HYBRID MALWARE DETECTION AND CLASSIFICATION IN REAL-TIME BY DEEP LEARNING TECHNIQUES

2022
MASTER THESIS
COMPUTER ENGINEERING

Hussein Sadraldeen ALMUSAWI

Thesis Advisor
Assist.Prof.Dr. Adnan ALAJEELI

HYBRID MALWARE DETECTION AND CLASSIFICATION IN REAL-TIME BY DEEP LEARNING TECHNIQUES

Hussein Sadraldeen ALMUSAWI

T.C.
Karabuk University
Institute of Graduate Programs
Department of Computer Engineering
Prepared as
Master Thesis

Thesis Advisor
Assist.Prof.Dr. Adnan ALAJEELI

KARABUK
August 2022

I certify that in my opinion the thesis submitted by Hussein Sadraldeen ALMUSAWI titled “HYBRID MALWARE DETECTION AND CLASSIFICATION IN REAL-TIME BY DEEP LEARNING TECHNIQUES” is fully adequate in scope and in quality as a thesis for the degree of Master of Science.

Assist.Prof.Dr. Adnan ALAJEELI ....
Thesis Advisor, Department of Computer Engineering

This thesis is accepted by the examining committee with a unanimous vote in the Department of Computer Engineering as a Master of Science thesis. August 18, 2022

Examining Committee Members (Institutions)		Signature
Chairman	: Assist.Prof.Dr. Abdulkadir TAŞDELEN (AYBU)	....
Member	: Assist.Prof.Dr. Adnan ALAJEELI (KBU)	....
Member	: Assist.Prof.Dr. Oğuzhan MENEMENCİOĞLU (KBU)	....

The degree of Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Prof. Dr. Hasan SOLMAZ ....
Director of the Institute of Graduate Programs

“I declare that all the information within this thesis has been gathered and presented in accordance with academic regulations and ethical principles and I have according to the requirements of these regulations and principles cited all those which do not originate in this work as well.”

Hussein Sadraldeen ALMUSAWI

ABSTRACT

M. Sc. Thesis

HYBRID MALWARE DETECTION AND CLASSIFICATION IN REAL-TIME BY DEEP LEARNING TECHNIQUES

Hussein Sadraldeen ALMUSAWI

Karabük University
Institute of Graduate Programs
The Department of Computer Engineering

Thesis Advisor:
Assist. Prof. Dr. Adnan ALAJEELI
August 2022, 78 pages

In the consequence of communication between people, the sending of crucial data, particularly between them, the downloading of a great number of programs and files are attractive for the cybercriminals. Because the cybercriminals are becoming more sophisticated in their methods, there is a need to develop a robust security mechanism against malicious software, which is growing daily and has become more risky and more complex.

In this research project, we presented two new datasets that belong to the same samples that we collected. The first is built on visualization (static analysis) whereas the second is built on API call sequences (dynamic analysis) to detect malware in different methods in case it is encrypted or uses obfuscation techniques.

In this study, different models of deep learning used to protect against malware by identifying and categorizing the family to which it belongs are presented. The first dataset, which contains benign and malware images after converted from malware binary numbers, used our custom model and three of the common pretrained network models of CNN (VGG16, Inception V3, and Resnet50). The second dataset, which contains API call sequences, uses two algorithms of RNN (LSTM and GRU). Also, with the second dataset, a CNN was used with API call sequence numbers after reshaping and normalizing it.

Finally, we choose three best models for real-time detection and classification: one for CNN using the first dataset, one for RNN using the second dataset, and one for the CNN model using the second dataset after normalizing and reshaping it. We selected the best models depending on their accuracy, number of parameters, and cost-effectiveness (memory).

Our framework achieved high accuracy in all models and when testing for examples of malware that belong to the same families but are absent from the dataset that was gathered. These models were found and categorized in a manner that was both very accurate and carried out in real time.

Key Words : CNN, LSTM, GRU, Hybrid analysis, API call, Malware images.
Science Code : 92403

ÖZET

Yüksek Lisans Tezi

DERİN ÖĞRENME TEKNİKLERİYLE HİBRİT ZARARLI YAZILIM TESPİTİ VE GERÇEK ZAMANLI SINIFLANDIRMA

Hussein Sadraldeen ALMUSAWI

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Dr. Öğr. Üyesi. Adnan ALAJEELI
Ağustos 2022, 78 sayfa

İnsanlar arasındaki iletişim sonucunda, özellikle kendi aralarında önemli verilerin gönderilmesi, çok sayıda program ve dosyanın indirilmesi siber suçlular için cazip hale gelmektedir. Siber suçlular yöntemlerinde daha karmaşık hale geldikleri için, her geçen gün büyüyen ve daha riskli ve daha karmaşık hale gelen kötü amaçlı yazılımlara karşı sağlam bir güvenlik mekanizması geliştirmeye ihtiyaç vardır.

Bu araştırma projesinde, topladığımız aynı örneklere ait 2 yeni veri seti sunulmuştur; bunlardan ilki görselleştirme statik analizi üzerine inşa edilmiştir, ikincisi ise şifrelenmiş olması veya gizleme teknikleri kullanması durumunda kötü amaçlı yazılımları farklı yöntemlerle tespit etmek için bir API çağrı dizileri dinamik analizi üzerine inşa edilmiştir.

Bu çalışmada, ait olduğu aileyi tanımlayarak ve kategorize ederek kötü amaçlı yazılımlara karşı korunmak için kullanılan farklı derin öğrenme modelleri sunulmuştur. Kötü amaçlı yazılım ikili sayılarından dönüştürüldükten sonra iyi huylu ve kötü amaçlı yazılım görüntülerini içeren ilk veri setinde özel modelimiz ve CNN'in yaygın ön eğitimli ağ modellerinden üçü (VGG16, Inception V3 ve Resnet 50) kullanılmıştır. API çağrı dizilerini içeren ikinci veri setinde, RNN (LSTM ve Gru) dışında iki algoritma kullanılmıştır. Ayrıca, ikinci veri setinde, yeniden şekillendirildikten ve normalleştirildikten sonra API çağrı dizisi numaraları ile bir CNN kullanılmıştır.

Son olarak, gerçek zamanlı tespit ve sınıflandırma için en iyi üç modeli seçtik: biri ilk veri setini kullanan CNN, diğeri ikinci veri setini kullanan RNN ve diğeri de normalleştirip yeniden şekillendirdikten sonra ikinci veri setini kullanan CNN modeli. Doğruluklarına, parametre sayılarına ve maliyet etkinliğine (bellek) bağlı olarak en iyi modelleri seçtik.

Aynı ailelere ait olan ancak toplanan veri setinde bulunmayan kötü amaçlı yazılım örnekleri için test yapıldığında tüm çerçeve tüm modellerde yüksek doğruluk elde etti. Bu modeller hem çok doğru hem de gerçek zamanlı olarak gerçekleştirilecek şekilde bulundu ve kategorize edildi.

Anahtar Kelimeler : CNN, LSTM, GRU, Hibrit analiz, API çağrısı, Kötü Amaçlı Yazılım görüntüleri.

Bilim Kodu : 92403

ACKNOWLEDGMENT

To begin with, from the bottom of my heart, I would like to express my thankfulness to God…..

To my supervisor, Assistant Professor Dr. Adnan ALAJEELI, for his contribution, and interest in the production of this thesis…...

To my family, who have stood by my side throughout this journey.......

To my wonderful mother, who has never wavered in her love and support……

To my father, whose faith in me was instrumental to my success, will always be in my memory........

CONTENTS

Page

Page

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVITIONS INDEX

ABBREVITIONS

CNN	: Convolutional Neural Network
API	: Application Programming Interface
RNN	: Recurrent Neural Network
SVM	: Support Vector Machine
PE	: Portable Executable
LSTM	: Long Short-Term Memory
BiLSTM	: Bidirectional Long Short-Term Memory
DLL	: Dynamic link library
BN	: Bayesian Network
GRU	: Gated Recurring Unit
BGRU	: Bidirectional Gated Recurring Unit
RBM	: Restricted Boltzmann Machine
IoT	: Internet of Things
DNN	: Deep Neural Network
SVD	: Single Value Decomposition
DCNN	: Deep Convolutional Neural Network
MKL	: Multiple Kernel Learning
SPP-Net	: Spatial Pyramid Pooling Network
AUC	: Area Under the Curve
MLP	: Multilayer Perceptron
NB	: Naive Bayes
LR	: Logistic Regression
DT	: Decision Tree
KNN	: K-Nearest Neighbors
RF	: Random Forest
PUA	: Potentially Unwanted Applications

PART 1

INTRODUCTION

1.1. OVERVIEW

Malware, also known as malicious software or software with malicious intent, is created by hackers for the purpose of performing a specific action, such as monitoring a user's computer to steal personal information (known as spyware) or encrypting data and holding it for ransom (known as ransomware). Figure 1.1 from the AV-TEST Institute's research depicts the meteoric rise of malware over the last decade. Over 450 thousand new harmful and potentially unwanted applications (PUA) software samples are logged daily by the institution [1]. Obfuscation, cryptography, and many other techniques are used by malware developers to keep their software out of the reach of security systems. This is a major factor in the wide variety of malicious software [2].

At the present, an investigation into malicious software may be broken down into two distinct categories: static analytics and dynamic behavior analytics. The primary distinction between the two lies in whether or not the program in concern is really executed [3].

When malicious code is encrypted, compressed, or obfuscated, static analysis is rendered useless despite its speed and efficiency. Obfuscation is a way to modify or enhance source code without impacting the functionality of the original. In this scenario, malware is detected by a dynamic analysis technique that examines the program as it runs [4].

Deep learning is a kind of artificial intelligence that relies heavily on computer simulations of the brain's neural networks. As with other aspects of data science, such as statistical data prediction and modeling, it is crucial. There is widespread agreement that deep learning algorithms are an effective tool for detecting malware, and they have found use in a wide variety of niche areas, including the protection of sensitive user information [5], vulnerability recognition [6], and others.

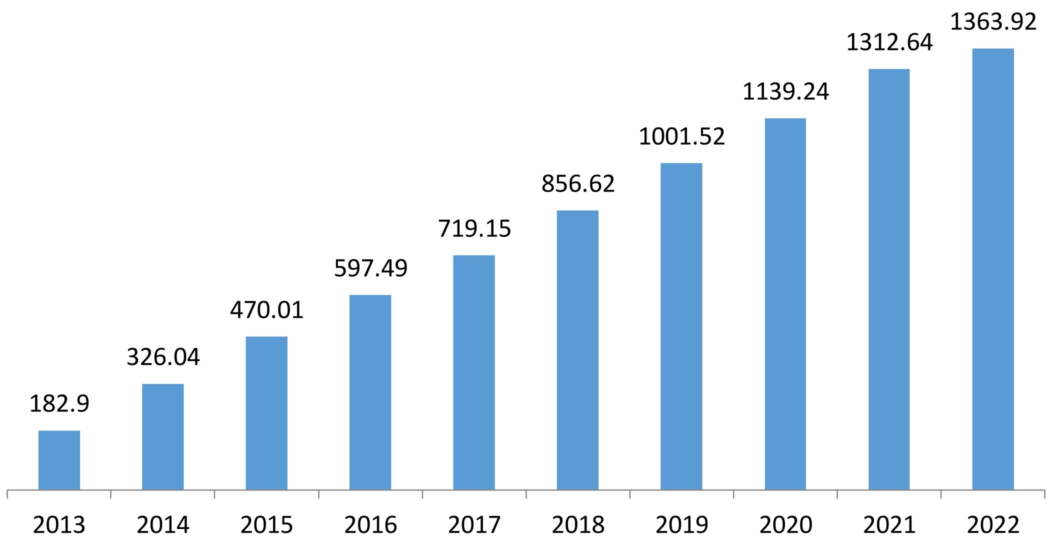


Figure 1.1. Malware statistics in millions for the previous ten years.

In this study, our primary emphasis is on hybrid approaches, which integrate aspects of static and dynamic analysis to improve performance based on two deep learning algorithms. The first is a CNN algorithm that identifies malware based on visualizing malware binary numbers as grayscale images and API call sequences after converting them to a 2D array and normalizing them. The second algorithm is RNN, which is utilized to deal with the API call sequences for each harmful program application.

1.2. COMMON TYPES OF MALWARES

Malicious software, or "malware," is defined as any program that was created with the goal of doing damage. There is a wide variety of malicious software, and each of these programs has its own unique technique for infiltrating your computer system. These techniques could involve snooping on you, trying to steal your private information, encrypting your essential data, or causing various kinds of harm to your systems.

1.2.1. Virus

In simple terms, a virus is malicious software that infiltrates your system and causes damage. It may slow down your device or steal your personal information. A virus can infect other files by copying itself and then attaching to them. When one virus infects a network, it has the potential to launch a denial-of-service attack or encrypt data in an attempt to hold the network hostage.

1.2.2. Worm

A self-replicating and infectious computer software is known as a "worm." Worms may delete data and files from a computer. By taking advantage of weaknesses in operating systems, a worm may travel from one computer network to another. The vast majority of the time, they are distributed by mass email with malicious attachments.

1.2.3. Adware

Adware is malicious software that secretly delivers advertisements to a user's computer without their knowledge or consent. Users often activate adware without their knowledge when they attempt to install legal programs that come packed with adware. There is also the risk that visitors may be tricked into downloading even more malicious software because of these advertising.

1.2.4. Trojans

Trojans are harmful programs that masquerade as legitimate programs in order to deceive users into downloading and installing them. As far as malware goes, this is the worst kind. Trojans can be used for a wide range of bad things, such as collecting sensitive financial information, stealing sensitive information (login credentials, electronic accounts), stealing computer system resources, or harming your information or network in some other way.

1.2.5. Spyware

Spyware is a kind of program that is placed on computers and mobile devices to spy on their activities and record what they find by installing itself into a user's system. This kind of malware is able to access sensitive data such as passwords and email addresses after being given administrative privileges. Key logging is a common method through which this is accomplished.

1.2.6. Backdoor

A backdoor is a way into a computer system without the need to authenticate using the system's normal authentication procedures. As a result, hackers may get access to sensitive data and databases used by the targeted applications.

1.2.7. Ransomware

Ransomware is software that is meant to locate all files on a computer, encrypt those files, and then transmit messages to the user. It requires customers to pay a ransom to regain access to their data. The distribution of ransomware often occurs either via network weaknesses or through downloaded files. The data on the computer is encrypted, and then it utilizes a key for the encryption that is only known to the attackers.

1.3. PROBLEM STATEMENT

The detection of a zero-day assault is notoriously tough. Anti-malware tools are frequently unsuccessful since new malware does not have a signature in the anti-malware database [7].

Obfuscated strings, used by most of the malicious software, conceal the instructions that inform an infected machine when to do certain actions. Obfuscation prevents static code analyzers from identifying harmful data. Before the malicious program is executed, the original code is hidden.

1.4. OBJECTIVE

The main goal is to collect samples (malware and benign) in portable executable file format and first convert them to images, and second, extract API call sequences from samples to create two new datasets.

Develop a hybrid deep learning method that can detect and accurately classify malware in real time. This will be done by using deep learning methods to develop three different models based on static and dynamic approaches to identify and classify malicious software in case it is encrypted or uses techniques to hide itself. These models will be used to overcome the zero-attack.

1.5. CONTRIBUTION

We can summarize contributions as:

- Created two new datasets, each containing 7,513 malware and 1,000 benign in 30 classes (29 classes of malware family and 1 class of benign): The first dataset includes grayscale images after converting all samples to grayscale images, while the second dataset (a csv file) includes API call sequences for each sample extracted by the Python Pefile library.
- After conducting experiments, we chose the most efficient models trained on two datasets in terms of accuracy, memory consumption, and speed to detect and classify malware in real time. These models successfully predicted real malware after passing samples that belong to the same families but do not exist in the dataset.
- The first 50 API call sequences were extracted by the Python Pefile module for each malicious and benign program to create the dynamic dataset as a CSV file to make detection as quick as possible.
- Designed a light-weighted 2D-CNN model with a 7x7 array of API call sequences after reshaping by calling the first 49 API call sequences of the dynamic dataset, which is a small array, and experiments proved the model has high accuracy and very high speed in detection and classification.

- The year 2022 was the source of the great majority of the samples of malicious software that were gathered.

1.6. THESIS STRUCTURE

In the first part of this study, we'll look at what malware is, what kinds of analysis can be used to find it, and what new techniques are being used to find and classify it.

In the second part, we'll go over the most important studies that have been done on how deep learning can be used to find malware.

The third part presents a detailed study of deep learning, focusing on CNNs and RNNs as well as the models associated with each network type.

In the fourth part, we'll talk about how to make our dataset, our method, and the methods we used in our study.

In the fifth part, we will talk about the study's results and what they mean.

The study's conclusion and what will come next in future work will be in the sixth part.

PART 2

LITERATURE REVIEW

An overview of the most significant approaches used in prior investigations is provided in this section based on deep learning for three types of analysis: static, dynamic, and hybrid analysis, and a comparison between the techniques used in each type in terms of model, accuracy, advantage, and dataset used.

2.1. STATIC ANALYSIS APPROACH

Malware may be analyzed using a method known as static analysis, which does not involve actually running the malware. The binary files of malware are turned around into readable format so that specialists may better comprehend the malware's intended behavior. Assembly codes and file header information are also retrieved as static parts. This form of investigation is usually a quick and simple way to examine malware without needing to run it.

In 2011, Nataraj et al. proposed a new and entirely distinct technique for visualizing and analyzing malware. Malware PE (portable executable) is displayed as a binary image and can identify malware based on visual similarities in images belonging to different families by identifying substantial visual similarities in image textures [8].
The malicious binary was converted into an image by first converting the Portable Executable (PE) file into an 8-bit vector and then translating that vector into a grayscale image. Since every pixel in the image is made up of 8 bits, with 0 representing black and 255 representing white, and grayscale gradients being in between those two values.

Jin et al. presented a way to detect malware by using the Autoencoder model, one of the unsupervised deep learning models, by understanding the functional properties of malware, and then examining the Autoencoder's reconstruction error in order to achieve malware classification and detection [9].

The ResNeXt model is one of the CNN models used to detect malware with a dataset consisting of the Malimg dataset, which consists of 25 families [8]. Then add malware samples from the websites of VirusTotal [10], Malshare [11], and VirusShare [12]. The result proved that ResNeXt is better than the ResNet and InceptionNet models in features and performance [13].

CNN's VGG19 fine-tuning is another of CNN's models used to fit visualization images of malware and solve imbalanced samples in the Malimg dataset without using data augmentation by undersampling the dataset via defining the highest limit of malware files, which must belong to every category of malware [14].

Another method used to solve the dataset imbalance in the Malimg dataset and the overfitting problem by using a fine-tuned CNN and data augmentation, as well as using a color-map to convert the Malimg grayscale dataset to color, explained that the calculation cost was reduced and the accuracy of the results improved over previous studies [15].

It achieved high classification performance to detect malware on the basis of a technique displaying malicious software in the manner of entropy graphs based on structural entropy and then extracting features using a combined framework consisting of CNN and Support Vector Machine (SVM) as classifiers [16].

Both packed and unpacked malware are effectively detected by the architecture, which consists of CNN and multiple class SVM for classification, fine-tuned CNN with Softmax for classification and fine-tuned CNN with multiple class SVM for classification [17].

Some studies used two approaches in deep learning, CNN and RNN, by making use of minhash to produce feature images based on the combination of the original codes and the RNN's predicted codes, and then training a CNN to identify feature images to detect malware [18].

The imbalance in the dataset is considered one of the most important problems facing researchers, which is the lack of evenly collected models within the dataset, which causes low accuracy with little data .To deal with imbalanced data problems, CNN's DenseNet model is implemented on four data sets(Malimg, Microsoft BIG 2015 [19], Malevis [20], and Malicia [21]), three are used to train the data, and the fourth is used for test data. [22].

Anandhi et al. suggested two models: VGG-3 and DenseNet. This method visualizes malware as Markov images and extracts textures from Markov images using the Gabor filter [23].

Mai et al. offer up A decomposing deep neural network with the purpose of improving the malware variants detection approach, which consists of deep convolutional neural networks (DCNN) and Single Value Decomposition (SVD), to process the problem of significant computational resource consumption and time cost by splitting the pre-trained conventional operation into two smaller convolution processes [24].

They used a disassembly technique to gather executable file samples and convert them to bytes and asm files. In this method, visualization techniques are utilized with data augmentation to extract key features from the data samples, then the samples will be converted into three-channel RGB images, and to boost the detection method's performance, a model consisting of three layers was implemented: the SEResNet50 layer, which consists of two models (ResNet50 and SENet), the Bidirectional LSTM layer, and the Attention layer [25].

The categorization of malware is accomplished by the use of multi-channel visualization, and the LeNet5 model was suggested. Assembly instructions and malware binary bytes were used to generate a matrix using Word2Vec. The findings demonstrate that the approach for classifying malware has a high level of consistency in its correctness, and further examples of malware were found [26].

To identify malicious applications, it employs machine learning and signature matching algorithms. It builds a random forest algorithm with the features in the Portable Executable file header to recognize the malware. This RF design will be employed to analyze a sample and determine whether or not it is harmful by utilizing the signature matching approach, which compares the MD5 hash of the sample to the database, which contains the MD5 hashes of the known malware and its families [27]. The drawback of this strategy is that it can be evaded by new malware.

Markel and Bilzor suggested a malware detection approach for Windows PE files that relied on information in the PE header. The method builds the model using the metadata for the file. The testing findings demonstrated that the executable's metadata may be utilized to distinguish between goodware and malicious software. On the constructed feature of the portable executable header, three machine learning algorithms were applied; however, the Decision Tree algorithm beat the logistic regression and NB algorithms [28,29]. The fundamental disadvantage of the NB classification is that it performs poorly when the data characteristics in the training data are correlated [29].

Nagano and Uda suggested a malicious detection mechanism in which runtime files were analyzed using static analytical techniques to acquire features such as hex dumps, DLL imports, and assembly code. These features were employed by the paragraph vectors, and the KNN and SVM algorithms were based on them. The investigation employed 3600 malicious files and had an accuracy rate of 99% [29,30]. Simple obfuscation strategies, on the other hand, can circumvent the suggested methodology [29].

Darabian et al. proposed approach for detecting IoT malware utilizing opcode sequences employs 247 IoT malicious files and 269 benign apps. It has been effectively detecting IoT malware, benign apps, and polymorphic malware by integrating sequential pattern mining methods with machine learning approaches (SVM, KNN, MLP, Decision Tree, Random Forest, and AdaBoost) and obtaining higher than 99% accuracy and an F1-score [31]. The disadvantage of this strategy is that some obscured code is never disclosed.

Lu offered an innovative and efficient technique for automatically learning malware opcode sequence patterns First, utilized the disassembly program IDA Pro to get the malware's opcode sequence. The feature vector representation of the opcode is then learned using the word embedding approach. Finally, for malware detection, it suggested a two-stage LSTM model. Training was conducted on a dataset including 969 malicious with 123 goodware software. The result proves that in the best situation, The suggested approach can obtain a mean AUC of 99% and a mean AUC of 98.7%. The study used only the opcodes and ignored the operands, which may have shown other sensitive information between malicious and goodware files [32].

Sanz suggested using machine learning algorithms to categorize Android applications. They retrieved three distinct set of features: the frequency of printed strings, the app's varied permissions, and the app's permissions obtained from the Android store. They performed studies on 820 data from seven different families using Random Forest, J48, KNN, Bayesian Networks, Naive Bayes, and SVM as classifier and determined that Bayes TAN was the best case with an AUC of 0.93 [33,34]. The study's disadvantage Malicious applications were not considered [34].

Milosevic et al. presented two static analysis strategies for Android malware that use machine learning. The first method relies on permissions. The logistic regression model as a classifier achieves 82% for three metrics (precision, recall and F-score). The alternative method collects features about code files. Following the inversion of Android applications into many Java files, followed by a bag-of-words model is utilized to produce feature vectors using the natural language processing approach. The framework incorporates logistic regression, SVM with SMO, simple logistic regression, and AdaBoostM1 with SVM algorithms, which obtain 95.8% Precision, 95.7% Recall, and 95.6% F1-score [34,35]. The authors' dataset in this study was tiny, and no tests were performed on the larger sample [34].

