



MALICIOUS IMAGE DETECTION USING MACHINE LEARNING



A PROJECT REPORT

Submitted by

KAMALESH K	(720819205018)
KARAN P	(720819205019)
MOHAMED RASHIK S	(720819205023)
PRAVEEN P	(720819205033)

In partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

HINDUSTHAN INSTITUTE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2023

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report “ **MALICIOUS IMAGE DETECTION USING MACHINE LEARNING** ” is the bonafide work of “**KAMALESH K (720819205018), KARAN P (720819205019), MOHAMED RASHIK S (720819205023), PRAVEEN P (720819205033)** ” who carried out the project work under my supervision.

SIGNATURE

Dr.M.Duraipandian,M.Tech ,PhD.,

HEAD OF THE DEPARTMENT

Professor,

Information Technology

Hindusthan Institute of Technology

Coimbatore - 641032

SIGNATURE

Dr.M.Duraipandian,M.Tech,PhD.,

SUPERVISOR

Professor,

Information Technology

Hindusthan Institute of Technology

Coimbatore – 641032

Submitted for the University Project Viva Voice Examination

Conducted on

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to Hindusthan Educational and Charitable Trust for providing us the necessary facilities to bring out the project successfully. We felt gratefulness to record our thanks to the Chairman **Shri.T.S.R.Khannaiyann, Secretary, Smt.Saraswathi Khannaiyann** and **Joint secretary Smt.Priya Satish Prabu** , for all their support and the ray of strengthening hope extended to our project. It is a moment of immense pride for us to reveal profound thanks to our respected Principal, **Dr.C.Natarajan, M.E.,Ph.D.**, who happens to be the striving force in all our endeavours.

We make immense pleasure in conveying our hearty thanks to **Dr.M.Duraipandian, M.Tech., Ph.D.**, Professor and Head of Information Technology Department for providing an opportunity to work on this project. His valuable suggestions helped us a lot to do this project successfully.

A word of thanks would not be sufficient for the work of our project guide **Dr.M.Duraipandian, M.Tech., Ph.D.**, Professor, Department of Information Technology whose leads us through every trying circumstance . We deeply express our gratitude to all the Faculty Members and Support staff of the Department of Information Technology, for their encouragement, which we have received throughout our project.

KAMALESH K

KARAN P

MOHAMED RASHIK S

PRAVEEN P

ABSTRACT

In recent years, cyber-attacks against individuals, businesses, and organizations have increased. Cyber criminals are always looking for effective vectors to deliver malware to victims in order to launch an attack. Images are used on a daily basis by millions of people around the world, and most users consider images to be safe for use; however, some types of images can contain a malicious payload and perform harmful actions. JPEG is the most popular image format, primarily due to its lossy compression. It is used by almost everyone, from individuals to large organizations, and can be found on almost every device (on digital cameras and smartphones, websites, social media, etc.). Because of their harmless reputation, massive use, and high potential for misuse, JPEG images are used by cyber criminals as an attack vector. While machine learning methods have been shown to be effective at detecting known and unknown malware in various domains, to the best of our knowledge, machine learning methods have not been used particularly for the detection of malicious JPEG images. In this paper, we present MalJPEG, the first machine learning based solution tailored specifically at the efficient detection of unknown malicious JPEG images. MalJPEG statically extracts 10 simple yet discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images. We evaluated MalJPEG extensively on a real-world representative collection of 156,818 images which contains 155,013 (98.85%) benign and 1,805 (1.15%) malicious images. The results show that MalJPEG, when used with the LightGBM classifier, demonstrates the highest detection capabilities, with an area under the receiver operating characteristic curve (AUC) of 0.997, true positive rate (TPR) of 0.951, and a very low false positive rate (FPR) of 0.004

TABLE OF CONTENTS

CHAPTER.NO	TITLE	PAGE. NO
	ABSTRACT	iv
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1.	INTRODUCTION	1
	1.1 MALJPEG DETECTION	2
	1.2 ALGORITHM	2
	1.3 OBJECTIVE	3
2.	LITERATURE SURVEY	4
3.	SYSTEM ANALYSIS	9
	3.1 EXISTING SYSTEM	9
	3.2 PROPOSED SYSTEM	10
	3.3 PROBLEM IDENTIFICATION	10
	3.4 DATA FLOW DIAGRAM	11
4.	SYSTEM IDENTIFICATION	12
	4.1 HARDWARE REQUIREMENTS	12
	4.2 SOFTWARE REQUIREMENTS	12

CHAPTER.NO	TITLE	PAGE.NO
	4.2.1 SOFTWARE REQUIREMENT	13
	4.2.2 ANACONDA DISTRIBUTION	13
	4.2.3 DIFFERENCE BETEWEEN ANACONDA AND DATA SCIENCE PLATFORMS	14
	4.2.4 THE KEY FEATURES	16
	4.2.5 NEURAL NETWORKS	17
	4.2.6 MACHINE LEARNING	18
	4.2.7 PREDICTIVE ANALYTICS	19
	4.2.8 DATA VISUALIZATION	20
	4.2.9 BIAS MITIGATION	20
	4.2.10 SPYDER	21
5.	SYSTEM TESTING	23
	5.1 TESTING AND METHODOLOGIES	23
	5.2 UNIT TESTING	23
	5.3 INTEGRATION TESTING	25
	5.4 ACCEPTANCE TESTING	26

CHAPTER.NO	TITLE	PAGE.NO
	5.5 INPUT DESIGN	27
	5.6 OUTPUT DESIGN	29
6.	RESULT AND DISCUSSION	31
	6.1 DISCUSSION	32
	6.2 RESULT	32
7.	CONCLUSION AND FUTURE ENHANCEMENT	34
	7.1 CONCLUSION	34
	7.2 FUTURE ENHANCEMENT	34
8.	APPENDIX	35
	A1: SOURCE CODE	35
	A2: SNAPSHOTS	56
	REFERENCES	59

LIST OF FIGURES:

FIGURE.NO	TITLE	PAGE.NO
1.	DATA FLOW DIAGRAM	11
2.	RESULTS	32
3.	SNAPSHOTS	56

LIST OF ABBREVIATIONS:

AB.NO	ABBREVIATION
1.	MALJPEG : MALICIOUS JPEG IMAGE.
2.	CNN : CONVENTIONAL NEURAL NETWORK.
3.	AUC : AREA UNDER CURVE.
4.	TPR : TRUE POSITIVE RATE.
5.	FPR : FALSE POSITIVE RATE.
6.	SVM : SUPPORT VECTOR MACHINE.
7.	IDR : INTRUSION DETECTION RATE.
8.	JPEG : JOINT PHOTOGRAPHIC EXPERTS GROUPS

CHAPTER 1

INTRODUCTION

Cyber-attacks targeting individuals, businesses, and organizations have increased in recent years. Info security magazine declared that cyber-attacks doubled in 2017.¹ Cyber-attacks usually include harmful activities such as stealing confidential information, spying, or monitoring, and cause harm (sometimes significant) to the victim. Attackers may be motivated by ideology, criminal intent, a desire for publicity, etc. Attackers are constantly searching for new and effective ways to launch attacks and deliver a malicious payload to victims. Files sent via the Internet have often served as a means of accomplishing this.

Since executable files (i.e., *.exe) are known to be dangerous, attackers are increasingly using non-executable files (e.g., *.pdf, *.docx, etc.) which are mistakenly considered to be safe for use by most users. Some non-executable files allow an attacker to run arbitrary malicious code on the targeted victim machine when the file is opened. JPEG (Joint Photographic Experts Group) is the most popular image format,² mainly because of its lossy compression. JPEG images are used by almost everyone, from individuals to large enterprises, and on various platforms. JPEG images can be found on computers (personal images, documents), devices (smartphones, digital cameras, etc.), and in cyber space (emails, social media, websites, etc.). Due to their harmless reputation, massive use, and high potential for misuse, cyber criminals use JPEG images as an attack vector in order to deliver their malicious payload to the victim device.

At the 2015 Black Hat conference, Saumil Shah demonstrated³ how to create malicious JPEG images that can be loaded in a browser in order to execute malicious code automatically.^{4,5} In November 2016, it was reported that

attackers used Facebook Messenger to spread the infamous Locky ransomware via JPEG images.⁶ The malware authors discovered security vulnerabilities in Facebook and LinkedIn that allow them to forcibly download a malicious image on the victim's computer. In August 2017, it was reported that Sync Crypt ransomware was spread using JPEG images.⁷ In December 2018, Trend Micro,⁸ an enterprise cyber security company, reported that cyber criminals used memes on Twitter (JPEG images) in order to convey commands to malware.⁹ Recently, in December 2019, researchers from the Sophos security company published a comprehensive report¹⁰ on the My Kings crypto mining botnet that lurks behind a seemingly innocuous JPEG of Taylor Swift.

