

Malware Classification Using Deep Learning Models

submitted by

Nilesh Kundu

Class Roll No: 002210503023

Exam Roll: MCA244022

Master of Computer Application

under the supervision of

Prof. SARMISTHA NEOGY

in the

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University

2024

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF RECOMMENDATION

To Whom It May Concern

This is to certify that the project report entitled “Malware Classification Using Deep Learning Models” has been carried out by Nilesh Kundu (Reg. No.: 163615 of 2022-23, Roll No.: 002210503023, Exam Roll: MCA244022), under my guidance and supervision and be accepted in partial fulfilment of the requirement for the degree of Master of COMPUTER APPLICATION in DEPARTMENT of COMPUTER SCIENCE and ENGINEERING, JADAVPUR UNIVERSITY during the academic year 2023-2024.

Prof. SARMISTHA NEOGY
Project Supervisor
Dept. of Computer Science and
Engineering
Jadavpur University

Prof. (Dr.) Nandini Mukhopadhyay
Head of the Department
Dept. of Computer Science & Engineering
Jadavpur University

Prof. Dipak Laha
Dean, Faculty Council
of Engineering & Technology
Jadavpur University, Kolkata-032

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF APPROVAL

This is to certify that the project report entitled “**Malware Classification Using Deep Learning Models**” is a bonafide record of work carried out by NILESH KUNDU, in fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

Signature of Examiner 1

Date :

Signature of Examiner 2

Date :

Faculty of Engineering and Technology
Jadavpur University

**DECLARATION OF COMPLIANCE WITH
ACADEMIC ETHICS**

I hereby declare that this project report entitled "Malware Classification Using Deep Learning Models" is an outcome of my genuine and independent work, as part of my Master of Computer Application program.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Nilesh Kundu

Registration No.: 163615 of 2022-23

Examination Roll No.: MCA244022

Thesis Title :- Malware Classification Using Deep Learning Models

Signature with Date

ACKNOWLEDGEMENT

With my most sincere and gratitude, I would like to thank Prof. SARMISTHA NEOGY, Department of Computer Science & Engineering, my supervisor, for his overwhelming support throughout the duration of the project. His motivation always gave me the required inputs and momentum to continue with my work, without which the project work would not have taken its current shape. His valuable suggestions and numerous discussions have always inspired new ways of thinking. I feel deeply honored that I got this opportunity to work under him.

I would like to express my sincere thanks to all my teachers for providing sound knowledge base and cooperation.

I would like to thank all the faculty members of the Department of Computer Science & Engineering of Jadavpur University for their continuous support.

Last, but not the least, I would like to thank my batch mates for staying by my side when I needed them.

Nilesh Kundu

Class Roll No.: 002210503023

Examination Roll No.: MCA244022

Department of Computer Science & Engineering

Jadavpur University

Contents

Sl. No	Topic	Page No.
	Abstract	01
Chapter 1	Introduction	02 - 08
1.1	What is malware	03 - 04
1.2	The intent of malware	04 - 05
1.3	Static and Dynamic Malware analysis	05
1.4	Need of Malware classification	06
1.5	Traditional ML vs Deep Learning for Malware classification	06 - 07
1.6	Objective	07
1.7	Organization of the Project Work	08
Chapter 2	Literature Review	09 - 11
Chapter 3	Methodology	12 - 25
3.1	Datasets	12 - 15
3.2	Binary to image conversion	16 - 17
3.3	Split the datasets	17
3.4	Data Collection and Preprocessing	17
3.5	Transfer Learning	18 - 23
3.6	Finetuning of the Models	24
Chapter 4	Experiment and Results	26 - 42
4.1	Evaluation metric	26 - 27
4.2	Train on Fine-tuned Xception Model	27 - 30
4.3	Train on Fine-tuned ResNet50 Model	31 - 34
4.4	Train on Fine-tuned DenseNet-169 Model	35 - 37
4.5	Comparison of models	40 - 42
Chapter 5	Conclusion and Future Work	43
	References	44 - 46

ABSTRACT

This project explores the effectiveness of fine-tuned Deep Convolutional Neural Network (DCNN) models for the classification of malware. Given the increasing threat of malware and the importance of accurate detection, the study leverages transfer learning to adapt pre-trained DCNN models, originally trained on large-scale image datasets, to the task of malware classification. The research specifically focuses on five prominent DCNN architectures: Xception, ResNet50, DenseNet-169, InceptionResNetV2, and VGG16. These models were evaluated using two distinct malware image datasets: the Blended dataset and the Malevis dataset.

The methodology involved several key steps: preprocessing the malware datasets images, fine-tuning the DCNN models to adapt them to the specific characteristics of the malware images, and applying techniques such as early stopping to optimize model training and prevent overfitting. Performance metrics such as accuracy, precision, recall, and F1-score were used to rigorously assess the models' classification capabilities.

The primary objective of this study was to determine the most effective DCNN architecture for malware classification, providing a comparative analysis of their performance. The findings aim to contribute to the field of cybersecurity by demonstrating the potential of advanced deep learning techniques in improving malware detection systems. By identifying the strengths and weaknesses of each model, this research offers valuable insights for the development of more robust and accurate malware classification tools.

Introduction

The digital age brings incredible convenience and connectivity, but it also introduces new threats. **Malware**, malicious software, lurks in the shadows of the internet, waiting to infiltrate our devices and steal information, disrupt operations, or cause harm. This is big issue for people, businesses, and governments all around the world. The growing reliance on technology and networked systems has increased the sophistication and prevalence of cyberattacks, posing a serious danger to data security and privacy. Phishing, ransomware, and distributed denial-of-service (DDoS) attacks are among the most typical forms of cyberattacks. These attacks can cause significant financial and reputational damage, disrupt essential services, and even pose a threat to national security. In response, there has been a growing emphasis on cybersecurity measures, including the adoption of advanced encryption technologies and the implementation of comprehensive cyberdefence strategies.

Early day malware was not encrypted using complex cypher algorithms and thus were easily detected and classified by cross-matching some code pieces. However, with the recent polymorphism and metamorphism concepts like obfuscation, malware classification becomes a challenging and tedious task. Polymorphic malware exploits an encryption algorithm, which encrypts the code each time it iterates, while the encryption key remains constant, which makes it easier to detect [1]. In comparison, Metamorphic malware, which not only encrypts the code each time it iterates but, also change its encryption key, which makes it hard to detect [2]. It is observed that the total number of instances per day is growing drastically over the years, and thus manual inspection of malware analysis is considered ineffective. One of the main reasons for generating a high volume of malware samples is the extensive use of malware developers' obfuscation techniques, which means that malicious files from the same malware family (i.e. similar code and common origin) are modified continuously and obfuscated.

According to Statista's report in 2023, the worldwide number of malware attacks reached 6.06 billion, an increase of 10 percent compared to the preceding year. In recent years, the highest number of malware attacks was detected in 2018, when 10.5 billion such attacks were reported across the globe.[3]

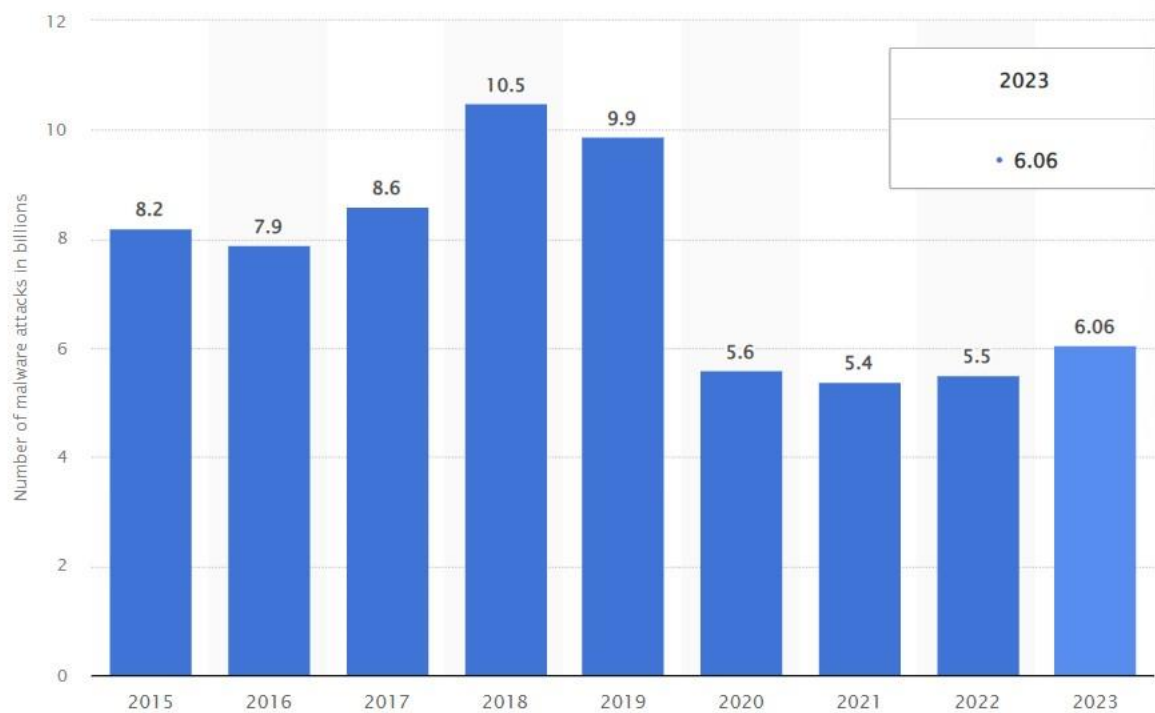


Figure 1.1-Annual number of malware attacks worldwide from 2015 to 2023(in billions)

1.1 What is malware:

Malware, short for malicious software, refers to any intrusive software developed by cybercriminals (often called hackers) to steal data and damage or destroy computers and computer systems. It is like a sneaky intruder that gets onto your computer to steal information, disrupt operations, or cause damage. It comes in many forms, and cybercriminals constantly develop new strains.[4]

Types of malware:

- ☐ **Viruses:** These replicate themselves and spread to other files or computers, much like a biological virus.
- ☐ **Worms:** Similar to viruses, worms can spread quickly but don't necessarily require attaching themselves to other programs.
- ☐ **Trojan Horses:** These disguise themselves as legitimate software, tricking users into installing them. Once installed, they unleash their malicious payload.

- ❑ **Spyware:** This malware lurks in the background, spying on your activity and stealing your personal information like passwords or browsing habits.
- ❑ **Ransomware:** This particularly nasty type encrypts your files, essentially holding them hostage until you pay a ransom to regain access.
- ❑ **Adware:** This bombards you with unwanted pop-up ads and can sometimes even track your browsing activity.

1.2 The intent of malware:

Malware, or malicious software, is designed to harm or exploit computer systems, networks, or devices. The intent of malware can vary, but common objectives include:

Data theft or exfiltration: Malware may be designed to steal sensitive information, such as passwords, credit card numbers, or personal data.

Data destruction or corruption: Malware can be programmed to delete, modify, or corrupt files, making them unusable or causing data loss.

System compromise or control: Malware can gain unauthorized access to a system, allowing attackers to remotely control or manipulate the device.

Financial gain: Malware can be used to extort money from victims through ransomware attacks, where data is encrypted and demands are made for payment to restore access.

Espionage or surveillance: Malware can be used to monitor user activity, track keystrokes, or capture sensitive information for malicious purposes.

Distributed Denial of Service (DDoS) attacks: Malware can be used to overwhelm a system or network with traffic, causing it to become unavailable or slow.

Spam or phishing: Malware can be used to send unsolicited emails or messages, often with the goal of tricking users into revealing sensitive information or installing additional malware.

System disruption or destruction: Malware can be designed to cause system crashes, blue screens, or other disruptions, making it difficult or impossible to use the device.

These malicious intentions can be achieved through various means, including:

- **Infecting devices:** Malware can spread through infected software downloads, infected websites, or infected email attachments.
- **Exploiting vulnerabilities:** Malware can exploit known vulnerabilities in software or operating systems to gain access to a system.
- **Social engineering:** Malware can be spread through social engineering tactics, such as phishing or pretexting, to trick users into installing or executing the malware. [5,6,7]

1.3 Static and Dynamic Malware analysis:

The objective of the malware analysis is to study the behaviour and structure of malware, and there are two types: static analysis and dynamic analysis.

Static analysis is a method used to examine the behavior and structure of a malware sample without executing it. This type of analysis involves analyzing the binary code of the malware and identifying its various components, such as function calls, system calls, libraries imported, and metadata like file size, timestamps, and digital signatures. Dynamic malware analysis involves running the malware in a controlled environment to observe its behavior, which helps to identify the malicious actions that it performs on a system.

Both types of analysis have their advantages and limitations and they complement each other. Static analysis is faster but, if malware is successfully concealed using code obfuscation techniques, it could evade detection. Contrarily, code obfuscation techniques and polymorphic malware hardly evades dynamic analysis as it monitors and analysis the runtime execution of a program. Nevertheless, traditional malware detection and malware analysis are unable to keep pace with new attacks and variants. Organizations are facing the daunting challenge of dealing with millions of attacks a day.

1.4 Need of Malware Classification:

Malware classification is crucial in the fight against cyber threats for a couple of key reasons:

1. **Understanding the Threat:** Different types of malware cause different kinds of damage. By classifying malware, security professionals can identify the specific threat posed by a particular piece of malicious software. This allows them to take targeted action to mitigate the damage and remove the malware. For instance, a Trojan horse might steal your data, while ransomware might lock you out of your files until you pay a ransom. Understanding the type of malware helps you respond accordingly.
2. **Developing Defences:** Classifying malware helps create more effective defences. Security software can be programmed to recognize specific signatures or behaviours associated with different malware families. This allows for better detection rates and helps prevent new variants of existing malware from infiltrating systems. Think of it like having different antibodies to fight different viruses in your body.

1.5 Traditional ML vs Deep Learning for Malware classification:

The limitations of a Machine Learning Approach for malware classification are as follows:

1. **Resources:** A significant amount of computing resources were required to process and extract features from the byte files. Additionally, increasing the number of n-grams (unigrams and bi-grams) from 1 and 2 to 4 or even 8 would require a lot of free RAM and processing power.
2. **Time:** Processing these features also required a significant amount of time, especially if the number of features were to be increased beyond a certain practical threshold.
3. **Domain Knowledge:** Prior domain knowledge in cybersecurity and machine learning may be required to understand and extract the necessary features required for model training.

The Deep Learning approach on the other hand, has the following benefits:

1. **Automatic Feature Extraction:** Since byte files were converted to images for convolutional networks, feature extraction is done automatically and does not require any manual effort or prior domain knowledge.
2. **Classification efficiency:** Since deep learning networks can incorporate a greater number of features than regular machine learning models, they have a better classification accuracy and higher confidence levels. In this malware example, CNNs were able to successfully extract author signatures and better predict malware classes. This also allowed for faster detection of malware classes, permitting a quicker response in anti-malware cybersecurity applications.

1.6 Objective:

Malware analysis is a fast-growing field demanding a great deal of attention because of technological development progresses in social networks, cloud computing, mobile environment, smart grid, Internet of Things (IoT) and Industrial Internet of Things (IIoT), etc. As the technologies developed The exponential increase in malware attacks has become one of the major threats to Internet security. A recent threat report from Symantec indicated that their 123 million sensors record thousands of malicious threat events per seconds on daily basis.[8]

Malware writers deploy many obfuscation methods such as dead-code insertion, subroutine reordering, and code transposition, to create variants of an existing malware family in order to evade detection [9].The most challenging part of internet security is discovering malware variants. The similarities between many malware variants like Nuwar, Kelihos and Storm suggest they were developed by the same malware coders [10] . According to Symantec, 317 million new malware variants appeared in 2014, and this number is increasing. Traditional antimalware solutions are mostly incapable of dealing with the diversity and volume of malware variants which is evident today [10].

In this project, we try to contribute in this field by employing transfer learning, relying on models that have been pre-trained on large image datasets for malware classification and compare their performance.

1.7 Organization of the Project Work:

Chapter 1: Introduction

Introduces the importance of malware classification in cybersecurity. It sets the context for the project, highlighting the challenges posed by malware and the need for advanced detection techniques. It also outlines the objectives of the project, focusing on the application of deep convolutional neural networks (DCNNs) for malware classification.

