as shown in Fig. 12, describes that the model has perfect performance in some classes and very high overall. 21 of the 25 malware families were classified correctly (approximately 100%). MobileNet FT made errors in the classification of the Swizzor.gen!E (21%) and Swizzot.gen!I (30%) malware classes, as both are TDownloaders and some samples of Swizzor.gen!E were incorrectly labeled as Swizzot.gen!I and vice versa.

In comparison with other research described in Table 8 we highlight the accuracy (99.07%), surpassing all other models, highlighting its ability to handle the imbalance in the dataset and achieve high classification accuracy under an architecture that uses transfer learning with fine tuning and is efficient in training time with just 50 epochs.

5.5 MobileNet fine-tuning model Applied to MaleVis Dataset

The model demonstrates high accuracy and overall performance in classifying the 25 malware families in the MaleVis dataset [Bozkir *et al*., 2019]. Optimization included limiting it to 50 epochs to reduce overfitting and using techniques such as bicubic interpolation to improve model generalization and training speed. Through techniques such as fine-tuning and class weight estimation, the model has managed to handle the imbalance inherent in the dataset.

As can be seen in Fig. 13, the confusion matrix shown the performance of the model for each class in the test set. In the case of the Malevis dataset (balanced), only 11 of the 25 families were perfectly classified (approximately 100%). The

Table 7. Comparison of the proposed model in the literature using the Microsoft BIG 2015 dataset.

Authors	Models	Image For.	Validation	Acc	Pr	Rec	F1-Sc
[Kalash *et al*., 2018]	CNN (25 ep.)	Bytes(Gray)	90%-10%	98.99%	-	-	-
Proposed model	**MobileNet FT**	**Bytes (Gray)**	**5-fold CV**	**98.86%**	**98.87%**	**98.86%**	**98.86%**
[Hemalatha *et al*., 2021]	Dense (100 ep.)	Bytes (Gray)	70%-30%	98.46%	98.58%	97.84%	98.21%
[Palma Salas *et al*., 2023]	MobileNet	Bytes (RGB)	70%-15%-15%	98.41%	94.44%	98.25%	95.96%
[Gibert *et al*., 2019]	CNN	Bytes	5-fold CV	98.28%	-	-	96.36%
[Le *et al*., 2018]	CNN (100 ep.)	Bytes	5-fold CV	98.20%	-	-	96.05
[Gibert *et al*., 2019]	CNN	Bytes (Gray)	10-fold CV	97.50%	-	-	94.00%
[Wang *et al*., 2021]	DN+DEAM	Bytes (Gray)	60%-20%-20%	97.3%	95.3%	95.4%	95.4%

Table 8. Comparison of the proposed model in the literature using the Malimg dataset.

Authors	Models	Image For.	Validation	Acc	Pr	Rec	F1-Sc
Proposed model	**MobileNet FT**	**Bytes (Gray)**	**5-fold CV**	**99.07%**	**99.06%**	**99.07%**	**99.06%**
[Singh *et al*., 2019]	ResNet-50	Bytes (Gray)	-	98.98%	96%	96%	96%
[Roseline *et al*., 2020]	Layered Ens.	Bytes (Gray)	5-fold CV	98.65%	98.86%	98.63%	98.74%
[Verma *et al*., 2020]	1st&2do ord tex	Bytes (Gray)	10-fold CV	98.58%	98.04%	98.06%	98.05%
[Hemalatha *et al*., 2021]	DN (100 ep.)	Bytes (Gray)	70%-30%	98.5%	96.9%	96.9%	96.7%
[Gibert *et al*., 2019]	CNN	Bytes	5-fold CV	98.5%	95.8	96.6	95.8%
[Aslan and Yilmaz, 2021]	hybrid model	Bytes (Gray)	70%-10%-20%	97.78%	98.75%	97.02%	95.84%
[Shaik *et al*., 2023]	EfficientNetB0	Bytes (Gray)	-	-	97%	96%	96%

Sality (virus), Neshta (virus), and Expiro-H (virus) classes had the worst classification accuracy at 83%, 84%, and 87%, respectively. Although these values are not low, six malware families obtained values close to 90%, which reduced the model's accuracy to 96.02%.

