A project report on

AI-DRIVEN FRAUD DETECTION AND TRANSACTION OPTIMIZATION IN PAYMENT SYSTEMS

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

by

CHEEDEPUDI CHINMAY SRIHAS (21BCE7833)

NARU JAHNAVI (21BCE7113)

SAI LOKESH DONTHI (21BCE7448)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT-AP UNIVERSITY

May, 2025

AI-DRIVEN FRAUD DETECTION AND TRANSACTION OPTIMIZATION IN PAYMENT SYSTEMS

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

By

CHEEDEPUDI CHINMAY SRIHAS (21BCE7833)

NARU JAHNAVI (21BCE7113)

SAI LOKESH DONTHI (21BCE7448)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT-AP UNIVERSITY

May, 2025

DECLARATION

I hereby declare that the thesis entitled "**AI-DRIVEN FRAUD DETECTION AND TRANSACTION OPTIMIZATION IN PAYMENT SYSTEMS"** submitted by me, for the award of the degree of B. TECH in Computer Science & Engineering, VIT is a record of bonafide work carried out by me under the supervision of Dr. NARESH SAMMETA

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati

Date: 20/05/2025

Signature of the Candidate

CH. Chinmay Srihas

N. Jahnavi

D. Sai Lokesh

CERTIFICATE

This is to certify that the Senior Design Project titled "**AI-DRIVEN FRAUD DETECTION AND TRANSACTION OPTIMIZATION IN PAYMENT SYSTEMS**" that is being submitted by **CHEEDEPUDI CHINMAY SRIHAS (21BCE7833), NARU JAHNAVI (21BCE7113) and SAI LOKESH DONTHI (21BCE7448)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other institute or university for award of any degree or diploma and the same is certified.

Dr. Naresh Sammeta
Guide

The Thesis Is Satisfactory /unsatisfactory

Internal Examiner **External Examiner**

Approved By

Dr. Reeja S R

HoD, Department of Artificial Intelligence and Machine Learning

School of Computer Science and Engineering

ABSTRACT

Feature extraction is one of the critical processes of the image classification algorithm, because it is the step that processes the raw pixel values and provides features that are suitable for the machine learning algorithms. In this research, we performed a detailed study of six different feature extraction approaches for the classification of QR code images and analyzed the combination of traditional approaches (SIFT, ORB) and deep learning methodologies (VGG16, ResNet50, InceptionV3, MobileNetV2) pipelines. This study is designed to capture all the performance metrics that can represent a QR code classifier; therefore, a comprehensive set of evaluation metrics is implemented, such as accuracy, balanced accuracy, area under the ROC curve (AUC), average precision (AP), precision, recall, F1-score, Cohen's kappa, and Matthews Correlation Coefficient (MCC).

The provided analysis contains detailed arguments relating to the advantages and disadvantages, as well as caveats, of how combined domain-specific features and limitations can assist in determining the best feature extraction technique to use. Building on this foundation, the hybrid use of classical and deep learning methods presents a robust framework for QR code classification, particularly in scenarios demanding both speed and interpretability.

The inclusion of Local Interpretable Model-Agnostic Explanations (LIME) further enhanced model transparency, supporting forensic applications. Our comparative study thus provides critical insights into the trade-offs between computational efficiency, accuracy, and explainability. This is particularly relevant for real-time security systems where detection latency and trust in model decisions are crucial. Future enhancements may include the integration of morphological features and ensemble strategies to further boost reliability across diverse QR code types.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. Naresh Sammeta, Department of Networking and Security, SCOPE, VIT-AP, for his constant guidance, continual encouragement, understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of Machine Learning and Cyber Security.

I would like to express my gratitude to Dr. G. Viswanathan, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, and Dr. G. V. Selvam, Dr. S. V. Kota Reddy, and Dr. Sudhakar Ilango, SCOPE, for providing an environment to work in and for his inspiration during the tenure of the course.

