A Study of Malware Classification Using Deep Learning

by

Mohammad Muhibur Rahman
19201079
Anushua Ahmed
19201067
Mutasim Husain Khan
19201082
Abrar Jamshed
19201002
Md Hafijur Rahman
19301058

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
September 2023

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Mohammad Muhibur Rahman

Mohammad Muhibur Rahman
19201079

Anushua

Anushua Ahmed
19201067

Mutasim

Mutasim Husain Khan
19201082

Abrar Jamshed.

Abrar Jamshed
19201002

[illegible]

Md Hafijur Rahman
19301058

Approval

The thesis titled "A Study of Malware Classification Using Deep Learning" submitted by

1. Mohammad Muhibur Rahman(19201079)
2. Anushua Ahmed(19201067)
3. Mutasim Husain Khan(19201082)
4. Abrar Jamshed(19201002)
5. Md Hafijur Rahman(19301058)

of Summer, 2023 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on September 17, 2023

Examining Committee:

Supervisor:
(Member)

[illegible]

Dr. Mohammad Kaykobad
Distinguished Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

Ziaul

Dewan Ziaul Karim
Lecturer
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Ethics Statement

The aforementioned remark functions as a disclaimer, asserting that the study article in issue adhered to legal provisions safeguarding the welfare, rights, and dignity of human beings. The data was obtained in a manner that ensured impartiality and fairness, and the outcomes were not manipulated for any discriminatory intentions. The document additionally features citations from many reliable sources, which are given particular emphasis. Furthermore, it upholds the honesty and integrity of the research participants. Our objective is to ensure that our work is directed towards our intended audience, with the aim of maximizing the benefits they receive.

Abstract

Malware represents an intrusive computer program that is engineered by cyber-criminals to destroy computer systems or steal and manipulate sensitive data. Malware classification is crucial to malware detection as it helps to assign malware to a specific category according to its characteristics. Characterizing and labeling variants of spyware is also useful as it will shed light on how they're able to gain access to our systems in the first place, the dangers they possess, and the necessary preventions to take against them. In order to tackle such a serious security-related issue, we have decided to develop an image-processing system that would help us be faster at detecting malware while also possibly being one step ahead of cyber-criminals. To describe and categorize sourced malware datasets, we will develop the system using various approaches for deep learning methods and even propose a simple CNN-based methodology of our own. The aim of our work is to show a comparative study of malware types with experimental results, making it easier to identify and keep track of malware that already exists while helping to detect new ones. To be more specific, we worked with four pre-trained CNN models in order to diversify our methods. These trained models include ResNet-50, Inception-V3, VGG-16, and DenseNet-201. After running and testing all of the models on the Malimg dataset, our suggested model was able to achieve a 97.64% accuracy rate in detecting malware greyscale images. This high level of testing accuracy also slightly outperformed some of the other cutting-edge models used in our comparison study on the dataset. These modern and highly developed models used for comparison include Involution, Vision Transformer (ViT), Compact Convolutional Transformer (CCT), and External Attention Network (EANet). Finally, we employed the use of an explainable artificial intelligence (AI) technique known as LIME to provide a more detailed clarification of the rationale behind our model's selection and classification of individual samples into their respective classes.

Keywords: Malware; Deep Learning; Classification; Neural Network; Convolutional Neural Network (CNN); Transformer; Involution; Explainable AI (XAI); Malware Binary Image;

Acknowledgement

Primary and most important, we express our gratitude to Allah subhanahu wa ta'ala, under whose blessings our thesis has been successfully concluded without significant disruptions.

Additionally, we want to extend our appreciation to our Supervisor and Co-Supervisor, Dr. Mohammad Kaykobad sir, and Mr. Dewan Ziaul Karim sir for their invaluable assistance and insightful advice. This thesis would not have been completed without their assistance. Both of them have dedicated a significant amount of their valuable time to instructing and assisting individuals such as ourselves.

We express appreciation for the collective endeavors of all individuals within the group, who have contributed to the timely completion of our tasks. In particular, the teammates who contributed the most effort and provided leadership in the execution of this thesis project are deserving of recognition, since their contributions were important in ensuring the successful completion of this university thesis. Ultimately, the attainment of our current position may not have been conceivable without the continuous support of our parents. We are presently in the process of completing our academic studies, and we attribute our forthcoming graduation to the benevolent prayers and assistance we have received.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

AI	Artificial Intelligence
ANN	Artificial Neural Network
CCT	Compact Convolutional Transformer
CNN	Convolutional Neural Network
DT	Decision Tree
EANet	External Attention Network
KNN	K - Nearest Neighbor
LIME	local Interpretable Model-agnostic Explanations
NB	Naive Bayes
RF	Random Forest
RNN	Recurrent Neural Network
SVM	Support Vector Machine
USB	Universal Serial Bus
ViT	Vision Transformer
XAI	Explainable Artificial Intelligence

Chapter 1

Introduction

1.1 Background Information

Malware uses deceit to inhibit a system from functioning correctly. Cybercriminals benefit from the scenario by deploying additional attacks, acquiring credentials, obtaining user information selling, making it available as cloud resources, or making the victims go through financial losses. They do this after acquiring access to a system via a number of methods, such as a spam email, contaminated file, system or firmware weakness, corrupted USB drive, or malignant web application.

Malware attacks can happen to anyone. Even though some people may be able to recognize specific tactics attackers use to target victims with malware, such as being able to identify a spam email, malicious hackers are professional and persistently improving their techniques to stay up with advancements in privacy and techniques. Depending on the type of malware, malware attacks also have different appearances and behaviors. This type of malware is designed to blend in and remain undetected for as long as possible. A victim of a rootkit attack, for example, may not even be aware of it.

Since ancient times, fraudsters have utilized variants of techniques to infect the maximum number of systems with malware. According to [32], Elk Cloner, the discovery of the first computer malware was made in 1982 and it was found on a Mac. This detection was quickly followed by the launch of the first computer virus for IBM computers in 1986. The name of this virus was “Brain”.

According to [32], in the late 1980s, most of the malignant programs were hidden in floppy disks and distributed as basic boot sectors and file-infecting viruses. As computer networks began to be adopted and expanded in the early 1990s, so did the amount of malicious software as distribution became easier.

As technologies standardize, the transmission of some malware types is facilitated. The increased use of email contributed to the expansion of macro viruses that target Microsoft Office products and enabled the distribution of spyware via email links. According to [32], in the midst of the 1990s, people started to realize that the transmission of malware was now majorly network-driven as enterprises began to suffer increasingly due to mostly macro viruses.

Chapter 2

Problem Statement

Every aspect of the world is now digitalized. Concepts like online banking, cryptocurrency, e-governance, etc. all emerged in order to fully utilize the possibilities of the Internet. However, the downside is that all sectors now concern themselves with data protection and security issues. [62] It is estimated that ransomware attacks will reach a rate of once every 11 seconds by 2022 and due to this, $20 billion of loss will be recorded globally in a year. According to [63] in 2018, 39% of malware victims cooperated with cybercriminals' ransom demands, but this year, that percentage is expected to climb to 58%. As more of these incidents happen, more resources will be available to attackers. This will help them to increase the number of attacks of a more severe nature. Corporate and IT sectors are making great efforts to identify malicious files and prevent security breaches. At the same time, more malicious files are being constantly created and modified. According to [41] in the first three-fourths of 2022, a record of 62.29 million unseen malware samples were discovered throughout all operating systems. This corresponds to around 228,164 daily malware attacks. This huge number and variants of malware make classifying it hard. Furthermore, one malware can exhibit identifying functions of multiple families. This can make grouping malware into their respective families even more confusing. [37] For example, malware can infect through not only an email attachment but via P2P networks as well. Based on these differences in functionality, the application may be categorized as a Mass-mailer,a Trojan-Mail finder, or a P2P-Worm. Consequently, all these factors make traditional ways of assorting malware non-precise. So, we have come up with a solution that should be a more effective and efficient way to detect and identify malware. This is useful not only for spotting previously unknown malware but also for identifying and classifying well-known threats. Specifically, the idea is to compare the similarities and dissimilarities between malware that we already know of to get a better understanding of the threats they pose. This will help us cope with newer threats more expertly and take the right action as a whole.

To be more exact we are going to test a number of existing deep learning neural network models available and benchmark them and produce a study that will help us to create our own novel model using CNN so that we can evaluate and compare them to get the best results. In that way, our model will contribute to building better systems and help eliminate threats by identifying and classifying malware more efficiently.

Chapter 3

Literature Review

In this literature review, a comparative study of existing works in Malware Classification using Deep Learning is conducted. A number of traditional Deep Learning approaches are studied which showed exemplary performance in reaching similar goals. Malware classification, detection, and identification can gain a lot of advantages from the traditional Deep Learning techniques because of their flexibility of frameworks which makes them an efficient tool for handling expert decisions. Neural networks can help in this situation as they deal with mathematical models. In this line of work, neural networks can help us achieve our expected goals as they will provide methodical tools.

Anti-malware organizations initialized developing comparatively complex models depending on data mining and deep learning architectures [4]. These architectures use various data representation models to develop better malware identification systems. Commonly SVM classifier [10][3], Naïve Bayes [1] or multiple classifiers (decision tree combining with Naïve Bayes and SVM) [2] are used. In the case of [40], the model is trained on converted images with a resolution of 112 $\times$112 and 56 $\times$56 using SVM, RF, DT, and XGBOOST deep learning identifiers, resulting in 97.1% accuracy on the Malimg dataset which transcends GIST and HOG models on both accuracy and precision. In [12] VGG-16 based architecture is used called M-CNN which outperforms the baseline GIST+SVM model and achieved a 98.52% accuracy[12] on the well-known Malimg dataset.

In [19] MSIC framework is used and the spectrogram images are fed through three CNN classifiers with 1 layer, 2 layers, and 3 layers respectively. In [19], MSIC's F-measure for the CNN with the most layers was 91.6% with an accuracy of 92.8% contrasted to 90.6% and 92.3% for general grayscale solution [19]. MSIC also proved to be faster than the grayscale solution as it scored 0.5% more on F-measure and accuracy in detecting malware from malignant files. A hybrid module is created in [28] based on pre-trained architectures containing ResNet-50 and AlexNet which gains an accuracy of 96.5% on the Malevis dataset and does even better on accuracy in the Maling dataset, reaching a high of 98.52%. For larger unseen malware samples a Frequency Domain-Based malware detection [35] is shown to be very effective with an accuracy of 96% on MaleX 1 dataset, giving it an edge over deep learning techniques such as ResNet. In [13], the novel model uses a "3C2D" network architecture with single-channel "bigram-dct" as inputs. As for in [17], the malware detection system referred to as DMDL uses a CNN architecture which shows astounding effectiveness in categorizing malware over other baseline models, and according to [17], they reached 99.17% accuracy on Microsoft whereas on Malimg it was 98.52%. Another image-based machine learning approach addressed as K-NN in [25] successfully reached 97.9% accuracy on Malimg, 94.41% on Malheur, and 95.63% on Microsoft BIG2015 dataset putting itself appreciably ahead of other classifiers of similar techniques such as NB, SVM, CNN. Authors of [14] focused on transfer learning by mainly considering the datasets Malimg and Malacia. Initially, K-NN outperforms DL by achieving 99.6% accuracy in binary and also successfully classifying multi-class problems with an accuracy of 99.4%. However, the zero-day simulation brought the most significant results. Here, the image-based DL model successfully identified 79% of the malware samples with a low false positive rate of 1%. The authors in [20] merged the global and local information, thus the malware can be classified using RF, KNN, and SVM and gains 95% system accuracy for kaggle dataset. Also, in [21] the presented novel machine learning solution claimed to achieve the following accuracy for Malimg, Ember, and BIG2015 malware datasets: 0.998, 0.911, and 0.997. These results tower over other malware classifiers such as autoencoder with Softmax, autoencoder with SVM, and PCA with Softmax. The authors of [21] were able to achieve this height through the use of ResNet-50 architecture including a dense CNN for classifying images. The authors in [36] developed a modified DNN model with deep denoising Autoencoder elements for feature compression. The test results show a fair amount of potential in [36], with 96% classification accuracy gained by means of MLP (as a subnet of complete DNN).

As stated in [18] the authors introduced ScaleMalNet, a highly scalable malware detection framework. This framework uses deep learning to analyze the malware that has been collected from hosts used by end-users in two steps. The tests were performed on Malimg datasets and promised an accuracy of 96%.

In [29] the authors use a CNN model called DeepVisDroid and compare its results with two other models, named ResNet and Inception V3. Here, the DeepVisDroid model outperformed the previously mentioned classical CNN models by resulting in a high 98.96% accuracy. In [27], a comparative malware classification is done between the common grayscale solution (GDMC) and deep learning based on markov images (MDMC). Data from Microsoft and Dreblin datasets is fed through a CNN classifier where it is proved that MDMC has better performance. It gave an average accuracy rate of a staggering 99.264% and 97.364% on the two datasets.

Instead of typical image classification methods, [13] used a malware classification algorithm called MCSC that extracts opcode sequences and encodes them while maintaining the malware's defining features. The final results were in favor of MCSC as accuracy rose as high as 99.260% while maintaining an average of 98.862%.

For evaluation, the authors of [22] prioritized the macro-averaged F1-score as accuracy can be a deceptive evaluation metric in large class disparity datasets. In [15], the authors used a hierarchical CNN (HCNN) and compared its results with two categories of approaches— hex-based and assembly-based. In [15], the HCNN performed better than the first approach by a small but noticeable margin, specifically, it reached a 0.9913 accuracy and 0.9830 F1 score. However, for the second approach, the results were unclear about a better performer due to the variations in it. The authors of [23] are one of the first ones to use intermediate fusion as a means to combine features from numerous modes and categories of data. It does this by implementing a multimodal deep learning method that considers both the binary part and the code of the malicious software that is in the assembly language. Finally, with an 0.9924 accuracy and a 0.9872 F1 score, this method showed potential by experimentally performing better than existing deep learning approaches. One of the most effective deep learning frameworks ever created is a multimodal hybrid system [22] with a staggering 0.9951 macro F1-score. The authors in [22] call it HYDRA which consists of API-based, mnemonics-based, byte-based, and feature fusion and classification components.

The paper [39] presents a self-supervised deep neural model built on the Vision Transformer design, referred to as SHERLOCK. This achieves a 97% accuracy rate in detecting binary malware, surpassing current methods in multi-class classification. In the research, they made use of the MalNet dataset, the largest publicly available collection of cybersecurity images, comprising 1.2 million images. This dataset includes diverse labels for each image, spanning two primary categories (malware or benign), 47 categories for classifying malware types, and 696 categories for identifying malware families. The system interprets malware images as sequences of patches, leveraging transformer encoders for efficient processing. SHERLOCK outperforms traditional supervised methods by generating training samples for malware synthesis.