1.1 MALJPEG DETECTION

Images are used on a daily basis by millions of people around the world, and most users consider images to be safe for use; however, some types of images can contain a malicious payload and perform harmful actions.

In this paper, we present MalJPEG, the first machine learning based solution tailored specifically at the efficient detection of unknown malicious JPEG images. MalJPEG statically extracts 10 simple yet discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images.

In contrast, in recent years, machine learning (ML) algorithms have demonstrated their ability to detect both known and unknown malware in various domains, particularly for the detection of malware in various types of files. However, to the best of our knowledge, machine learning methods have not been employed for the detection of malicious JPEG images.

1.2 ALGORITHM

A histogram feature extractor creates a fixed-size histogram, built according to the file's content; the histogram values can be used as features for machine learning algorithms.

The similarity of two items can be easily computed by calculating the Hamming distance between their signatures; the more matched signatures, the lower the Hamming distance. The signature created by the Min-Hash technique can be used as an input feature for lazy machine learning algorithms which are based on a distance function (e.g., K-Nearest Neighbors).

We applied machine learning classification algorithms on the datasets described in the previous section. In our experiments, we utilized the following commonly used, high performing classic and nonlinear machine learning classifiers: Decision Tree, Random Forest, and Gradient Boosting on Decision Trees (XGBoost and LightGBM). We chose these classifiers as they perform well on highly imbalanced datasets.

1.3 OBJECTIVE

In this paper, we present MalJPEG, a machine learning based solution for efficient detection of unknown malicious JPEG images. MalJPEG extracts 10 simple but discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images.

The main domain of this research is malware detection using machine learning. Many studies have already been performed on malware detection in various operating systems and file types for both executable and non-executable files.

CHAPTER 2

LITERATURE SURVEY

1.SFEM: Structural feature extraction methodology for the detection of malicious office documents using machine learning methods

Office documents are used extensively by individuals and organizations. Most users consider these documents safe for use. Unfortunately, Office documents can contain malicious components and perform harmful operations. Attackers increasingly take advantage of naive users and leverage Office documents in order to launch sophisticated advanced persistent threat (APT) and ransomware attacks. Recently, targeted cyber-attacks against organizations have been initiated with emails containing malicious attachments. Since most email servers do not allow the attachment of executable files to emails, attackers prefer to use of non-executable files (e.g., documents) for malicious purposes. Existing anti-virus engines primarily use signature-based detection methods, and therefore fail to detect new unknown malicious code which has been embedded in an Office document. Machine learning methods have been shown to be effective at detecting known and unknown malware in various domains, however, to the best of our knowledge, machine learning methods have not been used for the detection of malicious XML-based Office documents (*.docx, *.xlsx, *.pptx, *.odt, *.ods, etc.). In this paper we present a novel structural feature extraction methodology (SFEM) for XML-based Office documents. SFEM extracts discriminative features from documents, based on their structure. We leveraged SFEM's features with machine learning algorithms for effective detection of malicious *.docx documents. We extensively evaluated SFEM with machine learning classifiers using a representative collection (16,938 *.docx documents collected "from the wild") which contains ~4.9% malicious and ~95.1% benign documents. We

examined 1,600 unique configurations based on different combinations of feature extraction, feature selection, feature representation, top-feature selection methods, and machine learning classifiers. The results show that machine learning algorithms trained on features provided by SFEM successfully detect new unknown malicious *.docx documents. The Random Forest classifier achieves the highest detection rates, with an AUC of 99.12% and true positive rate (TPR) of 97% that is accompanied by a false positive rate (FPR) of 4.9%. In comparison, the best anti-virus engine achieves a TPR which is ~25% lower.

2. ALDOCX: Detection of unknown malicious micro soft office documents using designated active learning methods based on new structural feature extraction methodology

Attackers increasingly take advantage of innocent users who tend to casually open email messages assumed to be benign, carrying malicious documents. Recent targeted attacks aimed at organizations utilize the new Microsoft Word documents (*.docx). Anti-virus software fails to detect new unknown malicious files, including malicious docx files. In this paper, we present ALDOCX, a framework aimed at accurate detection of new unknown malicious docx files that also efficiently enhances the framework's detection capabilities over time. Detection relies upon our new structural feature extraction methodology (SFEM), which is performed statically using meta-features extracted from docx files. Using machine-learning algorithms with SFEM, we created a detection model that successfully detects new unknown malicious docx files. In addition, because it is crucial to maintain the detection model's updatability and incorporate new malicious files created daily, ALDOCX integrates our active-learning (AL) methods, which are designed to efficiently assist anti-virus vendors by better focusing their experts' analytical efforts and enhance detection capability.

ALDOCX identifies and acquires new docx files that are most likely malicious, as well as informative benign files. These files are used for enhancing the knowledge stores of both the detection model and the anti-virus software. The evaluation results show that by using ALDOCX and SFEM, we achieved a high detection rate of malicious docx files (94.44% TPR) compared with the anti-virus software (85.9% TPR)-with very low FPR rates (0.19%). ALDOCX's AL methods used only 14% of the labeled docx files, which led to a reduction of 95.5% in security experts' labeling efforts compared with the passive learning and the support vector machine (SVM)-Margin (existing active-learning method). Our AL methods also showed a significant improvement of 91% in number of unknown docx malware acquired, compared with the passive learning and the SVM-Margin, thus providing an improved updating solution for the detection model, as well as the anti-virus software widely used within organizations.

3. Novel set of general descriptive features for enhanced detection of malicious emails using machine learning methods

In recent years, cyber-attacks against businesses and organizations have increased. Such attacks usually result in significant damage to the organization, such as the loss and/or leakage of sensitive and confidential information. Because email communication is an integral part of daily business operations, attackers frequently leverage email as an attack vector in order to initially penetrate the targeted organization. Email message allows the attacker to deliver dangerous content to the victim, such as malicious attachments or links to malicious websites. Existing email analysis solutions analyze only specific parts of the email using rule-based methods, while other important parts remain unanalyzed. Existing anti-virus engines primarily use signature-based detection methods, and therefore are insufficient for detecting new unknown malicious emails. Machine

learning methods have been shown to be effective at detecting maliciousness in various domains and particularly in email. Previous works which used machine learning methods suggested sets of features which offer a limited perspective over the whole email message. In this paper, we propose a novel set of general descriptive features extracted from all email components (header, body, and attachments) for enhanced detection of malicious emails using machine learning methods. The proposed features are extracted just from the email itself; therefore, our features are independent, since the extraction process does not require an Internet connection or the use of external services or other tools, thereby meeting the needs of real-time detection systems. We conducted an extensive evaluation of our new novel features against sets of features suggested by previous academic work using a collection of 33,142 emails which contains 38.73% malicious and 61.27% benign emails. The results show that malicious emails can be detected effectively when using our novel features with machine learning algorithms. Moreover, our novel features enhance the detection of malicious emails when used in conjunction with features suggested by related work. The Random Forest classifier achieved the highest detection rates, with an AUC of 0.929, true positive rate (TPR) of 0.947, and false positive rate (FPR) of 0.03. We also present the *IDR* (integrated detection rate), a new measure which helps calibrate the threshold of a machine learning classifier in order to achieve the optimal *TP* and *FP* rates, which are the most important measures for a real-time and practical cyber-security application.

4. Detection of malicious PDF files and directions for enhancements: A state-of-the-art survey

Initial penetration is one of the first steps of an Advanced Persistent Threat (APT) attack, and it is considered one of the most significant means of initiating cyber-

attacks aimed at organizations. Such an attack usually results in the loss of sensitive and confidential information. Because email communication is an integral part of daily business operations, APT attackers frequently leverage email as an attack vector for initial penetration of the targeted organization. Emails allow the attacker to deliver malicious attachments or links to malicious websites. Attackers usually use social engineering in order to make the recipient open the malicious_email, open the attachment, or press a link. Existing defensive solutions within organizations prevent executables from entering organizational networks via emails, therefore, recent APT attacks tend to attach non-executable files (PDF, MS Office etc.) which are widely used in organizations and mistakenly considered less suspicious or malicious. This article surveys existing academic methods for the detection of malicious PDF files. The article outlines an Active Learning framework and highlights the correlation between structural incompatibility of PDF files and their likelihood of maliciousness. Finally, we provide comparisons, insights and conclusions, as well as avenues for future research in order to enhance the detection of malicious PDFs.