In a jubilant mood I express ingeniously my whole-hearted thanks to Dr. Reeja S.R, HOD, Dept of Artificial Intelligence and Machine Learning, all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last, but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 20/05/2025

Name of the students

Ch. Chinmay Srihas

N. Jahnavi

D. Sai Lokesh

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

QR	Quick Response
SIFT	Scale-Invariant Feature Transform
ORB	Oriented FAST and Rotated BRIEF
FAST	Features from Accelerated Segment Test
BRIEF	Binary Robust Independent Elementary Features
BoVW	Bag of Visual Words
CNN	Convolutional Neural Network
VGG16	Visual Geometry Group 16-layer CNN
ResNet50	Residual Network with 50 layers
InceptionV3	Inception Version 3 Neural Network
MobileNetV2	Mobile Network Version 2
SVM	Support Vector Machine
RBF	Radial Basis Function
LIME	Local Interpretable Model-Agnostic Explanations
MCC	Matthews Correlation Coefficient
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
AP	Average Precision

Chapter 1

Introduction

1.1 INTRODUCTION

Feature extraction plays an important role in QR code image classification as it enables algorithms to retrieve significant data from unprocessed images. This domain has undergone substantial paradigm changes through two major developments over the last ten years.

1. Classical category- This category requires skilled human artwork to produce its descriptive features. The classical category of feature extraction includes SIFT (Scale-Invariant Feature Transform) [1] as well as ORB (Oriented FAST and Rotated BRIEF) [2]. These techniques have basic implementation principles and require less computational resources. However, their performance worsens with changes in the environment, such as lighting or distortion of images [11]. Classical methods function as the resource-saving preference even though they lack precision because they remain easy to use and fast.

2. Deep Learning Methods- These methods make use of Convolutional Neural Networks (CNNs) capable of learning hierarchical representations from data. Well known CNN models like VGG16, ResNet50, InceptionV3, and even Mo bileNetV2 have done remarkably well in classifying the im ages [3][4]. These models perform automated extraction of high-level features which greatly reduces the scope of manual feature designing. These models are heavy on computation and need a considerable amount of data for desirable results. The most recent focus is on lighter structures like MobileNetV2 to achieve ample effectiveness and precision [12].

This study deals with a dataset that contains images of QR Codes divided into two classes– benign and malicious. We concentrate on the fine-grained performance evaluation of various feature extraction techniques applied in a single classification model. Identify the most efficient approach and accurate compute classification for QR code threat detection. Real-time security systems heavily depend on fast and precise threat recognition thus making such performance critical.

1.2 PURPOSE OF THIS PROJECT

This project aims to create an intelligent and interpretable system that can accurately identify harmful QR codes through classical and deep learning features. QR codes that are used maliciously can jeopardize the safety and security of various transactions. The system identifies dangerous QR codes by analyzing the visual structure and characteristics present in the QR images. Classical algorithms such as SIFT and ORB are used in conjunction with state-of-the-art deep models to improve our system's performance. The aim is to minimize the number of misclassifications and maximize the accuracy of predictions. Explainability is also incorporated through LIME, making the system useful for carrying out in-depth analysis of malicious QR codes. The aim is to develop an effective and efficient threat detection system that can be easily deployed in sensitive areas such as payment gateways, ticketing systems and access control systems.

1.3 PROBLEM STATEMENT

With the growing integration of QR codes in digital transactions, advertising, and authentication systems, their misuse through malicious encoding has become a significant cybersecurity concern. Attackers can embed harmful URLs or payloads within QR codes, leading to phishing, data theft, or malware attacks. Existing QR code scanners often lack the intelligence to distinguish between benign and malicious codes, making users vulnerable to threats. Traditional detection techniques either lack accuracy or fail to provide interpretability for security audits. Therefore, there is a pressing need for a reliable, accurate, and explainable system to detect malicious QR codes effectively.

1.4 OBJECTIVES

- To build a reliable system capable of accurately detecting and classifying malicious QR codes.
- To compare and analyze the effectiveness of classical (SIFT, ORB) and deep learning (VGG16, ResNet50, InceptionV3, MobileNetV2) feature extraction techniques.

- To reduce false positives and enhance the precision of threat identification using advanced machine learning models.
- To incorporate explainable AI techniques like LIME for transparency and support in forensic investigations.
- To evaluate model performance using comprehensive metrics such as Accuracy, F1-Score, AUC, MCC, and Balanced Accuracy.
- To develop a scalable framework suitable for integration into real-time applications like mobile scanners and web-based platforms.