The MobileNet fine-tuning-based model proves to be competitive in terms of accuracy (96.02%), precision (96.09%), recall (96.02%), and F1-score (96.04%). Although the model does not obtain the best expected results, having seen problems with three malware families, as shown in Fig. 13, the system manages to obtain good results using 5-fold cross-validation with a split of 70% for training, 15% for validation, and 15% for testing, avoiding presenting validation as test data, which is a very common error in these investigations that can be seen in Table 9.

The comparison with previous studies in Table 9 highlights the competitiveness of the MobileNet FT model in terms of accuracy and overall performance. These results support the effectiveness of the proposed approach in addressing the malware classification problem in the context of images.

5.6 MobileNet fine-tuning model Applied to Fusion Dataset

The creation of the Fusion dataset, which combines three benchmarking datasets (Microsoft Big 2015, Malimg, and MaleVis), contributes to research into the classification of malware families, achieving the following:

- Greater Diversity: Combining multiple datasets provides a broader diversity of malware samples. This is essential to improving the model's ability to generalize and recognize common patterns across a wide variety of threats.
- Increased Family Variety: By including 59 malware families, the Fusion dataset provides a more challenging and realistic dataset. This is crucial to evaluating the model's ability to handle a wide range of threats in real-world environments.
- Challenge of Class Imbalance: The variation in the number of samples per family introduces an additional challenge in terms of class imbalance. This scenario more accurately reflects malware distribution in reality and prepares the model for challenging situations.
- Reducing Dataset Bias: By combining datasets, the bias that could arise from using too few classes of malware is mitigated. This increases the robustness and confidence of the results obtained.

Results on the Fusion Dataset with MobileNet fine-tuning are described in Fig. 14. Our proposal achieved 98.04% accuracy with a logloss of 0.09280, reflecting that the system manages to identify the majority of malware families correctly, even with 59 classes.

The confusion matrix, as shown in Fig. 15, indicates that the model performs well in some classes and has a very high overall accuracy. 36 malware families achieved a 100% rating. For instance, the Simda family was correctly classified at 100% accuracy, which is an improvement compared to the 71% accuracy it obtained in the Big 2015 dataset. However, some families with erroneous classifications remained misclassified in the Fusion dataset, e.g., Expiro-H (Malevis) and Swizzor.gen!E (Malimg) had accuracies of 81% and 85% respectively. Additionally, Kelihos_ver1 (Big 2015) and Dontovo.A (Malimg) also faced classification issues with accuracies of 84% and 83% respectively.

The creation of the Fusion dataset, combining multiple datasets, has proven valuable in improving the ability of malware family classification models. This larger and more diverse set provides a more realistic and challenging setting to

