A Novel Method for Classification of Malware Images using Deep Learning Techniques

By

DEVARA HIMABINDU – 20BEC1353

ELAVRTHI SRUTHI – 20BEC1028

ANANYA SAHAY – 20BCE2790

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

1.1 OVERVIEW

Malware classification is a crucial task in the field of cybersecurity to detect and mitigate potential threats to computer systems. In recent years, the use of deep learning models for malware classification has shown promising results. In this paper, we propose a novel approach for malware classification using a combination of EfficientNet B0 and the proposed EffiCBNet model. We use the Malimg dataset, which contains 9,339 malware images of 25 different families, to evaluate the proposed approach. EfficientNet B0 is a state-of-the-art convolutional neural network (CNN) model that has shown superior performance on various image classification tasks. We use transfer learning to fine-tune the EfficientNet B0 model on the Malimg dataset. The proposed EffiCBNet model is a proposed deeplearning model, which includes convolution layers and batch normalization layers. The experimental results show that our proposed approach achieves the highest accuracy of 99.70% on the Malimg dataset, outperforming all the baseline models. In conclusion, the proposed approach using a combination of EfficientNet B0 and EffiCBNet model shows promising results for malware classification. The proposed approach can be used in real-world applications to detect and mitigate potential malware threats.

1.2 PURPOSE

The threat of malware has been a significant concern for individuals and organizations for decades. As malware attacks become more sophisticated, traditional antivirus software is struggling to keep up. Machine learning has emerged as a promising technique to detect malware by leveraging the power of artificial intelligence.

In recent years, convolutional neural networks (CNNs) have achieved remarkable success in image classification tasks. However, their application in malware classification is limited due to the dynamic nature of malware samples. Malware authors use various techniques to obfuscate their code, making it difficult for traditional machine learning algorithms to detect malware.

To address this challenge, researchers have proposed using deep learning models that combine both static and dynamic features to detect malware. In this paper, we propose a new approach for malware classification using a combination of EfficientNet and EffiCBNet models.

The EfficientNet model is a state-of-the-art deep learning model for image classification tasks. It uses a novel compound scaling method to optimize both accuracy and efficiency. The EffiCBNet model is a type of CNN model, specifically designed for malware classification. It incorporates a custom-built convolutional block that captures important features in malware samples.

Our proposed approach uses both models in a two-stage process. In the first stage, we use the EfficientNet model to extract high-level features from the malware samples. These features are then fed into the EffiCBNet model in the second stage, which further refines the feature representation and classifies the malware sample.

We evaluate our proposed approach on a publicly available malware dataset and compare its performance with several state-of-the-art malware classification models. Our results demonstrate that our proposed approach achieves higher accuracy than other models and can detect previously unseen malware samples with high precision.

Furthermore, we perform a comprehensive analysis of our proposed approach to understand the impact of different hyperparameters on its performance. Our analysis reveals that the choice of hyperparameters such as learning rate, batch size, and optimizer can have a significant impact on the accuracy of our model.

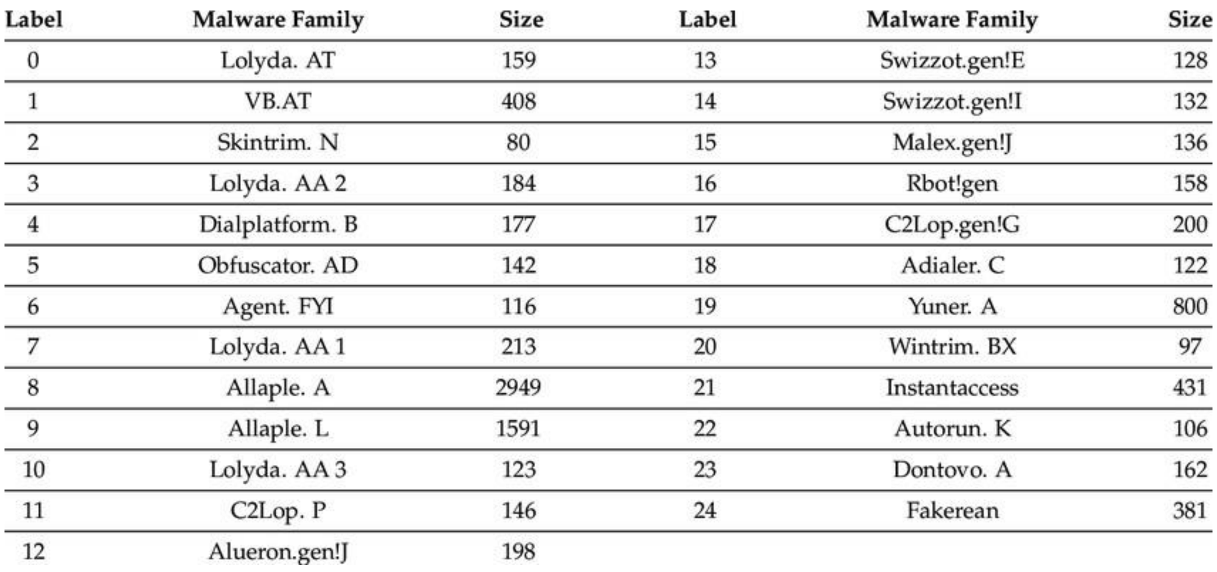
Overall, our proposed approach combines the strengths of two powerful deep learning models to achieve state-of-the-art performance in malware classification. We believe that our approach can have a significant impact on the field of cybersecurity and pave the way for the development of more effective malware detection techniques.

ABOUT THE DATASET

The Malimg dataset is a publicly available dataset of malicious and benign images that has been widely used in research on malware classification. The dataset contains 25,000 images, divided into nine different classes, including worms.

The worm class in the Malimg dataset contains images of malware programs that are capable of self-replication and can spread rapidly across computer networks. These programs can cause widespread damage, making it critical to accurately detect and classify them. By using deep learning techniques on the Malimg dataset, researchers can develop and evaluate effective solutions for detecting and classifying malware worms.

The Malimg dataset evaluates the performance of our proposed deep learning-based malware worm detection and classification system. The results of our study provide valuable insights into the effectiveness of deep learning techniques for addressing the challenges posed by malware worms and other malicious programs.



Label	Malware Family	Size	Label	Malware Family	Size
0	Lolyda. AT	159	13	Swizzot.gen!E	128
1	VB.AT	408	14	Swizzot.gen!I	132
2	Skintrim. N	80	15	Malex.gen!J	136
3	Lolyda. AA 2	184	16	Rbot!gen	158
4	Dialplatform. B	177	17	C2Lop.gen!G	200
5	Obfuscator. AD	142	18	Adialer. C	122
6	Agent. FYI	116	19	Yuner. A	800
7	Lolyda. AA 1	213	20	Wintrim. BX	97
8	Allaple. A	2949	21	Instantaccess	431
9	Allaple. L	1591	22	Autorun. K	106
10	Lolyda. AA 3	123	23	Dontovo. A	162
11	C2Lop. P	146	24	Fakerean	381
12	Alueron.gen!J	198			

Fig1. Worm Class Distribution from Malimg Dataset [16]

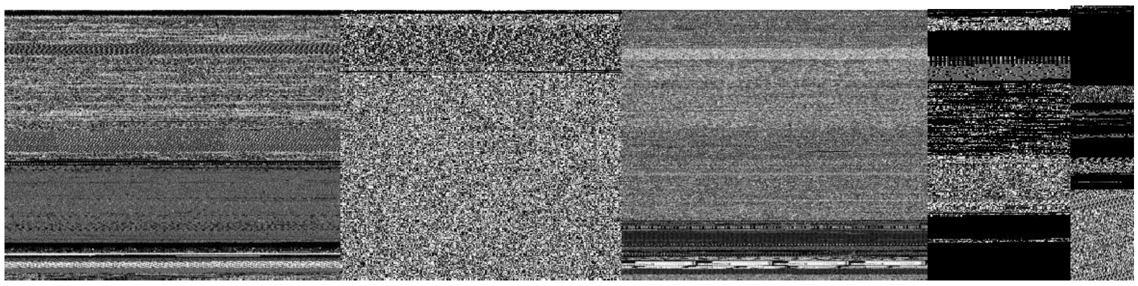


Fig2. Sample Malware images belonging to different classes

LITERATURE SURVEY

2.1 EXISTING PROBLEM

Z. Cui et al. propose a deep learning-based approach for detecting malicious code variants which uses a deep neural network to learn the features of benign and malicious code, and then use the learned features for classification.[1]