Chapter 2: Literature Review

This chapter reviews existing literature on malware classification techniques. It provides an overview of traditional methods and the evolution of machine learning and deep learning approaches. This chapter specifically examines the role of transfer learning and pre-trained models in enhancing malware detection.

Chapter 3: Methodology

It explains the theoretical concepts underlying the study, including an introduction to transfer learning and some deep convolutional neural networks (DCNNs). It details the methodology adopted for the project, describing the datasets used (Malevis and a blended dataset), the preprocessing steps, and the fine-tuning of pre-trained models for malware classification.

Chapter 4: Experiments and Results

This chapter presents the experiments conducted in the project. It describes the training and validation process for each pre-trained model on the given datasets. This chapter provides a comprehensive analysis of the results, including the performance metrics such as accuracy, precision, recall, and F1-score, comparing the effectiveness of the different models.

Chapter 5: Conclusion and Future Work

In this chapter discusses the findings, highlighting the strengths and limitations of the approaches used. It concludes with suggestions for future research, focusing on further fine-tuning techniques, model ensembling, and broader application to different types of malware datasets to improve robustness and performance.

Literature Review

The multiclass classification of malware images has been tried in literature by using various data augmentation techniques, sequential modeling, and convolutional neural networks.

Agarap, A. F. et al., 2017 discuss an SVM-based deep learning model to classify the byteplot images in the Maling dataset with various feature extractors like MLP, CNNs, and GRUs. They achieve a predictive accuracy of 84.92% with the Maling dataset and GRU-SVM model. The usage of sequential models to process the byteplot images of varied sizes is commendable but the accuracy is comparatively decent.[11] Kalash, M. et al., 2018 design a CNN-based framework that is proposed to render better performance than the traditional approaches of shallow learning for malware classification using Byteplot images. They test the model on the Maling dataset and Microsoft dataset resulting in an accuracy score of 98.52% and 99.97% accuracy respectively. The accuracy of the customized framework is commendable but on the other hand, only accuracy is used as the primary metric on an imbalanced dataset like the Maling dataset.[12]

Lo, W. W. et al., 2019 discuss an Xception model that performs better than existing models like VGG16 and other traditional models like KNN and SVM. XceptionNet obtains the highest validation accuracy against the other models VGG16, KNN, and SVM. The high accuracy is commendable but the usage of accuracy as the primary metric can be misleading on the actual nature of the model.[13]

Singh, A. et al., 2019 prepare a malware dataset using data collection, and deep neural networks are designed to classify the images across 22 families. An accuracy of 98.98% and 99.40% using deep CNN and ResNet-50 respectively is achieved. The high accuracy is commendable.[14]

J. H. Go et al., 2020 experiment ResNeXt model for the classification of malware byteplot images. They achieve an accuracy of 98.32% and 98.86% on the Maling dataset and Maling dataset after image enhancement. The enhancement of image

quality and the resulting high accuracy is commendable but accuracy cannot be a sufficient metric to evaluate the model quality for an imbalanced dataset like the Maling dataset.[15]

Ghouthi et al., 2020 discuss an approach of extracting image features after a Principal Component Analysis and then using an SVM to perform the classification. They use the Maling, Ember, and BIG 2015 malware datasets to reach accuracy values of 99.8%, 91.1%, and 99.7%, respectively. Evaluation of the model on different datasets gives to a good understanding of the model quality but dimensionality reduction can lead to the loss of information.[16]

Mitsuhashi, R. et al., 2020 discuss an approach to solve the data imbalance using the undersampling technique and fine-tuning VGG19 on the Maling dataset. They obtained an accuracy of 99.72%. The high accuracies and data augmentation is commendable but the usage of accuracy as the primary metric can be misleading for an imbalanced dataset like the Maling dataset.[17]

Danish Vasan et al., 2020 experiment with transfer learning using the Maling dataset and IoT-android mobile dataset. The performance of this model is compared with existing pre-trained CNNs. The Maling malware dataset shows accuracy of 98.82%, and the IoT-android mobile dataset shows accuracy of about 97.35%. The high accuracies are commendable but the usage of accuracy as the primary metric can be misleading for an imbalanced dataset like the Maling dataset.[18]

Aslan, O. et al., 2021 discuss a hybrid model integrating the performance of two pretrained models namely AlexNet and ResNet152 in an optimal manner. The model is tested on Maling, Microsoft BIG 2015, and Malevis datasets. For the Maling dataset, it gives 97.78% accuracy. The higher accuracy and usage of a hybrid model are commendable but the usage of accuracy as the primary metric can be misleading for an imbalanced dataset like the Maling dataset.[19]

Asam, M. et al., 2021 discuss an approach to the extraction of features from multiple CNNs and fusing their results. Finally, using an SVM to discriminate between them. The architecture achieves an accuracy of 98.61%, an F-score of 0.96, a precision

of 0.96, and a recall of 0.96. The performance of the model is good and its evaluation using different classification metrics is commendable.[20]

Awan,M.J. et al., 2021 discuss a spatial attention and convolutional neural network approach for the multiclass classification of malware. They achieve a precision of 97.42%, a recall of 97.95%, a specificity of 97.33%, and an F1 score of 97.32%. The performance of the model is good based on the reported classification metrics.[21]

Mallik, A. et al., 2022 describe an approach to resolve data imbalance using data augmentation and the augmented dataset is classified by using 2 LSTM layers and 1 VGG Net. The overall results from each are integrated and combined. Treating the malware file bits as a bidirectional dependency is commendable.[22]

AlGarni, M. D. et al., 2022 have compared the performance of EfficientNetB3 on Imagenet and Malimg datasets. They obtain an accuracy of 99.93% on the Malimg dataset. The comparison of the performance of the model on two datasets is commendable but accuracy cannot be a sufficient metric to evaluate the model quality for an imbalanced dataset like the Malimg dataset.[23]

Adem Tekerek et al., 2022 resolve the classification by data augmentation using CycleGAN and use different CNNs for classification. They achieve an accuracy of 99.86% for the BIG2015 dataset and 99.60% for the Dumpware10 dataset. The high accuracy is commendable.[24]

Methodology

The proposed method consists of the following basic steps:

1. Two datasets are provided: Malevis[25] and Blended image[26], each containing labeled malware images.
2. Images are rescaled to a standardized size suitable for input into the deep neural network.
3. Transfer learning is employed using pre-trained convolutional neural network (CNN) architectures.
4. The top (fully connected) layers of the pre-trained models are replaced with new layers suitable for malware classification. And the parameters of the convolutional layers from the pre-trained model are frozen to preserve their learned features during training.
5. Fine-tuned by the added fully connected layers are trained while keeping the pre-trained convolutional layers fixed to maintain learned features that are beneficial for classification.

3.1 Datasets:

1. Malevis dataset:

Malevis dataset is aimed for researchers to supply a RGB based ground truth dataset in order to evaluate their vision based multi-class malware recognition studies. For this purpose, we supply a corpus involving byte images of 26 (25+1) classes. Here, 1 class represents the "legitimate" samples while the rest of the 25 classes correspond to different malware types. To construct this corpus, we first extracted the binary images from malware files (supplied by Comodo Inc) in 3 channel RGB form by using bin2png script developed by Sultanik. Following to having vertically long images, we then resized the images in 2 different square sized resolution (224x224 and 300x300 pixels).

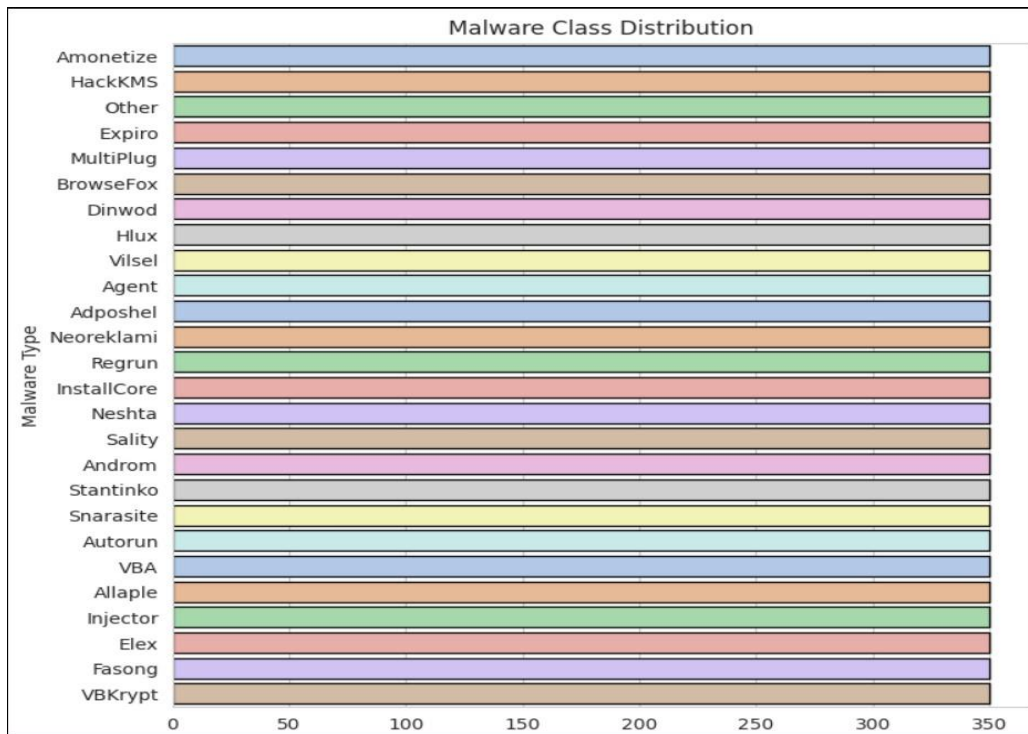


Figure 3.1 - Malevis dataset class distribution

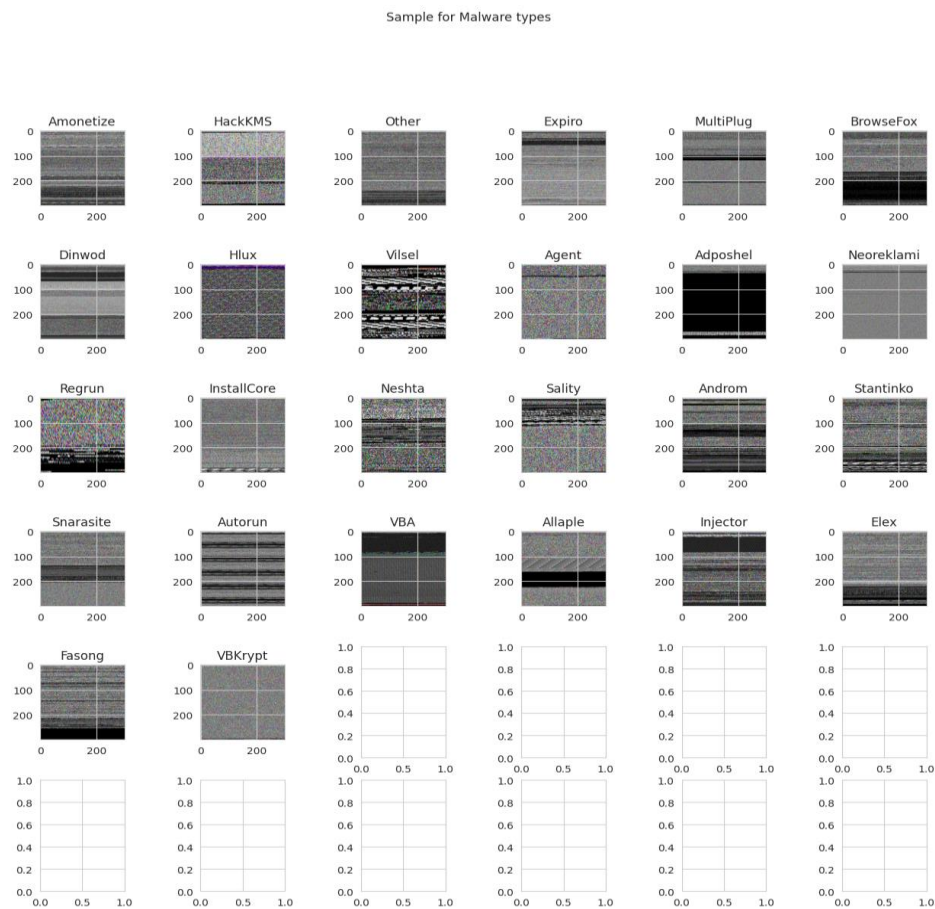


Figure 3.2 - Sample image of the Malevis dataset

2. Blended Dataset:

Blended image dataset for malware classification is the combination of the Maling dataset and the Malevis dataset. The Maling dataset is highly imbalanced in terms of class distribution and the Malevis dataset is a well-balanced dataset. Both of the datasets are blended together to form a single dataset which has all the classes(26) from the Malevis dataset and 5 classes from the Maling dataset(total 31 classes).