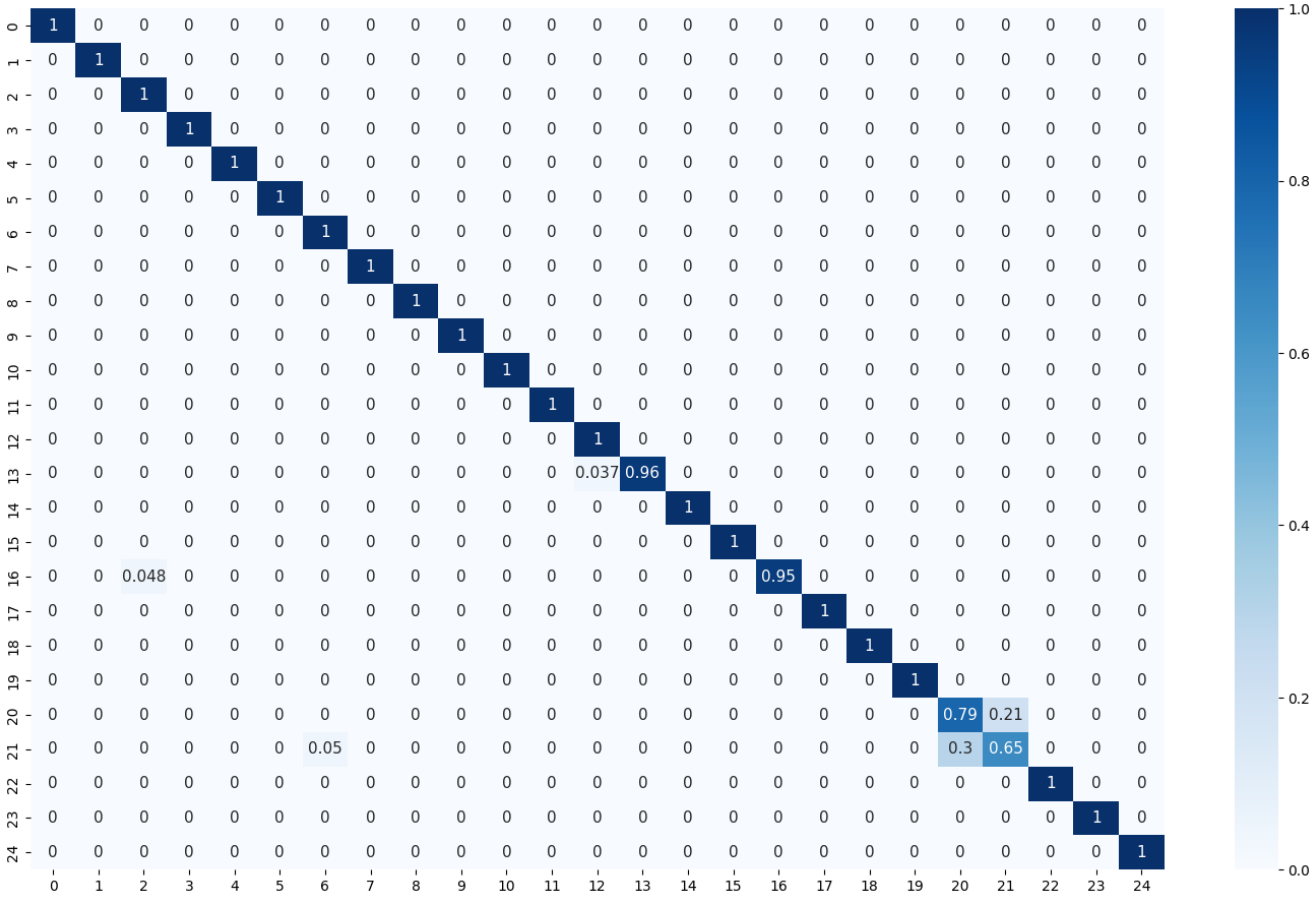

Figure 12. The confusion matrix for the Malimg dataset obtained by the MobileNet FT model.

Table 9. Comparison of the proposed model in the literature using the MaleVis dataset.

Authors	Models	Image For. t	Validation	Acc	Pr	Rec	F1-Sc
Proposed model	**MobN FT**	**Bytes (Gray)**	**5-fold CV**	**96.02%**	**96.09%**	**96.02%**	**96.04%**
[Bozkir *et al*., 2019]	DenseNet	Bytes (Gray)	tr:70%-tst:30%	97.48%	-	-	-
[Hemalatha *et al*., 2021]	DN (100 ep.)	Bytes (Gray)	tr:70%-tst:30%	98.21%	98.56%	97.74%	98.15%
[Roseline *et al*., 2020]	Layered Ens.	Bytes (Gray)	5-fold CV	97.43%	97.53%	97.32%	97.42%
[Aslan and Yilmaz, 2021]	hybrid model	Bytes (Gray)	70%-10%-20%	96.6%	97.1%	94.9%	94.5%
[Shaik *et al*., 2023]	Xception	Bytes (Gray)	-	-	95%	95%	95%

evaluate the effectiveness of the models.

The results describe in Table 10 are promising, with high levels of accuracy. The model's ability to handle a diverse and unbalanced set of malware families highlights the effectiveness of the Fusion dataset strategy and the fine-tuning technique employed.

The research presents significant advances in malware classification by addressing the issue of class imbalance and using a merged dataset that more realistically represents the real-world threat landscape.

Table 10. Results of the application of the MobileNet FT model to the Fusion dataset.

Metric	Value
Train Accuracy	99.978%
Train Log Loss	0.00158
Validation Accuracy	98.180%
Validation Log Loss	0.08960
Test Accuracy	98.037%
Test Log Loss	0.09280
precision	980807%
recall	980368%
f1-score	980390%

6 Effect on CNNs due to the Increase in Malware Families

In this section, we introduce the analysis of the decay in performance metrics due to the increase in malware families or classes. For this purpose, the Fusion dataset 4.2.4, composed of 59 malware families, was used as a basis, with a strategy of randomly adding malware families using the MobileNet FT architecture 5.2.