Upon conducting an extensive review of existing literature pertaining to binary image malware datasets, we have made the decision to focus on developing a lightweight Convolutional Neural Network (CNN) model for our study. Moreover, our investigation revealed a scarcity of research utilizing transformers on datasets of this nature. While there exist a few instances where Vision Transformer (ViT) has been applied to malware datasets, the majority of research on CCT (Compact Convolutional Transformer) has primarily focused on applications pertaining to crops and lung illness. Thus, the decision was made to execute the malware dataset on various transformers. In addition, we made the decision to include the Involution and EANet models in our research, as we discovered a lack of existing literature on the application of these models specifically for binary image malware datasets. In conclusion, our research intends to employ LIME as an explainable artificial intelligence technique to present our findings. It is worth noting that our investigation did not yield an adequate amount of existing literature on this particular topic as well. The papers that we did discover in close proximity to our chosen topic were LIME projects that mostly revolved around Android malware datasets, which exhibit subtle variations from the datasets we intend to analyze.

Chapter 4

Research Objectives

We tend to classify various malware types including Backdoor, Worm, Trojan as well as Trojan-Downloader and Rogue using traditional deep learning models. These deep learning models will be able to classify malware using various datasets which will help us to differentiate between these deep learning techniques and learn further about malware classification. Additionally, we want to develop a CNN-based Deep Learning Model. Our proposed CNN model will work on traditional datasets, namely Malimg, Malex, Malicia, Malevis, and Microsoft BIG 2015 in order to classify malware.

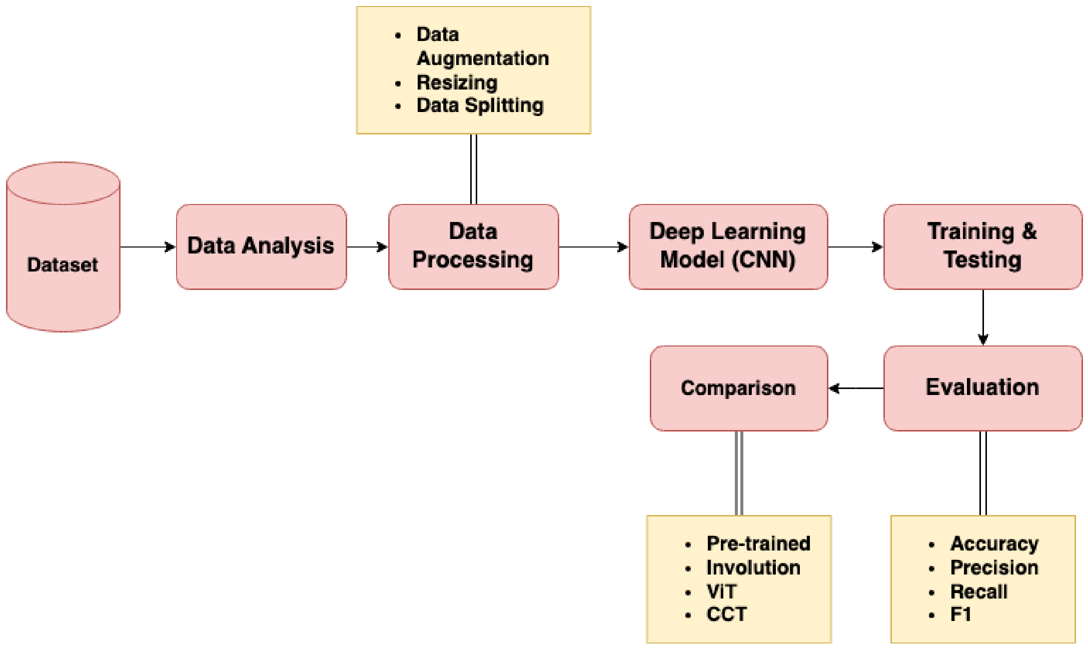


Figure 4.1: Proposed Workflow

After completing our research:
1. By the end, we will know exactly how Deep Learning methods can be used to categorize different types of malware.
2. Compare the deep learning models and determine which is the best for malware classification.
3. Develop a novel architecture that will classify malware.
4. Evaluate our suggested model and produce a comparative analysis with existing pre-trained models.

5. Evaluate our suggested model and produce a comparative analysis with existing modern cutting-edge models.
6. Work on explainable AI in order to comprehensively showcase how our model designated each malware sample to a specific class.

Chapter 5

Description of the Dataset

5.1 Data Analysis

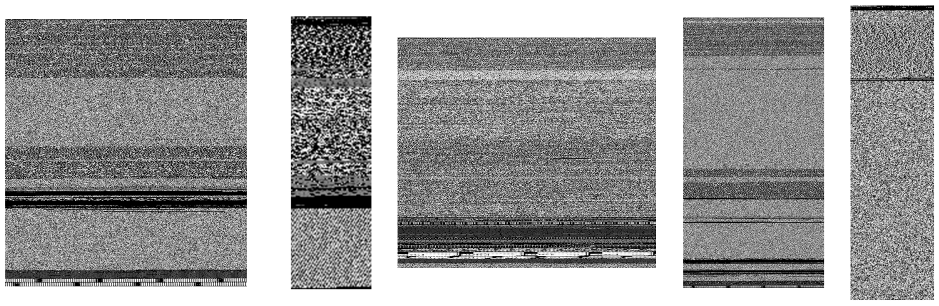


Figure 5.1: 5 random images from training dataset

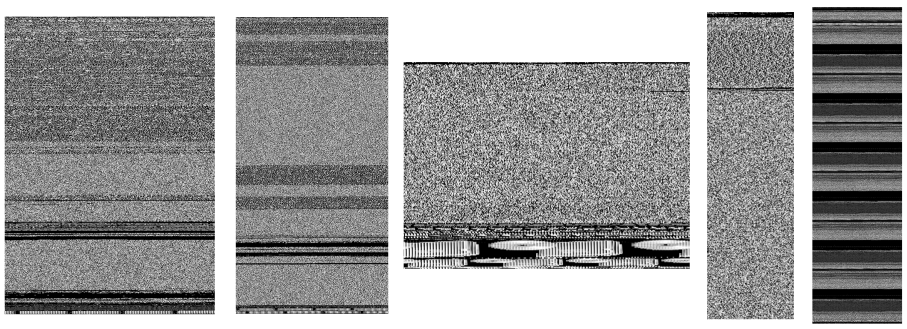


Figure 5.2: 5 random images from validation dataset

Our research will primarily make use of data from the Malimg Dataset. There are 25 distinct malware family image categories in this collection. The dataset that has been put together has a total of 9,339 images. The dataset is then divided into three more categories: testing, training, and validation, each of which contains 934, 7473, and 932 photos, respectively. The distributions depicted in Figure 5.5 pertain to the validation set, whereas those shown in Figure 5.6 correspond to the training set. Additionally, Figure 5.7 illustrates the distribution of the testing set. All the photos are grayscale representations of malware binaries [5], with the remark that for many malware classes, the layout and texture of the pictures within the same family are

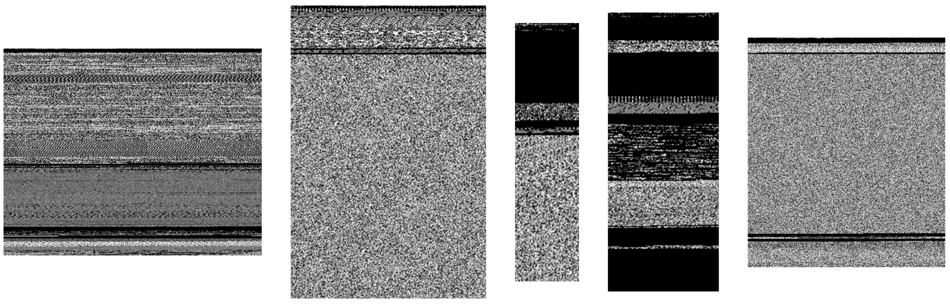


Figure 5.3: 5 random images from testing dataset

highly similar. These images have been arranged into 25 different groups as a result of this commonality in vision. According to [16], These categories contain examples of malicious software from families including Rbot!gen, Malex.gen!J, Yuner. A, VB.AT, and Autorun.K that are known to be UPX-packed. Pictures of other family variations including the C2Lop.gen!g and C2Lop.p as well as the Swizzor.gen!E and Swizzor.gen!Iare also shown [16].

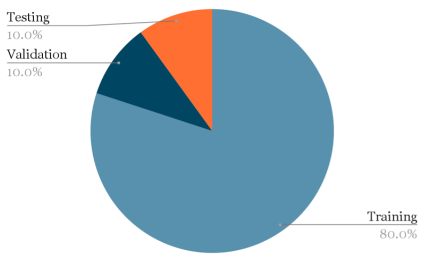


Figure 5.4: Distribution between testing, validation and training

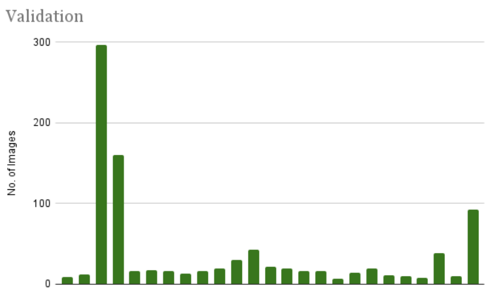


Figure 5.5: Distribution of Validation Images per class

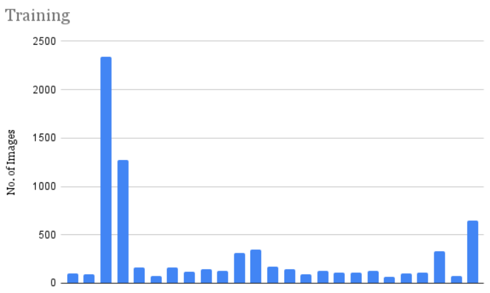


Figure 5.6: Distribution of Training Images per class

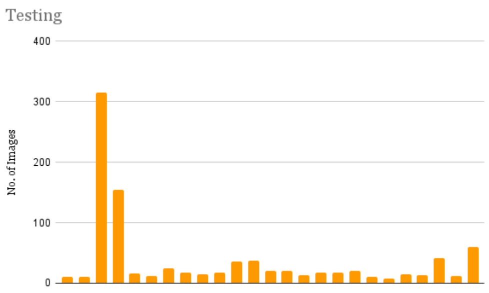


Figure 5.7: Distribution of Testing Images per class

5.2 Data Preprocessing

Data pre-processing is a method used to get rid of superfluous variables that don't help the accuracy of the CNN model. This is achieved by utilizing the "de-noising" approach. [24] The initially sourced data is altered to raise the bar on the performance of the CNN-based models, which yields better accuracy and outcomes. ”Image Resizing” and ”Data Augmentation” are the first and second steps in the processing of our data, respectively. The images in the dataset we utilize range in size; varying examples include 64 by 216 pixels, 512 by 410 pixels, etc. In accordance with the first step of our data processing, we initially resize all these photos to 100 by 100 pixels. Then, to perform “Data Augmentation”, horizontal flips are carried out to add to and enhance the data. Finally, to conserve computer resources, the photographs are converted into a matrix and normalized by dividing with 255.

5.3 Detailed Distribution of Dataset

Here, table 5.1 shows the number of samples for each class in the dataset and their specific distributions into training, testing, and validation sets.

Name	**Training**	**Testing**	**Validation**
Adiler.C	102	11	9
Agent.FYI	93	11	12
Allaple.A	2337	315	297
Allaple.L	1276	155	160
Alueron.gen!J	166	16	16
Autorun.K	77	12	17
C2LOP.gen!g	159	25	16
C2LOP.P	116	17	13
Dialplatform.B	146	15	16
Dontovo.A	126	17	19
Fakerean	315	36	30
Instantaccess	351	37	43
Lolyda.AA1	171	20	22
Lolyda.AA2	144	20	20
Lolyda.AA3	94	13	16
Lolyda.AT	125	18	16
Malex.gen!J	112	17	7
Obfuscator.AD	108	20	14
Rbot!gen	129	10	19
Skintrim.N	62	7	11
Swizzor.gen!E	103	15	10
Swizzor.gen!I	110	14	8
VB.AT	328	41	39
Wintrim.BX	75	12	10
Yuner.A	648	60	92
Total	7473	934	932

Table 5.1: Training, Testing and Validation Distribution per Class

5.4 Dataset Classes

5.4.1 Adiler.C

Adiler.C [42] is addressed as the first class of malware in the Malimg dataset which consists of 122 dialer-type malware. It's a trojan dialer that targets PCs with modems connected to phone lines. It secretly dials premium-rate telephone numbers, leading to unexpectedly high bills for the user. This particular malware was detected and addressed by Microsoft Defender Antivirus.

5.4.2 Agent.FYI

Agent.FYI [43] is the 2nd class of malware in Malimg dataset consisting of 116 backdoor-type malware. The primary intent of most Agent variants is to download and install adware or other harmful software onto the victim's computer. Furthermore, these Trojans may alter configuration settings for Windows Explorer and the Windows interface, potentially leading to additional harm and compromising the system's security.

5.4.3 Allaple.A

Allaple.A [44] is a malware class of Malimg dataset consisting of the highest number of samples of worm-type malware. The sample consists of 2949 malware. Allaple.A is a highly dangerous network worm with multi-threaded and polymorphic capabilities, posing a significant threat to computer systems. It has the ability to propagate across local area networks (LANs) and launch denial-of-service attacks against remote websites. The worm spreads through exploiting vulnerabilities in unpatched systems or weak passwords through a dictionary attack. Once infiltrated, it duplicates itself in various locations and modifies the system's registry to ensure execution upon startup. To make detection more difficult, it employs a polymorphic engine that encrypts its body uniquely for each infection.

5.4.4 Allaple.L

Similar to Allaple.A, Allaple.L [45] is also a class consisting of worm-type samples. The class has a total of 1591 samples which is the 2nd highest among all the 25 classes. Worms like Allaple.L have the ability to automatically propagate to other computers through various means, such as duplicating their own data onto portable storage devices and shared directories, or via electronic mail transmission.

5.4.5 Alueron.gen!J

Alueron.gen!J [46] is a class of malware of 198 samples that also have worm-type malware. It is a harmful trojan known for attempting to modify DNS settings on network routers. The trojan presents a significant danger as it grants attackers the potential to send malicious data to the infected computer.

5.4.6 Autorun.K

Now, Autorun.K [47] is a Worm:AutoIT type malware class consisting of a total of 106 malware. Unlike viruses, these worms possess the capacity for self-replication and spread to other computers without any user intervention. This particular worm employs the Windows Autorun feature to facilitate its propagation through removable drives, such as USB flash drives. When an infected drive is connected to another computer, the worm tries to execute itself automatically, leading to its dissemination to other PCs.

5.4.7 C2LOP.P

Another Trojan malware type class in malimg dataset is C2LOP.P [48] which consists of 146 samples. It exhibits malicious behavior by altering web browser settings, adding bookmarks, and displaying pop-up adverts on the affected system. This trojan is known to arrive on a computer bundled with other software and inject its harmful code into the Internet Explorer process. Additionally, it may make changes to the system registry by adding specific values and data. Once active, the trojan connects to remote websites to download and execute arbitrary files, often belonging to the TrojanDownloader:Win32/Swizzor malware group or other variants of the Trojan:Win32/C2Lop malware group. After accomplishing the installation of these files, the trojan generates undesired pop-up ads and adverts on the compromised machine.