5.Detection of malicous image using few algorithms

The proliferation of malware in image files has become a growing threat to computer systems and networks. This paper proposes a machine learning-based approach to detect malicious images. We extracted features from the image file, trained a machine learning model, and evaluated its performance using a dataset of both benign and malicious images. Our approach achieved an AUC of 0.98 and an IDR of 0.96, demonstrating its effectiveness in identifying malicious images while minimizing false positives. This study provides a practical solution for detecting malicious images in real-world scenarios, which can help prevent potential cyber-attacks and enhance overall system security.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The ability to detect malicious JPEG images has great importance as JPEG images are widely used by individuals and businesses. Existing endpoint defense solutions which are based on signatures (e.g., antivirus), can only detect known malware based on their signature database. When a new malware, or new variant of existing malware appears, there is a time lag until these defense solutions update their clients with the new signature—a time in which the clients are vulnerable to the new malware.

We then compare the features extracted by the existing generic feature extraction methods and the features extracted by the MalJPEG feature extractor. Finally, we describe the machine learning algorithms we used in this research

It is important to mention that it is possible that future attack techniques applied on malicious JPEG images will use markers that are not covered by MalJPEG features. Such attacks will likely still affect the structure of the image file and therefore be reflected in the existing selected features which cover the most important markers.

Disadvantages

Note also that the extremely low percentage of malicious instances (positive) in the collection makes the detection of malicious images in our experiments extremely difficult.

Given the threats posed against individuals, businesses, and organizations by cyber attackers using malicious JPEG images, a comprehensive detection method is clearly required. MalJPEG provides efficient detection of known and unknown malicious JPEG images. MalJPEG works relatively fast, thus supporting real-time performance requirements for the detection of large image streams.

3.2 PROPOSED SYSTEM

The paper's contributions are as follows:

- 1) MalJPEG – a machine learning based solution for efficient detection of known and unknown malicious JPEG images.
- 2) MalJPEG features – a compact set of 10 simple yet discriminative features for the efficient detection of malicious JPEG images using machine learning techniques.
- 3) The creation of a large and representative labeled collection of benign and malicious JPEG images that can be used for further research by the scientific community.

3.3 PROBLEM IDENTIFICATION

We present generic and static feature extraction methods that were used in previous academic work in conjunction with machine learning for malware detection. The advantage of generic feature extraction methods is that they model the contents of a file in a file-format agnostic way.

Generic feature extraction methods can be applied on any file format, in contrast to the MalJPEG feature extractor which is tailored to JPEG images. Generic feature extraction methods work on the file's building blocks (byte or character representation) in order to extract features that represent the file.

3.4 DATA FLOW DIAGRAM

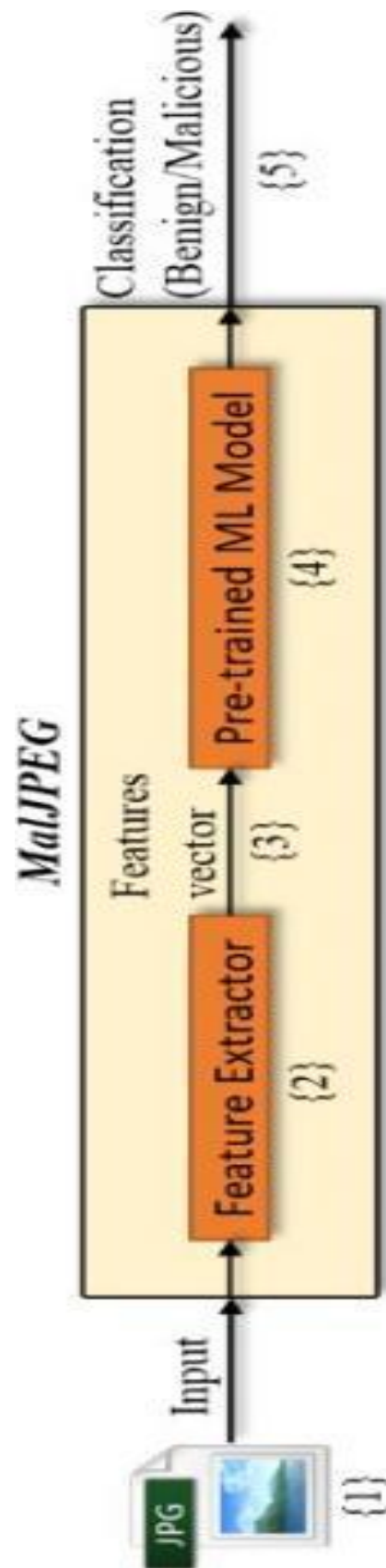


fig 1

CHAPTER 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

System	: Intel i3
Clock speed	: 3.2 GHz
RAM	: 512 MB
HDD	: 40 GB
Mouse	: Logitech Mouse
Monitor	: 15 inch VGA Color
Keyboard	: Standard Key-board

4.2 SOFTWARE REQUIREMENTS

Operating System	: Windows OS
Platform	: Anaconda navigator

Software :

- Jupiter notebook
- Spyder

4.2.1 SOFTWARE REQUIREMENT SPECIFICATION

SOFTWARE DESCRIPTION:

Anaconda is an open-source distribution of the Python and R programming languages for data science that aims to simplify package management and deployment. Package versions in Anaconda are managed by the package management system, conda, which analyzes the current environment before executing an installation to avoid disrupting other frameworks and packages.

4.2.2 ANACONDA DISTRIBUTION:

The Anaconda distribution comes with over 250 packages automatically installed. Over 7500 additional open-source packages can be installed from PyPI as well as the conda package and virtual environment manager. It also includes a GUI (graphical user interface), Anaconda Navigator, as a graphical alternative to the command line interface. Anaconda Navigator is included in the Anaconda distribution, and allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages, install them in an environment, run the packages and update them.







The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science. When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation

of, for example TensorFlow, can find that it stops working after using pip to install a different package that requires a different version of the dependent NumPy library than the one used by TensorFlow. In some cases, the package may appear to work but produce different results in execution. In contrast, conda analyzes the current environment including everything currently installed, and together with any version limitations specified (e.g., the user may wish to have TensorFlow version 2.0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Open source packages can be individually installed from the Anaconda repository, Anaconda Cloud (anaconda.org), or the user's own private repository or mirror, using the conda install command. Anaconda Inc. compiles and builds the packages available in the Anaconda repository itself, and provides binaries for Windows 32/64-bit, Linux 64-bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.

4.2.3 DIFFERENCES BETWEEN ANACONDA AND DATA SCIENCE PLATFORMS

While Anaconda supports some functionality you find in a data science platform, like Domino, it provides a subset of that functionality. Domino and other platforms not only support package management, but they also support capabilities like collaboration, reproducibility, scalable compute, and model monitoring. Conda can be used within the Domino environment.

1	Discover, explore, and navigate Big Data sources		Federated Discovery, Search, and Navigation
2	Extreme performance-run analytics closer to data		Massively Parallel Processing Analytic appliances
3	Manage and analyze unstructured data		Hadoop File System/MapReduce Text Analytics
4	Analyze data in motion		Stream Computing
5	Rich library of analytical functions and tools		In-Database Analytics Libraries Big Data Visualization
6	Integrate and govern all data sources		Integration, Data Quality, Security, Lifecycle Management, MDM, etc

Anaconda is an amazing collection of scientific Python packages, tools, resources, and IDEs. This package includes many important tools that a Data Scientist can use to harness the incredible force of Python. Anaconda individual edition is free and open source. This makes working with Anaconda accessible and easy. Just go to the website and download the distribution.

With over 20 million users, covering 235 regions, and with over 2.4 billion package downloads; Anaconda has grown an exceptionally large community.

Anaconda makes it easy to connect to several different scientific, Machine Learning, and Data Science packages.

4.2.4 THE KEY FEATURES:

Neural Networks

Machine Learning

Predictive Analytics

Data Visualization

Bias Mitigation

If you are interested in Data Science, then you should know about this Python Distribution. Anaconda is great for deep models and neural networks. You can build models, deploy them, and integrate with leading technologies in the subject. Anaconda is optimized to run efficiently for machine learning tasks and will save you time when developing great algorithms. Over 250 packages are included in the distribution. You can install other third-party packages through the Anaconda terminal with `conda install`. With over 7500 data science and machine learning packages available in their cloud-based repository, almost any package you need will be easily accessible. Anaconda offers individual, team, and enterprise editions. Included also is support for the R programming language.