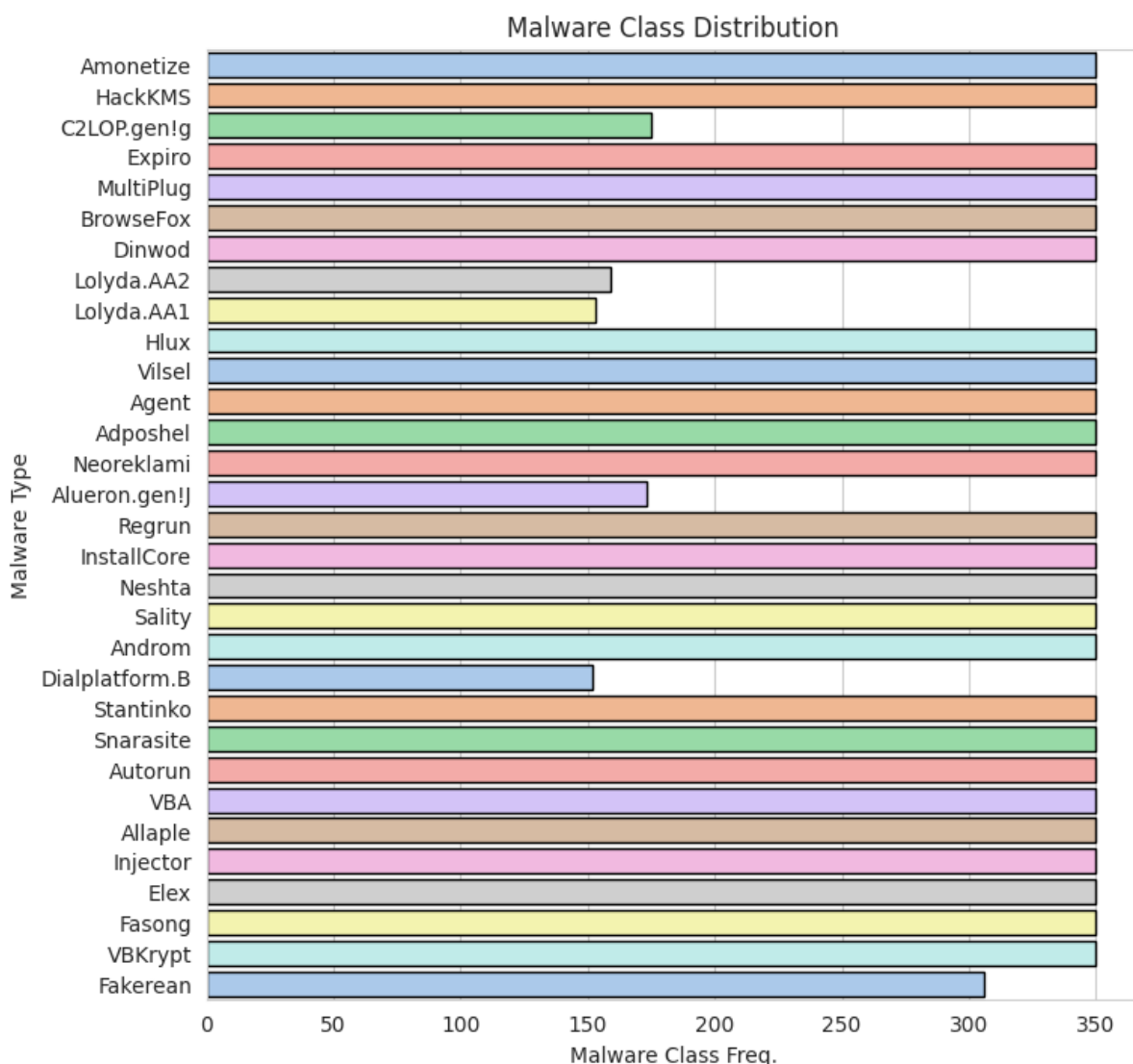


Figure 3.3 - **Blended dataset class Distribution**

Sample for Malware types

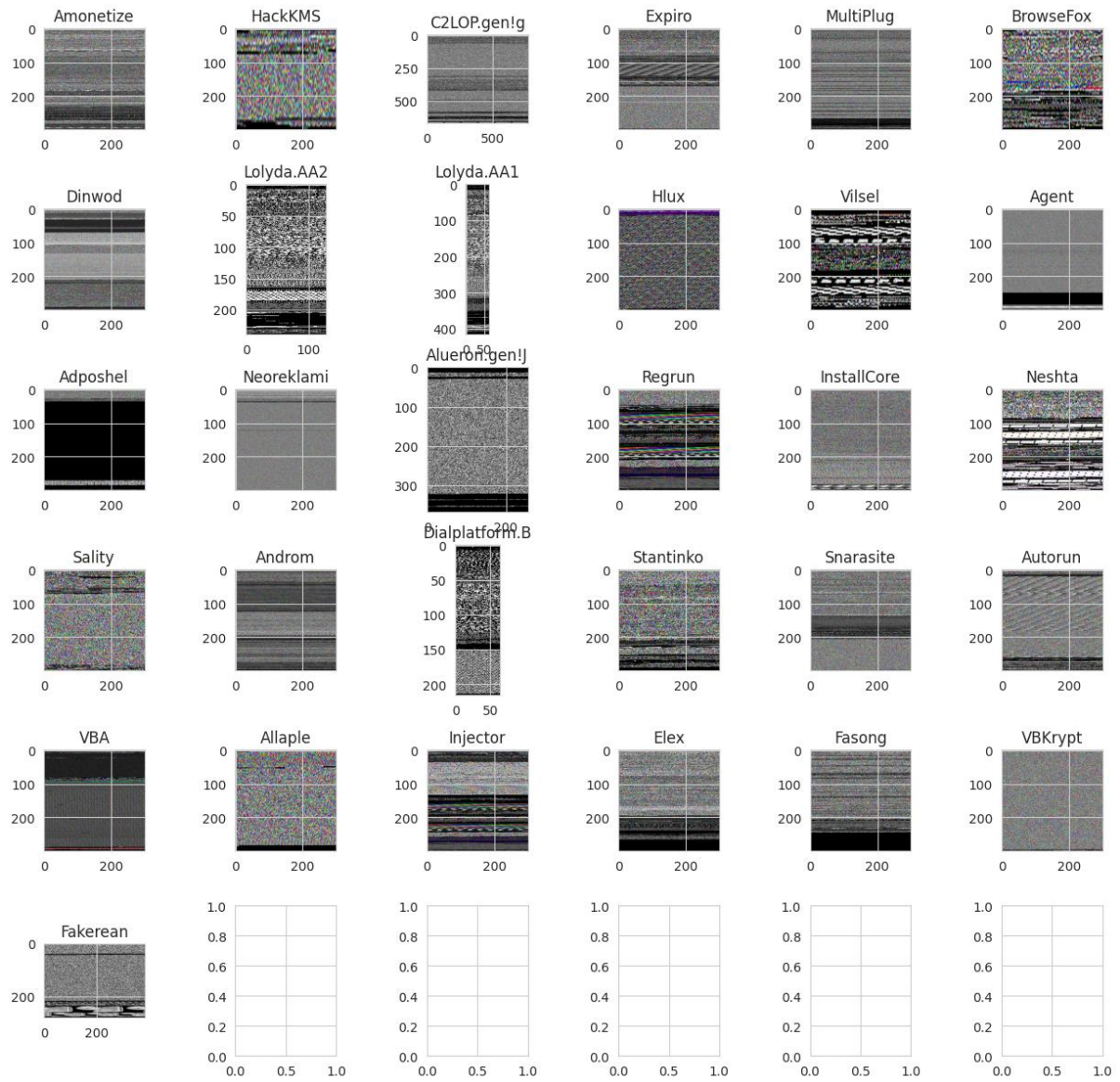


Figure 3.4 - Sample image of Blended dataset

3.2 Binary to image conversion:

As the Blended and Malevis dataset already contains malware images, we will use another dataset to first show you how to convert a malware into an image. We took as example files from Microsoft Malware Classification Challenge (BIG 2015) dataset.[27]

For each file, the raw data contains the hexadecimal representation of the file's binary content. The goal is to convert those files into PNG images and use them as the input of our transfer models.

We can convert each malware using the following function:[28]

```
def convertAndSave(array,name):
    print('Processing '+name)
    if array.shape[1]!=16: #If not hexadecimal
        assert(False)
    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))
    im.save(root+'\\'+name+'.png', "PNG")
    return im

#Get the list of files
files=os.listdir(root)
print('files : ',files)
#We will process files one by one.
for counter, name in enumerate(files):
    #We only process .bytes files from our folder.
    if '.bytes' != name[-6:]:
        continue
    f=open(root+'/'+name)
    array=[]
    for line in f:
        xx=line.split()
        if len(xx)!=17:
            continue
        array.append([int(i,16) if i!='??' else 0 for i in xx[1: ]])
    plt.imshow(convertAndSave(np.array(array),name))
    del array
    f.close()
```

Here is the resulting image after converting the 0ACDbR5M3ZhBJajygTuf.bytes (a image of the above mention dataset) binary file into a PNG.



0ACDbR5M3ZhBJajygTuf.bytes to PNG

Thanks to this method, we can create our own PNG Malware Dataset. However, we will not process the entire dataset and will now focus on the Malming dataset in which malware is already converted to images.

3.3 Split the datasets:

We splits the datasets into training and test sets using specified ratios with the Help of splitfolders library to handle the splitting process. In the datasets there is already present Validation set so we didn't split validation set. In this project we split the dataset in 80% in training and 20% in testing data.

3.4 Data Collection and Preprocessing:

In the project, a class named 'ImagePreprocess' was implemented to efficiently handle the preprocessing of images for deep learning models. This class facilitates the preparation of training, validation, and test datasets, ensuring they are properly formatted for input into CNN models. Here is a brief overview of how the class operates:

The class initializes with the selected CNN model variant and sets up paths for the training, validation, and test datasets. It also defines the target size for image resizing and the color mode (e.g., RGB or grayscale). A method named 'create_generators' creates image data generators for the training, validation, and test datasets. These generators apply the preprocessing function specific to the chosen CNN model, ensuring that the images are appropriately scaled and normalized. There another method named 'get_images' loads images from the specified directories, excluding any unwanted directories. It then preprocesses these images according to the defined target size and color mode. This method sets up the data flow for training, validation, and testing, ensuring that images are shuffled for training and kept in a fixed order for testing.

The 'ImagePreprocess' class provides a streamlined and consistent approach to handling image data, making the preprocessing phase efficient and ensuring that the data fed into the CNN models is correctly formatted.

3.5 Transfer Learning:

Concept of transfer learning is using the pretrained model of one task as the foundation for other tasks. Idea is to take the best pretrained models and used in a specific situation and use them in another setting by tuning the parameters that would be best for that situation. ResNet, Oxford VGG, Google Inception are the widely used pretrained models. These models were the winners of the ImageNet competition that used to take place before 2015. Before 2010 transfer learning was not given importance but after the rapid use of deep learning in industry started it has been a hot topic. Application of transfer learning comprises image classification, sentiment classification, simulation, gaming and many more. The transfer learning has been used by [16] for malware classification with fine tuning. Building any model from the scratch is a tedious and time-consuming process but via transfer learning it can be easily achieved. The difficult task in transfer learning is selecting a layer and freezing unwanted layers, setting weights and hyper parameter tuning.

In our project we use ResNet50, DenceNet-169, Xception, Inception-ResNetV2, VGG16 CNN based deep learning pretrained models to classify malwares.

3.5.1 ResNet50:

ResNet50 is a cutting-edge deep convolutional neural network architecture that was developed by Microsoft Research in 2015. It is a variant of the popular ResNet architecture and comprises of 50 layers that enable it to learn much deeper architectures than previously possible without encountering the problem of vanishing gradients. The architecture of ResNet50 is divided into four main parts: the convolutional layers, the identity block, the convolutional block, and the fully connected layers.

The convolutional layers in ResNet50 consist of several convolutional layers followed by batch normalization and ReLU activation. These layers are responsible for extracting features from the input image, such as edges, textures, and shapes. The convolutional layers are followed by max pooling layers, which reduce the spatial dimensions of the feature maps while preserving the most important features.

The identity block and convolutional block are the key building blocks of ResNet50. The identity block is a simple block that passes the input through a series of convolutional layers and adds the input back to the output. This allows the network to learn residual functions that map the input to the desired output. The convolutional block is similar to the identity block, but with the addition of a

1x1 convolutional layer that is used to reduce the number of filters before the 3x3 convolutional layer.

The final part of ResNet50 is the fully connected layers. These layers are responsible for making the final classification. The output of the final fully connected layer is fed into a softmax activation function to produce the final class probabilities.[29]

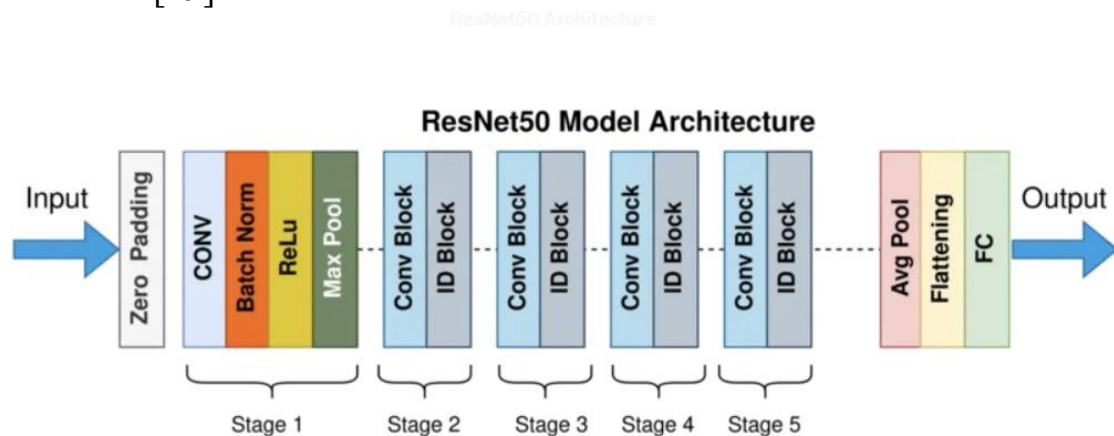


Figure 3.5 - ResNet50 model architecture

ResNet50 has been trained on large datasets and achieves state-of-the-art results on several benchmarks. It has been trained on the ImageNet dataset, which contains over 14 million images and 1000 classes. On this dataset, ResNet50 achieved an error rate of 22.85% which is on par with human performance, which is an error rate of 5.1%.

3.5.2 DenseNet-169:

The densenet-169 model is one of the DenseNet group of models designed to perform image classification. DenseNet is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other. It has far less trainable parameters when compared to its fellow DenseNet architectures with fewer layers. DenseNet-169 and the other DenseNet architectures have the ability to overcome the vanishing gradient problem, have a strong feature propagation strategy, minimize the number of trainable parameters, and encourage the reuse of features, thus making them a family of very reliable DL architectures.

The DenseNet169 architecture is composed of several types of layers including dense, transition layers, classification layers:

Dense Blocks: These are the building blocks of DenseNet-169, containing multiple convolutional layers. Each layer in a dense block receives not only the input from the previous layer but also the feature maps from all preceding layers within that block. This dense connectivity promotes feature reuse and alleviates the vanishing gradient problem, allowing for deeper networks.

Transition Layers: These layers are inserted between dense blocks to reduce the number of feature maps and control the network's size. They typically use pooling operations like average pooling to decrease the resolution of the feature maps.

Classification Layers: The final stages of the network consist of global average pooling, fully connected layers, and a softmax layer for outputting class probabilities.

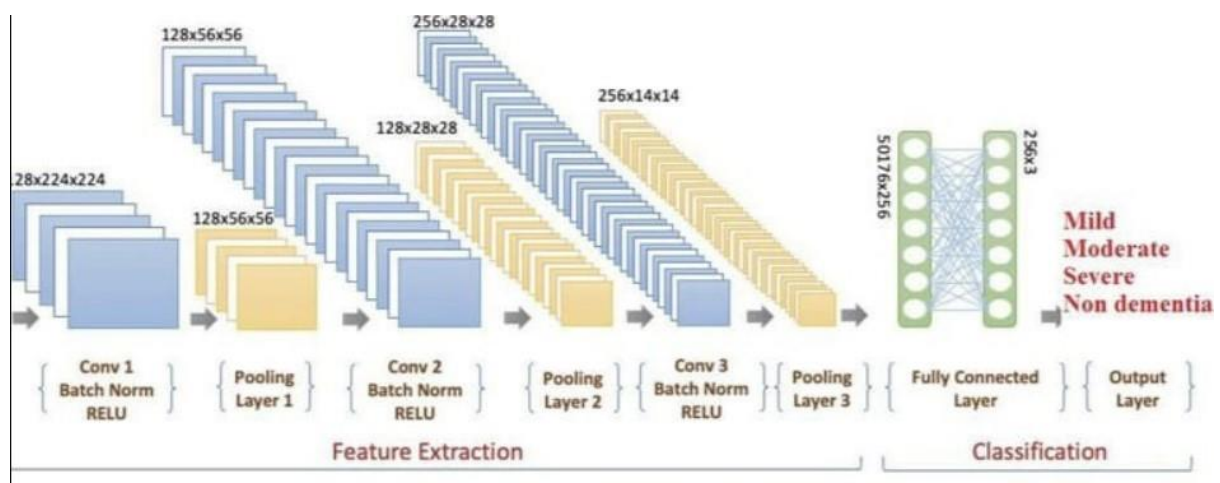


Figure 3.6 - **DenceNet-169 model Architecture**

3.5.3 XceptionNet:

Xception is a convolutional neural network. Xception is the pretrained network can classify images into 1000 object categories. It's architecture, centred around depthwise separable convolutions, and its strategic approaches contribute to its success in the realm of deep learning. As a scalable and efficient model, Xception continues to be influential in computer vision tasks and serves as a testament to the ongoing evolution of neural network architectures.[30]

Architecture:

Depthwise Separable Convolutions:— The hallmark of Xception is its utilization of depthwise separable convolutions. Unlike traditional convolutions, which operate on both spatial and depth dimensions simultaneously, depthwise separable convolutions separate these operations. This significantly reduces the number of parameters and computational cost while maintaining representational power.

Separable Convolutions: Xception's architecture is based on a series of separable convolutional blocks. Each block consists of a depthwise separable convolution followed by a linear transformation through a pointwise convolution. This modular design enhances the model's ability to capture complex patterns with fewer parameters.

Entry Flow and Exit Flow: Xception is organized into an entry flow and an exit flow. The entry flow extracts features from the input, while the exit flow refines these features for final predictions. This hierarchical structure aids in learning hierarchical representations and facilitates the flow of information through the network.

Skip Connections: Inspired by the success of residual connections in ResNet, Xception incorporates skip connections within its architecture. These connections enable the direct flow of information across different layers, fostering the efficient training of deep networks.