5.4.8 C2LOP.gen!g

Following we have C2LOP.gen!g [49] with the same malware type of trojan and a sample set of 200. C2Lop.gen!G is a generic trojan belonging to the Trojan:Win32/C2Lop malware group. The C2Lop trojan family is notorious for its actions, which involve modifying web application browser settings, adding bookmarks, and delivering undesired pop-up adverts on infected systems. It can infiltrate a computer as a bundled file with other software and inject its malevolent code into the Internet Explorer process upon execution. One of its hostile behaviors includes connecting to remote websites to download and execute arbitrary files, typically members of the TrojanDownloader:Win32/Swizzor malware group or other variants of the Trojan:Win32/C2Lop malware group. After accomplishing the installation of these files, the trojan inundated the compromised system with intrusive pop-up ads and advertisements.

5.4.9 DialPlatform.B

Class DialPlatform.B [50] has the dialer type malware with 177 samples. This malicious software takes advantage of PCs equipped with a modem connected to a phone line, using them to dial premium-rate phone numbers while the user is unaware of it. The consequence of this unauthorized activity is unexpected and substantial telephone bills for the victims. It exclusively targets systems with modems, and once infiltrated, it manipulates the dial-up connection to initiate costly premium-rate calls, potentially leading to significant financial losses.

5.4.10 Dontovo.A

Dontovo.A [51] contains 162 sample counts of Trojan downloader-type malware. This malicious trojan is programmed to download and run files of any kind on the compromised system. Upon activation, Dontovo.A creates a duplicate of %Windows%\svchost.exe and injects its code into it, subsequently removing its original executable. It then establishes contact with domains like "iframr.com" to obtain configuration data, which may include additional locations for downloads. Notably, in real-world instances, it has been observed connecting to "videofx4you1.com" for this purpose. The downloaded files are stored in the %temp% directory and executed accordingly. At the time of detection, Dontovo.A was found to download malware identified as Worm:Win32/Koobface.gen!D.

5.4.11 Fakerean

Under the Fakerean class [52], there are 381 samples of rogue-type malware. It is a group of rogue security programs that employ deceptive tactics to convince users that their PCs are infected with malware. These fraudulent programs mimic antivirus or antimalware scanners, conducting fake scans that generate false reports of multiple infections, prompting users to pay for the software to purportedly clean their systems. However, in reality, Fakerean does not detect any genuine malware, and it is not a legitimate security scanner; its primary objective is to trick users into sending money to the developers behind the program. Some versions of Fakerean unlawfully impersonate Microsoft products by using counterfeit product names or logos. Even if users fall for the ploy and pay to "unlock" the application, it remains ineffective since there is no actual malware present. Various iterations of Fakerean may also manipulate computer settings, terminate processes or system services, and restrict access to specific websites.

5.4.12 Instantaccess

Instantaccess [53] is a Dialer-type malware class with 431 samples. It is a deceptive dialer program that tempts users with the offer of premium services from a website but, instead, connects to high-cost numbers, leading to unexpected and costly phone charges. Furthermore, this program drops a trojan into the system, posing additional security risks. When activated, InstantAccess creates duplicates of itself and alters the system registry to ensure automatic implementation during Windows startup. It also introduces new files, folders, and shortcuts, including "Superbabes," attempting to lure users into activating them. If the "Superbabes" icon is clicked, a dialog window may appear. To maintain its persistence, the dialer may make further modifications to the system registry.

5.4.13 Lolyda.AA1, Lolyda.AA2, Lolyda.AA3

The 13, 14, and 15th classes are Lolyda.AA1, Lolyda.AA2, Lolyda.AA3 [54] all of them have the same password stealer type malware with slight variations. That's why Microsoft addressed them together under their Lolyda.AA threat. The ist, 2nd, and 3rd type has 213,184 and 123 samples respectively. PWS:Win32/Lolyda.AA variations are trojans that are designed to illicitly gather sensitive information associated with well-known online games and transmit it to a remote attacker. Additionally, this trojan has the capability to download and run files of any kind, potentially further compromising the infected system.

5.4.14 Lolyda.AT

Lolyda.AT [55] is a member of a family of trojans that transfers account information from well-known online games to a remote site for theft. It can also kill processes, collect screenshots, and hook certain APIs.

5.4.15 Malex.gen!J

Malex.gen!J [56] is a malicious trojan that is able to access affected PCs without a user's permission and knowledge. Once installed on a compromised machine, this trojan can execute numerous malicious activities that can harm your operating system and its core functionalities. It is able to open a backdoor to allow remote attackers to gain access to the corrupted machine, which can cause identity theft. Furthermore, it can change a PC's system settings, block anti-virus software, and show annoying popup advertisements in order to prevent the user from identifying the actual problem underneath.

5.4.16 Obfuscator.AD

The trojan is known as Obfuscator.AD [57]. It operates covertly by retrieving malevolent files from a distant server and thereafter initiating their installation and execution.

5.4.17 Rbot!gen

Rbot!gen [58] [59] is a type of backdoor malware - a remote administration utility program that, upon installation on a computer, enables a user to have access to and exercise control over a network or the Internet. Once a computer becomes infected, the trojan establishes a connection with a designated Internet Relay Chat (IRC) server and proceeds to join a certain channel in order to receive instructions from malicious actors. Furthermore, the trojan has the capability to disseminate to additional computer systems through several means, such as conducting scans for network shares that possess vulnerable passwords, exploiting vulnerabilities within the Windows operating system, and propagating via backdoor ports that have been opened by other types of malicious software families. The presence of a backdoor enables a remote assailant to engage in a range of activities, including data theft, execution of Denial-of-Service attack orders on the compromised system, or unauthorized access to other devices within a local network.

5.4.18 Skintrim.N

Skintrim.N falls under the category of Trojan that downloads other risks onto the victim's computer.

5.4.19 Swizzor.gen!E, Swizzor.gen!I

Swizzor.gen!E and Swizzor.gen!I are both Trojan Downloaders. Trojan downloaders install themselves and wait for an Internet connection to establish a connection with a remote server or website in order to initiate the downloading of malicious software onto the infected computer. .

5.4.20 VB.AT

VB.AT is a worm that propagates through peer-to-peer (P2P) networks. Additionally, it attempts to deactivate many programs on compromised PCs.

5.4.21 Wintrim.BX

Wintrim [60] is a group of trojans that exhibit pop-up adverts based on the user's keywords and surfing history. In addition, the various iterations of this software possess the capability to watch the user's actions, acquire apps, and transmit system data to a distant server.

5.4.22 Yuner.A

Yuner.A is a worm-type malware. Worms spread automatically across PCs. They can duplicate themselves on portable discs, network files, or over email.

Chapter 6

Pre-trained Models

6.1 ResNet-50

The ResNet 50 model (Residual Networks) can be implemented in a wide variety of ways. The '50' in its name indicates that it can function with as many as 50 layers of neural networks. According to [6], the majority of the uses for this ResNet 50 model are in the area of computer vision, and it can offer a solution to the vanishing gradient problem. We use 64 distinct kernels in the first convolutional layer of our model, each of which is 7 × 7 in size and has a stride of 2. Then, with a stride value of 2, a MaxPooling layer with dimensions of 3 by 3 is shown. Then, we have three 3 x 3 instances of the 64 kernels and three 1 x 1 instances of both 64 kernels and 256 kernels. At this point, we are bringing the total to nine levels. Then we will be allowed to observe 1 x 1 with 512 kernels, 3 by 3 with 128 kernels, and 1 by 1 with 128 kernels, bringing the total number of layers to 12. The next set of operations includes 1 x 1 and 3 x 3 with around 256 kernels, as well as 1 x 1 with 1024 kernels, producing 18 layers. Finally, we see 1 x 1 and 3 x 3 with a big number of 512 kernels, as well as 1 x 1 with an even bigger number of 2048 kernels. The last convolutional layer brings the total amount of layers to 9. This network was given the name ResNet 50 since we can observe a total of 50 layers in the final output. The bottom layer is an average pool, while the next layer is a fully-connected one. Then there is a softmax function that is used to set the total amount of neurons in this layer based on instances of unique classes in the Malimg Dataset.

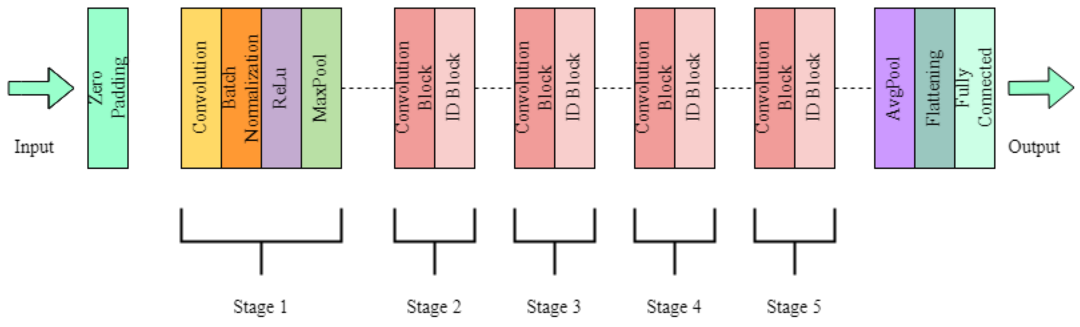


Figure 6.1: Architecture of ResNet-50

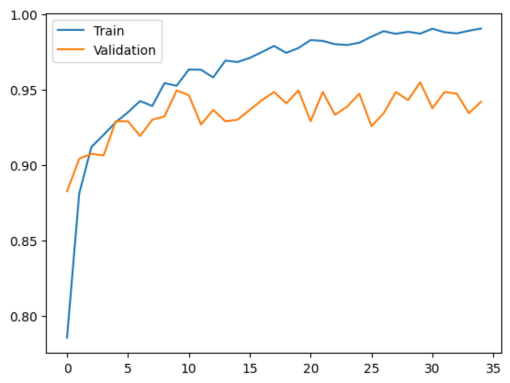


Figure 6.2: ResNet-50 Accuracy

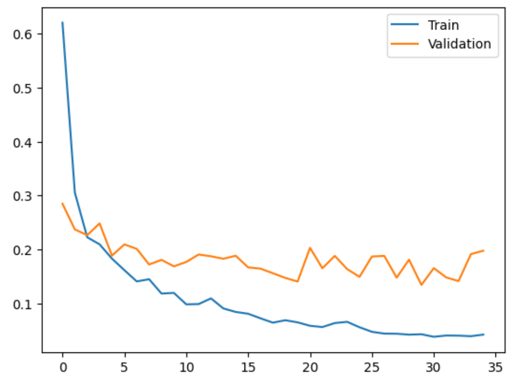


Figure 6.3: ResNet-50 Loss

Layers	ResNet 50	Number of Layers
2D Convolutional Layer	7 x 7, 64, stride 2	1
2D Convolutional Layer	3 x 3 max_pool, stride 2 [1 x 1, 64] x 3 [3 x 3, 64] x 3 [1 x 1, 256] x 3	9
2D Convolutional Layer	[1 x 1, 128] x 4 [3 x 3, 128] x 4 [1 x 1, 512] x 4	12
2D Convolutional Layer	[1 x 1, 256] x 6 [3 x 3, 256] x 6 [1 x 1, 1024] x 6	18
2D Convolutional Layer	[1 x 1, 512] x 3 [3 x 3, 512] x 3 [1 x1, 2048] x 3	9
	Average Pool, 25	1

Table 6.1: Layers of ResNet-50

6.2 Inception V3

According to [9], Reducing the amount of processing resources required to run the software by incorporating numerous changes to the preceding Inception architectures is a key focus of Inception version 3. We didn't make many changes to the layers during this stage of the process, but we did create an output layer with 25 nodes to account for the number of malware categories in the Malimg dataset. Because factorized convolution reduces the overall number of parameters used in a network, it is valuable for monitoring network efficiency and lowering the computer efficiency necessary to do so. Convolutions that are larger are gradually being replaced by convolutions that are smaller since they will speed up training. The amount of processing power required is decreased if a completely linked layer is present before a 3 x 3 convolution layer since the weights of the 3 x 3 layer can then be shared among themselves. This further reduces the overall amount of parameters needed for the method. The previous asymmetric convolutions approach uses a 1 x 3 convolutional layer, and then again a 3 x 1 convolutional layer, as opposed to a single 3 x 3 layer. Minor CNN layers are also produced between the layers during the training process,

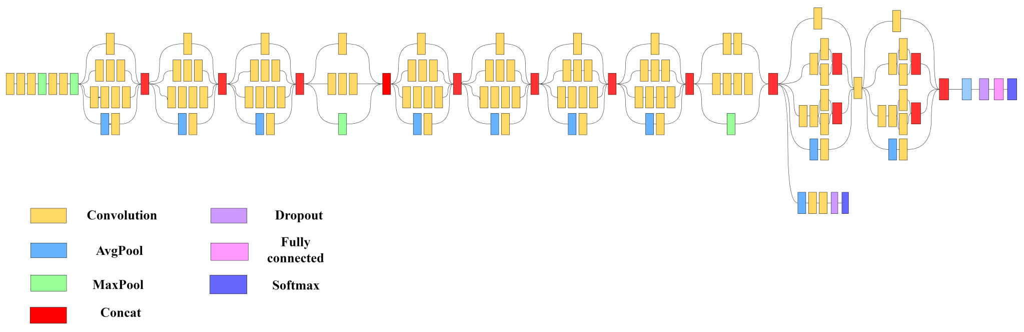


Figure 6.4: Architecture of Inception V3

and the loss from these layers is contributed to the loss from the primary network. In order to reduce the size of the grid, pooling layers are used as a solution.