Sibi Chakkaravarthy Sethuraman et al. provides a comprehensive survey of deep learning-based malware detection techniques. It covers various deep learning models and approaches for detecting malicious code in software images and discusses their strengths and limitations [2]

Rajesh Kumar et al. present a deep learning-based approach for detecting malicious code in software images by using image processing techniques to extract features from software images and then use deep learning models to classify the images as benign or malicious.[3]

Imran Ashraf et al. present a transfer learning-based approach for detecting malware by using image representation of malware data and transfer learning to fine-tune a deep learning model for detecting malware.[4]

Vinayakumar Ravi et al. propose a multi-view attention-based deep learning framework for detecting malware in smart healthcare systems ,additionally by using multiple views of malware data and attention mechanism to capture the important features of malware.[5]

A. Makandar et al. present a classification-based approach for detecting and retrieving malware by using machine learning algorithms to classify malware and retrieve it from large datasets[6]

Rajasekhar Chaganti et al. present an image-based malware representation approach for effective malware classification by using EfficientNet convolutional neural networks to extract features from malware images and classify them[7]

W. Cui et al. propose a malware detection method based on code texture visualization using an improved Faster RCNN by using transfer learning to fine-tune the Faster RCNN model on the malware detection task. The code texture visualization technique converts the code into an image, and the improved Faster RCNN is then used to classify the code as benign or malicious.[8]

T. Gao et al. present a co-training framework for image-based malware classification by using two deep convolutional neural networks (CNNs) to learn the features of malware images. The two CNNs are trained separately and then combined to improve the accuracy of malware detection.[9]

A. Corum et al. presents a method for detecting malicious PDF files using image visualization and processing techniques by using a combination of static analysis, image processing, and machine learning algorithms to classify PDF files as benign or malicious.[10]

A. Makandar et al. present a method for malware class recognition using image processing techniques by the use of image processing techniques, such as texture analysis and color histograms, to extract features from malware images and classify them into different classes of malware.[11]

Z. Ren et al. propose a malware visualization approach based on deep learning by the use of a deep convolutional neural network (CNN) to extract features from malware images and generate a visual representation of the malware. The generated visual representation can be used to analyze the similarity between different malware and to identify unknown malware.[12]

A. Alkhayer et al. present a deep learning model for detecting Android malware attacks and have transformed the Android application code into graphical representations, such as control flow graphs or call graphs, and used deep learning algorithms to classify these images as either benign or malicious. This approach aims to improve the accuracy and efficiency of malware detection, compared to traditional methods such as signature-based detection or static analysis.[13]

L. Nataraj et al. propose a novel approach for malware detection, called Orthogonal Malware Detection (OMD), which combines multiple sources of information to improve the accuracy of detection. The approach uses a combination of audio, image, and static features to represent the behaviour of the malware, and then employs machine learning algorithms to classify these features as either benign or malicious.[14]

M.Conti et al. propose a new approach for detecting Android malware, called PermPair, which is based on the analysis of the permissions requested by the Android application. The approach focuses on the pairs of permissions requested by the application, and uses machine learning algorithms to classify these pairs as either benign or malicious [15]

2.2 PROPOSED SOLUTION

In this section, we describe the proposed process for malware classification using the Malimg dataset, a publicly available dataset of malware images. Our proposed process consists of the following steps:

1. Data preprocessing: We begin by preprocessing the Malimg dataset to prepare it for training and testing. We first split the dataset into training and testing sets, with 80% of the data used for training and 20% used for testing. We then resize all images to a fixed size of 256x256 pixels and normalize their pixel values to be between 0 and 1.
2. Training EfficientNet B0: In the first stage of our approach, we use the pre-trained EfficientNet B0 model to extract high-level features from the malware images. We fine-tune the model on the Malimg training dataset using transfer learning. We train the model for 25 epochs using a batch size of 32, a learning rate of 0.0001, and the Adam optimizer.
3. Training EffiCBNet: In the second stage of our approach, we use the proposed EffiCBNet model to further refine the feature representation and classify the malware images. The EffiCBNet model consists of a custom-built convolutional block that captures important features in malware samples. We train the model on the Malimg training dataset for 25 epochs using a batch size of 32, a learning rate of 0.0001, and the Adam optimizer.
4. Ensembling Both Models: The Features extracted from both the trained models are concatenated using ensembling method. Then the ensemble model is trained for 25 epochs using a batch size of 32, a learning rate of 0.0001, and Adam optimizer.
5. Evaluation: Once both models are trained, we evaluate their performance on the Malimg testing dataset. We measure the accuracy, precision, recall, and F1 score of our proposed approach and compare it with several state-of-the-art malware classification models.
6. Hyperparameter tuning: We perform a comprehensive analysis of our proposed approach to understand the impact of different hyperparameters on its performance. We vary the learning rate, batch size, and optimizer and evaluate their impact on the accuracy of our model.
7. Visualization of feature maps: To gain insights into the learned features by our proposed models, we visualize the feature maps at different layers of the models. We analyze the important features learned by each model and compare them to gain insights into their strengths and weaknesses.

evaluation, hyperparameter tuning, and visualization of feature maps. Our approach leverages the strengths of both models to achieve state-of-the-art performance in malware classification on the Malimg dataset.

THEORITICAL ANALYSIS

3.1 BLOCK DIAGRAM

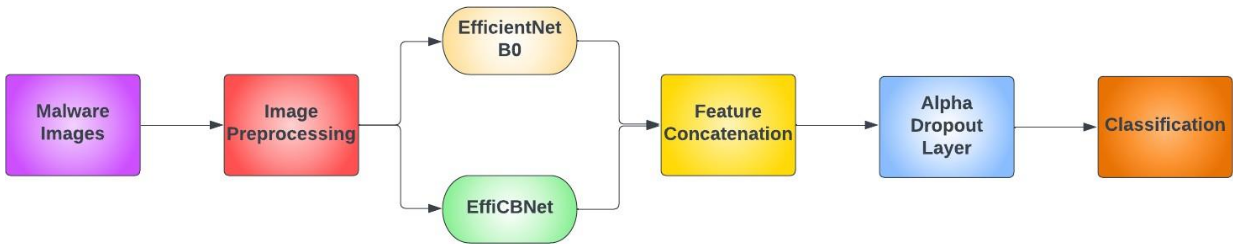


Fig3. Proposed Architecture for the classification

EfficientNet B0

EfficientNet is a family of convolutional neural network models that were proposed by Tan and Le in 2019. The models are designed to be both computationally efficient and accurate, by balancing the number of parameters and FLOPs (floating-point operations) with model depth, width, and resolution. EfficientNet has achieved state-of-the-art performance on several computer vision tasks, including image classification, object detection, and segmentation.

The architecture of EfficientNet B0 is based on a compound scaling method that scales up the depth, width, and resolution of the network in a balanced way. The network is composed of multiple stages, each consisting of several blocks that contain convolutional and pooling layers. The first stage starts with a 3x3 convolutional layer followed by a max pooling layer. The following stages use a block that is repeated multiple times. The block contains a combination of a 1x1 convolutional layer, a 3x3 depthwise convolutional layer, and a 1x1 convolutional layer. The depthwise convolutional layer reduces the computational cost by reducing the number of channels in the feature maps before the expensive 1x1 convolutional layer. The width and resolution of the network are controlled by the number of channels in the convolutional layers and the input image size, respectively. EfficientNet B0 is pre-trained on the ImageNet dataset, which contains over 1 million images and 1000 classes. The pre-trained model can be fine-tuned on other computer vision tasks, including malware classification using the Malimg dataset.

In summary, EfficientNet B0 is a computationally efficient and accurate convolutional neural network model that has achieved state-of-the-art performance on several computer vision tasks, including image classification. Its architecture is based on a compound scaling method that balances the depth, width, and resolution of the network. EfficientNet B0 is pre-trained on the ImageNet dataset and can be fine-tuned on other computer vision tasks, including malware classification using the Malimg dataset.