A new model was trained from scratch for each set of malware families (from 2 to 59) to identify the decay in

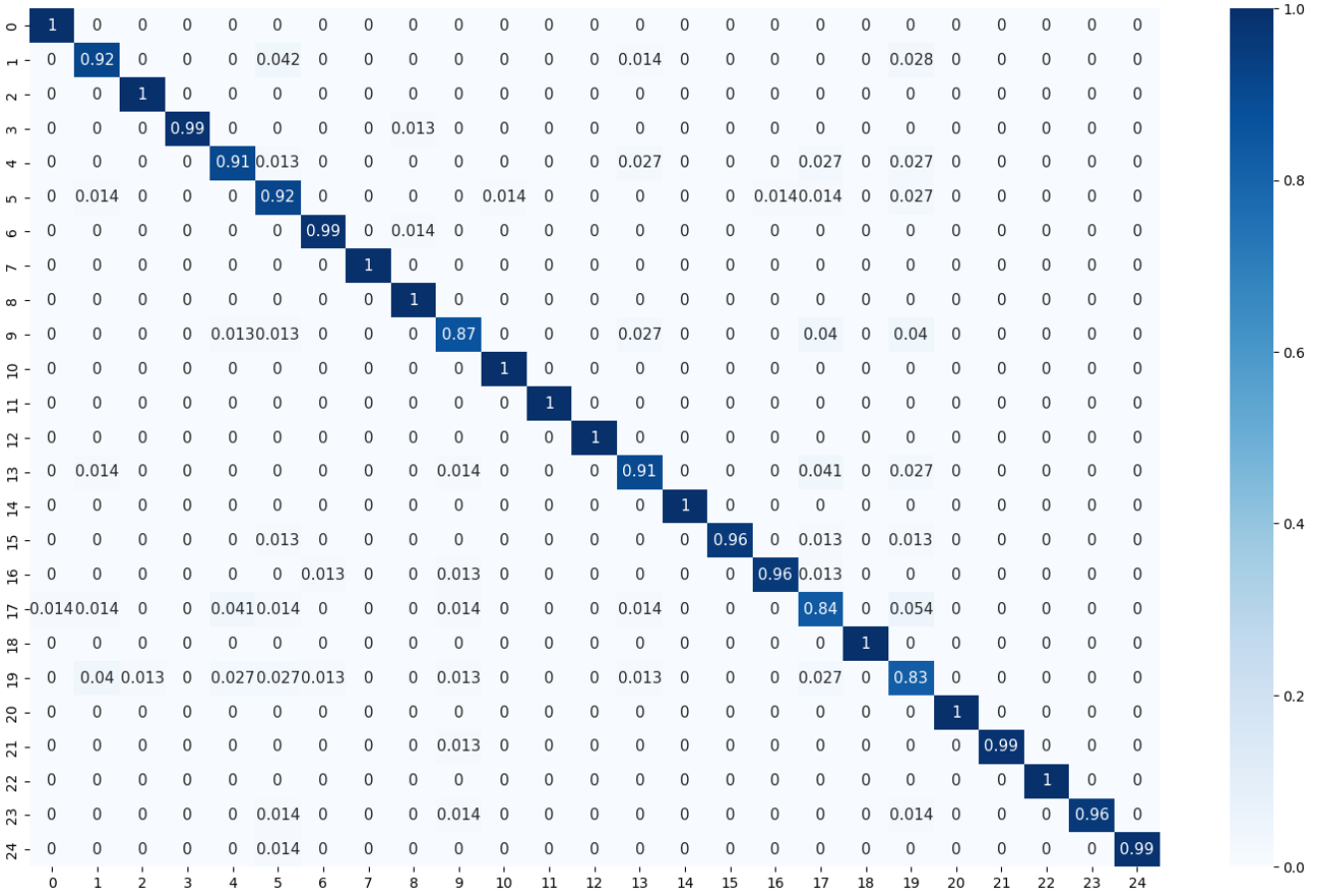

Figure 13. The confusion matrix for the Malevis dataset obtained by the MobileNet FT model.