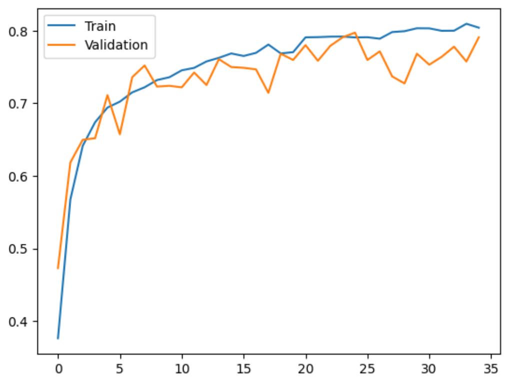


Figure 6.5: Inception V3 Accuracy

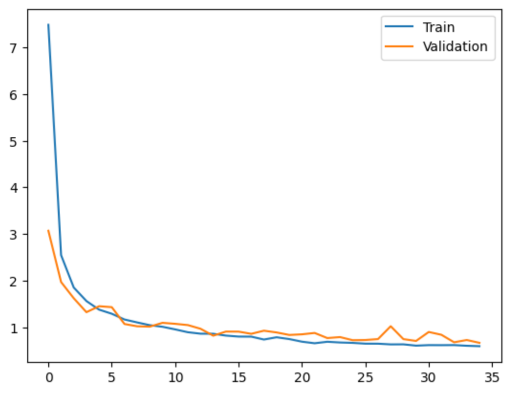


Figure 6.6: Inception V3 Loss

6.3 VGG-16

VGG16 is one of the most well-known CNN models for object detection and classification. This model was suggested by Simonyan and Zisserman in [8], they analyzed the networks and enhanced the depth using an architecture with incredibly small (3 by 3) convolution filters, which at that time demonstrated a notable advancement over many previous state-of-the-art setups. The model that they developed (which was later named VGG 16) attained an accuracy of 0.927 on the sourced collection of pictures called ImageNet which contains almost 14 million images. There are sixteen layers in this model with adjustable parameters as well as a few 2D max-pooling layers. Thirteen of these are convolutional layers and the rest three are completely linked layers due to the usage of the AlexNet ReLU. The initial 2 layers have 64 channels of a 3 by 3 filter size with similar padding. This value of 64 is multiplied by 2 in every block that follows after it until a final number of 512 is reached. For our model, we do not have any hidden layers but one output layer of 25 nodes that helps categorize the malware images.

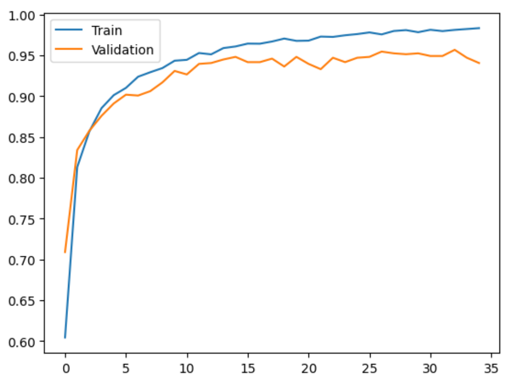


Figure 6.7: VGG-16 Accuracy

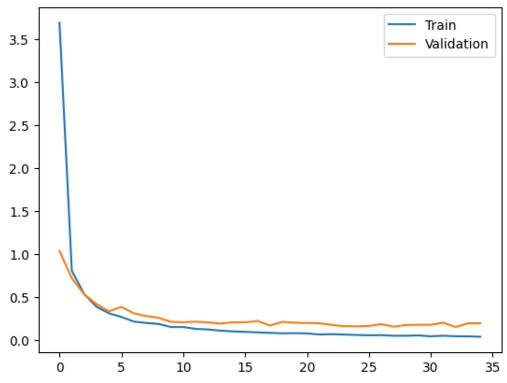


Figure 6.8: VGG-16 Loss

6.4 DenseNet201

DenseNet-201 translates to Dense Convolutional Network which is a pre-trained CNN model that is a certain layer deep. According to [11], the general DenseNet has $\frac{[L(L+1)]}{2}$ direct connections, when compared to regular L-layer convolutional networks, where just L connections are present (one between each layer and the next). So, what happens instead is that the previous layers' feature maps are used as inputs for every layer, and each layer's self-feature maps (that belong to themselves) are used as inputs for all upcoming layers. DenseNets offer a lot of advantages including a way to tackle the vanishing-gradient issue, having significantly lower parameters, improving feature propagation, and promoting feature reuse. The DenseNet-201 we use is one of the variations of this model except it has 201 layers. As we can see in [26], DenseNet-201 is one of the better variations to use because of its promising results.

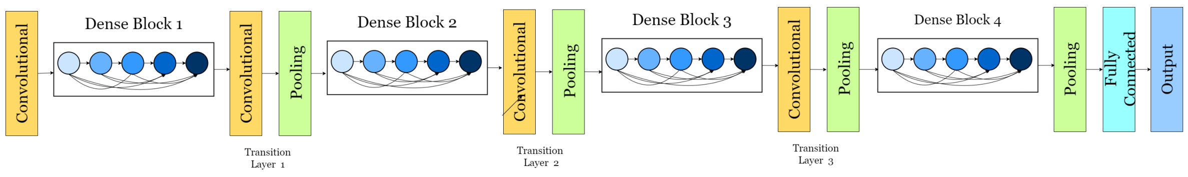


Figure 6.9: Architecture of DenseNet201

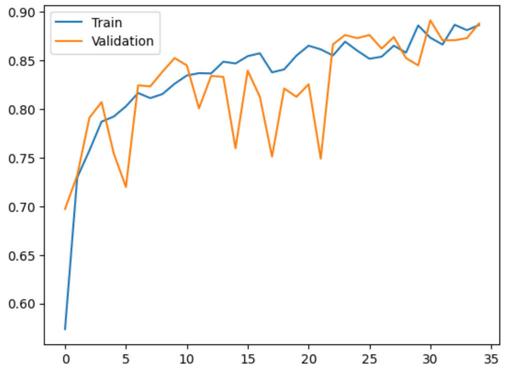


Figure 6.10: DenseNet201 Accuracy

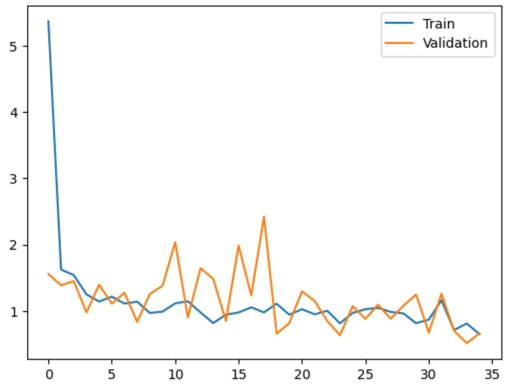


Figure 6.11: DenseNet201 Loss

Layer	**Output Shape**	**Parameters**
(InputLayer)	[(None, 100, 100, 3)]	0
(Conv2D)	(None, 100, 100, 64)	36928
(MaxPooling2D)	(None, 50, 50, 64)	0
(Conv2D)	(None, 50, 50, 128)	73856
(Conv2D)	(None, 50, 50, 128)	147584
(MaxPooling2D)	(None, 25, 25, 128)	0
(Conv2D)	(None, 25, 25, 256)	295168
(Conv2D)	(None, 25, 25, 256)	590080
(Conv2D)	(None, 25, 25, 256)	590080
(MaxPooling2D)	(None, 12, 12, 256)	0
(Conv2D)	(None, 12, 12, 512)	1180160
(Conv2D)	(None, 12, 12, 512)	2359808
(Conv2D)	(None, 12, 12, 512)	2359808
(MaxPooling2D)	(None, 6, 6, 512)	0
(Conv2D)	(None, 6, 6, 512)	2359808
(Conv2D)	(None, 6, 6, 512)	2359808
(Conv2D)	(None, 6, 6, 512)	2359808
(MaxPooling2D)	(None, 3, 3, 512)	0
(CustomFlatten)	(None, 4608)	0
(Dense)	(None, 25)	115225
Total Params: 14,829,913		
Trainable params: 115,225		
Non-trainable params: 14,714,688		

Table 6.2: Layers with output shape and parameters of VGG-16

Chapter 7

Methodology

7.1 Convolutional Neural Network

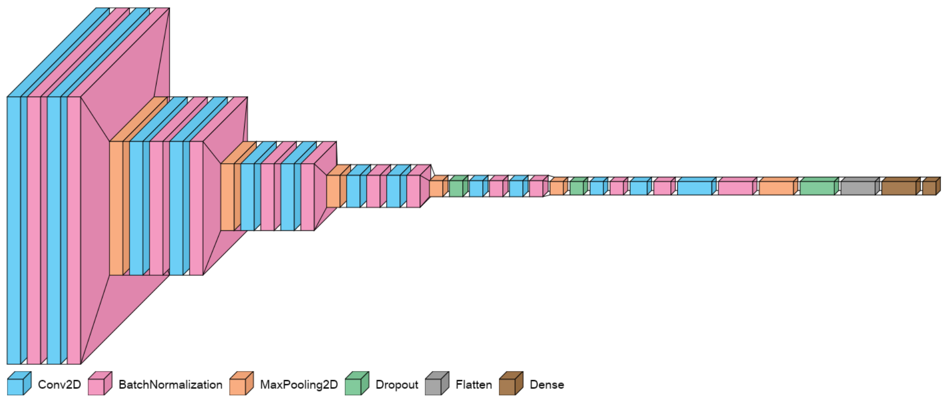


Figure 7.1: Illustration of our proposed Architecture

Convolutional Neural Networks are currently very well-known and widely used deep learning networks in machine vision implementations. Following the enrichment of large image-based data sets, CNNs were finally able to become particularly significant with the introduction of Alexnet in 2012. CNNs are specifically designed to emulate the visual processing capabilities of humans by transforming images into a more compact representation while preserving essential features.

The system employs a process known as convolution, where a filter or kernel is applied to the input image to generate convolved features. This operation is repeated across multiple layers of artificial neurons, allowing for the extraction of increasingly complex features. Ultimately, the final layer produces confidence scores that indicate the probability of the input image belonging to specific classes.

In order to address computational complexity, pooling layers are utilized. Convolved features' spatial dimensions can be shrunk with the use of layers like average and max pooling. Max pooling, as the name suggests, selects the biggest value within a kernel, and average pooling reveals the average. Max pooling is still the favored one due to its efficacy in suppressing noise.

While CNNs do have their limitations and face challenges in comprehending context, they have brought about a paradigm shift in artificial intelligence. They are extensively utilized in various domains, including facial recognition, image search, augmented reality, and other applications. However, it is also to be noted that despite these great advancements, achieving human-like intelligence still remains a formidable task.

7.1.1 Proposed Model

A model's computation speed rises with fewer parameters so our main goal is to implement a minimum number of parameterized variables while maintaining decent results. To begin with, a 3-channel image with a resolution of 100 by 100 pixels from the dataset's preprocessing step serves as the basis for our CNN model. Thirteen Conv2D layers with a kernel size of 3 x 3 are used. In addition, we apply a 15% dropout before the ninth and tenth Conv2D layers, a second 15% dropout after these two mentioned Conv2D layers, and then we add a third dropout of the same value to end the model in our experiment. We do this to prevent the model from perfectly matching its training data called overfitting. Additionally, to lessen the burden on the computer system, we use six MaxPooling layers that have a pool size of 2,2.

ReLU is the activation function that will be utilized for our model because it gives deep learning models the ability to be non-linear and addresses the problem of vanishing gradients. When compared to other functions of this sort, for example, the Sigmoid or Tanh, ReLU's gradient is more unsaturated, which drastically quickens the development of stochastic gradient descent. After converting the values into a 1D array, we start by adding fully connected layers to the CNN, beginning with 512 nodes. Moving on, the Softmax activation function is implemented as a network classifier in the final output layer. This classifier is a generalized binary variation of logistic regression. Since all nodes can be categorized using the softmax function, it is provided near the end of the result.

Furthermore, the optimizer is Adam and it helps to contribute to maximizing production efficiency with a learning rate of 0.0001. We also set the model's maximum runtime at 35 epochs with 32 batch sizes. After several model modifications, we develop our Deep CNN architecture. We created the model by making use of enhanced hyperparameters. This architecture uses a lesser amount of processing resources and offers a very promising performance while reducing the number of parameters.

7.1.2 Explainable AI

The Explainable AI model employed in our study is LIME. Once the prediction generated by our Convolutional Neural Network (CNN) model concludes, the image will be then passed to the Local Interpretable Model-agnostic Explanations technique, referred to as LIME. This technique will then provide an explanation for the assignment of the image to a certain class as determined by our suggested model.

LIME's model-independent nature enables its compatibility with a wide range of

Layer	**Output Shape**	**Parameters**
(Conv2D)	(None, 100, 100, 8)	224
(BatchNormalization)	(None, 100, 100, 8)	32
(Conv2D)	(None, 100, 100, 8)	584
(BatchNormalization)	(None, 100, 100, 8)	32
(MaxPooling2D)	(None, 50, 50, 8)	0
(Conv2D)	(None, 50, 50, 16)	1168
(BatchNormalization)	(None, 50, 50, 16)	64
(Conv2D)	(None, 50, 50, 16)	2320
(BatchNormalization)	(None, 50, 50, 16)	64
(MaxPooling2D)	(None, 25, 25, 16)	0
(Conv2D)	(None, 25, 25, 32)	4640
(BatchNormalization)	(None, 25, 25, 32)	128
(Conv2D)	(None, 25, 25, 32)	9248
(BatchNormalization)	(None, 25, 25, 32)	128
(MaxPooling2D)	(None, 12, 12, 32)	0
(Conv2D)	(None, 12, 12, 64)	18496
(BatchNormalization)	(None, 12, 12, 64)	256
(Conv2D)	(None, 12, 12, 64)	36928
(BatchNormalization)	(None, 12, 12, 64)	256
(MaxPooling2D)	(None, 6, 6, 64)	0
(Dropout)	(None, 6, 6, 64)	0
(Conv2D)	(None, 6, 6, 96)	55392
(BatchNormalization)	(None, 6, 6, 96)	384
(Conv2D)	(None, 6, 6, 128)	110720
(BatchNormalization)	(None, 6, 6, 128)	512
(MaxPooling2D)	(None, 3, 3, 128)	0
(Dropout)	(None, 3, 3, 128)	0
(Conv2D)	(None, 3, 3, 128)	147584
(BatchNormalization)	(None, 3, 3, 128)	512
(Conv2D)	(None, 3, 3, 256)	295168
(BatchNormalization)	(None, 3, 3, 256)	1024
(Conv2D)	(None, 3, 3, 512)	1180160
(BatchNormalization)	(None, 3, 3, 512)	2048
(MaxPooling2D)	(None, 1, 1, 512)	0
(Dropout)	(None, 1, 1, 512)	0
(Flatten)	(None, 512)	0
(Dense)	(None, 512)	262656
(Dense)	(None, 25)	12825
Total params: 2,143,553		
Trainable params: 2,140,833		
Non-trainable params: 2,720		

Table 7.1: Layers with output shape and parameters of proposed model

machine learning or deep learning models. LIME facilitates comprehension of the inner workings of a model and makes predictions by focusing on what features have been extracted from the image. The input image undergoes a slight alteration, and subsequently, the impact of this modification on the previous prediction is examined. When an interpretable model is used to localize it, there is a potential for faithfully explaining the predictions made by any classifier or regressor. LIME, in its fundamental essence, examines the relationship between input and output within a model by hypothesizing the presence of a black box machine learning model connecting the two.

In reference to the figures provided below, the malware class under analysis is the Dontovo.A. In order to comprehensively analyze and clarify the discernible patterns observed within a selected class, three different samples from Dontovo.A are chosen and examined. It is important to acknowledge that the aforementioned patterns have been generated through the use of the explainable AI technique called LIME. This technique serves the purpose of aiding our comprehension of how our suggested model assigns samples to their various classes.