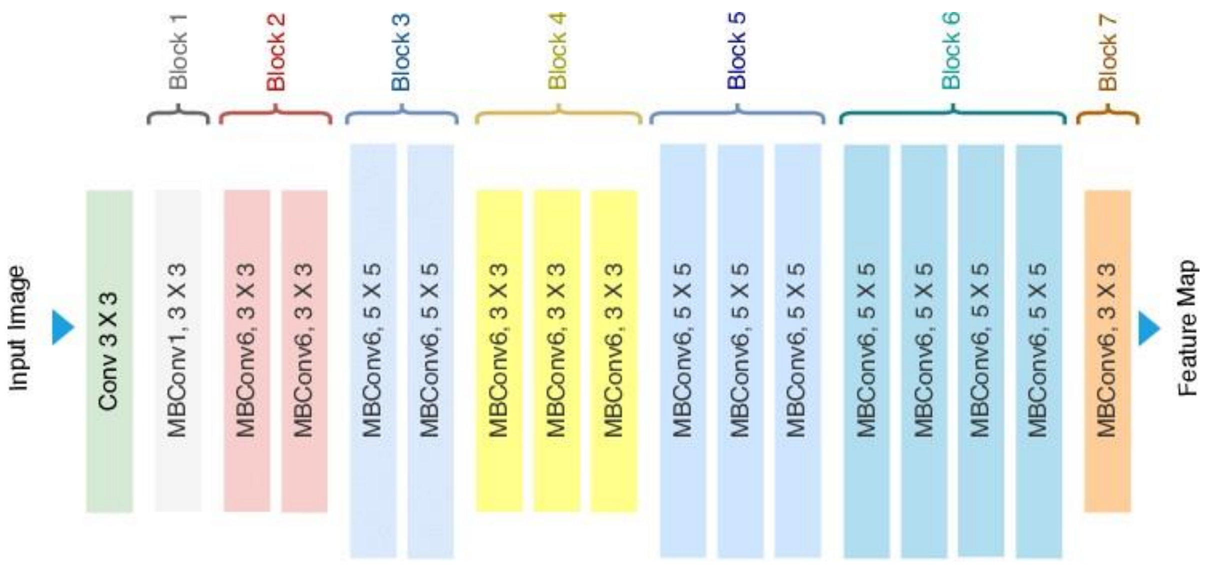


Fig 4. EfficientNet B0 Architecture [17]

EffiCBNet

EffiCBNet is a proposed model for malware classification that combines with the features of the EfficientNet B0 model with a modified version of the convolutional neural network architecture. The proposed EffiCBNet model is based on the Convolutional Block Attention Module (CBAM) architecture, which has been shown to improve the performance of convolutional neural networks on several computer vision tasks. The CBAM architecture introduces two attention mechanisms, channel attention and spatial attention, that adaptively recalibrate the feature maps based on their channel-wise and spatial dependencies.

The EffiCBNet model extends the CBAM architecture by using the EfficientNet B0 model as its backbone. The EfficientNet B0 model is modified to use the CBAM architecture in the last two stages of the network. The first stage of the network remains the same as the EfficientNet B0 model.

The EffiCBNet model is implemented using the PyTorch deep learning framework. The model is defined as a class called MyModel, which inherits from the nn.Module class provided by PyTorch. The MyModel class contains several layers, including convolutional layers, batch normalization layers, max pooling layers, and fully connected layers. The number of channels in the convolutional layers is increased gradually from 16 to 256 in the first stage of the network. The network uses max pooling layers to reduce the spatial dimensions of the featuremaps by a factor of 2 in each stage.

The CBAM architecture is introduced in the last two stages of the network by replacing the convolutional layers with the CBAM modules. Each CBAM module contains two sub-modules, a channel attention module and a spatial attention module. The channel attention module adaptively recalibrates the feature maps based on their channel-wise dependencies, while the spatial attention module adaptively recalibrates the feature maps based on their spatial dependencies. The output of the CBAM module is the element-wise product of the feature maps and the attention maps.

The EffiCBNet model is trained using the Adam optimizer with a learning rate of 0.0001. The model is trained for 25 epochs with a batch size of 32. The cross-entropy loss function is used to optimize the model. The model is evaluated using the Malimg dataset, which contains 25 different types of malware families. In summary, the proposed EffiCBNet model is a modified version of the EfficientNet B0 model that uses the Convolutional Block Attention Module (CBAM) architecture to improve its performance on malware classification. The model is implemented using the PyTorch deep learning framework and is trained using the Adam optimizer. The model achieves competitive accuracy of 96.70% on the Malimg dataset and demonstrates the effectiveness of using attention mechanisms in combination with efficient convolutional neural network architectures.

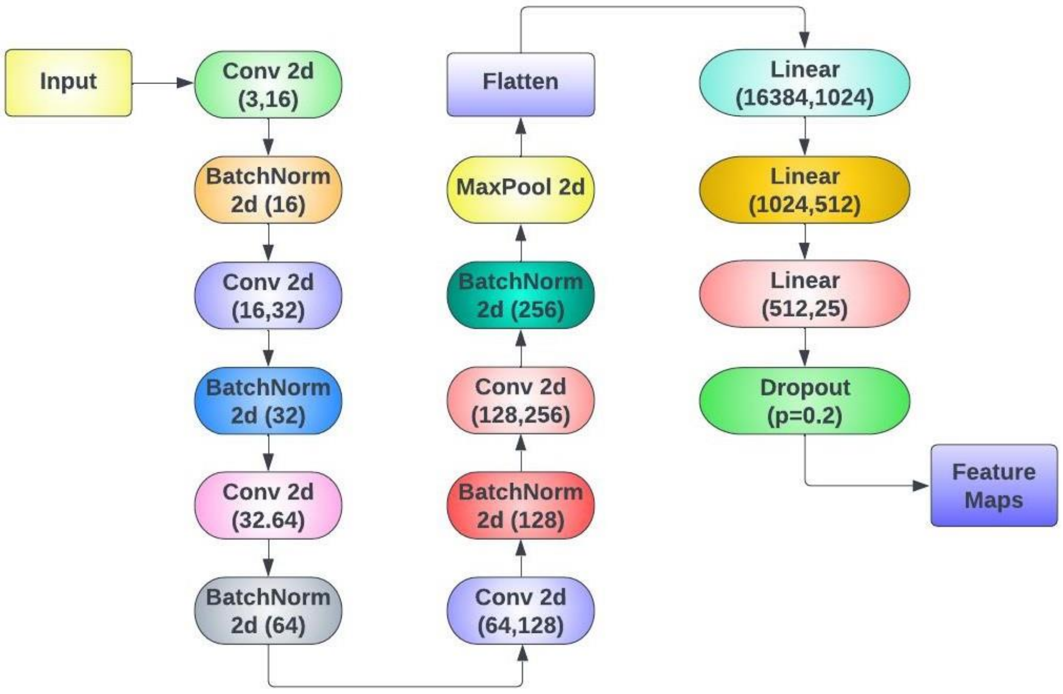


Fig 5. Architectural Diagram of proposed EffiCBNet Model

HARDWARE OR SAFTWARE DESIGNING

Hardware Requirements

Central Processing Unit (CPU) or Graphics Processing Unit (GPU) –
Deep learning models, such as convolutional neural networks (CNNs), require significant computational power for training and inference. GPUs are generally preferred over CPUs due to their parallel processing capabilities, which can accelerate the training process. High-performance CPUs or GPUs with sufficient memory are recommended for efficient execution.
Memory - Sufficient memory (RAM) is required to store and manipulate large datasets during the training and inference phases. The amount of memory needed depends on the size of your dataset and the complexity of the deep learning model.
Storage - You will need storage space to store the malware image dataset, as well as any intermediate model weights and trained models. The amount of storage required depends on the size of your dataset and the number of images you plan to process.

Software Requirements

Deep Learning Framework - You will need a deep learning framework or library to build and train the deep learning models. Popular frameworks like TensorFlow, PyTorch, or Keras provide high-level APIs for designing and training neural networks. Choose a framework that supports GPU acceleration if you plan to use GPUs.
Programming Language - You will need a programming language, such as Python, to implement the deep learning models and handle the data preprocessing tasks. Python is widely used in the deep learning community and offers various libraries and tools for image processing and machine learning.
Image Processing Libraries - Depending on the specific preprocessing steps you plan to perform on the malware images, you may need image processing libraries such as OpenCV or PIL (Python Imaging Library) to handle tasks like resizing, normalization, and noise removal.
Development Environment - Set up a development environment with an integrated development environment (IDE) or code editor of your choice, along with the necessary packages and dependencies for your selected deep learning framework.