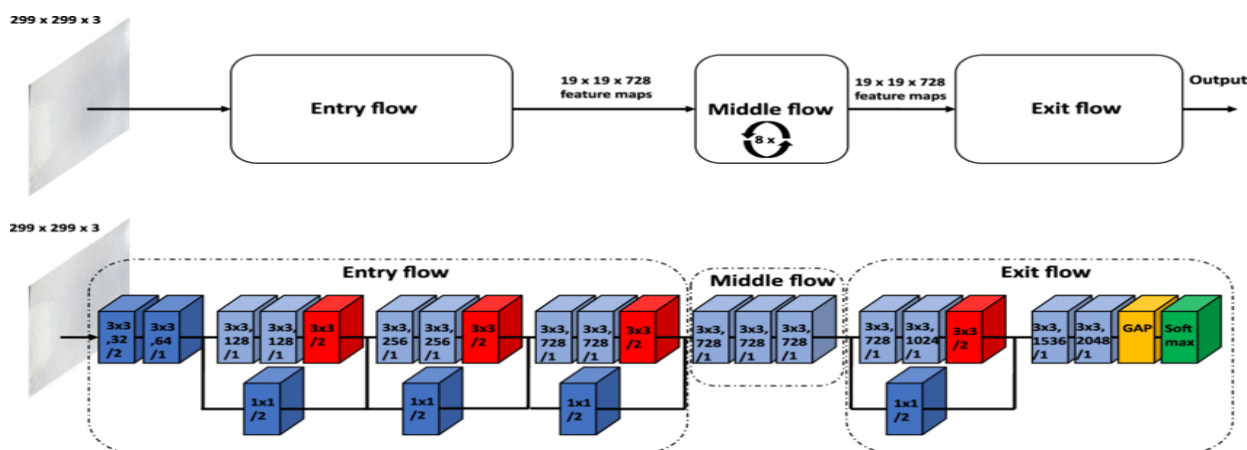


Figure 3.7 - XceptionNet Model Architecture

3.5.4 VGG16:

VGG16 is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3×3) convolution filters, which showed a significant improvement on the prior-art configurations. It is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning.[31]

Architecture:

The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer. It takes input tensor size as $224, 244$ with 3 RGB channel. Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3×3 filter with stride 1 and always used the same padding and maxpool layer of 2×2 filter of stride 2. The convolution and max pool layers are consistently arranged throughout the whole architecture. Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters. Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

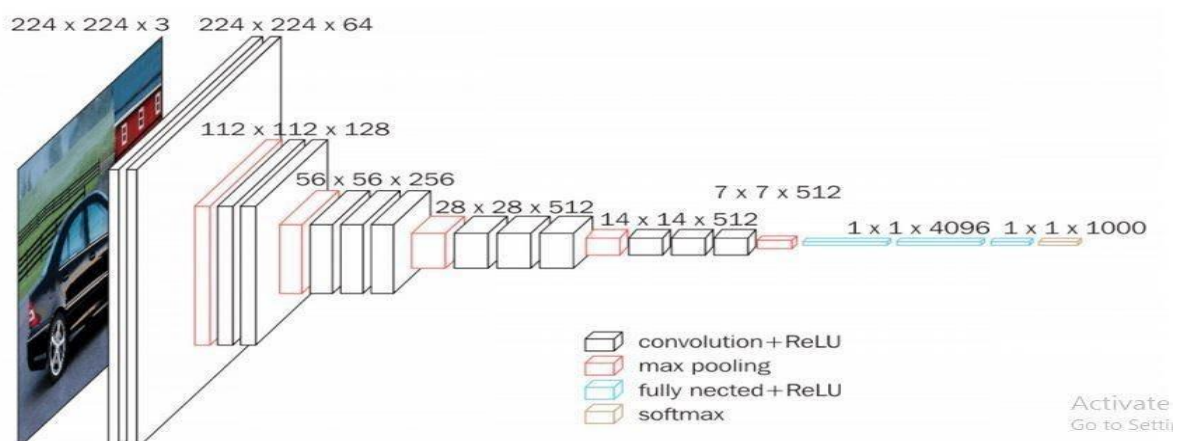


Figure 3.8 - VGG16 Model's Architecture

3.5.5 Inception-ResNetV2:

Inception-ResNet v2 is a hybrid architecture that combines the Inception architecture with residual connections from ResNet. This combination aims to leverage the strengths of both networks: the multi-scale feature extraction of Inception modules and the efficient training of deep networks using residual connections. The Inception modules are designed to capture multi-scale features by applying multiple convolution filters of different sizes (e.g., 1×1 , 3×3 , 5×5) in parallel within the same layer. These modules help in extracting rich features at various scales. Residual connections or shortcuts are added to the network to allow gradients to flow more easily during backpropagation. This helps in mitigating the vanishing gradient problem and enables the training of very deep networks.

Architecture:

Inception-ResNetv2 combines Inception modules with residual connections. It stacks multiple Inception modules with different configurations, progressively extracting features at various levels of complexity. Residual connections are strategically inserted within these modules, enabling information flow across deeper layers and potentially improving gradient propagation.

Inception Resnet V2 Network

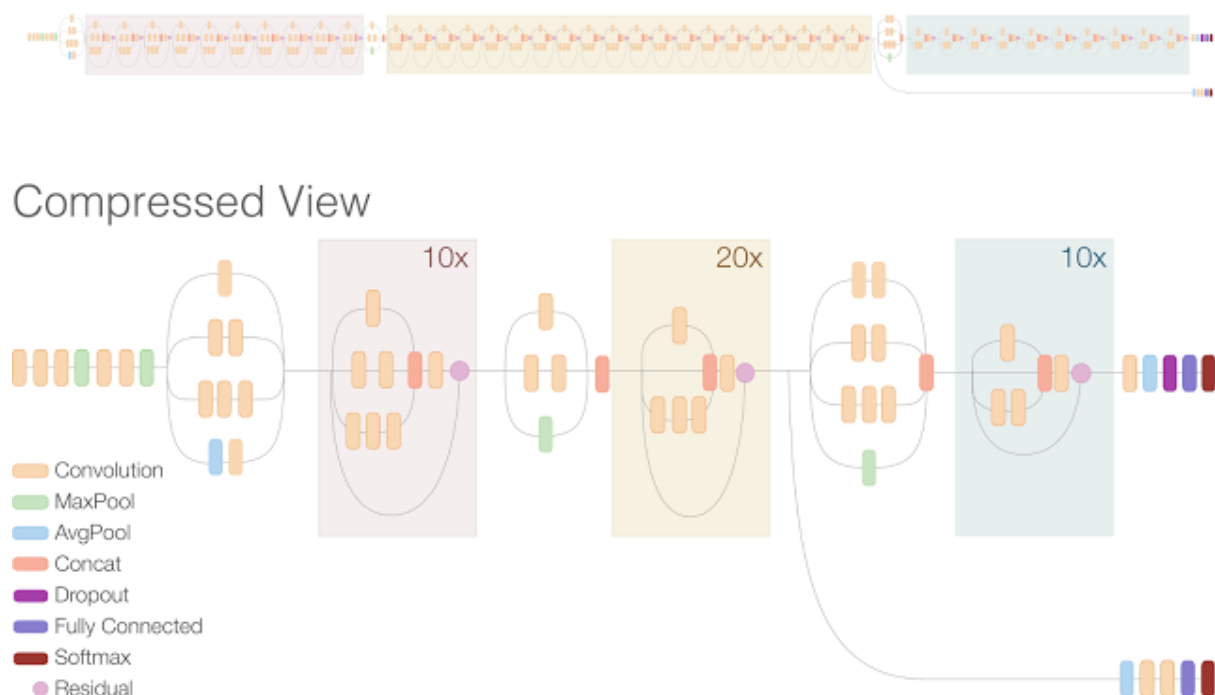


Figure 3.9 - Inception-ResNetV2 Models Architecture

3.6 Finetuning of the Models:

We create a class named **‘FineTuning’**. it presented here encapsulates a comprehensive pipeline for fine-tuning a pre-trained neural network model for a specific classification task. This approach leverages transfer learning by adapting an existing model to a new dataset, potentially improving performance and efficiency.

It takes training dataset, validation dataset, pre-trained model, number of classes in the target dataset, boolean flag indicating whether to apply regularization, patience parameter for early stopping during training as parameters. It has methods named **‘fine_tune’** , **‘add_final_layer’** , **‘compile_model’**, **‘train_model’** .

The **‘fine_tune’** method selectively freezes layers of the pre-trained model up to a specified layer and unfreezes the rest for fine-tuning. This process enables the model to retain the knowledge from the pre-trained layers while adapting the later layers to the new task.

The **‘add_final_layer’** method appends custom fully connected layers to the base model. If regularize is set to True, additional layers and dropout regularization are included to prevent overfitting.

The **‘compile_model’** method finalizes the model structure by combining the pre-trained model base with the new layers added. The model is then compiled with the Adam optimizer and categorical cross-entropy loss function. Several metrics are defined to evaluate the model's performance.

The **‘train_model’** method trains the fine-tuned model on the training dataset and validates it on the validation dataset. Early stopping is employed to halt training if the validation loss does not improve for a specified number of epochs (patience), ensuring the model retains the best weights encountered during training.

The **‘FineTuning’** class provides a structured and modular approach to fine-tuning pre-trained neural network models. By freezing and unfreezing specific layers, adding custom layers with optional regularization, and employing early stopping, this class ensures efficient transfer learning tailored to malware classification tasks.

In this chapter we discuss the methodologies that we used in this project. In the next chapter we will discuss about the experimentation and results of the above mention models on the given datasets.

Experiment and Results

In this section, we evaluated the performance of the proposed DCNN model using the dataset and compared them. The experimentation was done in Python language on the kaggle notebook integrated development environment. The models were trained on AMD RYZEN 5500U processor with 16 GB of RAM and with virtual environment of GPU P100. We mainly used TensorFlow's Keras library and also some other library's like numpy, pandas, sklearn, matplotlib etc.

4.1 Evaluation metric:

Standard performance metrics for evaluation including accuracy, precision, recall, F1-Score were used. Given that,

True Positive (*TP*): is the number of samples correctly detected as malware

False Positive (*FP*): is the number of samples incorrectly detected as malware

True Negative (*TN*): is the number of samples correctly detected as benign

False Negative (*FN*): is the number of samples incorrectly detected as benign.

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

Precision (P): is the number of true positives predictions (TP) divided by all true positive predictions (TP) plus the number of false positives (FP).

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

Recall (R): is the number of true positives (TP) divided by the number of true positives plus the number of false negatives (FN).

$$Recall = \frac{TP}{TP + FN},$$

F1 Score: F1 score is the weighted average of precision and recall.

$$F1-Score = 2 * \frac{Recall * Precision}{Recall + Precision}$$

4.2 Train on Fine-tuned Xception Model:

On Blended dataset:

At the beginning we preprocessed and resized the images from the dataset to 75x75 pixels. Then an Xception model is initialized using `tensorflow.keras.applications.Xception`. Then we initialized it without top layer and input shape of 75x75x3. The model is then fine-tuned using the above mentioned ‘**FineTuning**’ class, with the initial layers frozen and new dense layers added for classification into 31 classes. The model is compiled with the Adam optimizer, categorical cross-entropy loss, and evaluation metrics. Finally the model is trained on the training dataset and validation on validation dataset. It has a callback function for early stopping after 10 epochs if no improvement in validation loss.

The training begins with the first epoch out of the planned 80 epochs. Initially, the model shows very low accuracy (0.0729) and high loss (7.3513) on the training set. The validation accuracy is significantly higher (0.6020), indicating that the model might initially overfit or learn rapidly from the validation set. As training progresses, the training accuracy improves significantly. By Epoch 2, training accuracy jumps to 0.8041, and validation accuracy increases to 0.8146. By Epoch 5, training accuracy reaches 0.9281, but validation accuracy slightly fluctuates.

By Epoch 10, training accuracy reaches 0.9602, and validation accuracy stabilizes around 0.9152. The model uses early stopping based on validation loss with patience set to 10 epochs. This helps prevent overfitting by stopping training when validation performance stops improving. Thus, training stops after 20 epochs.



Figure 4.1 & 4.2 - Performance metric of Xception Model on Blended dataset

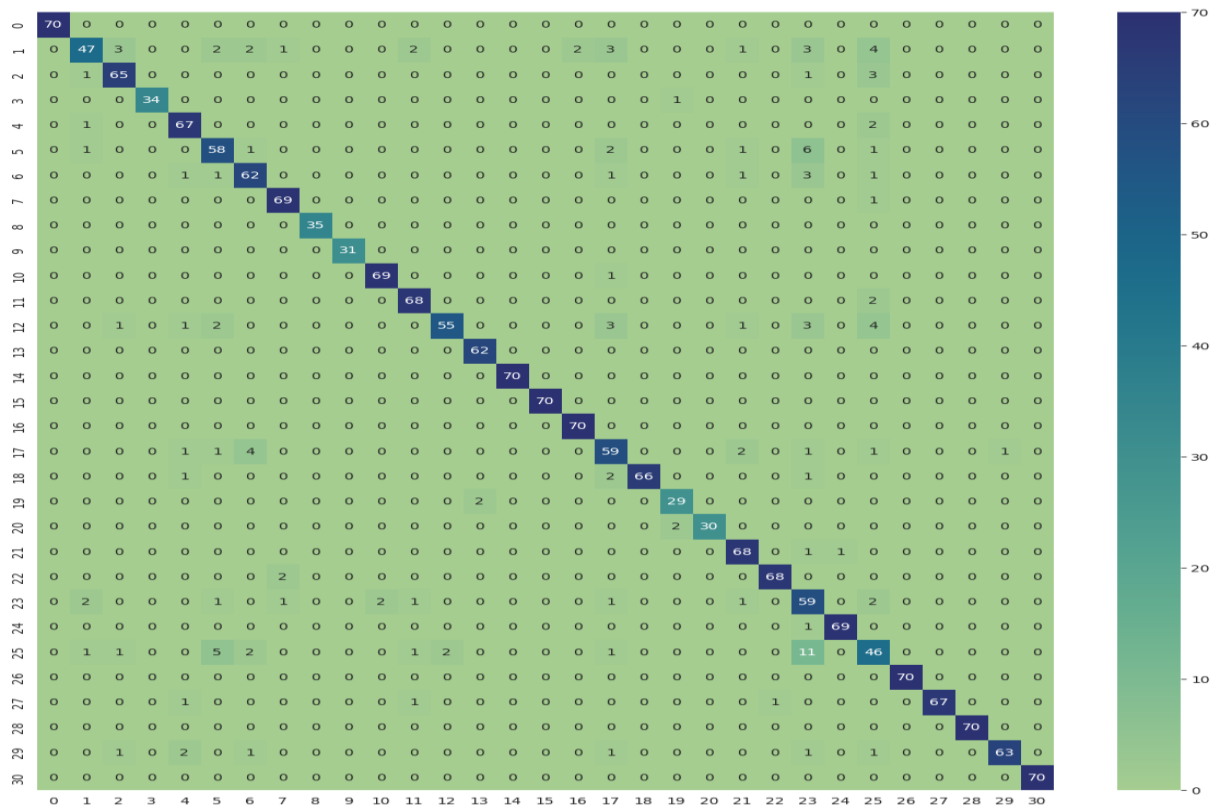


Figure 4.3 - Confusion Matrix of Xception Model on Blended dataset

On Malevis Dataset:

We train the same fine-tuned Xception model on Malevis dataset by changing the input shape of the image that is 200 x 200 x 3. Also changed the final layer to 25 as Malevis data contains 25 classes and callback function of 8 epochs.

	precision	recall	f1-score	support
Adposhel	1.00	0.99	0.99	70
Agent	0.90	0.74	0.81	70
Allaple	0.98	0.91	0.95	70
Amonetize	0.94	0.96	0.95	70
Androm	0.86	0.84	0.85	70
Autorun	0.81	0.94	0.87	70
BrowseFox	1.00	0.99	0.99	70
Dinwod	0.97	1.00	0.99	70
Elex	1.00	0.96	0.98	70
Expiro	0.85	0.89	0.87	70
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	1.00	1.00	70
Hlux	1.00	1.00	1.00	70
Injector	0.76	0.84	0.80	70
InstallCore	1.00	0.96	0.98	70
MultiPlug	0.94	0.97	0.96	70
Neoreklami	1.00	0.94	0.97	70
Neshta	0.70	0.74	0.72	70
Regrun	0.99	1.00	0.99	70
Sality	0.66	0.79	0.72	70
Snarasite	1.00	1.00	1.00	70
Stantinko	0.99	0.97	0.98	70
...				
accuracy			0.93	1750
macro avg	0.93	0.93	0.93	1750
weighted avg	0.93	0.93	0.93	1750

