Fall 12-9-2021

ANALYZING AND DETECTING ANDROID MALWARE AND DEEPFAKE

Md Shohel Rana
University of Southern Mississippi

ANALYZING AND DETECTING ANDROID MALWARE AND DEEPFAKE

by

Md Shohel Rana

A Dissertation
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Computing Sciences and Computer Engineering
at The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

Approved by:

Dr. Andrew H. Sung, Committee Chair
Dr. Beddhu Murali
Dr. Ahmed Sherif
Dr. Parthapratim Biswas
Dr. Sungwook Lee

December 2021

Published by the Graduate School

ABSTRACT

Rapid advances in artificial intelligence (AI), machine learning (ML), and deep learning (DL) over the past several decades have produced a variety of technologies and tools that, among numerous cybersecurity issues, have enticed cybercriminals and hackers to design malware for the Android operating systems and/or manipulate multimedia. For example, high-quality and realistic fake videos, images, or audios have been created to spread misinformation and propaganda, foment political discord and hate, or even harass and blackmail people; these manipulated, high-quality and realistic videos became known recently as Deepfake. There has been much work done in recent years on malware analysis and detection in Android applications, and many solutions have been suggested to cope with the issues highlighted by Deepfake.

This dissertation addresses a couple of research topics: first, based on manifest analysis, it introduces a feature-based detection technique for detecting Android malware; second, it investigates DL and non-DL methods for detecting Deepfake videos. The first research shows that the obtained results outperform all published work in Android malware detection concerning the Drebin dataset. The outcomes of the second work demonstrate that the classical ML-based methods alone can obtain superior performance in the detection of Deepfake.

In Android malware research, we propose a substring-based feature selection (SBFS) strategy and assess using various ML algorithms to identify Android malware. In addition, we apply ensemble-based learning techniques as well as advanced ensembled techniques.

In Deepfake research, this study presents both DL and non-DL methods. For identifying Deepfakes, we propose a deep ensemble learning-based method called DeepfakeStack in a DL-based approach where an enhanced composite classifier is created by combining a set of current DL-based models. In the non-deep learning-based method, we use a traditional ML method based on conventional feature creation and feature selection approaches to train, tune and test ML classifiers.

These two pieces of research can offer a promising basis for building effective systems for detecting the two cybersecurity threats: Android (Mobile) malware and Deepfake.

ACKNOWLEDGMENTS

I would sincerely like to thank my advisor and committee chair, Professor Andrew H. Sung, for his continuous guidance, tireless support, and excellent mentorship to complete my Ph.D. at the University of Southern Mississippi successfully. I would especially like to thank Professor Beddhu Murali, who has a remarkable influence on my academic career as a Ph.D. dissertation committee member. Also, I would like to thank other committee members, including Professor Ahmed Sherif, Professor Parthapratim Biswas, and Professor Sungwook Lee. They have provided valuable suggestions and comments for the completion of my Ph.D.

Also, I would like to extend my acknowledgment to the College of Arts and Sciences, all faculties, and staff (especially Ms. Chrissy Hudson) of the School of Computing Sciences and Computer Engineering to be a student and pursue my graduate study in this institution.

It will not be enough without thanking my beloved father, Mr. Md. Solaiman Hossain Talukder, my sister Mst. Shahnaz Pervin, and my spouse Mst. Mohsina Maliha for their moral and financial support, love and care, and faith and wishes for me. Finally, I would like to extend my gratitude to all my friends for their generous help during my stay and family members who have always been on my side during my Ph.D. journey.

DEDICATION

I am dedicating this Ph.D. dissertation to my beloved mother, Mst. Farzana Begum who passed away in 2017. I always love you; I know you are always with me no matter where you are. Only you had the trust in me that I can do it and fulfill your dream. I want to extend my dedication to my daughter, Sarah Sehrish, and my sweet, better half, Mst. Mohsina Maliha who are my daily inspiration and always provide me their unwavering support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF SAMPLE PSEUDOCODES

LIST OF ABBREVIATIONS

UID	--	User ID
SBFS	--	Substring-Based Feature Selection
OHA	--	Open Handset Alliance
API	--	Application Programming Interface
DVM	--	Dalvik Virtual Machine
JVM	--	Java Virtual Machine
UI	--	User Interface
APPS	--	Applications
APK	--	Android Package
ELF	--	Executable and Linkable Format
SDK	--	Software Development Kit
AI	--	Artificial Intelligence
ML	--	Machine Learning
DL	--	Deep Learning
GAN	--	Generative Adversarial Networks
GPA	--	Grade Point Average
ID	--	Identifier
SVM	--	Support Vector Machines
LIBSVM	--	A Library for Support Vector Machines
NB	--	Naïve Bayes
DT	--	Decision Tree
RF	--	Random Forest
ERT	--	Extremely Randomized Trees
SGB	--	Stochastic Gradient Boosting
LR	--	Logistic Regression
KNN	--	K-Nearest Neighbor
KMN	--	K-Means
XGB	--	Extremely Gradient Boosting
DA	--	Discriminant Analysis

MLP	--	Multilayer Perceptron Neural Network
DBN	--	Deep Belief Neural Network
CNN	--	Convolutional Neural Network
RNN	--	Recurrent Neural Network
PPCNN	--	Patch and Pair Convolutional Neural Networks
LSTM	--	Long Short-Term Memory
CN	--	Capsule Network
TP	--	True Positive
FP	--	False Positive
TN	--	True Negative
FN	--	False Negative
AC or ACC	--	Accuracy
REC	--	Recall
P or PR or PREC	--	Precision
F1-SC	--	F1-Score
TPR	--	True Positive Rate
FPR	--	False Positive Rate
MSE	--	Mean Squared Error
EM	--	Expectation Maximization
MMD	--	Maximum Mean Discrepancy
ROC	--	Receiver Operator Characteristic
AUC	--	Area Under Curve
AUROC		Area Under ROC Curve
MADAM	--	Multi-Level Anomaly Detector for Android Malware
ANDROSCANREG	--	Android Permissions Scan Registry
PERMBC	--	Blockchain of analysis and storage of permissions
BTCBC	--	Blockchain of Bitcoin
CB-MDEE	--	Consortium Blockchain for Malware Detection and Evidence Extraction
ABD	--	Android Debug Bridge

GPS	--	Global Positioning System
XML	--	Extensible Markup Language
URL	--	Uniform Resource Locator
SHA256	--	Secure Hash Algorithm 256
Gfycat	--	GIF Format Yoker
NVIDIA	--	Graphics Card Maker
DARPA	--	Defense Advanced Research Projects Agency
HOG	--	Histogram of Oriented Gradients
HTF	--	Haralic Texture Features
CH	--	Color Histogram
HM	--	Hu Moments
VCG	--	Visual Computing Group
FF++	--	Face Forensics++
DFDC	--	Deepfake Detection Challenge
CELEB-DF	--	DeepFake Forensics
VDFD	--	Versatile Deepfake Dataset
PRNU	--	Photo Response Non-Uniformity
FSTV	--	Free Speech TV
GDWCT	--	Group-wise Deep Whitening-and-Coloring Transformation
STARGAN	--	Star Generative Adversarial Networks
ATTGAN	--	Attention-based Generative Adversarial Networks
STYLEGAN	--	Style-Based Generative Adversarial Networks
IPFS	--	Inter Planetary File System
TV	--	Total Variational
KL	--	Kullback-Leibler Divergence
JS	--	Jensen-Shannon Divergence
PI	--	Pinsker’s Inequality
SIR	--	Susceptible-Infected-Recovered
SE	--	Stacking Ensemble

WRE	--	Randomized Weighted Ensemble
DFSC or DFC	--	DeepfakeStack Classifier
DFF	--	Deepfake Feature
ANOVA	--	Analysis of Variance
INCV3	--	Inception V3
XCEPT	--	Xception Network
RES50	--	Resnet 50
VGG16	--	VGG 16 Network
MOBILE	--	Mobile Network

Section I:

Analysis and Detection of Android Malware

CHAPTER I – PREFACE

1.1 Introduction

Even though modern smartphones were popular around the year 2005, particularly BlackBerry devices, Apple released the iPhone in 2007 and subsequently an HTC-based mobile phone running Google's Android operating system in 2008. One billion Android-based smartphones were sold in 2014 alone. In a decade, smartphones were costly gadgets either for tech-knowledge early adopters or company managers, but now belonged to both types of individuals.

Android quickly becomes the unofficial paradigm to embed a complete machine in any object. This can partly be made clear by the fact that in their 32-bit models, Android runs on three main CPU architectures, x86, ARM, and MIPS, as well as 64-bit. In Android licensing systems, an attribute that contributed more to Android’s adoption in embedded systems. Google allowed any company to use Android on its products without paying anything to Google by releasing most Android under an Open-Source permit, unlike conventional, proprietary, computer-based software packs. Finally, performance is much more excellent because Android provides the possibility to access the enormous catalog of already available Android apps.

1.2 Motivation to Analyze and Detect Android Malware

As smartphones and electronic devices are being progressively more adopted, it creates an unparalleled opportunity to damage such devices by malicious software or application that are hidden in the millions of free mobile applications on app markets [1].

Having billions of active users across the globe, Android has attracted marketers, hackers, and other cybercriminals who have developed malware for a variety of purposes. This realism can easily be projected on Android's platform, allowing more and more Android and other handheld device users to run third-party apps collected from both official and alternative sources. In this sense, both system safety and network safety have become a fundamental concern for end-users and service providers alike.

Malware is malicious software that has been built to harm users who use it, modify the truthful action of the device, and gain sensitive information from users of this. The possibility of a malware infection is equally significant both in desktop and mobile devices. People who use mobile devices often aren't aware of the risks they bring, and the problems are ahead of getting worse because antivirus vendors have still not performed in the same way as personal computers, nor are mobile malware creators providing them with time to make.

There has been a lot of work done on malware analysis and detection for Android devices in the last several years. To cope with malware, Android has developed several security measures, including a unique user ID (UID) for each program, system permissions, and its distribution channel Google Play. This is all part of Android's overall security strategy. Cybercriminals are becoming more skilled at developing malicious software, which increases the difficulty of coming up with a new solution. The developers and researchers have developed alternatives to improve protection where certain ideas are proposed: analytical methods, framework, sandboxes, and security systems. It's amongst the most effective approaches to apply ML solutions for Mobile malware analysis and detection.

1.3 This Thesis

This work identifies and analyzes many of these challenges and proposes methods to create trustworthy Android malware detectors based on machine learning by developing the feature sets.

In the first part of this research, we provide related knowledge on Android system architecture in chapter 2, where chapter 3 describes about ML algorithms and their applications in detecting such malware. Efforts in relevant studies, for example, static and dynamic analysis are discussed in Chapter 4. Chapter 5 provides the dataset description. Chapter 6 presents our proposed feature selection method, and Chapter 7 presents various machine learning techniques to assess the newly generated feature and provide performance results for detecting Android malware. Chapter 8 explores potential research directions for developing malware detectors that are both extremely reliable and practical to use in real-world situations and finally, Chapter 9 concludes first part of this thesis. The significant contributions of this thesis can be summarized as follows:

- Propose a technique for detecting Android malware based on substring-based feature selection (SBFS).
- In order to evaluate this SBFS, we conduct the following:
  - ✓ Implement tree-based machine learning methods.
  - ✓ Implement Stacked Generalization using tree-based machine learning techniques.
  - ✓ Apply advanced ensemble-based learning techniques.
  - ✓ Create a Blockchain network for comparing different ML algorithms and evaluating the results.

- Provide a solid basis for detection reliable malware detection with convincing experimental results.

CHAPTER II – BACKGROUND INFORMATION

Understanding how Android and its apps operate is critical before delving into our analysis framework's finer points. We present a brief overview of architecture of Android in this chapter. In Section 2.1 and 2.2, we will begin with a high-level description of Android and its architecture. These two segments describe Android's architecture and discuss the different levels of components. Section 2.3 addresses the Android applications' building blocks with other features, including activities, services, receivers, and intents. Section 2.4 shows the essential components of the Android app and its building process. In sections 2.5 and 2.6, we describe Android security features and its permission model and protection level. A short look into malware's usage of the Android platform can be found in Section 2.7.

2.1 Android

In order to create an efficacious real-world product for improving end users' mobile experience, in 2007, Android was launched as an operating system to be used for mobile device that turned into a second cousin to the iOS. After a few years, it beats Apple and becomes the world's most popular mobile operating system. It was created by Google and subsequently adopted by the Open Handset Alliance (OHA), which aims to advance open standards, provide services, and deploy Android-based smartphones. Users can do anything to customize their devices within some manufacturers' restrictions due to Android's openness while the hardware/software is fully integrated and tightly controlled in the Apple assemblage.

2.2 Android Architecture

There are two models of permissions in Android's architecture: one is at the kernel level and prevents access to the filesystem and resources, and the other is an API exposed to the user during application installation. Figure 2.1 shows how these permission models are divided between 5 different modules of Android's architecture [2].

These components can be described as follows [3]:

- Linux kernel: Android runs on a modified version of the Linux kernel that includes a few unique features for managing device drivers, power, memory, device, resource access, etc.
- Native Libraries: Using the Libraries component, the Application Framework and Android Runtime may access a collection of native C/C++ libraries. These libraries take care of browser support, databases, fonts, and audio and video playback and recording, whereas SQLite and FreeType take care of font support.
- Android Runtime: Core libraries and the Dalvik Virtual Machine (DVM) execute Android apps in the runtime environment. DVM is similar to the Java Virtual Machine in that it is mobile-friendly and uses less memory.
- Android Framework: With the Android Framework, developers may create android apps that take use of UI (User Interface), telephony, resources, geolocation, Content Providers (data), and package managers.
- Applications: Applications like the home, contacts, settings, games, and browsers all utilize the Android framework

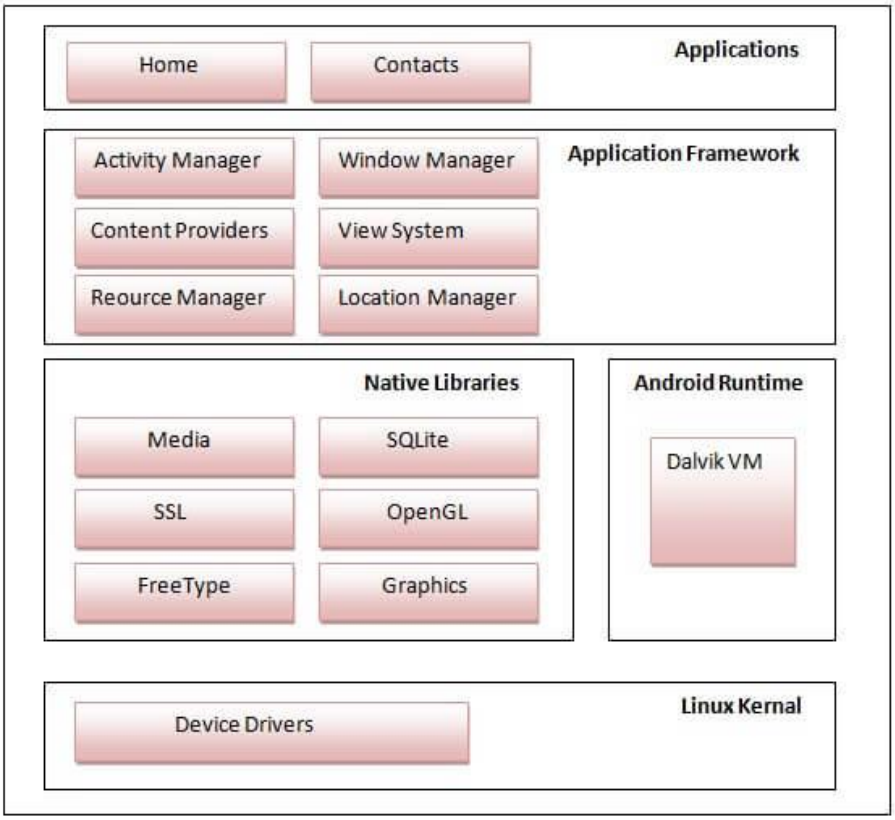


Figure 2.1 *Android Architecture.*

2.3 Core Building Blocks

There are five fundamental building blocks or components (see figure 2.2) in the Android platform. In order for the system or a user to enter the program and interact with a block or component, there is an entry point for each of the blocks [4].

1. Activities: Play as the entry points for a user to interact with an app and act as central to how a user navigates within an app or between apps.
2. Services: Perform long-running operations like the interaction between the application components, even inter-process communication in the

background without providing a user interface. Even if the user changes to another program, the service will continue operating.

3. Broadcast Receivers: While using Android, applications send and receive broadcast messages when certain events occur in the system. For example, when various system events happen in the Android system, it sends broadcast messages to the corresponding apps or other apps, such as when the system boots up or starts charging. In order to notify other apps, Android apps can also send custom broadcasts of something that they might be interested in, such as the completion of a download.
4. Content Providers: A standard interface for connecting data in one process to code running in another approach for managing access to structured data sets, encapsulating data, and providing data security mechanisms.
5. Intents: They're mostly utilized for things like starting services, launching activities, showing web pages, showing contacts, sending out broadcast messages, and dialing phone numbers.

Figure 2.2 *Android's core building blocks.*

2.4 Applications

APK (Android Package) files are unencrypted archives that include all the app's files. Figure 2.3 depicts how Java source code programs are converted to APK files using a streamlined method. [Image source Stack Overflow].

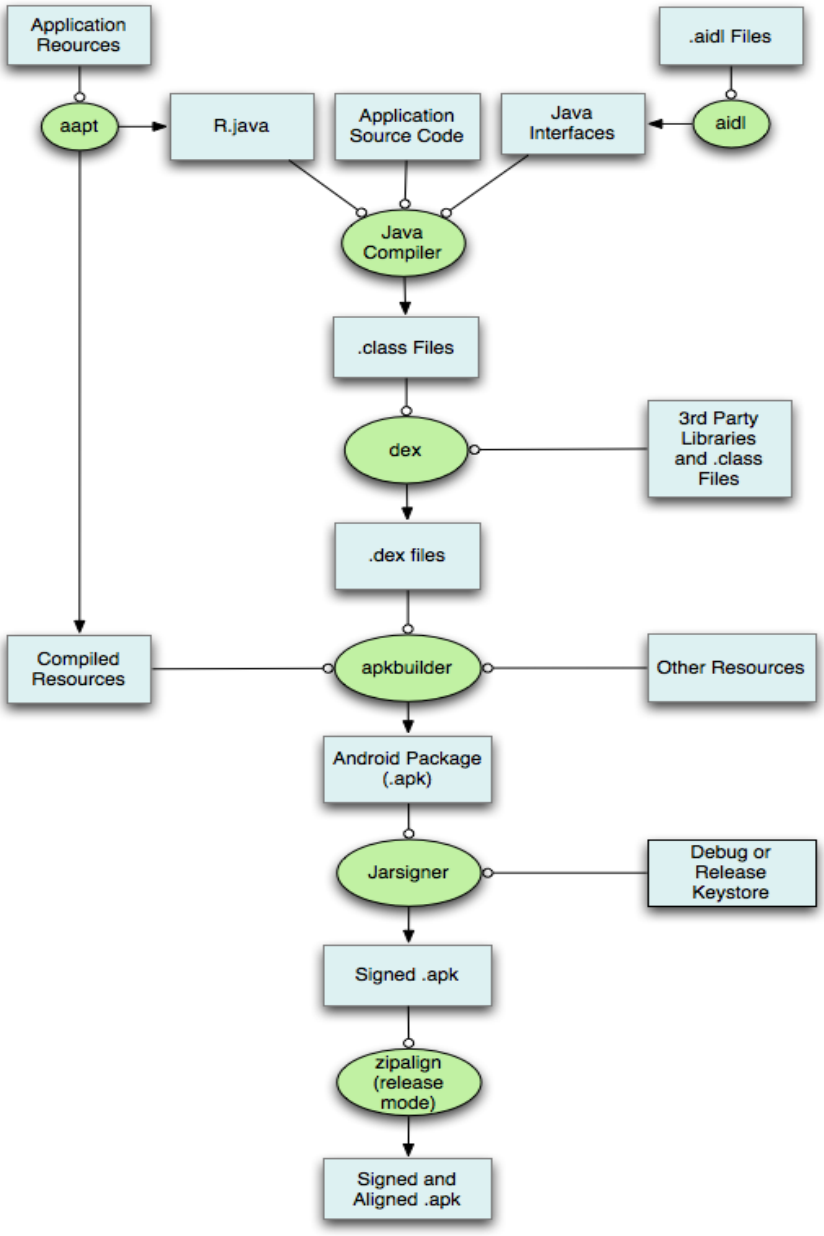


Figure 2.3 *Android applications build process.*

Figure 2.4 depicts how an Android app is put together, with the Android Manifest (.xml file), Application Code (.dex files), and Resources (.arsc files) providing information on an app's features and security configurations (e.g., permissions API, activities, etc.) [5].

1. *AndroidManifest.xml:* One of the most important files for an Android app is its manifest file, which contains all the necessary data that are required for that app to run properly and works as the bridge between the Android developer and platform by helping developers pass on functionality and requirements of the application to Android. Using the AndroidMenifest.xml file to examine how rights are being utilized once an APK has been decompiled. The Java files where the API functions are invoked in conjunction with the asset and resource files whether there is any DEX executable (ELF) image file or any code hiding image script available or not.
2. *Classes.dex*: The application logic codes that are written in Java and compiled to class files, then these class files are cross-compiled to Dalvik VM format where the Dalvik VM includes several features for performance optimization, verification, and monitoring. In simple steps, first, Java source code is compiled by the Java compiler into .class files. Then the dx (dexer) tool, part of the Android SDK, processes the .class files into a file format called DEX that contains Dalvik byte code. The dx tool removes all the redundant information that is present in the classes. And finally, in DEX, all the classes of the application are packed into one file.

3. *Resources.arsc*: The Android operating system's application resource file allows developers to access resources in a standard manner in the program's source code.

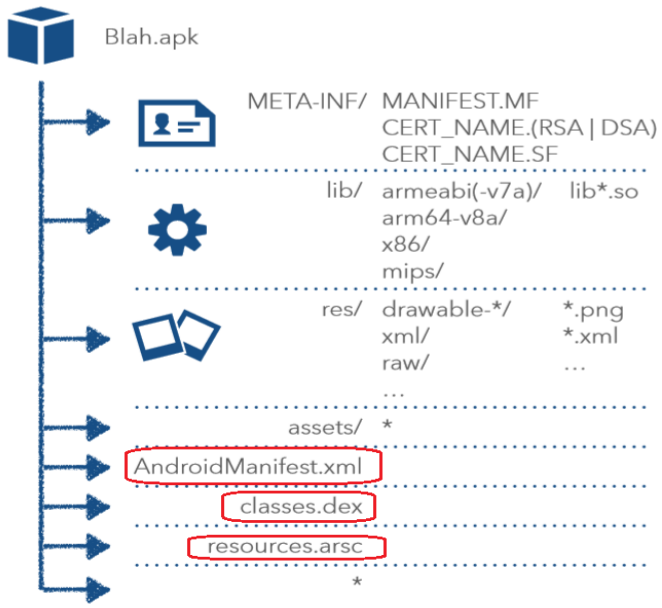


Figure 2.4 *Android APK decompile.*

2.5 Android Security Features

Protection for Android helps to secure the device and apps of users' data and devices. Application-defined and user-granted permissions are among the most important aspects of Android security because they allow developers to better control what their apps can and cannot do. The Linux kernel is one of them that is discovered in highly vulnerable settings. Because of its open ecosystem, Linux protection is frequently reinforced by the security experts and engineers who repair and patch security bugs. Linux security. It also allows the redundant and uncertain bits to be removed from the kernel. The sandboxing apps separate the operations and data of other applications in a single UID. Each app in the

kernel provides a different Linux user ID at installation time. The same app can have a new UID in other platforms because no one UID can delegate two separate applications. Each app does not have its UID because they do not have to run under their UID in the same method instead. A UID is associated with each app's data, such that data from one app cannot be accessed by another app's UID. The application-signing function allows the developer's certificate to recognize the creator of an app to sign any.apk app file. This function enables the applications to share a UID when marked with the same document. This also requires the device to authorize the authorization at the signature stage; if it is signed with the same certificate as other applications that proclaimed it, it gives the signature level approvals to a requested app. Finally, the permission model introduced by Android protects the resources and features of the device and makes them available to applications having the required privileges.

2.6 Android Permission Model and Protection Level

The permissions model is the principal definition of protection on which Android security depends. In sandboxes, Android runs apps separately. Each app runs in an inaccessible environment without having access to system components. The permissions must be provided to the app so that device services needed to run can be accessed and used. AndroidManifest.xml contains declarations of all of a developer's permissions. The device can issue these permissions at the time of implementation and cannot be changed until they are allowed unless they remove the program. You should assert permissions in the AndroidManipest.xml file (see figure 2.5) on one or more <permission> tags and must define with the necessary optional attributes. Attributes such as <label> and <description>

are the strings shown to the user while he or she installs. These features should have an explicit specification in order to help the user understand the rights suggested by authorization. Application resources and other application data are controlled by the two tags in the model permissions: <use-permission> and <permission>. According to the label <uses-permission>, the application must have permission to access certain resources, including hardware, software, and other resources on the device. Malicious apps frequently use permissions like "ACCESS_WIFI_STATE" "INTERNET" "ACCESS_NETWORK", "ACCESS_GET_TASKS", etc.

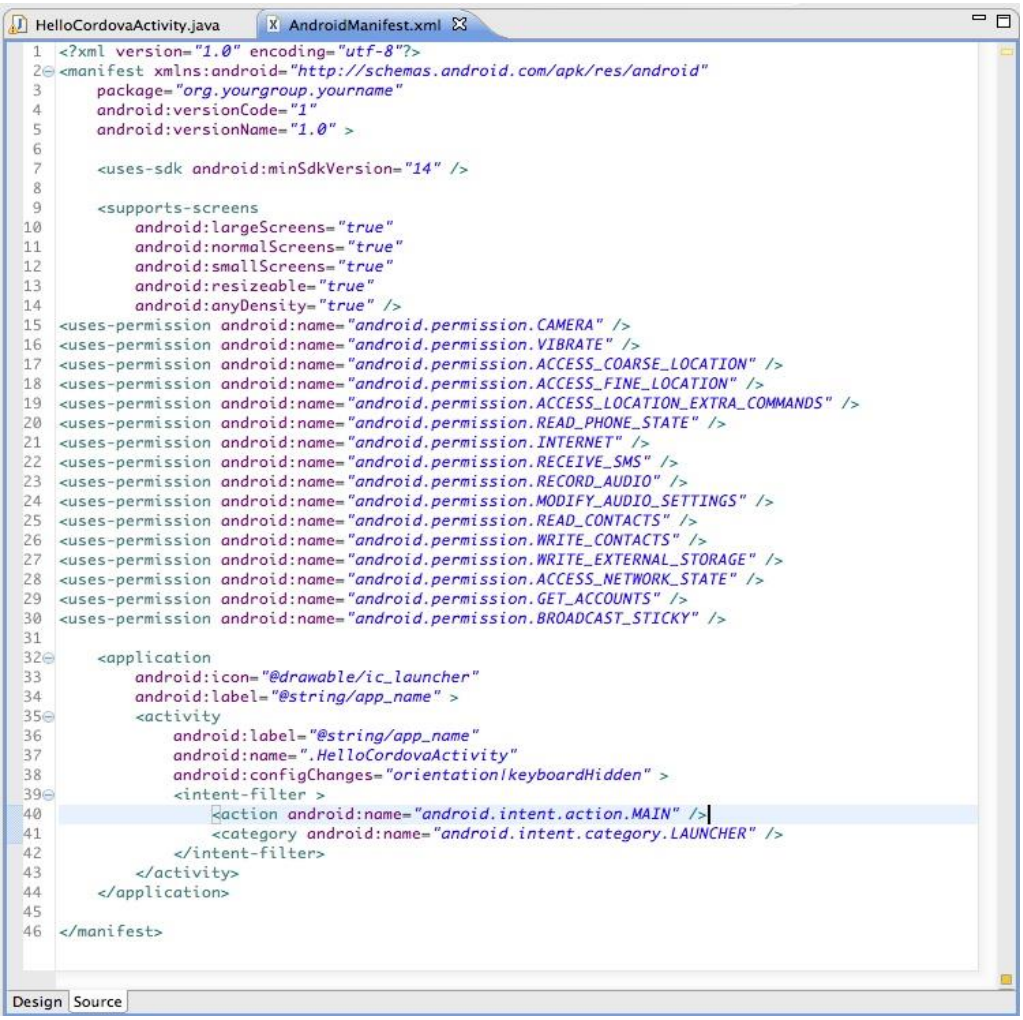


Figure 2.5 *An AndroidMenifest.xml file with used permissions.*

Some critical functionalities have also been related to malware, as seen below:

```
(TelephonyManager) context.getActivity().getSystemService("phone")).getDeviceId()
(TelephonyManager) context.getActivity().getSystemService("phone")).getLine1Number()
(ActivityManager) context.getActivity().getSystemService("activity")
List<RunningAppProcessInfo> procInfos = ((ActivityManager)
context.getActivity().getSystemService("activity")).getRunningAppProcesses();
for (int i = 0; i < procInfos.size(); i++) {
      if(((RunningAppProcessInfo)
      procInfos.get(i)).processName.equals(args[0].getAsString())){
         b = Boolean.valueOf(true);
     }
}
String[] strArr = new String[]{
   "android.permission.ACCESS_COARSE_LOCATION",
   "android.permission.ACCESS_FINE_LOCATION",
   "android.permission.ACCESS_WIFI_STATE",
   "android.permission.CHANGE_WIFI_STATE",
   "android.permission.VIBRATE",
   "android.permission.READ_CALENDAR",
   "android.permission.WRITE_CALENDAR",
   "com.google.android.gms.permission.ACTIVITY_RECOGNITION"
};
IntentFilter intentFilter = new IntentFilter("android.intent.action.PACKAGE_ADDED");
intentFilter.addAction("android.intent.action.PACKAGE_REMOVED");
intentFilter.addAction("android.intent.action.PACKAGE_REPLACED");
intentFilter.addDataScheme("package");
registerReceiver(this.f2955D, intentFilter);
intentFilter = new IntentFilter();
intentFilter.addAction("android.net.wifi.SCAN_RESULTS");
intentFilter.addAction("android.net.wifi.STATE_CHANGE");
intentFilter.addAction("android.net.wifi.WIFI_STATE_CHANGED");
intentFilter.addAction("android.net.wifi.supplicant.CONNECTION_CHANGE");
registerReceiver(this.an, intentFilter);

External links:

public static String f2920C = "http://www.shinhwa21.net/new/apps_judis_end.php?pkg=";
private static String al = "http://www.shinhwa21.net/new/apps_kakao_judis_7.php?pkg=";
```

Listing 2.1 *Malware related sensitive functions*.

More than 130 permissions are pre-installed in the Android operating system, and developers may declare additional permissions for their apps through a feature called dynamic permissions. With the Android operating system, you have four security-related permissions:

1. *Normal*: Allows the program to get access to the least sensitive device resource without informing the user beforehand.

2. *Dangerous*: Access to the device's hardware and software is permitted by a permission with a high degree of risk for apps. This type of permission is prompted on the screen and seeks user permission during installation. In order to continue the installation process, a user has to provide his/her authorization by understanding and accepting its consequences.
3. *Signature:* Authorization from the system only if the app is seeking authorization from a different application and all applications have the same license. The device immediately issues this authorization without notifying the user whether all applications have the same credential.
4. *Signature or system*: A device can provide this type of permission only if the applications have been signed-in Android system images. Without noticing the user, the device can automatically issue this type of permission.

2.7 Malware Threats and Types of Attack

Malware is malicious software built to harm users who use it, modify the truthful action of the device, and gain sensitive information from the device. There are many kinds of malware threats that have been discovered in Android apps, as described by Felt et al. [6] and Spreitzenbarth [7].

- ✓ **Trojan horse**: A Trojan horse is a kind of malware that masquerades as reputable third-party applications.
- ✓ **Spyware:** Gather data from a computer or other device and send it to a third party without the consent of the use.

- ✓ **Worms:** A malware computer program that replicates itself to spread to other computers, uses a computer network to spread itself, relying on security failures on the target computer to access it.
- ✓ **Rootkit:** One of the primary purposes of a root-kit software is for hackers to gain complete control over the machine they are attacking. The device may be controlled from another location using this method of remote management.
- ✓ **Ransomware:** Data on a victim's computer is encrypted and locked by ransomware, which demands payment before it can be unlocked, and access restored to the victim.

The following are some examples of attack types:

- *Hardware-based attacks*: Attackers employ particular instructions or procedures to cause hardware to malfunction or alter its properties.
- *Software-based attacks*: Malware is injected into legitimate official applications, which are then uploaded to unofficial Android app marketplaces and downloaded by irresponsible Android users.
- *Firmware-based attacks*: To implant malware, attackers manipulate or replace the devices' firmware by gaining administrative privileges or by building back doors.

CHAPTER III – MACHINE LEARNING AND MALWARE DETECTION

Due to the progress of modern computing technology, today's machine learning is different from the earlier version. The idea was that machines would learn without being scheduled into particular duties, and the artificial intelligence researchers wanted to see how machines could learn from data. The iterative nature of machine learning is essential as it can adapt independently to new data. They learn by training themselves from previous calculations to yield accurate and reliable decisions and outcomes that can be repeated [8]. This chapter describes machine learning and how machine learning helps in analyzing and detecting Android malware.

3.1 Machine Learning

Machine learning (ML) is a data processing approach that automates the creation of the analysis model. It is a subset of Artificial Intelligence that allows systems to learn from data, find patterns in data, and decide with minimal human interference [8]. The widely used area of ML is the classification problem, where the problem is about the finding of which class is a given object. Another potential usage of the ML technique deals with regression problems such as predicting an actual value instead of predicting class. For example, a student's GPA, house price, a patient survival score (i.e., days, months, years, etc.) after a major operation.

Many fields have a considerable number of problems that can be turned into classification or regression problems. In order to deal with these problems, ML techniques are applied, such as the automatic detection of oil spills with Satellite images [9]-[10],

detection of prostate cancer [11], forecasting river water quality for agriculture [12], or forecasting trend reversals in financial markets [13].

3.1.1 General Process

Two prominent ML techniques.

- *Supervised learning*: The aim of supervised learning is to train a computer to predict the class of objects it has never seen before.
- *Unsupervised learning*: Finding clusters of related items is a common aim in unsupervised learning.

Although unsupervised learning is the most effective method in malware detection, we concentrate on supervised techniques in this dissertation.

3.1.1.1 Feature Set

Finding the optimal feature set that works well with the right application is the key to learning the machine quickly and efficiently. A first step in using an ML-based technique to deal with any classification or regression problem is preparing a proper feature set and collecting a list of such features instead of objects. One of the many ML functions is the detection of correlations that we knew were not there. Researchers usually do their hardest to add features they know. If we look at anything as sophisticated as Android applications, the number of potential features and functions is almost infinite.