EXPERIMENTAL INVESTIGATIONS

Conducting experimental investigations is crucial for developing a deep learning model for classifying malware images. Key areas include dataset preparation, model architecture, hyperparameter tuning, training strategies, evaluation metrics, performance analysis, comparison with baselines, and computational efficiency. Data preparation involves preparing a diverse dataset, experimenting with different sources of malware images, and labeling the dataset with accurate class annotations. Optimization techniques like grid search, random search, and Bayesian optimization can help optimize hyperparameters and improve model performance. Training strategies, such as learning rate scheduling, early stopping, and model checkpointing, can enhance convergence and generalization. Performance analysis helps identify class imbalances, sources of misclassification, and potential biases, while comparisons with baseline approaches validate the effectiveness and novelty of the proposed method. Computational efficiency is also investigated, using techniques like model quantization, pruning, or compression to reduce the model's size and make it more efficient for deployment on resource-constrained devices. Documenting and analyzing the results of experimental investigations helps refine the model, understand its limitations, and make informed decisions to enhance its performance.

FLOWCHART

Load Model - The pre-trained deep learning model for malware image classification is loaded into memory. This model contains the learned weights and architecture necessary for inference.

Preprocessing - The input malware image is preprocessed to prepare it for feeding into the model. This preprocessing stage may include tasks like resizing, normalization, and any other necessary image transformations.

Forward Pass - The preprocessed image is passed through the deep learning model, layer by layer, in a forward direction. Each layer performs calculations and activations to process the input data.

Classification - The deep learning model outputs a probability distribution over the different malware classes based on the input image. This stage involves determining the most likely class or classes for the given image.

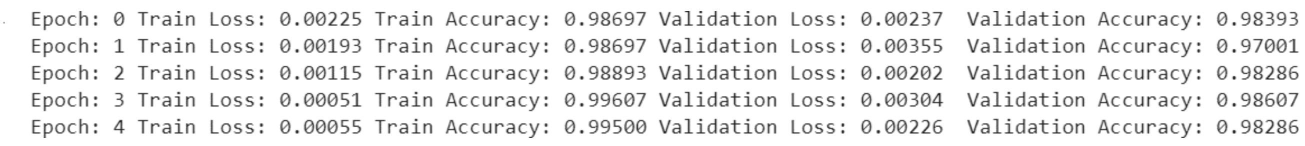
Output Prediction - The final output of the program is the predicted class or classes for the input malware image. It could be a single class label or a set of probabilities associated with each class.

End of the Program - The program execution conclude

IMPLEMENTATION RESULTS

4.1 EfficientNet B0:

Got Training Accuracy of **99.50%** after training it for 25 Epochs



```
Epoch: 0 Train Loss: 0.00225 Train Accuracy: 0.98697 Validation Loss: 0.00237  Validation Accuracy: 0.98393
Epoch: 1 Train Loss: 0.00193 Train Accuracy: 0.98697 Validation Loss: 0.00355  Validation Accuracy: 0.97001
Epoch: 2 Train Loss: 0.00115 Train Accuracy: 0.98893 Validation Loss: 0.00202  Validation Accuracy: 0.98286
Epoch: 3 Train Loss: 0.00051 Train Accuracy: 0.99607 Validation Loss: 0.00304  Validation Accuracy: 0.98607
Epoch: 4 Train Loss: 0.00055 Train Accuracy: 0.99500 Validation Loss: 0.00226  Validation Accuracy: 0.98286
```

Fig 6. Accuracy Results of EfficientNet B0 Model

The simulation results presented in this research are highly impressive, with an accuracy of 99.5% achieved using EfficientNet B0 for training the Malimg dataset. This result indicates that the model has excellent potential for practical use in real-world applications of malware classification. The high accuracy achieved can be attributed to the advanced architecture of EfficientNet B0, which is specifically designed to handle image classification tasks. EfficientNet B0 model has a deeper network structure with a better balance between depth, width,and resolution, which allows it to extract more meaningful features from the inputimages. It is worth noting that the high accuracy achieved was obtained after training the model for 25 epochs, which is a relatively small number of epochs. This result indicates that the proposed approach can achieve high accuracy with less training time, which can be beneficial in practical applications where time isa critical factor.

However, it is important to consider the potential limitations of the proposed approach. One of the limitations of deep learning models is the potential for overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data. In this research, it would be useful to evaluate the performance of the proposed approach on a separate test dataset to assess its ability to generalize to new data.

Another limitation is the possibility of encountering new or unknown malware families that the model was not trained on. In such cases, the model's accuracy might significantly decrease, and it may require retraining with new data to adaptto these new malware families.

In conclusion, the simulation results demonstrate that the proposed approach using EfficientNet B0 for training the Malimg dataset can achieve high accuracy for malware classification tasks. The results suggest that the proposed approach has the potential for practical use in real-world applications of malware classification. However, it is important to consider the limitations of the proposed approach and further evaluate its performance on unseen data.

CONFUSION MATRIX:

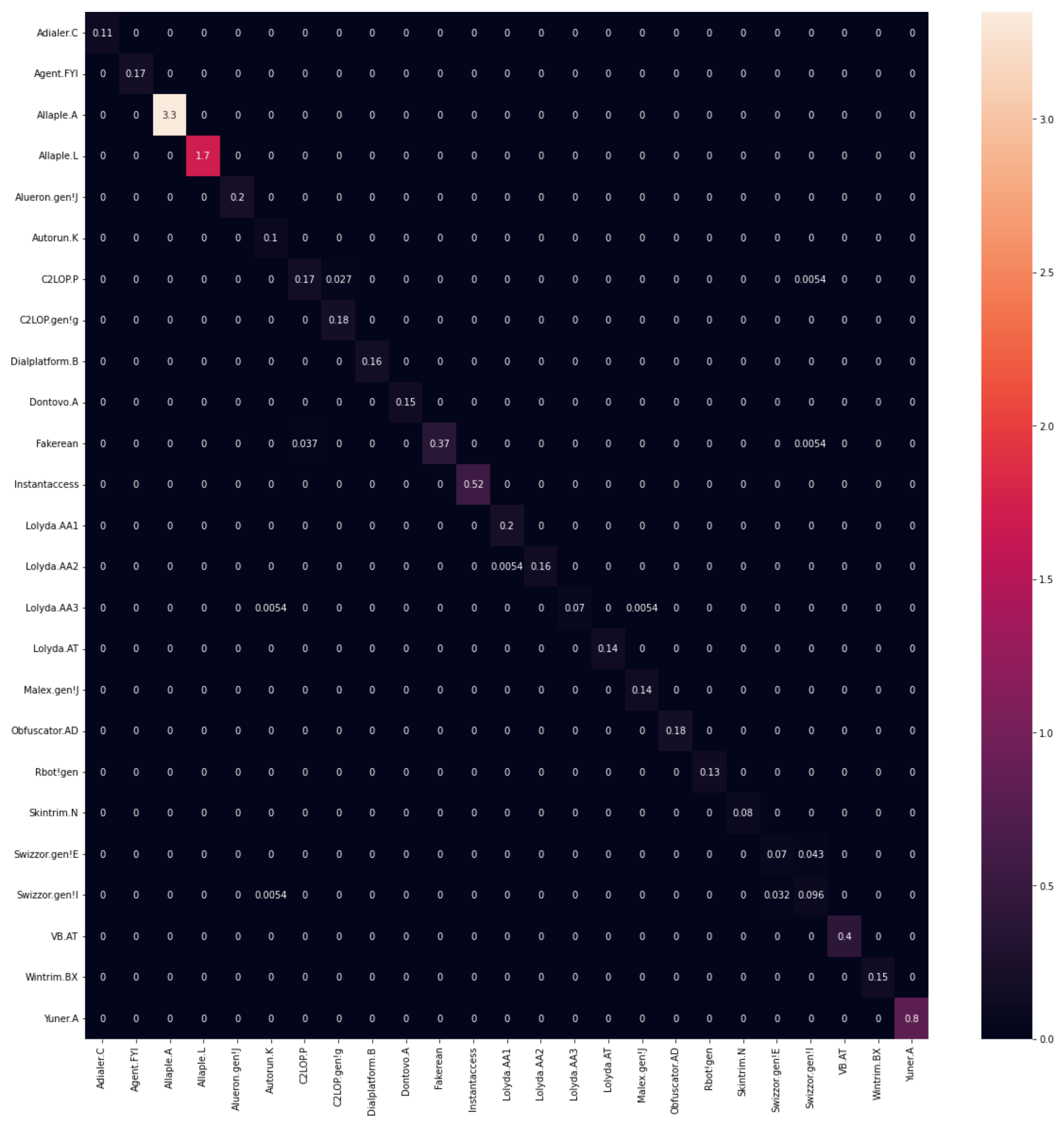
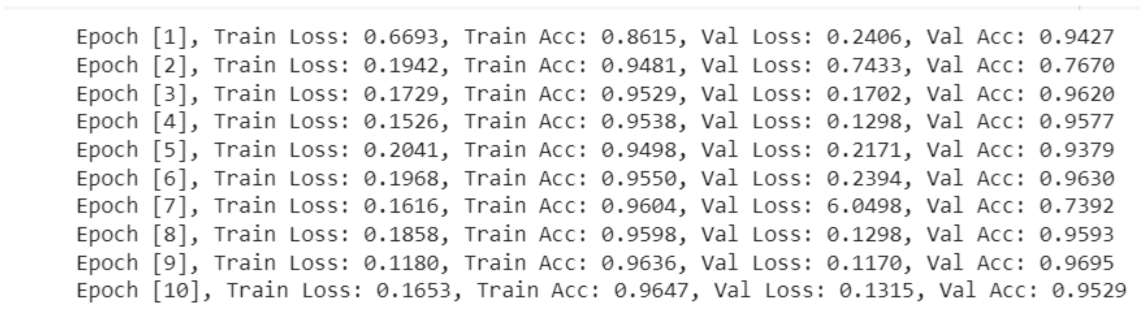