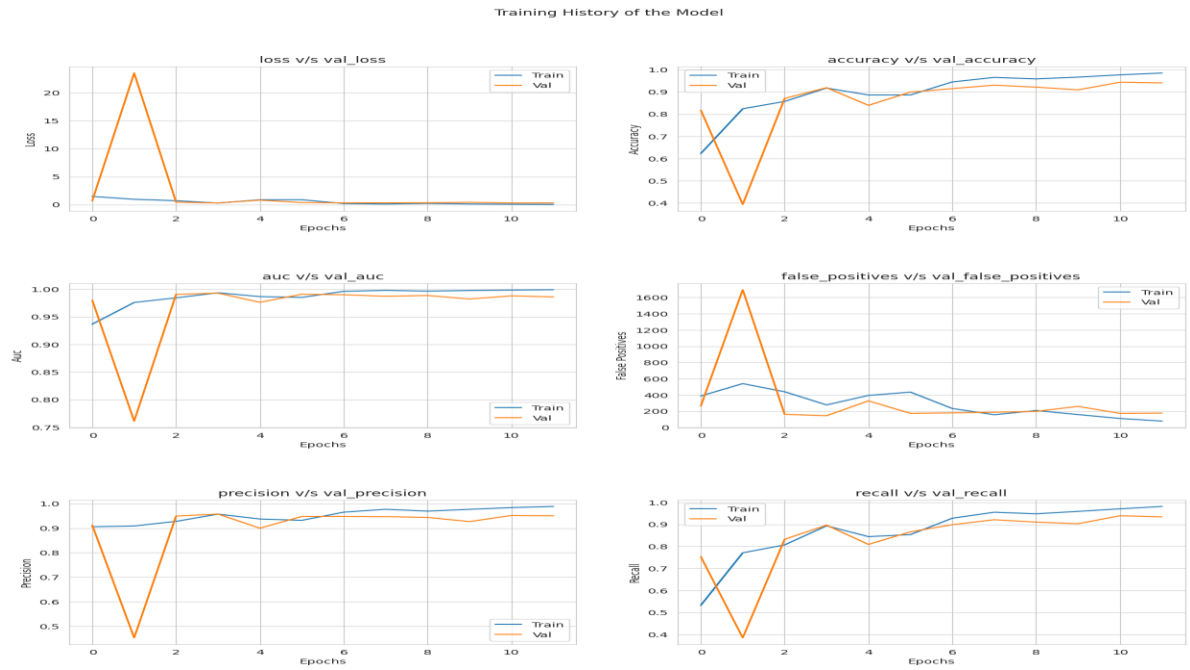


Figure 4.4 & 4.5 - Performance metric of Xception Model on Malevis dataset

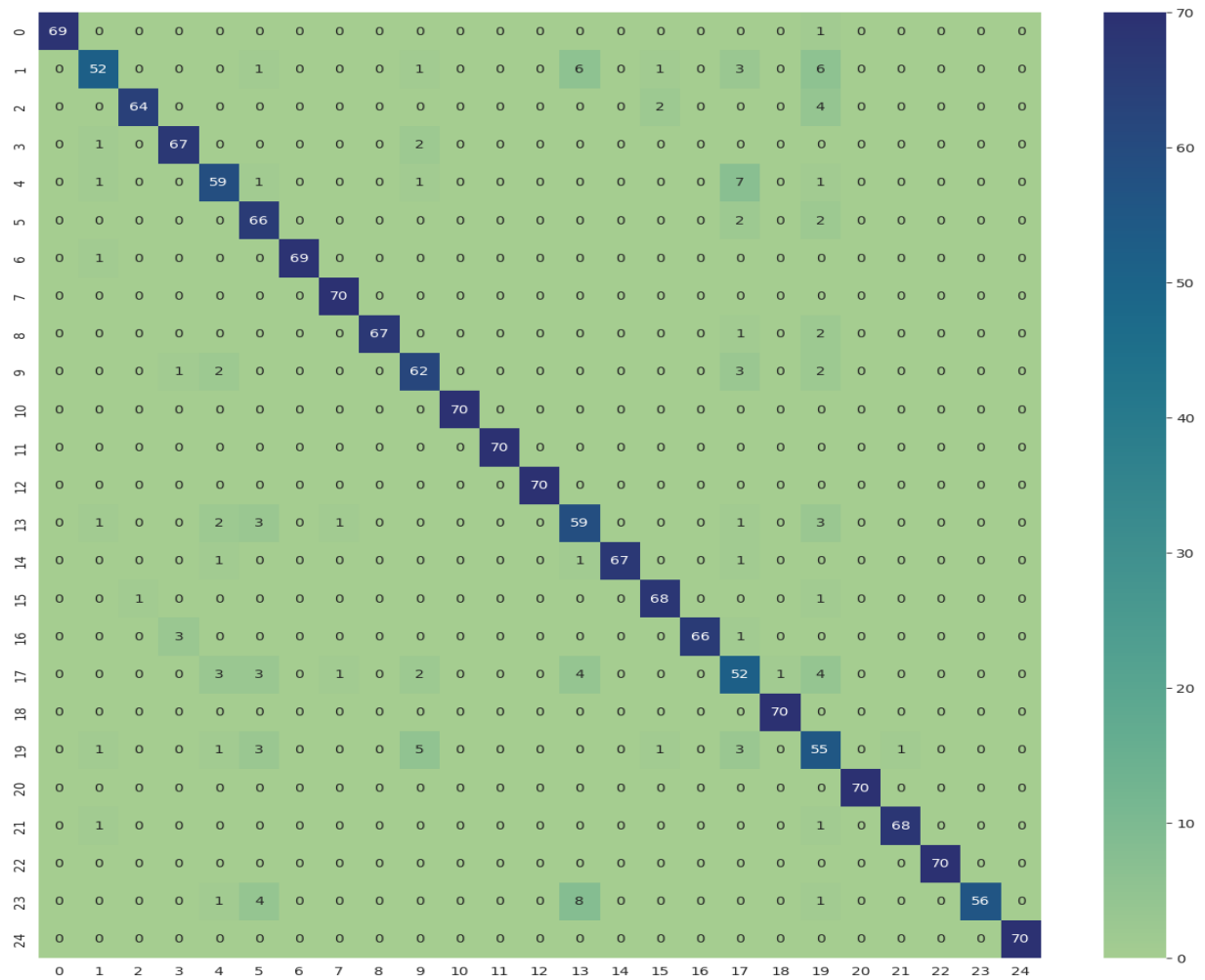


Figure 4.6 - Confusion Matrix of Xception Model on Malevis dataset

4.3 Train on Fine-tuned ResNet50 Model:

On Blended Dataset:

Here we also use the preprocessed and resized 75x75 pixels images from the dataset. Then an ResNet50 model is initialized using `tensorflow.keras.applications.ResNet50`. Then also we initialized it without top layer and its input shape of 75x75x3. The model is fine-tuned using the '**FineTuning**' class, with the initial layers frozen and makes the rest trainable. After that the model adds a Flatten layer(Converts 2D feature maps to 1D), optional Dense and Dropout layers (based on regularization), and a final Dense layer with softmax activation(Outputs probabilities for each of the 31 classes) for classification of the malware classes. Then the model is compiled with the Adam optimizer, categorical cross-entropy loss, and evaluation metrics. Finally the model is trained on the training dataset with a callback function for early stopping after 10 epochs if no improvement in validation loss and validate on validation dataset.

The training begins with the first epoch out of the planned 80 epochs. Initially, the model shows very low accuracy (0.1441) and high loss (7.2469) on the training set. The validation accuracy is significantly higher (0.7721), indicating that the model might initially overfit or learn rapidly from the validation set. As training progresses, the training accuracy improves significantly. By Epoch 2, training accuracy jumps to 0.9396, and validation accuracy increases to 0.9123. By Epoch 5, training accuracy reaches 0.9926, but validation accuracy fluctuates, showing signs of instability.

By Epoch 10, training accuracy is very high at 0.9947, but validation accuracy shows some decline and instability, potentially due to overfitting or data issues. The model uses early stopping based on validation loss with patience set to 10 epochs. This helps prevent overfitting by stopping training when validation performance stops improving. Thus, training stops after 13 epochs.

	precision	recall	f1-score	support
Adposhel	1.00	1.00	1.00	70
Agent	0.82	0.84	0.83	70
Allaple	0.94	0.86	0.90	70
Alueron.gen!J	1.00	1.00	1.00	35
Amonetize	0.96	0.97	0.96	70
Androm	0.89	0.89	0.89	70
Autorun	0.91	0.87	0.89	70
BrowseFox	0.97	0.99	0.98	70
C2LOP.gen!g	1.00	1.00	1.00	35
Dialplatform.B	1.00	1.00	1.00	31
Dinwod	1.00	1.00	1.00	70
Elex	0.96	0.97	0.96	70
Expiro	0.73	0.94	0.83	70
Fakerean	1.00	1.00	1.00	62
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	1.00	1.00	70
Hlux	1.00	1.00	1.00	70
Injector	0.89	0.81	0.85	70
InstallCore	1.00	0.97	0.99	70
Lolyda.AA1	0.94	1.00	0.97	31
Lolyda.AA2	1.00	0.94	0.97	32
MultiPlug	0.97	0.97	0.97	70
...				
accuracy			0.94	1976
macro avg	0.95	0.95	0.95	1976
weighted avg	0.94	0.94	0.94	1976

Training History of the Model

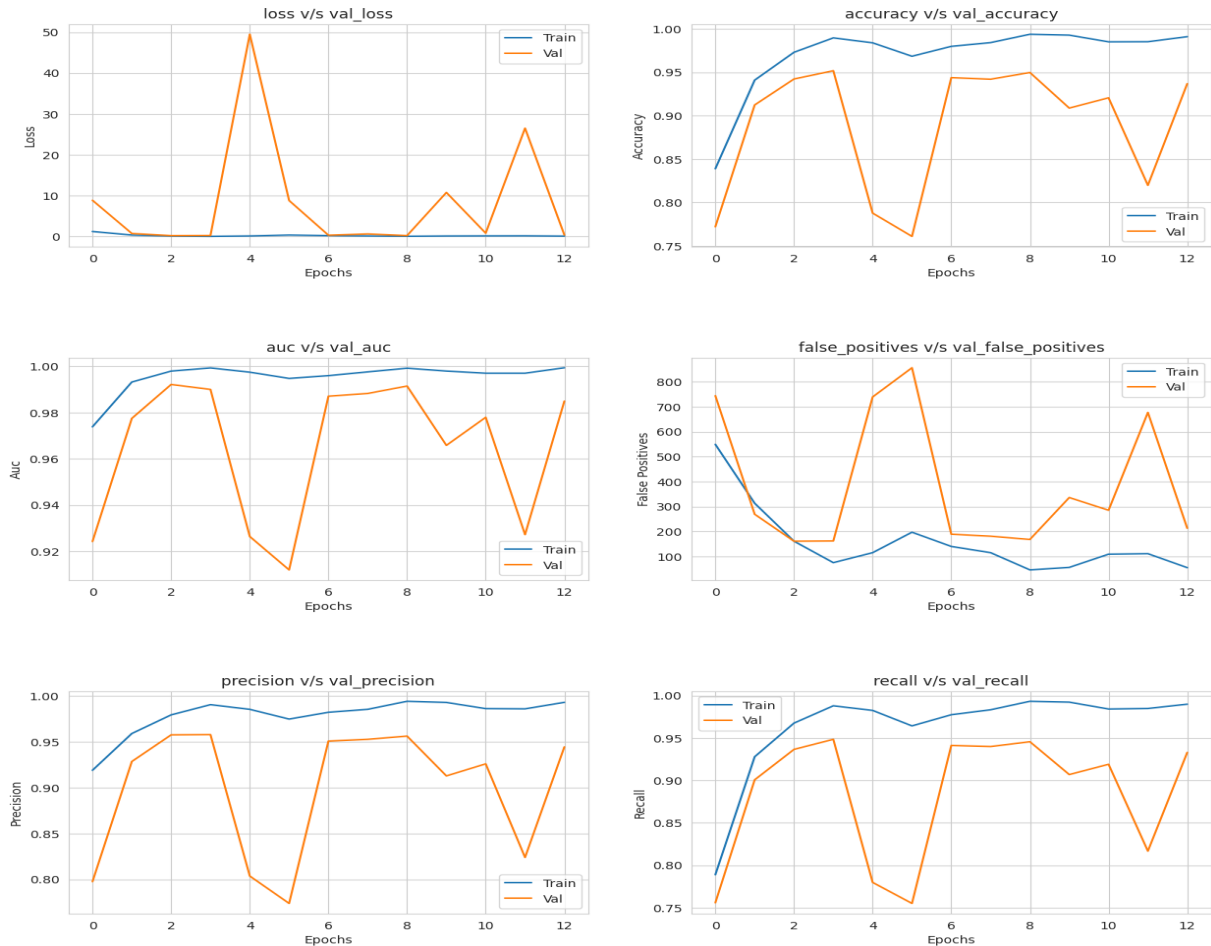


Figure 4.7 & 4.8 - Performance metric of ResNet50 Model on Blended dataset

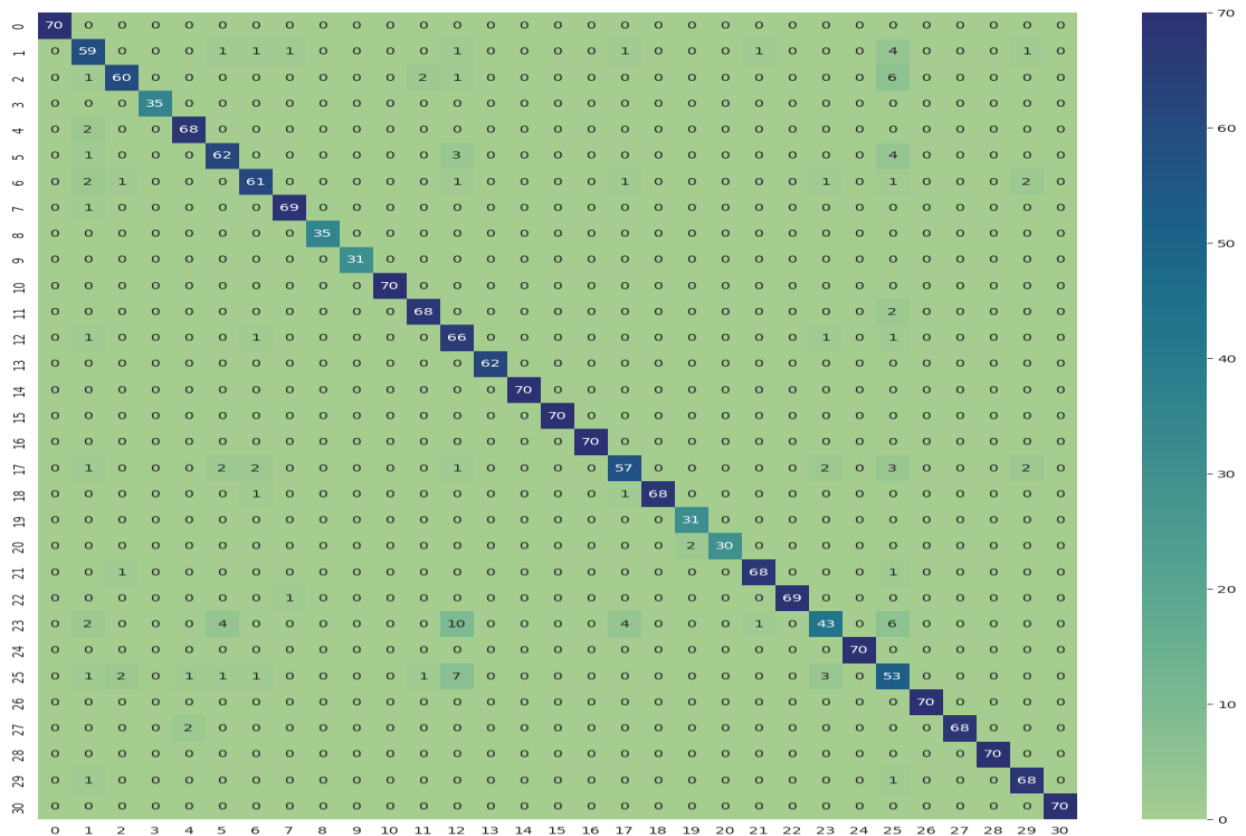


Figure 4.9 - Confusion Matrix of ResNet50 Model on Blended dataset

On Malevis Dataset:

We train the same fine-tuned ResNet50 model on Malevis dataset by changing the input shape of the image that is 200X200X3. Also changed the final dense layer to 25 as Malevis data contains 25 classes and callback function to 8 epochs.

	precision	recall	f1-score	support
Adposhel	0.99	1.00	0.99	70
Agent	0.89	0.79	0.83	70
Allapple	0.91	1.00	0.95	70
Amonetize	0.92	0.97	0.94	70
Androm	0.87	0.94	0.90	70
Autorun	0.90	0.90	0.90	70
BrowseFox	0.97	0.99	0.98	70
Dinwod	1.00	1.00	1.00	70
Elex	0.99	1.00	0.99	70
Expiro	0.96	0.93	0.94	70
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	1.00	1.00	70
Hlux	1.00	1.00	1.00	70
Injector	0.87	0.93	0.90	70
InstallCore	0.97	0.99	0.98	70
MultiPlug	0.99	0.94	0.96	70
Neoreklami	1.00	1.00	1.00	70
Neshta	0.89	0.90	0.89	70
Regrun	1.00	1.00	1.00	70
Salaty	0.74	0.61	0.67	70
Snarasite	1.00	1.00	1.00	70
Stantinko	1.00	1.00	1.00	70
...				
accuracy			0.95	1750
macro avg	0.95	0.95	0.95	1750
weighted avg	0.95	0.95	0.95	1750

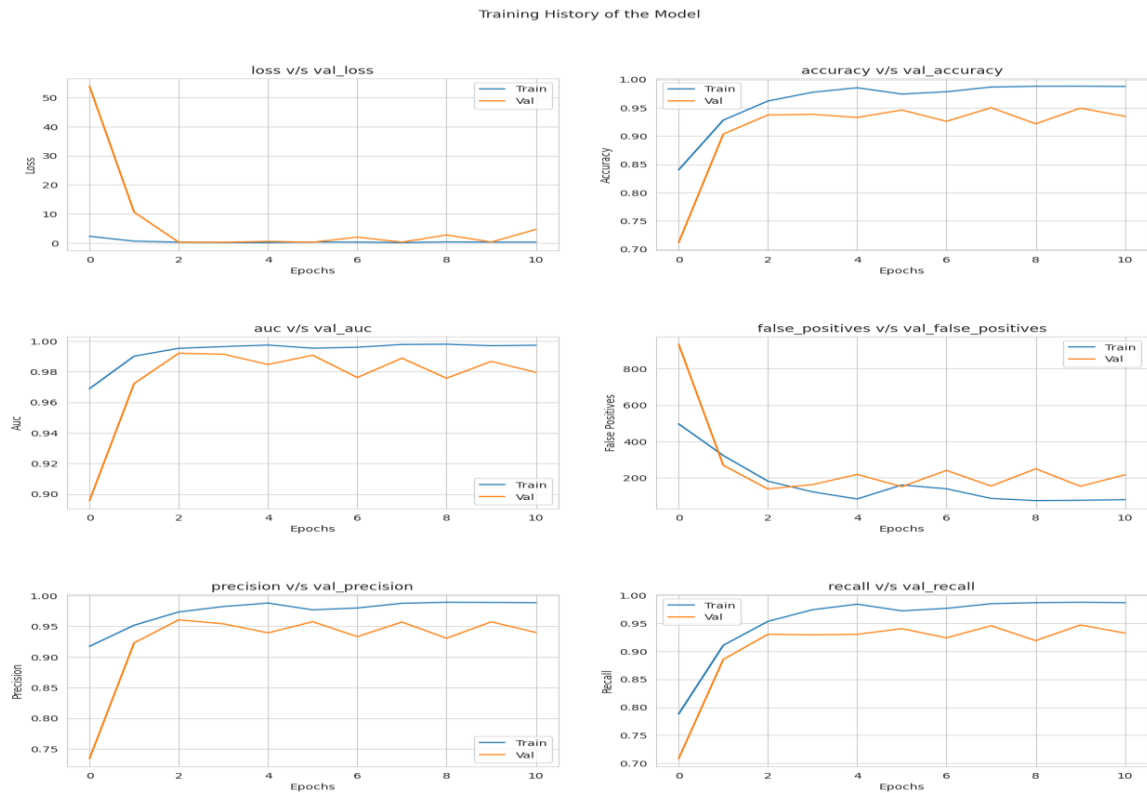


Figure 4.10 & 4.11 - Performance metric of ResNet50 Model on Malevis dataset

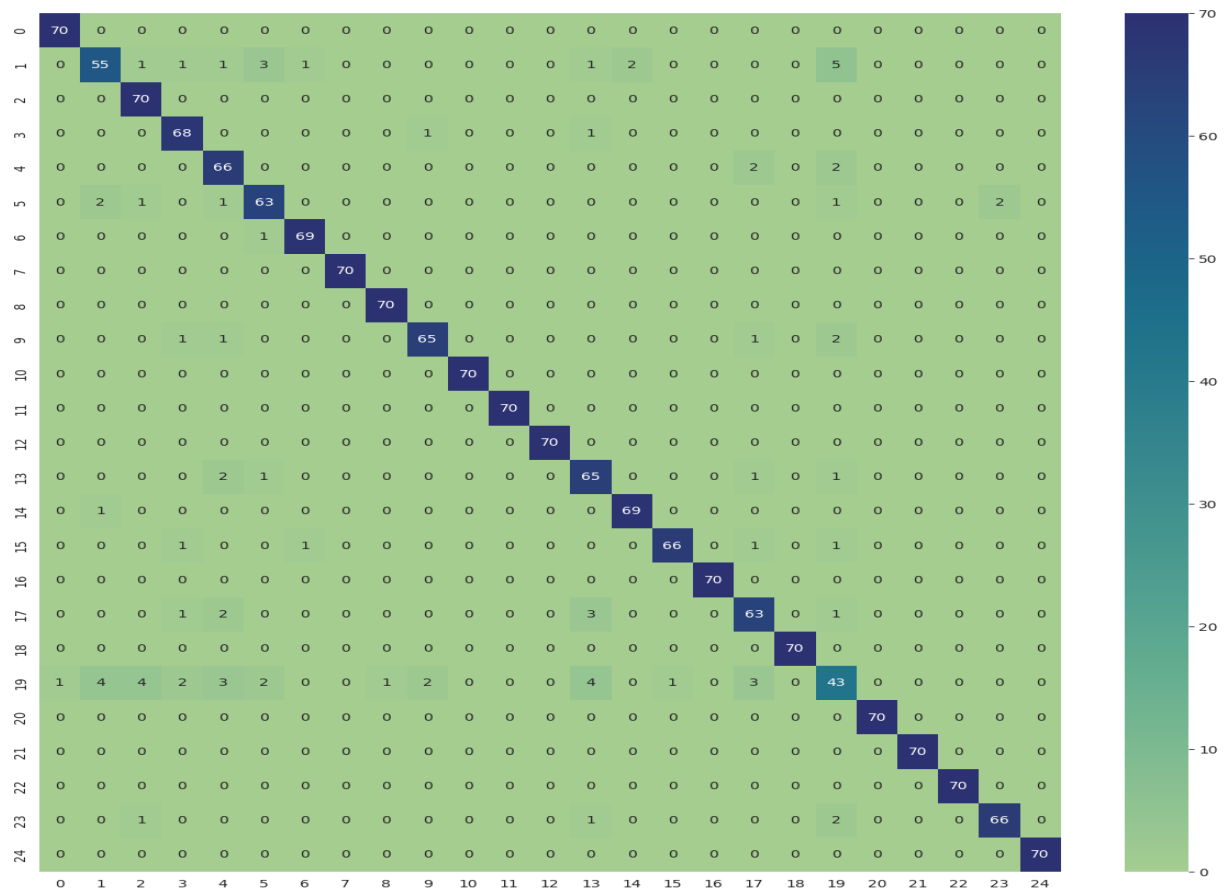


Figure 4.12 - Confusion Matrix of ResNet50 Model on Malevis dataset

4.4 Train on fine-tuned DenseNet-169 Model:

On Blended dataset:

	precision	recall	f1-score	support
Adposhel	1.00	1.00	1.00	70
Agent	0.81	0.80	0.81	70
Allaple	0.94	0.94	0.94	70
Alueron.gen!J	1.00	1.00	1.00	35
Amonetize	0.97	0.96	0.96	70
Androm	0.82	0.86	0.84	70
Autorun	0.86	0.90	0.88	70
BrowseFox	0.95	1.00	0.97	70
C2LOP.gen!g	1.00	1.00	1.00	35
Dialplatform.B	1.00	1.00	1.00	31
Dinwod	0.97	0.99	0.98	70
Elex	0.99	0.96	0.97	70
Expiro	0.72	0.97	0.83	70
Fakerean	1.00	1.00	1.00	62
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	0.99	0.99	70
Hlux	1.00	1.00	1.00	70
Injector	0.87	0.83	0.85	70
InstallCore	1.00	0.97	0.99	70
Lolyda.AA1	0.94	1.00	0.97	31
Lolyda.AA2	1.00	0.94	0.97	32
MultiPlug	1.00	0.91	0.96	70
Neoreklami	0.99	0.99	0.99	70
...				
accuracy			0.94	1976
macro avg	0.95	0.94	0.94	1976
weighted avg	0.94	0.94	0.94	1976

Training History of the Model

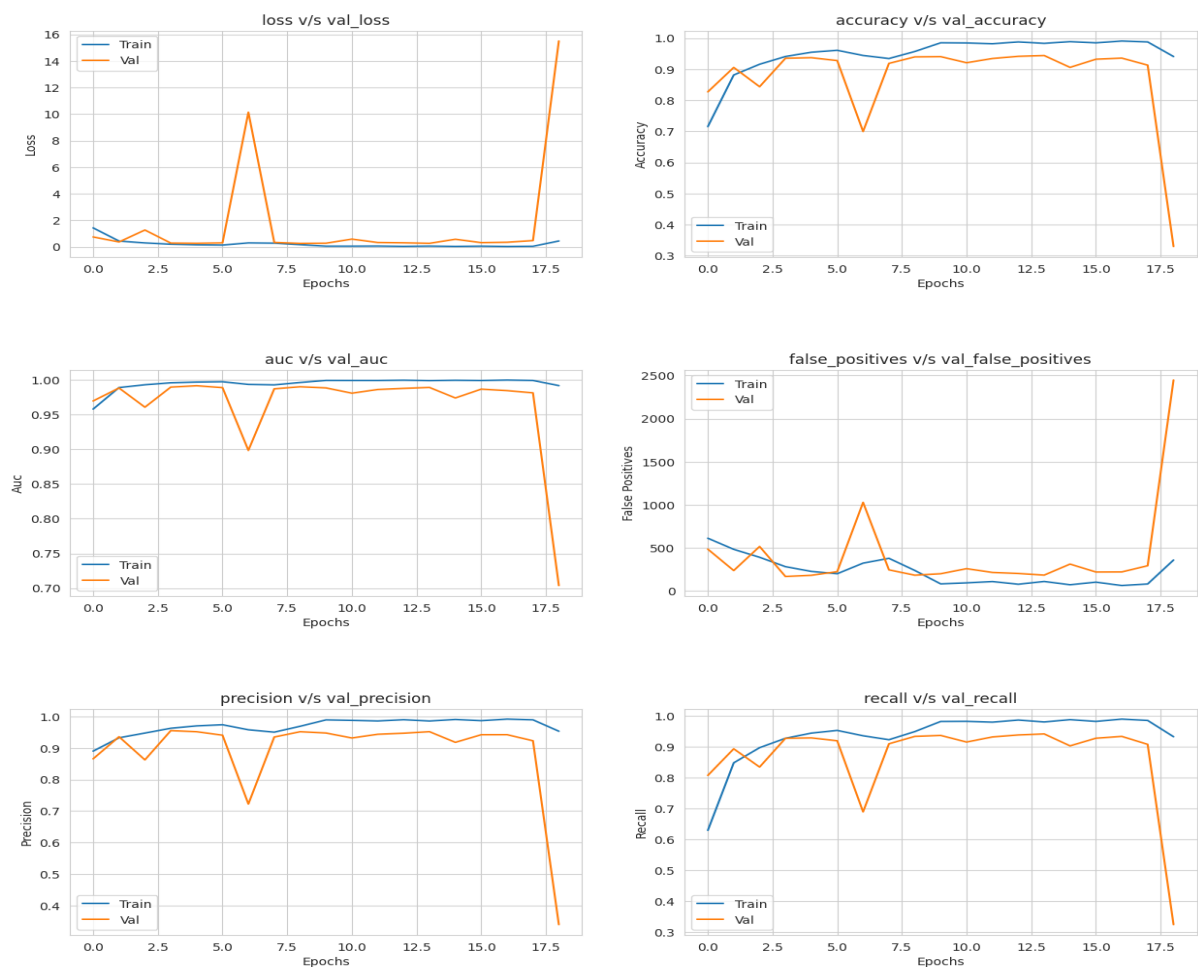


Figure 4.13 & 4.14 - Performance metric of DenseNet-169 Model on Blended dataset

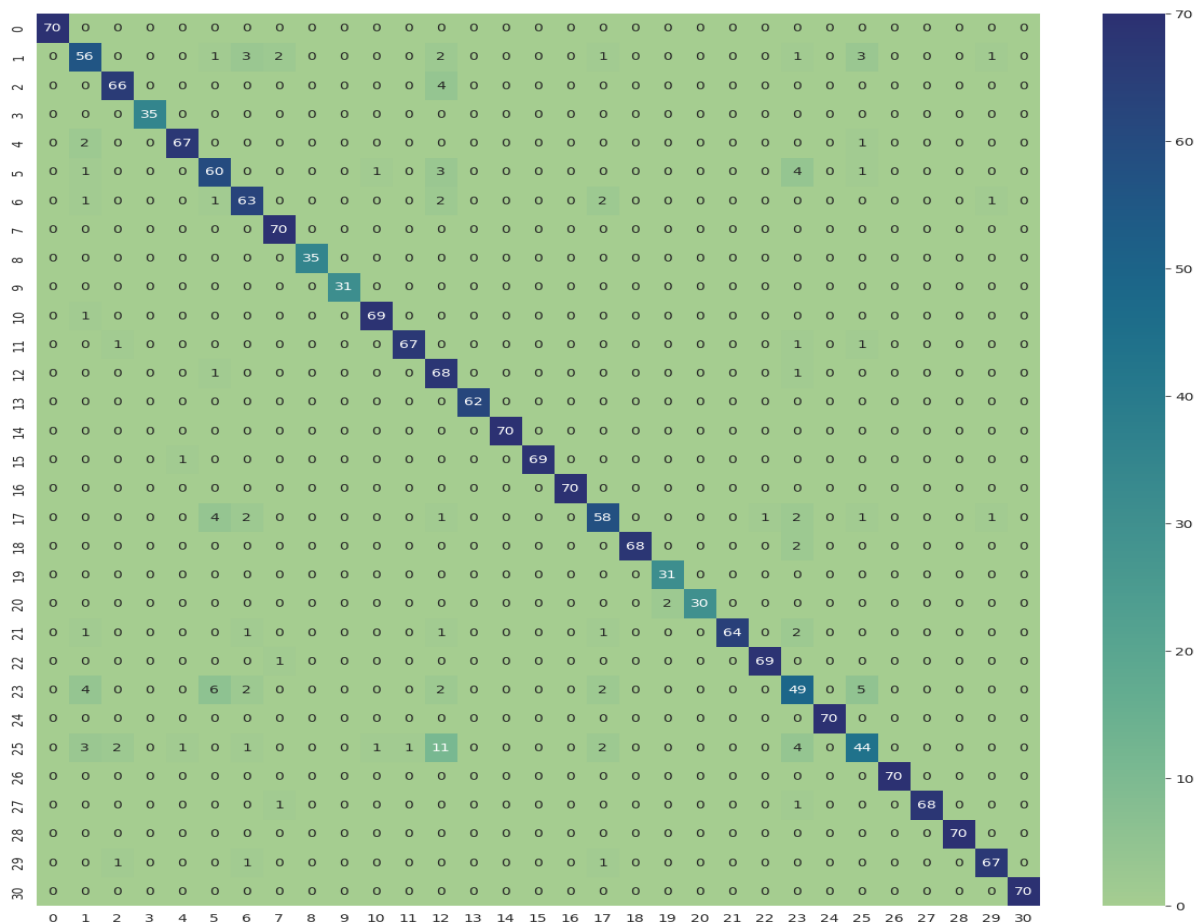


Figure 4.15 - Confusion Matrix of DenseNet-169 Model on Blended dataset

On Malevis Dataset:

	precision	recall	f1-score	support
Adposhel	1.00	1.00	1.00	70
Agent	0.88	0.84	0.86	70
Allaple	0.90	1.00	0.95	70
Amonetize	0.99	0.96	0.97	70
Androm	0.84	0.90	0.87	70
Autorun	0.89	0.93	0.91	70
BrowseFox	0.96	1.00	0.98	70
Dinwod	0.96	1.00	0.98	70
Elex	1.00	1.00	1.00	70
Expiro	0.93	0.96	0.94	70
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	1.00	1.00	70
Hlux	1.00	1.00	1.00	70
Injector	0.92	0.86	0.89	70
InstallCore	1.00	1.00	1.00	70
MultiPlug	0.97	0.97	0.97	70
Neoreklami	0.99	0.99	0.99	70
Neshta	0.87	0.84	0.86	70
Regrun	0.99	1.00	0.99	70
Sality	0.84	0.73	0.78	70
Snarasite	1.00	1.00	1.00	70
Stantinko	1.00	1.00	1.00	70
...				
accuracy			0.96	1750
macro avg	0.96	0.96	0.96	1750
weighted avg	0.96	0.96	0.96	1750