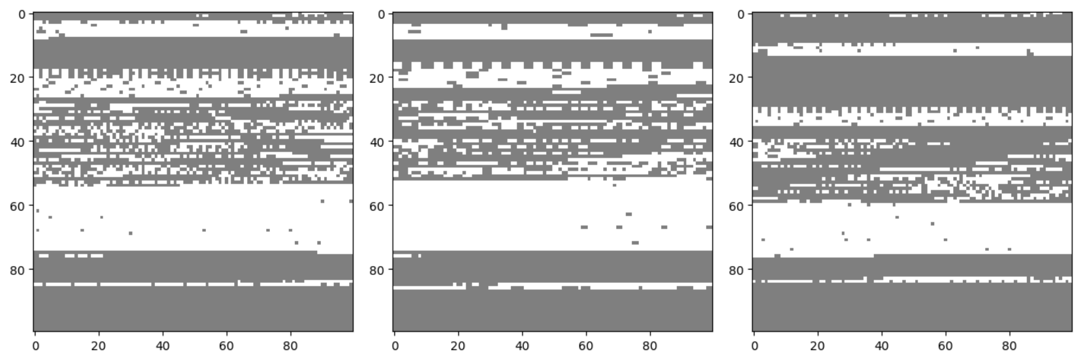


Figure 7.2: Original Predicted Image

Figure 7.2 depicts the images that our model deems most accurately represent the visual manifestation of Dontovo.A. The images, together with the model, are then transmitted to LIME for joint analysis. Subsequently, the system provides an explanation of the specific aspect of the image that led our Convolutional Neural Network (CNN) to determine that it represents the presence of Dontovo.A.

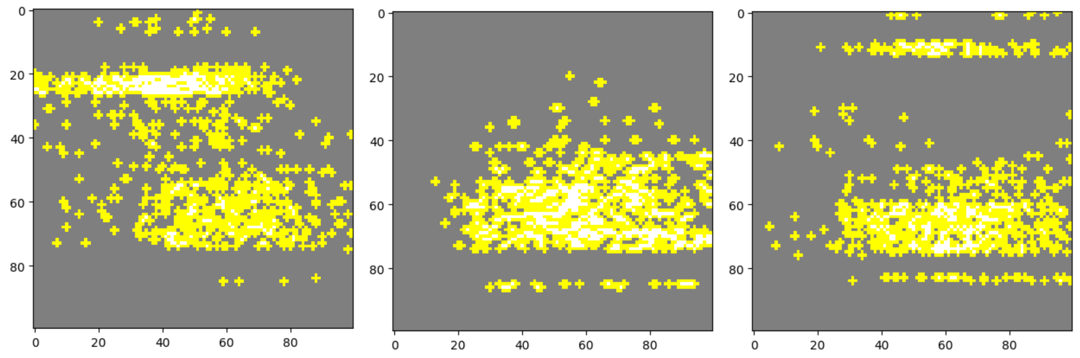


Figure 7.3: Excluded Part

Upon comparing Figure 7.2 with Figure 7.3, it becomes apparent that a significant portion of the pixelated grey area has been effectively removed from the image. This outcome mostly stems from the fact that our Convolutional Neural Network (CNN) model has reached a particular deduction that the excluded area is not useful for detecting the malware class, and this has been confirmed by LIME.

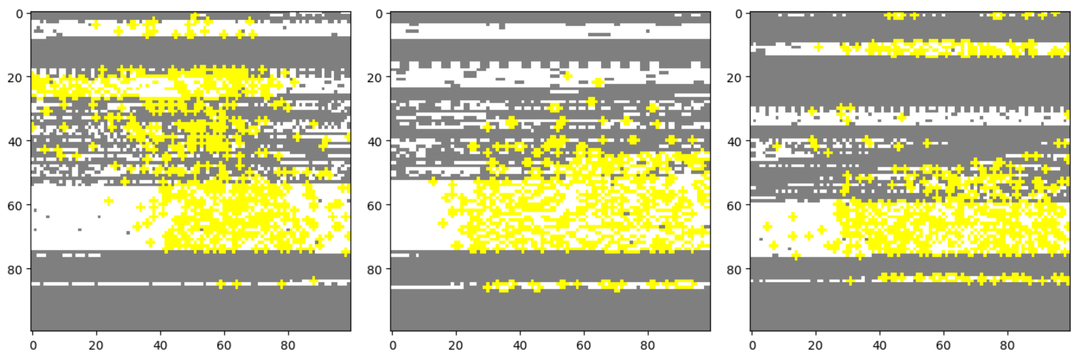


Figure 7.4: Differentiating Included and Excluded Part

Figure 7.4 represents a subset of Figure 7.2 and Figure 7.3, only focusing on the areas encompassed by the CNN prediction model. This subset disregards any external factors and solely considers the original image when making decisions.

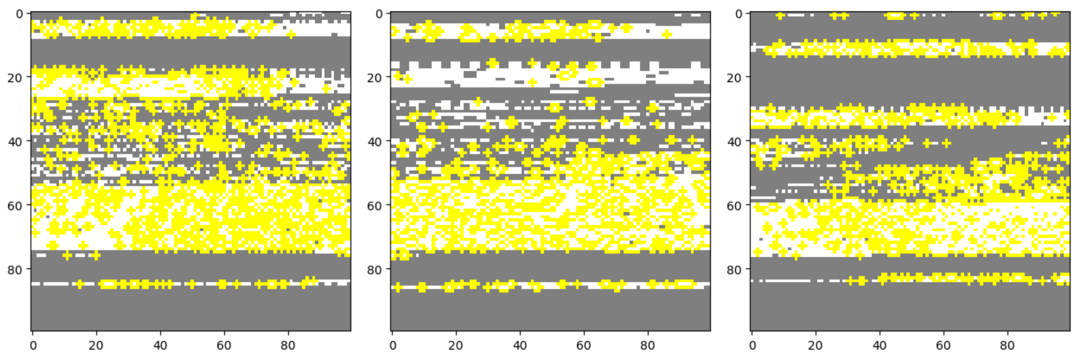


Figure 7.5: Color Coded Part

The output presented in Figure 7.5 depicts the outcomes obtained from the use of the LIME technique. This visualization effectively represents many facets of the convolutional neural network (CNN) model's prediction through the utilization of color codes. In this particular case, the yellow portions indicate that these regions have been utilized for the explicit aim of predicting and making informed choices.

The heatmap depicted in Figure 7.6 illustrates the binary malware representation in a chart-like manner. In this context, the data in Figure 7.6 is categorized into two groups, namely legitimate and invalid, based on the majority of the information provided. The blue segment represents valid data that was taken into account for the aforementioned prediction.

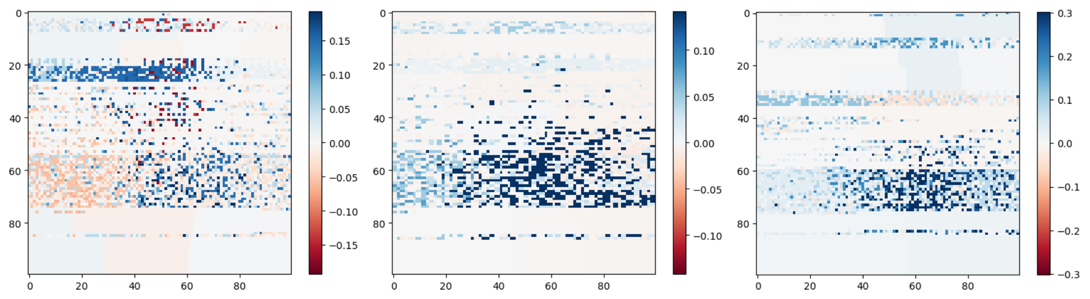


Figure 7.6: Heat Map

Upon comparing the heatmaps of three distinct samples of Dontovo.A malware class, a discernible pattern becomes evident. It is apparent that the largest cluster of the darker blue hue is consistently located in a similar area across all the pictures. As previously indicated, the use of these blue regions is of greatest importance in the decision-making process for the allocation of samples to their respective classes. In addition to this notable similarity, certain samples exhibit scattered occurrences of minor and insignificant quantities of red, which are deemed irrelevant for recognizing the malware class. Additionally, each of the three samples exhibits a thin, faint blue segment towards the end of all the heatmaps, resembling a straight line.

7.2 Involution

The involution neural network is proposed by the authors of [34]. The utilization of convolution has played a pivotal role in contemporary neural networks, serving as a catalyst for the fast advancement of deep learning in the area of machine vision. However, in the paper [34], the authors critically examine the essential ideas underlying standard convolution layers as applied to computer vision works, with a particular focus on channel-specific and spatial-agnostic aspects. In this study, they provide a novel synchronous procedure for deep neural architectures that involve the inversion of the established design principles of convolution. The authors refer to this operation as “Involution”. Furthermore, they want to clarify the recently famous self-attention procedure and incorporate it into the involution classification as an excessively sophisticated embodiment. The suggested involution mechanism has the potential to serve as a foundational component for constructing advanced neural networks designed for visual recognition. These networks can be utilized in various deep learning models to achieve high performance on well-known benchmarks such as ImageNet categorization, COCO detection, and segmentation, as well as Cityscapes segmentation. So, in light of the remarkable prospects for advancement in the domain of computer machine vision, we have made the decision to include the Involution Neural Network as a part of our work on the Malimg dataset.

When compared to regular or depth-wise convolution, involution kernels exhibit some unique properties. Unlike them, involution kernels are designed to incorporate transforms having opposite attributes in both the channel and spatial domains, giving them their name. The model incorporates equations that represent two linear transformations, which together form a bottleneck structure. This structure allows for efficient processing by controlling the intermediate channel dimension through

Layer (type)	Output Shape	Parameters
(InputLayer)	(None, 100, 100, 3)	0
(Involution)	((None, 100, 100, 3), (None, 100, 100, 9, 1,1))	26
(ReLU)	(None, 100, 100, 3)	0
(MaxPooling2D)	(None, 50, 50, 3)	0
(Involution)	((None, 50, 50, 3), (None, 50, 50, 9, 1, 1))	26
(ReLU)	(None, 50, 50, 3)	0
(MaxPooling2D)	(None, 25, 25, 3)	0
(Involution)	((None, 25, 25, 3), (None, 25, 25, 9, 1, 1))	26
(ReLU)	(None, 25, 25, 3)	0
(Flatten)	(None, 1875)	0
(Dense)	(None, 64)	120064
(Dense)	(None, 25)	1625
Total params: 121,767		
Trainable params: 121,761		
Non-trainable params: 6		

Table 7.2: Layers with output shape and parameters of the Involution

a reduction ratio “r”. The presence of Batch Normalisation and the utilization of non-linear activation functions that alternate between two linear projections are also suggested. In contrast to convolution kernels, the shape of involution kernels is also contingent upon the form of the input feature map. To construct the complete network incorporating involution, the authors in [34] adopt the design approach of ResNet [7], which involves the sequential arrangement of residual blocks. This choice is motivated by the sophisticated architecture of ResNet, which facilitates the exploration of novel concepts and enables effective comparisons. In this study, they propose the substitution of involution for 3 by 3 convolution at all bottleneck sites in both the stem and trunk of ResNet. Specifically, they suggest using 3×3 or 7×7 involution for categorization or dense prediction in the stem, and 7 by 7 involution for all tasks in the trunk. However, proceeded by keeping all the 1×1 convolutions for channel fusion and projection. The intricately reconfigured units come together to form a novel breed of exceptionally efficient foundational or fundamental networks, referred to as RedNet.

Redundancy in neural networks increases when spatial and channel information interweaves. However, the RedNet carefully decouples information exchanges for a good accuracy-efficiency trade-off. In particular, the kernel generation stage implicitly scatters one pixel’s channel dimension information to its spatial neighborhood, and the large and dynamic involution kernels receive information in an enriched receptive field. Interspersed with 1 x 1 convolutions, linear transformations are essential for channel data exchange. In summary, spatial-alone, channel-alone, and channel-spatial interactivity affect information transmission, enabling network miniaturization.

To examine the architectural aspects and operations of involution from an alternative standpoint, the final output after performing the involution operation is demon-

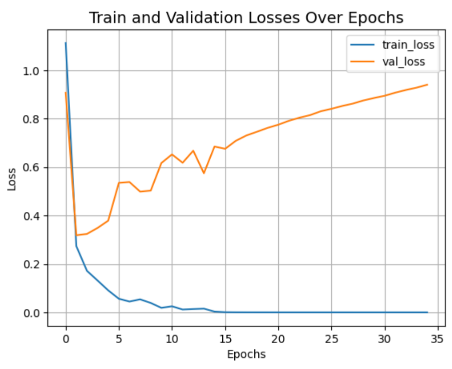


Figure 7.7: Involution Loss Curve

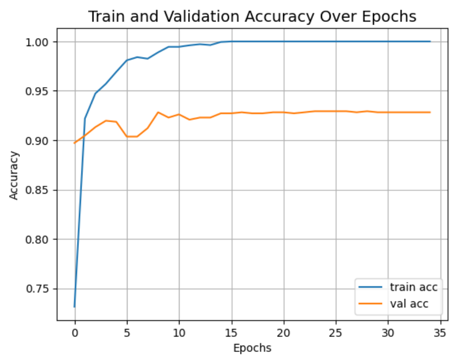


Figure 7.8: Involution Accuracy Curve

strated in Figure 7.9.

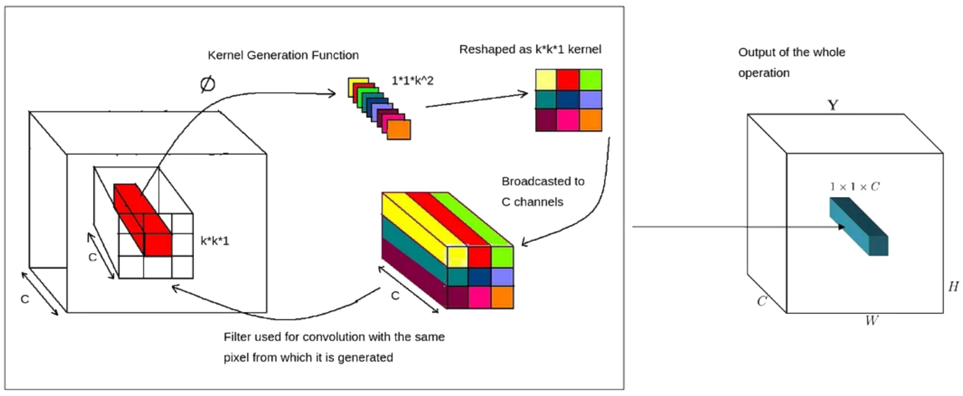

Kernel Generation Function
Reshaped as k*k*1 kernel
1*1*k^2
k*k*1
Broadcasted to
C channels
Filter used for convolution with the same
pixel from which it is generated
Output of the whole
operation
1 × 1 × C
H
W

Figure 7.9: Output from Involution

7.3 Transformers

7.3.1 Vision Transformer

According to the authors of [30], Vision Transformers or ViT needs a sizable amount of data. As a result, they advised training the model on a big dataset before fine-tuning it on a dataset of a smaller size. The model can outperform the most sophisticated Convolutional Neural Network models if this is followed.