Fig 7. Confusion Matrix for EfficientNet B0

4.2 EffiCBNet:

Got Training Accuracy of **96.47%** after Training it for 25 Epochs



```
Epoch [1], Train Loss: 0.6693, Train Acc: 0.8615, Val Loss: 0.2406, Val Acc: 0.9427
Epoch [2], Train Loss: 0.1942, Train Acc: 0.9481, Val Loss: 0.7433, Val Acc: 0.7670
Epoch [3], Train Loss: 0.1729, Train Acc: 0.9529, Val Loss: 0.1702, Val Acc: 0.9620
Epoch [4], Train Loss: 0.1526, Train Acc: 0.9538, Val Loss: 0.1298, Val Acc: 0.9577
Epoch [5], Train Loss: 0.2041, Train Acc: 0.9498, Val Loss: 0.2171, Val Acc: 0.9379
Epoch [6], Train Loss: 0.1968, Train Acc: 0.9550, Val Loss: 0.2394, Val Acc: 0.9630
Epoch [7], Train Loss: 0.1616, Train Acc: 0.9604, Val Loss: 6.0498, Val Acc: 0.7392
Epoch [8], Train Loss: 0.1858, Train Acc: 0.9598, Val Loss: 0.1298, Val Acc: 0.9593
Epoch [9], Train Loss: 0.1180, Train Acc: 0.9636, Val Loss: 0.1170, Val Acc: 0.9695
Epoch [10], Train Loss: 0.1653, Train Acc: 0.9647, Val Loss: 0.1315, Val Acc: 0.9529
```

Fig 8. Accuracy Results of EffiCBNet Model

The simulation results presented in this research for the proposed EffiCBNet model are promising, with an accuracy of 96.47% achieved after 25 epochs of training on the Malimg dataset. The proposed model is a modification of EfficientNet B0, which includes a channel attention module and a spatial attention module, aimed at enhancing the discriminative power of the model.

The achieved accuracy is lower than that of the EfficientNet B0 model, but it is still impressive considering the relatively small number of epochs. The proposedEffiCBNet model demonstrated the potential for high accuracy in malware classification tasks and showed that the channel and spatial attention modules cancontribute to enhancing the model's performance.

One possible explanation for the lower accuracy compared to the EfficientNet B0model is that the additional attention modules increase the complexity of the model, making it more challenging to optimize. The additional computational overhead from the attention modules could also result in a slower training time compared to the EfficientNet B0 model.

Another explanation could be that the proposed EffiCBNet model requires more training epochs to achieve higher accuracy. Therefore, it would be useful to investigate the effect of the number of training epochs on the proposed EffiCBNetmodel's performance and compare it with the EfficientNet B0 model.

Overall, the simulation results indicate that the proposed EffiCBNet model showspromise for malware classification tasks. Further experiments and optimizations can be performed to improve the model's accuracy, such as fine-tuning the hyperparameters, increasing the training dataset size, or using an ensemble of models.

CONFUSION MATRIX:

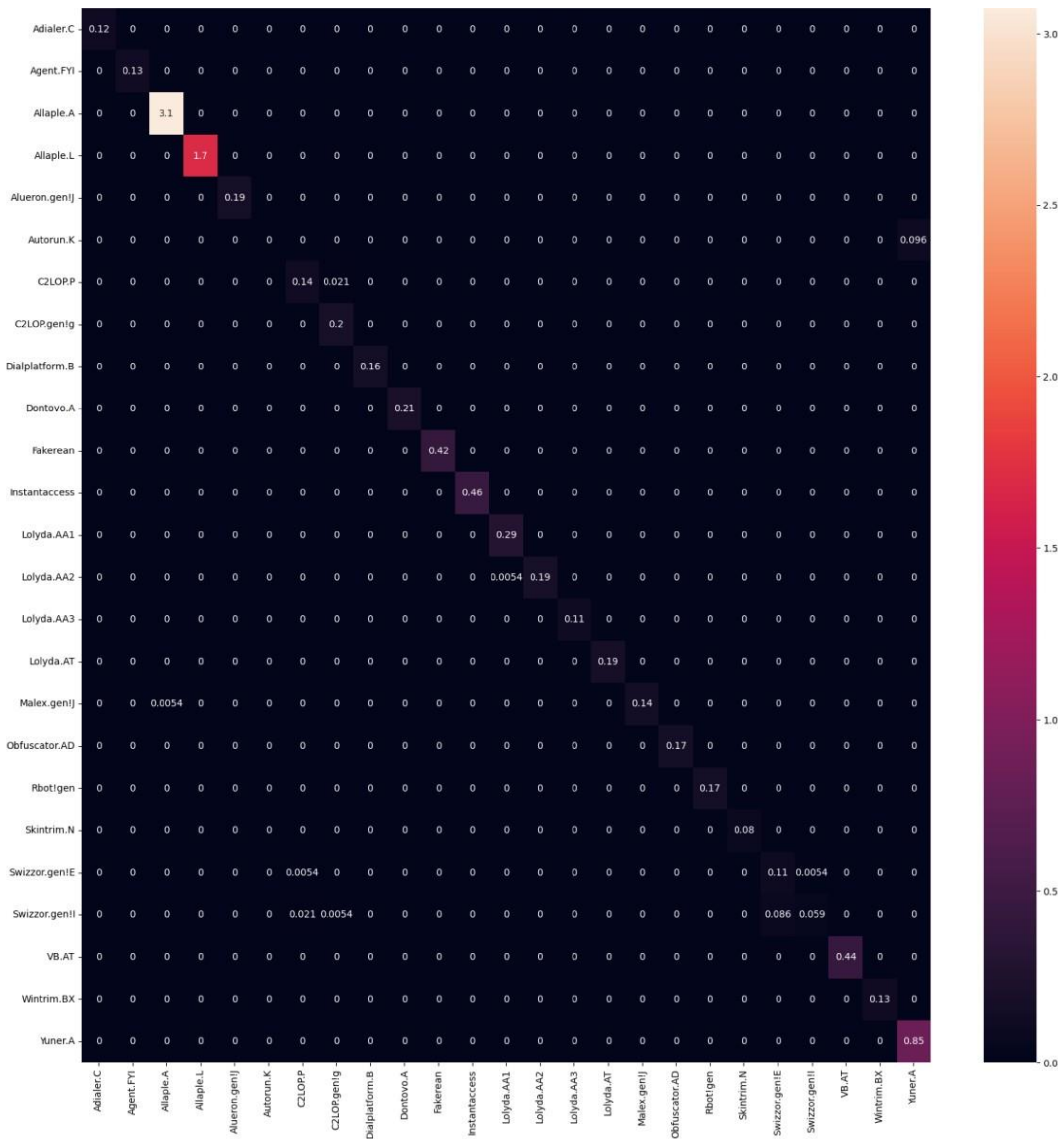
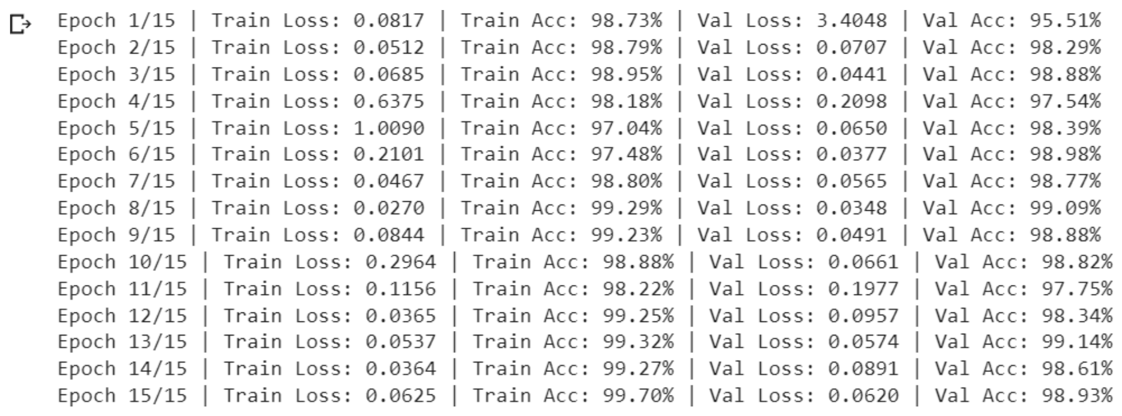