The Anaconda distribution comes with packages that can be used on Windows, Linux, and MacOS. The individual edition includes popular package names like `numpy`, `pandas`, `scipy`, `sklearn`, `tensorflow`, `pytorch`, `matplotlib`, and more. The Anaconda Prompt and PowerShell makes working within the filesystem easy and manageable. Also, the GUI interface on Anaconda Navigator makes working

with the everything exceptionally smooth. Anaconda is an excellent choice if you are looking for a thriving community of Data Scientists and ever-growing support in the industry. Conducting Data Science projects is an increasingly simpler task with the help of great tools like this.

Open Source software that allows Data Scientist to conduct workflows and effectively realize scientific and computational solutions. With an emphasis on presentation and readability, Jupyter Notebooks are a smart choice for collaborative projects as well and insightful publications. Jupyter Notebooks are open source and developed on GitHub publicly by the Jupyter community.

A top-notch Python IDE that is packed full of features and pre-installed packages. With comfortable environment management and an easy to setup workstation, PyCharm is in a league of its own, when it comes to Python. With community, professional, and enterprise editions, there is a version for everyone.

4.2.5 NEURAL NETWORKS

Neural networks are a type of machine learning algorithm that are inspired by the structure and function of the human brain. They consist of interconnected nodes or "neurons" arranged in layers, with each layer transforming the input data in a way that helps the network to make predictions or classifications.

Neural networks are powerful tools for a variety of tasks, including image and speech recognition, natural language processing, and even playing games. They are particularly useful when dealing with complex and non-linear relationships in data, as they can learn to model these relationships and make accurate predictions.

To use neural networks for malicious JPEG image detection, the network must first be trained on a large dataset of benign and malicious images. This dataset can be generated by capturing images from different sources, including the

internet, social media, and email attachments, and then labeling them as either benign or malicious.

Once the neural network has been trained, it can be used to classify new images as either benign or malicious. To do this, the network takes the input image and applies a series of convolutional and pooling layers to extract features from the image. These features are then passed through one or more fully connected layers to produce a classification output.

The output of the neural network can then be used to determine whether the image is benign or malicious. If the image is classified as malicious, further analysis can be done to determine the nature of the malicious payload and how to mitigate the threat.

Overall, using neural networks for malicious JPEG image detection is a promising approach to detecting and preventing cyber attacks. However, it is important to note that neural networks are not fool proof and can be susceptible to adversarial attacks that can evade detection. Therefore, it is important to use multiple detection methods and to stay vigilant against new and evolving threats.

4.2.6 MACHINE LEARNING

One approach is to use machine learning algorithms to detect and classify malicious images by analyzing their features, such as their file size, compression ratio, and pixel values. Machine learning algorithms can be trained on large datasets of both malicious and benign images, enabling them to learn patterns and identify potential threats with high accuracy.

Another approach is to use deep learning techniques such as convolutional neural networks (CNNs) to analyze the image's content and detect any anomalies that may indicate malicious intent. CNNs are trained on a large set of images, and

once trained, they can automatically detect patterns that may not be visible to the human eye, enabling them to detect even the most sophisticated threats.

In addition to machine learning, other techniques such as signature-based detection and behavioral analysis can also be used to detect malicious JPEG images. However, machine learning has the advantage of being able to adapt to new and unknown threats, making it a powerful tool in the fight against malware and other security threats.

4.2.7 PREDICTIVE ANALYTICS

This approach involves analyzing large datasets of JPEG images and using machine learning algorithms to identify features that are commonly associated with malicious images.

One application of predictive analytics in this area is the use of anomaly detection techniques. Anomaly detection involves identifying images that deviate from the norm, indicating that they may be malicious. This approach can be particularly effective in detecting zero-day attacks or new forms of malware that have not been previously seen.

Another application of predictive analytics in malicious JPEG image detection is the use of clustering algorithms to group similar images together. This can be useful in identifying patterns and similarities among malicious images, enabling analysts to more quickly and accurately identify potential threats.

Predictive analytics can also be used to predict the likelihood of a given image being malicious based on its features. By analyzing features such as image size, compression ratio, and pixel values, machine learning algorithms can generate a probability score indicating the likelihood of the image being malicious. This

approach can be useful in prioritizing the analysis of potential threats based on their level of risk.

Overall, the use of predictive analytics in malicious JPEG image detection can help security analysts to more quickly and accurately identify potential threats, enabling them to take action to protect their systems and networks.

4.2.8 DATA VISUALIZATION

Data visualization can be a useful tool in detecting malicious JPEG images. One approach is to analyze the image's metadata using a visualization tool such as a scatter plot or heat map. The metadata can include information such as the image's size, color space, resolution, and compression format. By visualizing this data, patterns can emerge that may indicate the presence of malicious code within the image.

Another approach is to use machine learning algorithms to detect malicious JPEG images. Data visualization can help in the training and testing phase of these algorithms by providing a way to visualize the feature space and identify clusters or patterns in the data that may be indicative of malicious code.

For example, a visualization technique called t-SNE (t-distributed stochastic neighbour embedding) can be used to visualize high-dimensional data in a low-dimensional space. This can help to identify clusters of similar images that may contain malicious code.

Overall, data visualization can be a powerful tool in detecting malicious JPEG images by helping to identify patterns and clusters in the data that may not be immediately apparent through traditional analysis techniques.

4.2.9 BIAS MITIGATION

Detecting malicious JPEG images requires analyzing the image data and identifying patterns that are indicative of malicious content. However, the image data may be biased towards certain types of images, making it difficult to accurately detect malicious content.

To mitigate bias in malicious JPEG image detection, there are a few strategies that can be employed:

Diversify the training data: To ensure that the model is not biased towards a specific type of image, it is important to have a diverse set of training data that includes images from different sources and with different characteristics.

Augment the training data: Data augmentation techniques, such as rotating, flipping, or cropping images, can be used to increase the diversity of the training data and reduce bias.

Use multiple models: Using multiple models with different architectures and training data can help reduce bias and increase the accuracy of malicious JPEG image detection.

Evaluate performance on diverse test sets: To ensure that the model is not biased towards a specific type of image, it is important to evaluate its performance on a diverse set of test data that includes images with different characteristics.

Regularly update the model: The distribution of JPEG images may change over time, so it is important to regularly update the model with new data to ensure that it remains effective in detecting malicious content.

4.2.10 SPYDER

A highly advanced Data Science Python platform. Created with Python for Python, this IDE boasts some immensely robust toolsets. With an editor, IPython

Console, Variable Explorer, Advanced Plot Functionality, a built-in debugger, and object doc helper tools, the Spyder IDE is a promising choice for a large amount of Data Science tasks.

Link your datasets and data to a single graph or figure with Glueviz. This Python library allows you to view data visualizations by combining datasets and using the logical links within them.

If Data Mining is your goal, then Orange 3 has you covered. Orange 3 is a toolset built for Data Mining. They offer great GUI, extendable functionality with add-ons, data management, and interactive data visualizations. Also, loved by the teaching and student communities for its immersive visualizations, figures, and graphs.

If you are new to Data Science and want to get the complete experience with Python or if you are an experienced seasoned Data Scientist that is looking for more functionality and efficiency, I really recommend you look at this amazing distribution. It makes package management and deployment quick and easy. Packed with tools, IDEs, packages & libraries Anaconda is a truly authentic decision for Data Science. Because popularity for Anaconda seems to be expanding in many industries and areas that are new to having the availability to such advanced capabilities.

CHAPTER 5

SYSTEM TESTING

5.1 TESTING AND METHODOLOGIES

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

5.2 UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Unit testing is a software testing technique used to verify the correct behavior of individual code units or functions in a program. In the context of machine learning

algorithms for malicious image detection, unit testing can be used to verify the correctness of the implementation of individual components of the algorithm, such as the data pre-processing steps, feature extraction, and the training and testing of the model.

Unit testing in machine learning algorithms typically involves testing each component or function of the algorithm in isolation, using a small set of test cases designed to cover a variety of possible scenarios. The goal is to ensure that each component or function performs as intended and produces the expected output for a given input.

For example, in the context of malicious image detection using a convolutional neural network (CNN), unit testing may involve testing the correctness of the forward propagation step of the network, the accuracy of the loss function, and the performance of the optimizer during the training process.

Unit testing can help identify errors or bugs in the code early in the development cycle, reducing the risk of introducing errors in the final product. It also facilitates code maintenance and refactoring by providing a suite of automated tests that can be run to ensure that changes to the codebase do not break existing functionality.