The most significant past research using static analysis with artificial intelligence approaches is outlined in Table 2.1 following.

Table 2.1. A summary of static malware detection methods.

Authors	**Model**	**Advantage**	**Accuracy**	**Dataset**
Sun and Qian [18]	CNN RNN	high accuracy and good generalization	99.5%	Microsoft BIG 2015
Hemalatha et al. [22]	DenseNet	Detect new malware samples and effective against obfuscation attacks.	98.23% 98.46% 98.21% 89.48%	Malimg Microsoft BIG 2015 Malevis Malicia
Anandhi et al. [23]	VGG-3 DenseNet	The detection, classification, and execution times have been enhanced.	99.94% 98.98%	Malimg Microsoft BIG 2015
Mai et al. [24]	Dec-DCNN	Reduce the computational resource consumption and time cost of malware detection.	98.5%	Microsoft BIG 2015
Jian et al. [25]	SEResNet50 + Bi-LSTM + Attention	High performance	98.31%	Microsoft BIG 2015
Mitsuhashi and Shinagawa [14]	VGG19	High accuracy and solved the problem of sample data imbalance	99.72%	Malimg
Jin et al. [9]	Autoencoder	results against redundant API injection	93%	Korea University
Go et al. [13]	ResNeXt	good performance in malware detection	98.8%	Malimg, VirusTotal, Malshare, VirusShare
Vasan et al. [15]	CNN	Using minimum run-time, identify hidden code, disguised malware, and malware family variations	98.82% 97.35%	Malimg IoT-Android
Xiao et al. [16]	CNN -SVM Entropy graphs	resistant to the effects of data imbalance and obfuscation techniques	99.7% 100%	Malimg Microsoft BIG 2015
Vasan et al. [17]	CNN -SVM	flexible, practical, and efficient by detecting new malware in 1.18 s	99% 98%	Malimg Virusshare
Qiao et al. [26]	LeNet5	High, stable accuracy and classifies new malware.	98.76%	Microsoft BIG 2015
Acharya et al. [27]	RF	identify malware that has been altered or polished by the hackers.	99%	MD5 hashes dataset
Markel and Bilzor [28]	DT LR NB	The implementation of the NB prediction model is simple and straightforward. It can do effectively with irrelevant datasets.	97% 94.5%	122799 Malware 42003 Benign

Nagano and Uda [30]	SVM k-NN	KNN is incredibly easy to implement and can be upgraded for very little cost as new samples with established class labels.	99%	MWS 2016 Malware Dataset
Darabian et al. [31]	SVM - KNN MLP - DT RF AdaBoost	Identifying polymorphic Internet of Things malicious files.	99%	VirusTotal
Lu [32]	LSTM	efficacy of the suggested opcode approach for identifying and classifying malware.	99%	Opcode dataset
Sanz [33]	RF - KNN BN - SVM	The suggested approach allows for quick categorization of benign applications.	93%	Collected by the researchers
Milosevic et al. [35]	SVM -SMO + LR + Simple LR + AdaBoostM1 + SVM	Good efficiency may be attained by using the ensemble learning approach.	95.8% 95.7% 95.6%	Collected by the researchers

2.2. DYNAMIC ANALYSIS APPROACH

The malicious functions are discovered via dynamic analysis while the program is operating. It digs deep into malware's code obfuscations, which static analysis may find difficult to grasp, and explores true functionality behind them. This method of malware analysis is always considered as the most efficient. Furthermore, the dynamic approach necessitates a closed and isolated setting with sufficient monitoring.

User programs in Windows need interfaces like kernel32.dll and user32.dll to communicate with the operating system and its hardware and software parts. These interfaces are provided by dynamic link libraries (DLL). Figure 2.1 shows how calls to the Windows API can be made. The Win32 API is the name of this interface. For instance, when a user application needs the Win32 API method for reading a file, the operation immediately goes to the NtReadFile procedure in the ntdll.dll kernel case. The NtReadFile function then calls the kernel mode service method. The best way to keep an eye on a program is to actively keep track of its API calls. The functionality of APIs cannot be classified as either dangerous or goodware. In other words, the malware exploits standard API calls to commit illegal actions. Both harmful and benign files have access to the same API. Only by analyzing the context of a series of API requests can malicious and benign activities be distinguished [36].

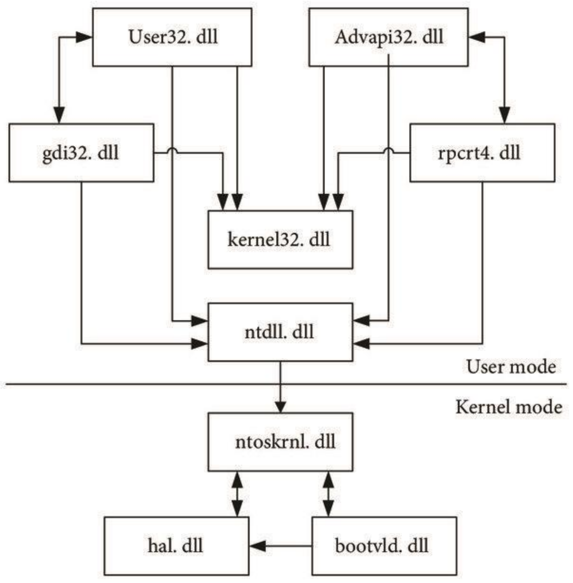


Figure 2.1. Windows API call mechanism [36].

A new dataset that portrays the behavior of malicious software and is offered by Catak et al. includes API calls that were executed on the Windows operating system. A method of categorization that is based on the different types of malwares was developed. In this particular piece of work, the LSTM classification technique was used; it is an approach to classification that is employed commonly when working with sequential data [37].

The cuckoo sandbox was used to retrieve the malware's API call sequence. After applying certain filters and doing some sorting, the repeated API calls were removed down to the distinct API sequences. The sequence was vectorized using the word2vec approach, and 21,378 samples from the Virus Share website were used as test datasets. On the massive dataset, BLSTM was shown to have the best malware detection performance when compared to GRU, BGRU, LSTM, and Simple RNN [38].
There are several ways to combine the LSTM model with other machine learning techniques, such as the Random Forest method, which uses API statistics as well, in order to design a system architecture that is a good alternative method. This

architecture can be designed by combining two algorithms. The malicious samples were chosen at random from VirusShare and VirusTotal, and the sequence data preparation method was explored in order to eliminate redundant data. Experiments demonstrate that the combined classifier outperforms machine learning or deep learning on its own [39].

The Malbert model suggested by Xu et al. [40] is a dynamic analysis-based deep learning model for identifying malicious Windows apps. The experiments made use of the Ki dataset [41] , which had a total of 44262 samples, whilst the Catak dataset [42] , which contained a total of 7207 samples, was used for the second dataset. The results demonstrated that the model outperformed previous models in detecting anomalies in perturbed test data.

To discover previously unidentified malware, researchers have developed a multimodal deep learning system made up of an autoencoder layer in the first place, numerous layers of Restricted Boltzmann Machines (RBM), and an associative memory layer. Each prediction for the detection of unidentified dangerous software takes 0.1 seconds [43].

Jindal et al. developed a neural network for malware detection that learns spontaneously from dynamic analysis reports that describe behavioral information rather than relying on feature engineering. The model is based on document classification principles and uses word sequences in reports to determine whether or not a report is from a malicious binary. The result is better than the previous works in terms of performance and can be used effectively [44].

In the IoT environment, A new deep learning malicious program identification system relying on behavior was built. Combining behaviors with the Stack Autoencoder yielded the greatest detection results. The model can acquire deeper abstract semantic features and improve detection precision by 1.5 percent on average, according to the results of the experiments [45].

A new method for detecting malware was proposed by Deep Graph of CNN during the conversion of API call sequences extracted from Cuckoo sandbox environments to behavior graphs. The experiment was applied to a dataset of more than 40,000 malware programs. The results showed the possibility of detecting malware through graphs converted from API calls [46].

Tang and Qian presented a deep learning and visualization strategy by extracting API calls based on dynamic way, after that producing important images feature that reflect virus behavior using color mapping algorithms. Finally, CNN is utilized to categorize the feature images. The result shows that visualization and CNN are efficient for malware categorization [47].

An API call-based deep learning framework was utilized to detect and categorize malware. LSTM and GRU recurrent neural networks were used to build the model. When the two architectures are compared, LSTM outperforms GRU. According to the test results, the model using the LSTM structure has an accuracy rate of 97.3 percent in binary classification and 56.05 percent in multiple-class classification [48].

The most important findings from previous research that combined dynamic analysis with deep learning methods are summarized in the table that follows (Table 2.2).

2.3. HYBRID ANALYSIS APPROACH

The benefits of a static approach and a dynamic approach are integrated in this method so that the best of both worlds may be achieved.

Huang et al. developed a method for merging static and dynamic images using two VGG-16 network models, the first for hybrid images and the second for static visualization with the dataset from virussign.com [49].

Table 2.2. A summary of dynamic malware detection methods.

Authors	Model	Advantage	Accuracy	Dataset
Xiaofeng et al. [39]	LSTM – RF	Combined classifier outperforms machine learning or deep learning on its own.	95.7%	Virus Share Virus Total
Liu and Wang [38]	GRU, BGRU LSTM, BLSTM Simple RNN	BLSTM increases model performance for sequence classification problems more than other models.	93.70% 93.72% 97.55% 97.85% 95.70%	Virus Share
Catak et al. [37]	LSTM	High level of accuracy	98.50%	API call dataset
Xu et al. [40]	Encoders + Attention layer	On altered test samples, it has a high detection rate and surpasses previous models.	99.98% 99.82%	Ki dataset Catak et al.
Ye et al. [43]	Autoencoder RBMs	detect newly unknown malware	98.20%	Comodo Cloud Security Center
Jindal et al. [44]	CNN-LSTM-Attention	outperforms similar approaches for malware classification	87% 86.7%	Vendor Dataset Ember Dataset
Xiao et al. [45]	Autoencoder -ML	Learn deeper features and enhance detection precision by 1.5 % on average.	98%	VX Heaven
Oliveira et al. [46]	LSTM	Malware detection using graphs generated from API calls	99%	API call dataset
Tang and Qian [47]	CNN	For malware categorization, visualization with color images and CNN are efficient.	98%-99%	Virus Share
Aditya et al. [48]	LSTM-Adam LSTM-RMSProp GRU-Adam GRU-RMSProp	The best binary classification model is achieved using LSTM and the RMSProp	96.44% 97.30% 96.44% 96.62%	Catak dataset Andrade et al. [50]

An efficient system is used that consists of static analysis, dynamic analysis, and image processing approaches relied on CNN as well as LSTM for detecting and classifying zero-day malware with two datasets, dataset1 is Malimg [8] and dataset 2 was collected from VirusSign [51] and VirusShare [12] , the results proved that BLSTM has the best performance [52].

A hybrid malware detection system was built by coupling a Bidirectional LSTM with a Spatial Pyramid Pooling Network (SPP-Net). The goal of this system was to secure

Internet of Things equipment and decrease the effect of malware using obfuscation techniques, It detects encrypted malware by doing simultaneous static and dynamic analyses, which is hard to achieve with static analysis alone and the Shannon entropy is used to detect obfuscated malware [53].

Researchers simplified both the static and dynamic analysis of crypto mining malicious files by using several methodologies from deep learning. They employed LSTM, Attention-based LSTM, and CNN to examine the opcodes of crypto mining malicious files, and as a result, they identified a high degree of accuracy with a low percentage of false – positive. The Cuckoo sandbox was used to run the malware in order to record system call event sequences for the dynamic analysis [54].

In a smart city environment, offer a two-stage hybrid malware detection approach for protecting IoT devices against obfuscated malware. After completing static analysis, the opcode is extracted, and benign files are recognized using the learnt knowledge using a Bi-LSTM. The files designated as benign are then subjected to a dynamic analysis in a layered virtual environment. Malware may be discovered using the EfficientNet-B3 model after extracting information on behavior and process memory from the behavior log [55].

To offer a new Android malware classification technique based on deep neural network (DNN), extracted static and dynamic information, then translated it into vector-based representations, after that, the dynamic information is transformed into graph-based forms, and graph kernels are used on the collections of graphs that have been created. hierarchical Multiple Kernel Learning, often known as MKL, is used as a hybrid classifier to combine a number of different vector and graph feature sets [56].

Table 2.3 summarizes the most significant results from prior studies that integrated hybrid analysis with deep learning approaches.

Table 2.3. A summary of hybrid malware detection methods.

Authors	**Model**	**Advantage**	**Accuracy**	**Dataset**
Huang et al. [49]	VGG-16	Detection of unknown malware effectiveness	94.70%	Virussign
Vinayakumar et al. [52]	CNN - LSTM	detecting and classifying zero-day malware	96.3%	Malimg Virussign Virusshare
Jeon et al. [53]	BiLSTM SPP-Net	Detect and classify IoT malware	92.5% 92.09%	Korea Internet & Security Agency (KISA)
Darabian et al. [54]	LSTM, ATT-LSTM, CNN	Detect crypto mining malware	95% 99%	Virustotal.com
Baek et al. [55]	BiLSTM, EfficientNet B3	Safeguard IoT devices against obfuscated malware in a smart city.	94.46% 94.98%	Korea Internet & Security Agency (KISA)
Xu et al. [56]	DNN	Detect malware for android.	94.7%	Google Play, VirusShare

PART 3

THEORETICAL BACKGROUND

The purpose of this section is to offer a description of the different deep learning approaches that were used in our research to identify and classify malware.

3.1. CONVOLUTIONAL NEURAL NETWORK

When it comes to deep learning methods for identifying and categorizing malicious software, Convolutional Neural Network (CNN) have become one of the most well-known and widely used techniques. As indicated in Figure 3.1, The basic components that make up a CNN are known as the convolution layers, the pooling layers, and the fully connected layers. While using kernels in the convolution layers, each kernel is convolved throughout the input's spatial dimensions to generate an activation map in two dimensions. The pooling layers will down sample the input dimension, thereby decreasing the number of parameters within that activation. The next step involves the fully connected layers making an effort to generate a class that can be applied to the data [57].

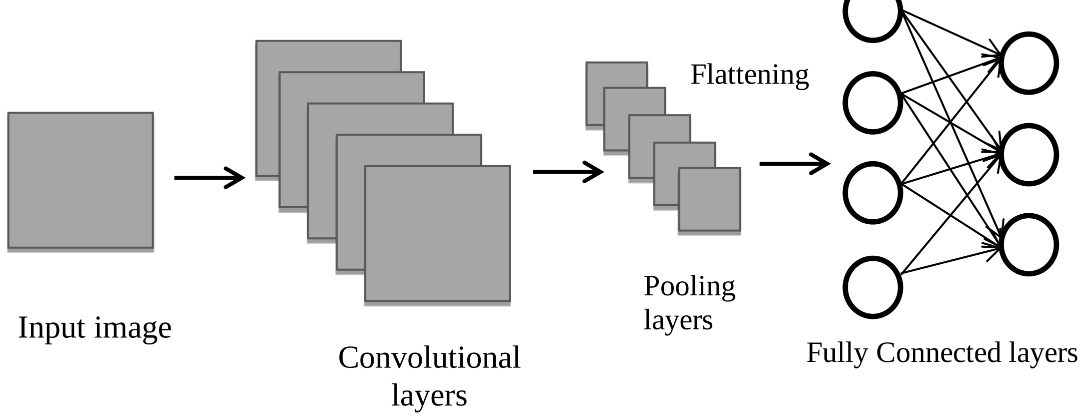


Figure 3.1. Basic CNN Architecture

3.1.1 Convolutional Layers

Convolutional layer is the major component that goes into the building of a CNN. It includes several filters or kernels, whose parameters must be learned as part of the training procedure. In most instances, the dimensions of the filter will be more diminutive than those of the image. To generate an activation map, each filter convolves the input image. In convolution, we iteratively move the kernel across the vertical and horizontal distance of the image, calculate the cross product between every kernel value, and identifying the input for every location [58]. Figure 3.2 offers a visual representation of the procedure that is known as convolution.

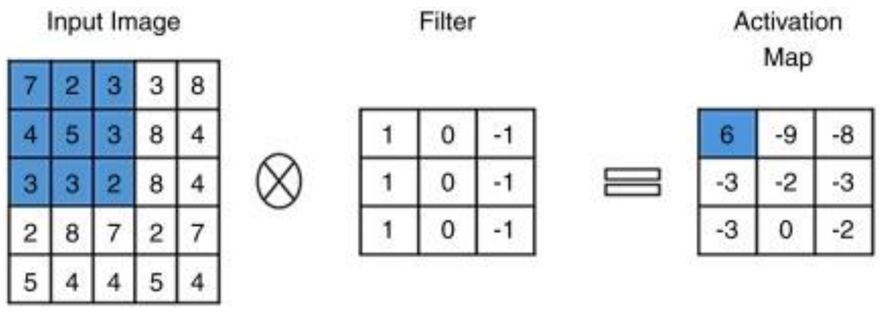


Figure 3.2. Convolutional layer process [58].

3.1.2. Pooling Layers

To do pooling, just move a 2-dim kerel over every channel for the feature map, and any features that lie within the kernel's coverage region will be added together to form the final feature set. Pooling layers is one way to cut down on the overall size of the feature maps. As a consequence of this, it reduces the amount of load that is imposed on the network as well as the quantity of parameters that need to be learned. The convolution layer's output, the feature map, is used as the basis for the pooling layer's creation of a summarization of the features found in a specific region of the map. Examples of pooling layers include Max pooling, which gets the greatest value in every kernel region part of the feature map. To do this, a max-pooling layer would create a feature map made up of the most important parts of the feature map that was given as input [59]. Example of the Max Pooling procedure is shown in Figure 3.3.

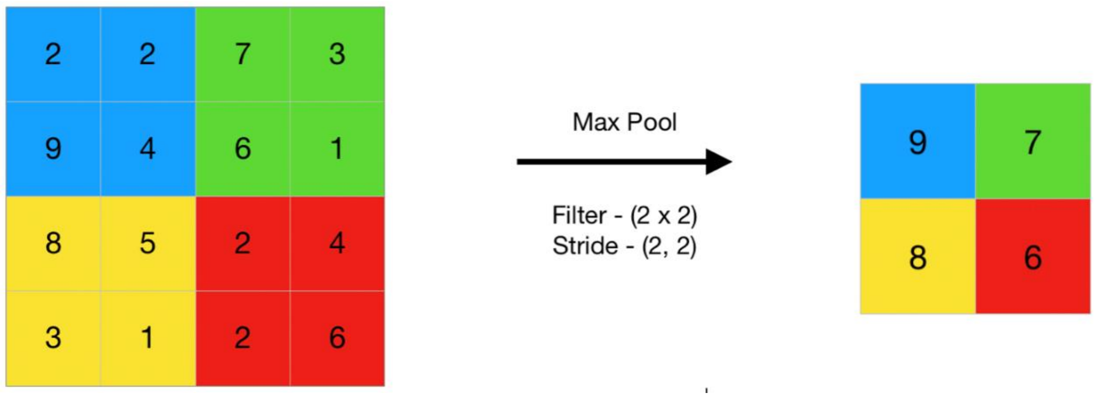


Figure 3.3. Max pooling operation [59].

3.1.3. Fully Connected Layers (FC)

Linking the final feature maps, which are produced by the last convolution layer or pooling layer, with a great number of layers that are entirely connected is a popular method. Dense layers are often used to describe these layers, in addition, it is common practice to guarantee that every input is related to every output by a weight that may be trained. Finally, a series of fully connected layers transforms the features discovered by the layers of convolution and pooling into the final outputs of the network, which include possibilities for categorization. [60]. Figure 3.4 shows layers that are fully connected to one another.

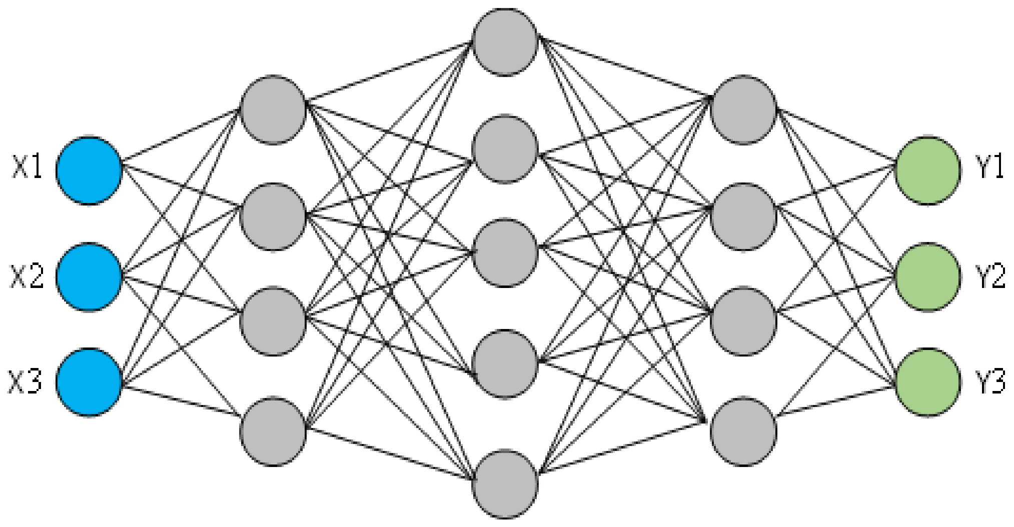


Figure 3.4. Fully connected layers.

3.1.4. Activation Functions

The activation function may behave as either a terminal or an intermediate node in neural networks. They play a role in deciding whether the neuron will fire. Rectified Linear Units are an example of a typical kind of activation function that may be found (ReLU). The fact that ReLU does not activate all neurons simultaneously is one of its major advantages over other activation mechanisms. ReLU function transforms all negative inputs to zero, preventing neuron activation. Few neurons are stimulated at a time, resulting in an extremely efficient use of computing resources [61].

3.1.5. Batch Size

The "batch size" hyperparameter specifies the number of samples that must be handled before the inner parameters of the model may be updated. Batches could be regarded as a for loop that predicts again and over again based on a collection of data from samples. By comparing the actual output variables with the projected ones, an error may be calculated after the batch has finished processing. These errors are taken into consideration by the update approach so that the model may be improved [62].

3.1.6. Epoch

For each iteration of the learning process, the number of times it will examine the whole training dataset is controlled by a hyperparameter called epochs. Every epoch has provided a chance for every training dataset to influence the inner parameters of the model. Numerous batches make up the epoch [62].

3.1.7. Loss Function

The loss function is the measure that establishes the degree to which the real output produced by the algorithm differs from the result that was expected by the approach. It is a technique for determining how well your algorithm mimics the data. It is possible to separate it into two distinct categories. One for classification (using discrete values such as 0, 1, 2, etc.), and the other for regression (continuous values) [63].

3.1.8. Dropout Learning

One of the strategies that is used in order to prevent memorization is known as Dropout. During the training phase of this approach, the activation of a number of neurons within the network is chosen at random and assigned the value zero. Each iteration of the training results in a different set of selected neurons. By using this strategy, the process of learning and the chance of overfitting are both slowed down [54,55].

3.2. CNN MODELS

This is the explanation and architecture of three common models of CNN implemented in this study.

3.2.1. VGG16

In 2014, Oxford University researchers Karen Simonyan and Andrew Zisserman [64] conceived of the VGG based on the architecture of a CNN. The model was entered into the 2014 Large Scale Visual Recognition Challenge (ILSVRC2014), where it got a score of 92.7 percent on the ImageNet dataset and a top-five test accuracy rating. As shown in Figure 3.5, there are sixteen weighted layers, thus the number "16" in VGG16. Even though VGG16 has a total of 21 layers (13-layer of convolution , 5-layer of max pooling , and 3-layer of dense), it only has sixteen weighted layers [65].