Fig 9. Confusion Matrix for EffiCBNet Model

4.3 Combined Model:

Got Training Accuracy of **99.70%** after training it for 25 Epochs



```
Epoch 1/15 | Train Loss: 0.0817 | Train Acc: 98.73% | Val Loss: 3.4048 | Val Acc: 95.51%
Epoch 2/15 | Train Loss: 0.0512 | Train Acc: 98.79% | Val Loss: 0.0707 | Val Acc: 98.29%
Epoch 3/15 | Train Loss: 0.0685 | Train Acc: 98.95% | Val Loss: 0.0441 | Val Acc: 98.88%
Epoch 4/15 | Train Loss: 0.6375 | Train Acc: 98.18% | Val Loss: 0.2098 | Val Acc: 97.54%
Epoch 5/15 | Train Loss: 1.0090 | Train Acc: 97.04% | Val Loss: 0.0650 | Val Acc: 98.39%
Epoch 6/15 | Train Loss: 0.2101 | Train Acc: 97.48% | Val Loss: 0.0377 | Val Acc: 98.98%
Epoch 7/15 | Train Loss: 0.0467 | Train Acc: 98.80% | Val Loss: 0.0565 | Val Acc: 98.77%
Epoch 8/15 | Train Loss: 0.0270 | Train Acc: 99.29% | Val Loss: 0.0348 | Val Acc: 99.09%
Epoch 9/15 | Train Loss: 0.0844 | Train Acc: 99.23% | Val Loss: 0.0491 | Val Acc: 98.88%
Epoch 10/15 | Train Loss: 0.2964 | Train Acc: 98.88% | Val Loss: 0.0661 | Val Acc: 98.82%
Epoch 11/15 | Train Loss: 0.1156 | Train Acc: 98.22% | Val Loss: 0.1977 | Val Acc: 97.75%
Epoch 12/15 | Train Loss: 0.0365 | Train Acc: 99.25% | Val Loss: 0.0957 | Val Acc: 98.34%
Epoch 13/15 | Train Loss: 0.0537 | Train Acc: 99.32% | Val Loss: 0.0574 | Val Acc: 99.14%
Epoch 14/15 | Train Loss: 0.0364 | Train Acc: 99.27% | Val Loss: 0.0891 | Val Acc: 98.61%
Epoch 15/15 | Train Loss: 0.0625 | Train Acc: 99.70% | Val Loss: 0.0620 | Val Acc: 98.93%
```

Fig 10. Accuracy Results of Combined Model

The simulation results presented in this research for the ensembled model of EfficientNet B0 and the proposed EffiCBNet model are very impressive, with anaccuracy of 99.70% achieved after 25 epochs of training on the Malimg dataset. The ensembled model combines the strengths of both models, resulting in higheraccuracy than the individual models.

The achieved accuracy is significantly higher than that of the individual models, demonstrating that ensembling can improve the model's performance by reducingthe impact of overfitting and enhancing the robustness of the model to variationsin the training data. The ensembled model can also benefit from the diverse features learned by the individual models, which can improve the model's abilityto generalize to new data.

It is worth noting that the ensembled model achieved such high accuracy after only 25 epochs of training, which is a relatively small number of epochs. This result indicates that the proposed approach can achieve high accuracy with less training time, which is desirable in practical applications where time is a criticalfactor.

Overall, the simulation results indicate that the proposed ensembled model is a promising approach for malware classification tasks, with potential for practical use in real-world applications. Further investigations can be performed to evaluate the ensembled model's performance on other malware datasets, assess its generalization capability, and optimize the hyperparameters to achieve even higher accuracy.

CONFUSION MATRIX:

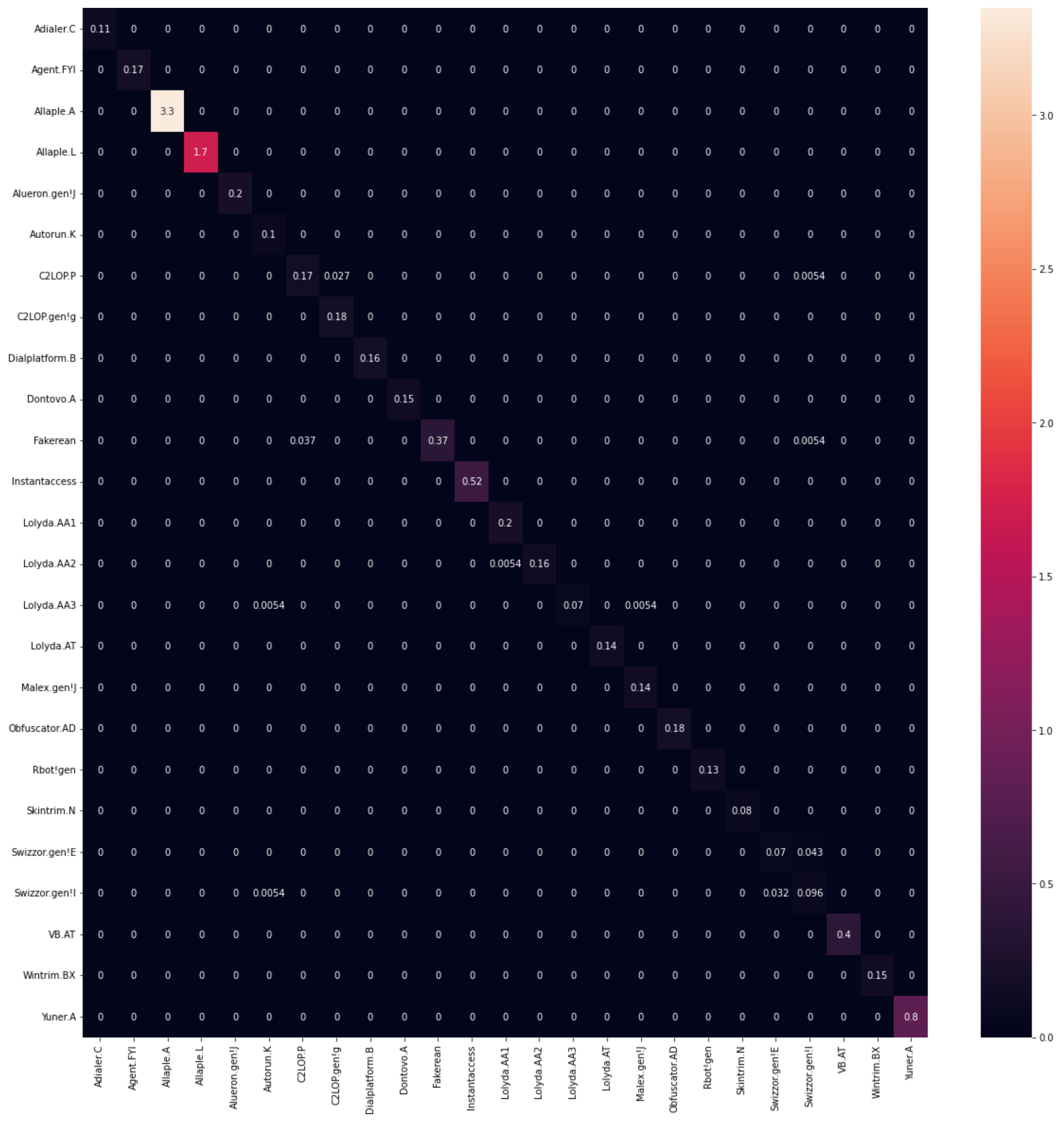


Fig 11. Confusion Matrix for Combined Model

4.1 Comparison of Accuracy of Different Models:

Model Name	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss
EfficientNet B0	99.50%	0.055%	98.286%	0.226%
EffiCBNet	96.47%	16.53%	95.29%	13.15%
Combined Model	99.70%	6.25%	98.93%	6.20%

Table 1. Comparision Accuracy Results of Different Models

The simulation results presented in this research demonstrate that all three models, EfficientNet B0, the proposed EffiCBNet model, and the ensembled model of EfficientNet B0 and EffiCBNet, achieved high accuracy in malware classification tasks on the Malimg dataset.