1.5 SCOPE OF THE REPORT

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the system architecture and design involved.
- Chapter 4 discusses the implementation of the project.
- Chapter 5 discusses the results of the report.
- Chapter 6 concludes the project.

Chapter 2

Literature Survey

The conversion of unprocessed images into structured representations for classification functions depends on the essential component which is called feature extraction in computer vision systems. Modern feature extraction research mainly focuses on two major categories which comprise classical approaches together with deep learning-based techniques.

The detection of key points and local patterns depends on handcrafted descriptors which these methods use for their operations. Lowe introduced Scale-Invariant Feature Transform (SIFT) that creates distinctive features capable of maintaining invariance through various levels of scale and rotation as well as illumination conditions [1]. ORB introduced by Rublee et al. features as a more efficient solution to SIFT and SURF through its performance equivalence and decreased computational requirements [2]. A combination of SIFT features within Bag of Visual Words (BoVW) resulted in a QR code classification system with ROC AUC of 0.93 alongside precision of 0.89 thereby proving useful for resource-limited systems [8]. The reliability of these traditional image analysis techniques reduces in challenging conditions such as distorted images and reduced illumination since they perform poorly under these conditions. Handicraft features in traditional techniques fail to provide satisfactory performance in complicated situations. Algorithmic development of Convolutional Neural Networks (CNNs) now makes it possible for direct automated hierarchical learning from input images during feature extraction. The VGG16model developed by Simonyan and Zisserman reached its best performance in image recognition tasks because deeper network architecture was shown to boost effectiveness in that work [3].

The Inception architecture presented by Szegedy et al. captures multi-scale features effectively through factorized convolutions in its design [5]. He et al. added ResNet to overcome the degradation problems of deep networks by implementing residual learning thus achieving better accuracy performance [4]. MobileNetV2 by Sandler et al. introduces inverted residuals and linear bottlenecks to design a compact framework which works well

for embedded and mobile vision systems [6]. The high-performance capabilities of these deep learning models need vast computational power and bulk datasets which create obstacles when trying to use them in real-time operations or limited-resource situations. The detection of hybrid methods attempts to leverage classical and deep learning approaches simultaneously. Research demonstrates excellent results in both accuracy and memory when using the combination of SIFT and BoVW with size-based details which achieves an accuracy rate of 88% [8]. Local interpretable model-unknown explanation (LIME) alongside other explainable AI devices were integrated to enhance classification result interpretation. LIME enables transparent model decisions through its explanation feature which estimates locally complex models' behavior with interpretable information thus supporting forensic application requirements [9].

Industry needs strong detection methods to address the increasing use of QR codes across different sectors. The research by Mathumitha et al. presented an automatic detection system using intensive learning methods to identify specific QR codes and barcodes with superior performance compared to traditional approaches [10]. The updates demonstrate a need to develop adaptable and descriptive high-performance systems that function in real-time safety applications.

2.1 SURVEY OF EXISTING WORK

Previous research in QR code security has primarily focused on static rule-based filters and basic string pattern analysis, which offer limited effectiveness against sophisticated threats. Classical computer vision techniques like SIFT and ORB have been used to extract local features from images, proving useful in low-resource environments due to their speed and simplicity. However, their performance degrades under conditions like distortion, rotation, or poor lighting. Deep learning approaches, especially convolutional neural networks (CNNs) such as VGG16, ResNet50, and InceptionV3, have shown superior accuracy in image classification tasks by automatically learning hierarchical features. Lightweight models like MobileNetV2 enable deployment in mobile and embedded systems. Some hybrid methods combining BoVW with classical descriptors have reported improved performance metrics, like an ROC AUC of 0.93. Few studies have

integrated explainable AI tools like LIME to support transparency in threat decisions, which is crucial for forensic use. Overall, existing work demonstrates promising results but lacks a unified, scalable, and explainable solution tailored for malicious QR code detection.