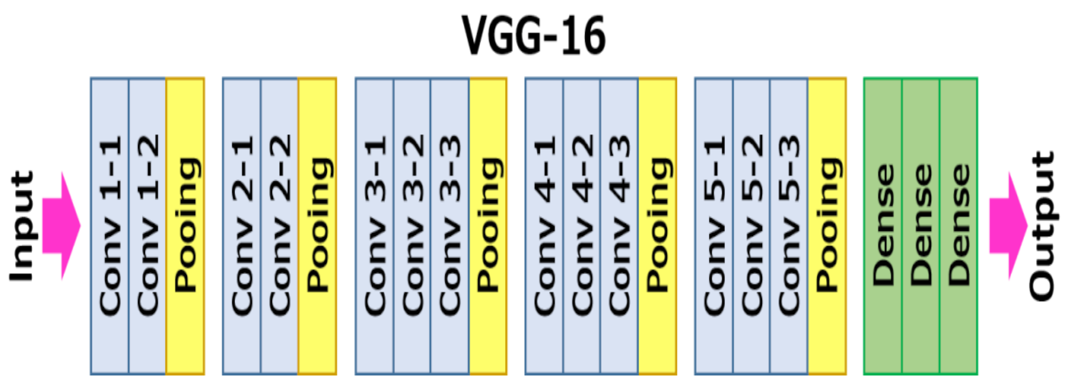


Figure 3.5. VGG16 layers [66].

The 1st and 2nd convolution layers each include 64 kernel filters that make them up, and the shape of each kernel is 3x3. When the image (3-channel) is processed through the 1st and 2nd convolution layers, its dimensions become 224,224,64. After that, a max pooling layer receives the outcome with two strides.

The 3rd and 4th convolution layers are consisting of 128 kernels, with the shape of 3x3. A layer of max pooling with two strides follows these two convolution layers, and after that the output is lowered to 56,56,128.

The 5th, 6th, and 7th layers are convolution with a filter shape of 3x3 using a 256-kernel. When they are complete, a max pooling layer that uses two strides will be added.

The 8th–13th layers are groups of convolution layers with a kernel shape of 3x3. These groups of layers of convolution use 512 kernels. A layer of max pooling with one stride follows these two convolution layers.

Both the 14th and 15th layers are hidden layers with 4096 units each, while the 16th layer is an output layer using SoftMax with 1000 units [67]. See Figure 3.6 for a schematic of the components that make up the VGG16 model.

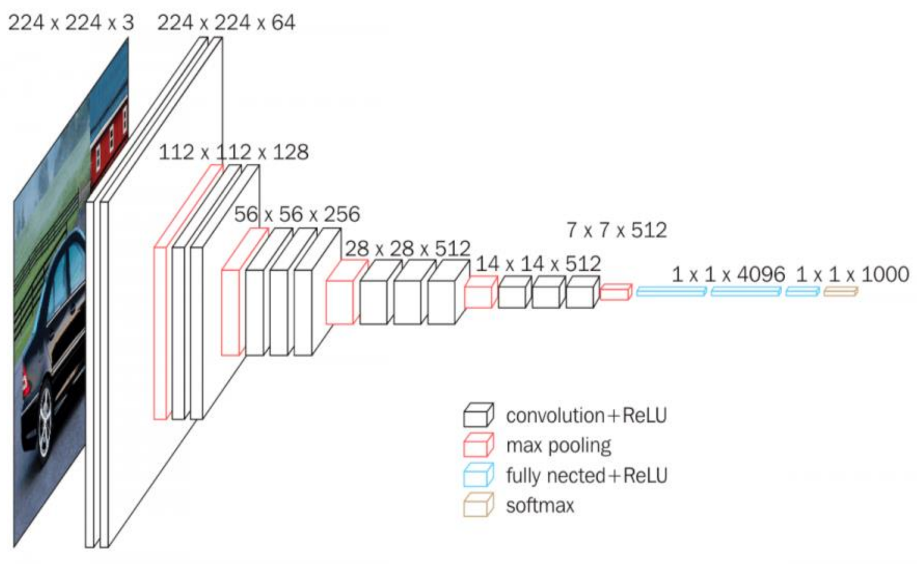


Figure 3.6. VGG16 architecture [66].

3.2.2. ResNet50

Residual Network was the winner of the ILSVRC [68] in 2015, which is a yearly competition in which software is judged on its ability to accurately identify and recognize objects. Kaiming He [69] is the inventor of ResNet to build ultra-deep networks that did not suffer from the vanishing gradient issue that plagued previous generations by creating shortcut pathways between layers. The identity connection between the layers is the sole addition that must be made to the basic network to transform it into a residual network. The remnant block that was utilized in the network is shown in the Figure 3.7. You can recognize the link to the identity by the curving arrow that starts at the input and travels all the way to the bottom of the residual block.

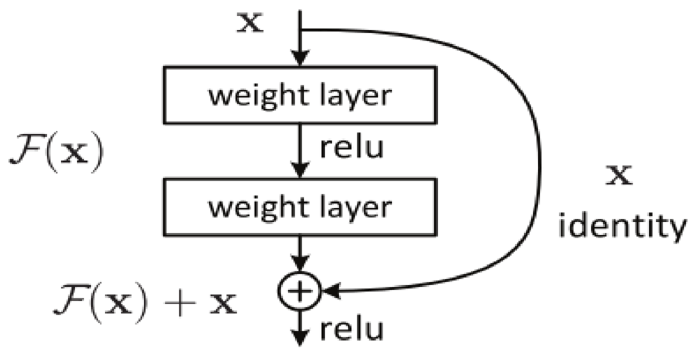

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.7. Skip connection [69].

The size of the image in this architecture that is sent in is 224x244 with 3 channels and started with the kernel of the convolution layer is 7x7x64 with a stride of 2 and following that will be max pooling using 2-step strides. After that, we have 4 stages of convolutional layers, as illustrated in Figure 3.8. The identity link is shown by the curving arrows in the diagram. The convolution process that takes place in the residual block makes use of two strides, as can be seen from dotted arrows.

Stage 1's convolution layers consist of a 1x1x64 kernel, a 3x3x64 kernel, and a 1x1x256 kernel. With three repetitions of these three layers, we now have nine total layers. The second stage of convolution layers consists of the following kernel sizes: (1x1x128), (3x3x128), and (1x1x512). This process of layering was performed four times, for a total of 12 layers. The third stage is made up of a 1x1x256 kernel and two more kernels that are 3x3x256 and 1x1x1024. For the fourth stage, we used a kernel of 1x1x512, followed by 3 x3x512, and 1x1x2048, for a grand total of 9 layers. Following that, a global average pooling is performed, and at the end of the model, we are left with a fully connected layer that has a total of one thousand neurons [70].

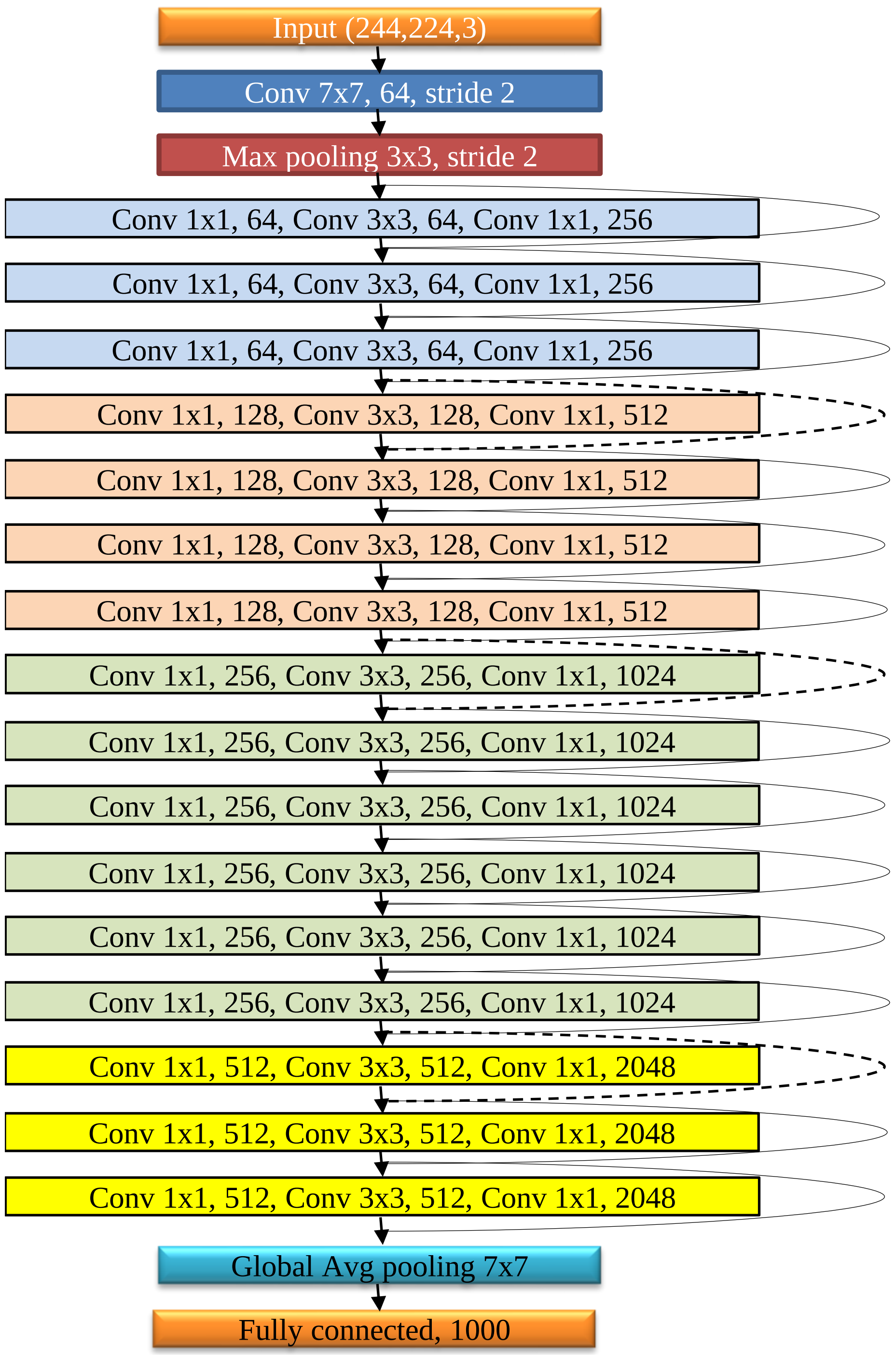

Input (244,224,3)
Conv 7x7, 64, stride 2
Max pooling 3x3, stride 2
Conv 1x1, 64, Conv 3x3, 64, Conv 1x1, 256
Conv 1x1, 64, Conv 3x3, 64, Conv 1x1, 256
Conv 1x1, 64, Conv 3x3, 64, Conv 1x1, 256
Conv 1x1, 128, Conv 3x3, 128, Conv 1x1, 512
Conv 1x1, 128, Conv 3x3, 128, Conv 1x1, 512
Conv 1x1, 128, Conv 3x3, 128, Conv 1x1, 512
Conv 1x1, 128, Conv 3x3, 128, Conv 1x1, 512
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 256, Conv 3x3, 256, Conv 1x1, 1024
Conv 1x1, 512, Conv 3x3, 512, Conv 1x1, 2048
Conv 1x1, 512, Conv 3x3, 512, Conv 1x1, 2048
Conv 1x1, 512, Conv 3x3, 512, Conv 1x1, 2048
Global Avg pooling 7x7
Fully connected, 1000

Figure 3.8. Resnet50 architecture.

3.2.3. Inception V3

The design of this model (3rd version) was released in the year 2015 [71].This version of the model consists of 42 layers and has a lower error rate than the previous two versions. In this type of version of Inception 3, there were a lot of important changes to increase the performance of the model, and these changes can be seen in the switching from larger to smaller convolutions, an asymmetrical convolution, an auxiliary classifier, and efficient scaling down of grid sizes. The Inception V1 model's considerable decrease in dimensions was one of its greatest assets and one of its most beneficial aspects. In Inception V3, the model's performance was increased by partitioning the larger convolutions into a collection of smaller convolutions. The factorization into smaller convolutions is shown in Figures 3.9, 3.10.

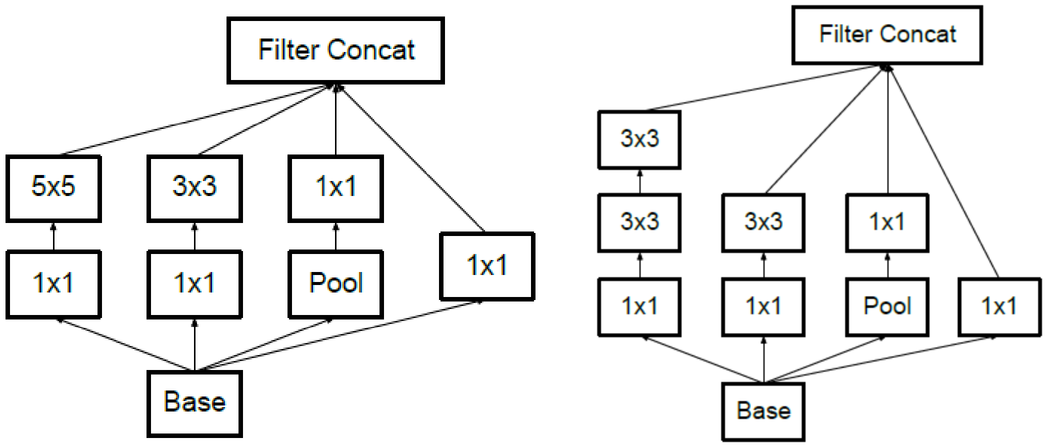


Figure 3.9. 5×5 conv layer(left) was replaced by two 3×3 conv layers(right) [71].

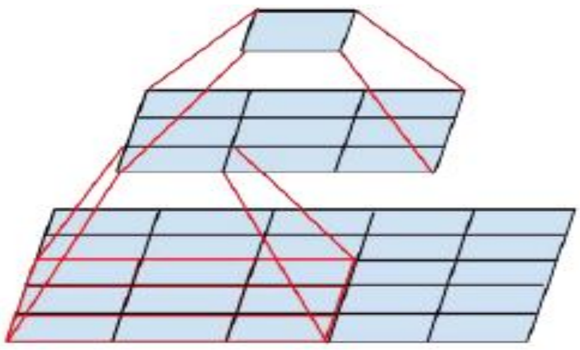


Figure 3.10. One 5x5 conv layer replaced two 3x3 conv layers [71].

The decrease in the total number of parameters leads to a drop in the total amount of computing effort required. The reduction of bigger convolutions by replacing them with smaller convolutions led to a gain of 28 percent as a consequence of this factorization. They were able to do this by performing a 1x3 convolutional operation first, followed by a 3x1 convolution in place of the conventional 3x3 convolutions. When considering the same shape of output and input filters, the two-layer technique is 33 percent more cost-effective. Figure 3.11 shows the factorization in the Inception module.

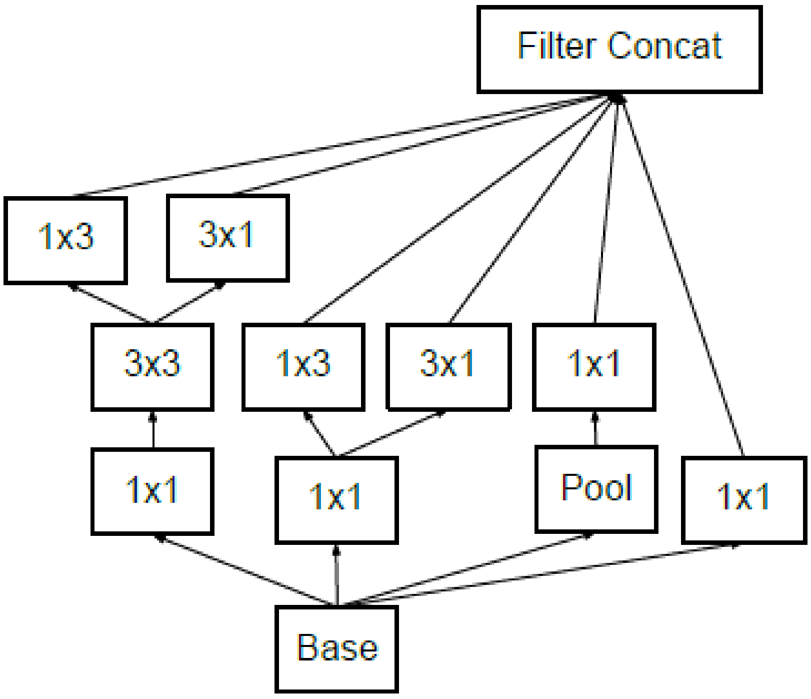


Figure 3.11. Asymmetric factorization in the Inception module [71].

Auxiliary classifiers are often utilized because they make it easier for very deep neural networks to converge. When dealing with very deep networks, the primary function of the auxiliary classifier is to overcome the issue of vanishing gradients. Early on in the training, the auxiliary classifiers did not contribute to any progress. At the conclusion, however, the network with auxiliary classifiers demonstrated more accuracy than the network without auxiliary classifiers. Thus, the auxiliary classifiers in the Inception V3 model architecture work as a regularizer. The auxiliary classifiers are explained in Figure 3.12.

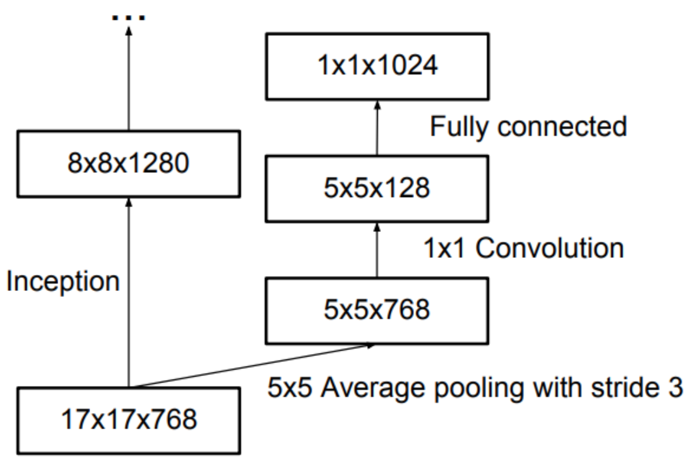


Figure 3.12. The Auxiliary classifier works as a regularization [71].

Typically, max pooling and average pooling were employed to lower the feature map grid size. As part of the Inception V3 model, the activation dimensionality for the network kernels has been expanded to help efficiently reduce the grid size. And this is accomplished by concatenating two parallel blocks of convolution and pooling [72], as shown in Figure 3.13.

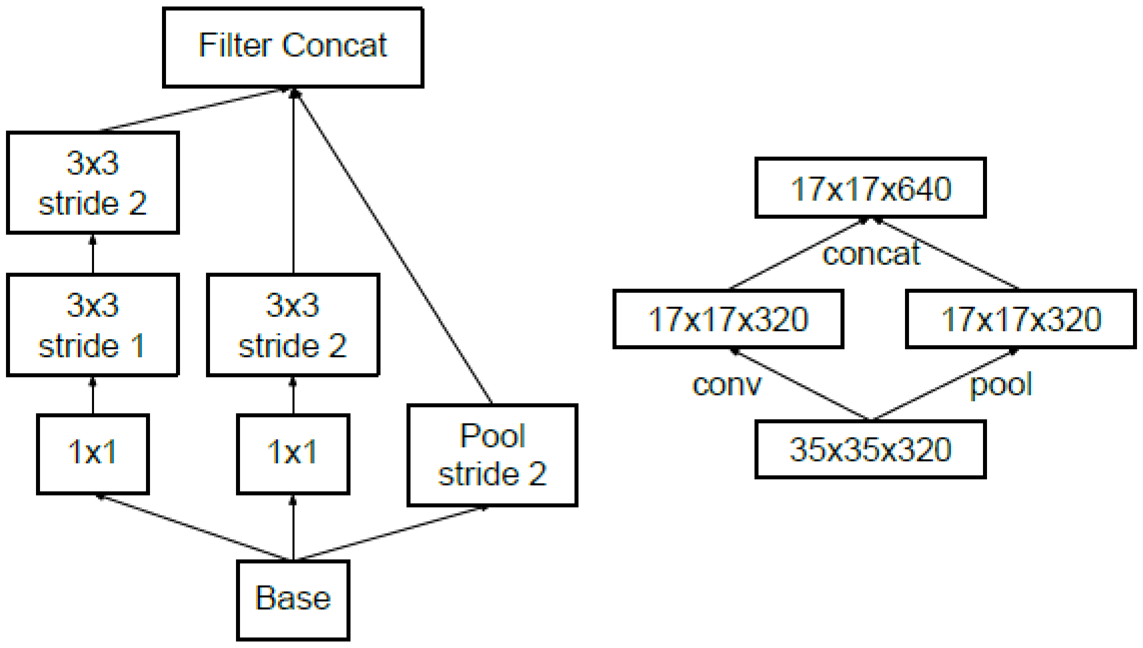


Figure 3.13. A Detail structure for efficient grid size reduction [71].

Following all optimizations, the final Inception V3 structure appears in Figure 3.14.

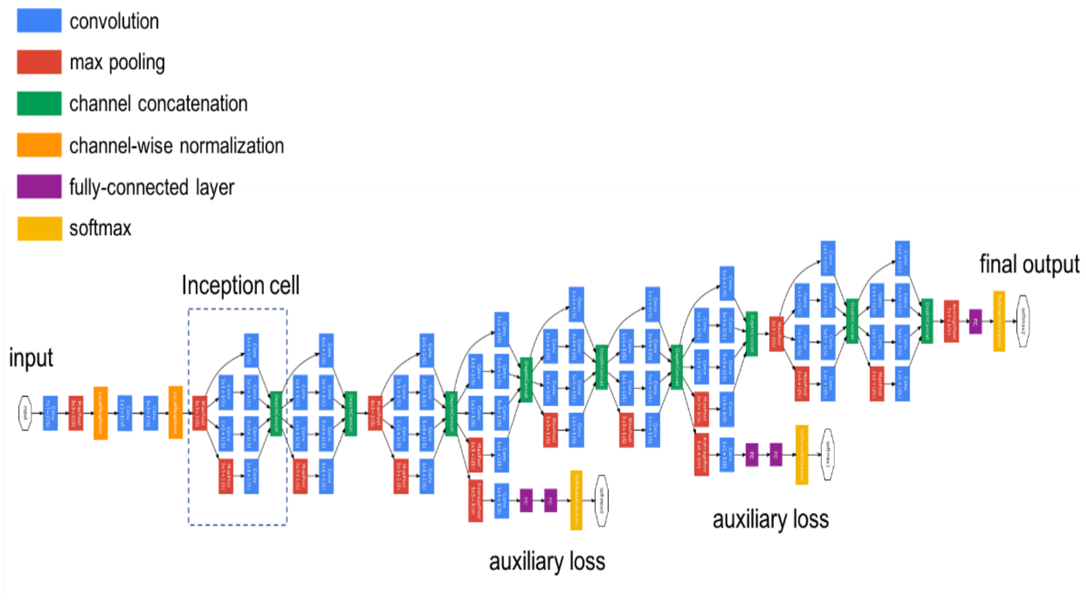


Figure 3.14. Google Inception v3 architecture [73].

3.3. RECURRENT NEURAL NETWORKS (RNN)

A recurrent neural network, often known as an RNN, is an artificial neural network that can analyze sequential input, detecting patterns, and predicting the final outcome. This neural network is referred to as "recurrent" because it can repeatedly execute the same task or operation on a set of inputs. An RNN has an internal memory that allows it to memorize information from the input it receives, which assists in context acquisition for the system. As a result, a recurrent neural network is an excellent choice for the handling of sequential data, for example, a time series.

The short-term memory of RNN is a concern because it will have difficulty sending information from earlier time phases to later time phases if the sequence is lengthy. When attempting to make a prediction, RNN may leave out essential information in the beginning. LSTM and GRU are two technologies that were created as a means of improving short-term memory, which include gateways that control the flow of data.