However, compared to what Vision Transformers needs, our dataset is far smaller. Instead, we will use a small data set to train our ViT model. Shifted Patch Tokenization and Locality Self Attention are two methods suggested by the authors of [33] for achieving this.

As illustrated in Fig. 7.12, Moved Patch Tokenization involves taking a photograph before shifting it diagonally. Then, we combine the original image with diagonally shifted pictures. The concatenated image patches are then extracted. After that,

Layer (type)	**Output Shape**	**Parameters**
(InputLayer)	(None, 100, 100, 3)	0
(Sequential)	(None, 72, 72, 3)	7
(Patches)	(None, None, 108)	0
(PatchEncoder)	(None, 144, 64)	16192
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128

Table 7.3: Layers with output shape and parameters of the ViT

the patches are flattened into one dimension. Before being projected, the image is then treated to layer normalization. A visual representation of this Patch Tokeniza-

Layer (type)	**Output Shape**	**Parameters**
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(MultiHeadAttention)	(None, 144, 64)	66368
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Dense)	(None, 144, 128)	8320
(Dropout)	(None, 144, 128)	0
(Dense)	(None, 144, 64)	8256
(Dropout)	(None, 144, 64)	0
(Add)	(None, 144, 64)	0
(LayerNormalization)	(None, 144, 64)	128
(Flatten)	(None, 9216)	0
(Dropout)	(None, 9216)	0
(Dense)	(None, 2048)	18876416
(Dropout)	(None, 2048)	0
(Dense)	(None, 1024)	2098176
(Dropout)	(None, 1024)	0
(Dense)	(None, 25)	25625
Total params: 21,682,144		
Trainable params: 21,682,137		
Non-trainable params: 7		

Table 7.4: Layers with output shape and parameters of the ViT (cont.)

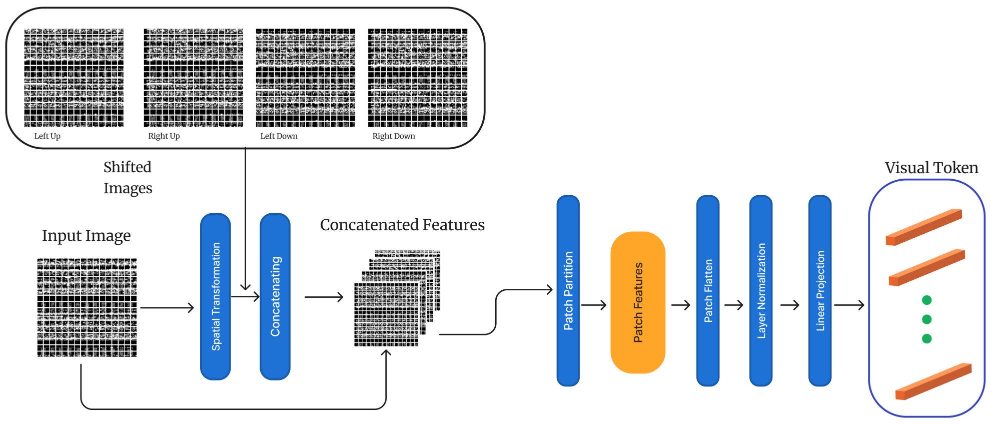


Figure 7.10: Illustrations of Shifted Patch Tokenization

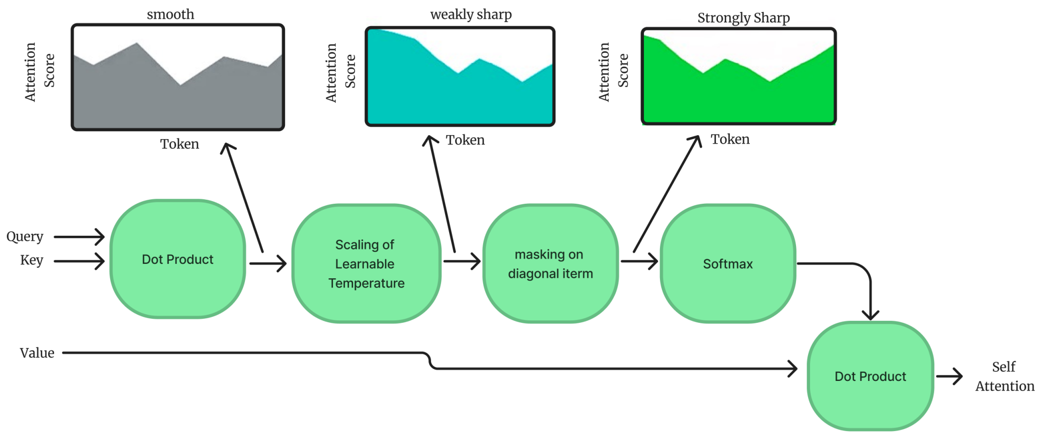


Figure 7.11: Locality Self Attention

tion technique is shown in Figure 7.10.

The Locality Self Attention technique is an additional approach that involves extracting a query, key, and value from a singular input source. The final step is utilizing the dot product operation to assess the degree of resemblance existing between our query and the key. The dot product in question will result in significant self-token relations, as opposed to interactions between different tokens. Prior to applying the softmax function, it is customary to adjust the dot product of the query and key by dividing it by the square root of the dimension of the key. This scaling is performed to mitigate the potential issue of encountering an extremely small gradient. Following the scaling of the dot product, the softmax function is employed. Furthermore, the softmax function has the potential to increase the probability of self-token links compared to interactions between different tokens. In order to address this concern, the authors of reference [61] suggest the implementation of a masking technique that involves concealing the diagonal of the dot product. The modification of the value is carried out by utilizing the attention weights as the concluding phase. The operational idea is depicted in Figure 7.11.

The Formula for the whole Locality self technique in its entirety is given below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{7.1}$$

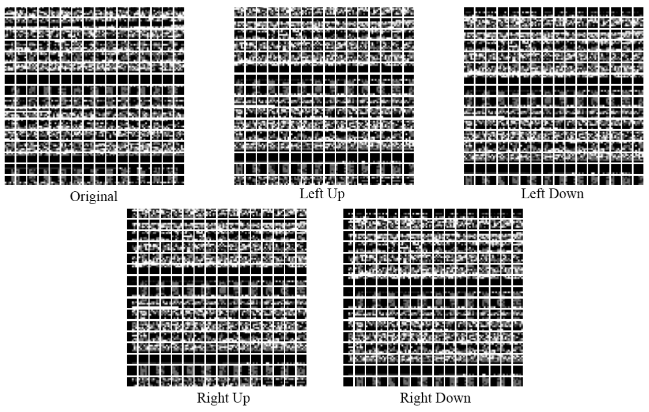


Figure 7.12: Tokenized Patches

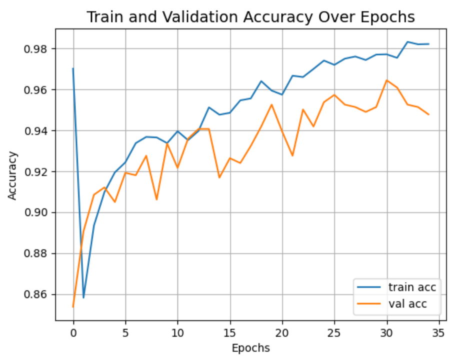


Figure 7.13: ViT Loss Curve

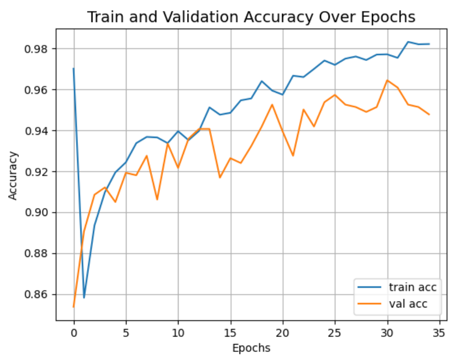


Figure 7.14: ViT Accuracy Curve

7.3.2 Compact Convolutional Transformers

The Compact Convolutional Transformer (CCT), often known as CCT, is a method proposed in a paper [38] for training transformers with limited data.

Layer (type)	**Output Shape**	**Parameters**
(InputLayer)	(None, 100, 100, 3)	0
(Sequential)	(None, 100, 100, 3)	0
(CCTTokenizer)	(None, 625, 128)	76224
(TFOpLambda)	(None, 625, 128)	0
(LayerNormalization)	(None, 625, 128)	256
(MultiHeadAttention)	(None, 625, 128)	131968
(StochasticDepth)	(None, 625, 128)	0
(Add)	(None, 625, 128)	0
(LayerNormalization)	(None, 625, 128)	256
(Dense)	(None, 625, 128)	16512
(Dropout)	(None, 625, 128)	0
(Dense)	(None, 625, 128)	16512
(Dropout)	(None, 625, 128)	0
(StochasticDepth)	(None, 625, 128)	0
(Add)	(None, 625, 128)	0
(LayerNormalization)	(None, 625, 128	256
(MultiHeadAttention)	(None, 625, 128)	131968
(StochasticDepth)	(None, 625, 128)	0
(Add)	(None, 625, 128)	0
(LayerNormalization)	(None, 625, 128)	256
(Dense)	(None, 625, 128)	16512
(Dropout)	(None, 625, 128)	0
(Dense)	(None, 625, 128)	16512
(Dropout)	(None, 625, 128)	0
(StochasticDepth)	(None, 625, 128)	0
(Add)	(None, 625, 128)	0
(LayerNormalization)	(None, 625, 128)	256
(Dense)	(None, 625, 1)	129
(TFOpLambda)	(None, 625, 1)	0
(TFOpLambda)	(None, 1, 128)	0
(TFOpLambda)	(None, 128)	0
(Dense)	(None, 25)	3225
Total params: 410,842		
Trainable params: 410,458		
Non-trainable params: 384		

Table 7.5: Layers with output shape and parameters of the CCT

The methodology employed in this approach bears resemblance to the process of Tokenization in the Vision Transformer (ViT) model. In this context, stochastic depth is utilized as a regularisation technique. It has a resemblance to a Dropout layer, with the distinction that instead of deactivating an individual node, an entire block of nodes inside a layer is rendered inactive. Used mainly prior to the residual blocks of a Transformer Encoder. Before reaching the transformer, the input undergoes Convolutional Tokenization. The components of the architecture include a convolutional layer, a pooling layer, and a reshape operation. Once the input data has undergone convolutional tokenization, it proceeds to the Transformer model with Sequence Pooling. The identical data augmentation procedures employed in prior models have been implemented. The incorporation of attention pooling, alternatively referred to as sequence pooling, has been implemented within our CCT model. In the Vision Transformer (ViT) model, only the feature maps that correspond to the tokens are utilized for the purpose of categorization. In the context of Compact Convolutional Transformers, it is noteworthy to mention that the resulting output of the Transformer Encoder undergoes a weighting process prior to being transmitted to the classification layer. Figure 7.15 depicts the functional mechanism of Compact Convolutional Transformers.

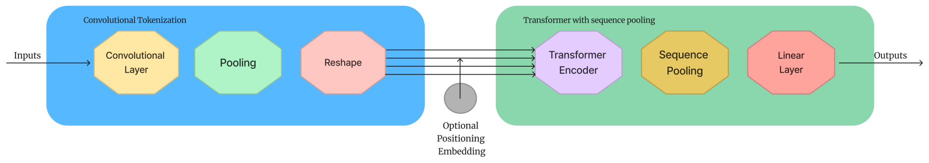


Figure 7.15: Architecture of CCT

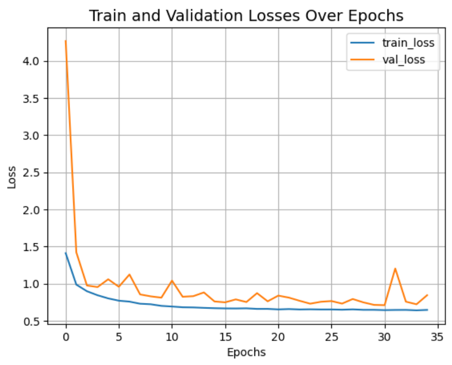


Figure 7.16: CCT Loss Curve

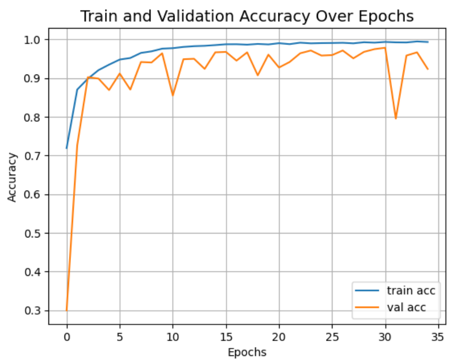


Figure 7.17: CCT Accuracy Curve

7.3.3 EANet

The idea of an External Attention Network was first suggested by the authors of [31].

Attention mechanisms, particularly self-attention, have assumed a progressively significant part in the deep feature depiction of visual computer works. Self-attention is a mechanism that enhances the feature representation at each position by calculating a weighted addition of features based on the pairwise affinities between all positions. This enables the model to capture long-range dependencies inside an individual sample. Nevertheless, it is worth noting that self-attention exhibits a quadratic complexity and fails to consider the potential correlation that may exist among distinct samples. This study introduces a novel attention operation referred to as external attention. This mechanism relies on two external, small-scale, comprehensible, and mutually accessible memory systems. Implementation of this mechanism is straightforward, involving the use of two cascaded linear layers and normalization layers. Notably, external attention serves as a convenient alternative to self-attention in prevalent architectures. The algorithmic complexity of external attention is linear, and it takes into account the connections between all data instances in an implicit manner. In order to enhance the external attention model for image classification, they also integrate the multi-head mechanism into the existing framework, resulting in an all-MLP architecture known as external attention MLP (EAMLP). The research conducted a series of comprehensive experiments on varieties of computer vision tasks, including picture categorization, object detection, semantic segmentation, instance segmentation, image production, and point cloud study. The results obtained from the method demonstrate similar or better performance compared to the self-attention operation and several of its versions. Additionally, the method achieves these results with significantly reduced computational and memory requirements.