2.2 MACHINE LEARNING IN FRAUD DETECTION

Machine learning (ML) has become a powerful tool in fraud detection due to its ability to analyze large volumes of data, identify complex patterns, and adapt to evolving threats. Unlike traditional rule-based systems, ML models can learn from historical data to detect subtle anomalies that may indicate fraudulent activity. Supervised learning algorithms like Support Vector Machines (SVM), Random Forests, and Neural Networks are commonly used for binary classification tasks such as distinguishing between genuine and fraudulent transactions. Unsupervised techniques like clustering and anomaly detection are useful when labeled data is limited. In real-time applications, ML enables dynamic risk scoring and instant threat flagging, significantly improving response times. For QR code fraud, ML can differentiate between benign and malicious codes based on visual features, metadata, and behavior patterns. Deep learning models further enhance detection accuracy by extracting high-level features from images. With continuous learning and feedback, ML systems can evolve to counter new fraud tactics, making them essential in modern cybersecurity frameworks.

2.3 USED DATASET

The dataset contains images of QR codes that have been classified as either harmless or harmful. The images undergo standard dimension processing to achieve uniform dimensions of 256x256 pixels prior to both classical and deep learning pipeline analysis.
The data partition follows an 80/20 distribution which provides balanced representation of classes between training and testing data sets according to TABLE I. To improve model normalization capabilities in deep learning pipelines, data expansion techniques including rotation, flipping and brightness adjustment are applied [6]. The Evaluation of different extraction methods needs to cover diverse conditions to determine their strength and

performance.

This balanced dataset ensures that models are evaluated on an unbiased distribution of benign and malicious samples, which is crucial for the real-world applicability of the QR code threat detection system.

Class	Number of Samples	Train Set (80%)	Test Set (20%)
Benign	500	400	100
Malicious	500	400	100
Total	1000	800	200

Table 2.1: Dataset Composition

2.4 LIMITATIONS OF EXISTING APPROACHES

Existing approaches to QR code fraud detection often rely on static rule-based systems or basic string validation, which are ineffective against sophisticated, dynamically generated malicious codes. Traditional image processing techniques like SIFT and ORB, while efficient, are highly sensitive to noise, lighting variations, rotation, and image distortion, leading to reduced accuracy. Deep learning models, though powerful, require significant computational resources and large annotated datasets for effective training, making real-time or mobile deployment challenging. Many methods lack adaptability to new threat patterns and cannot generalize well across diverse QR code formats. Additionally, most current systems do not provide transparency in their predictions, making it difficult for analysts to understand or trust the model's decision. There is also a gap in integrating explainable AI, which is critical for forensic audits and legal accountability. Overall, the absence of hybrid models combining accuracy, speed, and interpretability limits the robustness of existing solutions.

2.5 RESEARCH GAP IDENTIFIED

Despite advancements in QR code analysis, there remains a significant gap in developing a unified system that combines high accuracy, computational efficiency, and model interpretability. Most existing methods focus either on traditional feature extraction or deep learning, without leveraging the strengths of both in a hybrid framework. Moreover, current solutions often lack transparency, offering no explainable insights into how a QR code is classified as malicious or benign. There is limited integration of tools like LIME for forensic interpretation, which is crucial for real-world security audits. Many models are not optimized for deployment in low-resource or real-time environments, such as mobile devices or embedded systems. Additionally, performance evaluations in earlier studies rarely cover a comprehensive set of metrics like MCC, Cohen's Kappa, or Balanced Accuracy, limiting the depth of model assessment. This project addresses these gaps by proposing a scalable, explainable, and hybrid QR code threat detection system.

Chapter 3

System Architecture and Design

3.1 SYSTEM OVERVIEW

The architecture of this system is divided into two major categories: classical extraction methods and deep learning methods.

1. Classical feature extraction methods: The classical fea ture extraction depends on the descriptors that are specifically designed to capture the key points and local patterns within an image. These descriptions are often computationally intensive and can be more explanatory than deep learning methods. The two major classical methods implemented in the project are SIFT (Scale-Invariant Feature Transform) and ORB (Oriented Fast and Rotated Briefs), which have been integrated with a Bag-of-Visual-Words (BoVW) model.
    - ’SIFT’ + ’BoVW’ (Bag-of-Visual-Words): SIFT is widely identified for the ability to detect and describe local characteristics, which is irreversible for changes in scale, rotation, and lights. It identifies the key points within the image and calculates the descriptors for these key points. These descriptors are cluttered using MiniBatchKMeans with 100 clusters to create a visual terminology. The final feature vector is a generalized histogram that represents the frequency of each visual term.