In the experiments described in Chapter 4, we utilized three distinct feature sets, which are defined in this research.

3.1.1.2 Feature Extraction and Selection

To generate feature vectors, certain features must be extracted before choosing them. Feature extraction and selection are major issues in machine learning. To improve the model's performance, it is necessary to have accurate or discriminating features since characteristics that are irrelevant would result in undesired computation and accuracy loss. To achieve better classification accuracy, you'll need a larger training set with more data points. Accuracy rises to a point for a given amount of data, then classification accuracy decreases as the number of characteristics grows. It's important to reduce the input data dimension when selecting features since it helps with learning (inducing) accurate classifiers and identifying the most relevant features. The training process is simplified thanks to feature selection [14].

Table 3.1 shows the feature matrix, in which two columns are incorporated as is usual in ML.

Table 3.1 *A representation of feature matrix*

	Feat_1	**Feat_2**	**Feat_3**	**….**	**Feat_n**	**Class**
App_1	True	3	0.5	…	Network	Malware
App_2	False	1	1.8	…	Wifi	Goodware
.	.	.	.	…	.	.
.	.	.	.	…	.	.
.	.	.	.	…	.	.
App_m	True	8	0.07	…	Activity	Malware

An **identifier (ID)** sample allows anyone to know which application a feature vector stands for and a **class** that is derived from the classification of references. The first column of our example table is an identifier, and the last column stands for class (i.e.,

Malware, Goodware). Table 3.1 shows different types of features, including Boolean, integer, absolute value, or categorical.

3.1.1.3 Learning and Applying Model

After successfully generating a feature matrix, it is fed to an ML algorithm for creating a model that classifies unknown data as Goodware and malware. This step is the learning part of a method in which the algorithm attempts to draw rules from example data. A train set is considered a feature matrix that is used to train an ML algorithm. There are many ML algorithms available. In this thesis, we have used tree-based ML methods, including, Decision Tree (DT), Random Forest (RF), Extremely Randomized Trees (ERT), Stochastic Gradient Boosting (SGB).

After building a learning model, it is applied to various applications. A class predictor for each test case is built in this phase. The prediction may be either a class name or an actual number that usually represents a probability of one class.

3.2 Measurement Metrics

In order to validate the performance of classification algorithms, various measurement metrics are used during experimentations. Similarly, we address different measurement metrics that are used to evaluate our experiments. Although these are important for any classification challenge, we use malware detection to use malware-specific words instead of generalized terminology commonly used in ML literature.

*Confusion Matrix***:** Confusion matrix provides information regarding actual and predicted classifications to evaluate the algorithm's performance based on the matrices' data [15], as shown in Table 3.2.

Table 3.2 *Confusion matrix*

		Actual Class	
		Positive	**Negative**
Predictive Class	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Accuracy (AC) is the percentage of accurate predictions out of the total number of predictions made by the classifier.

$$accuracy\ (AC) = \frac{No.of\ correctly\ classified\ data\ (TP + TN)}{Total}$$

Percentage of properly predicted positive instances is known as **precision (P)**, and can be defined by

$$precision\ (P) = \frac{TP}{TP\ +\ FP}$$

When it comes to the **recall or true positive rate (TPR)**, the percentage of properly classified positive instances that can be defined by

$$recall = \frac{TP}{TP\ +\ FN}$$

Negative instances labeled erroneously as positive are known as the **False Positive rate (FPR)**, which is the percentage of such cases and defined by

$$false\ positive\ rate = \frac{FP}{TP\ +\ FP}$$

f1- Score or F-Measure is a weighted average of the True Positive (TP) rate or recall and Precision (P) defined by

$$F = \frac{(\beta^2 + 1) * P * TP}{\beta^2 * P + TP}$$

Where β is used to controlling the weight assigned to TP and P.

True Positive (TP) Rate in the Y-Axis against False Positive (FP) Rate in the X-Axis are shown on a **ROC Curve**, which summarizes the classifier's overall predicted performance.

For classifiers, the **PR Curve** represents how well they work by showing how many relevant instances they obtain and how many instances they return.

3.3 Malware Detection Specificities

Though ML methods are used in various fields, the topic of malware detection has many fundamental problems. Considering these problems is necessary because they may impose limitations on the applications of machine learning.

3.3.1 Scarcity of Ground Truth

All researchers in the malware detection community face a similar problem: in the absence of a precise and efficient Ground Truth, nothing can be said to be certain. Although some binaries are considered to be malware, such truthful information is very uncommon. Just a few hundred have been shown as malware in published works by an expert on the

several million Android Apps bundles. There are even fewer recorded cases demonstrating that a particular program is free of malware.

That's one reason why creating automatic malware detectors is the target of various worldwide research teams, but it also causes problems for the same groups when assessing their malware detectors. Furthermore, manually analyzing the program is an extensive process requiring highly qualified experts and is thus impractical for large-scale research in general.

This challenge persists in the entire market, making more than $4 billion per year [16]. Antivirus vendors say that they are aware of the application and know about which of them is malware and which one not. There is minimal agreement among various antivirus products, allowing the creation of a reference classification based on the expertise of those who say that they have a difficult challenge. As a result, the phrase "Ground Truth" may be acceptable in place of a more commonly used classification system.

3.3.2 Precision Vs. Recall Trade-off

Not all errors are comparably unsatisfactory concerning malware detection. For instance, the failure to detect and block malware may result in terrible consequences in a hypothetical high-quality security perspective, whereas erroneously blocking a few non-malwares might be considered to be cost-effective. However, in some contexts, possessing a few malwares might not be regarded as a highly significant case, but it may have undesirable financial implications if it is stopped from releasing any non-malware software, such as a spreadsheet.

Sometimes a trade-off between accuracy and recall is required primarily for ML-based methods. To be precise, the willingness to capture all malware raises the possibility of treating a benign application as malware. Equally, it increases the chance of allowing the actual malware to block an application that might not be blocked.

CHAPTER IV – RELATED WORK

For the same situation, research is required since not all ML methods are equally effective at classifying apps as malicious or benign (see Figure 4.1). The use of ML algorithms for detecting malware has been extensively studied in the literature. In this chapter, some studies on Android malware detection are conducted and focused on two analyses of malware detection: static analysis and dynamic analysis. Also, some Blockchain-based studies are included in this survey. Using static analysis, you may detect harmful activity in source code, data or binary files without having to execute the program directly. Cybercriminals' expertise developing apps has increased its complexity, and it has been shown that obfuscation methods can evade it. To put it another way, in dynamic analysis, we study malware behavior while it is operating by simulating user actions, such as establishing a network socket or using a web browser. Dynamic analysis helps to identify malware in sandbox settings and prevent its malicious activity.

Over the last several months, researchers have been working nonstop on trying to find a solution to Android malware's issues. It's possible to test the behavior of apps during runtime using “TaintDroid” [17], “DroidRanger” [18], or “DroidScope” [19]. “Kirin” [20], “Stowaway” [21], and “Risk-Ranker” [22] have all suggested static analysis techniques for detecting harmful activity in source code, data, or binary files.

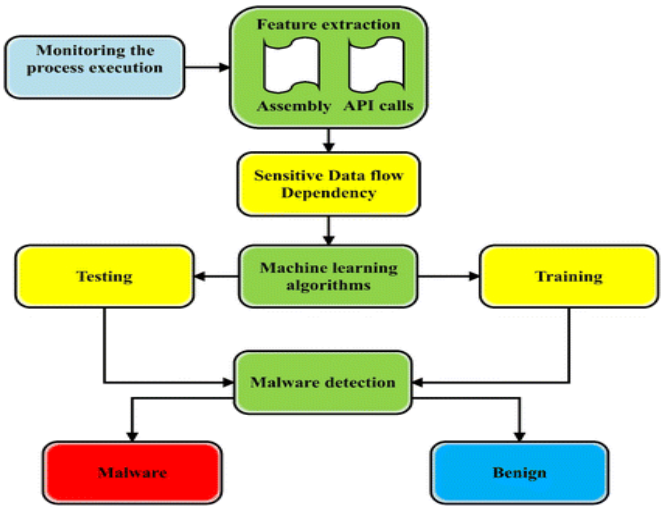


Figure 4.1 *A basic structure of ML techniques in malware detection.*

4.1 Static Analysis

A permissions-based detection method, APK Auditor, was suggested by Talha et al. [23] to classify Android applications as benign or malicious. There are three components to APK Auditor: (1) the signature database for storing retrieved information for the application, (2) the Android client for the application review queries, which is used by the end-users, and (3) the central server for setting up a communication between the signature database and smartphone clients in order to manage the entire analytical process. Method yielded an accuracy of 88% and a specificity of 0.925 by analyzing 8,762 programs, of which 1,853 were benign and 6,909 were malicious.

Fan et al. [24] offered a classification method, FamDroid, focused on static analysis. They extracted permissions, hardware components, app components, and intent filters from the AndroidManifest.xml and smali files and produced a two-dimensional feature vector by combining different features with evaluating the importance using the Random Forest algorithm. In the training phase of their experimentation, they used an

improved K-means algorithm to cluster the training samples and finally designed an adaptive weighted ensemble classifier for Android malware family classification. Based on the Drebin dataset, FamDroid can correctly classify 98.92% of malware samples into their families and achieve 99.12% F1-Score.

When analyzing [25], the authors used static analysis to extract malicious program logic from well-known Android malware, which they then turned into a set of threat patterns. It then detected malicious Android apps and classified the Android malware family depending on whether they had matching trends of threats. The proposed method achieved a 95.3% detection rate, with only a 0.4% false-positive rate, and able to classify malicious apps with having an accuracy of 92%.

A feature-based approach based on static analysis was developed by Wu et al. [26] to detect Android malware. Android applications' behavior may be characterized by examining at static information like as permissions, deployment of components, sending of Intent messages, and API calls in the code. To trace permission-related API calls, this method uses information from the application manifest file (e.g., permission requests, Intent messages passing, etc.) and components (Activity, Service, Receiver). Finally, the kNN method is used to categorize the program as either benign or malicious, with an accuracy of 97.87%.

Using ML methods, Sanz et al. [27] developed a static analysis-based method for classifying Android applications. Researchers' proposed technique would gather data on the frequency with which printed strings appear, as well as rights granted to individual apps and permissions obtained from Android Market listings for such apps. This random forest algorithm has an AUC score of 90%.

An Android malware detection system known as MADAM (Multi-Level Anomaly Detector for Android Malware) was presented in [28] as a new, multi-level and behavior-based approach for Android. This tool provides a check on system activities as well as user activity and running applications, which may help identify problems in an app and look for five different types of features at four different abstract levels: the kernel, the program, the user, and the package. One study found that, in a sample of 2,784 malicious applications compared to 9,804 benign ones, the detection accuracy was 95.9%.

Text mining and information retrieval techniques are used in Dendroid, a system developed by Tangil et al. [29] to address Android malware detection issues based on static analysis. The Dendroid reformulates the text mining modeling method by modifying the conventional Vector Space Model to automatically evaluate malware sample similarity and categorize them into families.

RevealDroid, a new ML-based method described by the authors in [30], detects Android malware and classifies the malware family using ML algorithms. The RevealDroid specifies a collection of characteristics for detection and family identification that allow obfuscation resilience, efficiency, and accuracy. RevealDroid can detect and identify harmful applications with a 93% accuracy rate for untransformed apps and an 87% accuracy rate for transformed apps. Data from 2,593 good applications and 9,054 malicious apps from two separate malware repositories is used to assess RevealDroid's performance. According to the researchers, the RevealDroid has a 14-60% better categorization rate than the Dendroid.

Aafer et al. [31] proposed a strategy for overcoming deficiencies in permission-based alert systems and building a solid and lightweight Android malware detection

classifier. The authors classified malware and benign applications based on API level data in bytecode. The KNN method yields the greatest accuracy (99%) and TPR (97.8%) while providing significant semantics about app behavior.

In [32], Zhang et al. proposed a semantic method for classifying Android malware by dependency graphs. The proposed technique extracts a weighted contextual API dependency network as program semantics and adds graphical similarity metrics to quickly identify essential app actions for building a feature set. The experimental data shows that the suggested approach has a 93% of accuracy rate and can identify zero-day malware with a false-negative rate of 2% and a false-positive rate of 5.15%.

4.2 Dynamic Analysis

Dash and al. [33] introduced ML-based approaches to classify Android malware into families using system calls that were detected during dynamic analysis. The authors created features in various levels using system calls, decoded binding communications, and abstracted behavior patterns. This collected data is then fed into a multi-class classifier (i.e., vector machine). After training a classifier with a train set labeling with malware family names, it can easily categorize malware into families using a test set. This technique improves detection accuracy by up to 93% when applied to the Drebin dataset.

Using ML algorithms by focusing on dynamic analysis, DroidDolphin [34] can find malicious apps on Android phones and tablets. By collecting data from API calls and 13 actions, it was able to conduct analysis with an F-score of 0.875 using SVM and the LIBSVM package and reach an accuracy of 86.1%.

Anti-malware tool developers and researchers may use OmniDroid [35] to help enhance or create new processes and tools for Android malware detection utilizing an automated framework for dynamic and static analysis of Android apps. In this dataset, AndroPyTool was used to build a set of ensemble classifiers, and a method based on the integration of static and dynamic characteristics via the use of ensemble classifiers was suggested for malware detection.

EnDroid is a dynamic analysis-based framework introduced in [36] with the goal of implementing precise malware detection based on a variety of dynamic behavior characteristics. Malicious activities including stealing personal information, subscribing to a premium service, and communicating with a malicious service are all covered by these characteristics. The feature selection method used by EnDroid removes unnecessary or noisy information and extracts important behavioral characteristics.

ICCDetector is a novel malware detection technique presented by Xu et al. [37] that generates a detection model after training using a set of benign applications and a set of malwares, and then uses the learned model to identify malware. With 5,264 malicious applications and 12,026 benign ones, ICCDetector's performance is compared. In comparison with the Peng et al. [38] benchmark, the ICCDetector obtains an accuracy of 97.4%, approximately 10% better than the benchmark, with a reduced false-positive rate of 0.67%, which is just about half of the benchmark's accuracy of 88%.

4.3 Blockchain-Based Solutions

Android malware was detected with the use of Blockchain technology by Gu et al. [39], who built a framework called CB-MMDE. Using a deep belief neural network (DBN)

as the detection engine, Raje et al. presented a decentralized security system built using Blockchain technology to classify Portable Executable (PE) files as dangerous or benign in [40]. Ten thousand files from a dataset are used to train a DBN network to categorize grayscale pictures into two groups.

Ouaguid et al. [41] presented a novel framework ANDROSCANREG (Android Permissions Scan Registry), by extracting and analyzing the requested permissions in an Android application via a decentralized and distributed system based on the developing technology known as Blockchain. ANDROSCANREG is made up of two Blockchains: I PERMBC, which handles the analysis, validation, and preparation of the raw findings for the second Blockchain; (ii) BTCBC, which saves the permissions history of each version of the apps being scanned through financial transactions. The new approach proposes a new way to look at Android apps. It allows malicious modules to be downloaded and placed in the device file to load, delete or encrypt the victim's personal data depending on administrator rights.

There was a bio-inspired ML technique proposed in [42] to discover root exploits by analyzing three kinds of characteristics: system command, directory path, and code-based features, together with the unique Android debug bridge (ADB). This approach suggested the use of 4 distinct boosting algorithms for classification, including AdaBoost, real AdaBoost, logit boost, and multiboot, and created a system known as RODS for it.

It was suggested by Moubarak [43] to investigate the possibility of developing untraceable malware by using Blockchain technology. As part of this proposal, a 4-ary virus was created and evaluated in real time, with each piece of code interacting with the bitcoin network to verify and confirm if it came from malicious software.

CHAPTER V – PROPOSED METHODOLOGY

Malware detection is still an open research problem because malware writers constantly update newer malware variants using different obfuscation techniques to evade existing detection methods. The problem of Android malware detection has been studied extensively by utilizing ML methods and Big Data technologies in recent years. However, these innovative approaches necessitate a considerable number of features for detecting and classifying malware correctly, and making that feature extraction and selection, training, testing take a considerable amount of time; even more, it has been unexplored which are the most critical features for accomplishing the correct classification [44]. So, it is now a hot issue and a big challenge how to detect Android malware at a highly accurate rate.

Feature extraction and selection are major issues in machine learning. To improve the model's performance, it is necessary to have accurate or discriminating features since characteristics that are irrelevant would result in undesired computation and accuracy loss. To achieve better classification accuracy, you'll need a larger training set with more data points. Classification accuracy tends to decrease as the number of features in a fixed data set grows, while for a variable data set, accuracy tends to rise. Features are selected to address two key problems: to make learning (inducing) correct classifiers easier and to discover the most suitable features. Features selection reduces the input data dimension. Selection of features lowers the complexity of the training process and, as a result, saves time [45].

5.1 DREBIN Dataset

The DREBIN [46] dataset, which contains 123,453 apps from various domains and 5,560 current malwares, was utilized to construct our suggested dataset. Each of the 5,560 recent malwares belongs to one of 179 malware families. DREBIN describes all the application characteristics for each application in the dataset, which is divided up by use case a total of eight variables drawn from two distinct sources are included in the dataset.

i. Feature sets from the manifest:

- *Feature 1 (Hardware components)*: The list of desired hardware components includes things like touchscreens, cameras, and GPS.
- *Feature 2 (Requested permissions)*: In the security mechanism that users enable during program installation, Android permissions play a crucial role. An increasing number of malicious applications are asking for permission that may give them access to sensitive data.
- *Feature 3 (App components)*: There are four kinds of app components: activities, services, content providers, and broadcast receivers.
- *Feature 4 (Filtered intents)*: Android's interprocess and intraprocess communication relies on intents. “BOOT COMPLETED”, for example, is commonly exploited by malware to perform malicious action accordingly as the Android phone is rebooted.

ii. Feature sets from disassembled code:

- *Feature 5 (Restricted API calls)*: Restrictions on the use of APIs are imposed during the installation of Android. However, the usage of

Android's restricted API function calls without asking permission in the manifest indicates malicious activity may be using root vulnerabilities.

- *Feature 6 (Used permissions)*: Non-malicious API function calls may sometimes need your consent. This feature set allows us to check if the API calls and permissions requested are being used for nefarious purposes.
- *Feature 7 (Suspicious API calls)*: Malware may obtain sensitive information about the device by using API functions, which are then obfuscated. As an example, consider the functions getDeviceId().
- *Feature 8 (Network addresses)*: Most malware sends and receives data by using network addresses or URLs.

Between August 2010 and October 2012, researchers collected samples. Table 5.1 lists the top 20 malware families.

Table 5.1 *Top malware families of DREBIN dataset*

Malware family	**# Entries**	**Malware family**	**# Entries**
FakeInstaller	925	Adrd	91
DroidKungFu	667	DroidDream	81
Plankton	625	ExploitLinuxL otoor	70
Opfake	613	Glodream	69
Ginmaster	339	MobileTx	69
BaseBridge	330	FakeRun	61
Iconosys	152	SendPay	59
Kwin	147	Gappusin	58
FakeDoc	132	Imlog	43
Geinimi	92	SMSreg	41

5.2 Substring Based Feature Selection (SBFS)

The proposed SBFS [47] consists of three phases, namely, data collection and balancing, feature selection and extraction, and machine learning classifier. The method is illustrated in figure 5.1.

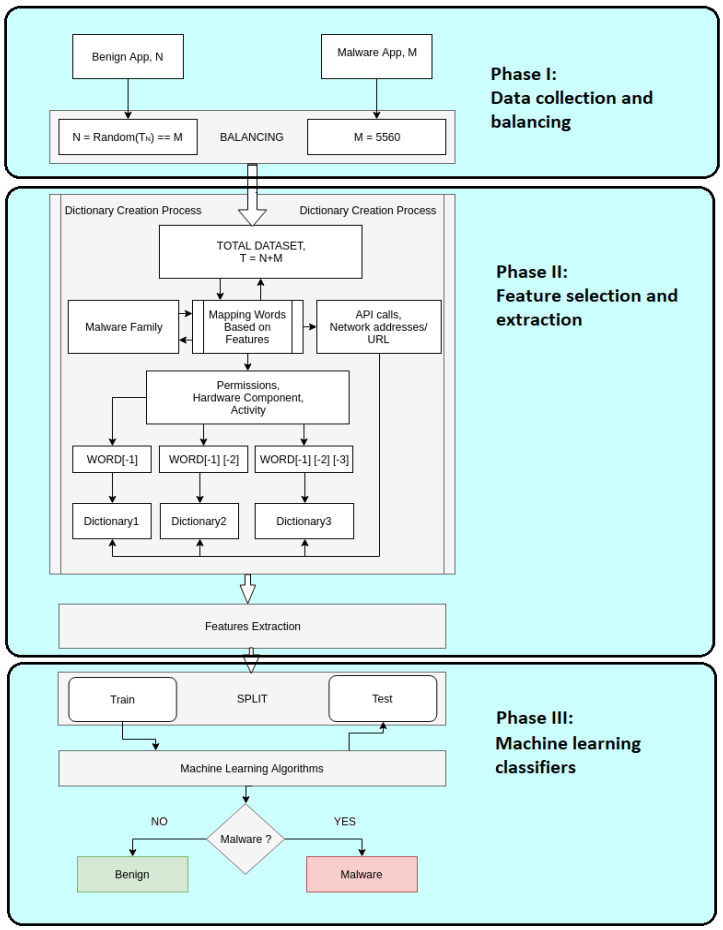


Figure 5.1 *Overview of malware detection techniques.*

We compose a balanced dataset for training and testing from the DREBIN dataset in the data balancing phase.

In Phase II, we perform feature selection and extraction. As the DREBIN dataset focuses on the manifest XML file and the disassembled (.DEX) code of the Android apps, we define some functions based on features that aim to extract the most helpful information

to ease detection. Every function receives each line of the files based on the category and returns a sequence of words stored in an array called a Substring array. Then, a feature vector is created by mapping words to an index of the Substring array. And finally, we define a function that creates a Dictionary using these words of files. For the Dictionary, all the words in each file are being used for each feature category. This same function is also used to extract features, but a vector of zeros is made with the length of the Dictionary, and its value is set to one only in the position of the words that are in the file. The feature set holds a collection of words; some features, like API calls, URLs, etc., can each be counted as a single word. But for the features like permissions, activities, intents, and services, they have multiple sequences of words (i.e., *Android.hardware.telephony*). So, we create three substrings using the last 1, 2, and 3 words, respectively, as meaningful information to find the most important word in any permissions, activities, intents, services, etc.

Phase III aims to obtain the best performance using various ML algorithms and compare them with the best available results discussed in sections 5.3, 5.4, 5.5, 5.6.

5.3 Evaluation of Tree-Based ML Classifiers for Android Malware Detection

Using static analysis on the DREBIN dataset, we test SBFS against tree-based ML methods: Decision Tree (DT) [48], Random Forest (RF) [49], Extremely Randomized Tree (ERT) [50], and Gradient Tree Boosting (GB) [51]. The SBFS technique facilitates in weeding out potentially unnecessary data while also expediting detection. There are a variety of characteristics used to evaluate the output of these tree-based classifiers. These include API call, URL, service receiver and permissions such as call, intent and actual

permission. It is necessary to do a static analysis classifier before utilizing the SBFS technique to identify malware in Android OS apps before using the method [52].

5.3.1 Results and Discussion

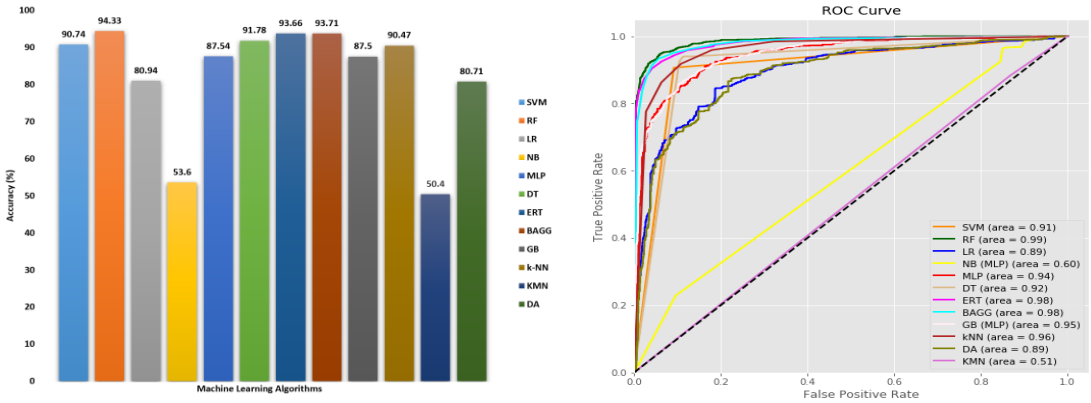
A summary of the findings may be found in Table 5.2.

Table 5.2 *Performance of ML algorithms on DREBIN dataset*

Algorithms	Precision		Recall		f1-score		Accuracy
	0	1	0		0	1	
SVM	0.91	0.91	0.91	0.91	0.91	0.91	90.74
RF	**0.95**	**0.94**	**0.94**	**0.95**	**0.94**	**0.94**	**94.33**
LR	0.78	0.84	0.86	0.75	0.82	0.80	80.94
NB	0.68	0.52	0.15	0.92	0.25	0.66	53.60
MLP	0.85	0.91	0.92	0.83	0.88	0.87	87.54
DT	0.93	0.88	0.88	0.94	0.91	0.91	91.78
ERT	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**0.93**	**93.66**
GB	0.87	0.88	0.89	0.86	0.88	0.87	87.50
k-NN	0.92	0.89	0.89	0.92	0.90	0.91	90.47
KMN	0.54	0.50	0.13	0.88	0.21	0.64	50.40
DA	0.76	0.88	0.91	0.70	0.83	0.78	80.71

We also see that Naïve Bayes (NB) and K-Means (KMN) classifiers have exceptionally poor performance in detection. NB performs well when we have multiple classes and for text classification, because it's simple, and if the conditional independence assumption holds, an NB classifier will converge quicker than discriminative models like logistic regression (LR), so we need less training data and less time for training the model even if the NB assumption does not hold. The KMN classifier works better for clustering problems. According to our experiment, the main difference of performances between NB and RF is their model size while NB model size is small and relatively constant concerning

the data, and NB models cannot be complex behavior so that it won't get into overfitting. On the other hand, ensembled-based learning (e.g., RF, ERT, etc.) model size is very large, and if not carefully built, it results in overfitting. So, when the data is dynamic and keeps changing, NB can adapt quickly to the changes and new data while using an RF that would have to rebuild the forest every time something changes. The overall accuracy of each ML algorithm using the same parameters can be summarized by using Figure. 5.2, where it has been seen that RF obtains the best performance using the features of Hardware Components, Requested Permissions, App Components, Filtered Intents, Restricted API Calls, Used Permissions, Suspicious API Calls, and Network Address. This produces the Random Forest's overall accuracy of 94.33%, TPR of 94.27%, FPR of 5.88%, and AUC of 99.21%



a. Accuracy b. ROC Curve

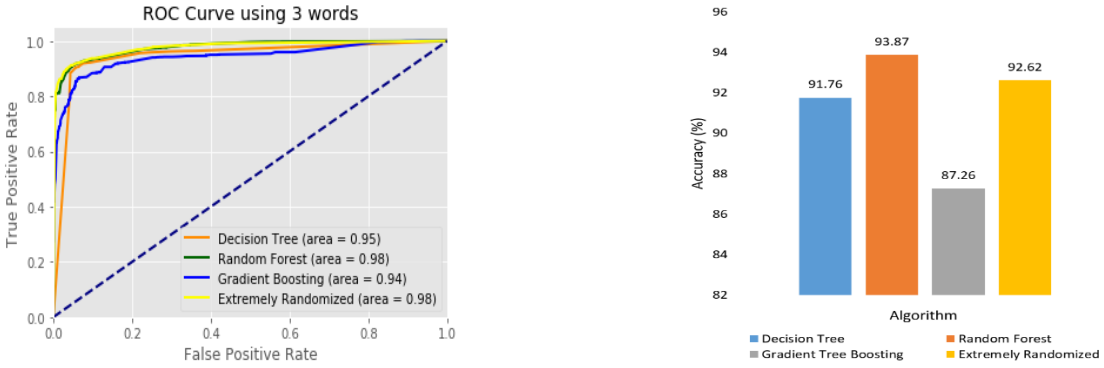
Figure 5.2 *Performance of ML algorithms.*

It's easy to see in Table 5.3 how tree-based ML algorithms perform on feature sets that are created using SBFS technique.

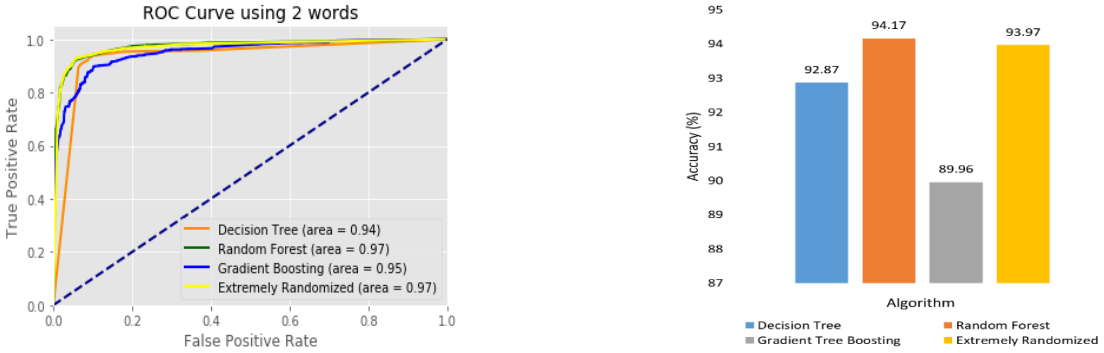
Table 5.3 *Performance of Tree-based ML algorithms on DREBIN dataset*

Substring	Algorithms	Precision		Recall		f1-score		Accuracy
		0	1	0	1	0	1	
3 words from each feature	DT	0.95	0.89	0.89	0.95	0.92	0.92	91.76
	RF	0.95	0.92	0.93	0.95	0.94	0.95	93.87
	GB	0.86	0.89	0.90	0.84	0.88	0.87	87.26
	ERT	0.95	0.91	0.91	0.95	0.93	0.93	92.62
2 words from each feature	DT	0.93	0.93	0.92	0.93	0.93	0.93	92.87
	RF	0.93	0.95	0.95	0.94	0.94	0.94	94.17
	GB	0.87	0.91	0.91	0.87	0.89	0.89	89.96
	ERT	0.94	0.94	0.94	0.94	0.94	0.94	93.97
1 word from each feature	DT	0.96	0.96	0.96	0.96	0.96	0.96	96.13
	RF	0.97	0.98	0.98	0.97	0.97	0.97	97.24
	GB	0.93	0.94	0.94	0.93	0.94	0.94	93.68
	ERT	0.97	0.97	0.97	0.97	0.97	0.97	96.97

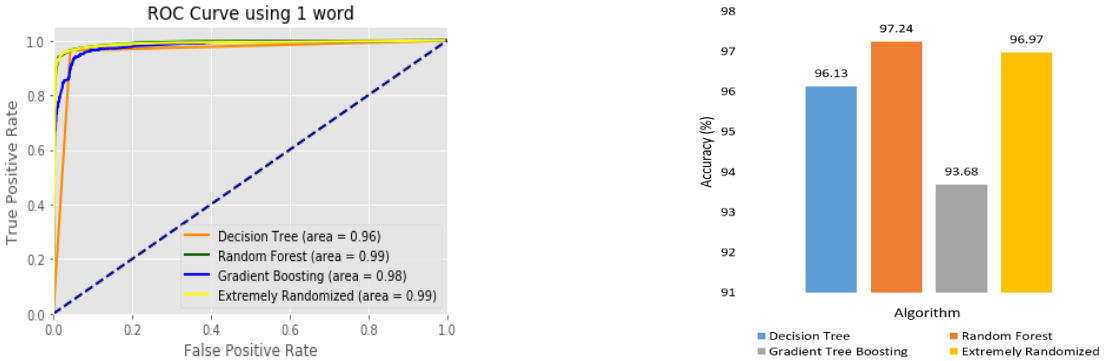
Finally, we can observe that the RF classifier consistently outperforms the others. The last words of any permissions, activities, intents, and services are usually the most significant, whereas the combination of other words may have a detrimental effect on categorization; for example, substrings formed by combining the last 2 or 3 words frequently provide unnecessary information to classifiers. Setting up the same settings allows us to compare the overall performance of each tree-based ML method, and the results are shown in Figure 5.3. Based on these findings, it seems that RF uses the features of Hardware Components, Requested Permission, App Component, Filtered Intent, Restricted API Calls, Used Permission, and Network Address to get the greatest outcomes. This resulted in an AUC of 97.24%, a TPR of 96.88%, an FPR of 2.39%, and an overall accuracy of 97.24% for RF. Since we got a higher TPR and a lower FPR, we may call the "best" experiment the one that maximizes the difference between the TPR and the FPR.



(a) using three words as substring



(b) using two words as substring



(c) using one word as substring

i. ROC Curves ii. Accuracy

Figure 5.3 *Performance of ML algorithms.*

5.4 Android Malware Detection Using Stacked Generalization

It's important to solve two difficulties before using stacking to classification problems. (i) the variety of characteristics used to generate level-1 data and (ii) the type of level-1 generalizer to enhance the accuracy of the stacking generalization technique by testing tree-based machine learning methods to identify malware on Android using the DREBIN dataset.

5.4.1 Stacked Generalization (Stacking)

Stacked generalization (or stacking), according to Wolpert [53], is a method for integrating several models in order to improve predictive acuity. Stacking, as opposed to bagging, and boosting, is often used to mix models of various kinds. Here's how to do it [54]:

a. Divide the training session into two halves.
b. Train the first portion to a large number of people.
c. Conduct test on a second part for the base learners.
d. Train a higher-level learner using predictions from c) as inputs and accurate responses as outputs

Instead of adopting a winner-takes-all strategy, we combine the base learners in stages a) to c), which are similar to cross-validation.

For a given dataset, $L = \{(y_n; x_n); n = 1, \ldots, N\}$, where y_n is the class value and x_n stands for the attribute values of the nth instance, randomly split the data into J almost equal parts $L_1, \ldots, L_J$. Define L_j and $L^{(-j)} = L\text{-}L_j$ as the test and training sets for the jth fold of a J-fold cross-validation. Given K learning algorithms, which we call *level-0*

generalizers, invoke the kth algorithm on the data in the training set $L^{(-j)}$ to induce a model $M_k^{(-j)}$, *for k = 1, …, K*. These are called *level-0* models.