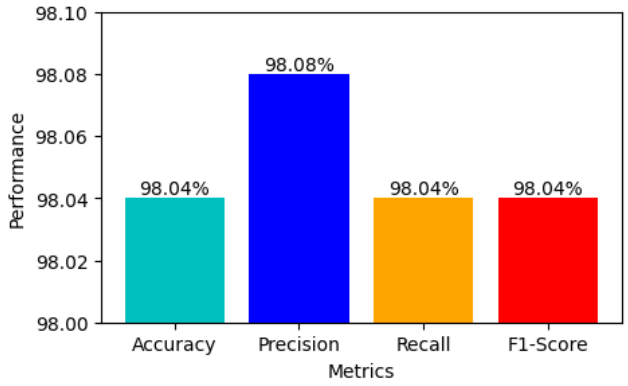

Figure 14. Performance Metrics obtained from Applied MobileNet FT to Fusion Dataset in test set.

performance metrics. This approach ensures that all malware classes are treated in a balanced manner from the beginning of training, avoiding biases towards previously trained classes.

The main premises of this procedure are described below:

- The MobileNet architecture serves as the foundation of the model used, with the last classification layer removed and the weights of the remaining layers frozen during training.
- The list of newly added layers is described in Section 5.2, with the modification that the number of classes in the final Dense layer was adjusted based on the number of malware families to be identified, using the softmax activation function.
- We chose to maintain 50 training epochs because the model generally stabilized around 20 epochs, and an additional 30 epochs were added to further refine the results.
- During training, the accuracy and log loss metrics were monitored in the training (70%) and validation (15%) sets.
- Finally, the trained model was evaluated using the test dataset (15%) to measure its performance.

This procedure was repeated incrementally from 2 to 59 malware families, allowing us to evaluate how the model adapted and maintained its performance as new classes were added.

Training from scratch enabled us to analyze the behavior of the MobileNet FT model from a known initial state and understand how it adapts to the increasing number of classes, providing a uniform basis for comparing performance.

6.1 Results of Performance Metrics

In this subsection, we delve into the performance metrics to understand how MobileNet responds to the incremental addition of malware families. We observed a gradual but continuous decline in accuracy and an opposite trend in log loss, indicating that as the number of classes increases, the model's performance deteriorates.