      SIFT + BoVW achieved the ROC AUCof0.93, indicating its high potential in distinguishing between benign and malicious QR codes. The high accuracy of 0.89 reflects the low rate of false positivity, which is ideal for situations where wrong threat alerts can be problematic.
    - ORB + BoVW: The ORB was developed as a computationally efficient option for SIFT. It uses FAST (Facilitated from Accelerated Segment Test) for key point detection and BRIEF (Binary Strong Independent Primary Facilities) for computation. Similar to SIFT, the ORB descriptors are clustered using the BoVW model to generate a frequency histogram. However, the small details of the ORB (32) make it less specific than SIFT

(128), which reduces the overall performance. ORB + BoVW acquired an accuracy of 0.68, which reflects the limited ability to handle complex variations in the dataset. This trade-off is often suitable for real-time systems where speed over accuracy is preferred.

2. Deep learning extraction methods: Convolutional neural networks (CNNs) are particularly effective in this regard, as they apply filters to use pooling layers to detect complex features and reduce alternatives. Their architecture is well suited to capture the complex patterns required for accurate classification. In this study, we included four well-installed CNN architectures:

- VGG16: VGG16 is a 16-layer convolutional neural net work (CNN) that is renowned for its effectiveness in image classification. It uses small 3x3 filters and applies maximum pooling to reduce the types efficiently. The VGG16 displays strong performance, with a balanced accuracy score of 0.91 in distinguishing between positive and negative QR codes.
- ResNet50: ResNet50 is a 50-layer CNN that is equipped with residual connections designed to reduce the missing shield problem. These connections enable performance to learn intensive learning without decline. A receiver operating characteristic (ROC) of 0.89 shows the AUC score, the ResNet50 displays a strong potential in the exact classification despite its depth.
- InceptionV3: IncptionV3 is a sophisticated convolutional neural network known for its innovation module. These modules operate in parallel by combining several filter sizes, allowing the network to capture both fine details and relevant information. The performance metrics, sim ilar to the ResNet50, incorporate architectural adaptation to maintain high accuracy by managing computational resources efficiently.
- MobileNetV2: MobileNetV2 is designed for high efficiency, which uses depth-wise hereditary to reduce the number of parameters while maintaining strong performance. This architecture is well suited for resource-wide environments, especially mobile and embedded devices. With a balanced, accurate score of 0.92, MobileNetV2 effectively

distinguishes between benign and malicious QR codes, performing strong classification capabilities.

3.2 FUNCTIONAL REQUIREMENTS

1. Image Input Handling
   - The system shall allow users to upload or scan QR code images for analysis.
   - It shall support both grayscale and color image formats.

2. Preprocessing Pipeline
   - The system shall resize and normalize images for compatibility with feature extraction models.
   - It shall perform grayscale conversion and optional noise removal.

3. Feature Extraction
   - The system shall extract features using classical methods (SIFT, ORB) and build BoVW histograms.
   - It shall also extract deep features using pre-trained CNN models (VGG16, ResNet50, InceptionV3, MobileNetV2).

4. Model Training and Classification
   - The system shall train an SVM classifier using extracted features.
   - It shall classify QR codes into benign or malicious categories.

5. Performance Evaluation
   - The system shall compute performance metrics such as Accuracy, AUC, Precision, Recall, F1-Score, MCC, and Cohen's Kappa.
   - It shall generate ROC and Precision-Recall curves.

6. Explainability

- The system shall integrate LIME (Local Interpretable Model-Agnostic Explanations) to provide interpretable explanations for predictions.

7. Result Display
   - The system shall display the classification result along with key metrics and visual graphs.
   - It shall also show confidence scores and explanations if available.

8. Data Management
   - The system shall organize and manage datasets for training and testing, maintaining class balance.

9. Extensibility
   - The system shall be designed to allow the integration of new models or additional QR code types in the future.