For each instance *x* in L_j, the test set for the *j*th cross-validation fold, let $v_k^{(-j)}(x)$ denote the prediction of the model $M_k^{(-j)}$ on *x*. Let,

$$z_{kn} = v_k^{(-j)}(x_n)$$

Once all of the cross-validation has been completed, the data collected from the K model's outputs is

$$L_{cv} = \{(y_n, z_{1n}, \dots, z_{Kn}), n = 1,2, \dots, N\}$$

This is the *level-1* data. Use some learning algorithm that we call the *level-1 generalizer* to derive from this data a model *M*. This is the *level-1 model*. To complete the training process, models M_k, *k = 1, …, K*, are derived using all the data in *L*. Now let us consider the classification process, which uses the models M_k, *k = 1, …, K*, in conjunction with *M*. Given a new instance, models M_k produce a vector (z_1, *…,* z_K). This vector is input to the *level-1 model M*, whose output is the final classification result for that instance. This completes the stacked generalization method as proposed [55].

5.4.2 Results and Discussion

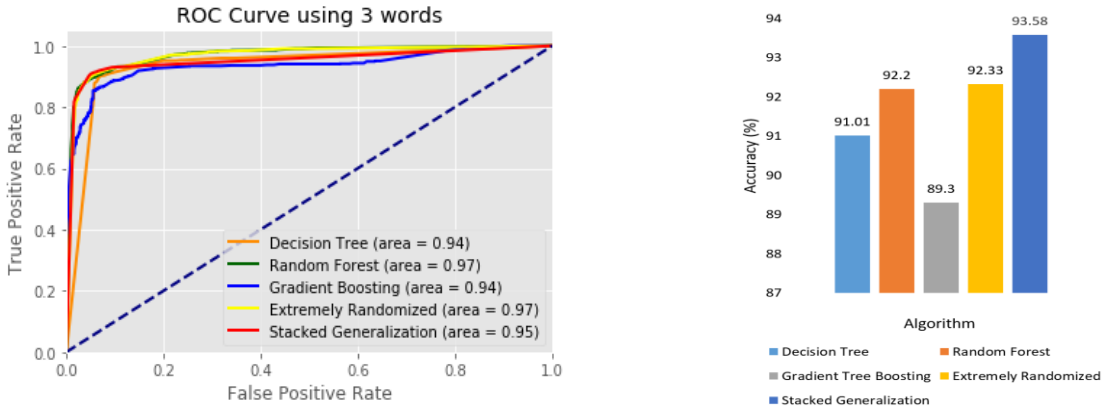
W-fold cross-validation is used for DREBIN datasets. The training dataset is utilized as L in this cross-validation, and the models that are generated are tested on the test dataset. This cross-validation is distinct from the *j*-fold cross-validations used as part of the stacking process since it is utilized for evaluating the whole method. W and J, on the other hand, are both set to 10 throughout the tests. As can be seen in Table 2, generalization

using the *level-1* model M yields classifications based on the *level-0* models M'. Table 5.4 shows the performance of stacking of ML classifiers.

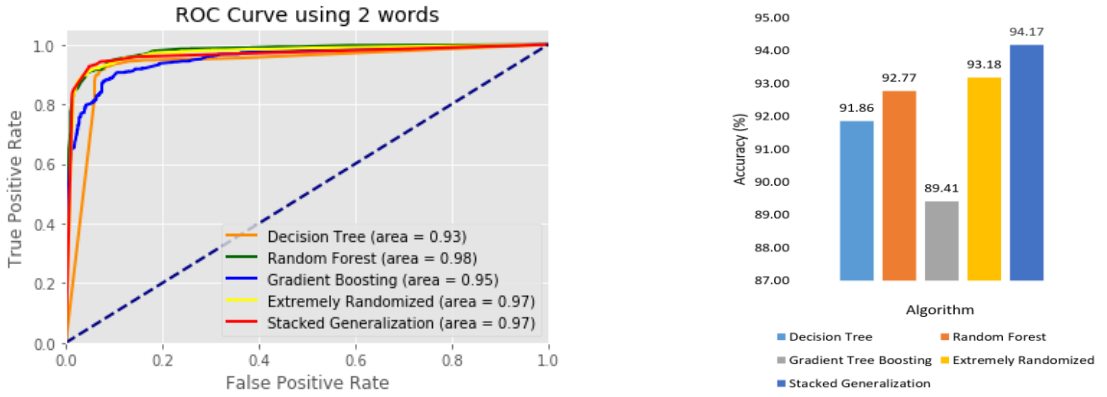
Table 5.4 *Performance results of stacked generalization with tree-based algorithms*

Substring	Algorithms	Precision		Recall		f1-score		Accuracy
		0	**1**	**0**	**1**	**0**	**1**	
3 words from each feature	DT	0.95	0.89	0.89	0.95	0.92	0.92	91.01
	RF	0.95	0.92	0.93	0.95	0.94	0.95	92.20
	GB	0.86	0.89	0.90	0.84	0.88	0.87	89.30
	ERT	0.95	0.91	0.91	0.95	0.93	0.93	92.33
	Stacking	**0.92**	**0.94**	**0.95**	**0.91**	**0.93**	**0.93**	**93.58**
2 words from each feature	DT	0.93	0.93	0.92	0.93	0.93	0.93	91.86
	RF	0.93	0.95	0.95	0.94	0.94	0.94	92.77
	GB	0.87	0.91	0.91	0.87	0.89	0.89	89.41
	ERT	0.94	0.94	0.94	0.94	0.94	0.94	93.18
	Stacking	**0.93**	**0.94**	**0.95**	**0.93**	**0.94**	**0.94**	**94.17**
1 word from each feature	DT	0.96	0.96	0.96	0.96	0.96	0.96	95.92
	RF	0.97	0.98	0.98	0.97	0.97	0.97	96.06
	GB	0.93	0.94	0.94	0.93	0.94	0.94	92.73
	ERT	0.97	0.97	0.97	0.97	0.97	0.97	96.34
	Stacking	**0.96**	**0.98**	**0.98**	**0.96**	**0.97**	**0.97**	**97.92**

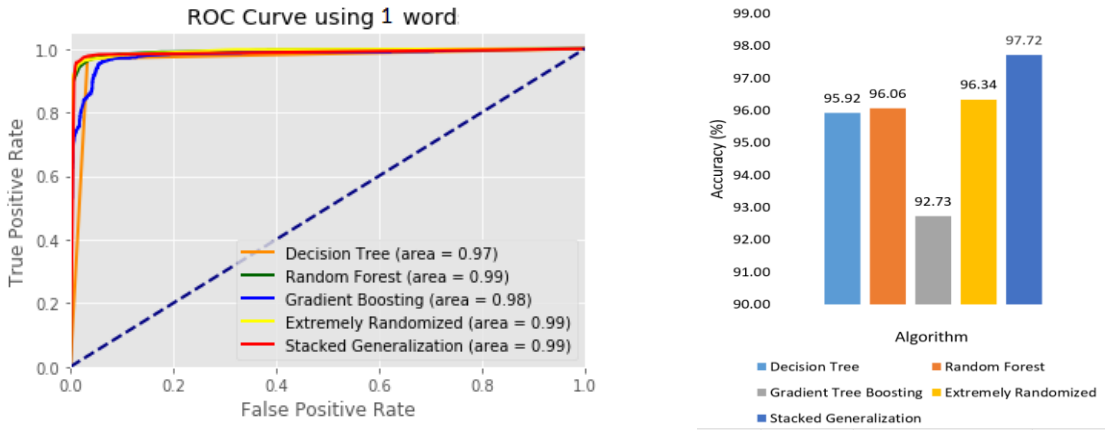
Last but not least, we see that the stacking is nearly always more accurate than the other single classifiers. The last words of permissions, activities, intents, and services are usually the most important terms, whereas alternative word combinations may have a detrimental effect on categorization. For example, substrings formed by utilizing the last 2 or 3 words frequently provide unnecessary information to classifiers. Figure 5.4 shows the results of comparing the overall performance with and without the same parameter configuration. That's an overall success rate of 97%.



(a) using three words as substring



(b) using two words as substring



(c) using one word as substring

i. ROC Curves ii. Accuracy

Figure 5.4 *Results of Stacked Generalization.*

5.5 Advanced Ensemble Learning Techniques for Android Malware Detection

In this section, we advance and assess various kinds of ML strategies and apply ensembled-based learning systems [56] that join a few ML procedures into one prescient model so as to diminish fluctuation (sacking), inclination (boosting), or improve prediction (stacking, mixing) [57] for taking care of the order issue to identify malware legitimately on the Android gadgets by performing static investigation on DREBIN dataset related to recently proposed Substring based component choice technique where the substring-based strategy helps in evacuating perhaps superfluous data and accelerates the recognition. The dataset is made from malware and favorable information and each record has different highlights of mentioned equipment parts, mentioned authorizations, application segments, arrange addresses, and so forth. The aftereffects of cutting-edge troupe learning procedures applied on various AI calculations (e.g., RF, DT, ERT, LR, SVM, GB) are evaluated dependent on different features, including api_call, highlight, URL, service_receiver, consent, call, goal, real_permission, movement, etc.

5.5.1 Overview of Ensemble Learning Techniques

The ensemble method is an ML technique that combines a few base models to create one ideal prescient model [58]. The ensemble learning technique can be categorized by the following:

- Basic Ensemble Techniques
- Advanced Ensemble Techniques
- Algorithms based on Bagging and Boost

5.5.1.1 Advanced Ensemble Techniques

To improve generalizability over a solitary estimator, the advanced ensemble method combines the predictions of a few base estimators worked with given learning techniques [59]:

- *Stacking*: An ensemble learning technique that uses predictions from multiple models (for example, decision tree, k-NN or SVM) to build a new model. This model is used to make predictions on the test set. A stepwise explanation is as follows:
    1. The train set is part into k (for example k = 10) sections.
    2. A base model (assume a decision tree) is fitted on k-1 sections and predictions are made for the kth part. This is carried out for each piece of the train set.
    3. The base model (for this situation, decision tree) is then fitted with the overall train dataset, and predictions are made on the test set using this model.
    4. Steps 2 to 3 are rehashed for another base model (say k-NN), bringing about another arrangement of predictions for the train set and test set.
    5. The predictions bought from the train set are utilized as features to assemble another model used to make the last prediction on the test prediction set.
- *Blending*: Unlike stacking, the predictions are made on the holdout set only. The holdout set and the predictions are used to build a model run on the test set. Steps are:

1. The train set is split into training and validation sets and models are fitted on the training set.
2. The predictions are made on the validation set and the test set.
3. The validation set, and its predictions are used as features to fabricate another model used to make final predictions on the test and meta-features.

- *Bagging*: The idea behind bagging is an inspecting strategy wherein subsets of perceptions are from the original dataset with a replacement where the size of the subsets is equivalent to the size of the original set. The size of subsets created for bagging might not be exactly the original set. Steps can be depicted as follows:
  1. Multiple subsets are created from the original dataset, selecting observations with replacement.
  2. A base model (weak model) is created on each of these subsets.
  3. The models run in parallel and are independent of each other.
  4. The final predictions are determined by combining the predictions from all the models.
- *Boosting*: A consecutive procedure where each ensuing model tries to correct the errors of the earlier model. The succeeding models are reliant on the earlier model. Let us understand the way boosting works in the below steps:
  1. A subset is made from the original dataset and all information focuses are given equivalent weights initially.
  2. A base model is made on this subset that is used to make predictions on the entire dataset.

3. Errors are determined to use the actual values and anticipated values and allocate higher weights for those perceptions which are erroneously anticipated.
4. Another model is made, and predictions are made on the dataset. Additionally, different models are made, each remedying the errors of the previous model.
5. The final model (strong learner) is defined by the weighted mean of the considerable number of models (weak learners).

5.5.2 Results and Discussion

For comparison, the classifiers' performances are depicted using the following Table 5.5 that presents the outcomes of advanced ensembled learning techniques by applying the SBFS method.

The most appropriate features are selected using the SBFS (Phase II) method and train different ML classifiers three times in order to build three unique models for three different types of Substring. For example, the 1st model is built using the last three words, the 2nd is built with the last two words, and 3rd is built with only the last word as Substring. Then, we confirm each of the models with the test set and compare the overall experimental result of each machine learning algorithm with previously reported results. Table 5.5 shows that the accuracy using the 1st model is about the same as the accuracy presented before applying the SBFS technique. The accuracy increases when using the 2nd model and achieves the best outcomes when running these classifiers using the 3rd model.

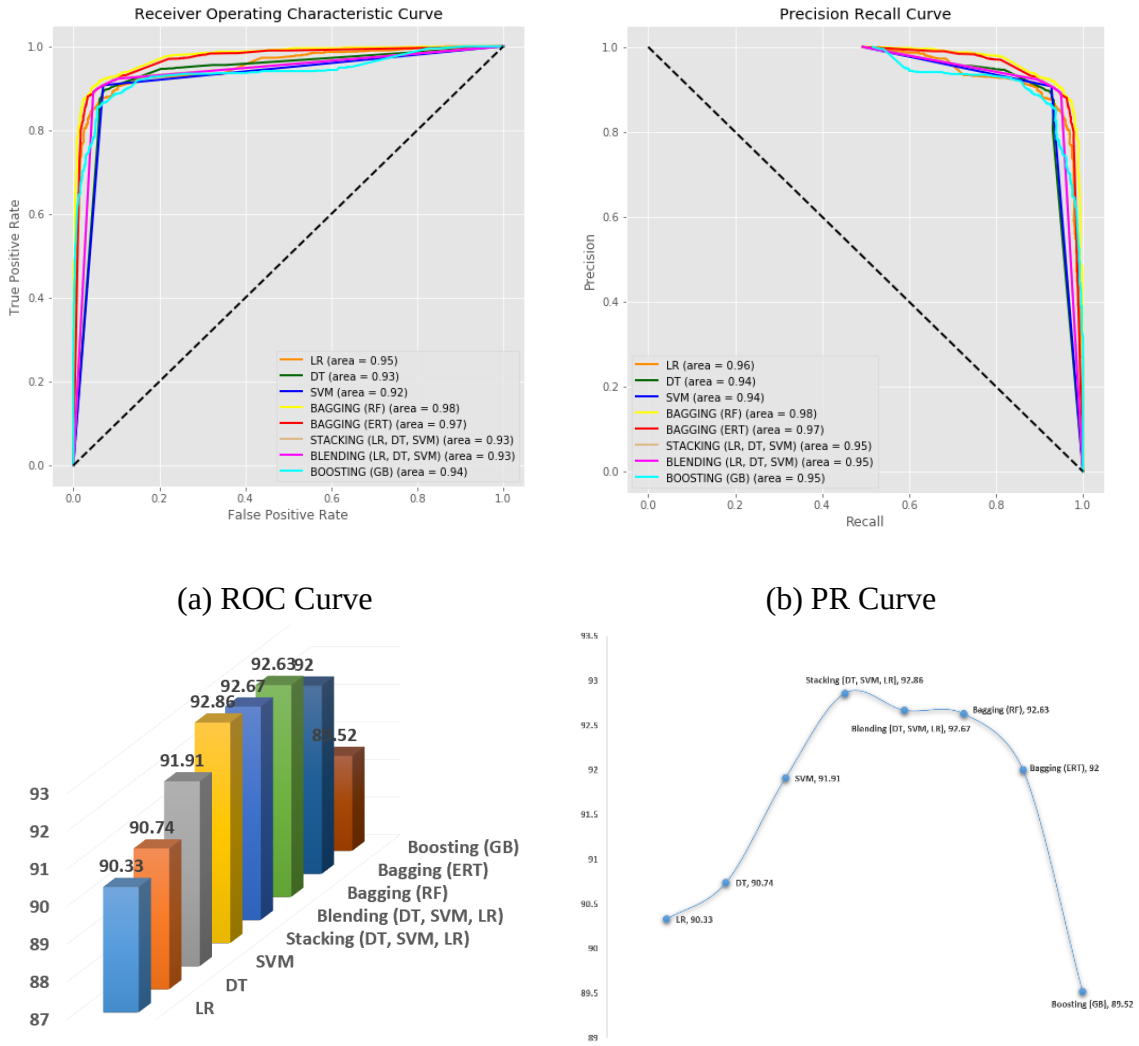
Table 5.5 *Performance results of advanced ensemble learning techniques*

Substring	Algorithms	Precision		Recall		f1-score		Accuracy
		0	1	0	1	0	1	
1st model (using 3 words from each feature)	LR	0.88	0.93	0.94	0.86	0.91	0.90	90.33
	DT	0.90	0.91	0.91	0.90	0.91	0.91	90.74
	SVM	0.91	0.93	0.93	0.91	0.92	0.92	91.91
	Stacking (DT, SVM, LR)	0.92	0. 93	0.93	0.92	0.93	0.93	92.86
	Blending (DT, SVM, LR)	0.92	0.94	0.93	0.92	0.92	0.92	92.67
	Bagging (RF)	0.92	0. 93	0.93	0.92	0.93	0.92	92.63
	Bagging (ERT)	0.91	0.93	0.94	0.90	0.92	0.92	92.00
	Boosting (GB)	0.86	0.94	0.94	0.84	0.90	0.89	89.52
2nd model (using 2 words from each feature)	LR	0.93	0.93	0.93	0.93	0.93	0.93	93.21
	DT	0.95	0.92	0.92	0.95	0.93	0.94	93.66
	SVM	0.96	0.95	0.95	0.96	0.95	0.95	95.14
	Stacking (DT, SVM, LR)	0.96	0.95	0.95	0.96	0.95	0.95	96.23
	Blending (DT, SVM, LR)	0.96	0.95	0.95	0.96	0.95	0.95	96.17
	Bagging (RF)	0.96	0.95	0.96	0.96	0.96	0.96	95.86
	Bagging (ERT)	0.95	0.96	0.96	0.95	0.95	0.95	95.37
	Boosting (GB)	0.93	0.92	0.92	0.92	0.92	0.92	92.37
3rd model (using 1 word from each feature)	LR	0.96	0.97	0.97	0.96	0.96	0.96	96.40
	DT	0.97	0.97	0.97	0.97	0.97	0.97	96.81
	SVM	0.98	0.96	0.97	0.98	0.97	0.97	97.12
	Stacking (DT, SVM, LR)	0.98	0.97	0.97	0.98	0.97	0.97	97.96
	Blending (DT, SVM, LR)	0.98	0.96	0.97	0.98	0.97	0.97	97.72
	Bagging (RF)	0.97	0.98	0.98	0.97	0.97	0.97	97.21
	Bagging (ERT)	0.96	0.98	0.98	0.95	0.97	0.97	96.58
	Boosting (GB)	0.93	0.95	0.95	0.92	0.94	0.94	93.89

However, we found in our experiments, while other words can have a positive impact on classification, the combination of words can also have a negative impact. For example, Substrings created by using words from the last 2 and 3 of a permission, activity, intent, or service may introduce irrelevant information to classifiers.

Finally, we see that the advanced ensembled learning technique produces the best results compared to the other classifiers. Based on the experiment, the main difference of performances among the several learning algorithms is their model size, while ensembled

learning (e.g., RF, ERT, etc.) model size is very large. Figures 5.5, 5.6, and 5.7 show the overall accuracy of each ML algorithm based on the same parameters in our experiment. Based on the depicted results, it is seen that Stacking obtains the best performance, and it produces an overall accuracy of 97.96%, TPR of 97.19%, FPR of 2.10%, and AUC of 0.98.



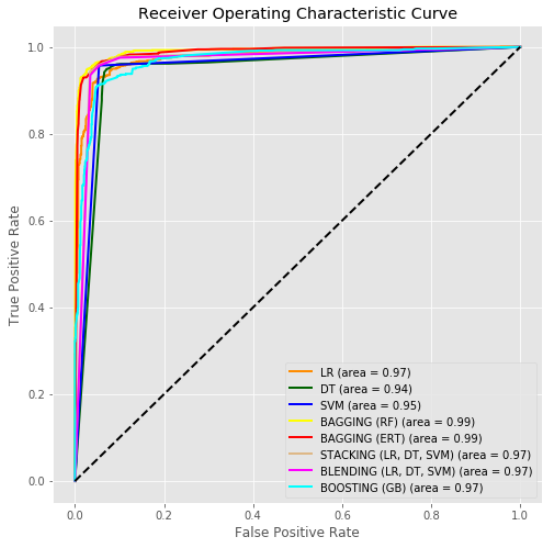
(a) ROC Curve

(b) PR Curve

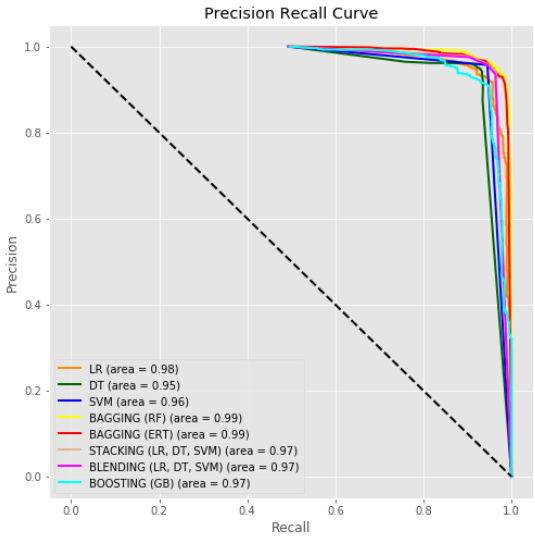
(c) Accuracy (Bar)

(d) Accuracy (Scatter Plot)

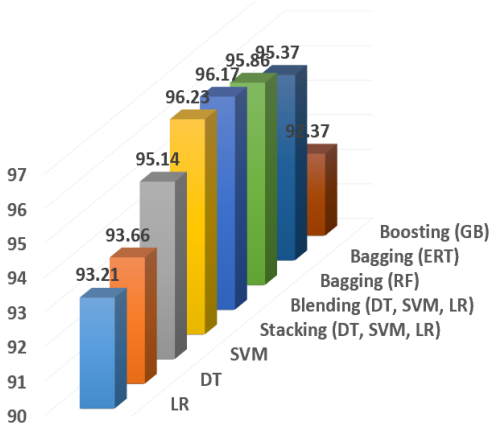
Figure 5.5 *Three words as Substring (1st model).*



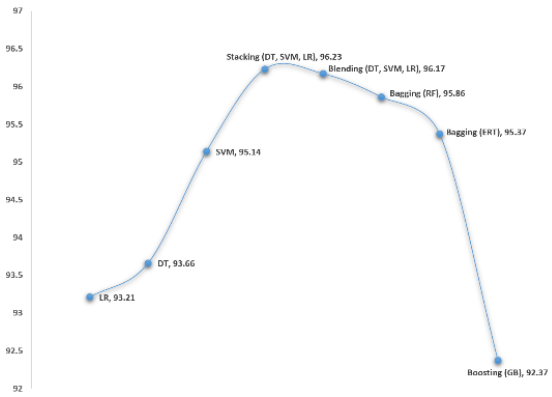
(a) ROC Curve



(b) PR Curve

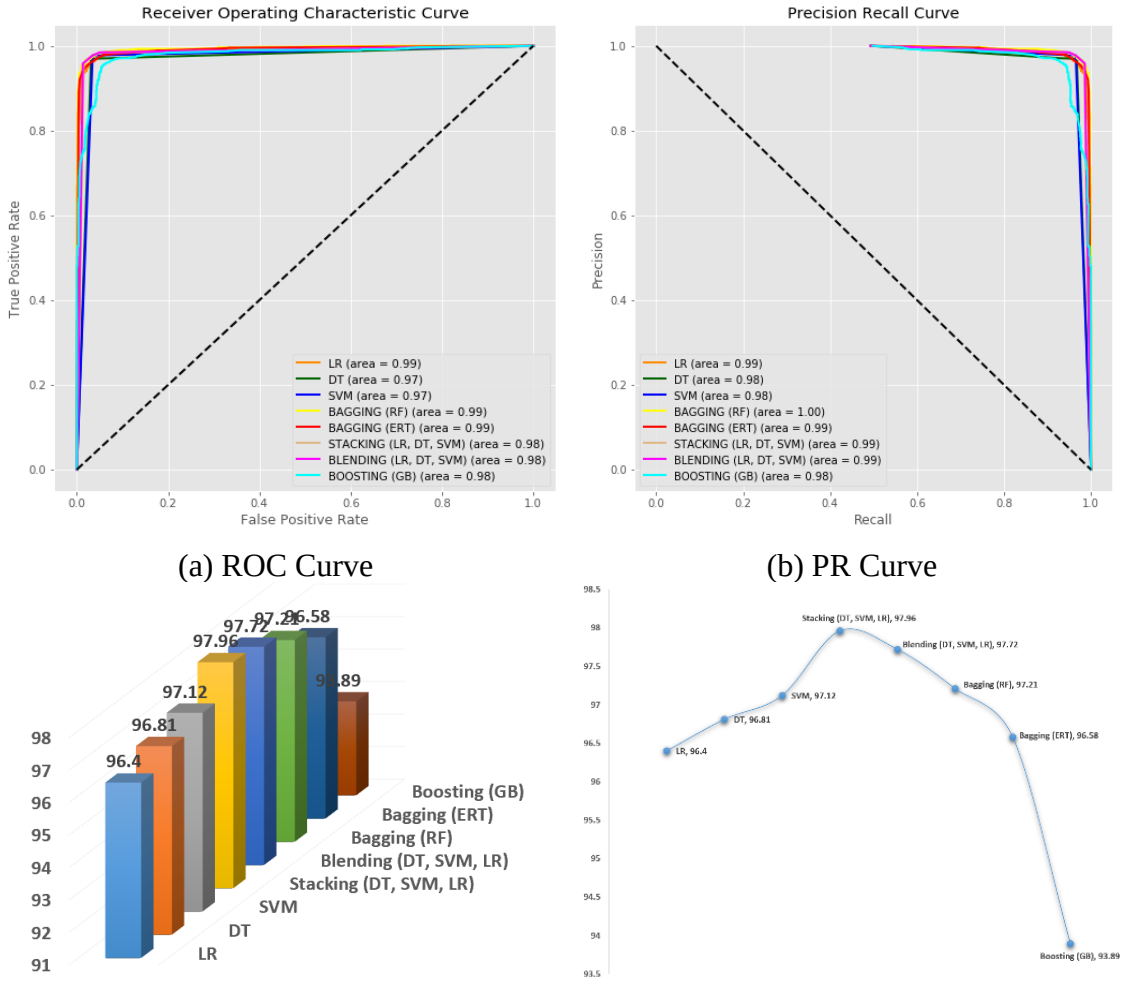


(c) Accuracy (Bar)



(d) Accuracy (Scatter Plot)

Figure 5.6 *Two words as Substring (2nd model).*



(a) ROC Curve (b) PR Curve

(c) Accuracy (Bar) (d) Accuracy (Scatter Plot)

Figure 5.7 *One word as Substring (3rd model).*

5.6 Evaluating ML Models on Ethereum Blockchain for Android Malware Detection

In order to solve the Android malware detection problem, we offer ML-based solutions [60] built on the DanKu [61] protocol that enables the initiator to publish a dataset, assign assessment criteria and offer a reward for the best machine learning model submitted. Figure 5.8 illustrates the suggested system. Smart contracts are used to reward competitors for their effort by enabling them to submit their solutions (models) after

training with specific ML algorithms in a safe and reliable way. Competitors' study of datasets aids a variety of network organizations in improving or enhancing their existing malware detection systems. By removing third parties, the decentralized network increases openness, enhances security, and lowers costs for maintaining all essential data.

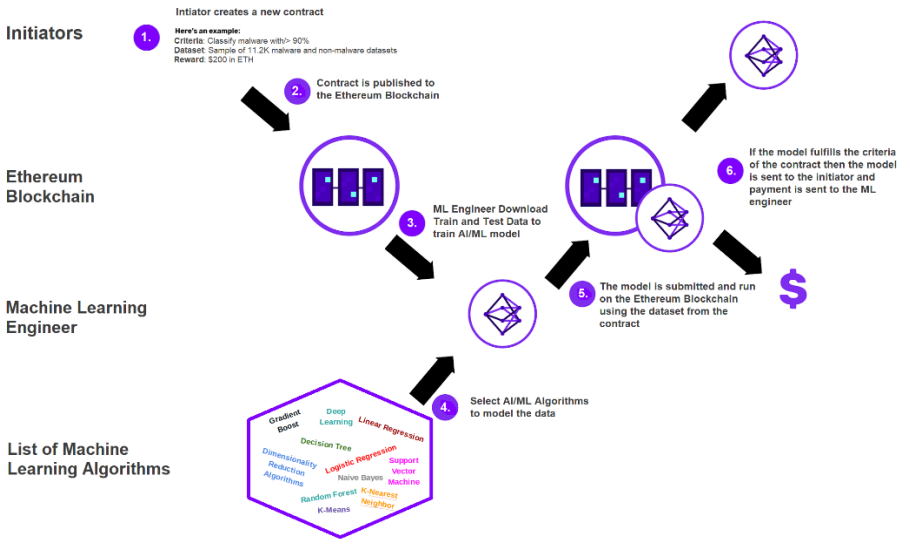


Figure 5.8 *Overview of the proposed framework.*

5.6.1 Proposed System Architecture

Our proposed consortium Blockchain system for evaluating different ML models for a particular malware dataset is described in this section.

5.6.1.1 Basic Structure

A three-phase method has been suggested to guarantee trust and transparency (see Figure 5.9).

Phase 1: A dataset defined with a train-to-test ratio is revealed by the initiator (Alice), along with an evaluation criterion that miners or machine learning engineers must

meet, and a reward amount typically denominated in cryptocurrency for the most successful model submission to the Ethereum smart contract. The Ethereum smart contract offers a collection of machine learning methods for data modeling and classification task evaluation.

Phase 2: It's Bob, Daniel, and Mice's job to download Alice's training dataset and use the machine learning method they've chosen to create a model of the information. Send their solutions to the Blockchain once they've been trained and evaluated. Submitter ID, which is provided by the smart contract, is used to uniquely identify submitted results in the Blockchain.

Phase 3: A competition winner will be chosen based on the accuracy criteria once the Blockchain has completed its analysis of the submissions.

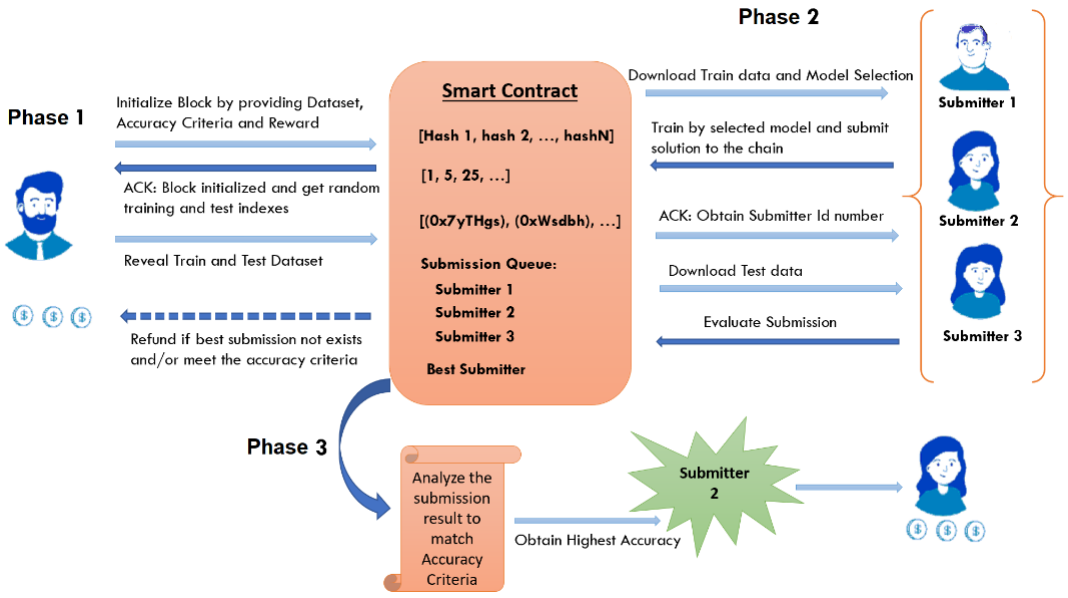


Figure 5.9 *Structure of the Proposed System.*

5.6.1.2 Definitions

- A **user** can be anyone who can interact with Ethereum contracts.
- An **initiator** is a user who initializes the Blockchain with a genesis file and creates smart contracts.
- A **Smart Contract** is an Ethereum contract.
- A **submitter (miner or machine learning engineer)** is a user who submits solutions to the Blockchain for a reward.
- A **period** is a timeframe needed to mine a block.
- A **data point** is made up of input(s) and prediction(s).
- A **data group** is a matrix made up of data points.
- A **hashed data group** is the hash of a data group, including a random nonce.
- A **contract wallet address** is an account that holds the reward amount until the finalization of the contract.

5.6.1.3 Features of Smart Contract

- ***Initialization()***, using this function, Bob will create a genesis block with some parameters, including Dataset, Accuracy criteria, and a reward amount.
- ***IsBlockInitialized()*** is responsible for whether the contract is created, initialized, and returns two arrays containing Training and Test indexes of the dataset.
- ***RevealData()*** splits the dataset into training and testing sets by creating two arrays containing the indexes generated randomly and reveals to the chain.

- ***DownloadTrainDataAndMLASelection()*** is used by miners/submitters to get the data revealed in the previous step and returns two arrays (i) a list of ML Algorithms and (ii) a list of Training Data Indexes.
- ***SubmitSolution()*** is invoked after modeling the data using training indexes with one machine learning algorithm and returns Submitter_ID as acknowledgment.
- ***EvaluateSubmission()*** takes submitter id as an input parameter to evaluate the model from the previous step and finally returns the evaluation result (Accuracy, Precision, Recall, F1-score, etc.) as an array.
- ***SubmissionQueue()*** is responsible for storing the submission information into the Blockchain along with Submitter_ID.
- ***BestSubmitter()*** retrieves submitted information from the Blockchain to analyze accuracy criteria and returns Submitter_Address and Submitter_ID.
- ***ContractFinalized()*** To finalize the competition, this function is used with returning Boolean Value (True or False).
- ***CancelContract()*** is used to cancel the contract if any exception occurs.
- ***RewardOrRefund()*** This function is used to send the reward amount to the best submitter and/or refund the reward amount to the initiator if no one fulfills the conditions.
- ***GetTrainingIndex()*** is used to retrieve training indexes.
- ***GetTestingIndex()*** is used to retrieve testing indexes.
- ***GetSubmissionId()*** is used to retrieve submission id contract.

5.6.2 Implementation

This section describes implementation details.

5.6.2.1 Dataset Hashing

The initiator who creates the contract splits the dataset into data groups. A nonce is a number generated randomly along with data groups, and it is used to identify each data group. The initiator hashes these data groups using SHA256 algorithms.