3.3.1. Long Short-Term Memory (LSTM)

Hochreiter and Schmidhuber were the ones that came up with the idea for LSTM first [74]. In the field of deep learning, one sophisticated version of RNN architecture known as long short-term memory (LSTM) is used. LSTM, in contrast to the more prevalent feed-forward in common neural networks, may also process information through its feedback connections. In addition to being able to process data streams such as audio, it can also handle visual information. LSTM is employed for a variety of tasks including handwriting recognition [75], voice recognition [76] and anomaly detection. An input gate, an output gate, and a forget gate make up each cell in a typical LSTM unit, as shown in Figure 3.15. It is up to the forget gate to decide which data has to be carefully considered and which may be safely discarded. With the aid of the input gate, it is possible to gauge the importance of the fresh data given by the input. The value of the following hidden state is set by the output gate [77]. The flow of data inside and outside of the cell is controlled by the cell's three gates, and the cell is able to retain information for extended periods of time. LSTM is ideal to categorizing and making expectations relied on time series information since major events in a time series may have unforeseen delays. LSTM was developed to address the issue of vanishing gradients that may arise when using normal RNN for training.

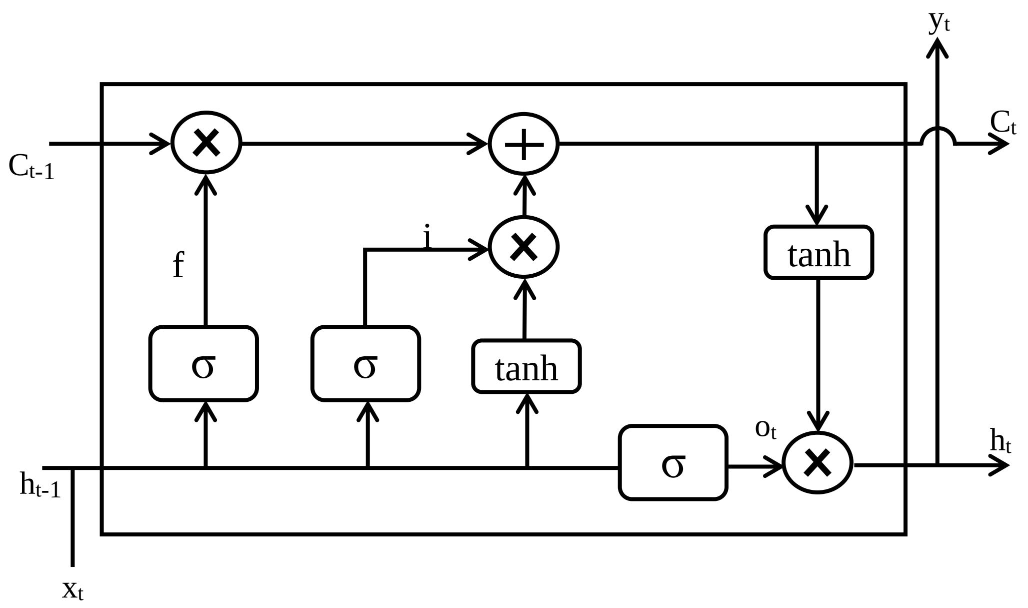


Figure 3.15. Structure of LSTM

3.2.2. Gated Recurring Units (GRU)

The GRU is a more recent form of recurrent neural network that mimics an LSTM in its work, introduced in 2014 by Kyunghyun Cho et al. [78]. It is composed of two gates, one of which serves as a reset gate, whereas the other is an update gate. Each gate performs a different function. The GRU's update gate performs the same function as the forget gate as well as the input gate in the LSTM model. It makes decisions on what data should be thrown away and what data should be kept, whereas it was decided how much of the previous data should be erased using the reset gate mechanism. The GRU will not take into consideration the cell state that made use of the concealed state to transmit data. The structure of the GRU is shown in Figure 3.16.

GRU has short parameters in training since it has two gates (reset and update), indicating that it uses less memory, executes quicker, and learns faster than LSTM, even though LSTM is more accurate on datasets with longer sequences.

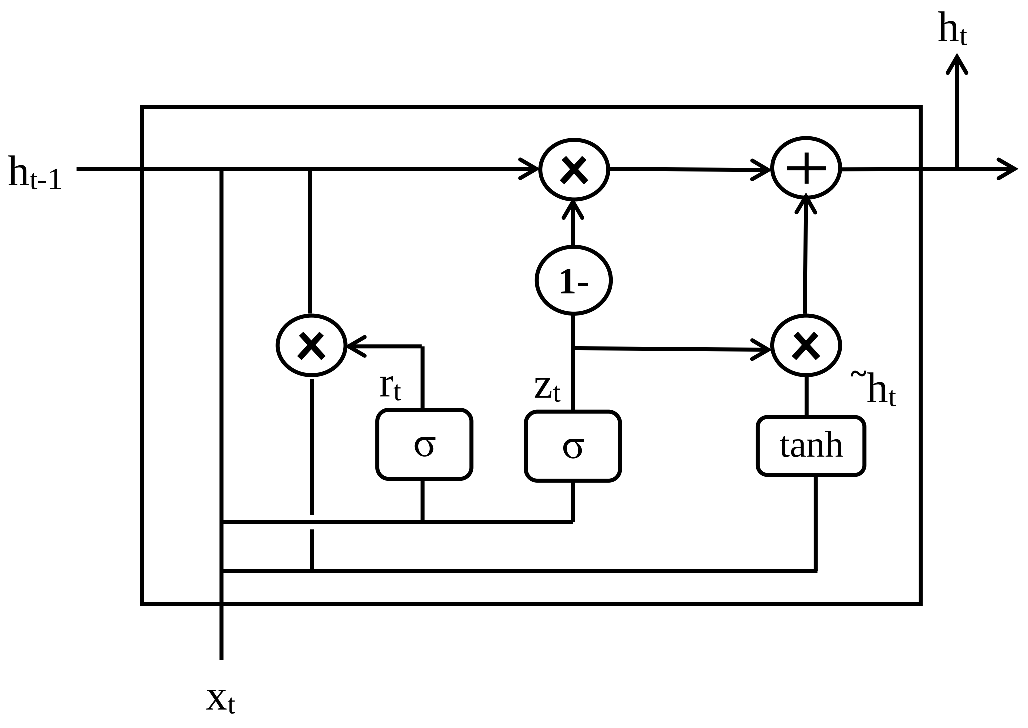


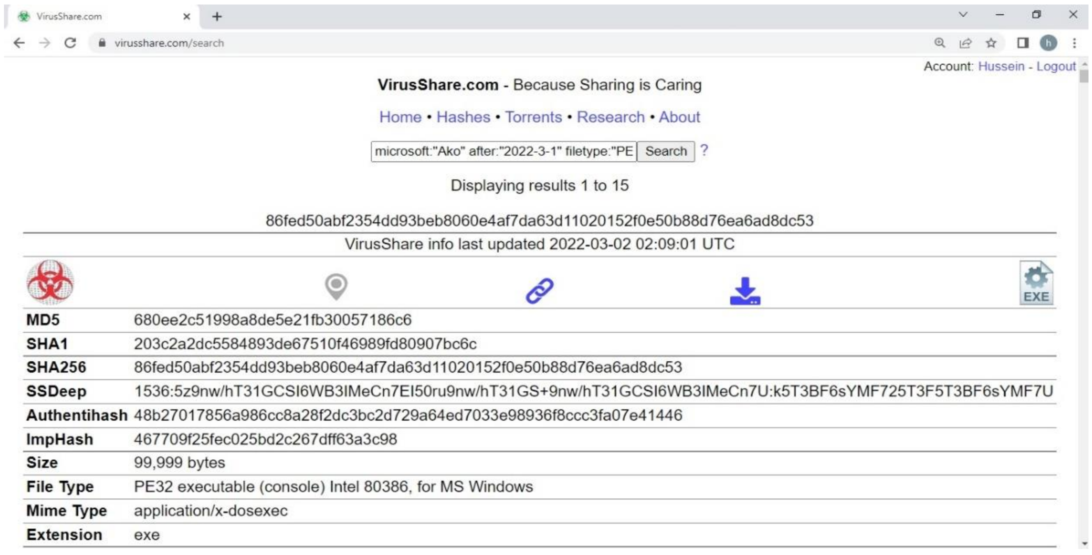
Figure 3.16. Structure of Gated Recurring Units (GRU)

PART 4

METHODOLOGY

4.1. DATA COLLECTION

The virusshare website [12], which is a store of malware samples for researchers working in the field of information security, was used to gather 7513 malicious Portable Executable (PE) files from 29 families for this investigation. We gather malware families by generating queries based on the Microsoft Malware Protection for labeling malware. As can be seen in Figure 4.1, which depicts the query that was used during the search for AKO family names that belong to the ransomware, which is a type of malware after 1-3-2022 in date. While 1000 of the EXE benign files were collected from the site [79], so that the classifications became 30 categories as shown in Table 4.1. We gathered data over the course of a month, and the primary purpose of this data gathering was to build two datasets, one by converting samples to images and the other by extracting API call sequences for each sample, to finally categorize malware families and benign applications using deep learning techniques.


VirusShare.com
virusshare.com/search
Account: Hussein - Logout
Home • Hashes • Torrents • Research • About
Search
?
EXE

VirusShare.com - Because Sharing is Caring

microsoft:"Ako" after:"2022-3-1" filetype:"PE

Displaying results 1 to 15

86fed50abf2354dd93beb8060e4af7da63d11020152f0e50b88d76ea6ad8dc53

VirusShare info last updated 2022-03-02 02:09:01 UTC

MD5	680ee2c51998a8de5e21fb30057186c6
SHA1	203c2a2dc5584893de67510f46989fd80907bc6c
SHA256	86fed50abf2354dd93beb8060e4af7da63d11020152f0e50b88d76ea6ad8dc53
SSDeep	1536:5z9nw/hT31GCSI6WB3IMeCn7EI50ru9nw/hT31GS+9nw/hT31GCSI6WB3IMeCn7U:k5T3BF6sYMF725T3F5T3BF6sYMF7U
Authentihash	48b27017856a986cc8a28f2dc3bc2d729a64ed7033e98936f8ccc3fa07e41446
ImpHash	467709f25fec025bd2c267dff63a3c98
Size	99,999 bytes
File Type	PE32 executable (console) Intel 80386, for MS Windows
Mime Type	application/x-dosexec
Extension	exe

Figure 4.1. A Screenshot of the website for VirusShare.com.

Table 4.1. Malware families and types.

No.	Family	Type	samples
1	Benign	Benign	1000
2	Ako	Ransomware	260
3	Autorun.NE	Virus	249
4	Banker.LY	TrojanSpy	260
5	Delf.DU	Backdoor	260
6	Drolnux.B	Worm	259
7	Eggnog.A	Worm	300
8	GandCrab.AE	Ransomware	220
9	Ganelp.E	Worm	260
10	Linkury.RS!MTB	Adware	244
11	Neconyd.A	Trojan	259
12	Nemucod	TrojanDownloader	260
13	Neojit.A	TrojanDownloader	300
14	OpenInstaller	PUA	260
15	Playtech	PUA	260
16	QQPass.GP	PWS	260
17	Qukart	TrojanSpy	260
18	Resur.A!epo	Virus	258
19	Shodi.A	Virus	220
20	Simda.D	PWS	159
21	Sivis.A	Virus	260
22	Small.M	TrojanSpy	260
23	Soltern!rfn	Worm	260
24	Trickbot.GML!MTB	Trojan	300
25	Unruy.F	TrojanDownloader	260
26	Upatre.A	TrojanDownloader	300
27	Urelas.AA	Trojan	260
28	Wabot.A	Backdoor	260
29	Yoof.E	Worm	289
30	Zombie!rfn	Trojan	256

During the course of this research, 8,513 malicious and benign samples were gathered and distributed among 30 different classes of samples. The statistical chart in Figure 4.2 illustrates the distribution of these samples.

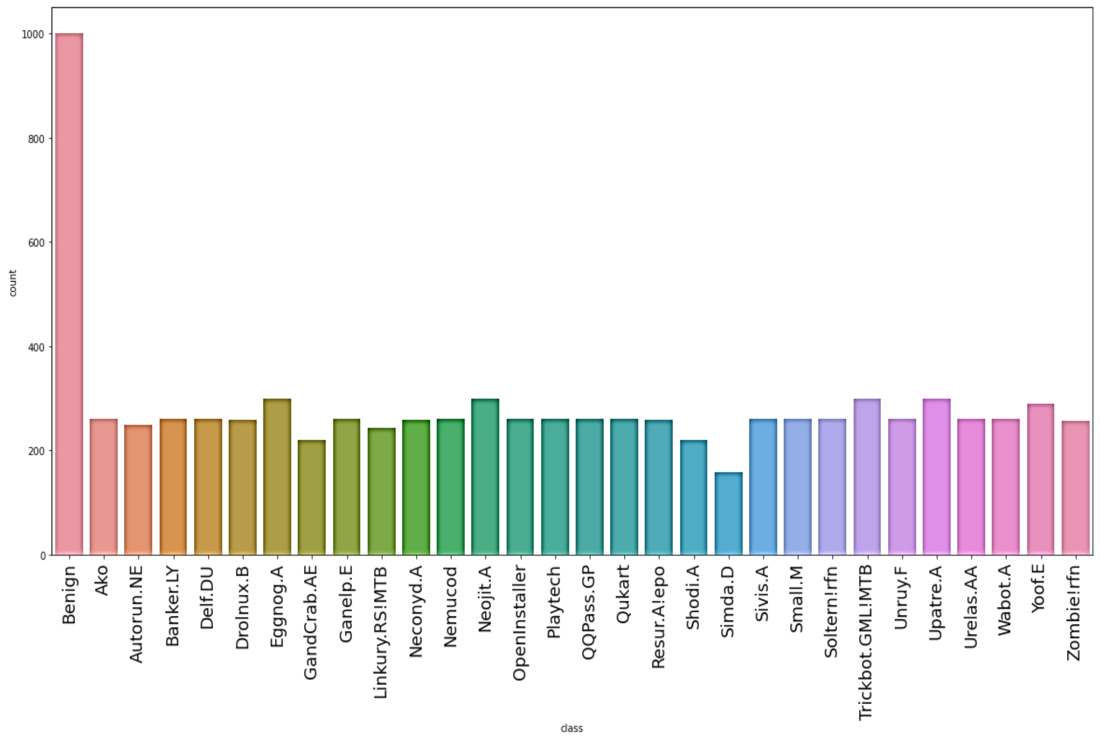


Figure 4.2. The distribution of malware samples collected.

4.1.1. Malware Images Dataset (First Dataset)

The method that was described by Nataraj et al. [8] was used to convert each sample that was collected (malware and benign) to grayscale images to create the first dataset. Figure 4.3, which depicts the process of translating data from one format into another, shows that the binary sample was first transformed into an 8-bit vector. This step of the process was necessary before moving on to the next step of the process. The 8-bit vector was then converted into an image when that was completed.

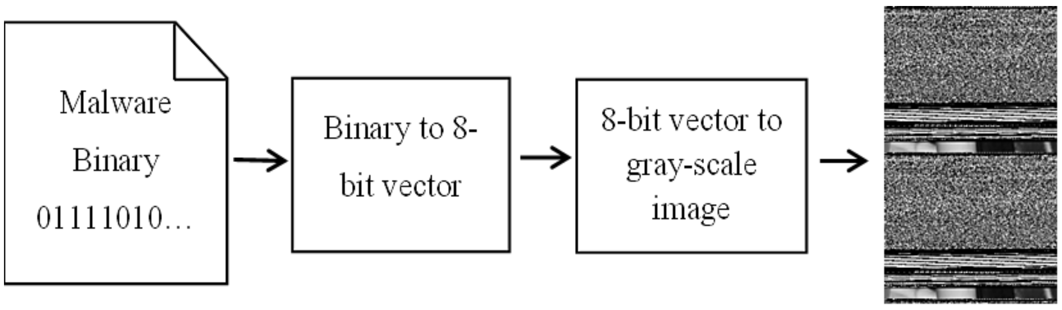
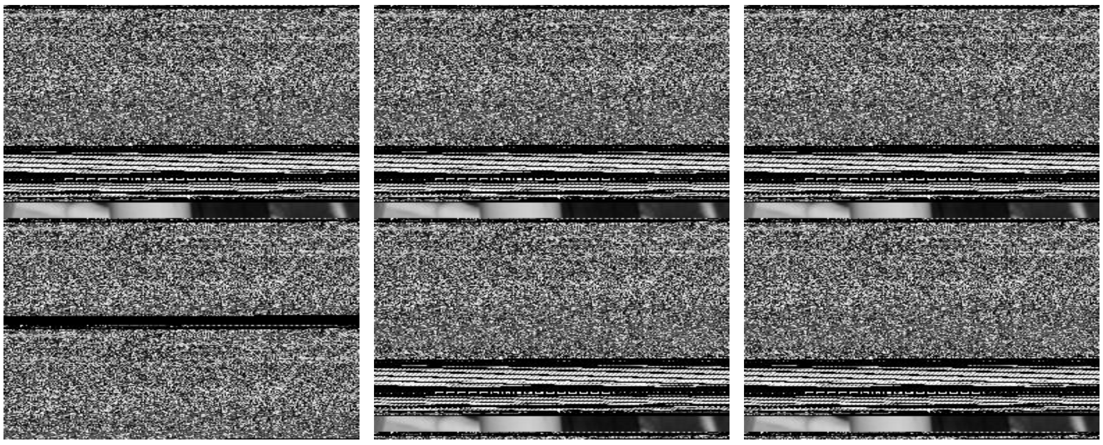
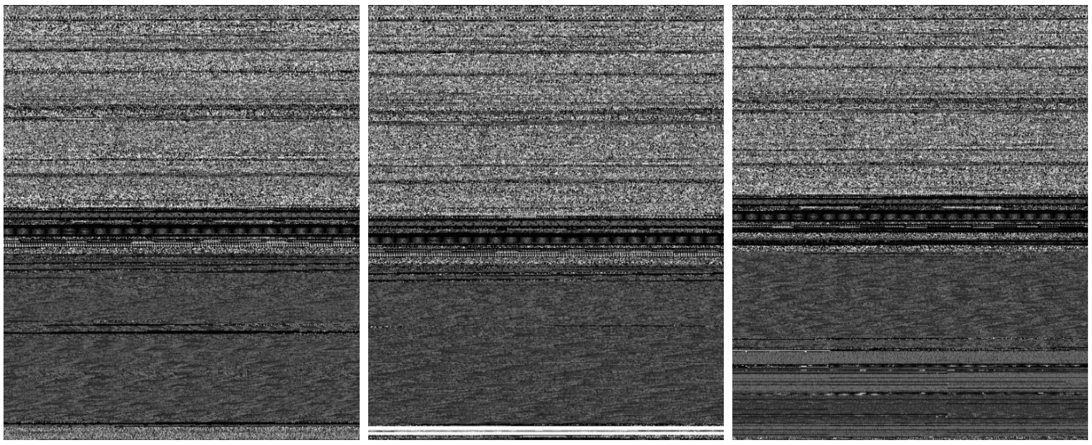
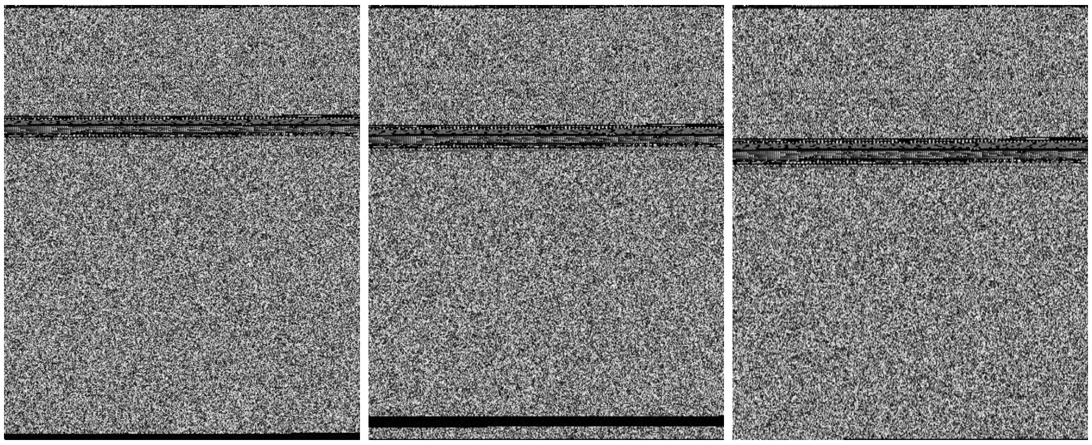


Figure 4.3. Method for converting malware to an image

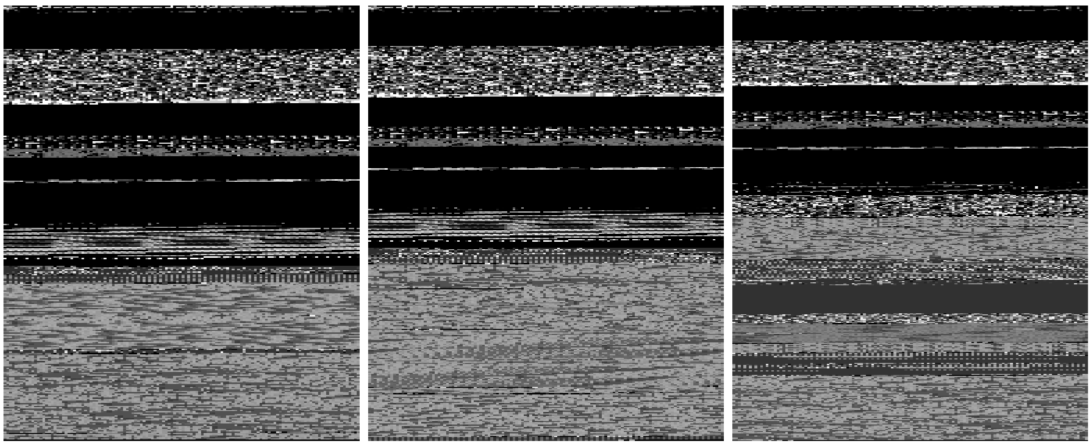
However, relying on the image's size, the height of the image may vary, although the width of the image will still stay constant, as shown in Table 4.2. Since every pixel in the image is made up of 8 bits, the final image is made up of integers ranging from 0 to 255. This is since 0 symbolizes black and 255 represents white, with gray scale gradients being in between (0-255). The ability to distinguish between the various components of a binary is the primary advantage of seeing a malicious executable in the form of an image. Figure 4.4 displays several examples of malware images, and we observe the similar appearance that exists between the visuals of different models that come from the same family.


(a) Ako

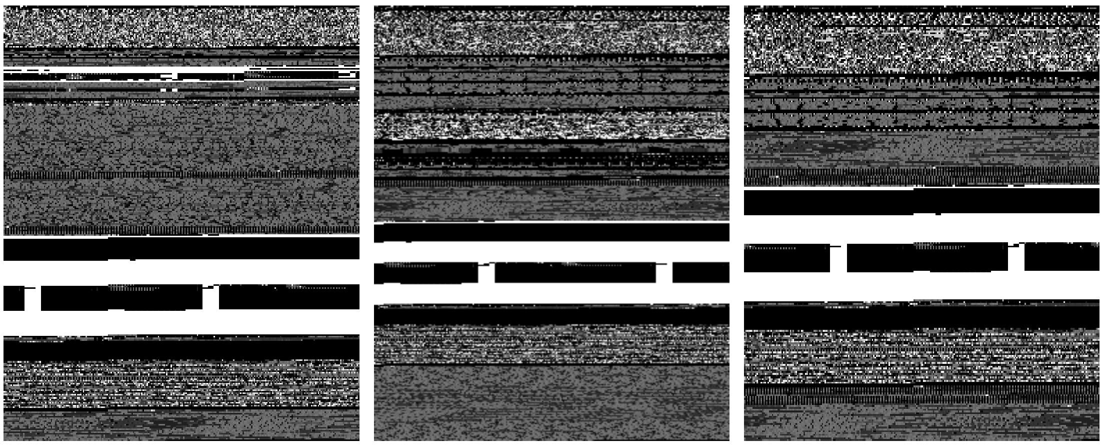

(b) Autorun.NE



(c) Delf.DU



(d) Zombie!rfn



(e) Sivis.A

Figure 4.4. Malware image samples belong to various families.