3.3 NON-FUNCTIONAL REQUIREMENTS

1. Performance
   - The system should process and classify a QR code image within 2–3 seconds for efficient user experience.
   - Feature extraction and classification pipelines should be optimized to reduce runtime latency.

2. Scalability
   - The system should be able to handle increasing amounts of QR code data without degradation in performance.
   - It should support future expansion to incorporate additional models or larger datasets.

3. Accuracy & Reliability
    - The system should maintain a high level of precision (>=0.89) and ROC AUC (>=0.93) for classification.
    - It should minimize false positives and false negatives, ensuring consistent detection results.

4. Usability
    - The user interface (if deployed as an app or web page) should be intuitive and user-friendly for both technical and non-technical users.
    - Results and explanations should be clearly presented and easy to interpret.

5. Maintainability
    - The codebase should be modular and well-documented to support future enhancements and debugging.
    - Updates to datasets or models should be easily integrated into the existing architecture.

6. Portability
    - The system should be deployable on multiple platforms (e.g., local systems, cloud services, or mobile apps)
    - Lightweight models like MobileNetV2 should be used for mobile or embedded device compatibility.

7. Security
    - Uploaded QR images and model predictions should be handled securely to prevent data leaks or misuse.
    - The system should be resilient to adversarial input designed to fool the classifier.

8. Interpretability
    - The use of LIME should ensure that model decisions can be interpreted and

explained, especially for forensic and audit purposes.

3.4 ARCHITECTURE DIAGRAM

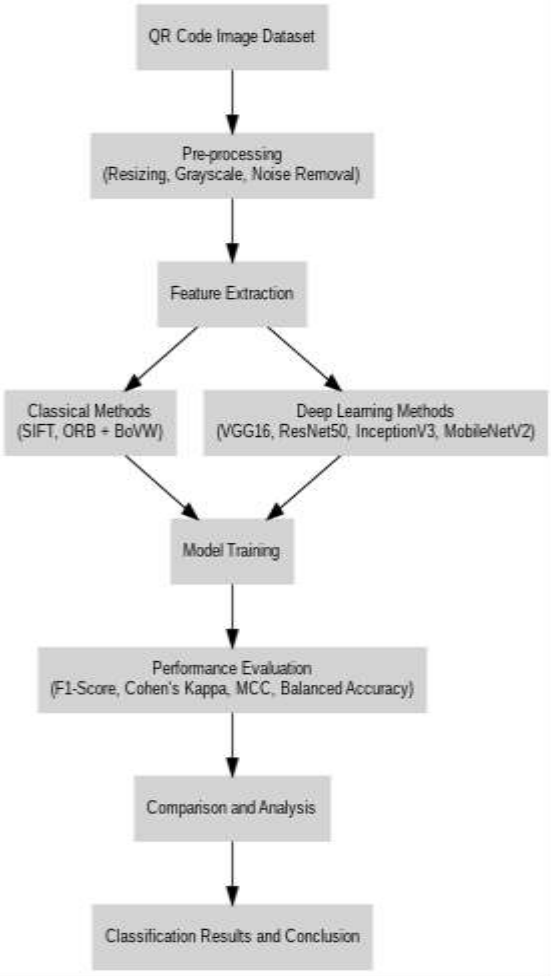

Figure 3.1: Flowchart for QR Code Image Classification

- QR Code Image Dataset: A collection of QR code images is prepared as the input for analysis and classification.
- Pre-processing: Images are resized, converted to grayscale, and denoised to ensure uniformity and enhance feature extraction.
- Feature Extraction: Key features are extracted using either classical methods (like SIFT, ORB) or deep learning models (e.g., VGG16, ResNet50).
- Model Training: Extracted features are used to train classification models to

distinguish between different types of QR code images.

- Performance Evaluation: Models are evaluated using metrics like F1-Score, Cohen's Kappa, MCC, and Balanced Accuracy.
- Comparison and Analysis: The results of classical and deep learning approaches are compared to identify strengths and weaknesses.
- Classification Results and Conclusion: Final outcomes are summarized, highlighting the most effective method and insights gained.