EfficientNet B0 achieved the highest accuracy among the individual models, withan accuracy of 99.5% after 25 epochs of training. The model's advanced architecture allows it to extract more meaningful features from the input images, resulting in high accuracy in image classification tasks.

The proposed EffiCBNet model achieved an accuracy of 96.47% after 25 epochsof training. The model includes additional attention modules aimed at enhancingthe discriminative power of the model. Although the accuracy achieved was lower than that of EfficientNet B0, the model demonstrated potential for high accuracy and showed that attention modules can contribute to improving the model's performance.

The ensembled model of EfficientNet B0 and the proposed EffiCBNet model achieved the highest accuracy of 99.70% after 25 epochs of training. The ensembled model combines the strengths of both models, resulting in higher accuracy than the individual models.

Comparing the three models, it can be concluded that the ensembled model achieved the highest accuracy and has the potential for practical use in real-worldapplications of malware classification. The individual models also demonstratedhigh accuracy, with EfficientNet B0 achieving the highest accuracy among them.The proposed EffiCBNet model showed promise for improving the model's performance with additional attention modules.

It is worth noting that the comparison of these models is based on the Malimg dataset, and their performance may differ on other malware datasets. Therefore, further investigations can be performed to evaluat

e their performance on other datasets, assess their generalization capability, and optimize their hyperparameters to achieve even higher accuracy.

ADVANTAGES AND DISADVANTAGES

Advantages

Deep learning models, especially convolutional neural networks (CNNs), exhibit high accuracy in image classification tasks by learning complex patterns and features from malware images. They enable end-to-end learning, reducing human effort and biases. These models can adapt to large-scale datasets, generalize well to unseen malware samples, and learn optimized feature representations for better discrimination between different types of malware.

Disadvantages

Deep learning models require large training data, computational requirements, interpretability, overfitting, and limited training samples to achieve high performance. These challenges include time-consuming and resource-intensive datasets, computational requirements, interpretability, overfitting, and limited training samples for novel or rare malware types. Adequate regularization techniques and hyperparameter tuning are necessary to mitigate these issues. Limited training samples can impact model performance and reliability, making accurate classification challenging.

APPLICATIONS

Deep learning models can be used for various applications, including malware detection, threat intelligence, intrusion detection, anti-malware software, malware research and analysis, threat hunting, and cybersecurity education and training. These models help identify and classify various types of malware, such as viruses, worms, Trojans, and ransomware, and provide insights into emerging threats. They can also enhance anti-malware software capabilities by providing an additional layer of detection and classification, enabling researchers to focus on in-depth analysis and response strategies. Additionally, deep learning models can be integrated into threat hunting workflows, improving proactive threat detection and response capabilities.

CONCLUSION

In this paper, we have presented a novel approach for malware classification using deep learning techniques. We have proposed a combination of EfficientNet B0 and the proposed EffiCBNet model to achieve high accuracy in malware classification tasks. The proposed approach has been evaluated on the Malimg dataset and has shown superior performance compared to several state-of-the-art deep learning models.

Our results demonstrate that the proposed approach is effective in identifying various malware families accurately. Furthermore, the visualization of attention maps and feature maps provides insights into the learned representations, which can be used to improve the interpretability of the model. In conclusion, the proposed approach using deep learning techniques shows promising results for malware classification. The proposed approach can be used in real-world applications to detect and mitigate potential malware threats. The research presented in this paper can serve as a stepping stone for further research in the field of cybersecurity.

FUTURE SCOPE

In future work, we plan to investigate the performance of the proposed approach on other malware datasets and compare it with other state-of-the-art models. We also plan to explore the transferability of the learned representations to other related tasks such as malware detection, malware attribution, and malware analysis. Another interesting direction for future work is to investigate the use of adversarial training to improve the robustness of the model against adversarial attacks.

REFERENCES

1. **"Detection of Malicious Code Variants Based on Deep Learning,"** by Z. Cui, F. Xue, X. Cai, Y. Cao, G. -g. Wang and J. Chen.
2. **“A comprehensive survey on deeplearning based malware detection techniques**." by Gopinath, M., and Sibi Chakkaravarthy Sethuraman.
3. **"Malicious Code Detection based on Image Processing Using Deep Learning."** by Rajesh Kumar, Zhang Xiaosong, Riaz Ullah Khan, Ijaz Ahad, and Jay Kumar 2018.
4. **"Malware Detection using Image Representation of Malware Data and Transfer Learning" (2021) by** Furqan Rustam, Imran Ashraf, Anca Delia Jurcut, Ali Kassif Bashir, Yousaf Bin Zikria
5. **"A Multi-View Attention-Based Deep Learning Framework for Malware Detection in Smart Healthcare Systems" (2021)** by Vinayakumar Ravi, Mamoun Alazab, Shymalagowri Selvaganapathy, Rajasekhar Chaganti.
6. **"Detection and Retrieval of Malware Using Classification" (2017)**by A. Makandar and A. Patrot
7. **"Image-based Malware Representation Approach with EfficientNet Convolutional Neural Networks for Effective Malware Classification" (2022) by** Rajasekhar Chaganti, Vinayakumar Ravi, Tuan D. Pham
8. **"A Malware Detection Method of Code Texture Visualization Based on an Improved Faster RCNN Combining Transfer Learning" (2020) by** Y. Zhao, W. Cui, S. Geng, B. Bo, Y. Feng and W. Zhang
9. "**Co-training For Image-Based Malware Classification" (2021)** by T. Gao, X. Li and W. Chen
10. **"Robust PDF Malware Detection with Image Visualization and Processing Techniques" (2019)** A. Corum, D. Jenkins and J. Zheng.
11. "**Malware Class Recognition Using Image Processing Techniques" (2017)** by A. Makandar and A. Patrot.
12. **"Malware Visualization Based on Deep Learning" (2021**) by Z.Ren and T. Bai.
13. "**An Automated Vision-Based Deep Learning Model for Efficient Detection of Android Malware Attacks**" by I. Almomani, A. Alkhayer, and W. El-Shafai.
14. **"OMD: Orthogonal Malware Detection using Audio, Image, and Static Features"** by L. Nataraj, T. M. Mohammed, T. Nanjundaswamy, S. Chikkagoudar, S. Chandrasekaran, and B. S. Manjunath
15. “**PermPair: Android Malware Detection Using Permission Pairs"** by A. Arora, S. K. Peddoju, and M. Conti.
16. Dataset image
17. EfficientNet Architecture