5.6.2.2 Determine Train and Test Dataset

During Initialization in phase 1, the initiator calls the function *Initialization(),* which uses previously mined block hash number as a seed for randomization. The dataset groups are randomly selected using a nonce. The dataset is split into 80% for training and 20% for testing. We repeat the procedure until all data group indexes are selected—the algorithm used in the DanKu protocol (in solidity) for randomly selecting hashes:

```
function randomly_select_index(uint[] array) private {
  uint t_index = 0;
  uint array_length = array.length;
  uint block_i = 0;
  // Randomly select training indexes
  while(t_index < training_partition.length) {
    uint random_index = uint(sha256(block.blockhash(block.number-block_i))) %
array_length;
    training_partition[t_index] = array[random_index];
    array[random_index] = array[array_length-1];
    array_length--;
    block_i++; t_index++;
  }
  t_index = 0;
  while (t_index < testing_partition.length) {
    testing_partition[t_index] = array[array_length-1];
    array_length--; t_index++;
  }
}
```

Listing 5.1 *Data randomization.*

5.6.3 Competition Rules

The proposed system has been designed in such a way that no participants can cheat or get advantages over other users (e.g., the initiators, the submitters, and/or ML engineers of the Ethereum Blockchain).

5.6.3.1 Overfitting by Submitter

In order to prevent overfitting problems, the test dataset is kept secret until the submission period ends. Submitters or machine learning engineers can overfit their selected machine learning model if they have access to the test dataset. The smart contract evaluates the submission using a test dataset.

5.6.3.2 Too Many Submissions

The competitor cannot submit a solution more than once because, after the first successful submission, the smart contract saves the information to the chain with his/her address along with the submission id.

5.6.3.3 Block Hash Manipulation by Miners

A miner does not have complete control over the working procedure of training index selection. After observing which data groups will be selected, the initiator can decide whether to mine the block or not, which could result in disagreeable training indexes.

5.6.3.4 Distributed Reward System Abuse

To prevent malicious submitters from re-submitting the similar solution by making small changes to the original submitter solution and calling the evaluation function before the original submitter, a distributed reward system may de-incentivize submitters. Due to this reason, it ensures that the best submitter will only be paid. The first submitter will be rewarded if there are two submissions with the same solution and evaluated.

5.6.4 Result and Analysis

We compared the results of the ML classifier submitted by the competitors or miners in a simulated environment. More than one competition is held, and their respective accuracy criteria are recorded (see Figure 5.10 and Figure 5.11). We conducted four rounds for our experimental purpose, which are described in Table 5.6.

Table 5.6 *Result of the performance*

Competition	Submitter	Algorithms	Precision		Recall		f1-score		ACC
			0	1	0	1	0	1	
COMP1 with Accuracy criteria 80%	1	k-NN	0.92	0.89	0.89	0.92	0.90	0.91	90.47
	2	DT	0.93	0.88	0.88	0.94	0.91	0.91	91.78
	3	LR	0.78	0.84	0.86	0.75	0.82	0.80	80.94
	4	NB	0.68	0.52	0.15	0.92	0.25	0.66	53.60
	5	MLP	0.85	0.91	0.92	0.83	0.88	0.87	87.54
COMP2 with Accuracy criteria 85%	1	ERT	0.93	0.93	0.93	0.93	0.93	0.93	93.66
	2	DA	0.76	0.88	0.91	0.70	0.83	0.78	80.71
	3	SVM	0.91	0.91	0.91	0.91	0.91	0.91	90.74
	4	GB	0.87	0.88	0.89	0.86	0.88	0.87	87.50
	5	k-NN	0.92	0.89	0.89	0.92	0.90	0.91	90.47

Table 5.6 (continued).

Competition	Submitter	Algorithms	Precision		Recall		f1-score		ACC
			0	1	0	1	0	1	
COMP3 with Accuracy criteria 90%	1	DA	0.76	0.88	0.91	0.70	0.83	0.78	80.71
	2	GB	0.87	0.88	0.89	0.86	0.88	0.87	87.50
	3	SVM	0.91	0.91	0.91	0.91	0.91	0.91	90.74
	4	LR	0.78	0.84	0.86	0.75	0.82	0.80	80.94
	5	RF	0.95	0.94	0.94	0.95	0.94	0.94	94.33
COMP4 with Accuracy criteria 95%	1	RF	0.95	0.94	0.94	0.95	0.94	0.94	94.33
	2	ERT	0.93	0.93	0.93	0.93	0.93	0.93	93.66
	3	DT	0.93	0.88	0.88	0.94	0.91	0.91	91.78
	4	MLP	0.85	0.91	0.92	0.83	0.88	0.87	87.54
	5	SVM	0.91	0.91	0.91	0.91	0.91	0.91	90.74

Finally, we were able to get the greatest results after a series of tests in which we used 80% for training and 20% for testing under the conditions of the top 100 most frequent API requests and system permissions combined with information gain. We ran a competition where five competitors competed to submit their Blockchain-trained models. For example, in the first competition that assigns the accuracy criterion (80%), we get various results if we see each competition, so anyone can get the reward who gets more or equal to accuracy criterion, here Submitter_2 gets the reward by bringing the highest accuracy 91.78% among the others by using DT classifier. Similarly, Submitter_1 gets awarded with an accuracy of 93.66% by utilizing ERT algorithm to meet the accuracy requirements (85%) in the second competition. In the third round, Submitter_5 won with an accuracy score of 94.33% by using a RRF classifier where the criterion was 90%. However, in the fourth competition, no one will be awarded since they failed to meet the

accuracy criterion set at 95%, thus the prize money will be returned to the person who started the competition.

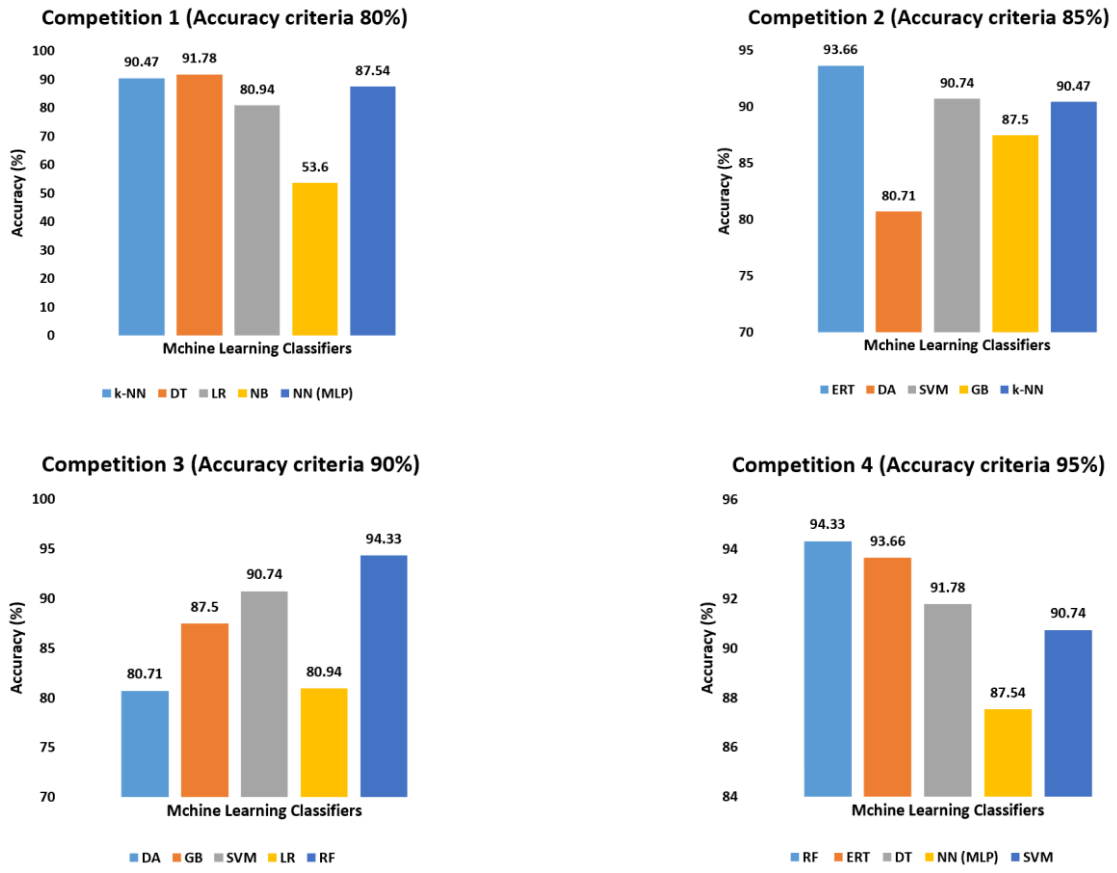
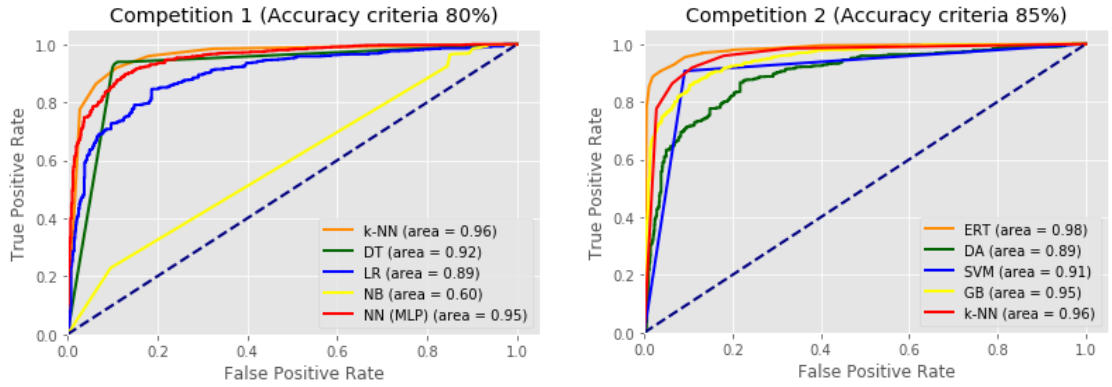


Figure 5.10 *Accuracy.*



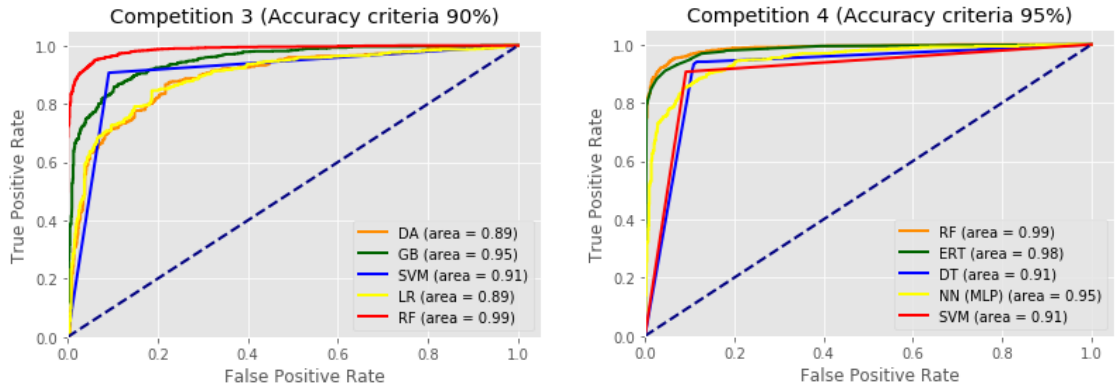


Figure 5.11 *ROC curve.*

CHAPTER VI – CONCLUSION

Detection of mobile malware has become a fundamental problem due to the increase in Android market applications. Accordingly, researchers have proposed many methods using various Android malware datasets to study the detection problem. The primary contribution of this research is the proposed SBFS method, which removes irrelevant information and automatically selects those features that contribute the most to the detection. The earlier research on Android malware has focused on using two standard features (i.e., system permissions and API calls); in our study, we use eight features and see their effects on detection accuracy.

For the DREBIN dataset, previous research has demonstrated that SVM performs the best. Our study focused on the issue of removing unnecessary data by using the SBFS feature selection technique by using a variety of ML algorithms on the same dataset. Using the DREBIN dataset, we ran tests on a variety of learning algorithms. The RF classifier performed the best in our first trial, with 97.24% of accuracy (with 96.88% APR, 2.39% FPR, and 97.23% f1-score, and 97.58% precision). On the other hand, we observe an improvement in accuracy of more than 3% after using the SBFS technique on the DREBIN dataset and this offers a solid foundation for creating powerful malware scanners in the future.

Aside from that, we tested a variety of ML models on a consortium Blockchain for a specific dataset. There is a prize for the best ML model submission, and the winner will get a bonus. It enables different network companies to improve or increase their existing malware detection systems by competitors' assessment of datasets. By removing third parties, the decentralized network increases openness, enhances security, and lowers

management costs for all essential data. Using this blockchain, a real-time malware detection system will be developed, helping network administrators identify and prevent potentially harmful activity.

For future work, we propose to study new feature extraction and selection techniques, the combinational effects of feature sets and learning machines, ensemble methods, and anomaly detection techniques for detecting unknown malware.

Section II:

Analysis and Detection of Deepfake

CHAPTER VII – PREFACE

7.1 Introduction

Rapid progress in AI, ML, and DL has resulted in various technologies and tools for manipulating multimedia. Though most applications developed are for legitimate purposes such as entertainment, education, etc., malicious users can also exploit them for unlawful or nefarious purposes. For example, high-quality and realistic fake videos, images, or audios have been created to spread misinformation and propaganda, foment political discord and hate, or even harass and blackmail people; these manipulated, high-quality and realistic videos became known recently as Deepfake. Various approaches have since been described in the literature to deal with the problems raised by Deepfake.

The keyword “Deepfake manipulation” permits anyone to swap the face of an individual with another’s face, including expressions, and creates photorealistic fake videos or images that are known as Deepfakes and readily visible in malicious use. In the past, video manipulation was an expensive task requiring an extensive workforce, time, and money. Still, it now needs a gaming laptop or desktop with an internet connection and a basic knowledge of artificial neural networks. Deepfakes became popular when fabricated porn videos of well-known faces; for example, celebrities or politicians are in the progress of making it online. The term violates not only the rules of consent but the victim’s privacy. Because creating Deepfakes without a person’s approval is a form of abuse leading in another way of crime.

In order to generate such counterfeit videos, a generative network and a discriminative network with a FaceSwap technique [62-63] are used. The generative network creates fake images using an encoder and a decoder. The discriminative network

defines the authenticity of the newly generated images. The combination of these two networks is called Generative Adversarial Networks (GANs), proposed by Ian Goodfellow [64].

Based on a yearly report [65] in Deepfake, DL researchers made several related breakthroughs in generative modeling, including a proposed method known as Face2Face [66] for facial re-enactment. This method transfers facial expressions from one person to a real digital 'avatar' in real-time. In 2017, UC Berkeley presented CycleGAN [67] to transform images and videos into unique styles. The University of Washington proposed a method to synchronize the lip movement in video with a speech from another source [68]. In January 2018, a Deepfake creation service was launched by various websites funded by user donations, and after a month, multiple sites, including Gfycat [69], Pornhub, and Twitter, banned Deepfakes successfully. Rossler et al. introduced a vast video dataset to train the media forensic and Deepfake detection tools called FaceForensic [70] in March 2018. After a month, researchers at Stanford University published a method, "Deep video portraits" [71], allowing the reanimation of portrait images in photo reality. UC Berkeley researchers offered another approach [72] for transferring a person's body movements to another person in the video. NVIDIA introduced a style-based generator architecture for GANs [73] for synthetic image generation.

As presented in the annual report [65] under the broader name of Deepfake, Google searches provide several web pages for the keyword 'Deepfake' that expanded rapidly since 2017 and searches for web pages containing related videos (see Figure 7.1). This report also presents.

- **1790+** of Deepfake videos hosted by the top 10 adult websites without considering pornhub.com, which has disabled searches for ‘Deepfakes’.
- **6174** of Deepfake videos hosted by adult websites featuring fake video content only.
- **3** new sites dedicated to hosting Deepfake pornography.
- **902** of papers published on the arXiv, including ‘GAN (Generative Adversarial Network)’ in titles or abstracts in 2018 only.
- **25** articles on the topic published, including non-peer, reviewed where DARPA funds **12** of them.

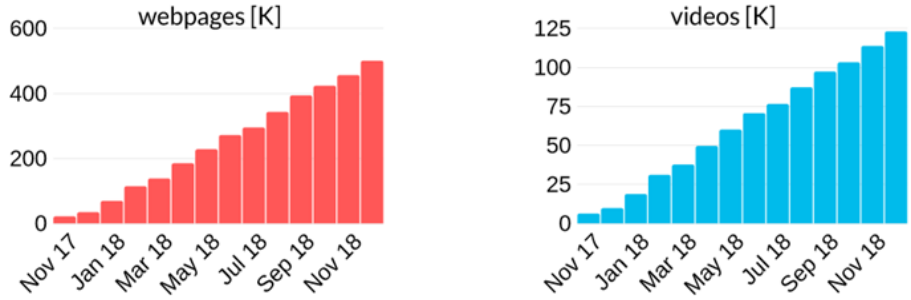


Figure 7.1 *(a) webpages that are provided by Google search engines, where the search keyword is “Deepfake,” (b) The no. of searches for a webpage that has Deepfake related video.*

7.2 Broader and Societal Impacts

In the current political and social environment where misinformation, fake news, and lies are endlessly propagated on social media, the nation’s democracy, security, and even public health are under unprecedented threat: Facebook, Microsoft, and Google understood the need for a fast and automatic Deepfake detector and have already mobilized resources in this domain. Therefore, developing an effective technology for Deepfake

detection will make a valuable contribution to the coordinated efforts of combating this severe threat.

7.3 Motivation to Analyze and Detect Deepfake

The ability to detect original images and videos from manipulated ones is crucial because it may sometimes be challenging. Fake media are computer-generated or modified images or videos with the specific purpose of deceiving human eyes. For this reason, advanced computer algorithms such as DL, ML, etc., have been developed to execute this work more precisely than humans.

Most ongoing research and mitigation efforts have focused on automated Deepfake detection using various DL methods as these approaches minimize the human effort in feature engineering; however, as much is done automatically, it increases the complexity of understanding and interpreting the model.

Explainability and Interpretability are significant disadvantages of DL-based methods. Sensitive applications, such as industrial robots that run with human beings or self-driving vehicles, are important for protection. In our case, it would improve trust in the model and simplify the future manual inspection if we explained the exact cause of a video as a Deepfake.

This thesis aims to propose an original solution for detecting Deepfake manipulation using ML-based and DL-based algorithms.

7.4 This Thesis

The second part of this thesis is divided into six chapters as follows. Chapter 7 is an introduction to the subject of the work. Chapter 8 provides background information on Deepfake, including its definition and generation pipeline. Chapter 9 deals with related research efforts and a summary of them. Chapter 10 describes various datasets that are used for Deepfake detection tasks. Chapter 11 discusses challenges and limitations by providing a hypothesis test. Chapter 12 presents our proposed detection methods using ML-based methods and DL-based methods, and chapter 13 analyzes performance results. Chapter 14 discusses possible research paths towards building Deepfake detectors that are highly dependable and usable in realistic settings. We conclude this thesis in chapter 15. The significant contributions of this thesis can be summarized as follows:

- ✓ Propose a deep ensemble learning technique called DeepfakeStack for detecting such manipulated videos.
- ✓ Create a unique set of features by combining HOG, Haralic, Hu Moments, and Color Histogram features
- ✓ Lessen the data's complexity and making patterns recognizable by splitting a task into two stages: object detection and object recognition.
    - The object detection phase scans an entire image and identifies all possible objects.
    - The object recognition step identifies relevant objects.
- ✓ The advantages of the ML method are:
    - Provide better understandability and interpretability of the model.

- Reduce training time and achieve the same level of performance. Using FF++, DFDC, and VDFD datasets with the same amount of train and test samples, the ML methods take between several seconds and a couple of hours, while the most advanced DL algorithm, for example, the Resnet, takes nearly two weeks.

CHAPTER VIII - BACKGROUND INFORMATION

Before discussing the details of our analysis framework, it is essential to understand what Deepfake is and how it is generated. We present a brief introduction to Deepfake in this chapter. In Section 8.1, we will begin with a definition of definition and two popular methods that have attracted cybercriminals in doing such image or video manipulation. Section 8.2 shows the creation pipeline.

8.1 Definition

A combination of "Deep Learning" and "Fake" can be called Deepfake that refers to any photo-realistic audiovisual content formed with the help of DL. The technique is initiated by analyzing plenty of photos or a video of one's face, training an AI algorithm to manipulate that face, and then using that algorithm to map the face onto a person in a video. In late 2017, the term "Deepfake" was named after a Reddit user known as Deepfakes, who used DL technology and attempted to replace a target actor's face with another's face in pornographic videos.

In recent, two popular facial manipulation methods have attracted a lot to cybercriminals in doing video manipulation job:

- *Facial expression manipulation*: Face2Face [66] allows anyone to transfer a person's facial expressions to another using standard tools or applications in real-time. For example, "Synthesizing Obama" [68] can animate a person's look by transferring another person's facial expression based on an audio input sequence.

- *Facial identity manipulation*: In FaceSwap [74], a person's face is replaced by another person’s face instead of changing expressions. For example, Snapchat. The same method is applied in Deepfake using DL technology.

8.2 Deepfake Generation Pipeline

A combination of two neural networks: (i) a synthesizer or generative network, and (ii) a detector or discriminative network builds GANs, and these two neural networks participate in creating realistic fake video or image. The generative network is built with an encoder and decoder and responsible for creating images, and the discriminative network decides whether the created image is accurate and believable. The below example describes how it works (see Figures 8.1-8.4).

1. After collecting many images for both actors (A and B), it builds an encoder for encoding all these images to retrieve the essential features. After then reconstructs the corresponding images by using the decoder.
2. Use different decoders for actor A and actor B for decoding the features. To do this, using the backpropagation algorithm train the generative network in such a way that the input is fitted tightly with the output.
3. After the training, it needs to deal with the video frame-by-frame to exchange A's face with the look of B. In order to this work, first, extract A’s face out using face detection and feed it into the encoder, then use the decoder of the actor B to reconstruct the image instead of feeding to its original decoder (i.e., decoder of A) (see Figure 8.1).

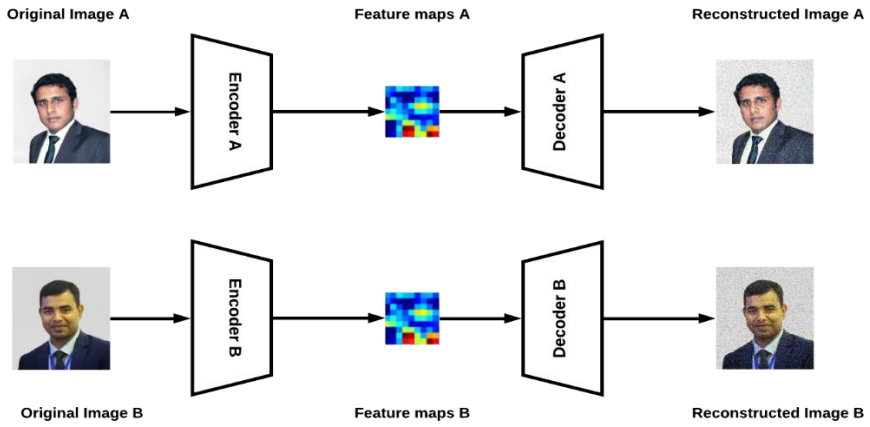


Figure 8.1 *Train encoder and decoder with source and target face in the video.*

4. Then, amalgamate the original image into the newly formed face image (see Figure 8.2).

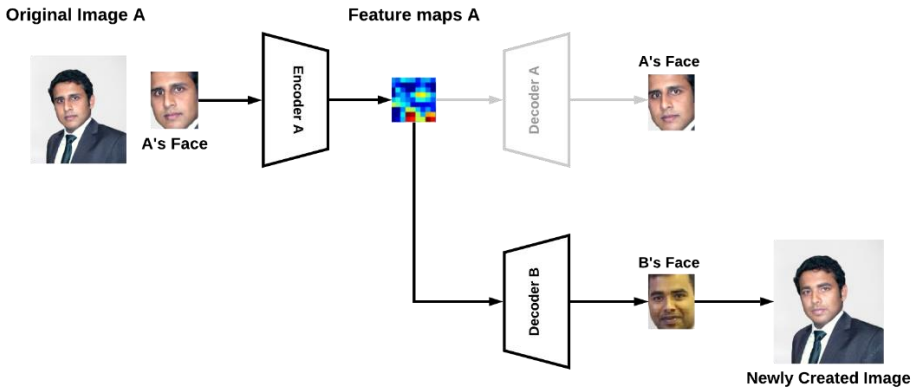


Figure 8.2 *Generate Deepfake look by merging new and source faces.*

Prior to training, it needs improvement of the overall quality of the facial images by doing some image processing tasks, for example, eliminating image frames containing multiple people at the same time or by extracting facial views (e.g., pose, face angle, facial expressions, etc.) with reducing poor quality, insufficient illumination or obstructed facial look. It generates a mask or kernel on

the forged face for blending it with the target video to create Deepfakes. After then applies a Gaussian filter to the scattered boundary of the kernel and sets up the application to magnify or reduce the mask further (see figure 8.3).

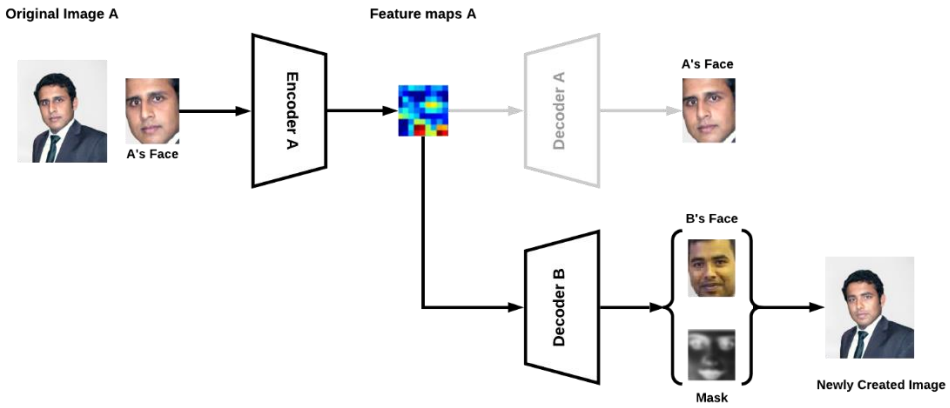


Figure 8.3 *Apply a Gaussian filter to diffuse the mask boundary area further.*

5. The last step (see Figure 8.4) feeds original images to the discriminative network and trains itself to identify originality better. It confirms the created images as human eyed indistinguishable to the authentic.

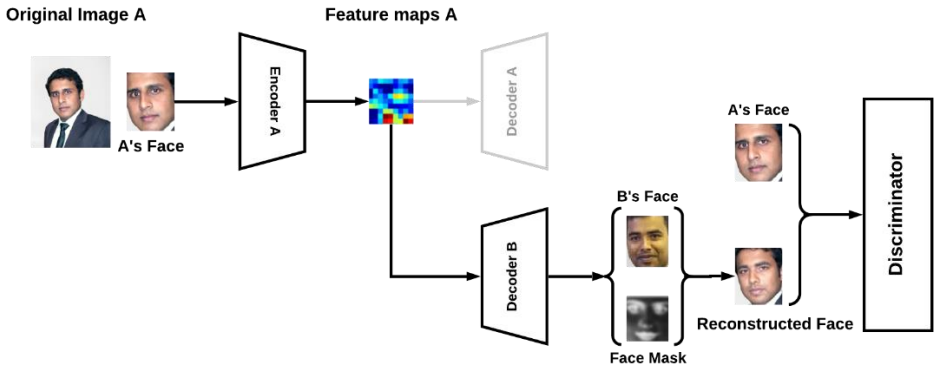


Figure 8.4 *Decide fake or real faces by the discriminator.*

CHAPTER IX – RELATED WORK

9.1 Machine Learning-Based Methods

These methods show certain irregularities found in the GAN method typically used to create a photorealistic fake face. Most of the GAN-created looks confuse certain regions of the face, for example, in the shade of both eyes, ears with a ring. In [75], all these features are exploited, in which acceptable feature sets are constructed to grab them. In [76], the consistency of the biological signs is measured along the spatial and the temporal [77] directions to use various landmark [78] points of the face (e.g., eyes, nose, mouth, etc.) as unique features for authenticating the legitimacy of GANs [79] generated videos/images or capture tampering artifacts [80]. Similar characteristics are also visible in Deepfake videos, which can be discovered by approximating the 3D head pose [81]. In most cases, we see that facial expressions are primarily associated with the head's movements. Habeeba et al. [82] applied MLP to detect Deepfake video with very lees computing power by exploiting visual artifacts in the face region.

Based on the studies that focus on machine learning methods, it is seen that these approaches can achieve up to 98% accuracy in detecting Deepfakes. However, the performance entirely relies on the type of dataset, the selected features, and the alignment between the train set and test set. We obtain a higher result when the experiment uses a similar dataset by splitting it into a specific ratio level, for example, 80% for a train set and 20% for a test set. The unrelated dataset drops the performance that is closed to 50%, which is an arbitrary assumption.

9.2 Deep Learning-Based Methods

In the case of Deepfake detection in images, we find some papers where deep learning-based methods are applied to detect specific artifacts generated by their generation pipeline. Zhang et al. [83] introduced a GAN simulator that replicates collective GAN-image artifacts and feeds them as input to a classifier to identity them as Deepfake. Zhou et al. [84] presented a network for extracting the standard features from RGB data, while a similar but generic resolution was found in [85]. Besides, in [86-89], researchers proposed a new detection framework based on physiological measurement, for example, Heartbeat.

At first, the deep learning-based method was proposed in [90] for Deepfake video detection. Two inception modules, (i) Meso-4 and (ii) MesoInception-4, are used to build their proposed network. In this method, the mean squared error (MSE) between the actual and expected labels is used as the loss function for training. An enhancement of Meso-4 has been done in [91]. In a supervised scenario, [92] shows that deep CNNs [93-95] outperform shallow CNNs. Some methods apply techniques for extracting the handcrafted features [96-97], spatiotemporal features [98-101], common textures [102-103], 68 face landmarks [104-106] with visual artifacts (i.e., eye, teeth, lip movement, etc.) from the video frames and use them as input to the network for detecting Deepfake manipulations. Besides data augmentation [107], super-resolution reconstruction [108], localization strategies in pixel levels [109] are formulated on the entire frame, and maximum mean discrepancy (MMD) loss [110] is applied to discover a more general feature.

Further innovations are achieved by introducing an attention mechanism [111], while promising outcomes are shown in [112-113] by using an architecture, which is called capsule-network (CN). The CN needs a smaller number of parameters to train than very

deep networks. An ensemble learning technique [114] is applied to increase such structures' performance, achieving more than 99% accuracy.

We see that many approaches are proposed to apply frame-by-frame analysis in videos or images to manipulate face and track facial movement to obtain better performance. For example, in [115-120], RNN based networks are proposed to extract the features at various micro and macroscopic levels for detecting Deepfake. Regardless of these exciting results in detection, it is seen that most of the methods lean towards overfitting. The optical flow based technique [121] and autoencoder-based architectures [122-125] are introduced to resolve such problems. A pixel-wise mask [126] is imposed on various models to get the essential depiction of the face's affected area. Fernando et al. [127] applied adversarial training approaches followed by attention-based mechanisms for concealed facial manipulations. In [128], researchers proposed a clustering technique by integrating a margin-based triplet embedding regularization term in their classification loss function. Finally, they converted the three-class classification problem to a two-class classification problem. In [129-130], researchers proposed a data preprocessing technique for detecting Deepfakes by applying CNN methods. The [131] proposed patch and pair convolutional neural networks (PPCNN). In [132], researchers performed an analysis in the frequency domain by exploiting the image latent patterns' richness. ID-revelation [133], a modern approach, was proposed to learn temporal facial features based on a person's movement during talking. A novel feature extraction method [134] was proposed to classify Deepfake images effectively, and in [135], a multimodal approach for detecting real and Deepfake videos. This method extracts and analyzes the similarities between the two audio and visual modalities from within the same video. In [136], a Deepfake detection

method is applied to find the discrepancies between faces and their context by combining multiple XeptionNet models.

In contrast, in [137], a separable convolutional network is used for detecting such manipulations. [138] resorts to the feature extraction process's triplet loss function to better classify fake faces. A patch-based classifier was introduced in [139] to focus on local patches rather than the global structure. In [140-141], the authors extracted features using improved VGG networks. A hypothesis test was performed in [142].

9.3 Statistical Methods

This technique measures average normalized cross-correlation scores between the original and the Deepfake videos leading to their classification as fake or real. Koopman et al. [143] examined the photo response non-uniformity (PRNU) for detecting Deepfakes in video frames. PRNU is called the strongly individualizing noise pattern and is regarded as the digital image fingerprint. The research generates a sequence of frames from input videos and stores them in chronologically categorized directories. In order to preserve and clarify the portion of the PRNU sequence, each video frame is clipped with the same pixel range. These frames are then divided into eight equal groups. It then makes the standard PRNU pattern for each frame using the second-order FSTV procedure. It correlates them with each other by measuring the normalized cross-correlation scores and calculating the differences between the correlation scores and the mean correlation score of each frame. For Deepfakes and original videos, it conducts a t-test [144] on these data, where the t-test shows the statistical difference in the findings on both videos. In [145], researchers extracted a collection of regional features for modeling a basic generative convolutional

structure using the Expectation-Maximization (EM) algorithm. They apply ad-hoc validation to those architectures after the extraction, such as GDWCT, STARGAN, ATTGAN, STYLEGAN, and STYLEGAN2, using preliminary experiments naive classifiers.

Conversely, Guarnera et al. [146] performed a hypothesis test by proposing a statistical framework [147] for detecting the Deepfakes. Firstly, this method defines the shortest path between distributions of original and GAN-created images. Based on the results in the hypothesis test, this distance measures the detection capability. For example, Deepfakes can easily be detected when this distance is increased. And it increases if and only if the GAN provides a lesser amount of correctness. On the other hand, an extremely precise GAN is mandatory to create high-resolution manipulated images that are harder to detect.

9.4 Blockchain-Based Methods

Blockchain technology provides essential features that can be applied for verifying the legitimacy and provenance of digital content in a highly trusted, secured, and decentralized manner. In public Blockchain technology, anyone has direct access to every transaction, logs, and tamper-proof records. For Deepfake detection, public Blockchain is considered as the most right technological solution for verifying videos or images' legitimacy in a decentralized way. Users usually need to explore the origin of videos or images when they are marked as suspected.