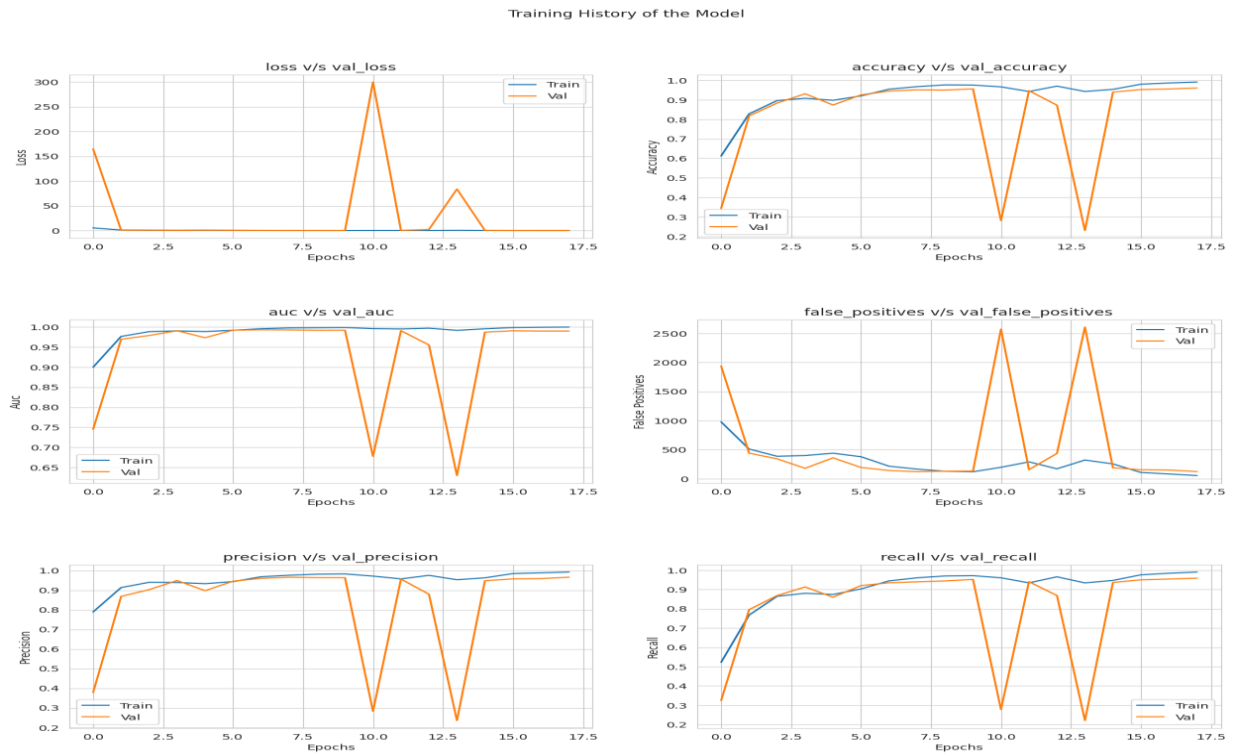


Figure 4.16 & 4.17 - Performance metric of DenseNet-169 Model on Malevis dataset

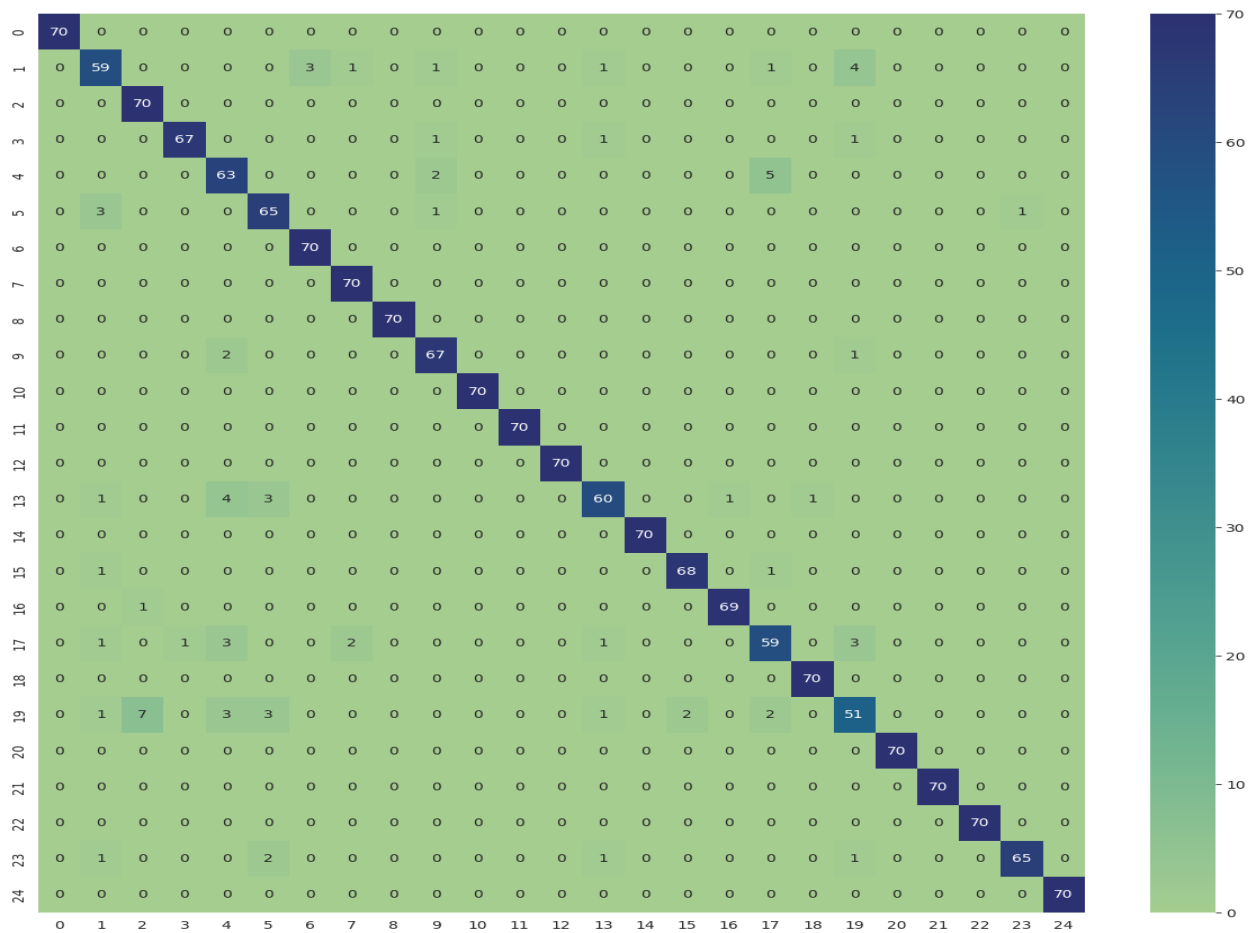


Figure 4.18 - Confusion Matrix of DenseNet-169 Model on Malevis dataset

In the same way we trained InceptionResNetv2 and VGG16 using the above mentioned ‘**FineTuning**’ class on both datasets. The results of the models given below:

On Blended Dataset:

	precision	recall	f1-score	support
Adposhel	1.00	0.99	0.99	70
Agent	0.90	0.74	0.81	70
Allaple	0.91	0.86	0.88	70
Alueron.gen!J	1.00	1.00	1.00	35
Amonetize	0.94	0.91	0.93	70
Androm	0.83	0.69	0.75	70
Autorun	0.87	0.86	0.86	70
BrowseFox	0.92	0.99	0.95	70
C2LOP.gen!g	0.92	1.00	0.96	35
Dialplatform.B	1.00	1.00	1.00	31
Dinwod	0.97	1.00	0.99	70
Elex	0.97	0.94	0.96	70
Expiro	0.80	0.63	0.70	70
Fakerean	1.00	0.95	0.98	62
Fasong	1.00	1.00	1.00	70
HackKMS	0.91	1.00	0.95	70
Hlux	1.00	1.00	1.00	70
Injector	0.96	0.76	0.85	70
InstallCore	0.99	0.97	0.98	70
Lolyda.AA1	0.97	1.00	0.98	31
Lolyda.AA2	0.97	0.97	0.97	32
MultiPlug	0.96	0.91	0.93	70
...				
accuracy			0.90	1976
macro avg	0.92	0.91	0.91	1976
weighted avg	0.92	0.90	0.91	1976

Figure 4.19 - Performance metric of VGG16 Model on Blended dataset

	precision	recall	f1-score	support
Adposhel	1.00	1.00	1.00	70
Agent	0.94	0.71	0.81	70
Allaple	0.97	0.97	0.97	70
Alueron.gen!J	1.00	1.00	1.00	35
Amonetize	0.96	0.96	0.96	70
Androm	0.76	0.83	0.79	70
Autorun	0.85	0.96	0.90	70
BrowseFox	0.99	0.99	0.99	70
C2LOP.gen!g	1.00	1.00	1.00	35
Dialplatform.B	1.00	0.97	0.98	31
Dinwod	0.97	1.00	0.99	70
Elex	0.99	0.99	0.99	70
Expiro	0.66	0.91	0.77	70
Fakerean	0.98	1.00	0.99	62
Fasong	1.00	1.00	1.00	70
HackKMS	1.00	1.00	1.00	70
Hlux	0.95	1.00	0.97	70
Injector	0.87	0.77	0.82	70
InstallCore	0.99	0.99	0.99	70
Lolyda.AA1	0.91	1.00	0.95	31
Lolyda.AA2	1.00	0.91	0.95	32
MultiPlug	0.98	0.93	0.96	70
...				
accuracy			0.93	1976
macro avg	0.94	0.94	0.94	1976
weighted avg	0.94	0.93	0.93	1976

Figure 4.20 - Performance metric of InceptionResNetV2 Model on Blended dataset

On Malevis dataset:

	precision	recall	f1-score	support
Adposhel	0.99	1.00	0.99	70
Agent	0.85	0.73	0.78	70
Allaple	0.94	0.93	0.94	70
Amonetize	0.96	0.93	0.94	70
Androm	0.90	0.74	0.81	70
Autorun	0.75	0.91	0.83	70
BrowseFox	0.92	0.97	0.94	70
Dinwod	0.75	1.00	0.86	70
Elex	0.77	1.00	0.87	70
Expiro	0.76	0.73	0.74	70
Fasong	1.00	1.00	1.00	70
HackKMS	0.96	1.00	0.98	70
Hlux	1.00	1.00	1.00	70
Injector	0.79	0.80	0.79	70
InstallCore	1.00	1.00	1.00	70
MultiPlug	0.92	0.96	0.94	70
Neoreklami	0.93	0.96	0.94	70
Neshta	0.70	0.44	0.54	70
Regrun	1.00	1.00	1.00	70
Sality	0.70	0.57	0.63	70
Snarasite	1.00	1.00	1.00	70
Stantinko	0.99	0.97	0.98	70
...				
accuracy			0.90	1750
macro avg	0.90	0.90	0.90	1750
weighted avg	0.90	0.90	0.90	1750

Figure 4.21 - Performance metric of VGG16 Model on Malevis dataset

	precision	recall	f1-score	support
Adposhel	1.00	1.00	1.00	70
Agent	0.81	0.79	0.80	70
Allaple	0.93	1.00	0.97	70
Amonetize	0.93	0.96	0.94	70
Androm	0.91	0.89	0.90	70
Autorun	0.84	0.90	0.87	70
BrowseFox	0.99	0.99	0.99	70
Dinwod	1.00	1.00	1.00	70
Elex	0.99	0.97	0.98	70
Expiro	0.77	0.91	0.84	70
Fasong	1.00	1.00	1.00	70
HackKMS	0.97	1.00	0.99	70
Hlux	1.00	1.00	1.00	70
Injector	0.91	0.83	0.87	70
InstallCore	0.99	0.99	0.99	70
MultiPlug	0.94	0.96	0.95	70
Neoreklami	1.00	1.00	1.00	70
Neshta	0.82	0.77	0.79	70
Regrun	1.00	1.00	1.00	70
Sality	0.80	0.64	0.71	70
Snarasite	1.00	1.00	1.00	70
Stantinko	0.97	0.97	0.97	70
...				
accuracy			0.94	1750
macro avg	0.94	0.94	0.94	1750
weighted avg	0.94	0.94	0.94	1750

Figure 4.22 - Performance metric of InceptionResNetV2 Model on Malevis dataset

4.5 Comparison of models:

In our experiment we trained five(Xception, ResNet50, VGG16, DenseNet-169 and InceptionResNetv2) pre-train CNN models after some fine-tuning on Blended and Malevis Dataset for malware classification. We compare the performance of the models on both datasets.