The behavior of accuracy and logarithmic loss is observed

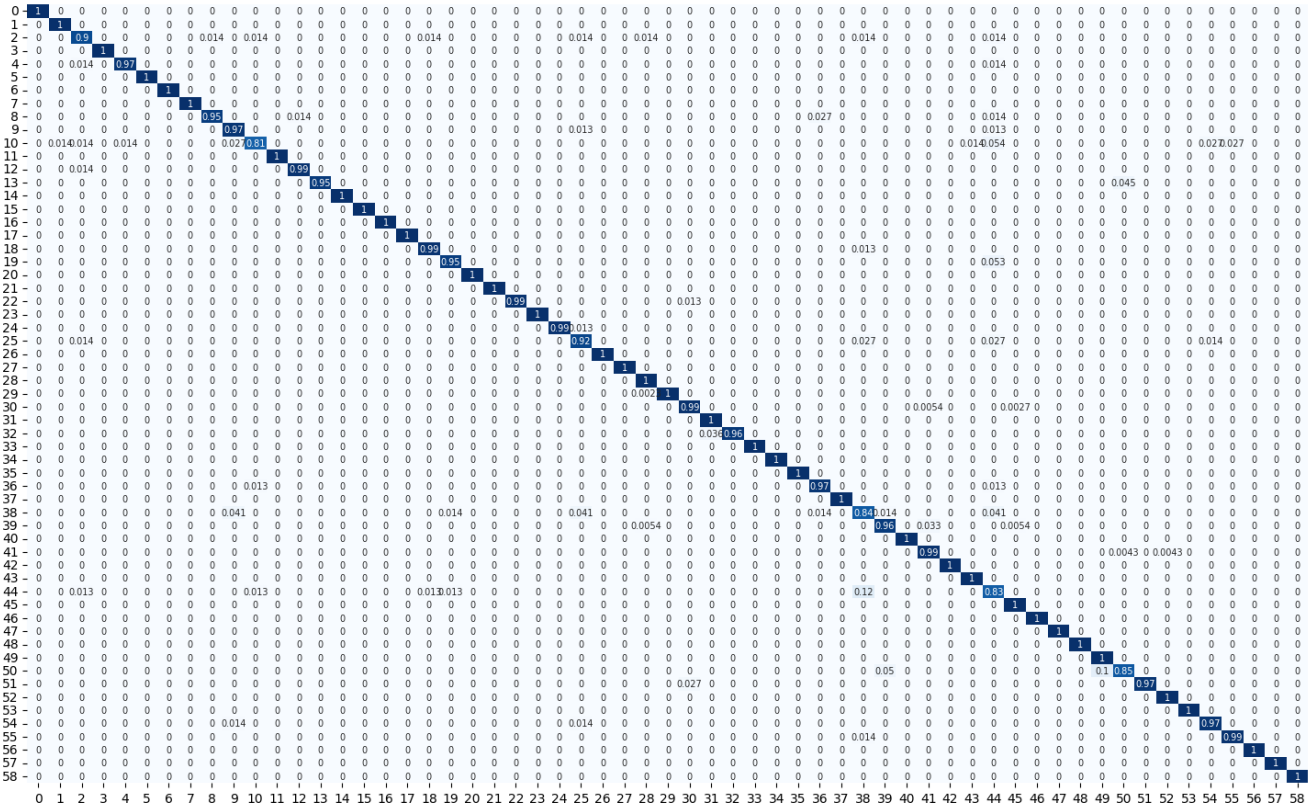

Figure 15. The confusion matrix for the Fusion dataset by the MobileNet FT model.

in Fig. 16 as a function of the increase in the number of families in the dataset in Fusion Dataset and the random entry of malware families. With the increase in malware families, it is noticeable that the Swizzor.gen!I (47), Swizzor.gen!E (48), and Lolyda.AA3 (49) families significantly increase the logloss. The greatest increase occurs when the number of samples begins to decrease between families with more than 500 samples (1–19) and families with fewer samples (20–59).

In general, performance metrics (accuracy and log loss) tend to fluctuate, as expected, remaining at high levels. However, we can observe that they have a tendency to reduce the performance of the metrics depending on the number of malware families to be classified.

This research allows us to identify that there is a decline in the performance metrics due to the increase in the number of malware families in the classification using the MobileNet FT model. Initially, these models manage to classify two classes perfectly, but as the number of families increases, they tend to lower levels, showing a downward trend in their performance.

This type of research can be carried out through the use of a dataset with a greater number of families, such as Fusion. Although this dataset is unbalanced, it attempts to reflect the difficulty of classifying malware in the real world. **The code used in this section can be accessed at** `https://github.com/ecram/malware_classification_cnn`.

7 Limitations

Although the results obtained in Tables 7, 8, and 9 demonstrate the ability of the MobileNet FT model to classify malware families, the present model may have limitations due to the complexity and diversity of the set of malware classes. This can be observed in the confusion matrices in Figures 10, 12, 13, where some specific characteristics of the malware families may not be well represented in the pretrained layers generating the decay of the performance metrics, as observed in Section 6 by the increase of malware families.

Although MobileNet FT may be effective in classifying the 59 known threats present in the Fusion dataset, it may not be as robust for new or unknown threats. If the model has not been trained with enough samples of new threats, its ability to recognize them may be limited.

MobileNet and other neural network-based models can be sensitive to variations in the input data, such as changes in image brightness, scale, or orientation. This can affect the model's ability to generalize to new samples that differ significantly from those in the training set. Thus, when using our proposal, it is recommended to apply resizing and bicubic interpolation to the size of 224x224 pixels to improve the results.

Regarding class imbalance, although a new dataset called Fusion was built, there may still be challenges related to the number of samples per class, i.e., classes with a large number of samples may not be adequately represented, which affects the model's ability to learn patterns specific to those classes.