Hasan and Salah [148] proposed a Blockchain-based generic framework to track suspected videos' origin to their sources. The proposed solution can trace its transaction

records, even though the material is copied several times. The basic principle says that digital content is considered genuine and authentic when convincingly traced to a reliable source. For Deepfakes, public Blockchain verifies video contents' legitimacy in a decentralized way, as the technology can provide some critical features helpful in proving its authenticity. The following are the main contributions of [148]:

- Presents a generic framework based on Blockchain technology by setting up a proof of digital content's authenticity to its trusted source.
- Presents the proposed solution's architecture and design details to control and administrate the interactions and transactions among participants.
- Integrates critical features of IPFS [149]-based decentralized storage ability to Blockchain-based Ethereum Name service.

Chan et al. [150] proposed a decentralized approach based on Blockchain to trace and track digital content's historical provenance (i.e., image, videos, etc.). In this proposed approach, multiple LSTM networks are being used as a deep encoder for creating discriminative features, which are then compressed and used to hash the transaction. The main contributions of this paper are as follows:

- Using multiple LSTM CNN models, image/video contents are hashed and encoded.
- High dimensional features are preserved as a binary coded structure.
- The information is stored in a permission-based Blockchain, which gives the owner control over its contents

9.5 A Summary

This section tries to show the performance of Deepfake detection methods. Based on 123 related Deepfake detections studies, the output assessment values are first obtained and stored in an Excel document. We split all these used methods into two groups: (i) Model A (DL-based models) and (ii) Model B (non-deep learning-based models) and calculates mean accuracy, AUC, recall, and precision. After then, we count the number of studies that use the same method and the same measurement metrics (precision, accuracy, and recall). And finally, we apply four statistical methods: the minimum, maximum, mean, and standard deviation on these values. In Table 9.1, based on the mean values of accuracy and AUC, DL methods outperform other methods and achieve 89.73% and 0.917, respectively. Besides, we also compare the recall and precision values for both techniques. As shown in Table 10, the results suggest that deep learning-based methods are effective in Deepfake detection.

Table 9.1 *Performance of various detection methods*

Category	Metrics	No	Min	Max	Mean	STD
Deep learning	Accuracy	50	63.15	100.0	89.73	10.08
	AUC	37	0.572	1.000	0.917	0.114
	Recall	5	82.74	100.0	89.47	12.88
	Precision	6	90.55	100.0	88.89	4.948
Non-deep learning	Accuracy	12	85.00	91.07	86.86	11.04
	AUC	12	0.531	1.000	0.909	0.127
	Recall	2	82.74	92.11	89.92	10.15
	Precision	2	90.55	96.40	93.48	4.137

In the last step, we apply a comparative analysis of these two models' performance and achieve an average result. Based on the evaluation of these models using performance measures (Accuracy, AUC, recall, and precision), it is seen that *Model A* outperforms *Model B*. The result is shown in Figure 9.1. As for accuracy, we see that the deep learning models outperform the non-deep learning models, but the AUC and Recall of these two models are remarkably close.

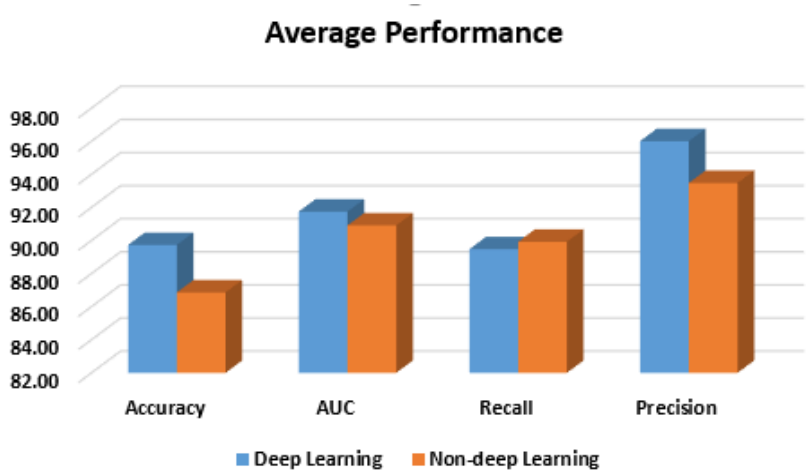


Figure 9.1 *The results of the performance.*

In addition to this, Table 9.2 gives a brief analysis of some prominent Deepfake detection methods.

Table 9.2 *Summary of some popular Deepfake detection methods*

Methods	Classifiers	Features	Deal with	Dataset Used
Intra-frame and temporal inconsistencies [115]	A convolutional LSTM comprised of a CNN and an LSTM	CNN extracts frame-level features and passes to LSTM for sequential analysis to build sequence descriptor useful for classification	Videos	A collection of 600 videos was gathered from several websites.

Table 9.2 (continued).

Methods	Classifiers	Features	Deal with	Dataset Used
Face warping artifacts [97]	VGG16, ResNet50, ResNet101 and ResNet152	Artifacts are detected using CNN models based on the resolution discrepancy between the distorted face area and its adjacent context.	Videos	- UADFV [81] dataset that contains 49 authentic and 49 fake videos. - DeepfakeTIMIT [151] dataset.
Eye blinking [96]	An LRCN comprised of a CNN and an RNN	Use LRCN to distinguish open and closed eye states with previous temporal knowledge as human eye blinking has a solid temporal correlation to prior states.	Videos	A total of 49 interviews and presentation videos with corresponding created Deepfake videos.
MesoNet [90]	CNN	Introduces two CNN-based architectures (Meso-4 and MesoInception-4) for face manipulation detection that exploit features at a mesoscopic level. The experiments validate a very effective detection rate with more than 98% for Deepfake 95% for Face2Face.	Videos	(i) Deepfake generated from online videos and (ii) Face2Face created by FaceForencies [70].
Two-stream network [80]	CNN and SVM	(i) In the first stream, a CNN-based face classification network is trained to capture tampering artifacts. (ii) In the second stream, a Steganalysis feature-based triplet network is trained to ensure patches from the same image are close in the embedding space, and an SVM trained on the learned features classifies each spot.	Images	Used Face Tampering dataset created by own. There are 1005 Two images for each tampering technique (2010 tampered images) and 1400 authentic images for each subset in this dataset.
Spatiotemporal features [117]	RCN	- A two-step process: cropping and alignment of faces from video frames, followed by manipulation detection over the preprocessed facial region using RCNN. - Improves performance in detection accuracy than the previously reported results by up to 4.55%	Videos	FaceForencies++ [92] dataset that contains 1000 videos.
GAN forensics face detection [79]	DNN, GANs	- Build train datasets using DC-GAN [152] and PG-GAN [67] that can be adapted to test datasets. - Build a DL network based on face recognition networks to extract face features using VGGFace [153] through the VGG-Net from GANs generated images.	Images	ImagesNet, CelebA face dataset with 292,599 images and validated the model with the author's created dataset.

Table 9.2 (continued).

Methods	Classifiers	Features	Deal with	Dataset Used
ForensicTransfer (FT) [122]	CNN	- Tackles two challenges: (i) It can rapidly adjust to other kinds of manipulation without the need for re-training the entire network and (ii) handles scenarios where only a small number of fake examples are available during training. - Achieves up to 80-85% in terms of accuracy.	Images	- FaceForensics [70] dataset. - To validate the model the authors used Progressive-GAN [152], Cycle-GAN [67], Glow [153], and StarGAN [154].
Multi-task learning [123]	CNN	- Simultaneously carries out detection and segmentation of manipulated facial images and videos. - Tackle seen and unseen attacks, as well as facial reenactment attacks and face-swapping attacks.	Videos/ Images	Two datasets used: FaceForensics [70] and FaceForensics++ [92].
CapsuleNetwork [19]	CNN	- Latent features extracted by the VGG-19 network are fed into the capsule network for classification. - A dynamic routing algorithm is used to route the outputs of 3 convolutional capsules to two output capsules, one for fake and another for real images, through a few iterations.	Videos/ Images	Four datasets: the Idiap Research Institute replay-attack [40], Deepfake [12], FaceForensics [34], and Computer-generated image dataset [41].
FakeCatcher [76]	CNN	It creates a high dimensional feature vector to aggregate the temporal and spatial consistency of the biological signals for the classification task and achieves 99.39% accuracy.	Videos	FaceForensics [70] dataset, authors created the Deepfake dataset.
Eye, teeth and facial texture [75]	Logistic regression and neural network	- Exploit facial texture differences and missing reflections and details in the eye and teeth areas of Deepfakes. - Logistic regression and neural networks are used for classifying.	Videos	Video dataset gathered from YouTube.
PRNU Analysis [143]	PRNU	- Analysis of noise patterns of light-sensitive sensors of digital cameras due to their factory defects. - Explore the differences of PRNU patterns between the authentic and Deepfake videos because face swapping is believed to alter the local PRNU patterns.	Videos	Using DeepFaceLab [157], the authors created a dataset that contains 10 authentic and 16 Deepfake videos.

CHAPTER X – DATASET DESCRIPTION

We found various Deepfake datasets used in numerous studies for training and testing purposes. With the help of these datasets, Deepfake detection has made tremendous progress. There are just a few professional actors and a restricted number of scenes in most of these real videos. Experts use Deepfake software to create the fake realistic videos. A variety of datasets were used in these investigations, as shown in Table 10.1. In this chapter, we describe three widely used datasets, including FaceForensics++ (FF++), DeepFake Forensics (Celeb-DF), and Deepfake Detection Challenge (DFDC). These have been used in our experiments. In addition, we introduce our newly generated dataset, called as Versatile Deepfake dataset (VDFD), which will be publicly available soon.

Table 10.1 *The List of Deepfake datasets*

Database Name	**#Actors**	**# Deepfake Videos**
FaceForensics [70]	977	1000
FaceForensics++ [92]	977	1000
DeepfakeDetection (DFD) by Google [159]	28	3000
DeepFake Forensics (Celeb-A) [160]	10,177	202,599 (images)
DeepFake Forensics (Celeb-DF V1.0+V2.0) [161]	13+59	795+ 590
Deepfake Detection Challenge (DFDC) [162]	66	5214
UADFV [81]	-	49
Deepfake-TIMIT [151]	64	620
DeeperForensics-1.0 [163]	100	60,000
WildDeepfake [164]	100	707
MANFA [158]	-	8950 (images)
SwapMe and FaceSwap [80]	-	1005 (images)
Deep Fakes [76]	-	142
Fake Faces in the Wild (FFW) [165]	-	150
FakeEt [166]	40	811
FaceShifter [167]	-	5000 (images)

Table 10.1 (continued).

Database Name	#Actors	# Deepfake Videos
Deepfake [89]	50	175
Swapped Face Detection [79]	86	420,053 (images)
Versatile Deepfake Dataset (VDFD)	<1200	5000 (aprox.)

10.2 Face Forensics++ (FF++)

The dataset has been collected by the Visual Computing Group (VGG), which is an active research group on computer vision, computer graphics, and machine learning. There are 1000 real videos included in this dataset that was downloaded from YouTube. Then they were manipulated by using three popular state-of-art manipulation techniques (i.e., Deepfake, FaceSwap, and Face2Face) [92].

10.3 DeepFake Forensics (Celeb-DF)

Celeb-DF [161] is a large-scale and challenging Deepfake Video Dataset generated for creating and assessing Deepfake detection algorithms. This dataset has a total of 5,639 Deepfake videos with over 2 million frames. The real source videos are based on YouTube video clips of 59 celebrities of different genders, ages, and ethnic groups. In order to generate Deepfake videos, a superior Deepfake Synthesis approach is used that dramatically improves the overall visual quality of these synthesized videos compared to prior datasets, with lower visual artifacts.

10.4 Deepfake Detection Challenge (DFDC)

Datasets from the third generation include bigger frames and videos than those from the second generation, as well as higher quality and agreement among the people in the dataset [162]. The data came from 3,426 individuals, each of whom had an average of 14.4 videos. The majority of the videos were recorded in 1080p, and there were 48,190 total videos with an average duration of 68.8 seconds each. This amounted to 38.4 days' worth of material. There are approximately 25 terabytes of data in the dataset.

10.5 Versatile Deepfake Dataset (VDFD)

We introduce a new challenging and large-scale Deepfake video dataset for Deepfake detection. To create the Deepfake videos, we have collected source videos from Youtube. We consider various regions globally, including South Asian, Asian, African, American, and European, Middle Eastern parts, since we are working on a multi-dimensional and global dataset and generate a versatile and diverse dataset. The quality of the original and Deepfake videos are in high definition, for example, 1080p. Our ongoing effort are:

- Multi-regional videos
- High-quality videos
- 10000 Deepfake videos with diverse genders, ages, skin ton, etc.
- Average video length 30 seconds with 50 frames per second

CHAPTER XI – LIMITATIONS AND CHALLENGES

There are several limitations and difficulties with the existing techniques to detect Deepfake, even though the results are impressive. In this chapter, we'll talk about the drawbacks and opportunities of using Deepfakes. In addition, we formulate a hypothesis testing problem for detecting Deepfake.

11.1 Challenges in Deepfake Generation

Using an incorrect setup or a model that hasn't been properly trained may cause the facial region to flicker even more, resulting in leaking color around the face area. The following flaws are found during the Deepfake generation if the configuration is wrong.

- **Blurry**: There are two significant reasons behind the blurriness of Deepfake videos:
    - First, the new face needs to blend well with the rest of the images. Therefore, filters are applied, which will blur the face slightly.
    - Second, many low-budget productions use low-resolution images to learn the encoder.
- **Skin tone:** The skin tone seems unnatural on specific swapped faces.
    - The solution to this problem is to include individuals with common skin tones, hairstyles, and facial to exchange shapes.
- **Double eyebrow:** If the mask or merging is not done correctly, we see two eyebrow sets, one set from the new face and the other from the original face, while swapping a face with the original face.

- **Double chin**: A double chin may appear. However, if you do not know the original person, it is harder to treat it as usual.
- **Spatial inconsistency:** We can compare the face with other body regions when trying to detect facial anomalies.
    - The texture of the skin and arm's smoothness does not match the face.
- **Flicking:** Video frames are created individually, frame after frame, which can provide video frames of distinct tones, illumination, and shadow compared to the last frame. Flickering happens when it is played back.
    - The quality of the swapped frames is so poor that the wrong frames are removed manually or automatically.
- **Shimmering:** The skin shimmering and unnatural tone changes while the head moves as we play at the videos below at 0.25 speed.
- **Border**: Border area of the face where it merges with the original. However, the artifacts are less visible or not observable for more professional projects.

Face swapping is done manually, as seen in Figure 11.1, which demonstrates that someone has pasted pictures onto her face by brute force. To illustrate, let's use the case of two ladies (1 and 2). It's evident in image 4 that someone tried to copy and paste the face of 2 onto the face of 1. During the merging process, it was discovered that the facial structures of the two actors were different, and the face cutout was found to be considerably bigger than the target, making it seem like someone used a paper mask of 2 on 1. After that, we went in the other direction. It was decided to go with a smaller cutout for image 3 rather than go with a bigger one. After that, some of the cutout's corners were removed

with the use of a mask to better integrate the piece together. The result was not spectacular, but it was an improvement over the previous job. However, there was a surprising variation in skin tone along the border. The kernel's opacity at the border was decreased, allowing the newly formed face to better blend in (see picture 5). In terms of color tone and intensity, there was still a discrepancy, therefore we changed that criteria to match picture 6. Even if our effort was not perfect, it wasn't awful either.

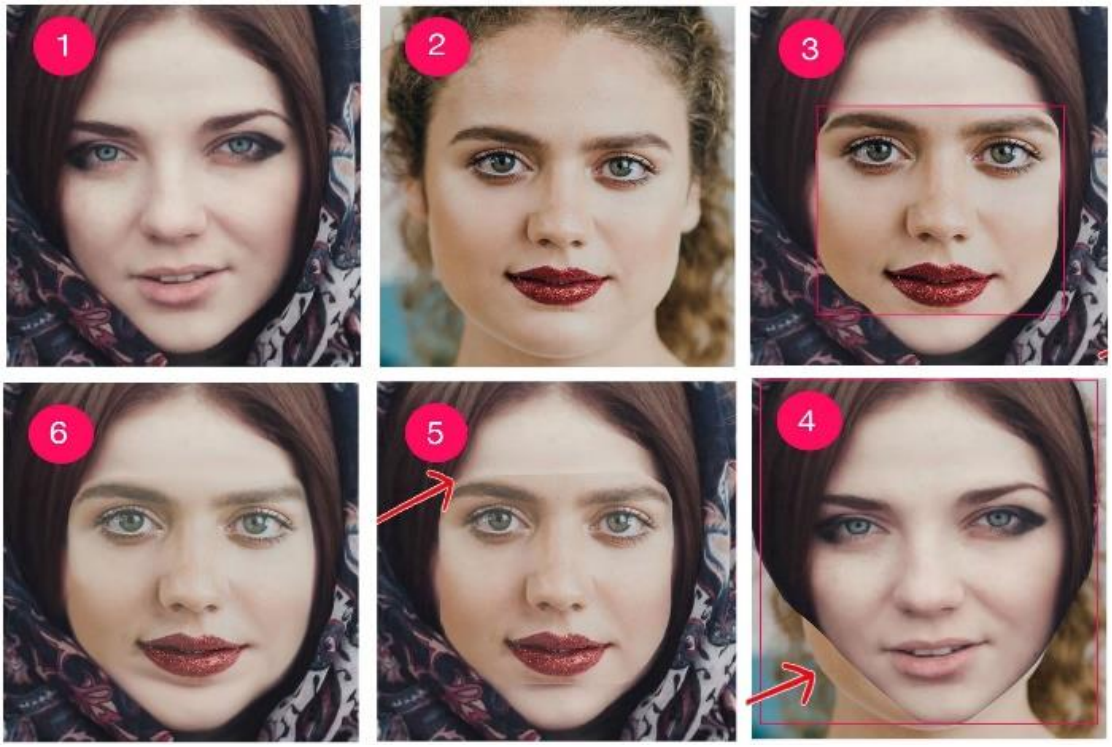


Figure 11.1 *Example of lousy implementation with the wrong configuration.*

11.2 Challenges in Deepfake Detection

In this section, a hypothesis testing problem is formulated for detecting Deepfake to classify an image as genuine or GAN-generated through a generalized robust statistics framework [24].

11.2.1 Problem Formulation and Hypothesis Test

The distribution of authentic images is $\mathbb{P}_X$. The placement generated by the GAN is $\mathbb{P}_{g(Z)}$. The *n* pixels of the input image are $Y_1, Y_2, Y_3, ..., Y_n$. Accordingly, the test is

$$H_1 \coloneqq Y \sim \mathbb{P}_{\hat{g}(Z)} \quad \text{-- GAN Generated}$$

Where $\hat{g}$ is the closest to the actual distribution $\mathbb{P}_X$ that can be defined by,

$$\hat{g} = \arg\min_{g \in \mathrm{G}} L(\mathbb{P}_X, \mathbb{P}_{g(Z)}) \tag{1}$$

Here, $L: \mathcal{M}(\mathbb{R}^d)\ x\ \mathcal{M}(\mathbb{R}^d) \rightarrow \mathbb{R} \geq 0$ is used to measure the distance between the two distributions, and GANs are considered as a generalization of the robust statistics framework [25] with the true distribution $\mathbb{P}_X$ that is a slightly perturbed version of a generated distribution $\mathbb{P}_{g(Z)}$ under the distance measure $L(.,.)$ and can be defined as an error, which is the minimum distance between the generated and input distributions for a particular GAN and input distribution.

$$OPT \coloneqq \inf_{g \in \mathrm{G}} L(\mathbb{P}_X, \mathbb{P}_{g(Z)}) \tag{2}$$

11.2.2 Error Bounds

The following error bounds [26] are used for hypothesis testing where *C* is the Chernoff information, which can further be bounded in terms of total variational (TV) distance [27].

$$Neyman - Pearson\ error: \beta_n^{\mathcal{E}} = \exp\left(-nD(\mathbb{P}_X || \mathbb{P}_{\hat{g}(Z)})\right) \tag{3}$$

$$Bayesian\ error: P_e^{(n)} \leq \exp\left(-nC(\mathbb{P}_X || \mathbb{P}_{\hat{g}(Z)})\right) \tag{4}$$

$$C(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)}) \geq \frac{1}{2}\log\left(1 - TV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right) \quad (5)$$

Now, derive bounds for the Neyman-Pearson and Bayesian error probabilities for the chosen L function implementations of the GAN using the Kullback-Leibler (KL) divergence [28-29], the TV distance and the Jensen-Shannon (JS) divergence [30] as follows:

- KL Divergence: Given,

$$L(\mathbb{P}_X, \mathbb{P}_{g(Z)}) = D(\mathbb{P}_X||\mathbb{P}_{g(Z)}) \geq OPT \quad \forall_g \in G$$

The Neyman-Pearson is obtained by,

$$\beta_n^{\mathcal{E}} = \exp\left(-nD(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})\right) \leq \exp\left(-nOPT\right) \quad (6)$$

And the Bayesian can be obtained using the bound on Chernoff information in (5),

$$P_e^{(n)} \leq \exp\left\{\frac{n}{2}log\left(1 - TV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right)\right\} \quad (7)$$

and from reverse Pinsker's inequality [31], assuming $P_g^x = min_x \mathbb{P}_{\hat{g}(Z)}(x) > 0$,

$$\begin{aligned} P_e^{(n)} &\leq \exp\left\{\frac{n}{2}log\left(1 - \frac{P_g^*}{4}D(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})\right)\right\} \\ &\leq \exp\left\{\frac{n}{2}log\left(1 - \frac{P_g^*}{4}OPT\right)\right\} \\ &= (1 - \frac{P_g^*}{4}OPT)^{n/2} \end{aligned} \quad (8)$$

- TV Distance: Given,

$$L(\mathbb{P}_X, \mathbb{P}_{g(Z)}) = TV(\mathbb{P}_X||\mathbb{P}_{g(Z)}) \geq OPT \quad \forall_g \in G$$

The Neyman-Pearson is obtained using Pinsker's inequality [32],

$$\beta_n^{\mathcal{E}} = \exp\left(-2nTV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right) \leq \exp\left(-2n(OPT)^2\right) \qquad (9)$$

And the Bayesian is obtained from the bound on Chernoff information in (5),

$$\begin{aligned} P_e^{(n)} &\leq \exp\left\{\frac{n}{2} log\left(1 - TV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right)\right\} \\ &\leq \exp\left\{\frac{n}{2} log(1 - OPT^2)\right\} \\ &= (1 - OPT^2)^{n/2} \end{aligned} \qquad (10)$$

- JS Divergence: From the bound-on JS divergence,

$$2TV\left(\mathbb{P}_X, \mathbb{P}_{g(Z)}\right) \geq JS(\mathbb{P}_X||\mathbb{P}_{g(Z)}) \geq OPT \quad \forall_g \in G$$

The Neyman-Pearson is obtained from Pinsker's inequality and the above equation.

$$\beta_n^{\mathcal{E}} = \exp\left(-2nTV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right) \leq \exp\left(-\frac{n}{2}OPT^2\right) \qquad (11)$$

And the Bayesian is obtained from the bound on Chernoff information in (5),

$$\begin{aligned} P_e^{(n)} &\leq \exp\left\{\frac{n}{2} log\left(1 - TV(\mathbb{P}_X||\mathbb{P}_{\hat{g}(Z)})^2\right)\right\} \\ &\leq \exp\left\{\frac{n}{2} log\left(1 - \frac{OPT^2}{4}\right)\right\} \\ &= \left(1 - \frac{OPT^2}{4}\right)^{n/2} \end{aligned} \qquad (12)$$

The error bounds can be concise in Table 11.1. It is seen that an increase of OPT results in the detection of Deepfakes makes it exponentially more accessible in the

Neyman-Pearson case, and polynomially so in the *Bayesian* fact (i.e., the GAN used is less accurate). The bound decays exponentially with the resolution *n*. Thus, if it requires an extremely high resolution to trust images, an extremely precise GAN would be needed to go undetected.

Table 11.1 *Summary of the Error Bounds*

L-function	**Neyman-Pearson Bound $\beta_n^{\varepsilon} \leq$**	**Bayesian Bound $P_e^{(n)} \leq$**
KL divergence	$\exp(-nOPT)$	$\left(1-\frac{P_g^*}{4}OPT\right)^{n/2}$
TV distance	$\exp(-2n(OPT)^2)$	$(1-OPT^2)^{n/2}$
JS divergence	$\exp\left(-\frac{n}{2}OPT^2\right)$	$\left(1-\frac{1}{4}OPT^2\right)^{n/2}$

11.2.3 Epidemic Threshold Theory and Deepfake

The SIR (susceptible-infected-recovered) model for epidemics can measure the dangers of Deepfake as it is diffusing hastily in social networks with the spread of dynamic forces like diseases. The epidemic threshold exemplifies the critical level λ_c for effective spreading rate, $\lambda = \frac{\beta}{\gamma}$ (where, β is the probability of transmission from an infected to a susceptible node, and γ is the probability of recovery, i.e., probability of correctly detecting the Deepfake $1 - P_e$). Above which, a global epidemic occurs, and the spreading cannot be controlled. Let the bound for $P_e^{(n)}$ be exp $(-nf(OPT))$, Thus,

$$\lambda \leq \frac{\beta}{1 - \exp\left(-nf(OPT)\right)} \tag{13}$$

$$f(OPT) \geq -\frac{1}{n}\ln\left(1 - \frac{\beta}{\lambda_c}\right) \tag{14}$$

The Deepfake can be locally confined when *f(OPT)* is higher than the expression on the right that depends on *n* and the network structure and the condition for containment $\lambda \leq \lambda_c$. Thus, the network will be less easily fooled by a weaker Deepfake generation system since a higher value of *OPT* assures higher robustness to the global spread of misinformation.

CHAPTER XII – PROPOSED DEEPFAKE DETECTION METHODOLOGY

Researchers and professionals have proposed and developed various ways to deal with Deepfake, a new layer of video manipulation technologies that poses new difficulties. This chapter offers two Deepfake detection methods: (i) DL-based method and (ii) ML-based method.

Before describing our proposed methods, we provide a brief description of deep ensemble learning techniques.

12.1 Deep Ensemble Learning

The ensemble technique combines a list of sub-models (also known as base-learners) to form an ideal perceptive model where each sub-model produces the final output. The newly generated combined model is called a **meta-model**. The two commonly used methods are [169].

1. *Stacking ensemble (SE)*: The SE takes the output of the base-learners and uses them as input to train the meta-learner so that the model learns how to best map the base learner's decisions into an enhanced result (see Figure 12.1).

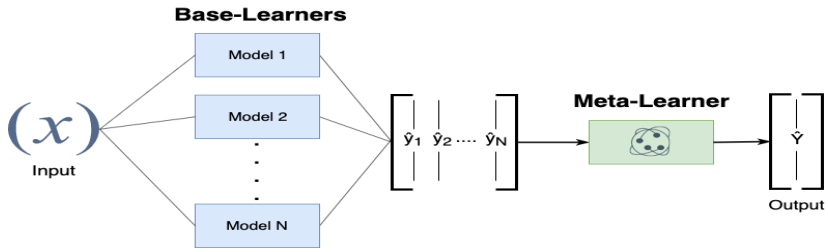


Figure 12.1 *Stacking combines multiple predictive models to generate a new combined model.*

2. *Randomized weighted ensemble (WRE)*: In the WRE technique, each base-learner is weighted by a value based on their performance evaluated on a hold-out validation dataset. The model receives a higher weight if it performs better than others. In other words, this technique is just optimizing weights that are used for weighting all the base learner's output and taking the weighted average (see Figure 12.2).

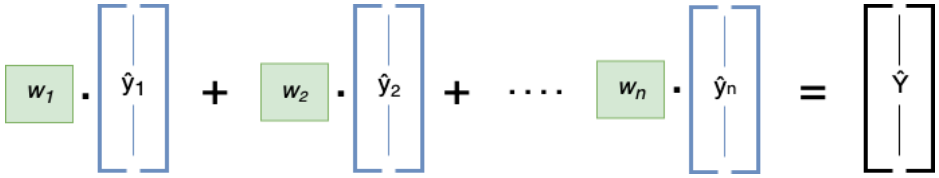


Figure 12.2 *Randomized Weighted Ensemble weights the predictions of each base model generating a combined prediction.*

12.1.1 DeepfakeStack: A Deep Ensemble-based Learning Technique

This section applies a deep ensemble learning technique, namely, DeepfakeStack [170], by evaluating several DL-based state-of-the-art models. The idea behind the DeepfakeStack is based on training a meta-learner on top of pre-trained base-learners and offers an interface to fit the meta-learner on the predictions of the base-learners and shows how ensemble technique performs classification task. The architecture of the DeepfakeStack involves two or more base learners, called *level-0 models*, and a meta-learner called *level-1 model* that combines the predictions of these *level-0 models*. The *level-1 model* is trained on the predictions made by base models on out-of-sample data. That is, data not used to train the base models is fed to the base models, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the meta-model.

12.1.1.1 Data Analysis and Preprocessing

To achieve the best performance of the used DL models, we need to preprocess the dataset by applying data analysis. Below the idea of how this dataset is organized as follows:

- In this experiment, we have used 200 real and 200 fake to make the balanced dataset. After then, we separate each of the videos based on its category under the directories (e.g., Original, Deepfakes).
- For each video, each folder is created that contains all extracted image sequences. For example, if the video file's name is '485.mp4' then we create a directory with the same name '485' where it contains all the frames of '485.mp4' for each of the original videos and we have followed the same procedure for Deepfakes data.
- We do not consider the entire video sequence instead; we take only 100 frames from each video to reduce the computational time.

As the main goal is to detect the manipulated face images, we are concerned only in the face area. So, it is an innovative idea to ignore all others like the body, background, etc. Therefore, we track the face in each of the images and feed them into the classifier.

12.1.1.2 Overview of DeepfakeStack

The DeepfakeStack provides a way of combining a set of k base-learners, C_1, C_2, …, C_k, to produce an enhanced classifier C_*. D, for a given dataset, splits it into k training sets, D_1, D_2, …, D_k, and uses D_i to build the base-learner, C_i. For classifying a new (unseen) data tuple, the DeepfakeStack returns a class prediction based on the votes of these base-

learners. Simply, for a given tuple *X* to classify, it accumulates the class label predictions obtained from the base-learners and yields the class in the majority. The below coding snippet shows the algorithm of DeepfakeStack.

Algorithm DeepfakeStack Classifier (DFSC)

Input: Training data, D = $\{x_i, y_i\}_{i=1}^{m} (x_i \in \mathbb{R}^n, y_i \in Y)$

Output: Ensemble-based classifier, DFSC

1. Generate base learners/classifiers, $c_1, c_2, \dots, c_T$
2. **foreach** base learner in C
3. Learn base learner c_t based on D
4. **end foreach**
5. Create new dataset from D
6. **for** i ← 1 to m **do**
7. Create new dataset, D' that contains $\{x_i', y_i\}, where\ x_i' = \{c_1(x_i), c_2(x_i), \dots, c_T(x_i)\}$
8. **end for**
9. Step 3: Learn meta learner (2nd level classifier)
10. Learn a new classifier C' based on the newly created dataset
11. **return** DFSC$(x) = C'(c_1(x), c_2(x), \dots, c_T(x))$

Listing 12.1 *Algorithm DeepfakeStack Classifier (DFSC)*.

The working procedure of DeepfakeStack can be split into two sections: (i) Base-Learners Creation, (ii) Stack Generalization.

- *Base-Learners Creation*: As we defined this work to solve the binary classification problem, we need to fix the label 0 for real and 1 for Deepfake and, measure both accuracy and categorical log loss. Once we are done with data analysis it is very crucial to decide what kind of models might work for this data. It is an innovative idea to prefer any CNN-based architecture as we have an image dataset. In addition to this, selecting picture-perfect

factors is a huge challenge, which may include the number of layers, number of units, dropout rates, activations, learning rates, etc. Contemplating all, we can adjust to fine-tune any architecture relevant to the model that has already been trained and assessed on a similar dataset. For example, CNN-based networks have already been trained and tested on the ImageNet dataset [171]. To adapt to any of these architectures, the dataset needs to be preprocessed and establish an environment accordingly. These CNN-based networks were trained with normalized images of equal size (224x224) on RGB images. Therefore, before feeding into the model, we must look after that the dataset should be normalized and preprocessed into the same size. In this experiment, we initialize 7 DL models (e.g., XceptionNet, MobileNet, ResNet101, InceptionV3, DensNet121, InceptionReseNetV2, DenseNet169) with ImageNet weights and apply the transfer learning by replacing only the topmost layer with 2 outputs with SoftMax activation. We consider these architectures as base-learners, and to train these models, we follow Greedy Layer-wise Pretraining (GLP) [172] technique. The GLP uninterruptedly adds a new hidden layer to a model and refit the model. Besides, it permits the newly added model to learn the inputs from the existing hidden layer, while keeping the weights for the existing hidden layers fixed. This procedure is called “layer-wise” as the model is trained one layer at a time and is referred to as “greedy” because of this layer-wise method can resolve the problem of training a deep network.

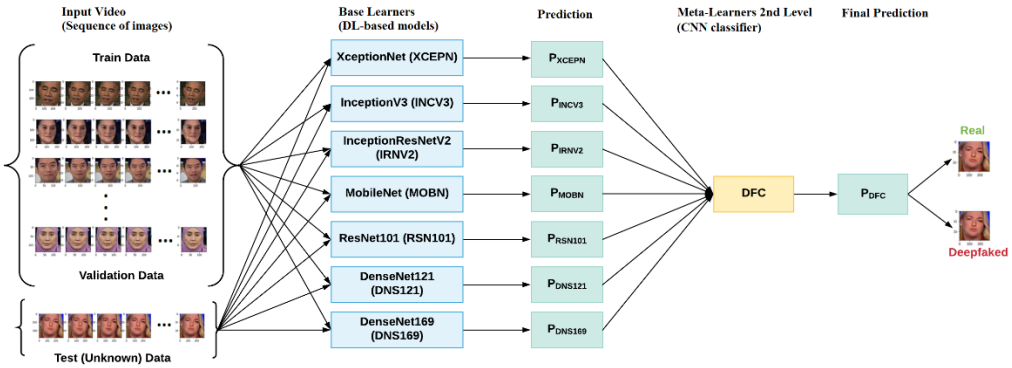


Figure 12.3 *Overview of DeepfakeStack*

- *Stack Generalization*: Once the base-learners are ready, we need to define the meta-learner. In the case of meta-learner, we create a CNN based classifier, namely, DeepfakeStackClassifier (DFC), and embed in a larger multi-headed neural network to learn to obtain the best combination of the predictions from each input base-learner. This approach permits the stacking ensemble to be treated as a single large model and the benefit is that the outputs of these base-learners are provided directly to the meta-learner. In addition to this, it makes it possible to update the weights of the base-learners as well as the meta-learner model (see Figure 12.3). The input layer of each base-learner is used as an individual input head to the DFC model. This means k copies of input data are fed to the DFC model, where k stands for the number of input models (base-learners) and merge the output of each of these models. In this experiment, a simple concatenation merge has been used, where a single 14-element vector is formed from the two class-probabilities predicted by each of the 7 base-learners. To interpret

this “input” to the meta-learner, we define a hidden layer in conjunction with an output layer that makes its probabilistic prediction.