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

5.3 INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

5.4 ACCEPTANCE TESTING

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance testing is a software testing technique used to determine whether a machine learning algorithm for malicious image detection meets the requirements and expectations of its end-users.

In the context of machine learning, acceptance testing for malicious image detection involves testing the model's ability to correctly classify malicious and benign images using a set of predefined acceptance criteria. These criteria may include metrics such as accuracy, precision, recall, and AUC, which are commonly used to evaluate the performance of machine learning models.

The acceptance testing process typically involves creating a set of test cases that cover a range of scenarios that the model is expected to encounter in real-world usage. These test cases may include images that the model has not seen during training and images with varying levels of complexity, resolution, and quality.

The acceptance testing phase is critical in ensuring that the machine learning algorithm for malicious image detection is effective and reliable in real-world scenarios. It provides a way to verify that the algorithm meets the requirements

and expectations of its users and stakeholders, and that it can be deployed with confidence.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

5.5 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy.

In machine learning algorithms for malicious image detection, input design is the process of selecting and preparing the input data that will be used to train and evaluate the model. Input design is a critical step in the development of a machine learning algorithm, as the quality and relevance of the input data can significantly impact the performance and accuracy of the model.

The input design process typically involves the following steps:

Data Collection: Collecting a diverse and representative set of images that cover a range of malware types, image formats, resolutions, and quality levels. This

may involve sourcing images from public datasets or creating a custom dataset based on the specific requirements of the project.

Data Preprocessing: Preprocessing the data to ensure that it is in a suitable format for use in the machine learning algorithm. This may involve resizing, cropping, or converting the images to a standardized format or resolution.

Feature Extraction: Extracting relevant features from the images that can be used as input to the machine learning algorithm. This may involve using techniques such as edge detection, texture analysis, or color histograms.

Data Augmentation: Augmenting the dataset to increase its size and diversity. This may involve techniques such as flipping, rotating, or adding noise to the images.

Data Splitting: Splitting the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set is used to tune the hyperparameters of the model, and the testing set is used to evaluate the performance of the final model.

The input design process is crucial in ensuring that the machine learning algorithm for malicious image detection is trained and evaluated using high-quality and relevant data, which is essential for achieving accurate and reliable results.

OBJECTIVES:

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

5.6 OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

In machine learning algorithms for malicious image detection, output design is the process of defining the format and structure of the output that the model will produce after analyzing an input image. The output design is an essential aspect of the development of a machine learning algorithm, as the output needs to be interpretable and actionable by the end-user

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and

effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2.Select methods for presenting information.

3.Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

The output design process is crucial in ensuring that the output produced by the machine learning algorithm for malicious image detection is interpretable, actionable, and integrated with other systems or applications, which is essential for its practical application in real-world scenarios.

CHAPTER 6

RESULT AND DISCUSSION

6.1 DISCUSSION :

Malicious JPEG image detection is a challenging task that requires advanced image processing and machine learning techniques. In this task, we want to classify whether an image is benign or malicious (contains a virus or other harmful code) based on its content.

The first step in this task is to collect a large dataset of both benign and malicious JPEG images. This dataset can be obtained from various sources, such as online image repositories or by generating malicious JPEG images using malware analysis tools.

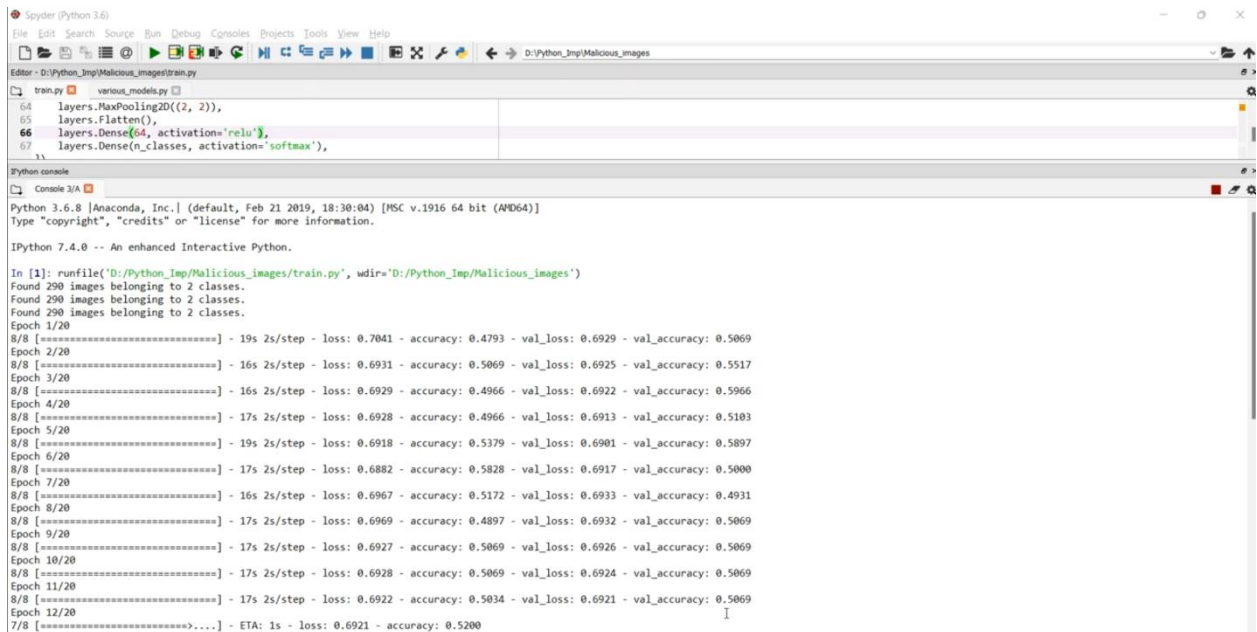
Once we have a dataset, we can use various image processing techniques to extract features from the images, such as color histograms, texture features, and edge detection. These features can then be used to train a machine learning model, such as a support vector machine (SVM) or a convolutional neural network (CNN), to classify the images as benign or malicious.

In Spyder, we can use various Python libraries such as NumPy, OpenCV, and scikit-learn to implement this image processing and machine learning pipeline. Spyder provides a convenient integrated development environment (IDE) to write and run Python code for this task.