On the other hand, the present research can benefit from the inclusion of additional experiments and validation on external datasets to evaluate the reproducibility and generaliz-

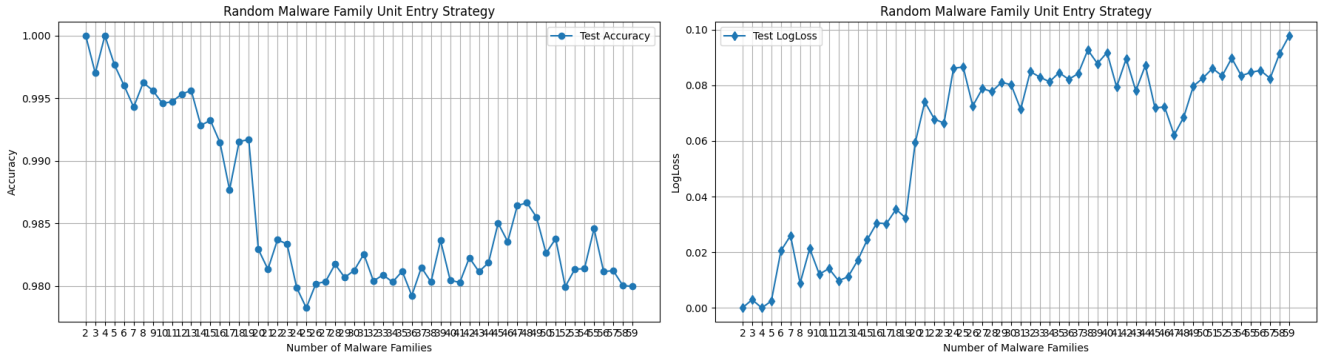

Figure 16. The behavior of accuracy and log loss is affected by the increase in malware families.

ability of the model to different real-world contexts and scenarios.

In turn, the model's ability to adapt and effectively classify future threats not observed in the current dataset may be a limitation. Research could explore strategies to improve model resilience to emerging threats.

8 Conclusions and Future Work

The MobileNet fine-tuning model has proven to be highly effective in classifying malware families on the Microsoft Big 2015 and Malimg datasets. The results show very good precision for the four datasets, low logarithmic loss, and evaluation metrics (precision, recall, F1-Score) greater than 99% in the case of the Malimg dataset. When comparing these results with other models in the literature, the proposal stands out in terms of performance. This success can be attributed to the optimization strategies, fine-tuning techniques, and data management considerations that have been implemented in the present research.

In summary, we have optimized the MobileNet fine-tuning model (MobileNet FT) model in the following aspects:

1. It has been limited to 50 epochs to reduce overfitting.
2. Resize 224x224 pixels and bicubic interpolation have been used to improve the generalization of the model and its training speed.
3. Fine-tuning techniques such as ReduceLROnPlateau (callback function) and Estimate class weights have been used to refine the results, avoiding local minimum.
4. The dataset has been divided through the scheme (70%-15%-15%) with 5-fold cross-validation for the validation of results.
5. We have analyzed the behavior of the model in the face of the increase in malware families, demonstrating the decline in performance metrics.
6. A new dataset called Fusion has been created by combining three known datasets (the Microsoft Big 2015, the Malimg, and the MaleVis), improving the generalization of our model against malware threats. It can be download at `https://www.kaggle.com/datasets/marcesalas/fusion-dataset-59-malware-families-in-png-\format`.

Based on these conclusions, we can propose the following future work with the objective of analyzing the performance of convolutional neural networks in the classification of malware in the wild through generalization and avoiding overfitting:

- Implement new models based on other architectures to obtain better results.
- Contribute to the analysis of information extraction from the process of converting bytes into images using other filters in order to improve performance metrics in malware detection and classification.
- Delve into the problem of the decay of performance metrics due to the increase in malware classes or families in order to determine the possible limits of this technique (CNN) in malware classification.

Acknowledgements

This article is an extension of the article Palma Salas *et al*. [2023] called Enhancing Malware Family Classification in the Microsoft Challenge Dataset via Transfer Learning selected among the best articles of the LADC 2023 main track.

References

Aslan, Ö. and Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms. *Ieee Access*, 9:87936–87951. DOI: 10.1109/ACCESS.2021.3089586.

AV-TEST GmbH (2023). AV-TEST Malware Statistics. Available at: `https://www.av-test.org/en/statistics/malware/` Accessed: May 20, 2023.

Bozkir, A. S., Cankaya, A. O., and Aydos, M. (2019). Utilization and comparision of convolutional neural networks in malware recognition. In *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4. IEEE. DOI: 10.1109/SIU.2019.8806511.

Fadnavis, S. (2014). Image interpolation techniques in digital image processing: an overview. *International Journal of Engineering Research and Applications*, 4(10):70–73. Available at: `https://ijera.com/papers/Vol4_issue10/Part%20-%201/K41007073.pdf`.