Table 4.2. Width of an image for different file sizes.

File Size Range	Image Width
<10 kB	32
10 kB – 30 kB	64
30 kB – 60 kB	128
60 kB – 100 kB	256
100 kB – 200 kB	384
200 kB – 500 kB	512
500 kB – 1000 kB	768
>1000 kB	1024

4.1.2. API Call Sequences Dataset (Second Dataset)

Using the Pefile module, A Python library called Pefile makes it easier to read and deal with portable executable files, the first 50 API call sequences were taken ("none" API call was ignored) from each sample to create the second dataset. Using 50 API call sequences for each malware to increase the reading speed for API call sequences to detect malicious software in the shortest amount of time. If a sample's API call sequence is less than 50, we added 0 to complete the 50 sequences. Examples of API call sequences extracted from malware samples are shown in Table 4.3.

Table 4.3. Examples of API call sequences taken from malware samples.

API call sequences	class
`LoadLibraryA, GetProcAddress, ExitProcess,RegOpenKeyA, ShellExecuteA, ShowWindow, InternetOpenA, gethostbyname`	Ako (Ransomware)
`LoadLibraryA, GetProcAddress, VirtualProtect, VirtualAlloc, VirtualFree, ExitProcess, InitCommonControls, memset, CoInitialize, ShellExecuteExA, MessageBoxA`	Sivis.A (Virus)
`LoadLibraryA, GetProcAddress, ExitProcess, RegOpenKeyA, SysFreeString, CharNextA`	Yoof.E (Worm)

Then we made indexes for all the unique API calls for all samples, and each one was given a unique id number. The names of API calls were then replaced by the unique numbers of each sample, equal to its index, and saved to a CSV file at the end. This CSV file dataset has a sha256 hash for all files, a class indicating whether the sample contained one of the malware family or benign, a class number containing numbers (0–29) for all classes, and 50 API calls. So, we have two datasets that belong to the same sample in different ways [80]. Table 4.4 displays the API index numbers, which range from 1 to 484 in this data collection.

Table 4.4. API's index numbers.

GetDiskFreeSpaceW=1
UpdateResourceA=2
SetLastError=3
FreeEnvironmentStringsA=4
DestroyCaret=5
GetActiveWindow=6
_ismbblead=7
MessageBoxW=8
FindNextFileW=9
_iob=10
GetSystemWindowsDirectoryW=11
ScaleViewportExtEx=12
CloseHandle=13
RegSetValueExW=14
GetSystemMetrics=15
WaitForSingleObject=16
MessageBoxA=17
SizeofResource=18
VirtualProtect=19
GetCurrentProcessId=20
:
GetWindowTextW=484

4.2. PROPOSED METHOD

In this research, we utilize a methodology that is relied on selecting the three models that are the most effective for real-time detection and classification based on accuracy. Additionally, the computational efficiency of a model is measured by how few parameters it generates and how little it costs to run simulations (memory and also other resources).

4.2.1. CNN Models with Malware Images Dataset

In the first dataset, which is made up of images of both malicious and good software, we will be doing experiments on the custom CNN model we created and three of the common pretrained network models of CNN, which are (VGG16, Inception V3, and Resnet50) to do training and testing and make comparisons between them.

In our model, Initially, images were reformatted, including re-sizing the images to 150 x 150 pixels, and normalization was done. The dataset was split so that 80% was used for training, 10% was used for testing, and 10% for validation. The data was also shuffled to improve accuracy.

The suggested model has three layers of convolution—64, 3x3, 128, 3x3, and 256, 3x3—with a Relu activation function to execute non-linear transformations. Additionally, the model contains six layers of Max pooling, as can be seen in Figure 4.5. (2x2 with stride 2). After that comes a layer that is fully connected and has 256 neurons, and then after that comes an output layer that has 30 different classes. In order to prevent overfitting, the dropout was placed after each Max pooling layer as well as after a layer that was fully connected. Adam was used as the optimizer for this model, while Sparse Category Crossentropy was utilized as the loss function [80]. Table 4.5 contains a summary that may be seen in its entirety for the CNN model.

In this study, there will also be tests on VGG16, Inception V3, and Resnet50 with the first dataset (malware images), which are three of the most popular CNN-pretrained network models. Before going through these models, the images are resized to 224 by

224 pixels so that they can be processed properly because these models were designed and worked with this size. Because our dataset only includes grayscale images for the samples in the first dataset, and because these models were meant to work with color images (three channels), we had to add a convolutional layer with a 3x3 kernel and an input size of 224x224x1 to make these models work with grayscale images correctly.

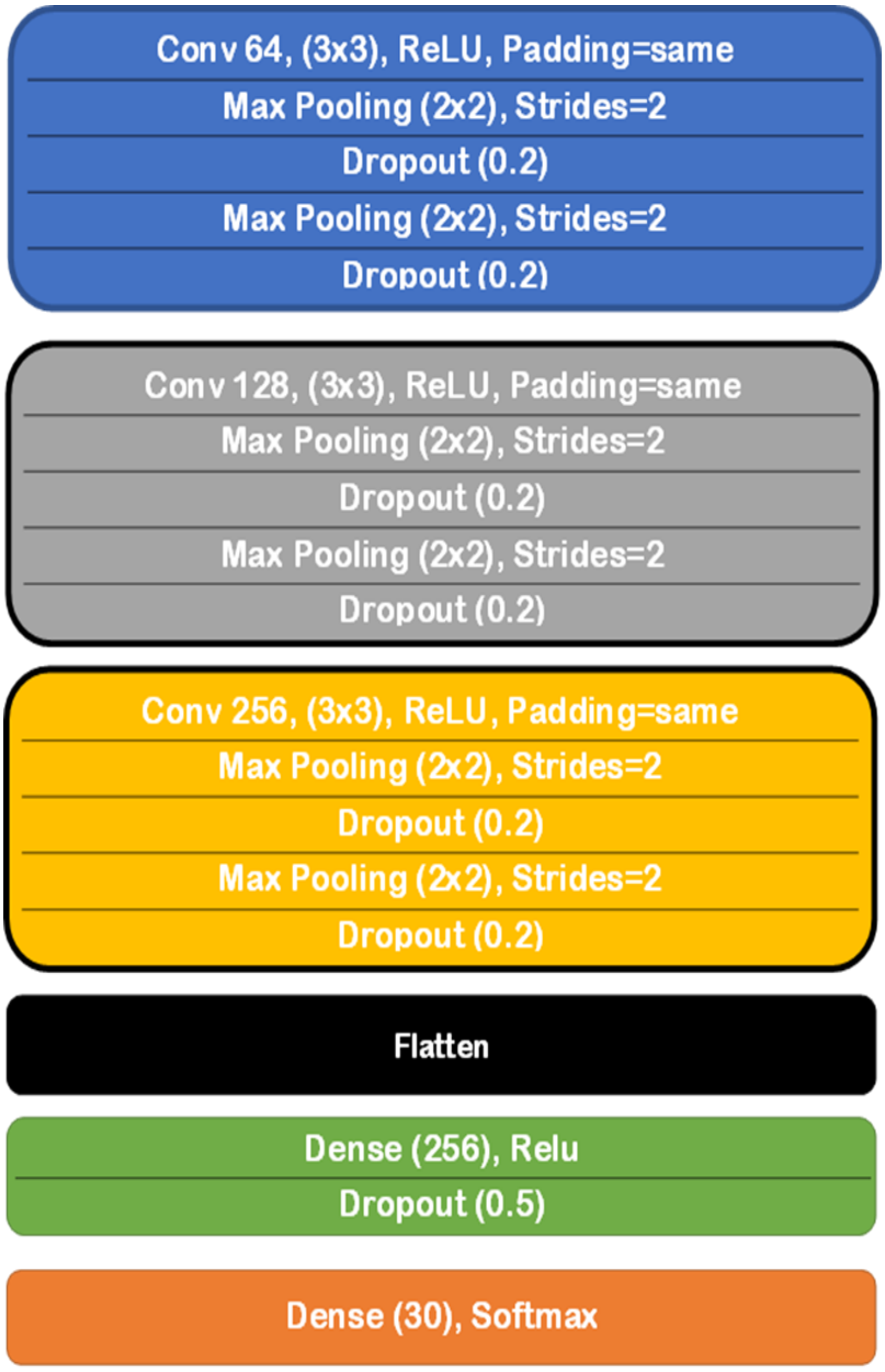

Conv 64, (3x3), ReLU, Padding=same
Max Pooling (2x2), Strides=2
Dropout (0.2)
Max Pooling (2x2), Strides=2
Dropout (0.2)
Conv 128, (3x3), ReLU, Padding=same
Max Pooling (2x2), Strides=2
Dropout (0.2)
Max Pooling (2x2), Strides=2
Dropout (0.2)
Conv 256, (3x3), ReLU, Padding=same
Max Pooling (2x2), Strides=2
Dropout (0.2)
Max Pooling (2x2), Strides=2
Dropout (0.2)
Flatten
Dense (256), Relu
Dropout (0.5)
Dense (30), Softmax

Figure 4.5. Our CNN architecture with the first dataset.

Table 4.5. Summary of the CNN model with the first dataset.

Layer	**Output Shape**	**Parameters**
Conv2D	(None, 150, 150, 64)	640
MaxPooling2D	(None, 75, 75, 64)	0
Dropout	(None, 75, 75, 64)	0
MaxPooling2D	(None, 37, 37, 64)	0
Dropout	(None, 37, 37, 64)	0
Conv2D	(None, 37, 37, 128)	73856
MaxPooling2D	(None, 18, 18, 128)	0
Dropout	(None, 18, 18, 128)	0
MaxPooling2D	(None, 9, 9, 128)	0
Dropout	(None, 9, 9, 128)	0
Conv2D	(None, 9, 9, 256)	295168
MaxPooling2D	(None, 4, 4, 256)	0
Dropout	(None, 4, 4, 256)	0
MaxPooling2D	(None, 2, 2, 256)	0
Dropout	(None, 2, 2, 256)	0
Flatten	(None, 1024)	0
Dense	(None, 256)	262400
Dropout	(None, 256)	0
Dense	(None, 30)	7710
Total params: 639,774		
Trainable params: 639,774		
Non-trainable params: 0		

4.2.2. RNN Models with API Call Sequences Dataset

In the second dataset, which consists of API call sequences, we use two RNN algorithms: LSTM and GRU. In the LSTM model, we divide the data into 70% training data, 15% testing data, and 15% validation data. The model consists of an embedding layer, a 64-cell long short-term memory (LSTM) layer with a linear activation function, and a dense layer using a SoftMax classifier for multiple classes. This model employed SpatialDropout1D to prevent dropout [80]. Summarized results from the LSTM model applied to the second dataset are shown in Table 4.6.

For the GRU model, we allocate the same portion of the dataset as in the LSTM model for training, testing, and validation. We used an embedding layer, a GRU layer with 64 cells and a linear activation function, and a dense layer with a SoftMax classifier for multiple classes and without SpatialDropout1D, which is used in the LSTM model. In Table 4.7, we can see a summary of the GRU model applied to the second dataset.

Table 4.6. Summary of the LSTM model with the second dataset.

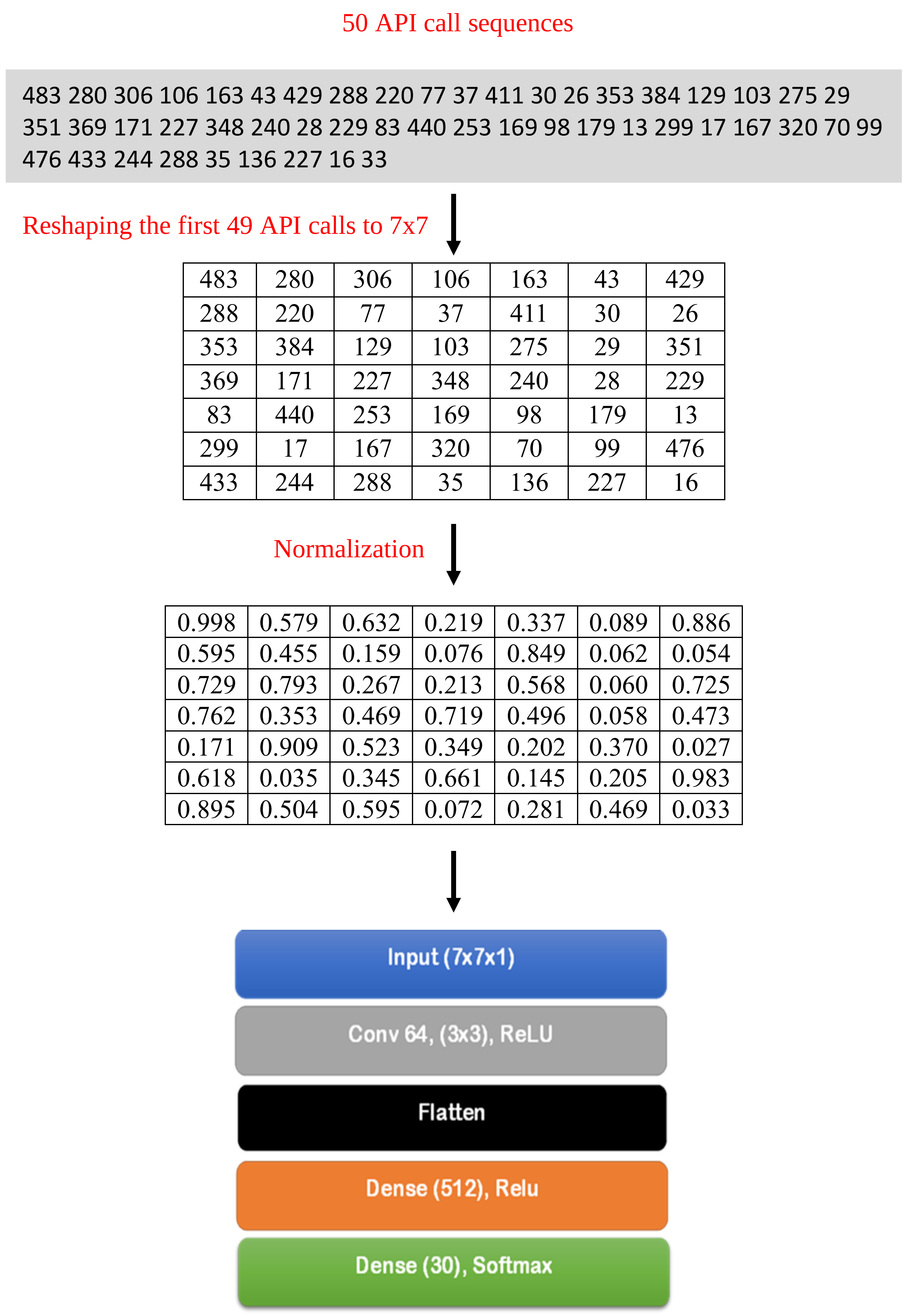
Layer	Output Shape	Parameters
Embedding	(None, 50, 50)	24250
SpatialDropout1D	(None, 50, 50)	0
LSTM	(None, 64)	29440
Dense	(None, 30)	1950
Total params: 55,640		
Trainable params: 55,640		
Non-trainable params: 0		

Table 4.7. Summary of the GRU model with the second dataset.

Layer	Output Shape	Parameters
Embedding	(None, 50, 50)	24250
GRU	(None, 64)	22272
Dense	(None, 30)	1950
Total params: 48,472		
Trainable params: 48,472		
Non-trainable params: 0		

4.2.3. CNN Model with API Call Sequences Dataset

The last of our proposed models also In the second dataset, which is made up of API Call Sequences, CNN technology is used to call 49 API Call Sequences, which are then reshaped into an array of 7x7, normalized by dividing them by a larger number in the API unique index , which is 484 in this study, to make them numbers between 0 and 1, and put into a simple model made up of the Convolutional 2D Layer (64, 3, 3) and a Fully Connected layer, with training taking up 70%, testing, and validation each taking up 15%. The flow chart of the CNN model using the API call dataset method is shown in Figure 4.6, and a summary of the CNN model architecture with parameters is shown in Table 4.8.


50 API call sequences

483 280 306 106 163 43 429 288 220 77 37 411 30 26 353 384 129 103 275 29 351 369 171 227 348 240 28 229 83 440 253 169 98 179 13 299 17 167 320 70 99 476 433 244 288 35 136 227 16 33

Reshaping the first 49 API calls to 7x7

483	280	306	106	163	43	429
288	220	77	37	411	30	26
353	384	129	103	275	29	351
369	171	227	348	240	28	229
83	440	253	169	98	179	13
299	17	167	320	70	99	476
433	244	288	35	136	227	16

Normalization

0.998	0.579	0.632	0.219	0.337	0.089	0.886
0.595	0.455	0.159	0.076	0.849	0.062	0.054
0.729	0.793	0.267	0.213	0.568	0.060	0.725
0.762	0.353	0.469	0.719	0.496	0.058	0.473
0.171	0.909	0.523	0.349	0.202	0.370	0.027
0.618	0.035	0.345	0.661	0.145	0.205	0.983
0.895	0.504	0.595	0.072	0.281	0.469	0.033

Figure 4.6. CNN model with API call sequences method.

Table 4.8. Summary of the CNN model with the second dataset.

Layer	Output Shape	Parameters
Conv2D	(None, 5, 5, 64)	640
Activation	(None, 5, 5, 64)	0
Flatten	(None, 1600)	0
Dense	(None, 512)	819712
Activation	(None, 512)	0
Dense	(None, 30)	15390
Activation	(None, 30)	0
Total params: 835,742		
Trainable params: 835,742		
Non-trainable params: 0		

4.2.4. Real Time Malware Detection and Classification

In the end, after training and making tests, the three most efficient models will be selected to classify and detect malware in real time. One model is for the first dataset , which includes malware and benign images. The second model is for the second dataset, which includes API call sequences. The last model is made up of a shallow CNN model that is based on API call sequences after reshaping it to 7x7 and applying normalization. The general flowchart of the approach that was suggested for this research can be seen in Figure 4.7.

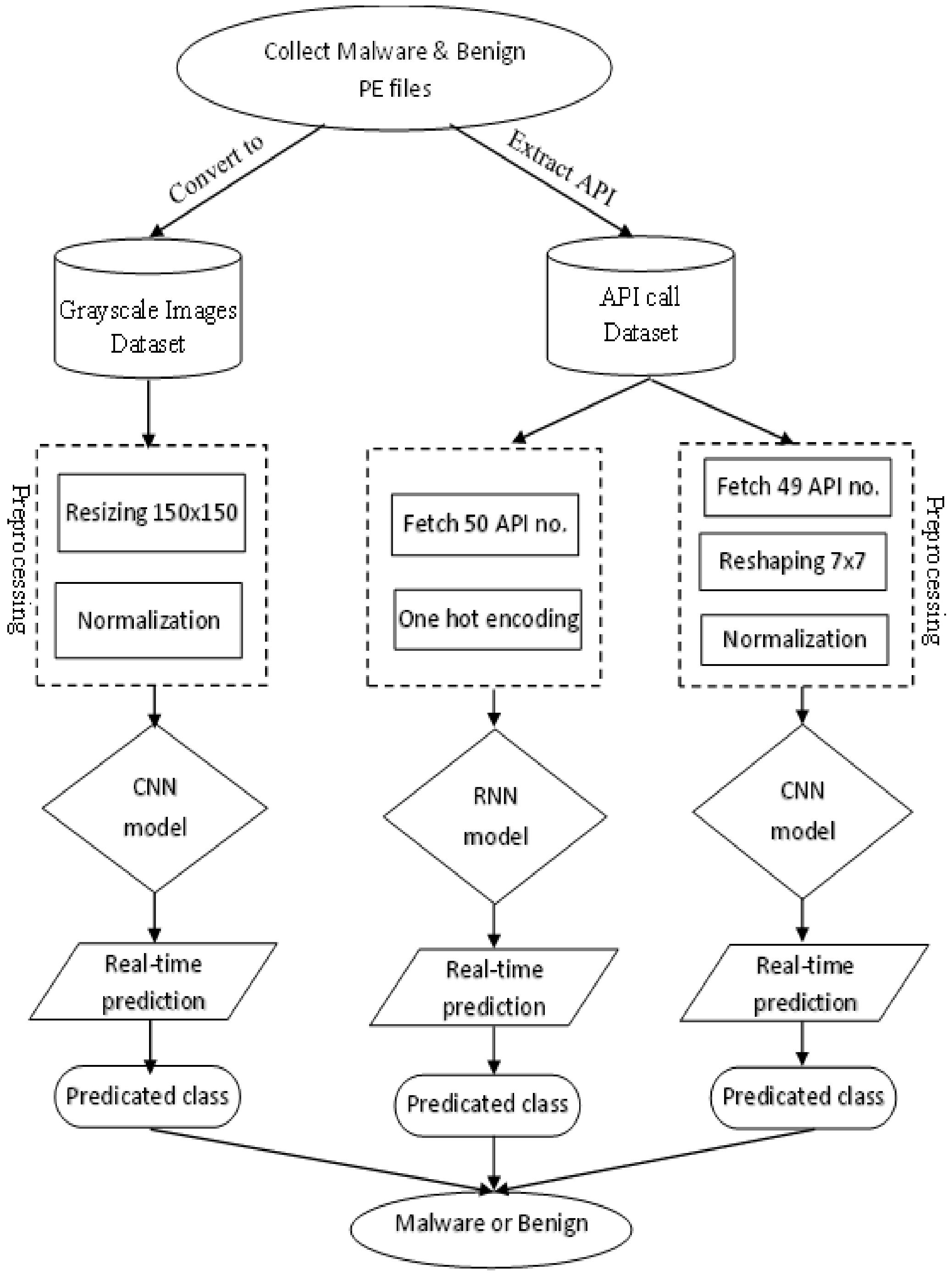


Figure 4.7. General flowchart of proposed method.

.

PART 5

RESULTS AND DISCUSSION

In this section displays the results of all experiments that were carried out in this study and explains the confusion matrix, as well as the discussion and comparison among CNN models and between RNN models with detailed figures and tables. Some experiments were carried out by using the Python programming language on a Jupyter notebook on a PC equipped with an Intel(R) Core (TM) i7- 6600U CPU @ 2.60GHz and 2.80 GHz with 8.00 GB of RAM, while others were carried out on Google Colab.

5.1. CONFUSION MATRIX

The performance of a classification algorithm may be described with the use of a table called a confusion matrix, which displays essential predictive metrics such as recall, f1 score, accuracy, and precision. Confusion matrices are helpful tools because they provide direct comparisons of measurements like True Positive (TP), False Negative (FN), False Positive (FP), and True Negative (TN). In our study, we used these measurements to measure our model's predictive classification. The following equations, numbered 5.1 through 5.4, represent the performance metrics:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

$$F1\,score = 2 * \frac{Recall * Precision}{Recall + Precision} \tag{5.4}$$

TP: The situation in which the real number and expected number are identical is a True Positive number.

FN: A class's False-negative number is the total of all the numbers in the relevant rows, excluding the TP number.

FP: A class's False-positive number is the total of all the numbers in the relevant column, excluding the TP number.

TN: The total of all columns and rows, excluding those for the class for which we are computing the numbers, will represent the True Negative number for a given class.

5.2. IMPLEMENTING CNN MODELS

5.2.1. Implementing our Model

The first model that was developed used CNN technology with the malware and benign images that were used to construct the dataset. We noticed that after testing the model with a total of 50 epochs, it had an accuracy of 98.23%. Figure 5.1 demonstrates the accuracy and the validation accuracy of this model. Additionally, the figure shows both the loss and the validation loss for this model. Table 5.1 outlines the classification metrics (accuracy, precision, recall, and F1 score) that our CNN model achieved on the very first dataset it was applied to.