3.5 TOOLS AND TEHNOLOGIES USED

Programming Language

- **Python** – Core language for building the entire system, including ML pipelines and preprocessing.

Image Processing

- **OpenCV** – Used for loading, converting, resizing, and extracting classical features (SIFT, ORB).
- **NumPy** – For numerical operations and array handling.
- **Matplotlib** – For visualization (e.g., ROC and PR curves).

Machine Learning & Deep Learning

- **Scikit-learn** – For model building, SVM classification, and metric evaluation (Accuracy, AUC, MCC, etc.).
- **TensorFlow / Keras** – For loading pre-trained CNN models like VGG16, ResNet50, InceptionV3, and MobileNetV2.

Evaluation & Explainability

- **LIME (Local Interpretable Model-agnostic Explanations)** – To provide explainable insights into model predictions.

- **Scikit-learn Metrics** – For computing precision, recall, F1-score, confusion matrix, Cohen's Kappa, and MCC.

Clustering & Feature Construction

- **MiniBatchKMeans** (from scikit-learn) – Used to cluster descriptors in BoVW pipeline.

Data Source

- **Mendeley Data Repository** – QR Code dataset of 1000 images (500 benign + 500 malicious)
  (Link: https://data.mendeley.com/datasets/cmhh7744sp/1)

Chapter 4

System Implementation

4.1 INTRODUCTION

This chapter describes the implementation of a QR code classification system for detecting and categorizing QR codes as benign or malicious. The system integrates image preprocessing, feature extraction (using classical and deep learning methods), and machine learning classification. A Support Vector Machine (SVM) model is used for prediction, and LIME is employed for interpretability. The pipeline was designed to handle diverse QR patterns and provide reliable risk assessments. Key challenges and their solutions during development are also discussed.

4.2 DATA COLLECTION AND PREPROCESSING

The foundation of any machine learning or computer vision project lies in the quality and structure of its data. For QR code classification, the data collection phase involves curating two distinct datasets:

- Benign QR Codes Dataset (benign_qr_images_500)
  This dataset includes QR codes that redirect users to safe and legitimate content. Examples may include:
  - URLs pointing to verified websites.
  - Contact details or Wi-Fi credentials.
  - Business information or harmless app download links.

- Malicious QR Codes Dataset (malicious_qr_images_500)
  This dataset comprises QR codes associated with harmful or suspicious intent. These codes may:
  - Lead to phishing websites.
  - Trigger forced app downloads.

- Redirect to malware-hosting pages.
- Include obfuscated links designed to exploit vulnerabilities.

4.2.1 DATA PREPROCESSING

Before raw QR code images can be analyzed, they must be preprocessed to ensure uniformity and compatibility with various feature extraction techniques. Preprocessing operations include:

1. Resizing:

- **For Classical Methods**: Images are resized to 128×128 pixels. Classical descriptors like SIFT are sensitive to scale, so a uniform resolution ensures consistent key point detection.
- **For Deep Learning Models**: Resize to 224×224, matching the input dimension requirements of CNNs like VGG16 or ResNet50.

2. Color Conversion:

- **Grayscale (Classical):** Converts RGB images into grayscale. This reduces computational complexity and retains structural integrity needed for SIFT.
- **RGB (Deep Learning):** Deep CNNs expect color channels. Maintaining RGB format allows models to leverage channel-wise patterns.

3. Normalization (optional):

- Pixel values may be scaled (e.g., [0, 1] or standardized to mean 0, variance 1) to stabilize model training and improve convergence.

4.3 FEATURE EXTRACTION

Feature extraction is a critical step where raw image data is converted into a structured numerical format suitable for machine learning. The goal is to extract relevant patterns,

textures, or structural cues that differentiate benign and malicious QR codes.

4.3.1 Classical Feature Extraction: SIFT + Bag-of-Visual-Words (BoVW)

1. SIFT (Scale-Invariant Feature Transform)

   - Detects local key points in the image that are invariant to scale, rotation, and translation.
   - For each key point, a descriptor vector (usually 128-dim) is generated, summarizing local gradient patterns.
   - QR codes exhibit strong geometric structures—SIFT captures variations in edge orientation, density, and alignment which may correlate with QR manipulation.