Gibert, D., Mateu, C., Planes, J., and Vicens, R. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques*, 15:15–28. DOI: 10.1007/s11416-018-0323-0.

Hemalatha, J., Roseline, S. A., Geetha, S., Kadry, S., and Damaševičius, R. (2021). An efficient Densenet-based deep learning model for malware detection. *Entropy*, 23(3):344. DOI: 10.3390/e23030344.

HobbyMaker (2023). Adding new pixels to a picture, an inexact comparison of several approaches to resampling. Available at: `http://www.hobbymaker.narod.ru/English/Articles/resampling_eng.htm` Accessed on 2023.11.15.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. DOI: 10.48550/arXiv.1704.04861.

Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., and Iqbal, F. (2018). Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*, pages 1–5. IEEE. DOI: 10.1109/NTMS.2018.8328749.

Keras (2023). keras._legacy preprocessing image imagedatagenerator. Available at: `https://github.com/keras-team/keras/blob/master/keras/legacy/preprocessing/image.py` Accessed on 2023.11.15.

Le, Q., Boydell, O., Mac Namee, B., and Scanlon, M. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. *Digital Investigation*, 26:S118–S126. DOI: 10.1016/j.diin.2018.04.024.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444. DOI: 10.1038/nature14539.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. DOI: 10.1109/5.726791.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*, pages 1–7. DOI: 10.1145/2016904.2016908.

Palma Salas, M. I., De Geus, P., and Botacin, M. (2023). Enhancing malware family classification in the Microsoft challenge dataset via transfer learning. In *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, LADC '23, page 156–163, New York, NY, USA. Association for Computing Machinery. DOI: 10.1145/3615366.3615374.

Patterson, J. and Gibson, A. (2017). *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.". Book.

Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., and De Geus, P. (2017). Malicious software classification using transfer learning of Resnet-50 deep neural network. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1011–1014. IEEE. DOI: 10.1109/ICMLA.2017.00-19.

Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F., and Geus, P. d. (2018). Malicious software classification using VGG16 deep neural network's bottleneck features. In *Information Technology-New Generations: 15th International Conference on Information Technology*, pages 51–59. Springer. DOI: 10.1007/978-3-319-77028-4$_9$.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M. (2018). Microsoft malware classification challenge. *arXiv preprint arXiv:1802.10135*. DOI: 10.48550/arXiv.1802.10135.

Roseline, S. A., Geetha, S., Kadry, S., and Nam, Y. (2020). Intelligent vision-based malware detection and classification using deep random forest paradigm. *IEEE Access*, 8:206303–206324. DOI: 10.1109/ACCESS.2020.3036491.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al*. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252. DOI: 10.1007/s11263-015-0816-y.

Shaik, A., Pendharkar, G., Kumar, S., Balaji, S., *et al*. (2023). Comparative analysis of imbalanced malware byteplot image classification using transfer learning. *arXiv preprint arXiv:2310.02742*. DOI: 10.48550/arXiv.2310.02742.

Singh, A., Handa, A., Kumar, N., and Shukla, S. K. (2019). Malware classification using image representation. In *Cyber Security Cryptography and Machine Learning: Third International Symposium, CSCML 2019, Beer-Sheva, Israel, June 27–28, 2019, Proceedings 3*, pages 75–92. Springer. DOI: 10.1007/978-3-030-20951-3$_6$.

Srudeep, P. (2020). An overview on MobileNet: An efficient mobile vision CNN. Available at: `https://medium.com/@godeep48/an-overview-on-mobilenet-an-efficient-mobile-vision-cnn-f301141db94d` Accessed on 2023.11.15.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852. DOI: 10.1109/ICCV.2017.97.

Talebi, H. and Milanfar, P. (2021). Learning to resize images for computer vision tasks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 497–506. DOI: 10.1109/ICCV48922.2021.00055.

Verma, V., Muttoo, S. K., and Singh, V. (2020). Multiclass malware classification via first-and second-order texture statistics. *Computers & Security*, 97:101895. DOI: 10.1016/j.cose.2020.101895.

Wang, C., Zhao, Z., Wang, F., and Li, Q. (2021). A novel malware detection and family classification scheme for IoT based on DEAM and DenseNet. *Security and Communication Networks*, 2021:1–16. DOI: 10.1155/2021/6658842.