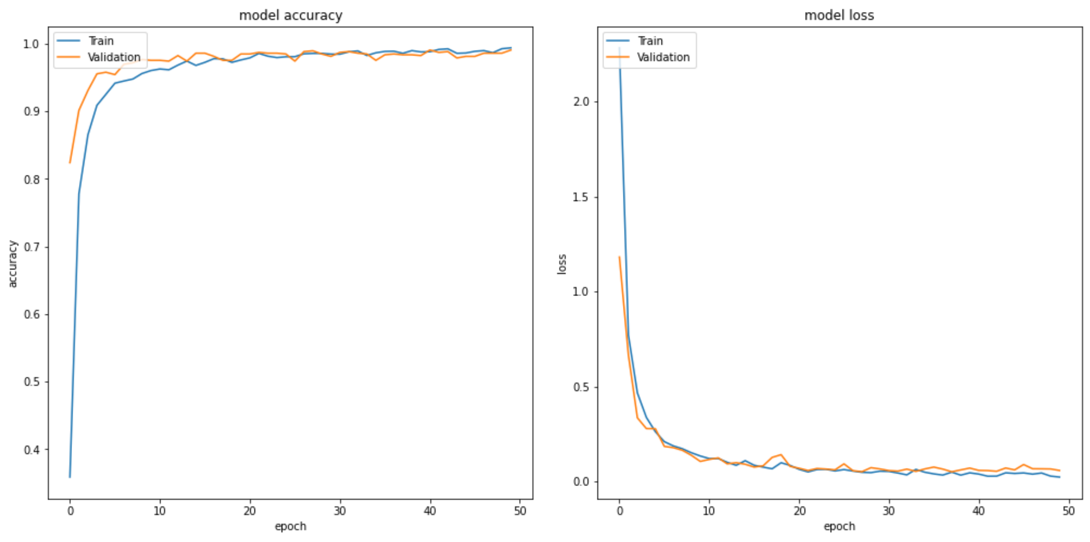


Figure 5.1. Accuracy and loss for our CNN model with first dataset.

Table 5.1. Classification metrics for our CNN model with first dataset.

Class	**Accuracy**	**Precision**	**Recall**	**F1-score**
Benign	0.99	0.96	0.95	0.95
Ako	1.00	1.00	1.00	1.00
Autorun.NE	1.00	0.94	1.00	0.97
Banker.LY	1.00	1.00	1.00	1.00
Delf.DU	0.99	0.81	0.94	0.87
Drolnux.B	1.00	1.00	1.00	1.00
Eggnog.A	1.00	1.00	1.00	1.00
GandCrab.AE	1.00	1.00	0.96	0.98
Ganelp.E	1.00	1.00	1.00	1.00
Linkury.RS!MT	1.00	1.00	1.00	1.00
Neconyd.A	1.00	0.96	1.00	0.98
Nemucod	1.00	1.00	1.00	1.00
Neojit.A	1.00	1.00	1.00	1.00
OpenInstaller	1.00	1.00	1.00	1.00
Playtech	1.00	0.96	1.00	0.98
QQPass.GP	1.00	1.00	1.00	1.00
Qukart	1.00	1.00	1.00	1.00
Resur.A!epo	1.00	1.00	1.00	1.00
Shodi.A	1.00	0.94	1.00	0.97
Simda.D	1.00	1.00	0.88	0.93
Sivis.A	1.00	0.91	1.00	0.95
Small.M	1.00	1.00	1.00	1.00
Soltern!rfn	1.00	1.00	1.00	1.00
Trickbot.GML!	1.00	1.00	1.00	1.00
Unruy.F	1.00	1.00	1.00	1.00
Upatre.A	1.00	1.00	1.00	1.00
Urelas.AA	1.00	1.00	0.96	0.98
Wabot.A	1.00	1.00	0.97	0.98
Yoof.E	1.00	1.00	1.00	1.00
Zombie!rfn	1.00	1.00	0.89	0.94
Accuracy	0.98			
Macro Avg		0.98	0.98	0.98
Weighted Avg.		0.98	0.98	0.98

We observed that all classes had a high degree of accuracy in their classifications by comparing the predicted and actual values, as shown in Figures 5.2, which demonstrate the confusion matrices produced by our approach.

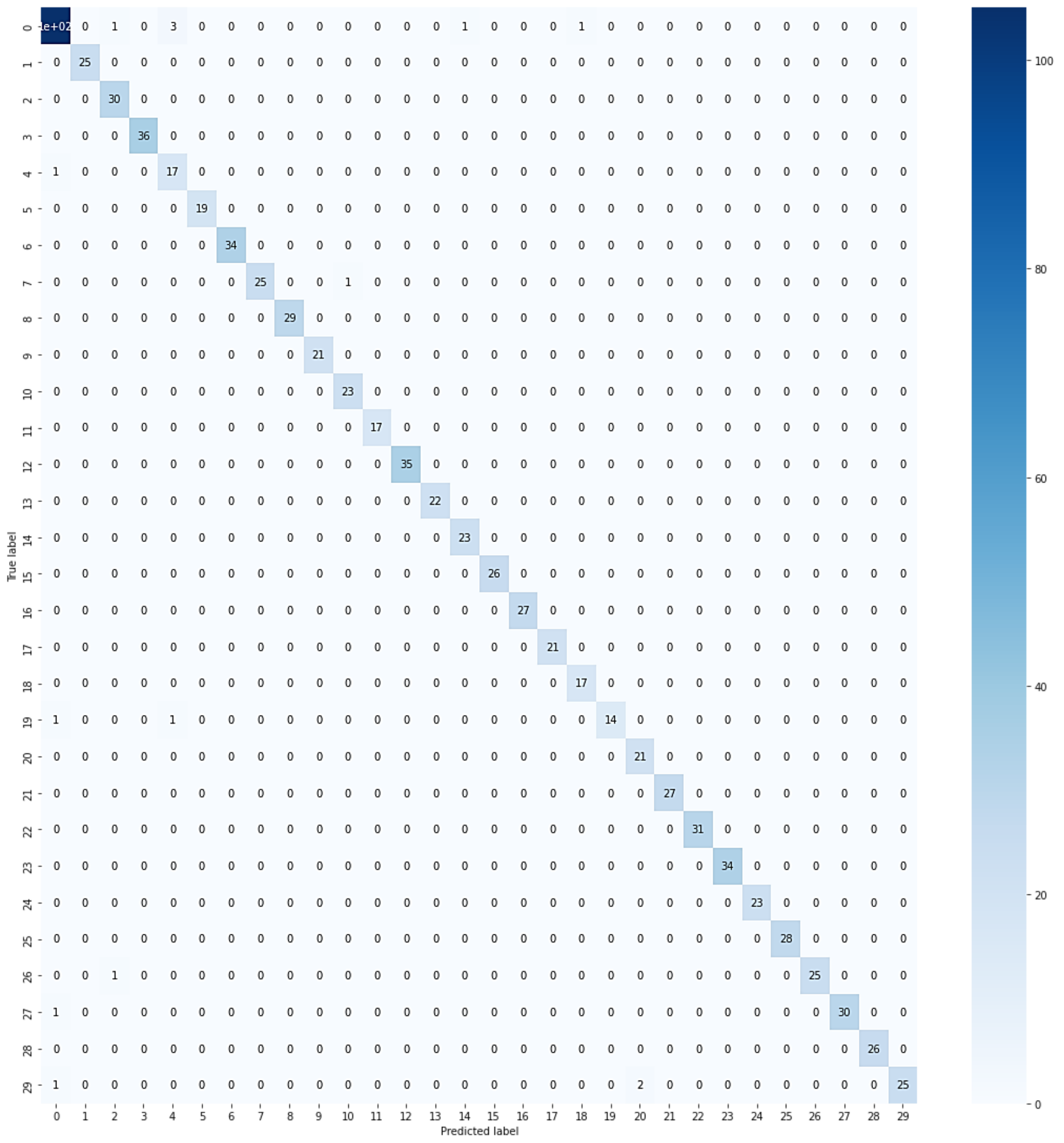


Figure 5.2. Confusion Matrix for our CNN model.

5.1.2. Implementing Pretrained Models

We conducted experiments on our first dataset, which is comprised of images of samples, using three pre-trained CNN models, including VGG16, Inception V3, and Resnet50, so that we could perform a comparison between our CNN model and those of other models. In the preprocessing stage, the images are resized to 224x224 before passing through the models. These models are designed with color images (3 channels) and our dataset only contains grayscale images for samples in the first dataset.

In the Vgg16 model, after the normalization process was done and the dataset was split into three parts (70:15:15) for training, testing, and validation, respectively, the model had a 98 percent accuracy rate.

After the dataset was normalized for the Inception V3 model, it was then split into thirds: (80:10:10) for training, testing, and validation, respectively. The model's accuracy was determined to be 97.17 percent.

Without doing any normalization, the dataset was partitioned in the Resnet50 model as follows: (70:15:15) for training, testing, and validation, respectively. When we used this model to normalize the data, we discovered that the accuracy was lower than when we used normalization. The accuracy of the model was 98.35 percent.

Classification metrics for pretrained CNN models (VGG16, Inception V3 and Resnet50) are given in Table 5.2. Figures 5.3, 5.4, and 5.5, respectively, illustrate accuracy and loss for the VGG16, Inception V3, and Resnet50.

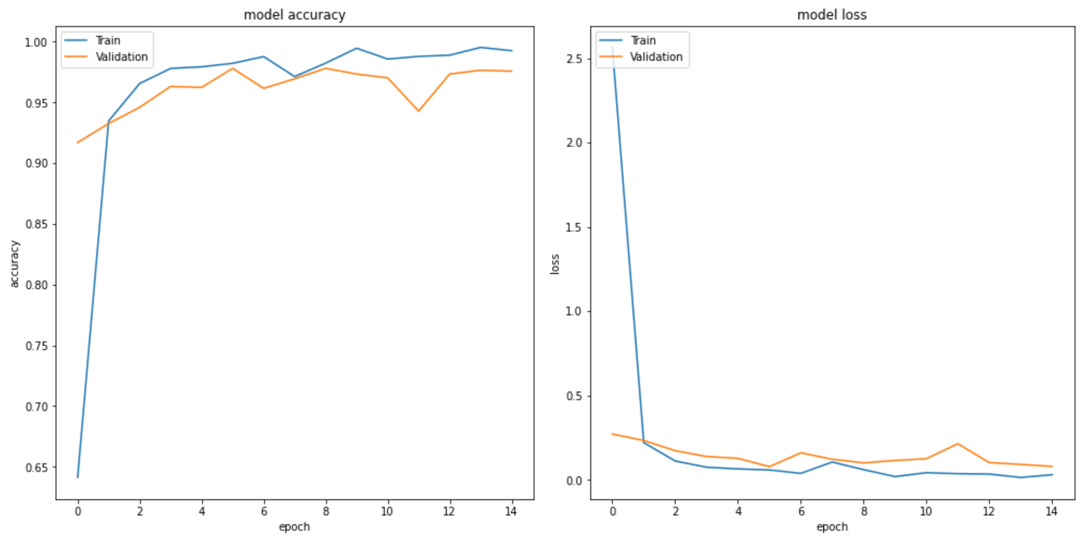


Figure 5.3. The VGG16 model's accuracy and loss with the first dataset.

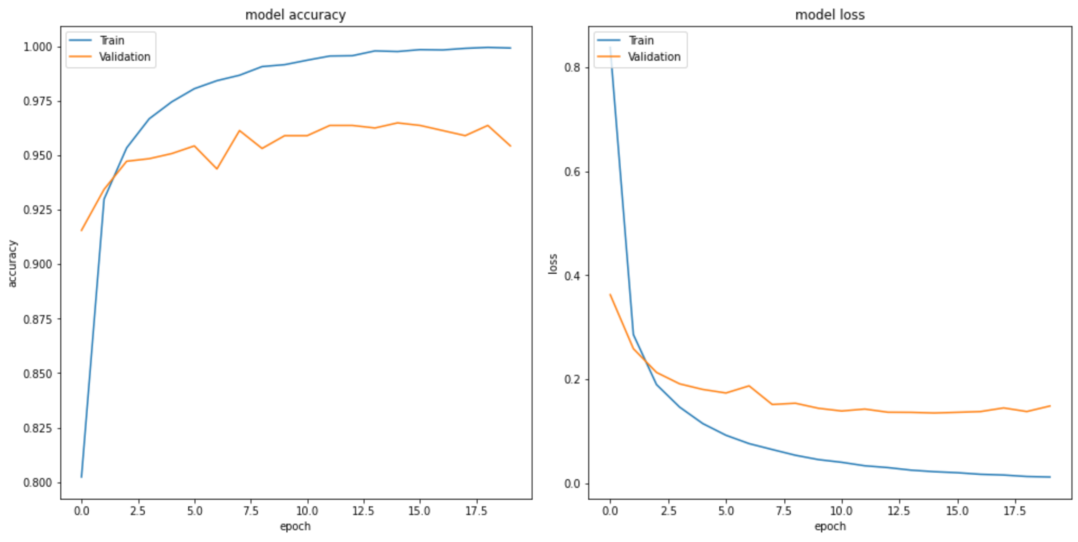


Figure 5.4. The Inception V3 model's accuracy and loss with the first dataset.

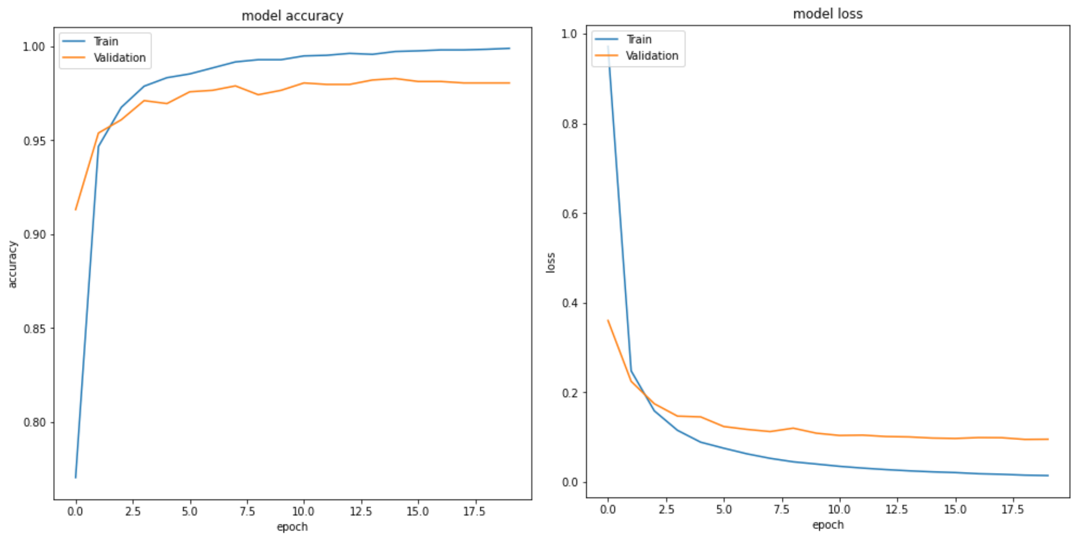


Figure 5.5. The Resnet50 model's accuracy and loss with the first dataset.

Figures 5.6, 5.7, and 5.8 illustrate the confusion matrices for the VGG16, Inception V3, and Resnet 50 models, respectively.

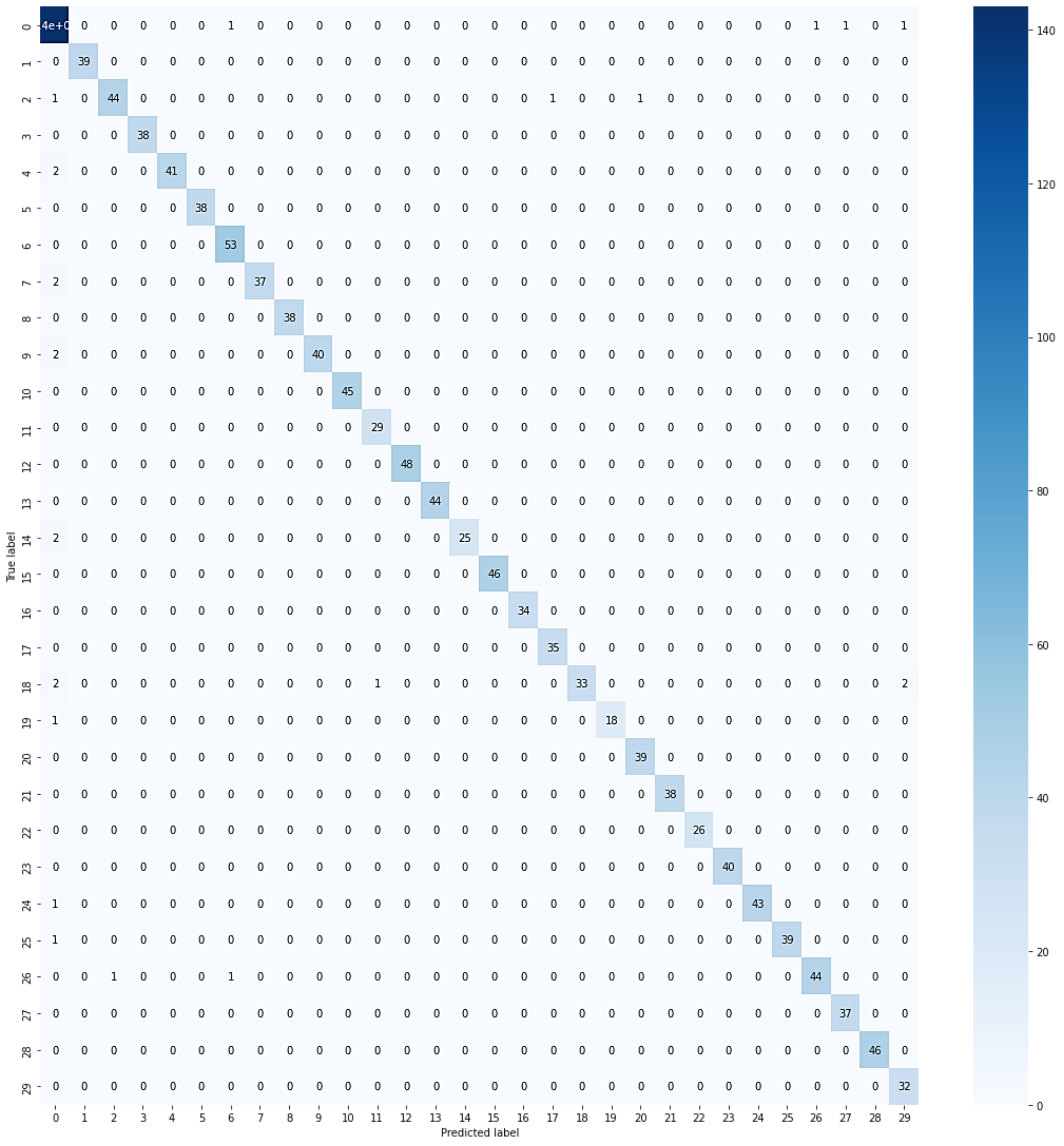


Figure 5.6. Confusion matrix for VGG16 model.

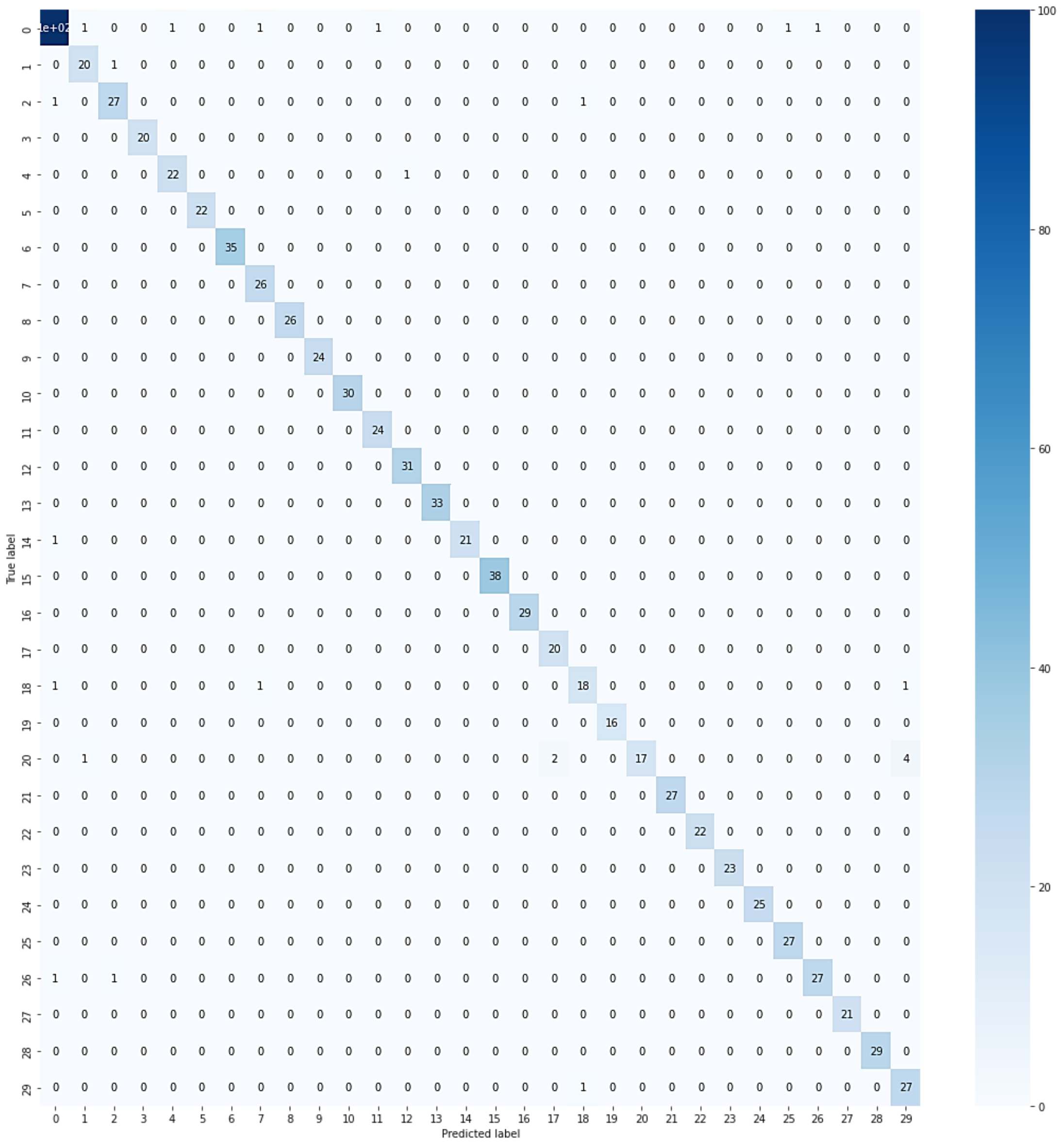


Figure 5.7. Inception V3 model's confusion matrix.