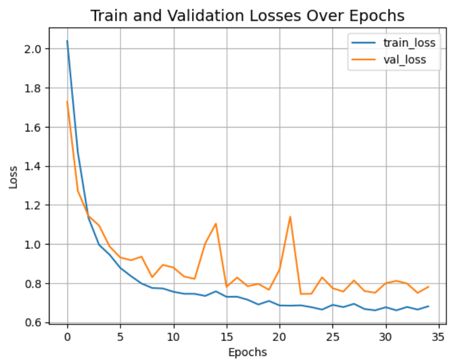


Figure 7.18: EANet Loss Curve

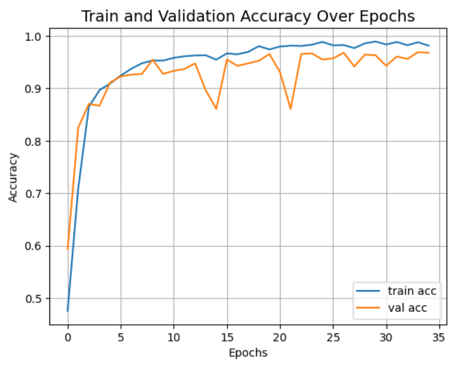


Figure 7.19: EANet Accuracy Curve

Chapter 8

Result & Analysis

8.1 Experimental Results

We compared our carefully designed CNN model results with those of the four other pre-trained models that incorporate the same network architecture. The names of these pre-trained models are Inception V3, DenseNet-201, ResNet 50, and VGG 16. Out of these 4 models, VGG 16 has the least amount of parameters (14,829 thousand of them to be exact). However, it will be astounding to note that our proposed model even far surpasses a popular image recognition model like VGG 16 in terms of parameter optimization. To be more specific, the model we have designed has high assessment metrics and great outcomes while maintaining only 2.1 million parameters. A model achieving an accuracy of 0.9930 during training and 0.9764 during testing with such a low amount of parameters is an impressive feat. These comparison values for the parameters can be viewed in Table 8.3. This distinguishes our model from others that are already in use. Other than that, certain measurements and metrics have been used in this study to properly compare results. These measures include recall, F1, accuracy, and precision. It is to be noted that the metrics for this study will be calculated according to the multiclass classification purposes.

Training Details	
Optimizer	Adam
Learning Rate	0.0001
Batch Size	32
Epoch	35
Image Size	100 X 100

Table 8.1: Parameters used to train our model

A model's accuracy is the proportion of correct predictions a model makes compared to all the right answers that are actually possible.

The level of precision is used to assess how accurate an identification percentage is in a model, basically the quality of a positive prediction. Take the total of all expected positive outcomes (TP) and divide it by the total of all expected and actual positive outcomes (TP + FP) to get the precision. It is calculated by applying the formula:

$$Precision = \frac{TP}{TP + FP} \tag{8.1}$$

Recall is a metric that helps us understand a model's ability to detect positive samples. It measures the proportion of accurate positive predictions among all possible positive predictions. The recall rate is calculated by finding the ratio of true positives (TP) to total data (TP + FN). The formula for determining recall is given below:

$$Recall = \frac{TP}{TP + FN} \tag{8.2}$$

The F1 Score is frequently used to assess the performance of machine learning (ML) models. This measure is created by averaging accuracy and recall. The F1 Score is known by using the following formula:

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{8.3}$$

Deep Learning Architecture	**Accuracy**	**Epochs**	**Recall**	**Precision**	**Loss**	**F1-Score**
Inception V3	0.7909	35	0.7780	0.8044	0.6659	0.7909
ResNet50	0.9418	35	0.9418	0.9418	0.1979	0.9418
DenseNet-201	0.8879	35	0.8879	0.8879	0.6606	0.8879
VGG16	0.9407	35	0.9407	0.9407	0.1992	0.9407
Proposed Model	0.9795	35	0.9795	0.9795	0.0796	0.9795

Table 8.2: Comparison of Validation using various measures

The testing accuracy of our model, along with the other measures, shows potential when compared to older methods and research that use a dataset like Malimg. The cost of the preprocessing power needed by the alternative models was higher but our method is different as it requires fewer parameters. The comparison between the validation and testing phases and their respective measurements including accuracy, recall, precision, etc. can be found in tables 8.2 and 8.3.

We made sure that the number of epochs, picture size, and learning rate were all kept constant across all pre-trained models so that we can draw an unbiased comparison with our proposed model.

The Malimg dataset has been subjected to analysis using contemporary machine-learning models that have been recently developed. Some of these models were introduced as recently as 2021. This is conducted in order to establish a comprehensive comparison between our proposed model and newer computer vision technology. The dataset is trained using modern models such as Involution, Vision Transformer

(ViT), Compact Convolutional Transformer (CCT), and External Attention Network (EANet). These aforementioned models are characterized by their complex nature, offering numerous advanced benefits. As an example, the parameters of the CCT (410,842) exhibit a notably low value, surpassing even the already deemed low value of our proposed model (2.1 million). These comparison values for the parameters can be seen in Table 8.4. All of these models have been run on the Malimg dataset and have yielded good and comparable outcomes. However, our proposed model surprisingly outperformed all of them in terms of testing accuracy with a value of 97.64%.

Deep Learning Architecture	Accuracy	Recall	Precision	Loss	F1 Score	Parameters
Inception V3	0.7602	0.7462	0.7759	0.8684	0.7608	21,854,009
ResNet50	0.9283	0.9236	0.9256	0.2834	0.9246	24,406,937
DenseNet-201	0.8694	0.8729	0.8729	0.7894	0.8729	18,754,009
VGG16	0.9411	0.9427	0.9427	0.2291	0.9427	14,829,913
Proposed Model	0.9764	0.9708	0.9729	0.1568	0.9718	2,143,553

Table 8.3: Comparison of Testing using various measures

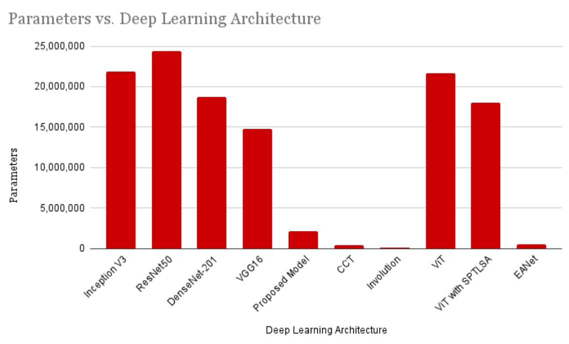


Figure 8.1: Comparison of parameters

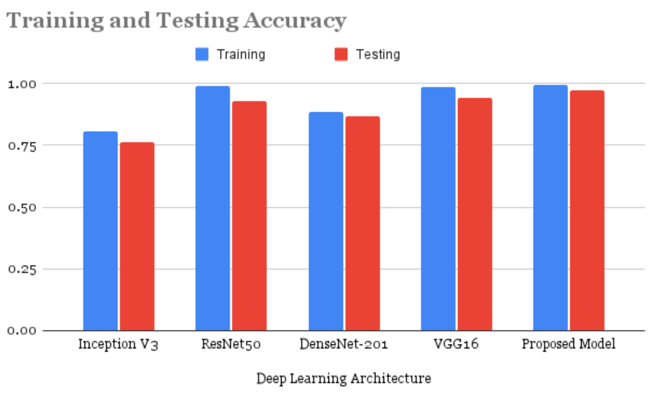


Figure 8.2: Training vs Testing Accuracy of Models

Accuracy Comparison with Transformers		
Architecture Name	**Testing Accuracy (%)**	**Parameters**
Compact Convolutional Transformers (CCT)	97.43%	410,842
Involution	92.83%	121,767
Vision Transformer (ViT)	95.93%	21,682,144
Vision Transformer with Shifted Patch Tokenization and Locality Self Attention	95.18%	18,033,696
External Attention Network (EANet)	96.79%	500,128
Proposed Model	97.64%	2,143,553

Table 8.4: Accuracy Comparison with Transformers and other state of the art models

Graph of model accuracy and model loss of our projected methodology is shown below:

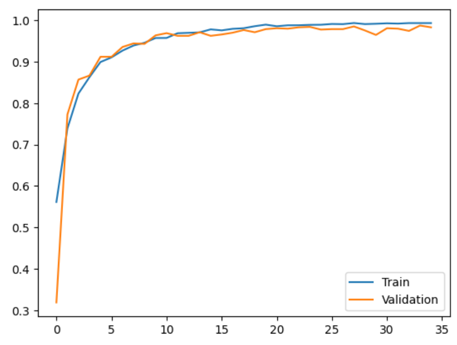


Figure 8.3: Accuracy of our model

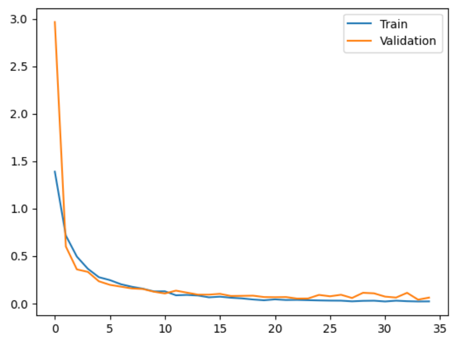
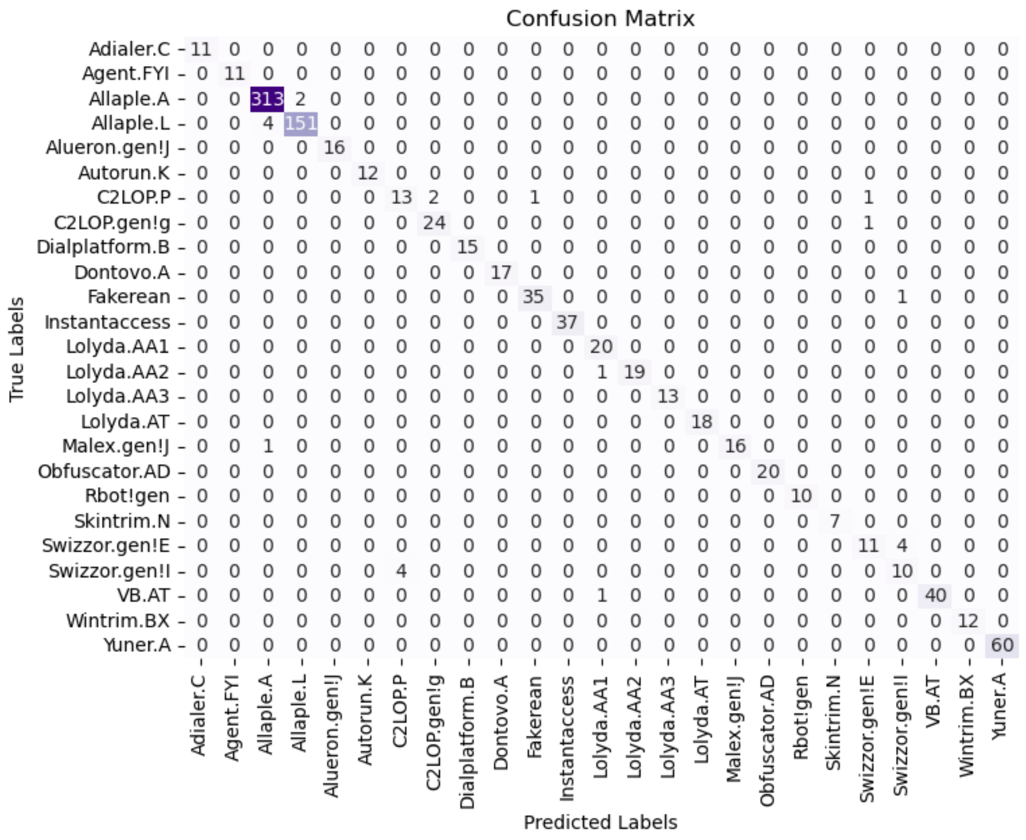


Figure 8.4: Loss of our model

The accuracy and loss curves are shown in Figures 8.3 and 8.4, respectively. The curves demonstrate that our model has a very excellent fit with no under-fitting or over-fitting because there is little difference between training and validation. Additionally, the accuracy of its predictions is fairly stable. Dropouts and group normalization were used to make the model stable and less complex. These methods allow neurons to learn more on their own.

8.2 Confusion Matrix

Confusion Matrix of our suggested methodology is displayed here. On the horizontal side the predictions are showed as well as on the vertical side true responses are showed. As we can see, our suggested model is acting quite satisfactory predicting it.



	Adialer.C	Agent.FYI	Allaple.A	Allaple.L	Alueron.gen!J	Autorun.K	C2LOP.P	C2LOP.gen!g	Dialplatform.B	Dontovo.A	Fakerean	Instantaccess	Lolyda.AA1	Lolyda.AA2	Lolyda.AA3	Lolyda.AT	Malex.gen!J	Obfuscator.AD	Rbot!gen	Skintrim.N	Swizzor.gen!E	Swizzor.gen!I	VB.AT	Wintrim.BX	Yuner.A
Adialer.C	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Agent.FYI	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Allaple.A	0	0	313	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Allaple.L	0	0	4	151	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Alueron.gen!J	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Autorun.K	0	0	0	0	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C2LOP.P	0	0	0	0	0	0	13	2	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
C2LOP.gen!g	0	0	0	0	0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Dialplatform.B	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Dontovo.A	0	0	0	0	0	0	0	0	0	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fakerean	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Instantaccess	0	0	0	0	0	0	0	0	0	0	0	37	0	0	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA1	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA2	0	0	0	0	0	0	0	0	0	0	0	0	1	19	0	0	0	0	0	0	0	0	0	0	0
Lolyda.AA3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0
Lolyda.AT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0
Malex.gen!J	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0
Obfuscator.AD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0
Rbot!gen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
Skintrim.N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0
Swizzor.gen!E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	4	0	0	0
Swizzor.gen!I	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0
VB.AT	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	40	0	0
Wintrim.BX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	12	0
Yuner.A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60

Figure 8.5: Confusion Metrix

8.3 Classification Report

Classification report is shown in the table below:

Classes	**Precision**	**Recall**	**F1-Score**	**Support**
Adialer.C	1.00	1.00	1.00	11
Agent.FYI	1.00	1.00	1.00	11
Allaple.A	0.98	1.00	0.99	315
Allaple.L	1.00	0.97	0.98	155
Alueron.gen!J	1.00	1.00	1.00	16
Autorun.K	1.00	1.00	1.00	12
C2LOP.P	0.76	0.76	0.76	17
C2LOP.gen!g	0.89	0.96	0.92	25
Dialplatform.B	1.00	1.00	1.00	15
Dontovo.A	1.00	1.00	1.00	17
Fakerean	0.97	0.97	0.97	36
Instantaccess	1.00	1.00	1.00	37
Lolyda.AA1	0.91	1.00	0.95	20
Lolyda.AA2	1.00	0.95	0.97	20
Lolyda.AA3	1.00	1.00	1.00	13
Lolyda.AT	1.00	1.00	1.00	18
Malex.gen!J	1.00	0.94	0.97	17
Obfuscator.AD	1.00	1.00	1.00	20
Rbot!gen	1.00	1.00	1.00	10
Skintrim.N	1.00	1.00	1.00	7
Swizzor.gen!E	0.69	0.60	0.64	15
Swizzor.gen!I	0.53	0.57	0.55	14
VB.AT	1.00	0.98	0.99	41
Wintrim.BX	1.00	0.92	0.96	12
Yuner.A	1.00	1.00	1.00	60
Accuracy			0.97	934
Macro Average	0.95	0.94	0.95	934
Weighted Average	0.97	0.97	0.97	934

Table 8.5: Classification Report

Chapter 9

Analysis & Future Research

9.1 Analysis

Due to the presence of several interconnected layers, our model has shown great stability. Our model is able to avoid the typical problems brought on by overfitting by incorporating several methods like batch normalization and dropouts.