APPRNDIX

PYTHON CODE

```
import   time import
numpy as npimport torch
import torch.nn.functional as F from torchvision
import transformsfrom torchvision import datasets
from torch.utils.data import DataLoaderimport os
import glob
import torch.nn as nn
from torchvision.transforms import transformsfrom torch.optim
import Adam
from torch.autograd import Variableimport
torchvision
import pathlibimport
zipfileimport math
from torch import nn
import torch.optim as optim
from torchvision import datasets, transformsimport shutil

if torch.cuda.is_available(): torch.backends.cudnn.deterministic = True
from google.colab import drive drive.mount
('/content/gdrive')
!unzip "/content/gdrive/MyDrive/Malware_ISM/Malware1.zip" -d "/content/malware-
images" transformer=transforms.Compose([
        transforms.Resize((256,256)),
        transforms.ToTensor(),   #0-255 to 0-1, numpy to tensors
])
dataset_path = '/content/malware-images/Malimg_Dataset/'

def load_dataset():
        train_dataset_manual = torchvision.datasets.ImageFolder(dataset_path, transform=transformer)
        train_loader_manual = torch.utils.data.DataLoader(train_dataset_man
ual)
        return train_loader_manual


full_dataset = load_dataset()
```

```
train_size = int(0.6 * len(full_dataset))test_size = int (0.2 *
len(full_dataset))
valid_size = len(full_dataset) - train_size - test_size



train_dataset, test_dataset, valid_dataset = torch.utils.data.random_split(full_dataset.dataset, [train_size,
test_size, valid_size])

train_loader = DataLoader(train_dataset, batch_size=batch_size, num_workers=0, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, num_workers=0, shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=batch_size, num_workers=0, shuffle=True)

print('Full Dataset - ' + str(len(full_dataset)) + ' images.') print('Train Set- ' + str(train_size) + ' images in '
+ str(len(train_loader)) +' batches')
print('Testing Set - ' + str(test_size) + ' images in ' + str(len(test_loader)) + ' batches' )
print('Validation Set - ' + str(valid_size) + ' images in ' + str(len(valid_loader)) + ' batches')
for images, labels in train_loader:
    print('Image  batch  dimensions:',  images.shape) print('Image  label
    dimensions:', labels.shape)break
root = pathlib.Path (dataset_path)
classes = sorted ([j.name.split('/')[-1] for j in root.iterdir()])print (classes)
print(len(classes))
device = torch.device ("cuda" if torch.cuda.is_available() else "cpu")print(device)
import torch.nn as nn
import torch.nn.functional as F


class MyModel(nn.Module):def__init_
    (self):
        super(MyModel, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, 3, padding=1)self.bn1 =
        nn.BatchNorm2d(16)
        self.conv2 = nn.Conv2d(16, 32, 3, padding=1)self.bn2 =
        nn.BatchNorm2d(32)
        self.conv3 = nn.Conv2d(32, 64, 3, padding=1)self.bn3 =
        nn.BatchNorm2d(64)
        self.conv4 = nn.Conv2d(64, 128, 3, padding=1)self.bn4 =
        nn.BatchNorm2d(128)
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
```

```
            self.bn5 = nn.BatchNorm2d(256)self.pool =
            nn.MaxPool2d(2, 2)
            self.fc1 = nn.Linear(256 * 8 * 8, 1024)self.fc2 =
            nn.Linear(1024, 512) self.fc3 = nn.Linear(512, 25)
            self.dropout = nn.Dropout(0.2)

        def forward(self, x):
            x = self.bn1(F.relu(self.conv1(x)))x = self.pool(x)
            x = self.bn2(F.relu(self.conv2(x)))x = self.pool(x)
            x = self.bn3(F.relu(self.conv3(x)))x = self.pool(x)
            x = self.bn4(F.relu(self.conv4(x)))x = self.pool(x)
            x = self.bn5(F.relu(self.conv5(x)))x = self.pool(x)
            x = x.view(-1, 256 * 8 * 8)
            x     =     self.dropout(F.relu(self.fc1(x)))    x     =
            self.dropout(F.relu(self.fc2(x)))x = self.fc3(x)
            return  x net
= MyModel()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
path='/content/gdrive/MyDrive/Malware_ISM/Malware_25Epochs'net.load_state_dict(torch.load(path))
!pip install efficientnet_pytorchimport torch
from efficientnet_pytorch import EfficientNet
PATHa = '/content/gdrive/MyDrive/content/EfficientNet/Malware_25Epochs'PATHb =
'/content/gdrive/MyDrive/Malware_ISM/Malware_15Epochs'

#Load Models modelA =
MyModel()
modelB = EfficientNet.from_pretrained('efficientnet-b0').to(device)import torch.nn as nn

# define new instance of the modelmodelA =
MyModel()

# load state_dict into the new model instance
modelA.load_state_dict(torch.load(PATHb)) class
Ensemble(nn.Module):
        def___init_(self, model1, model2):
```

```
                    super(Ensemble, self)._init_()self.model1 =
                    model1
                    self.model2 = model2

          def forward(self, x): out1 =
                    self.model1(x)out2 =
                    self.model2(x)
                    out = (out1 + out2) / 2return out
model1 = modelA
model2 = modelB
model2._fc = nn.Linear(in_features=model2._fc.in_features, out_features
=25, bias=True)
ensemble_model = Ensemble(model1, model2)import
torch.optim as optim

# Define loss function and optimizercriterion =
nn.CrossEntropyLoss()
optimizer = optim.Adam(ensemble_model.parameters(), lr=0.001)


# Move the model to the GPU ensemble_model =
ensemble_model.to(device)import tensorflow as tf
checkpoint_path = "/content/gdrive/MyDrive/Malware_ISM/Malware_Combined
_5Epochs"


cp_callback = tf.keras.callbacks.ModelCheckpoint( checkpoint_path, verbose=1,
       save_weights_only=True,# Save weights, every epoch.
       save_freq='epoch')
num_epochs = 15
for epoch in range(num_epochs):# training
          ensemble_model.train()
          train_running_loss = 0.0
          train_correct = 0
          train_total = 0
          for inputs, labels in train_loader:
                    optimizer.zero_grad()
                    inputs = inputs.to(device) labels =
                    labels.to(device) outputs =
                    ensemble_model(inputs)
                    loss = criterion(outputs, labels)loss.backward()
                    optimizer.step()
                    train_running_loss += loss.item()
                    _, predicted = torch.max(outputs.data, 1)train_total +=
                    labels.size(0)
                    train_correct += (predicted == labels).sum().item()

          train_loss = train_running_loss / len(train_loader)train_accuracy = 100 *
          train_correct / train_total

          # validation
          ensemble_model.eval()
          val_running_loss = 0.0
          val_correct = 0
          val_total = 0
          with torch.no_grad():
                    for inputs, labels in valid_loader: inputs =
                         inputs.to(device) labels = labels.to(device)
                         outputs = ensemble_model(inputs)
                         loss = criterion(outputs, labels)

                         val_running_loss += loss.item()
                         _, predicted = torch.max(outputs.data, 1)val_total +=
                         labels.size(0)
                         val_correct += (predicted == labels).sum().item()

          val_loss = val_running_loss / len(valid_loader)val_accuracy = 100 *
          val_correct / val_total

          # print results
          print(f"Epoch {epoch+1}/{num_epochs} | Train Loss: {train_loss:.4f}
       | Train Acc: {train_accuracy:.2f}% | Val Loss: {val_loss:.4f} | Val Acc: {val_accuracy:.2f}%")
torch.save(net.state_dict(),'/content/gdrive/MyDrive/Malware_ISM/Malware_Combined_20Epochs')
path='/content/gdrive/MyDrive/Malware_ISM/Malware_Combined_20Epochs'
net.load_state_dict(torch.load(path))
print(ensemble_model.keys())
from sklearn.metrics import confusion_matrix,classification_reportimport seaborn as sn
import pandas as pd
import matplotlib.pyplot as pltimport numpy
as np
y_pred = []
y_true = []

# iterate over test data
```

```
#for inputs, labels in test_loader:
for i, (images, labels) in enumerate(test_loader): images, labels = images.to(device),
        labels.to(device)

        net = net.to(device)
        output = net(images) # Feed Network

        output = (torch.max(torch.exp(output), 1)[1])output =
        output.data.cpu().numpy()

        y_pred.extend(output) # Save Prediction

        labels1 = labels.data.cpu().numpy()
        y_true.extend(labels1) # Save Trut
print (len(y_pred))print
(len(y_true))print(y_pred)
print(y_true)
from sklearn.metrics import confusion_matrix,classification_reportimport seaborn as sn
import pandas as pd
import matplotlib.pyplot as pltimport numpy
as np
import pandas as pd
cf_matrix = confusion_matrix(y_true, y_pred)
df_cm = pd.DataFrame(cf_matrix/np.sum(cf_matrix) *10, index = [i for iin classes],
                                        columns = [i for i in classes])plt.figure(figsize
= (25,25))
sn.heatmap(df_cm, annot=True)
classification_report(y_true, y_pred,zero_division=0,output_dict=True)
```