Comparison on Blended dataset:

Comparison of Model Performance on Blended dataset

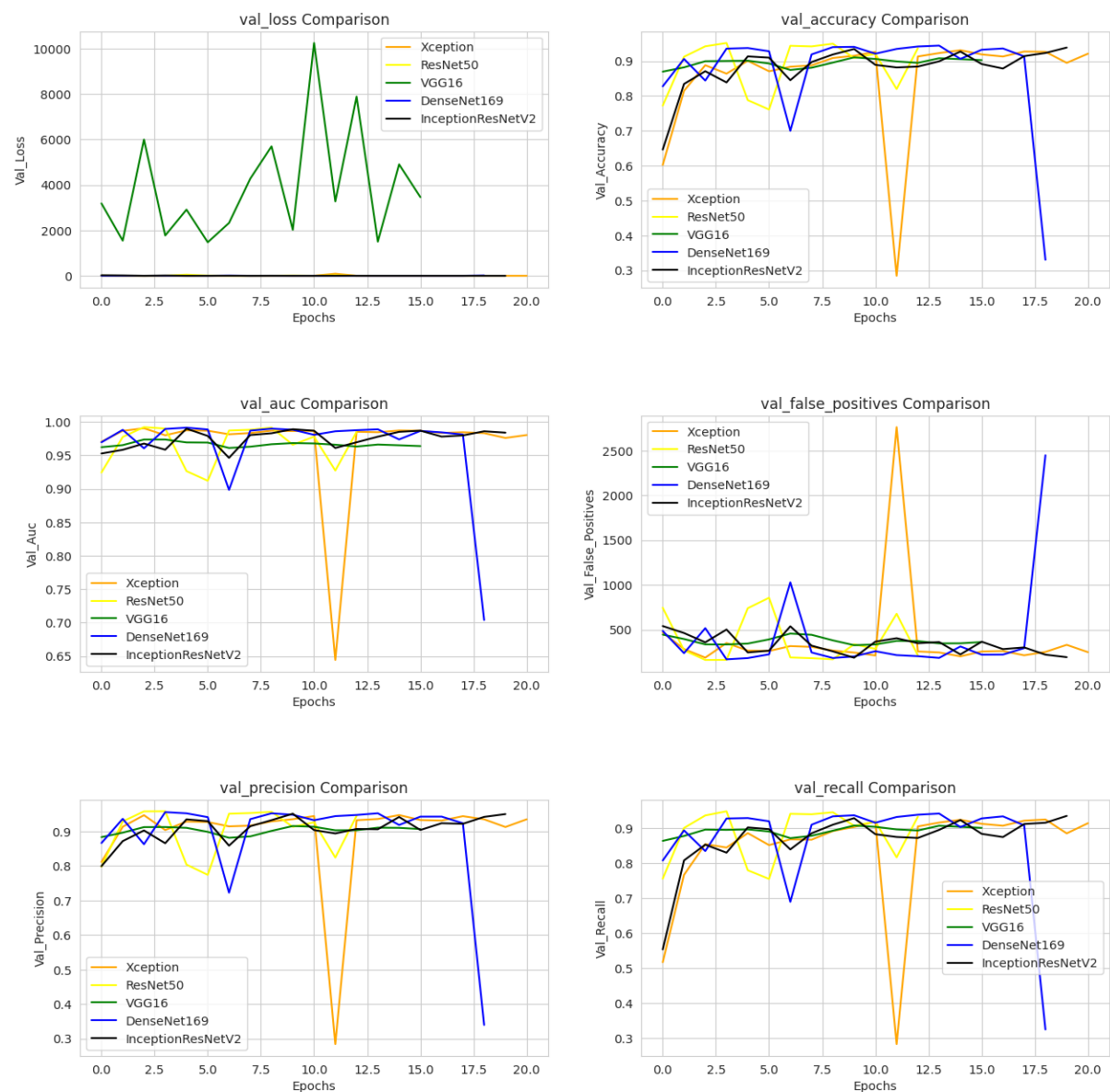


Figure 4.23

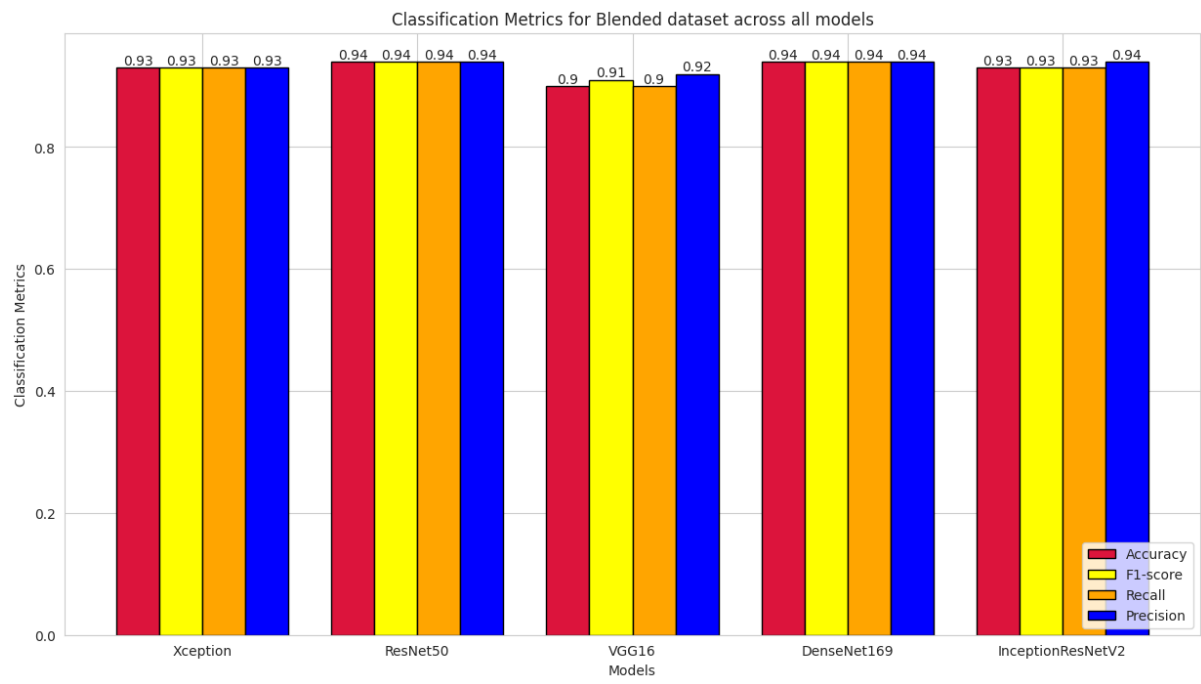


Figure 4.24

After comparison we found that Densenet-169 and ResNet50 perform better on Blended dataset with accuracy of 94%. Then Xception and inceptionResNetV2 both models accuracy is 93%. VGG16 is worst performing model with accuracy of 90%.

Comparison on Malevis dataset:

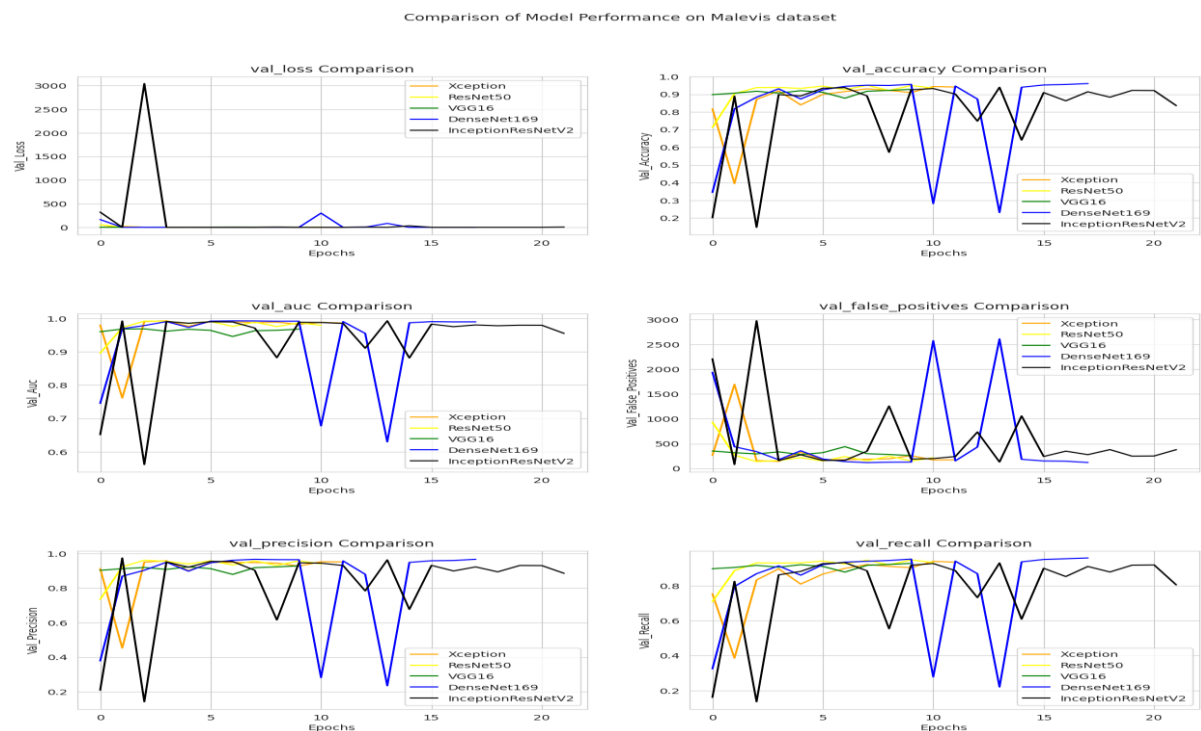


Figure 4.25

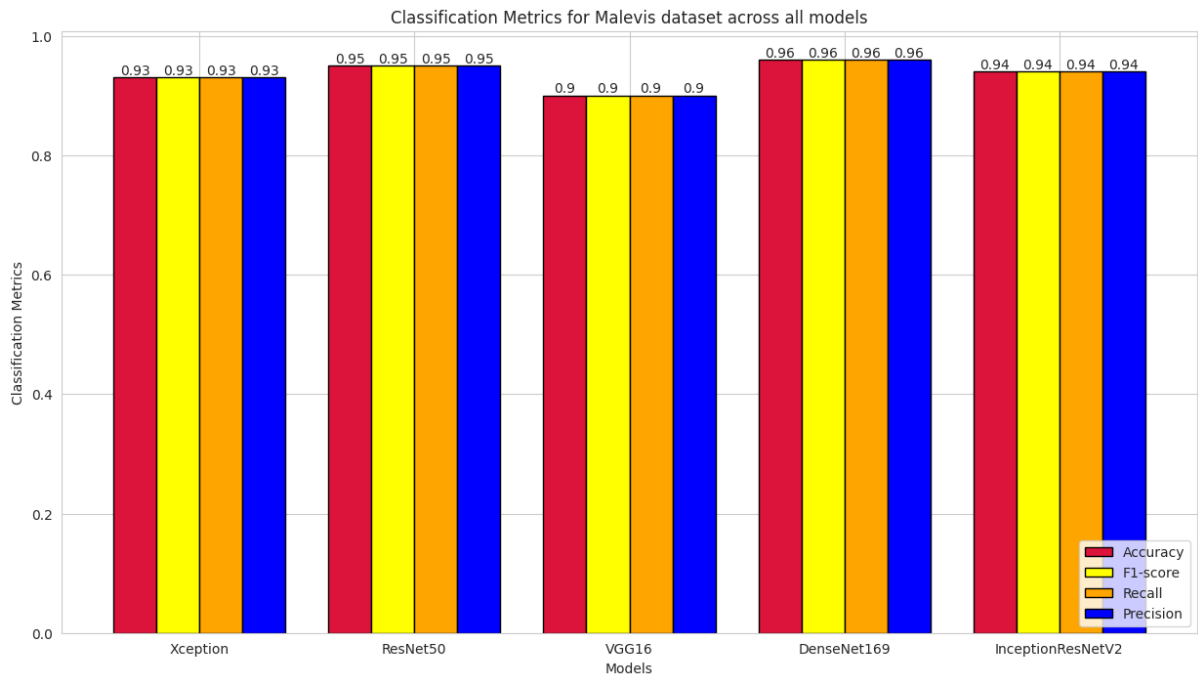


Figure 4.26

After comparison we found that Densenet-169 perform best on Malevis dataset with accuracy of 96%. Then ResNet50 with accuracy of 95% and inceptionResNetV2 with accuracy is 94%. After that Xception model with accuracy of 93%. VGG16 is worst performing model with accuracy of 90%.

Conclusion and Future Work

In this project, we investigated the performance of several fine-tuned deep convolutional neural network (DCNN) models for the task of malware classification. Using a blended dataset and the Malevis dataset, we evaluated the models Xception, ResNet50, DenseNet-169, InceptionResNetV2, and VGG16. Our evaluation was based on standard performance metrics such as accuracy, precision, recall, and F1-score.

Our experiments demonstrated that the choice of model architecture significantly impacts classification performance. Specifically, DenseNet-169 and ResNet50 consistently achieved the highest accuracy on both datasets. DenseNet-169 outperformed all other models on the Malevis dataset with an accuracy of 96%, while ResNet50 performed equally well on the blended dataset with an accuracy of 94%.

Xception and InceptionResNetV2 also showed strong performance, achieving accuracies of 93% on both datasets, slightly lower than DenseNet-169 and ResNet50. VGG16, however, was the least effective model in our experiments, achieving a maximum accuracy of 90%.

These results suggest that DenseNet-169 and ResNet50 are particularly well-suited for malware classification tasks, likely due to their deeper architectures and more effective feature extraction capabilities. The use of early stopping in training helped prevent overfitting and ensured that the models generalized well to unseen data.

In conclusion, our study highlights the effectiveness of using fine-tuned DCNN models for malware classification. Future work could explore further fine-tuning techniques, model ensembling, and the application of these models to other types of malware datasets to improve robustness and performance even further.

References:

- [1] T. Harmonen, “Identifying polymorphic malware.” Google Patents, 25-Mar-2014.
- [2] Q. Zhang and D. S. Reeves, “Metaaware: Identifying metamorphic malware,” in Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 2007, pp. 411–420.
- [3] <https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide>
- [4] <https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html>
- [5] <https://www.paloaltonetworks.com/cyberpedia/what-is-malware>
- [6] <https://www.mcafee.com/en-us/antivirus/malware.html>
- [7] <https://www.malwarebytes.com/malware>
- [8] Symantec Corporation, Symantec internet security threat report Netw. Secur. (2019).
- [9] A. Shabtai, R. Moskovitch, Y. Elovici, C. Glezer, Detection of malicious code by applying machine learning classifiers on static features: a state-of-the-art survey Inf. Secur. Tech. Rep. (2009).
- [10] J. Su, V. DaniloVasconcellos, S. Prasad, S. Daniele, Y. Feng, K. Sakurai Lightweight classification of iot malware based on image recognition 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2 (2018), pp. 664-669.
- [11] Awan, M. J., Masood, O. A., Mohammed, M. A., Yasin, A., Zain, A. M., Damaševičius, R., & Abdulkareem, K. H. (2021). Image-based malware classification using VGG19 network and spatial convolutional attention. Electronics, 10(19),2444.
- [12] Kalash, M., Rochan, M., Mohammed, N., Bruce, N. D., Wang, Y., & Iqbal, F.(2018, February). Malware classification with deep convolutional neural networks. In 2018 9th IFIP international conference on new technologies, mobility and security (NTMS) (pp. 1-5). IEEE.
- [13] Lo, W. W., Yang, X., & Wang, Y. (2019, June). An xception convolutional neural network for malware classification with transfer learning. In 2019 10th IFIP international conference on new technologies, mobility and security (NTMS) (pp. 1-5).IEEE.

- [14] Singh, A., Handa, A., Kumar, N., & 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* (pp. 75-92). Springer International Publishing.
- [15] Go, J. H., Jan, T., Mohanty, M., Patel, O. P., Puthal, D., & Prasad, M. (2020, July). Visualization approach for malware classification with ResNeXt. In *2020 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1-7). IEEE.
- [16] Ghouti, L., & Imam, M. (2020). Malware classification using compact image features and multiclass support vector machines. *IET Information Security*, 14(4), 419-429.
- [17] Mitsuhashi, R., & Shinagawa, T. (2022, June). Exploring Optimal Deep Learning Models for Image-based Malware Variant Classification. In *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)* (pp. 779-788). IEEE.
- [18] Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., & Zheng, Q. (2020). IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks*, 171, 107138.
- [19] Aslan, O., & Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms. *Ieee Access*, 9, 87936-87951.
- [20] Asam, M., Khan, S. H., Jamal, T., Zahoora, U., & Khan, A. (2021). Malware classification using deep boosted learning. *arXiv preprint arXiv:2107.04008*.
- [21] Awan, M. J., Masood, O. A., Mohammed, M. A., Yasin, A., Zain, A. M., Damaŕsevićius, R., & Abdulkareem, K. H. (2021). Image-based malware classification using VGG19 network and spatial convolutional attention. *Electronics*, 10(19), 2444.
- [22] Mallik, A., Khetarpal, A., & Kumar, S. (2022). ConRec: malware classification using convolutional recurrence. *Journal of Computer Virology and Hacking Techniques*, 18(4), 297-313.
- [23] AlGarni, M. D., AlRoobaea, R., Almotiri, J., Ullah, S. S., Hussain, S., & Umar, F. (2022). An efficient convolutional neural network with transfer learning for malware classification. *Wireless Communications and Mobile Computing*, 2022, 1-8.

- [24] Tekerek, A., & Yapici, M. M. (2022). A novel malware classification and augmentation model based on convolutional neural network. Computers & Security, 112,102515.
- [25] <https://www.kaggle.com/datasets/sohamkumar1703/malevis-dataset>
- [26] <https://www.kaggle.com/datasets/gauravpendharkar/blended-malware-image-dataset>
- [27] <https://www.kaggle.com/c/malware-classification/data?select=train.7z>
- [28] <https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f>
- [29] <https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f>
- [30] <https://medium.com/@saba99/xception-cd1adc84290f>
- [31] <https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918>