12.1.1.3 Results and Discussion

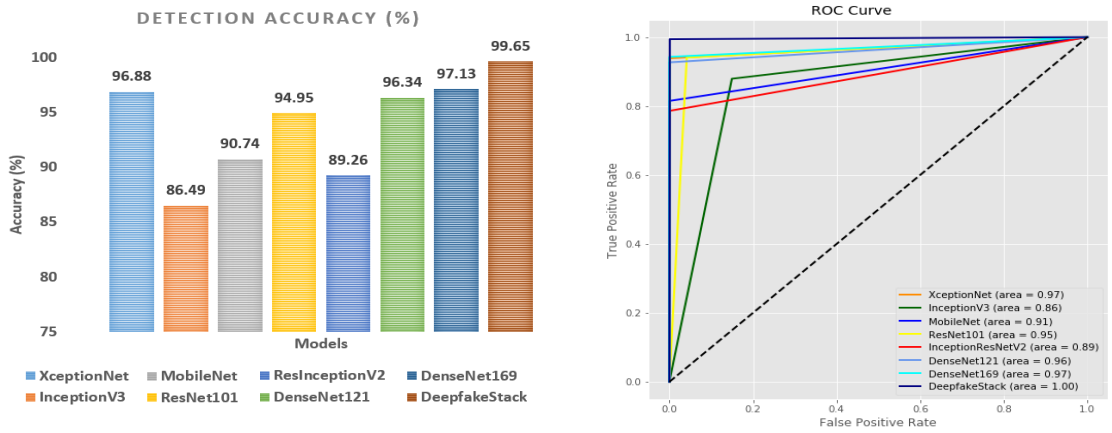
After defining the DFC model, we fit it directly on the holdout test dataset for 300 epochs. Note that the weights of the base-learners will not be updated during the training since their trainable property is set to *False* (i.e., not trainable) while defining them. Only the weights of the new hidden and output layer will be updated. After successful fitting, we use this DFC model to predict unseen data and expect the DFC to perform better than any individual sub-model (base-learner). For comparison, the performances are shown in Table 12.1.

Table 12.1 *Performance of the DeepfakeStack and individual DL model (base learners)*

Model	**Precision**		**Recall**		**F1-score**		**Accuracy**	**AU ROC**
	0	***1***	***0***	***1***	***0***	***1***		
XCEPN	0.94	1.00	1.00	0.94	0.97	0.97	96.88	0.976
INCV3	0.88	0.95	0.85	0.88	0.86	0.87	86.49	0.866
MOBN	0.84	1.00	1.00	0.81	0.92	0.90	90.74	0.911
RSN101	0.94	0.96	0.96	0.94	0.95	0.95	94.95	0.954
IRNV2	0.82	1.00	1.00	0.79	0.90	0.88	89.26	0.899
DNS121	0.93	1.00	1.00	0.93	0.96	0.96	96.34	0.969
DNS169	0.95	1.00	1.00	0.94	0.97	0.97	97.13	0.971
DFC	**0.99**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**99.65**	**1.000**

The results of the overall accuracy of each DL model using the same parameters are summarized in Figure 12.4 (a), where it is seen that the DeepfakeStack (DFC) model obtains the best performance. The DFC achieves an accuracy of 99.65%. Based on the

experiment, we can say that the DFC model now learned to detect the Deepfake manipulated videos/images.



(a) Accuracy (b) ROC Curve

Figure 12.4 *Performance DFC and other state-of-art models.*

To summarize, a ROC curve is produced per model by varying the threshold values from 0 to 1 which helps to visualize the tradeoff between sensitivity and specificity and recognize how well-separated our data classes are. As shown in figure 12.4 (b), the ROC tells us how good the model is for classifying the two classes: Original and Deepfake. The area covered by the curve is the area between the colored line and the axis where each color line represents an individual model/classifier (i.e., the blue line represents the DFC model). The bigger the area covered, the better the models are at classifying the given classes. In other words, the closer the AUCROC is to 1, the better. Based on the experiment, we can see that the DFC achieves an AUROC of 1.0, which shows that the positive and negative data classes are perfectly separated, and the model is as efficient as it can get.

12.2 Machine Learning-Based Methods

Deep learning (DL)-based approaches minimize the human effort in feature engineering; however, as much is done automatically, it increases the complexity of understanding and interpreting the model. Interpretability is a significant disadvantage of DL-based methods as the model is too difficult to understand with its high nonlinearity and interactions between inputs. Typically, a tradeoff between accuracy and interpretability occurs when analyzing any classical machine learning (ML) method [96]. DL methods are languid to train and require many computing resources, making them resource-intensive because of their complexity, a large number of layers, and large volumes of data. On the other hand, ML methods are easier to evaluate and understand.

Explainability and Interpretability are significant disadvantages of DL-based methods. Sensitive applications, such as industrial robots that run with human beings or self-driving vehicles, are essential for protection. In our case, it would improve trust in the model and simplify the future manual inspection if we explained the exact cause of a video as a Deepfake.

This research aims to study the explainability and interpretability of Deepfake detection models in depth. We address the following questions.

1. ***Can Deepfake Detection be explained in an interpretable manner?*** In order to clarify which parts of an image are predicted to be manipulated, some Deepfake detectors generate different face localized information along with prediction. Our research indicates that no one has applied ML-based methods for interpreting this issue. The main reason for looking into

this is that ML-based methods can be applied to a variety of DL architectures.

2. ***Do DL models and ML-based methods use different features in Deepfake detection?*** Since various DL models have various detection accuracy, they can focus on unique features or on the same features but in diverse ways. Applying feature engineering with ML methods will explain the distinctive features that each method values for its prediction and, most importantly, why specific methods perform better than others.
3. ***How do we develop the model or dataset with additional knowledge?*** Important insights into the classification model rations are provided in explanations. We can study whether we can use these ideas implicitly by creating improved models or using them specifically during the training process.
4. ***How thorough can we go with our explanation? Is it possible to be intuitive while still maintaining a certain degree of completeness?*** There is a tradeoff between completeness and intuitivity of an explanation. Due to their inherent complexity, DL methods are languid to train and require a lot of computational power, making them very time- and resource-intensive [173-174]; therefore, a complete explanation, in terms of all these cases, would be useless since impossible to understand. A too intuitive interpretation, on the other hand, cannot provide enough knowledge for our purposes. We will choose a proper balance between these two properties during the work.

The above context motivated us to, in this research, experiment with and evaluate a classical ML method in detecting Deepfakes. The process and the results are described below:

- Create a unique set of features by combining HOG, Haralic, Hu Moments, and Color Histogram features.
- Lessen the data's complexity and making patterns recognizable by splitting a task into two stages: object detection and object recognition.
    - The object detection phase scans an entire image and finds all possible objects.
    - The object recognition step finds relevant objects.
- The advantages of the ML method
    - Provide better understandability and interpretability of the model.
    - Reduce training time and achieve the same level of performance. Using FF++, DFDC, Celeb-DF, and VDFD datasets with the same amount of train and test samples, the ML methods take between several seconds and a couple of hours, while the most advanced DL algorithm instance, ResNet, takes nearly two weeks.

12.2.1 Data Preprocessing and Feature Selection

We use FF++, DFDC, Celeb-DF, and VDFD (our newly proposed) datasets to conduct the experiment and evaluate our proposed technique. We applied the following preprocessing and analysis methods to reach the most excellent performances for ML models:

- We use a total of 400 videos with 200 real faces and 200 fake faces.
- From each video, we randomly select 200 frames to reduce the cost of computing.
- Track and extract each image's face and feed them into the classifiers using the 'dlib' Python package.

In ML, feature extraction and feature selection are significant problems. Accurate or discriminating features are mandatory to enhance the model's performance because undesirable computation and accuracy loss may result from irrelevant features. A more comprehensive training dataset will usually get reasonable classification accuracy as the number of features increases. For a fixed data set, accuracy increases to a point with a variety of features, and the accuracy of classification tends to decline as the number of features increases. In image processing and pattern recognition, feature extraction involves obtaining new features with higher-level info of an image, for example, color, shape, texture, etc. And feature selection reduces the input data dimension to find out the most appropriate features. Also, feature selection reduces the time complexity of the training process [175]. To achieve these goals, we used the following feature extraction and selection techniques:

- *Haralic Texture Features (HTF)*: A new type of global feature descriptor [176]. The texture forms an essential aspect of human visual perception, defining the consistency of designs and colors in an object/image. To classify objects in an image based on texture, we have to look for the consistent spread of patterns and colors on the object's surface. For example, Rough-Smooth, Hard-Soft, Fine-Coarse are some of the texture

pairs. The HT is used to quantify an image based on texture. The fundamental concept involved in computing HT features is the Gray Level Co-occurrence Matrix (GLCM) that uses the adjacency concept in images. The basic idea is that it looks for pairs of adjacent pixel values that occur in an image and keeps recording it over the entire image—four types of adjacencies, including Left-to-Right, Top-to-Bottom, Top-Left-to-Bottom-Right, Top-Right-to-Bottom-Left. From the four GLCM matrices, 14 textural features are computed that are based on some statistical theory. Usually, the feature vector is taken to be of 13-dim as computing 14th dim might increase the computational time. In this experiment, we followed the same concept and have the feature vector is of length 13, which is implemented in Listing 12.2.

```
def ft_haralick(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  haralick = mahotas.features.haralick(image).mean(axis=0)
  haralick = scaler.fit_transform(haralick.reshape(-1, 1))
  haralick = haralick.flatten()
  return haralick
```

Listing 12.2 *Implementing Haralick trait recalculation.*

- *Histogram of Oriented Gradients (HOG)*: A feature descriptor [177-179] used to extract features from an image to focus on the structure or the shape of an object in an image. How is this different from the edge features we extract for images? In the case of edge features, we only identify if the pixel is an edge or not. HOG can provide the edge direction as well. This is done by extracting the gradient and orientation of edges, where these orientations are calculated in ‘localized’ portions. This means that the complete image

is broken down into smaller regions, and for each area, the gradients and orientation are calculated. Finally, the HOG generates a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values. In this experiment, based on the input image size 224 x 224 x 3, the HOG feature descriptor generates six output feature vectors of lengths: 109, 117, 133, 228, 717, 2296. The whole thing is graphically presented in Figure 12.5, and the code of the function itself performing this task is shown in Listing 12.3.

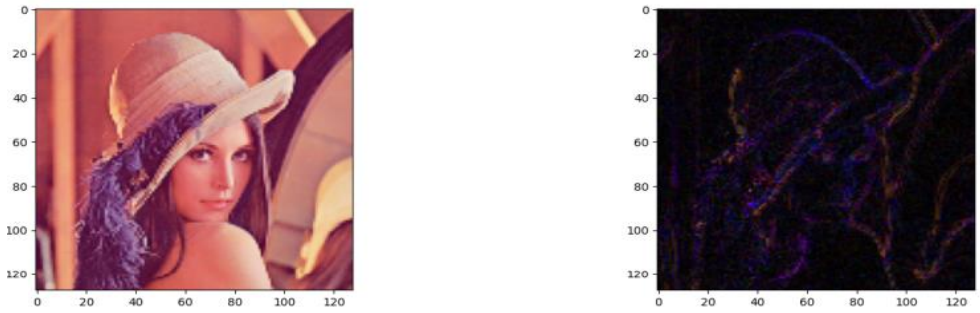


Figure 12.5 *Original image and after making the gradient histogram.*

```
def ft_hog(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  hog_features, hog_image = hog(image, block_norm='L2-Hys',
                                pixels_per_cell=(16, 16), cells_per_block=(1, 1),
                                visualize=True)
  return hog_features, hog_image
```

Listing 12.3 *Implementing the calculation of a gradient histogram.*

- Color Histogram (CH): A color histogram [180] is one standard method used to measure image similarity. The whole thing is graphically shown in Figure 12.6 and implemented in Listing 12.4.

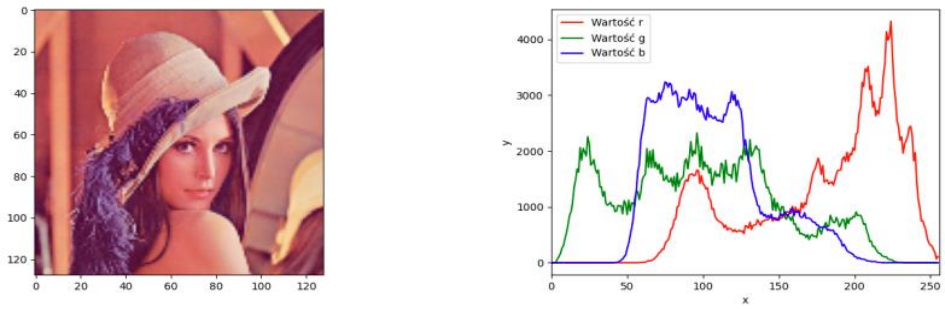


Figure 12.6 *Image and corresponding color histogram.*

```
def ft_histogram(image, mask=None):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  hist  = cv2.calcHist([image], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
  cv2.normalize(hist, hist)
  return hist.flatten()
```

Listing 12.4 *Implementation of computing a color histogram.*

- Hu Moments (HMs): Moments are prevalent features extracted from images to be used in pattern recognition tasks, such as face recognition and shape retrieval. It is excellent that central moments are translation invariant. But that is not enough for shape matching. We want to calculate moments that are invariant to translation, scale, and rotation. HMs are a set of 7 numbers calculated using central moments that are invariant to image transformations. The first six moments have been proven to be consistent with translation, scale, rotation, and reflection. The 7^{th} moment's sign changes for image reflection [181-182] and shown in Listing 12.5.

```
def ft_hu_moments(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  hu_moments = cv2.HuMoments(cv2.moments(image)).flatten()
  hu_moments = scaler.fit_transform(hu_moments.reshape(-1, 1))
  hu_moments = hu_moments.flatten()
  return hu_moments
```

Listing 12.5 *Implementing hu moment extraction*.

A feature set defines a single image after applying these feature engineering techniques using various combinations of these features. Finally, we concatenated the individually extracted features from each dataset to form a unique feature vector, namely, Deepfake Feature (DFF) and used it to feed ML classifiers. Table 12.2 shows various feature set by combining features extracted using these feature engineering or descriptors.

Table 12.2 *Contribution of feature engineering technique to the construction of DFF*

Feature Set	**#Elements per Feature Engineering Techniques**			
	HMs	**HTF**	**CH**	**HOG**
DFF-109	7	13	8	81
DFF-117	7	13	16	81
DFF-133	7	13	32	81
DFF-228	7	13	64	144
DFF-717	7	13	256	441
DFF-2296	7	13	512	1764

12.2.2 Multiple Hypotheses Testing

A common approach is applied for comparing the performances of multiple ML models. The steps are:

1. Conduct an omnibus test under the null hypothesis that there is no difference between the classification accuracies.
2. When the omnibus test resulted in a rejection of the null hypothesis, pairwise posthoc tests should be conducted to show where the differences between models occurred, with adjustment to multiple comparisons.

Omnibus tests are statistical tests to decide if random samples are different from null hypotheses. The so-called Variance Analysis (ANOVA) [183], used widely to evaluate

the importance of null hypotheses that the means of several groups are equal, is a notable example of the omnibus test.

- *Cochran's Q-test:* This test may be viewed as a generalized McNemar test, which may be used to compare three or more classification methods. In a way, the Cochran Q-test is ANOVA-like but works for nominal pairing data. It does not tell us, as does ANOVA, whether groups (or patterns) vary — it merely informs us that there is a difference between the models. The Q test statistic is approximate (like McNemar's test) distributed as chi-squared with the M−1 degrees of freedom, where the M is the number of models we are evaluating.
- *F-test:* The F-test is the same as Cochran's Q-test used to evaluate numerous ML models. After obtaining an F-value, the p-value for the associated degrees of freedom can be selected or computed using a cumulative F-distribution function from an F-distribution table.

Our experiment conducts an omnibus test to assess ML classifiers' significance, using Cochran's Q-test and F-test.

In practice, we could do several post-hoc peer tests after we effectively rejected the null hypothesis.

- *5x2-Fold Cross-Validated Paired t-test*: Dietterich proposed this technique for evaluating the performance of two models (classifier or regressor) to overcome limitations of two approaches: resampled t-testing pairing and k-fold cross-validated t-testing strategies. While the general approach is

identical to the t-test variations mentioned above, we repeat the splitting process precisely five times in this test.

- *Combined 5x2 Cross-Validated F-Test*: This test is an alternative to Dietterichen's 5x2cv paired t-test approach to measure two models' performance.
- *Nested Cross-Validation*: This procedure is relatively easy; that is, just layering two k-fold cross-validation loops: the inside loop takes charge of selecting a model while the external loop estimates the accuracy of the generalization.

12.2.3 Interpretation of Obtained Results

This section presents our results and compares the performance of ML-based methods, and Table 12.3 shows the results of the Q-test and the F-test.

Table 12.3 *Q-test and F-test*

Dataset	***Feature Set***	***Q-test***		***F-test***	
		Q-stat	***p-value***	***F-stat***	***p-value***
F++	DFF-109	9329.233	0.000002	1722.1539	0.000003
	DFF-117	9703.819	0.000002	1799.0783	0.000003
	DFF-133	8509.299	0.000002	1556.0731	0.000003
	DFF-228	7273.790	0.000002	1311.6142	0.000002
	DFF-717	6771.821	0.000001	1214.2285	0.000002
	DFF-2296	8290.169	0.000002	1512.2134	0.000003
DFDC	DFF-109	1934.997	0.000002	337.8089	0.000001
	DFF-117	2026.625	0.000002	354.6049	0.000003
	DFF-133	2255.071	0.000001	396.8126	0.000001
	DFF-228	1751.854	0.000000	304.4638	0.000002
	DFF-717	1263.535	0.000003	217.0004	0.000001
	DFF-2296	1285.951	0.000002	220.9699	0.000001

Table 12.3.3 (continued).

Dataset	Feature Set	Q-test		F-test	
		Q-stat	**p-value**	**F-stat**	**p-value**
Celeb-DF	DFF-109	7549.388	0.000003	1366.3963	0.000003
	DFF-117	7050.527	0.000002	1268.8916	0.000002
	DFF-133	6499.617	0.000002	1162.4864	0.000002
	DFF-228	7112.409	0.000003	1280.9267	0.000002
	DFF-717	5084.560	0.000003	895.1325	0.000001
	DFF-2296	5152.525	0.000002	907.7816	0.000001
VDFD	DFF-109	7628.235	0.000002	1381.6975	0.000003
	DFF-117	7647.576	0.000002	1385.5058	0.000003
	DFF-133	7184.748	0.000002	1294.8315	0.000003
	DFF-228	7497.939	0.000002	1356.0855	0.000003
	DFF-717	5430.538	0.000001	959.62038	0.000001
	DFF-2296	6705.140	0.000001	1201.8674	0.000002

Based on the results shown in Table 12.3, the p-value is less than α (α = 0.05) that leads to rejecting the null hypothesis. Based on the nested cross-validation test, Table 12.4 shows the outcomes of classical ML-based classifiers.

Table 12.4 *Performance of classical ML-based algorithms*

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
FF++	DFF-109	SVM	94.60	93.99	94.29	94.22	94.23
		RF	**99.28**	**99.04**	**99.16**	**99.15**	**99.15**
		ERT	**99.64**	**99.46**	**99.55**	**99.54**	**99.55**
		DT	92.57	92.32	92.44	92.33	92.34
		SGB	81.41	78.78	80.08	80.10	80.12
		MLP	95.97	97.59	96.77	96.69	96.68
		KNN	98.42	98.82	98.62	98.59	98.59
	DFF-117	SVM	94.35	92.76	93.55	93.51	93.52
		RF	**99.44**	**99.27**	**99.36**	**99.35**	**99.35**

Table 12.4.4 (continued).

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
FF++	DFF-117	**ERT**	**99.58**	**99.50**	**99.54**	**99.52**	**99.53**
		DT	93.51	93.68	93.60	93.47	93.49
		SGB	81.96	78.17	80.02	80.19	80.22
		MLP	98.34	96.97	97.65	97.63	97.64
		KNN	98.41	98.82	98.61	98.59	98.58
	DFF-133	SVM	93.02	91.20	92.10	92.06	92.07
		RF	**99.49**	**99.32**	**99.41**	**99.40**	**99.39**
		ERT	**99.65**	**99.53**	**99.59**	**99.60**	**99.59**
		DT	92.83	92.89	92.86	92.76	92.75
		SGB	83.66	80.91	82.27	82.29	82.31
		MLP	98.83	98.08	98.45	98.44	98.44
		KNN	98.54	98.99	98.77	98.74	98.73
	DFF-228	SVM	98.41	97.5	97.95	97.93	97.94
		RF	**99.79**	**99.53**	**99.66**	**99.67**	**99.66**
		ERT	**99.80**	**99.75**	**99.78**	**99.77**	**99.78**
		DT	93.22	93,68	93.45	93.34	93.33
		SGB	87.34	87.67	87.51	87.30	87.29
		MLP	98.99	99.88	99.43	99.42	99.41
		KNN	99.63	99.68	99.66	99.65	99.64
	DFF-717	SVM	97.86	97.30	97.58	97.55	97.55
		RF	**99.60**	**99.26**	**99.43**	**99.42**	**99.43**
		ERT	**99.52**	**99.32**	**99.42**	**99.41**	**99.41**
		DT	89.29	88.73	89.01	88.90	88.88
		SGB	87.46	86.75	87.10	86.97	86.96
		MLP	99.56	99.22	99.39	99.38	99.38
		KNN	96.75	97.43	97.09	97.04	97.02
	DFF-2296	**SVM**	**99.42**	**99.52**	**99.47**	**99.46**	**99.47**
		RF	**99.28**	**99.42**	**99.85**	**99.84**	**98.84**
		ERT	**99.15**	**98.92**	**99.03**	**99.02**	**99.04**
		DT	83.32	84.26	84.04	83.76	83.75
		SGB	87.60	90.36	88.96	88.61	88.58
		MLP	99.23	98.60	98.90	98.91	98.90
		KNN	95.92	96.23	96.08	96.01	96.00

Table 12.4.4 (continued).

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
DFDC	DFF-109	SVM	93.25	88.59	90.86	91.46	91.34
		RF	**99.08**	**97.94**	**98.51**	**98.58**	**98.55**
		ERT	**99.08**	**97.85**	**98.46**	**98.53**	**98.51**
		DT	93.11	92.59	92.85	93.16	93.14
		SGB	91.04	80.68	85.54	86.93	86.69
		MLP	95.79	97.00	96.39	96.53	96.54
		KNN	96.93	96.47	96.70	96.84	96.83
	DFF-117	SVM	93.27	87.65	90.37	91.05	90.91
		RF	**99.11**	**98.06**	**98.58**	**98.65**	**98.62**
		ERT	**99.17**	**98.44**	**98.80**	**98.86**	**98.84**
		DT	94.01	94.15	94.08	94.32	94.13
		SGB	91.29	81.74	86.25	87.51	87.28
		MLP	97.01	97.29	97.15	97.27	97.26
		KNN	96.88	96.68	96.78	96.91	96.90
	DFF-133	SVM	93.02	84.32	88.46	89.46	89.25
		RF	**99.08**	**98.32**	**98.70**	**98.76**	**98.74**
		ERT	**99.35**	**98.59**	**98.97**	**99.01**	**98.99**
		DT	94.35	94.29	94.32	94.56	94.55
		SGB	91.92	83.32	87.41	88.50	88.29
		MLP	**97.89**	**98.41**	**98.15**	**98.22**	**98.23**
		KNN	97.20	96.82	97.01	97.14	97.13
	DFF-228	SVM	97.02	93.91	95.44	95.70	95.63
		RF	**99.35**	**98.71**	**99.03**	**99.07**	**99.06**
		ERT	**99.44**	**99.26**	**99.35**	**99.38**	**99.37**
		DT	94.35	94.38	94.37	94.60	94.59
		SGB	93.97	85.68	89.63	90.50	90.31
		MLP	99.00	99.29	99.15	99.18	99.19
		KNN	98.64	98.12	98.38	98.45	98.44
	DFF-717	SVM	96.60	93.71	95.13	95.41	95.34
		RF	**98.99**	**97.76**	**98.37**	**98.45**	**98.42**
		ERT	**99.07**	**97.53**	**98.30**	**98.38**	**98.35**
		DT	94.57	93.79	94.18	94.45	94.42

Table 12.4.4 (continued).

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
DFDC	DFF-717	SGB	93.79	86.65	90.08	90.85	90.67
		MLP	**99.32**	**98.97**	**99.15**	**99.18**	**99.17**
		KNN	96.58	96.21	96.39	96.55	96.53
	DFF-2296	**SVM**	**99.15**	**99.38**	**99.27**	**99.30**	**99.29**
		RF	**99.20**	**99.00**	**99.10**	**99.14**	**99.02**
		ERT	**99.23**	**98.94**	**99.09**	**99.15**	**99.14**
		DT	94.63	94.88	94.76	94.97	95.26
		SGB	95.80	89.32	92.45	93.01	92.86
		MLP	99.44	99.76	99.60	99.62	99.19
		KNN	97.83	98.06	97.94	98.03	98.01
Celeb-DF	DFF-109	SVM	92.26	90.33	91.28	91.27	91.28
		RF	**98.95**	**98.19**	**98.57**	**98.56**	**98.56**
		ERT	**98.95**	**98.49**	**98.72**	**98.70**	**98.71**
		DT	92.18	91.53	91.86	91.80	91.79
		SGB	81.54	77.43	79.43	79.72	79.75
		MLP	95.35	94.91	95.13	95.08	95.08
		KNN	96.39	95.85	96.12	96.09	96.10
	DFF-117	SVM	91.79	88.82	90.28	90.33	90.35
		RF	**99.10**	**98.59**	**98.85**	**98.84**	**98.83**
		ERT	**99.07**	**98.52**	**98.80**	**98.78**	**98.80**
		DT	92.95	93.34	93.14	93.05	93.04
		SGB	83.92	79.24	81.51	81.42	81.85
		MLP	97.01	96.94	96.97	96.94	96.94
		KNN	96.46	95.53	96.19	96.17	96.16
	DFF-133	SVM	91.13	88.63	89.86	89.89	89.90
		RF	**99.19**	**98.74**	**98.96**	**98.95**	**98.96**
		ERT	**99.24**	**98.89**	**99.06**	**99.08**	**99.06**
		DT	93.35	92.85	93.10	93.04	93.05
		SGB	83.99	79.82	81.85	82.10	82.13
		MLP	91.75	99.17	95.32	95.07	95.02
		KNN	96.52	95.98	96.25	96.22	96.22

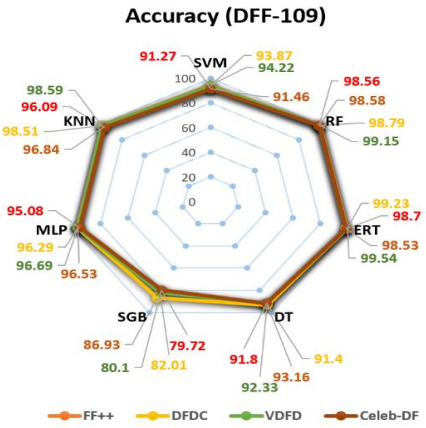
Table 12.4.4 (continued).

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
Celeb-DF	DFF-228	SVM	96.27	95.56	95.91	95.88	95.89
		RF	**99.69**	**99.18**	**99.43**	**99.43**	**99.43**
		ERT	**99.56**	**99.18**	**99.37**	**99.36**	**99.37**
		DT	93.79	93.40	93.59	93.53	93.54
		SGB	86.44	83.00	84.68	84.81	84.84
		MLP	**99.23**	**99.27**	**99.25**	**99.24**	**99.23**
		KNN	97.85	97.47	97.66	97.64	97.64
	DFF-717	SVM	95.98	95.79	95.89	95.84	95.85
		RF	**99.06**	**98.56**	**98.81**	**98.82**	**98.80**
		ERT	**98.88**	**98.07**	**98.48**	**98.46**	**98.47**
		DT	92.74	92.10	92.42	92.36	92.35
		SGB	87.16	84.74	85.93	85.97	85.98
		MLP	**99.34**	**98.64**	**98.99**	**98.98**	**98.98**
		KNN	94.62	94.57	94.60	94.54	94.53
	DFF-2296	SVM	98.15	97.81	97.98	97.96	97.96
		RF	98.36	97.47	97.86	97.85	97.85
		ERT	97.91	97.29	97.60	97.58	97.58
		DT	89.18	89.14	89.16	89.07	89.04
		SGB	86.76	86.21	86.48	86.37	86.37
		MLP	98.85	98.69	98.77	98.78	98.76
		KNN	93.92	93.76	93.84	97.79	93.77
VDFD	DFF-109	SVM	93.63	94.13	93.88	93.87	93.87
		RF	98.91	98.65	98.78	98.79	98.79
		ERT	**99.22**	**99.23**	**99.23**	**99.23**	**99.22**
		DT	91.62	91.14	91.38	91.40	91.41
		SGB	80.29	84.80	82.48	82.01	82.16
		MLP	95.88	96.71	96.30	96.29	96.27
		KNN	98.27	98.70	98.49	98.51	98.49
	DFF-117	SVM	93.28	93.54	93.40	93.42	93.41
		RF	**99.05**	**98.82**	**98.94**	**98.94**	**98.93**
		ERT	**99.31**	**99.30**	**99.28**	**99.31**	**99.31**
		DT	92.71	91.87	92.29	92.33	92.34

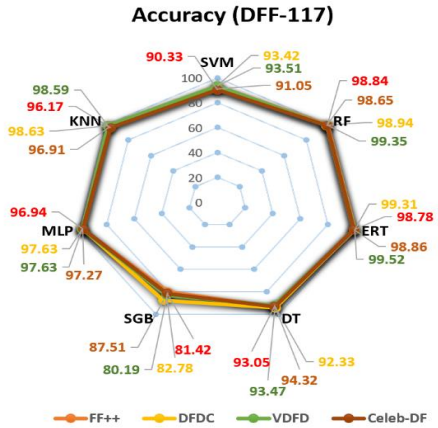
Table 12.4.4 (continued).

Dataset	Feature	Model	PREC	REC	F1-SC	ACC	AUC
VDFD	DFF-117	SGB	81.07	85.46	83.21	82.78	82.79
		MLP	98.59	96.63	97.60	97.63	97.62
		KNN	**98.42**	**98.83**	**98.62**	**98.63**	**98.62**
	DFF-133	SVM	92.33	92.03	92.18	92.20	92.21
		RF	**99.23**	**99.03**	**99.13**	**99.13**	**99.13**
		ERT	**99.40**	**99.38**	**99.37**	**99.38**	**99.39**
		DT	92.53	91.72	92.12	92.17	92.16
		SGB	81.89	85.92	83.85	83.48	83.49
		MLP	99.54	95.44	97.45	97.52	97.50
		KNN	98.39	98.73	98.56	98.58	98.56
	DFF-228	SVM	97.40	96.94	97.17	97.18	97.17
		RF	**99.36**	**99.33**	**99.35**	**99.36**	**99.35**
		ERT	**99.53**	**99.62**	**99.58**	**99.58**	**99.58**
		DT	93.95	92.98	93.46	93.51	93.50
		SGB	84.23	88.41	86.27	85.95	85.95
		MLP	**99.66**	**99.37**	**99.51**	**99.52**	**99.52**
		KNN	**99.18**	**99.21**	**99.19**	**99.20**	**99.20**
	DFF-717	SVM	98.66	98.40	98.53	98.54	98.53
		RF	99.21	98.55	98.88	98.89	98.89
		ERT	**99.32**	**99.27**	**99.29**	**99.31**	**99.30**
		DT	91.21	90.43	90.82	90.89	90.87
		SGB	87.89	90.76	89.30	89.16	89.14
		MLP	**99.72**	**99.57**	**99.66**	**99.65**	**99.64**
		KNN	97.00	97.21	97.10	97.12	97.10
	DFF-2296	**SVM**	**99.71**	**99.62**	**99.67**	**99.66**	**99.67**
		RF	99.16	97.55	98.34	98.36	98.35
		ERT	98.90	98.68	98.79	98.79	98.79
		DT	86.44	84.82	85.62	85.79	85.78
		SGB	91.60	93.12	92.35	92.32	92.30
		MLP	**99.46**	**99.63**	**99.55**	**99.55**	**99.54**
		KNN	95.96	96.29	96.12	96.13	96.12

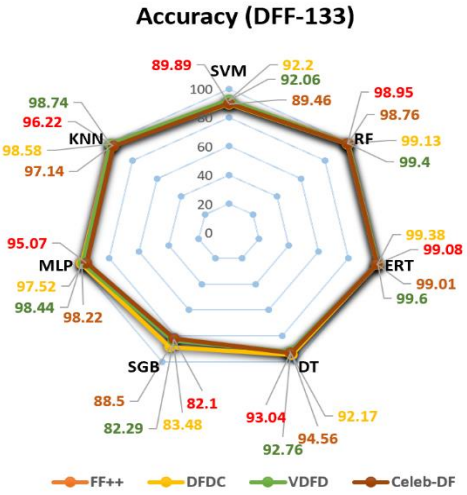
We conducted our experiment using four different datasets for ML-based methods: FF++, DFDC, Celeb-DF, and VDFD. The accuracy results are shown in figure 12.7. We have conducted our experiments using various combinations of feature sets. Based on the experiments using FF++, we see that RF and ERT achieved the best performance using any feature set and obtained more than 99% accuracy, but SVM only achieved the same performance using feature set DFF-2296. MLP and KNN achieved 98% accuracy, where SGB and DT achieved about 95% accuracy. In the case of the DFDC dataset, using the feature sets DFF-2296, SVM, RF, and ERT achieved 99% accuracy, where MLP achieved the best performance using all feature sets. But SVM decreased its performance in detecting Deepfake while decreasing the size of the feature set. For example, its accuracy decreased to 89% for DFF-133 and 91% for DFF-117. A similar performance is obtained by SVM using the same feature set for Celeb-DF Dataset. In VDFD dataset-based experiments, it is seen that MLP obtained better performance than RF and ERT for any feature set and obtained more than 99% accuracy. It is also seen that using VDFD, KNN improved its detection performance for a certain feature set, for example, DFF-117 and DFF-228.