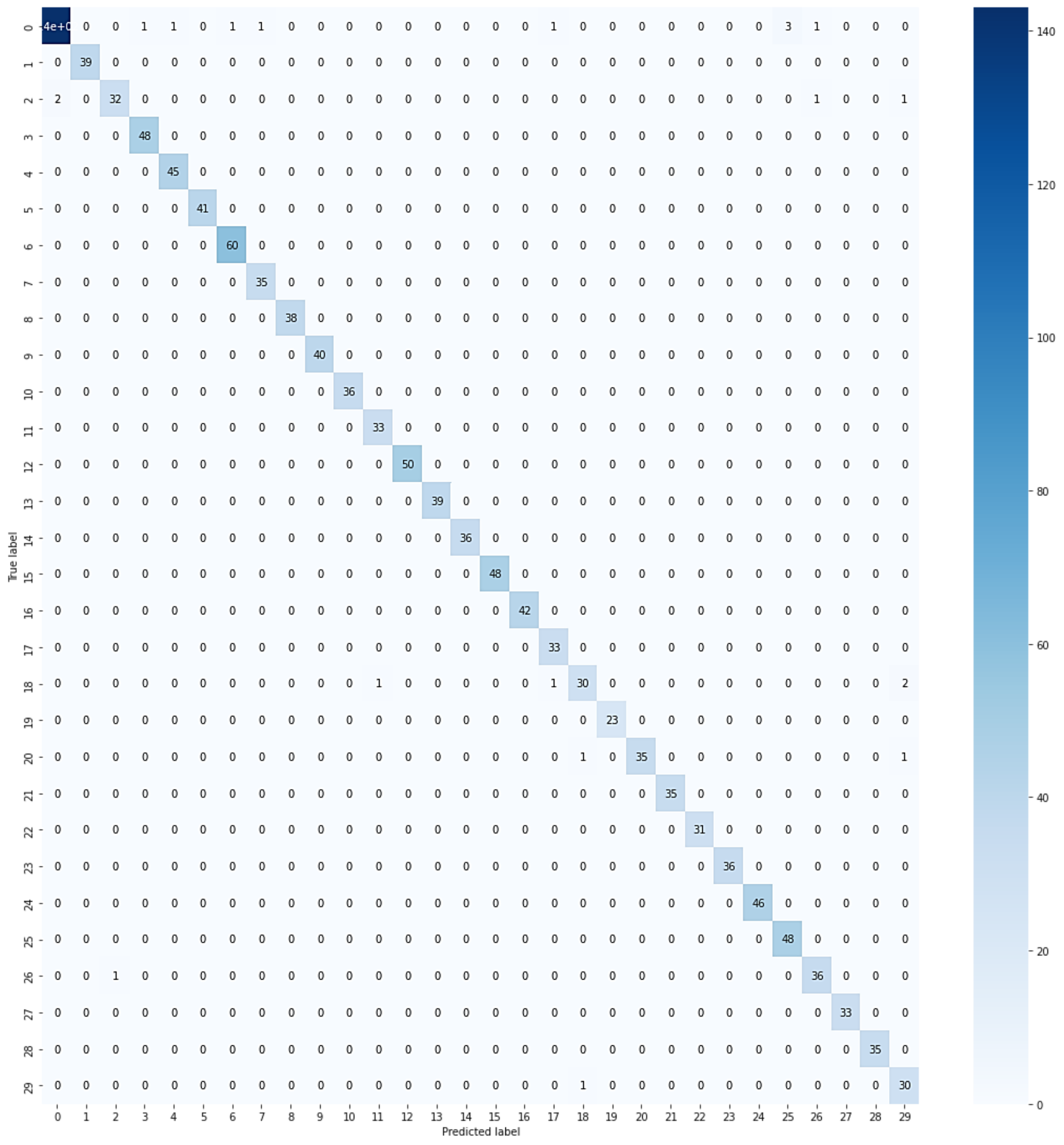


Figure 5.8. Resnet50 model's confusion matrix.

Table 5.2. Classification metrics for pretrained CNN models.

Class	Inception V3				Resnet 50				VGG 16			
	Acc	**Pre**	**Rec**	**F1**	**Acc**	**Pre**	**Rec**	**F1**	**Acc**	**Pre**	**Rec**	**F1**
Benign	0.99	0.96	0.94	0.95	0.99	0.99	0.94	0.96	0.99	0.91	0.97	0.94
Ako	1.00	0.91	0.95	0.93	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Autorun.NE	1.00	0.93	0.93	0.93	1.00	0.97	0.89	0.93	1.00	0.98	0.94	0.96
Banker.LY	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.99	1.00	1.00	1.00	1.00
Delf.DU	1.00	0.96	0.96	0.96	1.00	0.98	1.00	0.99	1.00	1.00	0.95	0.98
Drolnux.B	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Eggnog.A	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.99	1.00	0.96	1.00	0.98
GandCrab.AE	1.00	0.93	1.00	0.96	1.00	0.97	1.00	0.99	1.00	1.00	0.95	0.97
Ganelp.E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Linkury.RS!MT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.98
Neconyd.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Nemucod	1.00	0.96	1.00	0.98	1.00	0.97	1.00	0.99	1.00	0.97	1.00	0.98
Neojit.A	1.00	0.97	1.00	0.98	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
OpenInstaller	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Playtech	1.00	1.00	0.95	0.98	1.00	1.00	1.00	1.00	1.00	1.00	0.93	0.96
QQPass.GP	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Qukart	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Resur.A!epo	1.00	0.91	1.00	0.95	1.00	0.94	1.00	0.97	1.00	0.97	1.00	0.99
Shodi.A	1.00	0.90	0.86	0.88	1.00	0.94	0.88	0.91	1.00	1.00	0.87	0.93
Simda.D	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.97
Sivis.A	0.99	1.00	0.71	0.83	1.00	1.00	0.95	0.97	1.00	0.97	1.00	0.99
Small.M	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Soltern!rfn	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Trickbot.GML!	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Unruy.F	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.99
Upatre.A	1.00	0.96	1.00	0.98	1.00	0.94	1.00	0.97	1.00	1.00	0.97	0.99
Urelas.AA	1.00	0.96	0.93	0.95	1.00	0.95	0.97	0.96	1.00	0.98	0.96	0.97
Wabot.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97	1.00	0.99
Yoof.E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Zombie!rfn	0.99	0.84	0.96	0.90	1.00	0.88	0.97	0.92	1.00	0.91	1.00	0.96
Accuracy	0.97				0.98				0.98			
Macro Avg		0.97	0.97	0.97		0.98	0.99	0.98		0.99	0.98	0.98
Weighted Avg.		0.97	0.97	0.97		0.98	0.98	0.98		0.98	0.98	0.98

The results showed that our proposed CNN network-based model is more efficient than the other models (VGG16, Inception V3, and Resnet50) in terms of how many parameters the model generates and how much memory it consumes, especially since our model used images of 150x150 input size as shown in Table 5.3. The Resnet50 model and our own model both have an accuracy that is about close to one another.

Table 5.3. Comparison between our model and pretrained network models.

Model	**Image size**	**Total layers**	**Total parameters**	**Accuracy**	**Precision**	**Recall**	**F1 score**
VGG16	220x220	16	134,383,484	0.98	0.99	0.98	0.98
Inception V3	220x220	42	21,864,284	0.97	0.97	0.97	0.97
Resnet50	220x220	50	23,649,212	0.98	0.98	0.99	0.98
Our Model	**150x150**	**5**	**639,774**	**0.98**	**0.98**	**0.98**	**0.98**

The model was also trained with 5-fold cross validation and the result is shown in Table 5.4.

Table 5.4. The test accuracy of our model for 5-fold cross validation.

Fold	**Test accuracy**
1	0.987
2	0.978
3	0.985
4	0.978
5	0.981
Average	**0.982**

We compared our CNN model in this study with many previous studies, and it was found that our model is characterized by high accuracy as well as high efficiency in terms of speed, low memory consumption, and the least number of weighted layers for the model in dealing with grayscale malware images, as shown in Table 5.5.

Table 5.5. A comparison between our model and a set of previous studies.

Reference	Model	Accuracy	weighted layers	İmage size	Type
Sun and Qian [18]	CNN RNN	99.5%	7	128x128	Classification
Hemalatha et al. [22]	DenseNet201	98.23% 98.46% 98.21% 89.48%	201	64x64	Detection and Classification
Anandhi et al. [23]	DenseNet201	99.94% 98.98%	201	256x256	Detection and Classification İn Real time
Jian et al. [25]	SEResNet50 + Bi-LSTM + Attention	98.31%	> 50	256x256x3 3- channel	Detection and Classification
Mitsuhashi and Shinagawa [14]	VGG19	99.72%	19	224 x 224	Classification
Jin et al. [9]	Autoencoder	93%	10	Unknown	Detection
Go et al. [13]	ResNeXt50	98.8%	50	224 x 224	Classification
Vasan et al. [15]	CNN	98.82% 97.35%	> 15	224x224x3 3-channel	Detection and Classification
Xiao et al. [16]	CNN -SVM Entropy graphs	99.7% 100%	14	300 x 300	Classification
Vasan et al. [17]	VGG16 Resnet50 SVM	99% 98%	16 50	224 x 224	Classification
Qiao et al. [26]	LeNet5	98.76%	5	256x256x3 3- channel	Classification
Our method [80]	**CNN**	**98%**	**5**	**150x150**	**Detection and Classification İn Real time**

5.2. IMPLEMENTING RNN MODELS

5.2.1. LSTM

When applied to the second dataset, which is comprised of API call sequences, the suggested LSTM model was assessed using 30 epochs, and the findings showed that it had an accuracy of 99.45 percent. A representation of the LSTM's accuracy and validation accuracy can be seen in Figure 5.9. The figure also shows both losses and the validation loss. A confusion matrix is shown in Figure 5.10, which was generated using the LSTM model.

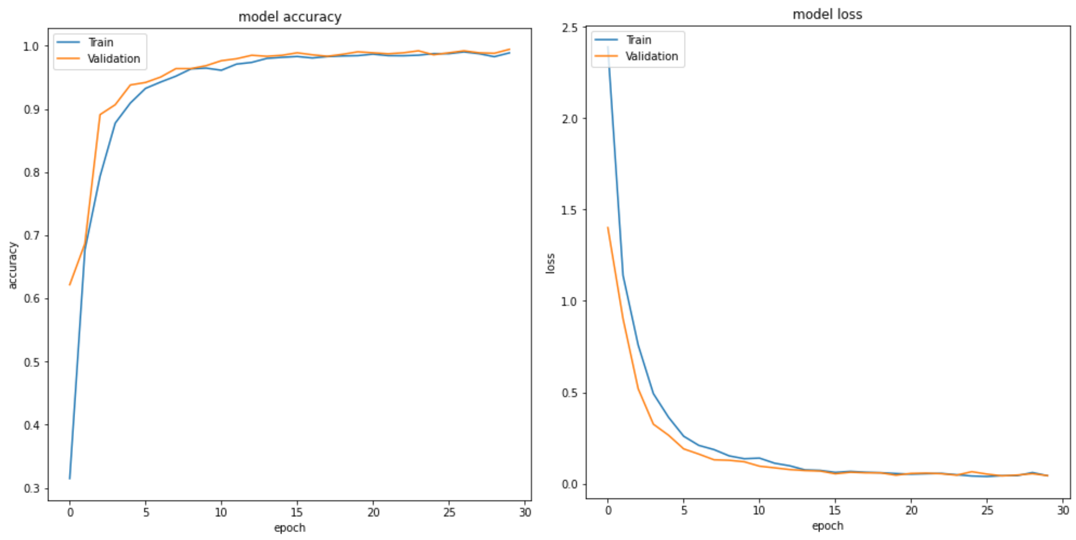


Figure 5.9. The LSTM model's accuracy and loss.

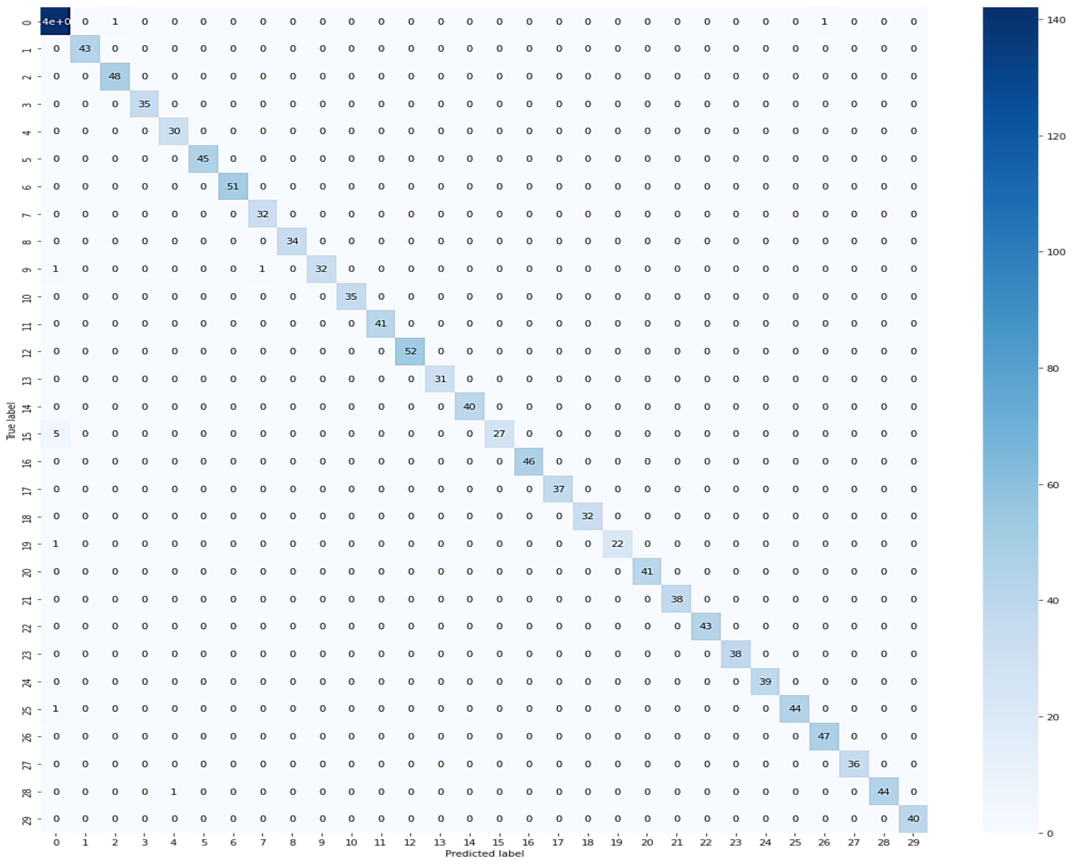


Figure 5.10. The LSTM model's confusion matrix.

5.2.2. GRU

The accuracy of the recommended GRU model was measured using 30 epochs, and the findings indicated that it had the same accuracy as the LSTM model, which was 99.45 percent. The training accuracy, validation accuracy, loss, and validation loss of the GRU are shown graphically in Figure 5.11. The GRU model's confusion matrix is displayed in Figure 5.12.

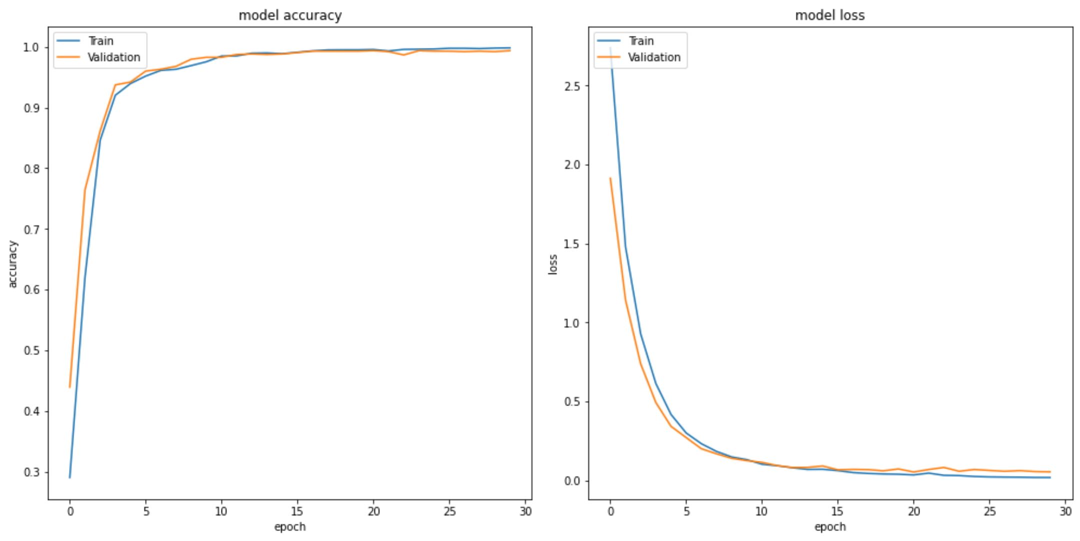


Figure 5.11. The GRU model's accuracy and loss.

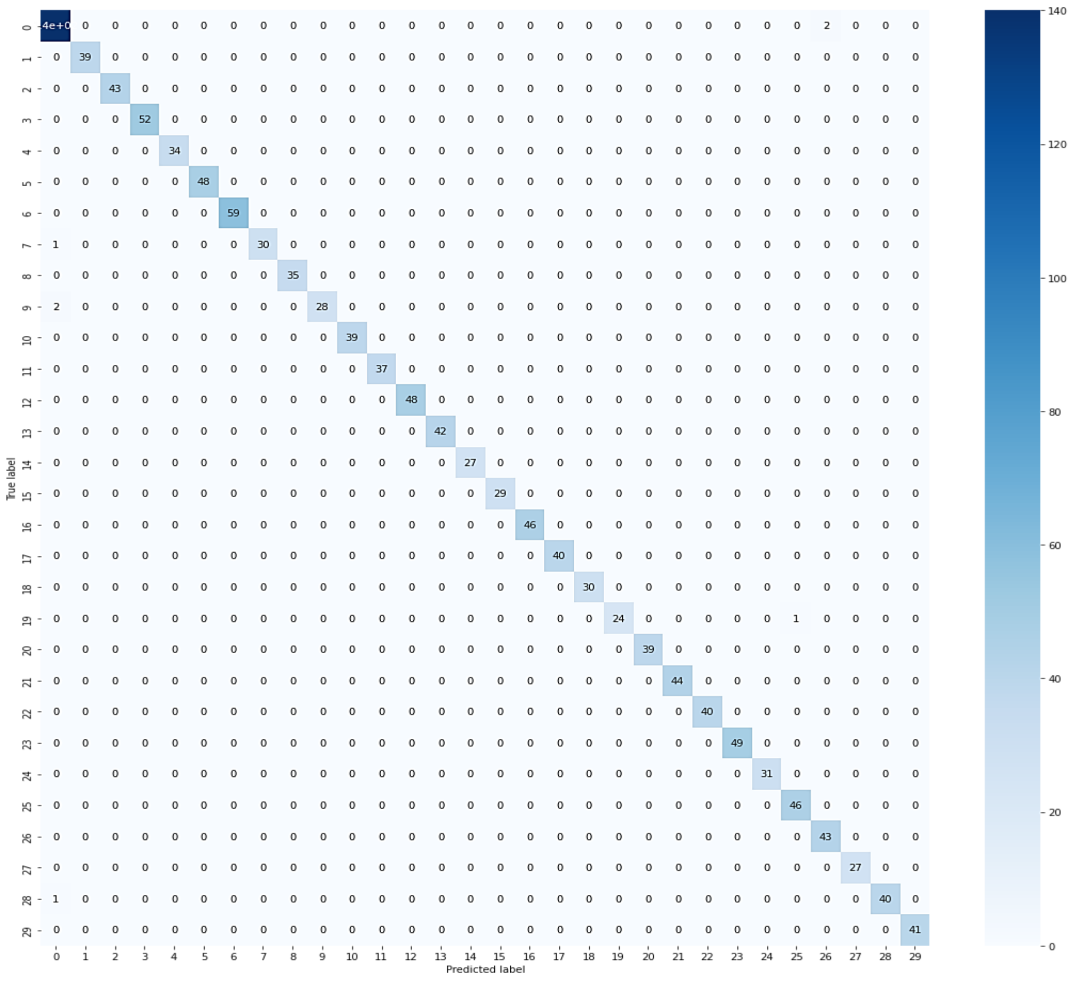


Figure 5.12. The GRU model's confusion matrix.

In this study the findings showed that the proposed model based on the GRU network outperforms the LSTM method in term of having fewer parameters although they have the same accuracy. Table 5.6 compares classification metrics for LSTM and GRU models with a second dataset.

Table 5.6. Classification metrics for LSTM and GRU models with second dataset.

Class	**LSTM**				**GRU**			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
Benign	0.99	0.95	0.99	0.97	1.00	0.97	0.99	0.98
Ako	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Autorun.NE	1.00	0.98	1.00	0.99	1.00	1.00	1.00	1.00
Banker.LY	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Delf.DU	1.00	0.97	1.00	0.98	1.00	1.00	1.00	1.00
Drolnux.B	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Eggnog.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
GandCrab.AE	1.00	0.97	1.00	0.98	1.00	1.00	0.97	0.98
Ganelp.E	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Linkury.RS!MTB	1.00	1.00	0.94	0.97	1.00	1.00	0.93	0.97
Neconyd.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Nemucod	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Neojit.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
OpenInstaller	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Playtech	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
QQPass.GP	1.00	1.00	0.84	0.92	1.00	1.00	1.00	1.00
Qukart	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Resur.A!epo	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Shodi.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Simda.D	1.00	1.00	0.96	0.98	1.00	1.00	0.96	0.98
Sivis.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Small.M	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Soltern!rfn	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Trickbot.GML!M	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Unruy.F	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Upatre.A	1.00	1.00	0.98	0.99	1.00	0.98	1.00	0.99
Urelas.AA	1.00	0.98	1.00	0.99	1.00	0.96	1.00	0.98
Wabot.A	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Yoof.E	1.00	1.00	0.98	0.99	1.00	1.00	0.98	0.99
Zombie!rfn	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Accuracy	0.99				0.99			
Macro Avg		0.99	0.99	0.99		1	0.99	1
Weighted Avg.		0.99	0.99	0.99		0.99	0.99	0.99

5.3. IMPLEMENTING OUR CNN MODEL WITH API CALL SEQUENCES

This model was applied to the second dataset, which was based on API call sequence numbers, after calling 49 API call sequences to reshape it to 7x7 and applying CNN with a division of 70%, 15%, and 15% of training, testing, and verification data, respectively, and 15 epochs to get 99% accuracy. The model's accuracy and loss during training and validation are represented in Figure 5.13. Our CNN model's confusion matrix for the second dataset is shown in Figure 5.14 and classification metrics are included in Table 5.7. In the end, we compared the three models trained on the API call dataset, as can be seen in Table 5.8. The accuracy was nearly the same for all three, but the CNN model was the fastest because it was working with a 2D image consisting of a 7x7 matrix, and also because the CNN models were characterized by feature extraction.

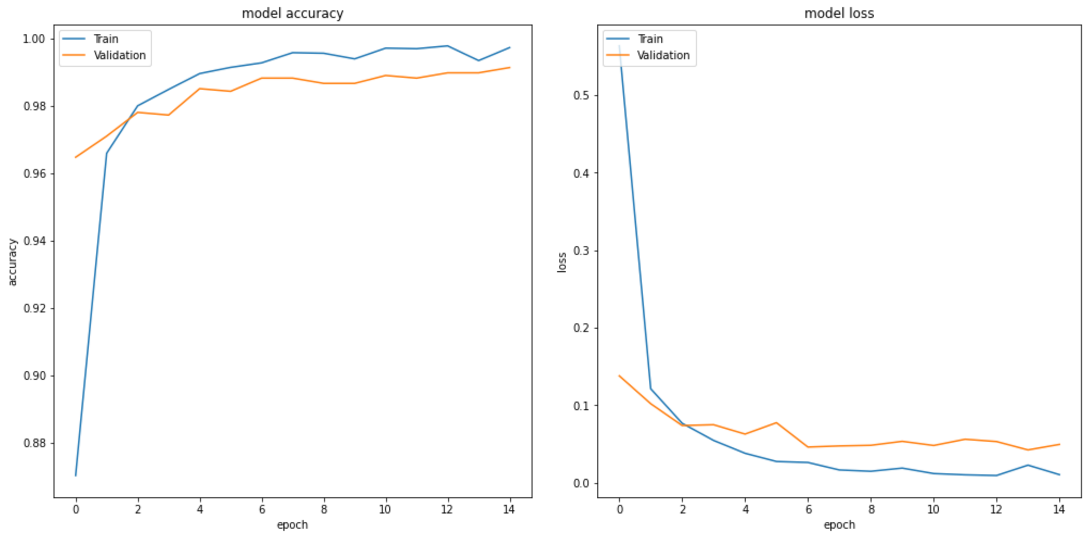


Figure 5.13. Accuracy and loss for our CNN model with second dataset.