6.2 RESULTS :

The below images are shown results

Fig: 2



```
train.py
64 layers.MaxPooling2D((2, 2)),
65 layers.Flatten(),
66 layers.Dense(64, activation='relu'),
67 layers.Dense(n_classes, activation='softmax'),
68
```

```
Python console
Python 3.6.8 [Anaconda, Inc.] (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Python_Imp/Malicious_images/train.py', wdir='D:/Python_Imp/Malicious_images')
Found 290 images belonging to 2 classes.
Found 290 images belonging to 2 classes.
Found 290 images belonging to 2 classes.
Epoch 1/20
8/8 [=====] - 19s 2s/step - loss: 0.7041 - accuracy: 0.4793 - val_loss: 0.6929 - val_accuracy: 0.5069
Epoch 2/20
8/8 [=====] - 16s 2s/step - loss: 0.6931 - accuracy: 0.5069 - val_loss: 0.6925 - val_accuracy: 0.5517
Epoch 3/20
8/8 [=====] - 16s 2s/step - loss: 0.6929 - accuracy: 0.4966 - val_loss: 0.6922 - val_accuracy: 0.5966
Epoch 4/20
8/8 [=====] - 17s 2s/step - loss: 0.6928 - accuracy: 0.4966 - val_loss: 0.6913 - val_accuracy: 0.5103
Epoch 5/20
8/8 [=====] - 19s 2s/step - loss: 0.6918 - accuracy: 0.5379 - val_loss: 0.6901 - val_accuracy: 0.5897
Epoch 6/20
8/8 [=====] - 17s 2s/step - loss: 0.6882 - accuracy: 0.5828 - val_loss: 0.6917 - val_accuracy: 0.5000
Epoch 7/20
8/8 [=====] - 16s 2s/step - loss: 0.6967 - accuracy: 0.5172 - val_loss: 0.6933 - val_accuracy: 0.4931
Epoch 8/20
8/8 [=====] - 17s 2s/step - loss: 0.6969 - accuracy: 0.4897 - val_loss: 0.6932 - val_accuracy: 0.5069
Epoch 9/20
8/8 [=====] - 17s 2s/step - loss: 0.6927 - accuracy: 0.5069 - val_loss: 0.6926 - val_accuracy: 0.5069
Epoch 10/20
8/8 [=====] - 17s 2s/step - loss: 0.6928 - accuracy: 0.5069 - val_loss: 0.6924 - val_accuracy: 0.5069
Epoch 11/20
8/8 [=====] - 17s 2s/step - loss: 0.6922 - accuracy: 0.5034 - val_loss: 0.6921 - val_accuracy: 0.5069
Epoch 12/20
7/8 [=====] - ETA: 1s - loss: 0.6921 - accuracy: 0.5200
```

```
In [2]: runfile('D:/Python_Imp/Malicious_images/variou_models.py', wdir='D:/Python_Imp/Malicious_images')
Reloaded modules: tmp39zyv16, tmpy0hhined

Naive Bayes
The TPR is : 0.6509
The FPR is : 0.0000
The IDR is : 0.6509
The AUC value is : 0.8254

Decision Tree
C:\Program Files\Anaconda3\lib\site-packages\sklearn\metrics\_ranking.py:808: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
C:\Program Files\Anaconda3\lib\site-packages\sklearn\metrics\_ranking.py:808: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
The TPR is : 0.9527
The FPR is : 0.0006
The IDR is : 0.9521
The AUC value is : 0.9760

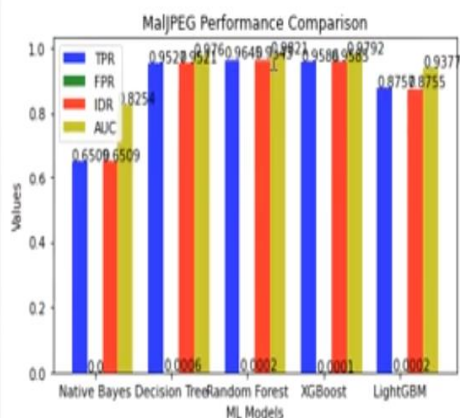
Random Forest
C:\Program Files\Anaconda3\lib\site-packages\sklearn\metrics\_ranking.py:808: UndefinedMetricWarning: No positive samples in y_true, true positive value should be meaningless
  UndefinedMetricWarning)
The TPR is : 0.9645
The FPR is : 0.0002
The IDR is : 0.9643
The AUC value is : 0.9821

XGBoost
C:\Program Files\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do
the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[23:13:10] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release.1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was
changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

```

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
0.9796088719294062
The TPR is : 0.8757
The FPR is : 0.0002
The IDR is : 0.8755
The AUC value is : 0.9377

```



CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

In this paper, we present MalJPEG, a machine learning based solution for efficient detection of unknown malicious JPEG images. To the best of our knowledge, we are the first to present a machine learning-based solution tailored specifically for the detection of malicious JPEG images. MalJPEG extracts 10 simple but discriminative features from the JPEG file structure and leverages them with a machine learning classifier, in order to discriminate between benign and malicious JPEG images. MalJPEG features are extracted based on the structure of the JPEG image. MalJPEG features were defined based on an understanding of how attackers use JPEG images in order to launch attacks and how it affects the JPEG file structure in comparison to regular benign JPEG images. The features are simple and relatively easy to extract statically (without actually viewing the image) when parsing the JPEG image file.

7.2 FUTURE ENHANCEMENT

These detection results are slightly worse than the corresponding results obtained in Experiment 1, yet they still demonstrate that a machine learning-based classifier trained on MalJPEG features can also efficiently detect unknown malicious JPEG images in a real-life scenario in which the classifier is tested on future unknown instances.

In the fourth experiment, we tested MalJPEG on five datasets which represent a real-life scenario in which we trained the machine learning model on prior

instances (chronologically) and tested the model only on unknown future instances.

A2: Source code:

Train.py

```
from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.utils import plot_model

import matplotlib.pyplot as plt

import numpy as np

import tensorflow as tf

from tensorflow.keras import models, layers


IMAGE_SIZE = 256

BATCH_SIZE = 32

CHANNELS = 3

EPOCHS = 20

from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator ( rescale=1./255,

                                     horizontal_flip=True,

                                     zoom_range=0.3,

                                     shear_range = 0.2
```

)

```
train_dataset = train_datagen.flow_from_directory(  
    'D:/project/Code_Malicious_images/dataset2/train',  
    target_size=(IMAGE_SIZE, IMAGE_SIZE),  
    batch_size=40)
```

#validation data

```
validation_dataset = train_datagen.flow_from_directory(  
    'D:/project/Code_Malicious_images/dataset2/test',  
    target_size=(IMAGE_SIZE, IMAGE_SIZE),  
    batch_size=40)
```

#test data

```
test_dataset = train_datagen.flow_from_directory(  
    'D:/project/Code_Malicious_images/dataset2/val',  
    target_size=(IMAGE_SIZE, IMAGE_SIZE),  
    batch_size=1)
```

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
```

```
n_classes = 2
```

```

model = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    layers.Conv2D(32, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

```

```

model.compile(
    loss='categorical_crossentropy',

```

```

optimizer='adam',

metrics=['accuracy']

)

history = model.fit(

    train_dataset,

    steps_per_epoch=len(train_dataset),

    batch_size=32,

    validation_data=validation_dataset,

    validation_steps=len(validation_dataset),

    epochs=20,

)

model.save("mal_img_model.h5")

```

2).various_modules.py

```

import pandas as pd

import numpy as np

from sklearn import utils

from glob import glob

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve

from sklearn.metrics import roc_auc_score

```



```
from sklearn.metrics import plot_roc_curve
```

```
from matplotlib import pyplot as plt
```

```
def perf_measure(y_actual, y_hat):
```

```
    T = 0
```

```
    F = 0
```

```
    TP = 0
```

```
    FP = 0
```

```
    TN = 0
```

```
    FN = 0
```

```
    for i in range(len(y_hat)):
```

```
        if y_actual[i] == y_hat[i]:
```

```
            T+=1
```

```
        if y_actual[i] != y_hat[i]:
```

```
            F+=1
```

```
        if y_actual[i] == 0 and y_hat[i] == 100:
```

```
            FP += 1
```

```
        if y_actual[i] == 100 and y_hat[i] == 100:
```

```
            TP += 1
```

```
        if y_actual[i] == 0 and y_hat[i] == 0:
```

```
TN += 1
```

```
if y_actual[i] == 100 and y_hat[i] == 0:
```

```
    FN += 1
```

```
return (TP, FP, TN, FN)
```

```
def cal_values(TP, FP, TN, FN, y_test, model_output):
```

```
    TPR = TP / (TP + FN)
```

```
    print("The TPR is : %.4f" % TPR)
```

```
    FPR = FP / (FP + TN)
```

```
    print("The FPR is : %.4f" % FPR)
```

```
    IDR = TPR * (1 - FPR)
```

```
    print("The IDR is : %.4f" % IDR)
```

```
    AUC = (roc_auc_score(y_test, model_output))
```

```
    print("The AUC value is : %.4f" % AUC)
```

```
    return (TPR, FPR, IDR, AUC)
```

```

def dec_tree(x_train, y_train, x_test, y_test):

    from sklearn.tree import DecisionTreeClassifier

    dtc = DecisionTreeClassifier()

    dtc.fit(x_train, y_train)

    model_output=dtc.predict(x_test)

    # print(model_output)

    # print(y_test)

    fpr,tpr, _ = roc_curve(y_test, model_output,pos_label=1)

    # plot_roc_curve(dtc,x_test,y_test)

    # #plt.show()

    # plt.savefig('Decision Tree.png')

    TP, FP, TN, FN = perf_measure(y_actual=y_test, y_hat=model_output)

    TPR,FPR, IDR, AUC = cal_values(TP,FP,TN,FN,y_test,model_output)

    return (TPR,FPR, IDR, AUC)

```

```

def native_bayes(x_train, y_train, x_test, y_test):

    from sklearn.naive_bayes import BernoulliNB

    naive_bayes_model= BernoulliNB()

    naive_bayes_model.fit(x_train, y_train)

    model_output=naive_bayes_model.predict(x_test)

    # print(model_output)

    # print(y_test)


    fpr, tpr, _ = roc_curve(y_test, model_output, pos_label=1)

    # plot_roc_curve(naive_bayes_model, x_test, y_test)

    # plt.show()

    # plt.savefig('Native Bayes.png')


    TP, FP, TN, FN = perf_measure(y_actual=y_test, y_hat=model_output)


    TPR, FPR, IDR, AUC = cal_values(TP, FP, TN, FN, y_test, model_output)