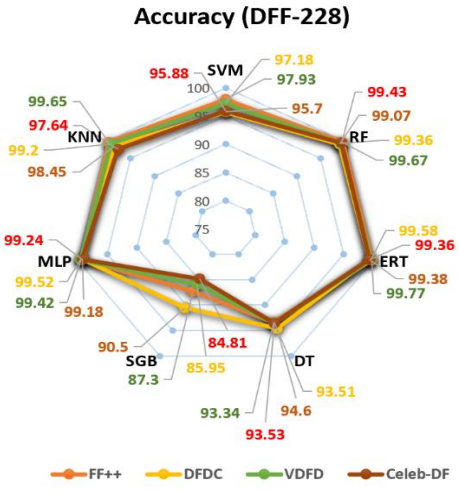
(a) DFF-109



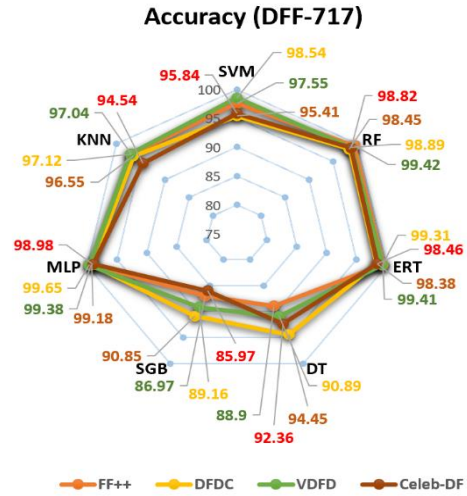
(b) DFF-117



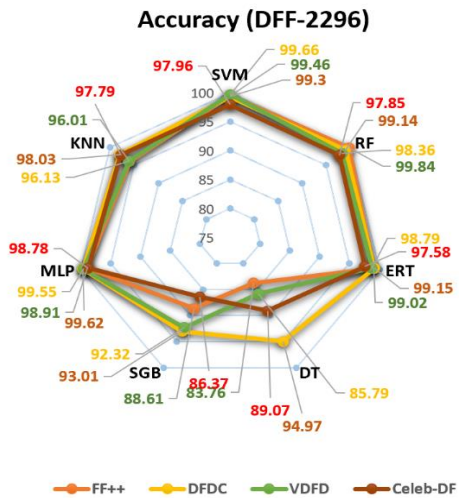
(c) DFF-133



(d) DFF-228



(e) DFF-717

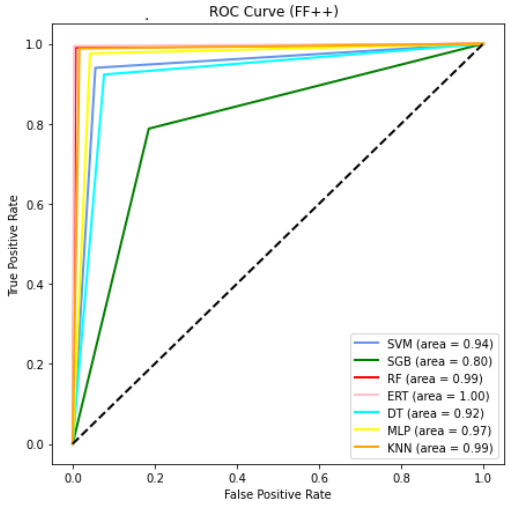


(f) DFF-2296

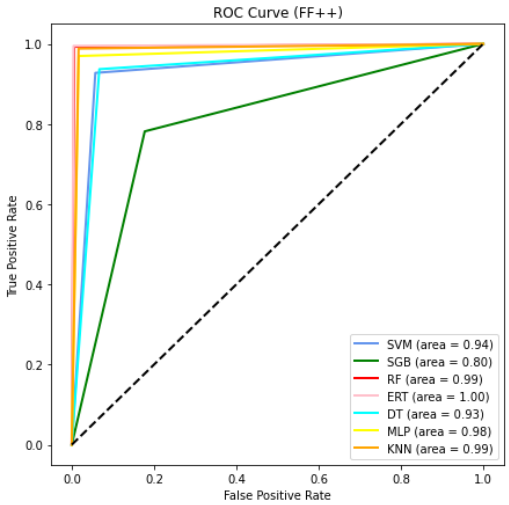
Figure 12.7 *Accuracy of ML classifiers using various feature sets and datasets.*

Based on the overall experiments using FF++, VDFD, Celeb-DF, DFDC datasets, it is seen that RF, ERT, and MLP obtain state-of-the-art performances by achieving more than 99% accuracy. Besides, DFF-133 and DFF-228 provide the best accuracy. Based on the results presented above, the classical ML models now learned to detect the Deepfake videos.

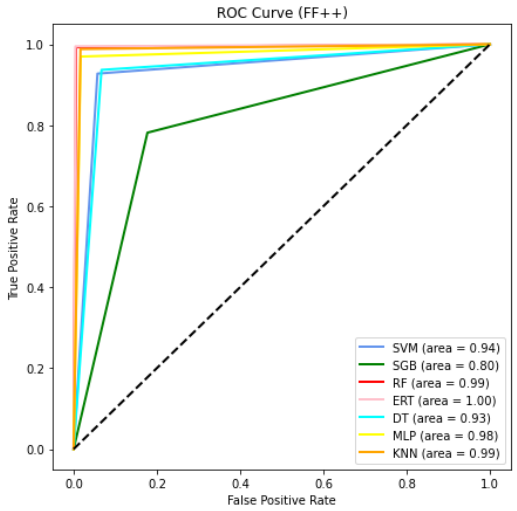
In summary, each model generates a ROC curve that shows us how good the model is for classifying the original and the Deepfake classes, as seen in Figures 12.8-12.11. The curve fills the area between the colored line and the x-axis. A single model/classifier is specified as each color line. The bigger the size covered, the better the models are at classifying the given classes. In other words, the closer the AUC is to 1.0, the better. Based on the experiments, ERT achieves an AUC of 1.0 using any feature sets of the FF++ dataset, but using other datasets, it reaches 0.99. MLP obtains an AUC of 0.99 for any dataset with any feature set. Overall, ERT, RF bring an AUC of 0.99 for any dataset with any feature set. The AUC value varies for KNN and SVM for various feature sets. Based on the overall results, it is seen that the classical ML-based methods can correctly separate the positive and negative data classes.



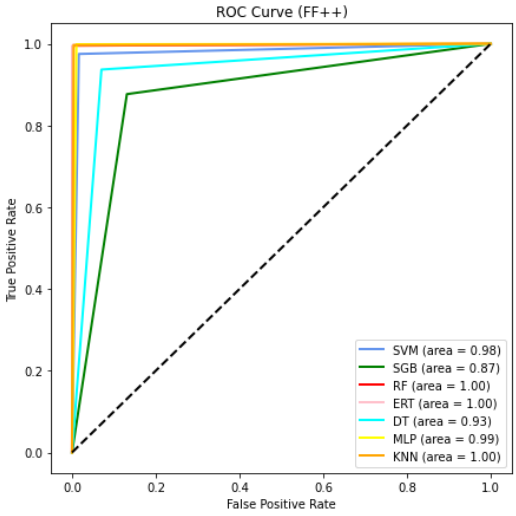
(a) DFF-109



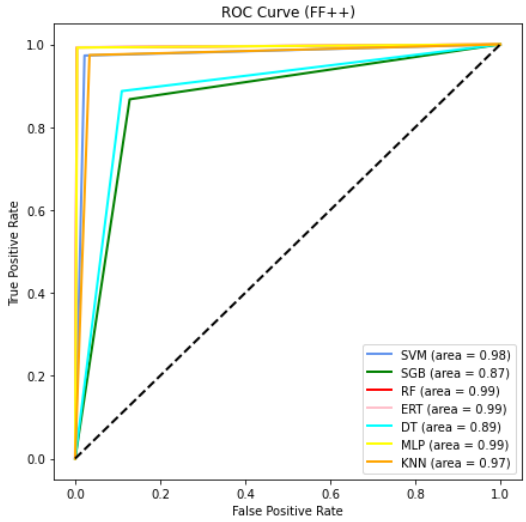
(b) DFF-117



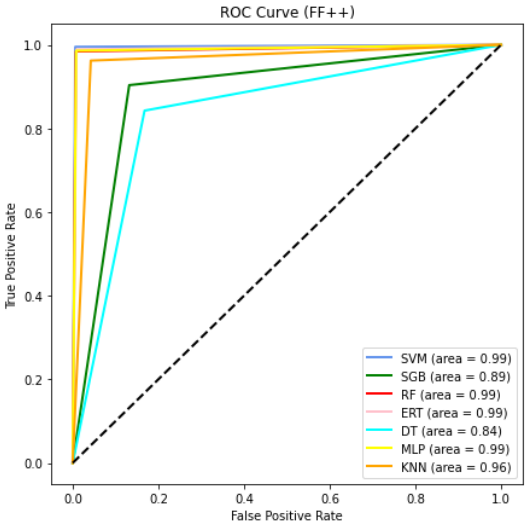
(c) DFF-133



(d) DFF-228

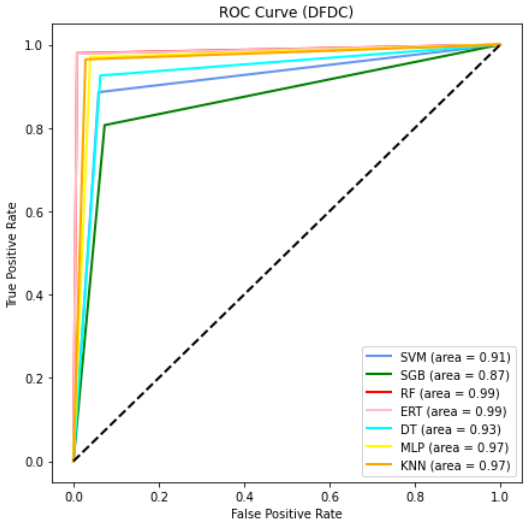


(e) DFF-717

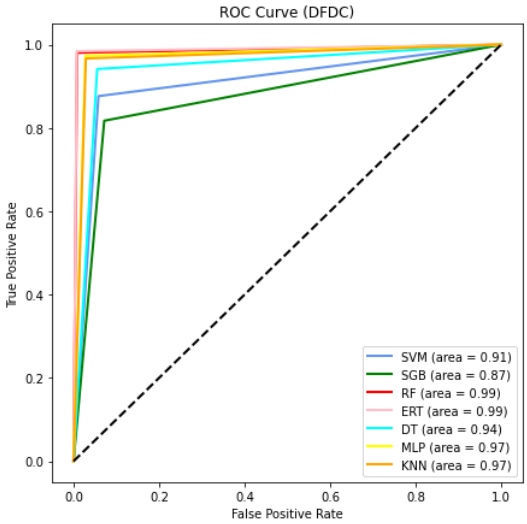


(f) DFF-2296

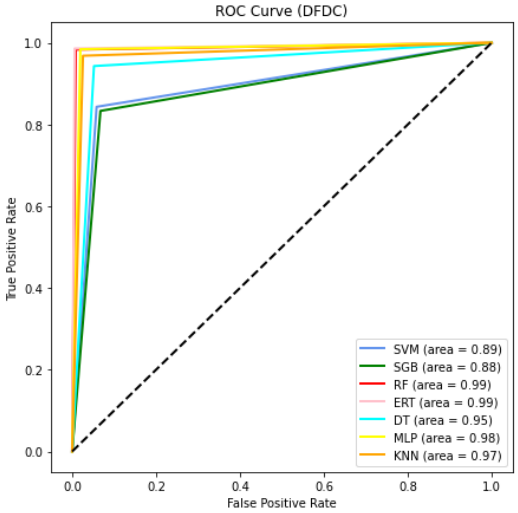
Figure 12.8 *AUC of ML classifiers using various feature sets on the FF++ dataset*



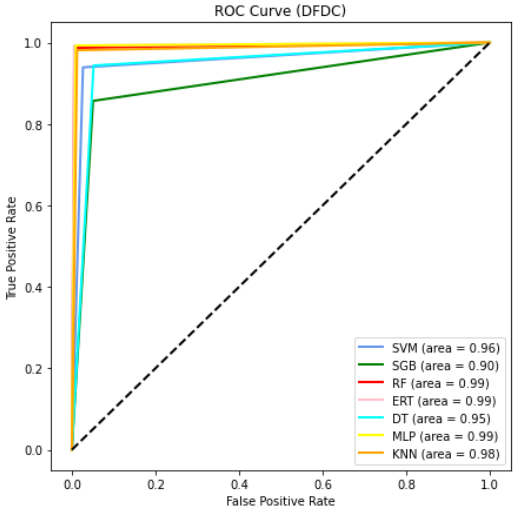
(a) DFF-109



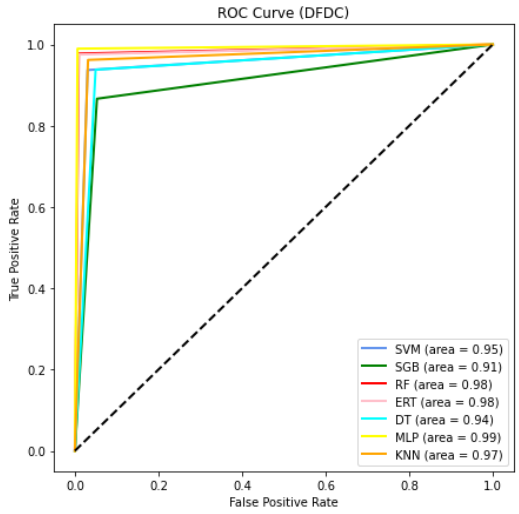
(b) DFF-117



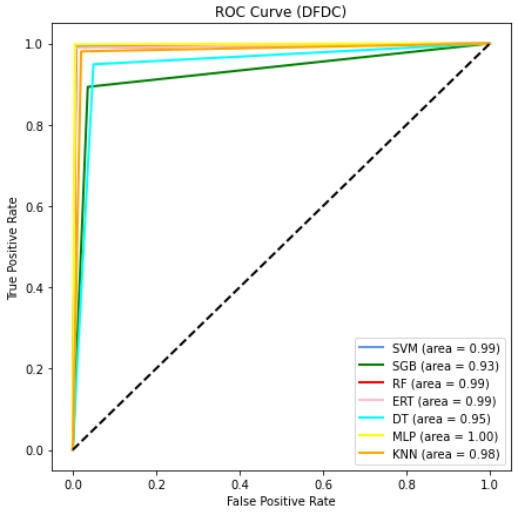
(c) DFF-133



(d) DFF-228

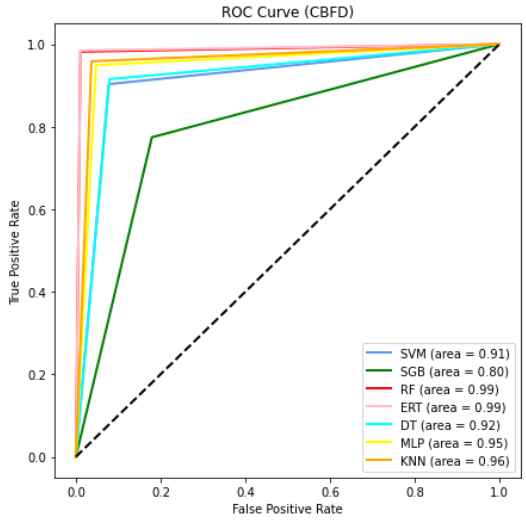


(e) DFF-717

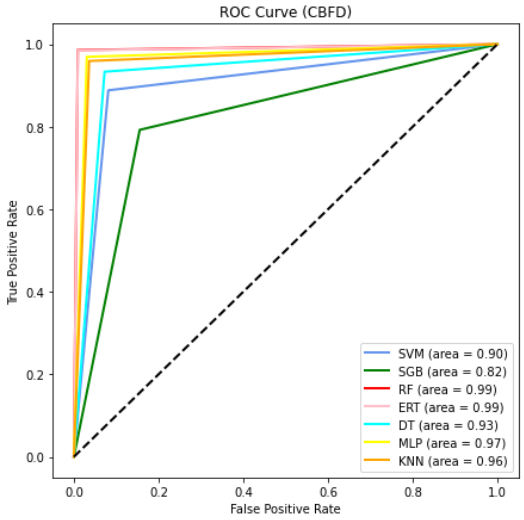


(f) DFF-2296

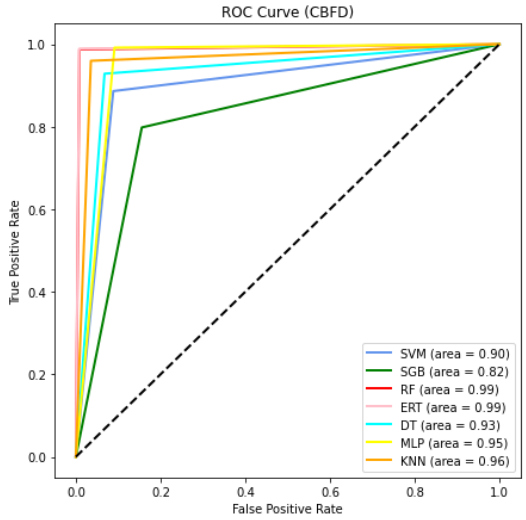
Figure 12.9 *AUC of ML classifiers using various feature sets on the DFDC dataset.*



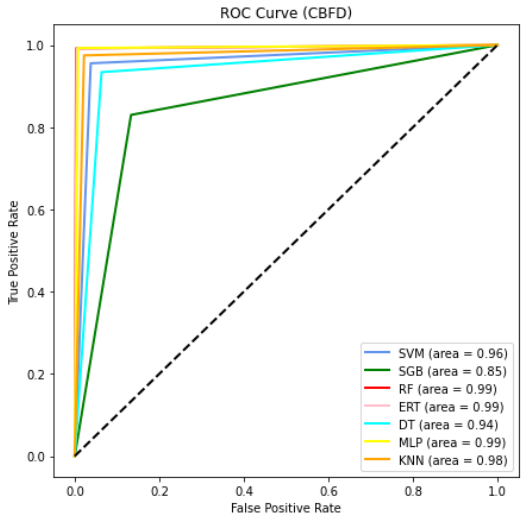
(a) DFF-109



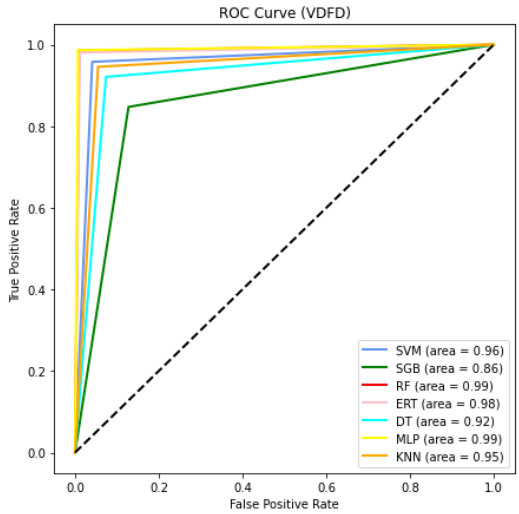
(b) DFF-117



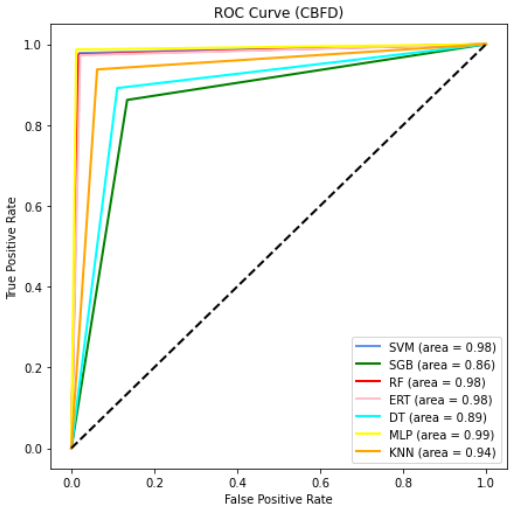
(c) DFF-133



(d) DFF-228

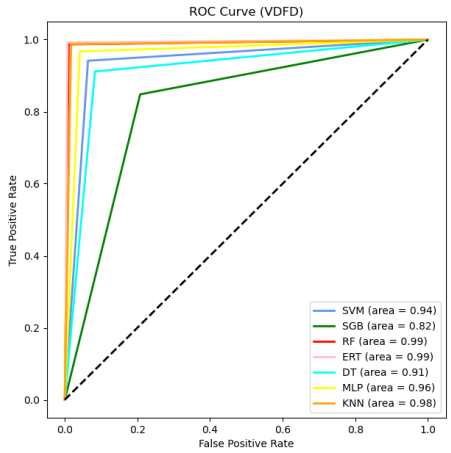


(e) DFF-717

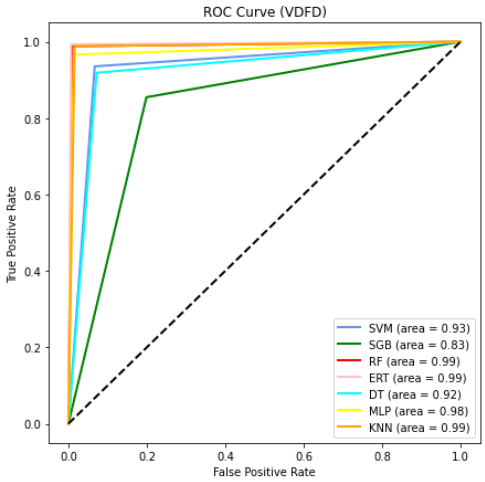


(f) DFF-2296

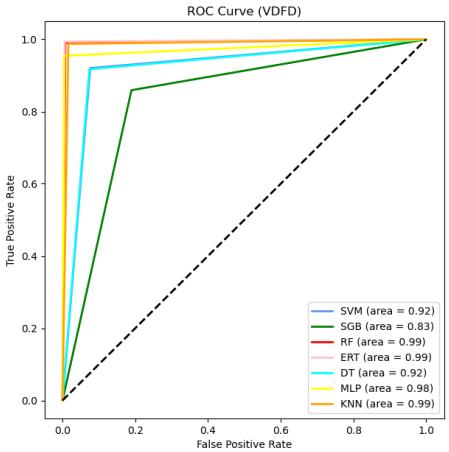
Figure 12.10 *AUC of ML classifiers using various feature sets on the Celeb-DF dataset.*



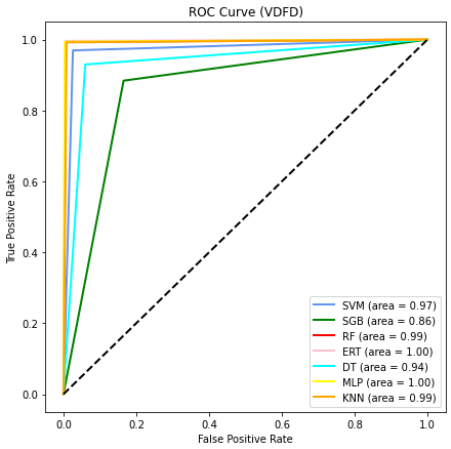
(a) DFF-109



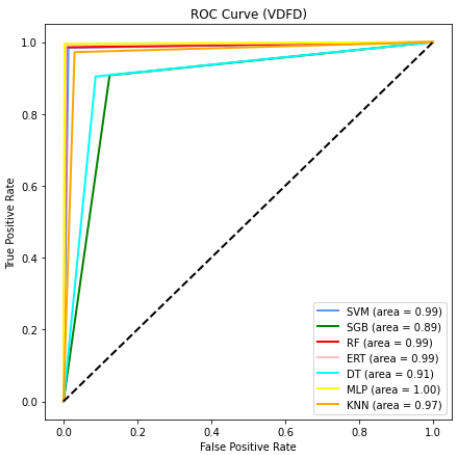
(b) DFF-117



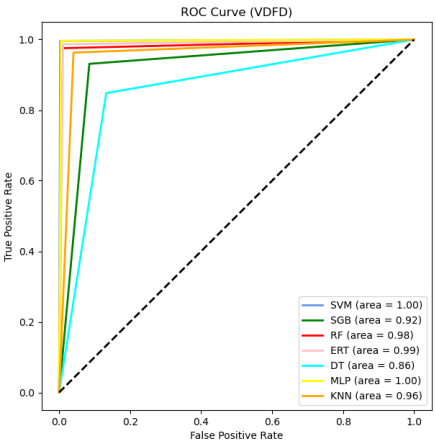
DFF-133



DFF-228



DFF-717



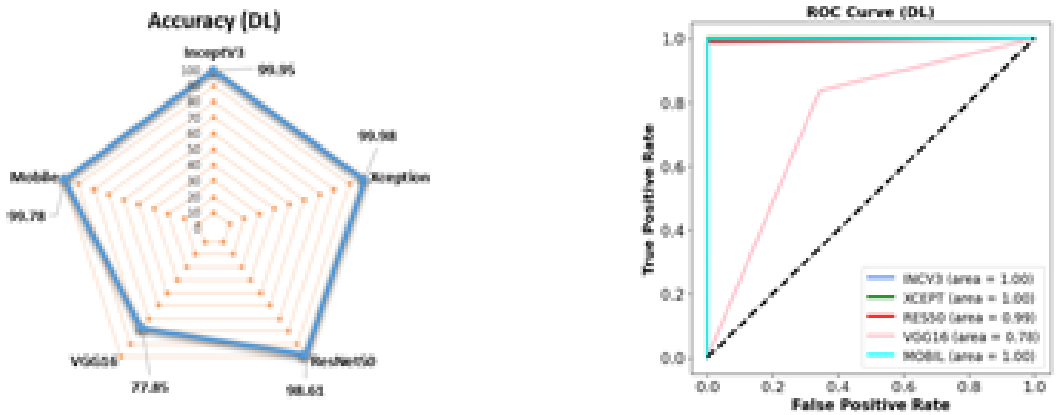
DFF-2296

Figure 12.11 *AUC of ML classifiers using various feature sets on the VDFD dataset.*

In order to compare the proposed ML-based method, we conducted an experiment using DL-based models. Based on the FF++ dataset, the results of these DL-based state-of-the-arts models can be summarized in Table 12.5 and Figure 12.12.

Table 12.5 *Performances of DL-based state-of-the-arts models*

Model	PREC	REC	F1-SC	ACC	AUC
INCV3	99.98	99.95	99.98	99.95	99.95
XCEPT	**100.0**	**99.95**	**99.98**	**99.98**	**99.97**
RES50	100.0	98.12	98.05	98.61	98.69
VGG16	81.91	71.49	76.34	77.85	77.84
MOBILE	99.90	99.65	99.78	99.78	99.63



(a) Accuracy (b) AUC Curve

Figure 12.12 *AUC of DL classifiers using various feature sets on the FF++ dataset.*

In order to get the statistical significance among various ML methods for multiple datasets, we conduct other experiments. Based on the Combined 5x2CV F-test on FF++ and VDFD (our own) datasets, Table 12.6 shows different ML classifiers' outcomes, where ONE stands for a statistical significance between the classifiers' pair and ZERO if NOT.

Table 12.6 *Statistical significance in ML methods based on Combined 5x2 CV F-test.*

Features	Model	FF++							VDFD						
		SVM	**SGB**	**RF**	**ERT**	**DT**	**MLP**	**KNN**	**SVM**	**SGB**	**RF**	**ERT**	**DT**	**MLP**	**KNN**
Accuracy	SVM	0	1	1	0	1	0	1	0	1	1	0	1	1	1
	SGB	1	0	1	1	1	1	1	1	0	1	1	1	1	1
	RF	1	1	0	1	1	1	1	1	1	0	1	1	1	1
	ERT	0	1	1	0	1	0	1	1	1	0	0	1	1	1
	DT	1	1	1	1	0	1	1	1	1	1	1	0	1	1
	MLP	0	1	1	0	1	0	1	1	1	1	1	1	0	1
	KNN	1	1	1	1	1	1	0	1	1	1	1	1	1	0
Precision	SVM	0	1	1	0	1	0	1	0	0	1	1	1	1	1
	SGB	1	0	1	1	1	1	1	0	0	1	1	1	1	1
	RF	1	1	0	1	1	1	1	1	1	0	1	1	1	1
	ERT	0	1	1	0	1	0	1	1	1	1	0	1	1	1
	DT	1	1	1	1	0	1	1	1	1	1	1	0	1	1
	MLP	0	1	0	0	1	0	1	1	1	1	1	1	0	0
	KNN	1	1	1	1	1	1	0	1	1	1	1	1	0	0
Recall	SVM	0	1	0	0	1	1	1	0	0	1	1	1	1	1
	SGB	1	0	1	1	1	1	1	0	0	1	1	1	1	1
	RF	0	1	0	1	1	1	0	1	1	0	1	1	1	1
	ERT	0	1	1	0	1	0	1	1	1	1	0	1	1	1
	DT	1	1	1	1	0	1	1	1	1	1	1	0	1	1
	MLP	0	1	1	0	1	0	1	1	1	1	1	1	0	0
	KNN	1	1	0	1	1	1	0	1	1	1	1	1	0	0
F1-Score	SVM	0	1	0	0	1	1	1	0	0	1	1	1	1	1
	SGB	1	0	1	1	1	1	1	0	0	1	1	1	1	1
	RF	1	1	0	1	1	1	1	1	1	0	1	1	1	1
	ERT	0	1	1	0	1	0	1	1	1	1	0	1	1	1
	DT	1	1	1	1	0	1	1	1	1	1	1	0	1	1
	MLP	0	1	1	0	1	0	1	1	1	1	1	1	0	1
	KNN	1	1	1	1	1	1	0	1	1	1	1	1	1	0

CHAPTER XIII – CONCLUSION

Detecting Deepfakes has become a significant challenge because even though many such manipulated videos are intended for entertainment, many of them could still be harmful to individuals and society. Based on the research needs, a few datasets of Deepfake manipulation have been made available.

In the first phase of this research, we propose a deep ensemble learning technique, DeepfakeStack, by experimenting with various DL-based models on the FF++ dataset. The experiment shows that a larger stacking ensemble neural network (called DFC) model is defined and fit on the test (unseen) dataset, then the new model is used to predict the test dataset. Evaluating the results, we see that the proposed DFC model achieves an accuracy of 99.65% and AUROC 1.0, outperforming the DL-based models, thereby provides a strong basis for developing an effective Deepfake detector.

While modern deep learning (DL) based methods have achieved highly accurate results in detecting Deepfakes, they carry with them disadvantages that include understandability, interpretability, data complexity, and high computational cost. The DL model is highly complicated and cannot easily be interpreted [174]. Further, they are precise to their context and, being very deep, tend to extract the underlying semantics from the images without having a single fingerprint [184]. In contrast, traditional machine learning (ML) methods allow better interpretability; for example, tree-based ML methods illustrate the detection process in the form of a decision tree.

For an objective evaluation, we introduce in the second phase of this research a classical ML-based method to detect Deepfakes using the standard method of feature development, extraction, and classifier training and testing. The experimental outcomes

prove that the ML techniques alone can obtain state-of-the-art performance in detecting Deepfakes.

We believe that our proposed two methods can lay the basis for developing an effective solution for detecting Deepfakes and other facial manipulations. Our ongoing work includes further research on feature development and selection and ensemble detection for enhanced performance.

BIBLIOGRAPHY

1. W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of android application security," In Proceedings of the 20th usenix conference on security (pp. 21–21). SEC'11. San Francisco, CA: USENIX Association.
2. J. J. Drake, Z. Lanier, C. Mulliner, P. O., Fora, S. A., Ridley, and G. Wicherski, "Android Hacker's Handbook," Wiley, Indianapolis, 2014.
3. Android Architecture [Internet]. Available from: https://www.javatpoint.com/android-software-stack [Accessed: 2021-07-04]
4. C. Li, R. Zhu, D. Niu, K. Mills, H. Zhang, and H. Kinawi, "Android Malware Detection based on Factorization Machine," arXiv:1805.11843v1.
5. Android Manifest [Internet]. 2014. Available from: https://javapapers.com/android/android-manifest/ [Accessed: 2021-07-04]
6. A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," In the Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11). ACM, New York, NY 2011, pp. 3-14.
7. M. Spreitzenbarth, "Dissecting the Droid: Forensic Analysis of Android and its malicious Applications," PhD Thesis, University of Erlangen-Nürnberg 2013.
8. Machine Learning (What it is and why it matters) [Internet], https://www.sas.com/en_us/insights-/analytics/machinelearning.html#:~:text=Machine%20learning%20is%20a%20method,decisions%20with%20minimal%20human%20intervention [Accessed: 2021-04-30]

9. M. Kubat, R. Holte, and S. Matwin, “Machine learning for the detection of oil spills in satellite radar images,” Machine Learning, 30(2-3), 195–215, 1998, DOI: 10.1023/A:1007452223027.
10. M. Cococcioni, L. Corucci, A. Masini, and F. Nardelli, “SVME: an ensemble of support vector machines for detecting oil spills from full resolution modis images,” Ocean Dynamics, 62(3), 449–467. DOI:10.1007/s10236-011-0510-8.
11. A. Madabhushi, J. Shi, M. Feldman, M. Rosen, and J. Tomaszewski, “Comparing ensembles of learners: detecting prostate cancer from high resolution MRI,” In R. Beichel & M. Sonka (Eds.), Computer vision approaches to medical image analysis (Vol. 4241, pp. 25–36). Lecture Notes in Computer Science. Springer Berlin Heidelberg. DOI: 10.1007/11889762_3.
12. M. Liu, and J. Lu, “Support vector machine – an alternative to artificial neuron network for water quality forecasting in an agricultural nonpoint source polluted river?” Environmental Science and Pollution Research, 21(18), pp. 11036–11053, 2014. DOI: 10.1007/s11356-014-3046-x.
13. A. Azzini, M. De Felice, and A. Tettamanzi, “A comparison between nature-inspired and machine learning approaches to detecting trend reversals in financial time series,” In A. Brabazon, M. O’Neill, & D. Maringer (Eds.), Natural computing in computational finance (Vol. 380, pp. 39–59). Studies in Computational Intelligence. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-23336-4_3.
14. M. N. Murti and V. S. Devi, “Feature Extraction and Feature Selection, Introduction to Pattern Recognition and Machine Learning,” IISc Lecture Notes Series, June 2015, pp. 75-110. DOI: 10.1142/9789814335461_0003

15. Confusion Matrix [Internet]. Available from: http://www2.cs.uregina.ca/dbd/-cs831/notes/confusion-matrix/-confusion-matrix.html, [Accessed: 2019-05-31].
16. Anti-virus market hits $4bn [Internet], Available from: http://www.theregister.co.uk/2006/06/26/av-market-gartner/, [Accessed: 2021-05-30].
17. W. Enck, P. Gilbert, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, "Taintdroid: an information-flow tracking system for real-time privacy monitoring on smartphones," In the Proceeding of USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 393–407, 2010.
18. Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, you, get off my market: detecting malicious apps in official and alternative android markets," In the Proceeding of Network and Distributed System Security Symposium (NDSS), 2012.
19. L. K. Yan, and H. Yin, "Droidscope: seamlessly reconstructing OS and Dalvik semantic views for dynamic android malware analysis," In the Proceeding of USENIX Security Symposium, 2012.
20. W. Enck, M. Ongtang, and P. D. McDaniel, "On lightweight mobile phone application certification," In the Proceeding of ACM Conference on Computer and Communications Security (CCS), pp. 235–245, 2009.
21. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," In the Proceeding of ACM Conference on Computer and Communications Security (CCS), pp. 627–638, 2011.
22. M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Risk-ranker: scalable and accurate zero-day android malware detection," In the Proceeding of International Conference on Mobile Systems, Applications, and Services (MOBISYS), pp. 281–294, 2012.