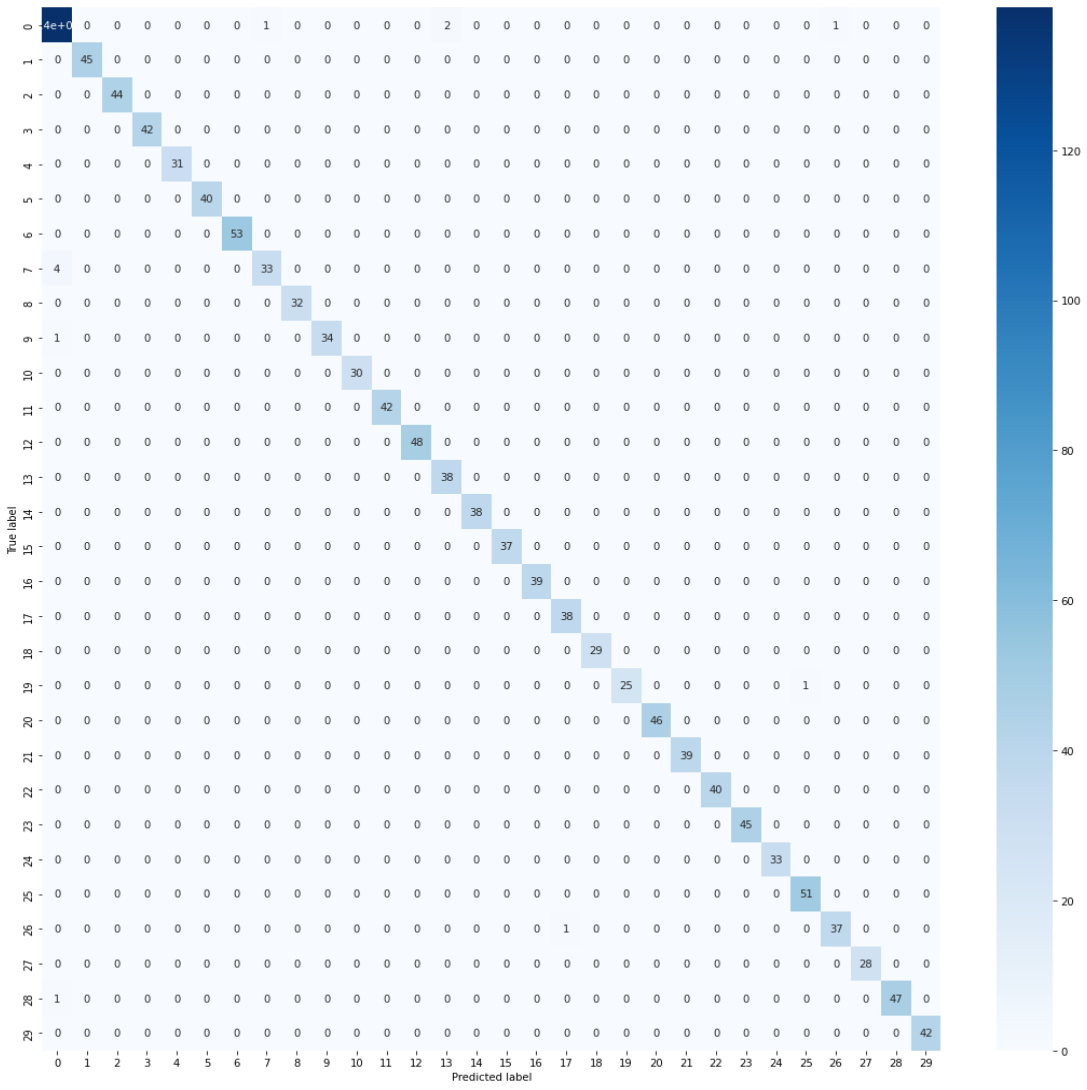


Figure 5.14. The CNN model's confusion matrix with the second dataset.

Table 5.7. Classification metrics for our CNN model with second dataset.

Class	Accuracy	Precision	Recall	F1-score
Benign	0.99	0.96	0.97	0.97
Ako	1.00	1.00	1.00	1.00
Autorun.NE	1.00	1.00	1.00	1.00
Banker.LY	1.00	1.00	1.00	1.00
Delf.DU	1.00	1.00	1.00	1.00
Drolnux.B	1.00	1.00	1.00	1.00
Eggnog.A	1.00	1.00	1.00	1.00
GandCrab.AE	1.00	0.97	0.89	0.93
Ganelp.E	1.00	1.00	1.00	1.00
Linkury.RS!MTB	1.00	1.00	0.97	0.99
Neconyd.A	1.00	1.00	1.00	1.00
Nemucod	1.00	1.00	1.00	1.00
Neojit.A	1.00	1.00	1.00	1.00
OpenInstaller	1.00	0.95	1.00	0.97
Playtech	1.00	1.00	1.00	1.00
QQPass.GP	1.00	1.00	1.00	1.00
Qukart	1.00	1.00	1.00	1.00
Resur.A!epo	1.00	0.97	1.00	0.99
Shodi.A	1.00	1.00	1.00	1.00
Simda.D	1.00	1.00	0.96	0.98
Sivis.A	1.00	1.00	1.00	1.00
Small.M	1.00	1.00	1.00	1.00
Soltern!rfn	1.00	1.00	1.00	1.00
Trickbot.GML!MTB	1.00	1.00	1.00	1.00
Unruy.F	1.00	1.00	1.00	1.00
Upatre.A	1.00	0.98	1.00	0.99
Urelas.AA	1.00	0.97	0.97	0.97
Wabot.A	1.00	1.00	1.00	1.00
Yoof.E	1.00	1.00	0.98	0.99
Zombie!rfn	1.00	1.00	1.00	1.00
Accuracy	0.99			
Macro Avg		0.99	0.99	0.99
Weighted Avg.		0.99	0.99	0.99

Table 5.8. Comparison between CNN model and RNN models in API call dataset.

Model	API length	Accuracy	Precision	Recall	F1 Score
LSTM	50	0.99	0.99	0.99	0.99
GRU	50	0.99	1	0.99	1
CNN	**49 (7x7)**	**0.99**	**0.99**	**0.99**	**0.99**

The model was also trained using 5-fold cross validation, and Table 5.9 displays the outcome. According to Table 5.10, which compares our weighted lite CNN model to much other research, our model is distinguished by high accuracy as well as great efficiency in terms of low memory consumption and the fewest number of weighted layers.

Table 5.9. The test accuracy of our model for 5-fold cross validation.

Fold	**Test accuracy**
1	0.991
2	0.987
3	0.989
4	0.992
5	0.997
Average	**0.99**

Table 5.10. A comparison between our model and a set of previous studies.

Authors	**Model**	**Accuracy**	**weighted layers**	**API length**	**Type**
Xiaofeng et al. [39]	LSTM – RF	95.7%	Unknown	>100	Detection
Liu and Wang [38]	BLSTM	97.85%	Unknown	Unknown	Detection
Catak et al. [37]	LSTM	98.50%	> 4	100	Detection and Classification
Xu et al. [40]	Malbert	99.98% 99.82%	> 12	Unknown	Detection
Ye et al. [43]	Autoencoder RBMs	98.20%	> 5	Unknown	Detection
Oliveira et al. [46]	LSTM	99%	Unknown	100	Detection
Tang and Qian [47]	CNN	98%-99%	4	$16 \times 16 \times 3$ 3-channel	Classification
Aditya et al. [48]	LSTM-RMSProp	97.30%	4	Unknown	Detection and Classification İn Real time
Our method	**2D-CNN**	**99%**	**3**	**49 (7x7)**	**Detection and Classification İn Real time**

5.4. IMPLEMENTING REAL TIME FRAMEWORK

In this study, our framework consists of the best three models. After training and testing, we discovered the best model treated with the first dataset, which contained images of malicious and goodware software, was our approach in regard to the parameters and amount of memory used. When it came to accuracy, our model was very close to the resnet50 model.

The second model used for real-time detection is GRU, which proved it's efficient in terms of smaller parameters and faster than LSTM, although the two models have the same accuracy.

The third model was the proposed model with the second dataset, which contained API call sequences numbers by calling 49 numbers from the API call and converting them to a 2D array and making normalization.

Our framework proved it's efficient in classification and detection in real time when implemented on samples collected and not existing in our dataset but belonging to the same families.

PART 6

CONCLUSION

6.1. CONCLUSION

In this study, we presented two new datasets that we created using two methods: The first dataset consists of images of samples (malware and benign) to detect samples statically. The second dataset includes API call sequences for the same samples in the first dataset to detect samples dynamically.

Different models were developed based on deep learning approaches for training and testing on these two datasets. Our proposed CNN model based on the first dataset which included malicious and benign images proved it has the best performance compared to pretrained networks (VGG16, Inception V3, and Resnet50) through which it reached an accuracy of 98.23% with a smaller quantity of parameters and memory consumption.

In the second dataset, which comprises API call sequences, we ran experiments on two model types of the RNN algorithms (LSTM and GRU). Although the two models got the same accuracy, which is 99.45%. In terms of the quantity of parameters generated by these two RNN models, the GRU model has been shown to have the best performance.

also in the second dataset, which included API call sequence numbers, our proposed model, which used CNN as its foundation, was implemented and achieved 99% accuracy while being faster than all other models in our experiments.
After the training was done, best three models were saved and use to find and classify malware in real time.

Through the results, it was found that the detection and classification of malware into the families to which it belongs by the dynamic method based on API call sequences is faster than the static method based on malware images. According to the findings of our investigation, different methods of models are superior to one. Even if one of the models can't find malware, the others can, especially if the malware is encrypted or uses other methods to hide itself.

6.2. FUTURE WORK

In a future study, we will collect a larger number of benign files and malicious software and convert them to RGB color images since they have good feature extraction. SPP.NET will be used to deal with images of different sizes and not be restricted to a certain size. In terms of detecting malware with API call sequences, there are types of malware models from which API call sequences cannot be extracted by the Pefile library for python language. In the future, we would like to analyze these kinds of malware samples in-depth.

REFERENCES

1. Internet: AV-TEST, "Last 10 Years Malware Statistics", **https://www.av-test.org/en/statistics/malware/** (2021).

2. You, I. and Yim, K., "Malware obfuscation techniques: A brief survey", (2010).

3. Gandotra, E., Bansal, D., and Sofat, S., "Malware analysis and classification: A survey", ***Journal Of Information Security***, 2014: (2014).

4. Moser, A., Kruegel, C., and Kirda, E., "Limits of static analysis for malware detection", (2007).

5. Qiu, H., Noura, H., Qiu, M., Ming, Z., and Memmi, G., "A user-centric data protection method for cloud storage based on invertible DWT", ***IEEE Transactions On Cloud Computing***, 9 (4): 1293–1304 (2019).

6. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., and Zhong, Y., "Vuldeepecker: A deep learning-based system for vulnerability detection", ***ArXiv Preprint ArXiv:1801.01681***, (2018).

7. Idika, N. and Mathur, A. P., "A survey of malware detection techniques", ***Purdue University***, 48 (2): (2007).

8. Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S., "Malware images: Visualization and automatic classification", ***ACM International Conference Proceeding Series***, (July): (2011).

9. Jin, X., Xing, X., Elahi, H., Wang, G., and Jiang, H., "A malware detection approach using malware images and autoencoders", ***Proceedings - 2020 IEEE 17th International Conference On Mobile Ad Hoc And Smart Systems, MASS 2020***, 631–639 (2020).

10. "VirusTotal", **https://www.virustotal.com/** (2021).

11. "Malshare", **https://malshare.com/** (2021).

12. "Virusshare", **https://virusshare.com/** (2022).

13. Go, J. H., Jan, T., Mohanty, M., Patel, O. P., Puthal, D., and Prasad, M., "Visualization Approach for Malware Classification with ResNeXt", ***2020 IEEE Congress On Evolutionary Computation, CEC 2020 - Conference Proceedings***, 4–10 (2020).

14. Mitsuhashi, R. and Shinagawa, T., "High-Accuracy Malware Classification with a Malware-Optimized Deep Learning Model", (2020).

15. Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., and Zheng, Q., "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture", ***Computer Networks***, 171 (April): (2020).

16. Xiao, G., Li, J., Chen, Y., and Li, K., "MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks", ***Journal Of Parallel And Distributed Computing***, 141: 49–58 (2020).

17. Vasan, D., Alazab, M., Wassan, S., Safaei, B., and Zheng, Q., "Image-Based malware classification using ensemble of CNN architectures (IMCEC)", ***Computers And Security***, 92: 101748 (2020).

18. Sun, G. and Qian, Q., "Deep Learning and Visualization for Identifying Malware Families", ***IEEE Transactions On Dependable And Secure Computing***, 18 (1): 283–295 (2021).

19. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., and Ahmadi, M., "Microsoft malware classification challenge", ***ArXiv Preprint ArXiv:1802.10135***, (2018).

20. Bozkir, A. S., Cankaya, A. O., and Aydos, M., "Utilization and comparision of convolutional neural networks in malware recognition", (2019).

21. Nappa, A., Rafique, M. Z., and Caballero, J., "The MALICIA dataset: identification and analysis of drive-by download operations", ***International Journal Of Information Security***, 14 (1): 15–33 (2015).

22. Hemalatha, J., Roseline, S. A., Geetha, S., Kadry, S., and Damaševičius, R., "An Efficient DenseNet-Based Deep Learning Model for Malware Detection", ***Entropy***, 23 (3): 344 (2021).

23. Anandhi, V., Vinod, P., and Menon, V. G., "Malware visualization and detection using DenseNets", ***Personal And Ubiquitous Computing***, (c): (2021).

24. Mai, J., Cao, C., Shi, F., and Chen, X., "Malware Variant Detection Based on Decomposed Deep Convolutional Network", ***2021 IEEE 6th International Conference On Big Data Analytics, ICBDA 2021***, 333–338 (2021).

25. Jian, Y., Kuang, H., Ren, C., Ma, Z., and Wang, H., "A novel framework for image-based malware detection with a deep neural network", ***Computers & Security***, 109: 102400 (2021).

26. Qiao, Y., Jiang, Q., Jiang, Z., and Gu, L., "A multi-channel visualization method for malware classification based on deep learning", ***Proceedings - 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing***

And Communications/13th IEEE International Conference On Big Data Science And Engineering, TrustCom/BigDataSE 2019, 757–762 (2019).

27. Acharya, J., Chuadhary, A., Chhabria, A., and Jangale, S., "Detecting malware, malicious URLs and virus using machine learning and signature matching", ***2021 2nd International Conference For Emerging Technology, INCET 2021***, 1–5 (2021).

28. Markel, Z. and Bilzor, M., "Building a machine learning classifier for malware detection", ***WATeR 2014 - Proceedings Of The 2014 2nd Workshop On Anti-Malware Testing Research***, (2015).

29. Singh, J. and Singh, J., "A survey on machine learning-based malware detection in executable files", ***Journal Of Systems Architecture***, 112 (July 2020): 101861 (2021).

30. Nagano, Y. and Uda, R., "Static analysis with paragraph vector for malware detection", (2017).

31. Darabian, H., Dehghantanha, A., Hashemi, S., Homayoun, S., and Choo, K. R., "An opcode-based technique for polymorphic Internet of Things malware detection", ***Concurrency And Computation: Practice And Experience***, 32 (6): e5173 (2020).

32. Lu, R., "Malware Detection with LSTM using Opcode Language", (2019).

33. Sanz, B., "On the automatic categorisation of android applications", ***In Proceedings Of The 2012 IEEE Consumer Communications And Networking Conference (CCNC), IEEE, Las Vegas, NV, USA***, .

34. Wu, Q., Zhu, X., and Liu, B., "A Survey of Android Malware Static Detection Technology Based on Machine Learning", ***Mobile Information Systems***, 2021: (2021).

35. Milosevic, N., Dehghantanha, A., and Choo, K.-K. R., "Machine learning aided Android malware classification", ***Computers & Electrical Engineering***, 61: 266–274 (2017).

36. Zhao, Y., Bo, B., Feng, Y., Xu, C., Yu, B., and Chen, J., "A Feature Extraction Method of Hybrid Gram for Malicious Behavior Based on Machine Learning", ***Security And Communication Networks***, 2019: (2019).

37. Catak, F. O., Yazi, A. F., Elezaj, O., and Ahmed, J., "Deep learning based Sequential model for malware analysis using Windows exe API Calls", ***PeerJ Computer Science***, 6 (July): 1–23 (2020).

38. Liu, Y. and Wang, Y., "A robust malware detection system using deep learning on API calls", ***Proceedings Of 2019 IEEE 3rd Information Technology,***

Networking, Electronic And Automation Control Conference, ITNEC 2019, (Itnec): 1456–1460 (2019).

39. Xiaofeng, L., Xiao, Z., Fangshuo, J., Shengwei, Y., and Jing, S., "ASSCA: API based Sequence and Statistics features Combined malware detection Architecture", ***Procedia Computer Science***, 129: 248–256 (2018).

40. Xu, Z., Fang, X., and Yang, G., "Malbert : A novel pre-training method for malware detection", ***Computers & Security***, 111: 102458 (2021).

41. Ki, Y., Kim, E., and Kim, H. K., "A novel approach to detect malware based on API call sequence analysis", ***International Journal Of Distributed Sensor Networks***, 2015: (2015).

42. Internet: Catak, F. O., "Malware API Call Dataset", **https://ieee-dataport.org/open-access/malware-api-call-dataset** (2022).

43. Ye, Y., Chen, L., Hou, S., Hardy, W., and Li, X., "DeepAM : a heterogeneous deep learning framework for intelligent malware detection", ***Knowledge And Information Systems***, 54 (2): 265–285 (2018).

44. Jindal, C., Salls, C., Aghakhani, H., Long, K., Kruegel, C., and Vigna, G., "Neurlux: Dynamic malware analysis without feature engineering", ***PervasiveHealth: Pervasive Computing Technologies For Healthcare***, 444–455 (2019).

45. Xiao, F., Lin, Z., Sun, Y., and Ma, Y., "Malware Detection Based on Deep Learning of Behavior Graphs", ***Mathematical Problems In Engineering***, 2019: (2019).

46. Oliveira, A. S. de and Jos´e Sassi, R., "Behavioral Malware Detection using Deep Graph Convolutional Neural Networks", ***International Journal Of Computer Applications***, 174 (29): 1–8 (2021).

47. Tang, M. and Qian, Q., "Dynamic API call sequence visualisation for malware classification", ***IET Information Security***, 13 (4): 367–377 (2019).

48. Aditya, W. R., Girinoto, Hadiprakoso, R. B., and Waluyo, A., "Deep Learning for Malware Classification Platform using Windows API Call Sequence", 25–29 (2022).

49. Huang, X., Ma, L., Yang, W., and Zhong, Y., "A Method for Windows Malware Detection Based on Deep Learning", ***Journal Of Signal Processing Systems***, (August 2020): 265–273 (2020).

50. Internet: Andrade, E. de O., "MC-Dataset-Multiclass", **https://figshare.com/articles/dataset/MC-dataset-multiclass/5995468/1** (2022).

51. "VirusSign", **https://www.virussign.com/** (2021).

52. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., and Venkatraman, S., "Robust Intelligent Malware Detection Using Deep Learning", ***IEEE Access***, 7: 46717–46738 (2019).

53. Jeon, J., Jeong, B., Baek, S., and Jeong, Y.-S., "Hybrid Malware Detection Based on Bi-LSTM and SPP-Net for Smart IoT", ***IEEE Transactions On Industrial Informatics***, PP (c): 1–1 (2021).

54. Darabian, H., Homayounoot, S., Dehghantanha, A., Hashemi, S., Karimipour, H., Parizi, R. M., and Choo, K. K. R., "Detecting Cryptomining Malware: a Deep Learning Approach for Static and Dynamic Analysis", ***Journal Of Grid Computing***, 18 (2): 293–303 (2020).

55. Baek, S., Jeon, J., Jeong, B., and Jeong, Y., "Two-Stage Hybrid Malware Detection Using Deep Learning", ***Human Centric Computing And Information Sciences***, 11: (2021).

56. Xu, L., Zhang, D., Jayasena, N., and Cavazos, J., "HADM: Hybrid Analysis for Detection of Malware", ***Lecture Notes In Networks And Systems***, 16: 702–724 (2018).

57. O'Shea, K. and Nash, R., "An introduction to convolutional neural networks", ***ArXiv Preprint ArXiv:1511.08458***, (2015).

58. Mostafa, S. and Wu, F.-X., "Diagnosis of autism spectrum disorder with convolutional autoencoder and structural MRI images", Neural Engineering Techniques for Autism Spectrum Disorder, ***Elsevier***, 23–38 (2021).

59. Internet: GeeksforGeeks, "CNN | Introduction to Pooling Layer", **https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/** (2022).

60. Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K., "Convolutional neural networks: an overview and application in radiology", ***Insights Into Imaging***, 9 (4): 611–629 (2018).

61. Internet: Udofia, U., "Basic Overview of Convolutional Neural Network (CNN)", **https://medium.com/dataseries/basic-overview-of-convolutional-neural-network-cnn-4fcc7dbb4f17** (2022).

62. Internet: Brownlee, J., "Difference Between a Batch and an Epoch in a Neural Network", **https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/** (2022).

63. Internet: Pere, C., "What Are Loss Functions?", **https://towardsdatascience.com/what-is-loss-function-1e2605aeb904** (2022).

64. Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition", ***ArXiv Preprint ArXiv:1409.1556***, (2014).

65. Internet: G, R., "Everything You Need to Know about VGG16", **https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918** (2022).

66. Internet: neurohive, "VGG16 - Convolutional Network for Classification and Detection", **https://neurohive.io/en/popular-networks/vgg16/** (2022).

67. Internet: Bansal, M., "Face Recognition Using Transfer Learning and VGG16", **https://medium.com/analytics-vidhya/face-recognition-using-transfer-learning-and-vgg16-cf4de57b9154** (2022).

68. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., and Bernstein, M., "Imagenet large scale visual recognition challenge", ***International Journal Of Computer Vision***, 115 (3): 211–252 (2015).

69. He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition", (2016).

70. Internet: SACHAN, A., "Detailed Guide to Understand and Implement ResNets", **https://cv-tricks.com/keras/understand-implement-resnets/** (2022).

71. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z., "Rethinking the Inception Architecture for Computer Vision", ***Proceedings Of The IEEE Computer Society Conference On Computer Vision And Pattern Recognition***, 2016-Decem: 2818–2826 (2016).

72. Internet: OpenGenus, "Inception V3 Model Architecture", **https://iq.opengenus.org/inception-v3-model-architecture/** (2022).

73. Internet: JORDAN, J., "Common Architectures in Convolutional Neural Networks.", **https://www.jeremyjordan.me/convnet-architectures/** (2022).

74. Hochreiter, S. and Schmidhuber, J., "Long short-term memory", ***Neural Computation***, 9 (8): 1735–1780 (1997).

75. Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J., "A novel connectionist system for unconstrained handwriting recognition", ***IEEE Transactions On Pattern Analysis And Machine Intelligence***, 31 (5): 855–868 (2008).

76. Li, X. and Wu, X., "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition", (2015).

77. Internet: Phi, M., "Illustrated Guide to LSTM’s and GRU’s: A Step by Step Explanation", **https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21** (2022).

78. Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y., "On the properties of neural machine translation: Encoder-decoder approaches", ***ArXiv Preprint ArXiv:1409.1259***, (2014).

79. "Https://Download.Cnet.Com/", **https://download.cnet.com/** (2022).

80. Al-Musawi, H. S. and Mohammed, A. S., "Hybrid Malware Detection and Classification in Real-Time by Deep Learning Techniques", ***Proceedings Of 2022 SAARD 184th World Conference On Applied Science Engineering And Technology, WCASET 2022, Putrajaya, Malaysia***, 43–47 (2022).

RESUME

Hussein Sadraldeen ALMUSAWI graduated from high school education at Kirkuk, Iraq. He obtained a bachelor's degree from Northern Technical University/Kirkuk/ Software Engineering Techniques in 2011. He worked as a computer teacher at an industrial secondary school for over eight years.

In 2020, he moved to Karabük, Turkey, and started his master's degree in Computer Engineering at Karabük University.