```

```
return (TPR,FPR, IDR, AUC)
```

```
def xgboost(x_train, y_train, x_test, y_test):
```

```
    from xgboost import XGBClassifier
```

```
    gbo = XGBClassifier()
```

```
    gbo.fit(x_train,y_train)
```

```
    model_output=gbo.predict(x_test)
```

```
    # print(model_output)
```

```
    # print(y_test)
```

```
    from sklearn.metrics import roc_curve
```

```
    from sklearn.metrics import roc_auc_score
```

```
    from sklearn.metrics import plot_roc_curve
```

```
    from matplotlib import pyplot as plt
```

```
    fpr, tpr, _ = roc_curve(y_test, model_output, pos_label=1)
```

```
    # plot_roc_curve(gbo,x_test,y_test)
```

```
    # #plt.show()
```

```

plt.savefig('XGBoost.png')

TP, FP, TN, FN = perf_measure(y_actual=y_test, y_hat=model_output)

TPR, FPR, IDR, AUC = cal_values(TP, FP, TN, FN, y_test, model_output)

return (TPR, FPR, IDR, AUC)

def random_forest(x_train, y_train, x_test, y_test):

    from sklearn.ensemble import RandomForestClassifier

    rfc = RandomForestClassifier(n_estimators=100)

    rfc.fit(x_train, y_train)

    model_output=rfc.predict(x_test)

    # print(model_output)

    # print(y_test)

    from sklearn.metrics import roc_curve

    from sklearn.metrics import roc_auc_score

```

```

from sklearn.metrics import plot_roc_curve

from matplotlib import pyplot as plt


fpr, tpr, _ = roc_curve(y_test, model_output, pos_label=1)

# plot_roc_curve(rfc, x_test, y_test)

# plt.show()

# plt.savefig('Random Forest.png')


TP, FP, TN, FN = perf_measure(y_actual=y_test, y_hat=model_output)


TPR, FPR, IDR, AUC = cal_values(TP, FP, TN, FN, y_test, model_output)


return (TPR, FPR, IDR, AUC)


def light_gbm(x_train, y_train, x_test, y_test):

    import lightgbm as lgb

    from sklearn.model_selection import GridSearchCV

    from sklearn.metrics import accuracy_score, roc_curve, auc

    import matplotlib.pyplot as plt

```

```
lgb_train = lgb.Dataset(x_train, label=y_train)
```

```
gridParams = {  
    'learning_rate': [0.07],  
    'n_estimators': [8,16],  
    'num_leaves': [20, 24, 27],  
    'boosting_type' : ['gbdt'],  
    'objective' : ['binary'],  
    'random_state' : [501],  
    'colsample_bytree' : [0.64, 0.65],  
    'subsample' : [0.7,0.75],  
    #'reg_alpha' : [1, 1.2],  
    #'reg_lambda' : [ 1.2, 1.4],  
}
```

```
params = {'boosting_type': 'gbdt',  
    'max_depth' : -1,  
    'objective': 'binary',  
    'nthread': 5,  
    'num_leaves': 64,  
    'learning_rate': 0.07,
```



```
'max_bin': 512,  
'subsample_for_bin': 200,  
'subsample': 1,  
'subsample_freq': 1,  
'colsample_bytree': 0.8,  
'reg_alpha': 1.2,  
'reg_lambda': 1.2,  
'min_split_gain': 0.5,  
'min_child_weight': 1,  
'min_child_samples': 5,  
'scale_pos_weight': 1,  
'num_class': 1,  
'metric': 'binary_error'  
}
```

```
mdl = lgb.LGBMClassifier(boosting_type= 'gbdt',  
    objective = 'binary',  
    n_jobs = 5,  
    silent = True,  
    max_depth = params['max_depth'],  
    max_bin = params['max_bin'],
```

```

subsample_for_bin = params['subsample_for_bin'],
subsample = params['subsample'],
subsample_freq = params['subsample_freq'],
min_split_gain = params['min_split_gain'],
min_child_weight = params['min_child_weight'],
min_child_samples = params['min_child_samples'],
scale_pos_weight = params['scale_pos_weight'])

grid = GridSearchCV mdl, gridParams, verbose=2, cv=4, n_jobs=-1)

# Run the grid
grid.fit(x_train, y_train)

# Print the best parameters found
print(grid.best_params_)
print(grid.best_score_)

lgbm = lgb.train(params,
                 lgb_train,
                 280,
                 #early_stopping_rounds= 40,

```

```

        verbose_eval= 4

    )

    predictions_lgbm_prob = lgbm.predict(x_test)

    predictions_lgbm_01 = np.where(predictions_lgbm_prob > 0.5, 1, 0)


    acc_lgbm = accuracy_score(y_test,predictions_lgbm_01)

    print(acc_lgbm)

    for i in range(len(y_test)):

        if y_test[i] == 100:

            y_test[i] = 1


    #plot ROC curve

    #plt.figure()

    #false_positive_rate, recall, thresholds = roc_curve(y_test,
predictions_lgbm_prob)

    #roc_auc = auc(false_positive_rate, recall)

    #plt.plot(false_positive_rate, recall, 'b', label = 'LGBMClassifier(AUC =
%0.3f)' %roc_auc)

    #plt.legend(loc='lower right')

    #plt.xlim([0.0,1.0])

    #plt.ylim([0.0,1.0])

    #plt.ylabel('True Positive Rate')

```

```

plt.xlabel('False Positive Rate')

plt.show()

predictions_lgbm = [ele * 100 for ele in predictions_lgbm_01]

y_test = [ele * 100 for ele in y_test]

TP, FP, TN, FN = perf_measure(y_actual=y_test, y_hat=predictions_lgbm)

TPR,FPR, IDR, AUC = cal_values(TP,FP,TN,FN,y_test,predictions_lgbm)

return (TPR,FPR, IDR, AUC)

def main():

    train = pd.read_csv('./dataset/train.csv',header=None)

    x_train = np.array(train.iloc[:, 0:9])

    y_train = np.array(train.iloc[:,10])

    test = pd.read_csv('./dataset/test.csv',header=None)

    x_test = np.array(test.iloc[:, 0:9])

    y_test = np.array(test.iloc[:,10])

```

```
rows,cols = (5,1)      # change to (5,1) for 5 different models
```

```
data = [[0 for i in range(cols)] for j in range(rows)]
```

```
print("Naive Bayes")
```

```
tpr1,fpr1,idr1,auc1 = native_bayes(x_train, y_train, x_test, y_test)
```

```
data[0].append(tpr1)
```

```
data[1].append(fpr1)
```

```
data[2].append(idr1)
```

```
data[3].append(auc1)
```

```
print("Decision Tree")
```

```
tpr2,fpr2,idr2,auc2 = dec_tree(x_train, y_train, x_test, y_test)
```

```
data[0].append(tpr2)
```

```
data[1].append(fpr2)
```

```
data[2].append(idr2)
```

```
data[3].append(auc2)
```

```
print("Random Forest")
```

```
tpr3,fpr3,idr3,auc3 = random_forest(x_train, y_train, x_test, y_test)
```

```
data[0].append(tpr3)
```

```
data[1].append(fpr3)
```

```
data[2].append(idr3)
```

```
data[3].append(auc3)
```

```
print("XGBoost")
```

```
tpr4 ,fpr4 ,idr4 ,auc4 = xgboost(x_train, y_train, x_test, y_test)
```

```
data[0].append(tpr4)
```

```
data[1].append(fpr4)
```

```
data[2].append(idr4)
```

```
data[3].append(auc4)
```

```
tpr5 ,fpr5 ,idr5 ,auc5 = light_gbm(x_train, y_train, x_test, y_test)
```

```
data[0].append(tpr5)
```

```
data[1].append(fpr5)
```

```
data[2].append(idr5)
```

```
data[3].append(auc5)
```

```
for i in range(5):
```

```

data[i].pop(0)

# create plot

fig, ax = plt.subplots()

index = np.arange(5)    #change to 5 with addition of Light GBM

bar_width = 0.20

opacity = 0.8

# def autolabel(rects):

#     for rect in rects:

#         height = rect.get_height()

#         ax.text(str(height), 1.05*height,

#                 '%d' % int(height),

#                 ha='center', va='bottom')

rects1 = plt.bar(index, data[0], bar_width,

alpha=opacity,

color='b',

label='TPR')