Furthermore, a comparison was made between our suggested model and other cutting-edge models, namely Vision Transformer (ViT), CCT (Compact Convolutional Transformer), External Attention Network (EANet), and Involution. These mentioned complex and advanced models have been recently introduced in the domain of computer vision. These models serve as exemplary instances that might inspire and inform the ongoing development of our own model. All these state-of-the-art models yielded favorable and comparable outcomes on the Malimg dataset, which had been mainly optimized for our proposed model. Furthermore, some of these models, like CCT, have remarkably low parameters beyond the capabilities of our proposed model. Ultimately, we made the decision to employ the explainable artificial intelligence (AI) methodology known as LIME in order to get a deeper comprehension and provide a clear demonstration of the rationale behind our model's classification of malware classes. This approach can be crucial in advancing our research and enhancing the progress of our model's development.

The assessment measures show that our model is likewise very well-balanced in terms of the outputs with minimal parameters when using images with a lower resolution-based foundation. This model may also be used by an integrated web-based system embedded with our deep learning models to identify the exact class of malware that is infecting their computers and take appropriate actions accordingly.

There are a few issues that need to be handled with the research despite the model's well-balanced ability to detect malware families. As is well known, there are more and more various sorts of cyberattacks every day, which results in a steady influx of fresh malware. The Malimg Dataset is unquestionably not updated to reflect these new kinds of computer infections, and its data count remains constant. Working with datasets that automatically update themselves or creating one that can do the same might be a solution to this problem.

9.2 Future Research

We intend to evaluate our proposed model against more state-of-the-art models in the future. Our methodology can be enhanced by integrating sophisticated mechanisms from other advanced models. Increasing the number of models that are trained and studied would facilitate the achievement of this objective. For instance, the Compact Convolutional Transformer (CCT) exhibits a notably low parameter value, and it can be feasible to integrate the technology responsible for maintaining this low parameter count into our own model. Other than that, examples of newer models that we plan to run a comparison on in the future include the ConvMixer Transformer, Swin Transformer, etc. Future plans also include working on a reduction in time and space-based complexity.

We plan to keep on gathering fresh malware photos from more sources and families, as well as details about the malware's greyscale structure patterns, image quality, etc. This will help us to broaden the limitations of the study and increase the amount of raw information we have available. One of the best ways to collect more data about malware is to retrieve it from other existing datasets like Microsoft BIG 2015, Malex, MalNet, etc. We will have the capacity to do a comparative analysis encompassing all the datasets in this manner.

Because of how small and light our model is, we can easily publish it to a website or create an application that anyone with ordinary access to the internet may utilize on low-powered computing devices. However, we need to incorporate a technology that converts the malware found in the victim's computer into their binary image representations beforehand. Indeed, it is a verifiable fact that numerous antivirus software applications possess the capability to promptly notify users of the identification of malicious software present on their computer systems. Nevertheless, there exists a subset of individuals who harbor reservations and exhibit reluctance toward the utilization of antivirus software. Our primary objective is not to promote ourselves as an antivirus solution, but rather to provide educational resources on various categories of malware, their origins, and their functionalities in a comprehensive and informative manner. Everyone will be able to easily identify the spyware that may have potentially harmed or will harm their machine using this method. This will also raise awareness about the different types of computer infections and help the general public be more knowledgeable and better equipped against cyber attacks.

Chapter 10

Conclusion

To diversify our techniques and properly prove the effectiveness of our work, we worked with four pre-trained CNN models. These trained models including ResNet-50, Inception-V3, VGG-16, and DenseNet-201 helped us to show that we developed a very optimized model. We were able to detect malware in greyscale photos with a 97.64% accuracy rate after running and evaluating our model (in the testing phase). We used various metrics to further evaluate and strengthen our comparative study namely accuracy, precision, recall, F1 Score, etc. The final model provided satisfying results with a surprisingly low amount of parameters (2.1 million to be precise). This low amount definitely helped us to set our model apart from the others that we tested as low parameters make a model more versatile and help conserve computer resources.

Our suggested model was tested on the Malimg dataset, and a comparative study was conducted among recently introduced state-of-the-art models, including Vision Transformer (ViT), CCT (Compact Convolutional Transformer), External Attention Network (EANet), and Involution. It is imperative to acknowledge the limited availability of literature pertaining to the performance of these advanced models on binary image malware datasets. As a result, our comprehensive comparative research represents one of the initial efforts in this area. Furthermore, the comparative analysis facilitated the generation of innovative ideas to enhance the advancement of our own model. While our model did not outperform them in all aspects, it demonstrated a minor improvement in testing accuracy. In the end, we opted to employ an explainable artificial intelligence (AI) technique known as LIME in our study. This approach allows us to thoroughly demonstrate the precise manner in which our model assigns samples to distinct classes. Furthermore, we are also very hopeful of making further changes in the future by including a wider dataset, AI components, and the incorporation of the comparison analysis of more advanced and state-of-the-art models.

Now, why is our project important? Malware intrusions have become a severe security concern in recent years, resulting in hefty losses in every aspect of our lives. At this time, malware variations and their classifications are rapidly evolving. Undeniably, it is paramount to have profound knowledge regarding malware classification so that we can swiftly and effectively categorize malware for better understanding and control of our ever-evolving cyberspace. Consequently, with this comprehensive research using diverse deep learning methods we can exhibit a coherent and less asset-consuming image processing system. We used various methods and measurements in order to mold a well-organized system for better identification and analysis of different malware attributes. Hence, this all-inclusive research will not only help normal civilians to better understand how to deal with cyber attacks but help malware and forensic analysts to understand, adapt and design a better deterrent against today's ever-evolving malware.

Bibliography

[1] M. Schultz, E. Eskin, F. Zadok, and S. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings 2001 IEEE Symposium on Security and Privacy. SP 2001*, 2001, pp. 38–49. DOI: 10.1109/SECPRI.2001.924286.

[2] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04, Seattle, WA, USA: Association for Computing Machinery, 2004, pp. 470–478, ISBN: 1581138881. DOI: 10.1145/1014052.1014105. [Online]. Available: https://doi.org/10.1145/1014052.1014105.

[3] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, D. Zamboni, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 108–125, ISBN: 978-3-540-70542-0.

[4] M. Siddiqui, M. C. Wang, and J. Lee, "A survey of data mining techniques for malware detection using file features," in *Proceedings of the 46th Annual Southeast Regional Conference on XX*, ser. ACM-SE 46, Auburn, Alabama: Association for Computing Machinery, 2008, pp. 509–510, ISBN: 9781605581057. DOI: 10.1145/1593105.1593239. [Online]. Available: https://doi.org/10.1145/1593105.1593239.

[5] L. Nataraj, S. Karthikeyan, G. Jacob, and B. Manjunath, "Malware images: Visualization and automatic classification," Jul. 2011. DOI: 10.1145/2016904.2016908.

[6] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[7] ——, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV].

[8] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556 [cs.CV].

[9] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. arXiv: 1512.00567 [cs.CV].

[10] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl 4 md : A deep learning framework for intelligent malware detection," 2016.

[11] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2018. arXiv: 1608.06993 [cs.CV].

[12] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2018, pp. 1–5. DOI: 10.1109/NTMS.2018.8328749.

[13] S. Ni, Q. Qian, and R. Zhang, "Malware identification using visualization images and deep learning," *Computers Security*, vol. 77, pp. 871–885, 2018, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2018.04.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404818303481.

[14] N. Bhodia, P. Prajapati, F. Di Troia, and M. Stamp, "Transfer learning for image-based malware classification," Jan. 2019. DOI: 10.5220/0007701407190726.

[15] D. Gibert, C. Mateu, and J. Planes, "A hierarchical convolutional neural network for malware classification," in *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852469.

[16] D. Gibert, C. Mateu, J. Planes, and R. Vicens, "Using convolutional neural networks for classification of malware represented as images," *Journal of Computer Virology and Hacking Techniques*, vol. 15, pp. 15–28, 2019.

[17] M. F. Rafique, M. Ali, A. S. Qureshi, A. Khan, and A. M. Mirza, *Malware classification using deep learning based feature extraction and wrapper based feature selection technique*, 2019. DOI: 10.48550/ARXIV.1910.10958. [Online]. Available: https://arxiv.org/abs/1910.10958.

[18] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019. DOI: 10.1109/ACCESS.2019.2906934.

[19] A. Azab and M. Khasawneh, "Msic: Malware spectrogram image classification," *IEEE Access*, vol. PP, pp. 1–1, Jun. 2020. DOI: 10.1109/ACCESS.2020.2999320.

[20] K. Barure, Z. Shaikh, S. More, S. Kalbhor, and Y. Ingle, "Malware classification using deep learning," Jun. 2020.

[21] L. Ghouti and M. Imam, "Malware classification using compact image features and multiclass support vector machines," *IET Information Security*, vol. 14, no. 4, pp. 419–429, 2020. DOI: https://doi.org/10.1049/iet-ifs.2019.0189. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-ifs.2019.0189. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-ifs.2019.0189.

[22] D. Gibert, C. Mateu, and J. Planes, "Hydra: A multimodal deep learning framework for malware classification," *Computers Security*, vol. 95, p. 101 873, 2020, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101873. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820301462.

[23] ——, "Orthrus: A bimodal learning architecture for malware classification," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206671.

[24] S. Tang, S. Yuan, and Y. Zhu, "Data preprocessing techniques in convolutional neural network based on fault diagnosis towards rotating machinery," *IEEE Access*, vol. 8, pp. 149 487–149 496, 2020. DOI: 10.1109/ACCESS.2020.3012182.

[25] T. The Son, C. Lee, H. Le-Minh, N. Aslam, M. Raza, and N. Long, "An evaluation of image-based malware classification using machine learning," in. Nov. 2020, pp. 125–138, ISBN: 978-3-030-63118-5. DOI: 10.1007/978-3-030-63119-2_11.

[26] S. Wang and Y. Zhang, "Densenet-201-based deep neural network with composite learning factor and precomputation for multiple sclerosis classification," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 16, pp. 1–19, Jun. 2020. DOI: 10.1145/3341095.

[27] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Bao, "Byte-level malware classification based on markov images and deep learning," *Computers Security*, vol. 92, p. 101 740, 2020, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101740. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820300262.

[28] Ö. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, pp. 87 936–87 951, 2021. DOI: 10.1109/ACCESS.2021.3089586.

[29] K. Bakour and H. Ünver, "Deepvisdroid: Android malware detection by hybridizing image-based features with deep learning techniques," *Neural Computing and Applications*, vol. 33, pp. 1–18, Sep. 2021. DOI: 10.1007/s00521-021-05816-y.

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[31] M.-H. Guo, Z.-N. Liu, T.-J. Mu, and S.-M. Hu, *Beyond self-attention: External attention using two linear layers for visual tasks*, 2021. arXiv: 2105.02358 [cs.CV].

[32] M. Landesman, *The first 25 years of malware*, Mar. 2021. [Online]. Available: https://www.lifewire.com/brief-history-of-malware-153616.

[33] S. H. Lee, S. Lee, and B. C. Song, *Vision transformer for small-size datasets*, 2021. arXiv: 2112.13492 [cs.CV].

[34] D. Li, J. Hu, C. Wang, X. Li, Q. She, L. Zhu, T. Zhang, and Q. Chen, *Involution: Inverting the inherence of convolution for visual recognition*, 2021. arXiv: 2103.06255 [cs.CV].

[35] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. Manjunath, "Malware detection using frequency domain-based image visualization and deep learning," Jan. 2021. DOI: 10.24251/HICSS.2021.858.

[36] T. K. Toai, R. Senkerik, V. T. X. Hanh, and I. Zelinka, "Malware classification by using deep learning framework," in *Computational Intelligence Methods for Green Technology and Sustainable Development*, Y.-P. Huang, W.-J. Wang, H. A. Quoc, L. H. Giang, and N.-L. Hung, Eds., Cham: Springer International Publishing, 2021, pp. 84–92, ISBN: 978-3-030-62324-1.

[37] *Types of Malware*, en-ZA, Jan. 2021. [Online]. Available: https://www.kaspersky.co.za/resource-center/threats/malware-classifications (visited on 01/12/2023).

[38] A. Hassani, S. Walton, N. Shah, A. Abuduweili, J. Li, and H. Shi, *Escaping the big data paradigm with compact transformers*, 2022. arXiv: 2104.05704 [cs.CV].

[39] S. Seneviratne, R. Shariffdeen, S. Rasnayaka, and N. Kasthuriarachchi, "Self-supervised vision transformers for malware detection," *IEEE Access*, vol. 10, pp. 103 121–103 135, 2022. DOI: 10.1109/access.2022.3206445. [Online]. Available: https://doi.org/10.1109%2Faccess.2022.3206445.

[40] S. S. H. Shah, A. R. Ahmad, N. Jamil, and A. u. R. Khan, "Memory forensics-based malware detection using computer vision and machine learning," *Electronics*, vol. 11, no. 16, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11162579. [Online]. Available: https://www.mdpi.com/2079-9292/11/16/2579.

[41] E. e. staff, *Over 95 per cent of 2022's new malware threats aimed at Windows*, en-US, Dec. 2022. [Online]. Available: https://eandt.theiet.org/content/articles/2022/12/over-95-per-cent-of-2022-s-new-malware-threats-aimed-at-windows/ (visited on 01/12/2023).

[42] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Dialer:Win32/Adialer.C.

[43] en. [Online]. Available: https://www.f-secure.com/v-descs/agent.shtml.

[44] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Allaple.A.

[45] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Allaple.L.

[46] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Alureon.gen!J.

[47] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Autorun.K&threatId=-2147369124.

[48] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.P.

[49] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/C2Lop.gen!G&threatId=139219.

[50] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Dialer:Win32/DialPlatform.B.

[51] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Dontovo.A&threatId=-2147342037.

[52] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Win32/FakeRean.

[53] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?name=dialer:win32/instantaccess.

[54] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Lolyda.AA&threatId=-2147345828.

[55] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=PWS:Win32/Lolyda.AT.

[56] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Malex.gen!J.

[57] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Trojan:Win32/Obfuscator.AD!MTB&ThreatID=2147757929.

[58] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Backdoor:Win32/Rbot.gen.

[59] en. [Online]. Available: https://www.f-secure.com/v-descs/rbot.shtml.

[60] [Online]. Available: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Wintrim.BX.

[61] [Online]. Available: https://peterbloem.nl/blog/transformers.

[62] *Ransomware Statistics in 2022: From Random Barrages to Targeted Hits*, en. [Online]. Available: https://dataprot.net/statistics/ransomware-statistics/ (visited on 01/12/2023).

[63] Y. P. Yeap, *Council Post: Why Ransomware Costs Businesses Much More Than Money*, en. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/04/30/why-ransomware-costs-businesses-much-more-than-money/ (visited on 01/12/2023).