23. K. A. Talha, D. I. Alper and C. Aydin, “APK Auditor: Permission-based Android malware detection system,” Digit. Invest. 13 (2015) 1–14.
24. W. Fan, L. Zhao, J. Wang, Y. Chen, F. Wu and Y. Liu, “FamDroid: Learning-Based Android Malware Family Classification Using Static Analysis,” arXiv:2101.03965v2, 2021.
25. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, “Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications,” in European symposium on research in computer security, pp. 163–182, Springer, 2014.
26. D. Wu, C. Mao, T. Wei, H. Lee and K. Wu, “DroidMat: Android Malware Detection through Manifest and API Calls Tracing,” 2012 Seventh Asia Joint Conference on Information Security, 2012, pp. 62-69, DOI: 10.1109/AsiaJCIS.2012.18.
27. B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero and P. G. Bringas, “On the automatic categorisation of android applications,” 2012 IEEE Consumer Communications and Networking Conference (CCNC), pp. 149-153, 2012. DOI: 10.1109/CCNC.2012.6181075.
28. A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, “Madam: Effective and efficient behavior-based android malware detection and prevention,” IEEE Transactions on Dependable and Secure Computing, PP (99):1–1, 2016.
29. G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, “Dendroid: A text mining approach to analyzing and classifying code structures in android malware families,” Expert Systems with Applications, 41(1):1104–1117, 2014.

30. J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S. Malek, “Obfuscation-resilient, efficient, and accurate detection and family identification of android malware,” Technical report, Dept. of Computer Science, George Mason University, 2015.
31. Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” International Conference on Security and Privacy in Communication Networks (SecureComm), 2013.
32. M. Zhang, Y. Duan, H. Yin, and Z. Zhao, “Semantics-aware Android malware classification using weighted contextual API dependency graphs,” ACM SIGSAC Computer and Communications Security (CCS), 2014.
33. S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, “DroidScribe: Classifying Android Malware Based on Runtime Behavior,” 2016 IEEE Security and Privacy Workshops (SPW), 2016, pp. 252-261, DOI: 10.1109/SPW.2016.25.
34. W. C. Wu and S. H. Hung, “DroidDolphin: A dynamic Android malware detection framework using big data and machine learning,” In the Proc. 2014 Conf. Research in Adaptive and Convergent Systems (ACM, 2014), pp. 247–252.
35. A. Martín, R. Lara-Cabrera and D. Camacho, “Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset,” Inf. Fusion 52 (2019) 128–142, DOI: 10.1016/j.in®us.2018.12.006.
36. P. Feng, J. Ma, C. Sun, X. Xu and Y. Ma, “A novel dynamic Android malware detection system with ensemble learning,” IEEE Access 6 (2018) 30996–31011.

37. K. Xu, Y. Li and R. H. Deng, “ICCDetector: ICC-based malware detection on Android,” IEEE Trans. Inf. Forensics Sec. 11(6) (2016) 1252–1264.

38. H. Peng, C. Gates, B. Sarma, N. Li, Y. Qi, R. Potharaju, C. N. Rotaru and I. Molloy, “Using probabilistic generative models for ranking risks of Android apps,” ACM Conf. Computer and Communications Security (2012), pp. 241–252, 2012.

39. J. Gu, B. Sun, X. Du, J. Wang, Y. Zhuang, and Z, Wang, “Consortium blockchain-based malware detection in mobile devices,” IEEE Access, vol. 6, pp. 12118–12128, 2018. DOI: 10.1109/access.2018.2805783.

40. S. Raje, S. Vaderia, N. Wilson, and R. Panigrahi, “Decentralised firewall for malware detection,” 2017 International Conference on Advances in Computing, Communication and Control (ICAC3), pp. 1–5, 2017.

41. A. Ouaguid, N. Abghour, and M. Ouzzif, “A novel security framework for managing android permissions using blockchain technology,” Int. J. Cloud Appl. Comput. (IJCAC) 8(1), 55–79, 2018.

42. A. Firdaus, N. B. Anuar, M. F. Razak, I. A. Hashem, S. Bachok, A. K. Sangaiah, “Root exploit detection and features optimization: mobile device and Blockchain based medical data management,” J. Med. Syst. 42, pp. 1–23, 2018.

43. J. Moubarak, M. Chamoun and E. Filiol, “Developing a K-ary malware using blockchain,” IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), pp. 1–4, 2018. DOI: 10.1109/NOMS.2018.8406331.

44. X. Hu, “Large-Scale Malware Analysis, Detection, and Signature Generation,” The University of Michigan, 2011.

45. M. N. Murti, V. S. Devi, “Feature Extraction and Feature Selection, Introduction to Pattern Recognition and Machine Learning,” IISc Lecture Notes Series, June 2015, pp. 75–110. DOI: 10.1142/9789814335461_0003.
46. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, “DREBIN: Effective and explainable detection of Android malware in your pocket,” In Symposium on Network and Distributed System Security, 2014, p. 1–12.
47. M. S. Rana, S. S. M. M. Rahman, and A. H. Sung, “Evaluation of Tree Based Machine Learning Classifiers for Android Malware Detection,” In Nguyen N., Pimenidis E., Khan Z., Trawiński B. (eds) Computational Collective Intelligence. ICCCI 2018. Lecture Notes in Computer Science, vol 11056. Springer, Cham. DOI: 10.1007/978-3-319-98446-9_35
48. Decision Trees for Classification: A Machine Learning Algorithm, [Internet], Available from: https://bit.ly/3wiuNvb, [Accessed: 2021-07-05].
49. Towards Data Science | The Random Forest Algorithm [Internet], Available from: https://bit.ly/3dLzOG7, [Accessed: 2018-10-29]
50. P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees,” Machine Learning, 63(1), pp. 3-42, 2006.
51. A Comprehensive Guide to Ensemble Learning [Internet], Available from: https://bit.ly/3xlPR58, [Accessed: 2021-07-05].
52. M. S. Rana, C. Gudla, and A. H. Sung, “Evaluating Machine Learning Models for Android Malware Detection: A Comparison Study,” In Proceedings of the 2018 VII International Conference on Network, Communication and Computing (ICNCC 2018).

Association for Computing Machinery, New York, NY, USA, 2018, pp. 17–21. DOI: 10.1145/3301326.3301390.

53. D. H. Wolpert, “Stacked Generalization,” Neural Networks, Vol. 5, pp. 241–259, 1992, DOI: 10.1016/S0893-6080(05)80023-1.
54. Stacked Generalization (Stacking) [Internet], Available from: https://bit.ly/36dD8p8, [Accessed: 2021-07-05].
55. K. M. Ting, and I. H. Witten, “Stacked generalization: when does it work? (Working paper 97/03),” Hamilton, New Zealand: University of Waikato, Department of Computer Science, pp. 866–871, 1997.
56. M. S. Rana, and A. H. Sung, “Evaluation of Advanced Ensemble Learning Techniques for Android Malware Detection,” Vietnam Journal of Computer ScienceVol. 07, No. 02, pp. 145–159, 2020. DOI: 10.1142/S2196888820500086
57. V. Smolyakov, “Ensemble Learning to Improve Machine Learning Results (2017),” [Internet], Available from: https://bit.ly/3hBnZn9, [Accessed: 2021-07-05].
58. E. Lutins, “Towards Data Science | Ensemble Methods in Machine Learning: What are They and Why Use Them?” [Internet], Available from: https://bit.ly/3yqHTYF, [Accessed: 2021-07-05].
59. A. Singh, “A Comprehensive Guide to Ensemble Learning (with Python codes),” [Internet], Available from: https://bit.ly/2UjHSHf, [Accessed: 2021-07-05]
60. M. S. Rana, C. Gudla, and A. H. Sung, “Evaluating Machine Learning Models on the Ethereum Blockchain for Android Malware Detection,” In Arai K., Bhatia R., Kapoor S. (eds) Intelligent Computing. CompCom 2019. Advances in Intelligent Systems and Computing, vol 998. Springer, Cham. DOI: 10.1007/978-3-030-22868-2_34

61. A. B. Kurtulmus, and K. Daniel, “Trustless Machine Learning Contracts; Evaluating and Exchanging Machine Learning Models on the Ethereum Blockchain,” Algorithmia Research,” [Internet], available from: https://bit.ly/3xnDy8G, [Accessed: 2018-09-18].
62. G. Oberoi, “Exploring DeepFakes,” [Internet], Available from: https://bit.ly/3hEDlY0, [Accessed: 2021-07-05].
63. J. Hui, “How deep learning fakes videos (Deepfake) and how to detect it,” [Internet], Available from, https://bit.ly/2UrSvrG, [Accessed: 2021-07-05].
64. I. Goodfellow, J. P. Abadie, M. Mirza, B. Xu, D. W. Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. and Weinberger, K. Q. editors, Advances in Neural Information Processing Systems 27, pp. 2672–2680, 2014.
65. G. Patrini, F. Cavalli, and H. Ajder, “The state of Deepfakes: reality under attack,” Annual Report v.2.3. 2018.
66. J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner, “Face2Face: Real-Time Face Capture and Reenactment of RGB Videos,” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 2387–2395, DOI: 10.1109/CVPR.2016.262.
67. J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks,” 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 2242–2251, DOI: 10.1109/ICCV.2017.244.
68. S. Suwajanakorn, S. M. Seitz, and I. K. Shlizerman, “Synthesizing Obama: learning lip sync from audio,” ACM Transactions on Graphics (TOG), 36(4), 2017.

69. L. Matsakis, “Artificial intelligence is now fighting fake porn,” [Internet], Available from: https://bit.ly/3wlWHGB, [Accessed: 2021-07-05].
70. A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, “Faceforensics: A large-scale video dataset for forgery detection in human faces,” arXiv preprint arXiv:1803.09179.
71. H. Kim, P. Garrido, A. Tewari, W. Xu, J. Thies, M. Niessner, P. Pérez, C. Richardt, M. Zollhöfer, and C. Theobalt, “Deep video portraits,” ACM Transaction Graph. 37, 4, Article 163, 2018. DOI: 10.1145/3197517.3201283.
72. C. Chan, S. Ginosar, T. Zhou, and A. A. Efros, “Everybody Dance Now,” arXiv:1808.07371, 2019.
73. T. Karras, S. Laine, and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks,” 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 2019, pp. 4396–4405, DOI: 10.1109/CVPR.2019.00453
74. Faceswap [Internet], Available from: https://bit.ly/3jLEcZq, [Accessed: 2021-07-05].
75. F. Matern, C. Riess, and M. Stamminger, “Exploiting Visual Artifacts to Expose Deepfakes and Face Manipulations,” 2019 IEEE Winter Applications of Computer Vision Workshops (WACVW), Waikoloa Village, HI, USA, 2019, pp. 83–92, DOI: 10.1109/WACVW.2019.00020.
76. U. A. Ciftci, I. Demir and L. Yin, “FakeCatcher: Detection of Synthetic Portrait Videos using Biological Signals,” In IEEE Transactions on Pattern Analysis and Machine Intelligence, DOI: 10.1109/TPAMI.2020.3009287.

77. X. Li, Y. Lang, Y. Chen, X. Mao, Y. He, S. Wang, H. Xue, and Q. Lu, “Sharp Multiple Instance Learning for DeepFake Video Detection,” arXiv:2008.04585, 2020.
78. G. Wang, J. Zhou, and Y. Wu, “Exposing Deep-faked Videos by Anomalous Co-motion Pattern Detection,” arXiv:2008.04095, 2020.
79. N. T. Do, I. S. Na, and S. H. Kim, “DeepFakes: Forensics Face Detection from GANs Using Convolutional Neural Network,” International Symposium on Information Technology Convergence (ISITC 2018), South Korea, 2018.
80. P. Zhou, X. Han, V. I. Morariu and L. S. Davis, “Two-Stream Neural Networks for Tampered Face Detection,” 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Honolulu, HI, 2017, pp. 1831–1839, DOI: 10.1109/CVPRW.2017.229.
81. X. Yang, Y. Li, and S. Lyu, “Exposing Deep Fakes Using Inconsistent Head Poses,” IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), United Kingdom, 2019, pp. 8261–8265, DOI: 10.1109/ICASSP.2019.8683164.
82. M. A. S. Habeeba, A. Lijiya, and A. M. Chacko, “Detection of Deepfakes Using Visual Artifacts and Neural Network Classifier,” in Favorskaya M., Mekhilef S., Pandey R., Singh N. (eds) Innovations in Electrical and Electronic Engineering. Lecture Notes in Electrical Engineering, vol 661. Springer, Singapore, 2020, pp. 411–422, DOI: 10.1007/978-981-15-4692-1_31.
83. X. Zhang, S. Karaman, and S. F. Chang, “Detecting and simulating artifacts in GAN fake images,” IEEE Workshop on Information Forensics and Security (WIFS), 2019.
84. P. Zhou, X. Han, V. I. Morariu and L. S. Davis, “Learning Rich Features for Image Manipulation Detection,” 2018 IEEE/CVF Conference on Computer Vision and

Pattern Recognition, Salt Lake City, UT, 2018, pp. 1053–1061, DOI: 10.1109/CVPR.2018.00116.

85. K. Chugh, P. Gupta, A. Dhall, and R. Subramanian, “Not made for each other- Audio-Visual Dissonance-based Deepfake Detection and Localization,” arXiv:2005.14405, 2020.
86. H. Qi, Q. Guo, F. J. Xu, X. Xie, L. Ma, W. Feng, Y. Liu, and J. Zhao, “DeepRhythm: Exposing DeepFakes with Attentional Visual Heartbeat Rhythms,” arXiv:2006.07634, 2020.
87. S. Fernandes, S. Raj, E. Ortiz, I. Vintila, M. Salter, G. Urosevic, and S. Jha, “Predicting Heart Rate Variations of Deepfake Videos using Neural ODE,” ICCV Workshops, 2019.
88. J. H. Ortega, R. Tolosana, J. Fierrez, and A. Morales, “DeepFakesON-Phys: DeepFakes Detection based on Heart Rate Estimation,” arXiv:2010.00400, 2020.
89. J. Bappy, C. Simons, L. Nataraj, B. Manjunath, and A. R. Chowdhury, “Hybrid LSTM and Encoder Decoder Architecture for Detection of Image Forgeries,” IEEE Trans. Image Process., vol. 28, no. 7, pp. 3286–3300, July 2019.
90. D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, “Mesonet: a compact facial video forgery detection network,” IEEE International Workshop on Information Forensics and Security, 2018, pp. 1–7.
91. P. Kawa, and P. Syga, “A Note on Deepfake Detection with Low-Resources,” arXiv:2006.05183, 2020.

92. A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, “FaceForensics++: Learning to detect manipulated facial images,” International Conference on Computer Vision (ICCV), 2019.
93. G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, “Densely Connected Convolutional Networks,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261–2269, DOI: 10.1109/CVPR.2017.243.
94. F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” IEEE Conference on Computer Vision and Pattern Recognition, pp. 1800–1807, 2017.
95. A. Khodabakhsh, and C. Busch, “A Generalizable Deepfake Detector based on Neural Conditional Distribution Modelling,” 2020 International Conference of the Biometrics Special Interest Group (BIOSIG), Darmstadt, Germany, pp. 1–5, 2020.
96. Y. Li, M.-C. Chang, and S. Lyu, “In Ictu Oculi: Exposing AI created fake videos by detecting eye,” IEEE Workshop on Information Forensics and Security, 2018.
97. Y. Li and S. Lyu, “Exposing Deepfake videos by detecting face warping artifacts,” IEEE CVPR Workshops, 2019.
98. I. Ganiyusufoglu, L. M. Ngô, N. Savov, S. Karaoglu, and T. Gevers, “Spatio-temporal Features for Generalized Detection of Deepfake Videos,” arXiv preprint arXiv: arXiv:2010.11844, 2020
99. A. Singh, A. S. Saimbhi, N. Singh, and M. Mittal, “DeepFake Video Detection: A Time-Distributed Approach,” SN Computer Science, 212 (1), 2020, DOI: 10.1007/s42979-020-00225-9.

100. I. Kukanov, J. Karttunen, H. Sillanpää, and V. Hautamäki, “Cost Sensitive Optimization of Deepfake Detector,” arXiv preprint arXiv:2012.04199, 2020.

101. A. Haliassos, K. Vougioukas, S. Petridis, and M. Pantic, “Lips Don’t Lie: A Generalisable and Robust Approach to Face Forgery Detection,” arXiv preprint arXiv:2012.07657, 2020.

102. X. Zhu, H. Wang, H. Fei, Z. Lei, and S. Z. Li, “Face Forgery Detection by 3D Decomposition,” arXiv preprint arXiv:2011.09737, 2020.

103. X. Wang, T. Yao, S. Ding, and L. Ma, “Face Manipulation Detection via Auxiliary Supervision,” in Yang H., Pasupa K., Leung A.CS., Kwok J.T., Chan J.H., King I. (eds) Neural Information Processing. ICONIP 2020. Lecture Notes in Computer Science, vol 12532. Springer, Cham, 2020, pp. 313–324, DOI: 10.1007/978-3-030-63830-6_27

104. M. T. Jafar, M. Ababneh, M. Al-Zoube, and A. Elhassan, “Forensics and Analysis of Deepfake Videos,” 11th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2020, pp. 053–058, DOI: 10.1109/ICICS49469.2020.239493.

105. X. Dong, J. Bao, D. Chen, W. Zhang, N. Yu, D. Chen, F. Wen, and B. Guo, “Identity-Driven DeepFake Detection,” arXiv preprint arXiv:2012.03930, 2020.

106. T. Zhao, X. Xu, M. Xu, H. Ding, Y. Xiong, and W. Xia, “Learning to Recognize Patch-Wise Consistency for Deepfake Detection,” arXiv preprint arXiv:2012.09311, 2020.

107. L. Bondi, E. D. Cannas, P. Bestagini, and S. Tubaro, “Training Strategies and Data Augmentations in CNN-based DeepFake Video Detection,” arXiv preprint arXiv:2011.07792, 2020.

108. Z. Hongmeng, Z. Zhiqiang, S. Lei, M. Xiuqing, and W. Yuehan, “A Detection Method for DeepFake Hard Compressed Videos based on Super-resolution Reconstruction Using CNN,” Proceedings of the 2020 4th High Performance Computing and Cluster Technologies Conference & 2020 3rd International Conference on Big Data and Artificial Intelligence, Association for Computing Machinery, New York, NY, USA, pp. 98–103. DOI: 10.1145/3409501.3409542.

109. A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, “Faceforensics: A large-scale video dataset for forgery detection in human faces,” arXiv preprint arXiv:1803.09179.

110. J. Han, and T. Gevers, “MMD based Discriminative Learning for Face Forgery Detection,” Proceedings of the Asian Conference on Computer Vision (ACCV), 2020. (In press)

111. H. Dang, F. Liu, J. Stehouwer, X. Liu and A. K. Jain, “On the Detection of Digital Face Manipulation,” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 2020, pp. 5780–5789, DOI: 10.1109/CVPR42600.2020.00582.

112. H. H. Nguyen, J. Yamagishi, and I. Echizen, “Capsule-forensics: Using Capsule Networks to Detect Forged Images and Videos,” IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2019), Brighton, United Kingdom, 2019, pp. 2307–2311, DOI: 10.1109/ICASSP.2019.8682602.

113. H. Nguyen, J. Yamagishi, and I. Echizen, “Use of a Capsule Network to Detect Fake Images and Videos,” arXiv preprint arXiv:1910.12467, 2019.

114. N. Bonettini, E. D. Cannas, S. Mandelli, L. Bondi, P. Bestagini, and S. Tubaro, "Video Face Manipulation Detection Through Ensemble of CNNs," Computing Research Repository (CoRR), abs/2004.07676, 2020.

115. D. Guera, and E. Delp, "Deepfake video detection using recurrent neural networks," IEEE International Conference on Advanced Video and Signal Based Surveillance, 2018.

116. S. Sohrawardi, A. Chintha, B. Thai, S. Seng, A. Hickerson, R. Ptucha, and M. Wright, "Poster: Towards robust open-world detection of deepfakes," ACM SIGSAC Conference on Computer and Communications Security, 2019.

117. E. Sabir, J. Cheng, A. Jaiswal, W. Abd-Almageed, I. Masi, and P. Natarajan, "Recurrent convolutional strategies for face manipulation detection in videos," CVPR Workshops, pp. 80–87, 2019.

118. S. Tariq, S. Lee, and S. S. Woo, "A Convolutional LSTM based Residual Network for Deepfake Video Detection," arXiv preprint arXiv:2009.07480, 2020.

119. I. Masi, A. Killekar, R. M. Mascarenhas, S. P. Gurudatt, and W. Abd-Almageed, "Two-branch Recurrent Network for Isolating Deepfakes in Videos," 16th European Conference on Computer Vision, August 2020. (In press)

120. A. Chintha, B. Thai, S. J. Sohrawardi, K. Bhatt, A. Hickerson, M. Wright, and R. W. Ptucha, "Recurrent Convolutional Structures for Audio Spoof and Video Deepfake Detection," IEEE Journal of Selected Topics in Signal Processing, vol. 14, no. 5, pp. 1024–1037, Aug. 2020, DOI: 10.1109/JSTSP.2020.2999185.

121. I. Amerini, L. Galteri, R. Caldelli, and A. D. Bimbo, "Deepfake Video Detection through Optical Flow based CNN," ICCV Workshops, 2019.

122. D. Cozzolino, J. Thies, A. R¨ossler, C. Riess, M. Nießner, and L. Verdoliva, "ForensicTransfer: Weakly-supervised domain adaptation for forgery detection," arXiv preprint arXiv:1812.02510, 2018.

123. H. Nguyen, F. Fang, J. Yamagishi, and I. Echizen, "Multi-task learning for detecting and segmenting manipulated facial images and videos," IEEE International Conference on Biometrics: Theory, Applications, and Systems, 2019.

124. M. Du, S. Pentyala, Y. Li, and X. Hu, "Towards generalizable forgery detection with locality-aware autoencoder," arXiv preprint arXiv:1909.05999v1, 2019.

125. L. Trinh, M. Tsang, S. Rambhatla, and Y. Liu, "Interpretable Deepfake Detection via Dynamic Prototypes," arXiv preprint arXiv:2006.15473, 2020.

126. M. Du, S. Pentyala, Y. Li, and X. Hu, "Towards Generalizable Deepfake Detection with Locality-Aware AutoEncoder," 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland. ACM, New York, NY, USA, DOI: 10.1145/3340531.3411892.

127. T. Fernando, C. Fookes, S. Denman, and S. Sridharan, "Exploiting human social cognition for the detection of fake and fraudulent faces via memory networks," arXiv preprint arXiv:1911.07844v1, 2019.

128. K. Zhu, B. Wu, and B. Wang, "Deepfake Detection with Clustering-based Embedding Regularization," 2020 IEEE Fifth International Conference on Data Science in Cyberspace (DSC), Hong Kong, Hong Kong, 2020, pp. 257–264, DOI: 10.1109/DSC50466.2020.00046.

129. P. Charitidis, G. K. Zilos, S. Papadopoulos, and I. Kompatsiaris, “a face preprocessing approach for improved deepfake detection,” arXiv preprint arXiv: 2006.07084, 2020.

130. P. Charitidis, G. K. Zilos, S. Papadopoulos, and I. Kompatsiaris, “Investigating the Impact of Pre-processing and Prediction Aggregation on the DeepFake Detection Task,” arXiv preprint arXiv:2006.07084, 2020.

131. X. Li, K. Yu, S. Ji, Y. Wang, C. Wu, and H. Xue, “Fighting Against Deepfake: Patch&Pair Convolutional Neural Networks (PPCNN),” In Companion Proceedings of the Web Conference 2020 (WWW ‘20). Association for Computing Machinery, New York, NY, USA, pp. 88–89, 2020, DOI: 10.1145/3366424.3382711.

132. C. X. T. Du, L. H. Duong, H. T. Trung, P. M. Tam, N. Q. V. Hung, J. Jo, and T. Nguyen, “Efficient-Frequency: a hybrid visual forensic framework for facial forgery detection,” 18th Australasian Data Mining Conference, December 2020, Canberra, Australia. (In press)

133. D. Cozzolino, A. Rössler, J. Thies, M. Nießner, and L. Verdoliva, “ID-Reveal: Identity-aware DeepFake Video Detection,” arXiv preprint arXiv:2012.02512, 2020.

134. W. Zhang, C. Zhao, and Y. Li, “A Novel Counterfeit Feature Extraction Technique for Exposing Face-Swap Images Based on Deep Learning and Error Level Analysis,” Entropy, 22(2), 249, 2020, pp. 1–17, DOI: 10.3390/e22020249.s.

135. T. Mittal, U. Bhattacharya, R. Chandra, A. Bera, and D. Manocha, “Emotions Don’t Lie: An Audio-Visual Deepfake Detection Method using Affective Cues,” 28th ACM International Conference on Multimedia (MM ’20), October 12–16, 2020, Seattle, WA, USA, ACM, DOI: 10.1145/3394171.3413570.

136. Y. Nirkin, L. Wolf, Y. Keller, and T. Hassner, “DeepFake Detection Based on Discrepancies Between Faces and their Context,” arXiv preprint arXiv:2008.12262, 2020.

137. C. M. Yu, C. T. Chang, and Y. W. Ti, “Detecting Deepfake-Forged Contents with Separable Convolutional Neural Network and Image Segmentation,” arXiv preprint arXiv:2008.12262, 2020.

138. D. Feng, X. Lu, and X. Lin, “Deep Detection for Face Manipulation,” arXiv preprint arXiv:2009.05934, 2020.

139. L. Chai, D. Bau, S. Lim, and P. Isola, “What makes fake images detectable? Understanding properties that generalize,” 16th European Conference on Computer Vision, August 2020. (In press)

140. X. Chang, J. Wu, T. Yang, and G. Feng, “DeepFake Face Image Detection based on Improved VGG Convolutional Neural Network,” 2020 39th Chinese Control Conference (CCC), Shenyang, China, 2020, pp. 7252–7256, DOI: 10.23919/CCC50068.2020.9189596.

141. U. A. Ciftci, I. Demir, and L. Yin, “How Do the Hearts of Deep Fakes Beat? Deep Fake Source Detection via Interpreting Residuals with Biological Signals,” arXiv preprint arXiv:2008.11363, 2020.

142. H. M. Nguyen and R. Derakhshani, “Eyebrow Recognition for Identifying Deepfake Videos,” International Conference of the Biometrics Special Interest Group (BIOSIG 2020), Darmstadt, Germany, 2020, pp. 1–5.

143. M. Koopman, A. M. Rodriguez, and Z. Geradts, “Detection of Deepfake Video Manipulation,” 20th Irish Machine Vision and Image Processing conference (IMVIP 2018), United Kingdom, 2018.

144. B. L. Welch, “The generalization of students’ problem when several different population variances are involved,” Biometrika 34(1/2), pp. 28–35, 1947.

145. L. Guarnera, O. Giudice, and S. Battiato, “DeepFake Detection by Analyzing Convolutional Traces,” 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 2841–2850, DOI: 10.1109/CVPRW50498.2020.00341.

146. S. Agarwal, and L. R. Varshney, “Limits of Deepfake detection: A robust estimation viewpoint,” Thirty-sixth International Conference on Machine Learning (ICML 2019), Long Beach, CA, USA, 2019. (In press)

147. U. M. Maurer, “Authentication theory and hypothesis testing,” IEEE Transactions on Information Theory, vol. 46, no. 4, pp. 1350–1356, July 2000, DOI: 10.1109/18.850674.

148. H. R. Hasan, and K. Salah, “Combating Deepfake Videos Using Blockchain and Smart Contracts,” IEEE Access, vol. 7, pp. 41596–41606, 2019, DOI: 10.1109/ACCESS.2019.2905689.

149. IPFS powers the Distributed Web [Internet], Available from: https://ipfs.io/, [Accessed: 2021-07-05].

150. C. C. Ki Chan, V. Kumar, S. Delaney and M. Gochoo, "Combating Deepfakes: Multi-LSTM and Blockchain as Proof of Authenticity for Digital Media," 2020

IEEE/ITU International Conference on Artificial Intelligence for Good (AI4G), 2020, pp. 55–62, DOI: 10.1109/AI4G50087.2020.9311067.

151. P. Korshunov, and S. Marcel, “DeepFakes: A New Threat to Face Recognition? Assessment and Detection,” arXiv preprint arXiv:1812.08685, 2018.

152. A. Radford, L. Metz, and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” In ICLR, 2016.

153. D. Kingma, and P. Dhariwal, “Glow: Generative flow with invertible 1x1 convolutions,” arXiv:1807.03039 (2018).

154. Y. Choi, M. Choi, M. Kim, J. W. Ha, S. Kim, and J. Choo, “StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

155. I. Chingovska, A. Anjos, and S. Marcel, “On the effectiveness of local binary patterns in face anti-spoofing,” International Conference of Biometrics Special Interest Group (BIOSIG), pp. 1–7, 2012.

156. N. Rahmouni, V. Nozick, J. Yamagishi, and I. Echizen, “Distinguishing computer graphics from natural images using convolution neural networks,” 2017 IEEE Workshop on Information Forensics and Security (WIFS), pp. 1–6, 2017.

157. DeepFaceLab [Internet], Available from: https://bit.ly/3dJopqe, [Accessed: 2021-07-05].

158. D. M. Montserrat, H. Hao, S. K. Yarlagadda, S. Baireddy, R. Shao, J. Horváth, E. Bartusiak, J. Yang, D. Güera, F. Zhu, and E. J. Delp, “Deepfakes Detection with Automatic Face Weighting,” arXiv preprint arXiv:2004.12027, 2020.

159. Contributing Data to Deepfake Detection Research [Internet], Available from: https://bit.ly/36g1xKQ, [Accessed: 2021-07-05].

160. Z. Liu, P, Luo, X. Wang, and X. Tang, "Large-scale CelebFaces Attributes (CelebA) Dataset," Multimedia Laboratory, The Chinese University of Hong Kong, 2016.

161. Y. Li, X. Yang, P. Sun, H. Qi and S. Lyu, "Celeb-DF: A Large-Scale Challenging Dataset for DeepFake Forensics," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 3204–3213, DOI: 10.1109/CVPR42600.2020.00327.

162. B. Dolhansky, J. Bitton, B. Pflaum, J. Lu, R. Howes, M. Wang and C. C. Ferrer, "The DeepFake Detection Challenge (DFDC) Dataset," arXiv preprint arXiv:2006.07397, 2020.

163. L. Jiang, W. Wu, R. Li, C. Qian, and C. C Loy, "DeeperForensics-1.0: A Large-Scale Dataset for Real-World Face Forgery Detection," arXiv preprint arXiv:2001.03024, 2020.

164. B. Zi, M. Chang, J. Chen, X. Ma, and Y. G. Jiang, "WildDeepfake: A Challenging Real-World Dataset for Deepfake Detection," 28th ACM International Conference on Multimedia (MM' 20), Association for Computing Machinery, New York, NY, USA, pp. 2382–2390. DOI: 10.1145/3394171.3413769.

165. A. Khodabakhsh, R. Ramachandra, K. Raja, P. Wasnik, and C. Busch, "Fake Face Detection Methods: Can They Be Generalized?" 2018 International Conference of the Biometrics Special Interest Group (BIOSIG), Darmstadt, 2018, pp. 1–6, DOI: 10.23919/BIOSIG.2018.8553251.

166. P. Gupta, K. Chugh, A. Dhall, and R. Subramanian, “The eyes know it: FakeET -- An Eye-tracking Database to Understand Deepfake Perception,” arXiv preprint arXiv:2006.06961, 2020.

167. L. Li, J. Bao, H. Yang, D. Chen, and F. Wen, “Advancing high fidelity identity swapping for forgery detection,” IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 5074–5083, 2020.

168. J. Hui, “How deep learning fakes videos (Deepfake) and how to detect it?” [Internet], Available from: https://bit.ly/3hhfmyU, [Accessed: 2021-07-05].

169. The Power of Ensembles in Deep Learning [Internet], Available from: https://bit.ly/3hjZtrA, [Accessed: 2021-07-05].

170. M. S. Rana and A. H. Sung, “DeepfakeStack: A Deep Ensemble-based Learning Technique for Deepfake Detection,” 2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), 2020, pp. 70–75, DOI: 10.1109/CSCloud-EdgeCom49738.2020.00021.

171. ImageNet [Internet], Available from: https://bit.ly/3xjovgd, [Accessed: 2020-07-05].

172. Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, “Greedy layer-wise training of deep networks,” Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS’06), Cambridge, MA, USA, pp. 153–160, December 2006.

173. L. Guarnera, O. Giudice and S. Battiato, “Fighting Deepfake by Exposing the Convolutional Traces on Images,” arXiv: 2008.04095v1, 2020.

174. N. Koleva, “When and When Not to Use Deep Learning,” [Internet], Available from: https://bit.ly/3isoZdd, [Accessed: 2021-07-05].

175. M. N. Murti and V. S. Devi, “Feature Extraction and Feature Selection, Introduction to Pattern Recognition and Machine Learning,” IISc Lecture Notes Series, June 2015, pp. 75–110.

176. R. M. Haralick, “Statistical and structural approaches to texture,” Proceedings of the IEEE, Vol. 67, No. 5, pp. 786–804, 1979.

177. L. Weng, “Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS” [Internet], Available from: https://bit.ly/3oRFtxJ, [Accessed: 2021-07-05].

178. R. Ahmed, “A Take on HOG Feature Descriptor,” [Internet], Available from: https://bit.ly/2LutMyU, [Accessed: 2021-07-05].

179. A. Singh, “Feature Engineering for Images: A Valuable Introduction to the HOG Feature Descriptor,” [Internet], Available from: https://bit.ly/2LtjVt9, [Accessed: 2021-07-05].

180. F. Alamdar and M. R. Keyvanpour, “A New Color Feature Extraction Method Based on QuadHistogram,” Procedia Environmental Sciences, Volume 10, 2011, pp. 777–783.

181. J. Žunić, K. Hirota and P. L. Rosin, “A Hu moment invariant as a shape circularity measure, Pattern Recognition,” Volume 43, Issue 1, 2010, pp. 47–57.

182. Z. Huang, and J. Leng, “Analysis of Hu’s moment invariants on image scaling and rotation,” 2nd International Conference on Computer Engineering and Technology, Chengdu, 2010.

183. S. Raschka, “Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning,” arXiv: 1811.12808, 2020.

184. L. Guarnera, O. Giudice and S. Battiato, “Fighting Deepfake by Exposing the Convolutional Traces on Images,” arXiv: 2008.04095v1, 2020.

.