```

```

rects2 = plt.bar(index + bar_width, data[1], bar_width,
alpha=opacity,
color='g',
label='FPR')

rects3 = plt.bar(index + 2*bar_width, data[2], bar_width,
alpha=opacity,
color='r',
label='IDR')

rects4 = plt.bar(index + 3*bar_width, data[3], bar_width,
alpha=opacity,
color='y',
label='AUC')

plt.xlabel('ML Models')
plt.ylabel('Values')
plt.title('MaJPEG Performance Comparison')

plt.xticks(index + bar_width, ('Native Bayes', 'Decision Tree', 'Random Forest',
'XGBoost','LightGBM'))

plt.legend()

```



```
for bar in rects1:

    yval = round(bar.get_height(),4)

    plt.text(bar.get_x(), yval + .005, yval)
```

```
for bar in rects2:

    yval = round(bar.get_height(),4)

    plt.text(bar.get_x(), yval + .005, yval)
```

```
for bar in rects3:

    yval = round(bar.get_height(),4)

    plt.text(bar.get_x(), yval + .005, yval)
```

```
for bar in rects4:

    yval = round(bar.get_height(),4)

    plt.text(bar.get_x(), yval + .005, yval )
```

```
plt.tight_layout()

plt.savefig('5050comparison.png')
```

```
if __name__=='__main__':

    main()
```

A2: SCREENSHOT

Fig: 3

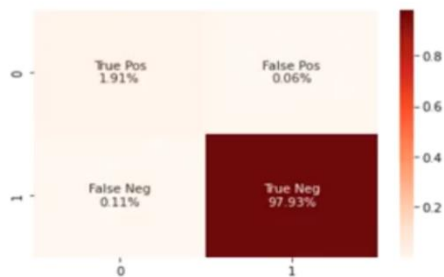
DECISION TREE

```
In [13]: from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
dtc_model_output = dtc.predict(x_test)
models.append(dtc)

dtc_results = performance_measure(y_actual=y_test, y_predicted=dtc_model_output)
confusion_matrix(dtc_results)
model_results["Decision Tree"] = dtc_results

dtc_metrics = cal_values(dtc_results, y_test=y_test, model_output=dtc_model_output)
add_to_lists(dtc_metrics)
```

The TPR is : 0.9467
The FPR is : 0.0006
The IDR is : 0.9462
The AUC value is : 0.9731



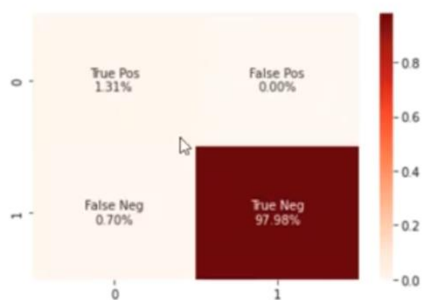
NAIVE BAYES

```
In [20]: from sklearn.naive_bayes import BernoulliNB
nb = BernoulliNB()
nb.fit(x_train, y_train)
nb_model_output = nb.predict(x_test)
models.append(nb)

nb_results = performance_measure(y_actual=y_test, y_predicted=nb_model_output)
model_results["Naive Bayes"] = nb_results
confusion_matrix(nb_results)

nb_metrics = cal_values(nb_results, y_test=y_test, model_output=nb_model_output)
add_to_lists(nb_metrics)
```

The TPR is : 0.6509
The FPR is : 0.0000
The IDR is : 0.6509
The AUC value is : 0.8254



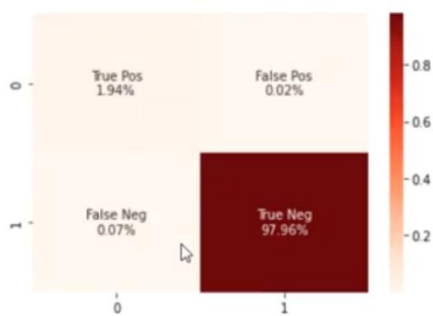
RANDOM FOREST

```
In [25]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
rfc_model_output = rfc.predict(x_test)
models.append(rfc)

rfc_results = performance_measure(y_actual=y_test, y_predicted=rfc_model_output)
confusion_matrix(rfc_results)
model_results["Random Forest"] = rfc_results

rf_metrics = cal_values(rfc_results, y_test=y_test, model_output=rfc_model_output)
add_to_lists(rf_metrics)
```

The TPR is : 0.9645
The FPR is : 0.0002
The IDR is : 0.9643
The AUC value is : 0.9821



XGBOOST

```
In [30]: import xgboost as xgb
from xgboost import XGBClassifier
xgm = XGBClassifier()
xgm.fit(x_train, y_train)
xgm_model_output = xgm.predict(x_test)
models.append(xgm)

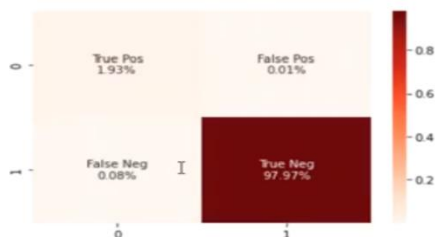
xgm_results = performance_measure(y_actual=y_test, y_predicted=xgm_model_output)
confusion_matrix(xgm_results)
model_results["xgboost"] = xgm_results

xgm_metrics = cal_values(xgm_results, y_test=y_test, model_output=xgm_model_output)
add_to_lists(xgm_metrics)
```

C:\Program Files\Anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
warnings.warn(label_encoder_deprecation_msg, UserWarning)

[22:48:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

The TPR is : 0.9586
The FPR is : 0.0001
The IDR is : 0.9585
The AUC value is : 0.9792



LIGHTGBM

```
In [31]: lgb_train = lgb.Dataset(x_train, label=y_train)
```

```
models.append mdl)
```

```
lgb_results = performance_measure(y_actual=y_test, y_predicted=lgb_model_output)
confusion_matrix(lgb_results)
model_results["lightgbm"] = lgb_results
```

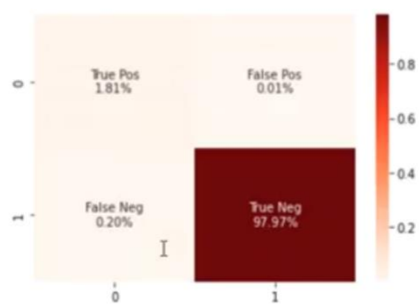
```
lgb_metrics = cal_values(lgb_results, y_test=y_test, model_output=lgb_model_output)
add_to_lists(lgb_metrics)
```

The TPR is : 0.8994

The FPR is : 0.0001

The IDR is : 0.8993

The AUC value is : 0.9496



REFERENCES

- [1] Henry D Samuel; M Santhanam Kumar; R. Aishwarya; G. Mathivanan, “Automation Detection of Malware and Stenographical Content using Machine Learning,” IEEE paper November 2022.
- [2] Pruthvi Priya P M; Hemavathi P “PDF Malware Detection System based on Machine Learning Algorithm” IEEE paper November 2022.
- [3] Radifa Akbar Abhesa, Hendrawan and Setia Juli Irzal Ismail “Classification of Malware Using Machine Learning Based on Image Processing” IEEE paper November 2021.
- [4] D. Nahmias, A. Cohen, N. Nissim, and Y. Elovici, “TrustSign: Trusted malware signature generation in private clouds using deep feature transfer learning,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2019.
- [5] A. Cohen, N. Nissim, and Y. Elovici, “Novel set of general descriptive features for enhanced detection of malicious emails using machine learning methods,” Expert Syst. Appl., vol. 110, pp. 143–169, Nov. 2018.
- [6] N. Nissim, A. Cohen, and Y. Elovici, “ALDOCX: Detection of unknown malicious microsoft office documents using designated active learning methods based on new structural feature extraction methodology,” IEEE Trans. Inf. Forensics Security, vol. 12, no. 3, pp. 631–646, Mar. 2017.
- [7] N. Nissim, A. Cohen, R. Moskovitch, A. Shabtai, M. Edri, O. Bar-Ad, and Y. Elovici, “Keeping pace with the creation of new malicious PDF files using an active-learning based detection framework,” Secur. Inform., vol. 5, p. 1, Dec. 2016.

- [8] N. Nissim, A. Cohen, C. Glezer, and Y. Elovici, “Detection of malicious PDF files and directions for enhancements: A state-of-the-art survey,” *Comput. Secur.*, vol. 48, pp. 246–266, Feb. 2015.
- [9] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, “Novel active learning methods for enhanced PC malware detection in windows OS,” *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5843–5857, Oct. 2014.
- [10] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, “Detecting unknown computer worm activity via support vector machines and active learning,” *Pattern Anal. Appl.*, vol. 15, no. 4, pp. 459–475, Nov. 2012.