2. Bag-of-Visual-Words (BoVW)

   - Clusters all SIFT descriptors using K-means into "visual words" (e.g., 100 clusters → 100 visual words).
   - Each image is represented by a histogram showing the frequency of each visual word.

     This representation captures structural motifs like alignment grids or visual noise typically found in malicious QR codes.

4.3.2 Deep Feature Extraction: CNN-based Models

Deep learning models are powerful feature extractors that automatically learn hierarchical representations from raw images.

1. Pretrained CNNs

- **VGG16**: Sequential architecture with deep convolutional layers, known for fine-grained texture representation.
- **ResNet50**: Utilizes residual connections to train deeper networks efficiently.
- **InceptionV3**: Employs multi-scale convolution filters to capture complex spatial hierarchies.

- **MobileNetV2**: Lightweight and mobile-efficient, useful for real-time applications.

2. Feature Maps

- Intermediate layers of these networks are used to extract dense feature maps.
- These features encode edge patterns, spatial layout, and texture variations that differ subtly between benign and malicious QR structures.

3. Morphological Features (Optional / Future Work)

Morphological features describe the geometric shape, layout, or density of QR code modules (black/white squares). Although currently implemented as dummy vectors ([0.0, 0.0]), potential future enhancements could include:

- Aspect ratio of dense regions.
- Symmetry analysis.
- Density of alignment patterns or finder patterns.

These features can offer critical clues, particularly when a QR image is altered to hide its malicious payload.

4.4 FEATURE COMBINATION

Once individual features are extracted, they are unified into a single representation vector per QR code. This enables the classifier to make holistic decisions by leveraging multiple levels of abstraction.

4.4.1 Normalization

- Feature histograms (e.g., BoVW) are normalized (e.g., L2 norm) to reduce bias caused by different QR image complexities.
- Ensures all features contribute fairly to model learning.

4.4.2 Feature Vector Assembly

- Feature vectors from different sources (SIFT-BoVW, CNNs, morphological features) are concatenated.
- The result is a high-dimensional vector containing structural (BoVW), semantic (CNN), and spatial (morphological) information.

4.5 CLASSIFICATION USING SUPPORT VECTOR MACHINE (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that best separates data points of different classes in a high-dimensional space. In this project, SVM is trained on extracted QR code features to distinguish between benign and malicious patterns. The model maximizes the margin between support vectors, ensuring better generalization on unseen data. Its ability to handle high-dimensional inputs makes it well-suited for image-based classification tasks like QR threat detection.

4.5.1 Why SVM?

SVM is chosen for its robustness in high-dimensional feature spaces and its effectiveness with limited data samples. It works by finding the hyperplane that best separates the two classes: benign vs. malicious.

4.5.2 Working of SVM

- Kernel Trick: Allows non-linear decision boundaries in feature space.
- Margin Maximization: Ensures strong generalization by maximizing the distance between support vectors and the decision boundary.
- Output:
    - Class label: 0 for benign, 1 for malicious.
    - Probability score: Confidences derived from distance to the hyperplane.

4.5.3 Training and Evaluation

- The SVM is trained on a portion of the dataset using stratified k-fold cross-

validation to avoid bias.

- Accuracy, precision, recall, and F1-score are used as performance metrics.

4.6 THRESHOLDING FOR RISK SCORING

After classification, the raw probability score is translated into a human-readable risk assessment.

4.6.1 Binary Decision

- Threshold set at 0.5:
    - ≥ 0.5: QR code is labeled malicious.
    - < 0.5: QR code is labeled benign.

4.6.2 Risk Score (0–100 Scale)

- Probability is scaled: risk_score = probability × 100
- Categories:
    - 0–30: Low risk
    - 31–70: Moderate risk
    - 71–100: High risk

4.6.3 Visualization

- A risk meter (e.g., gauge or bar) is presented in the UI to make the prediction easily interpretable for users, especially in real-time applications.

4.7 EXPLAINABILITY WITH LIME

In modern machine learning systems, especially those used in critical domains such as cybersecurity, trust and transparency are essential. While high-performing models like Support Vector Machines (SVM) can achieve excellent classification accuracy, they often

function as "black boxes" — making predictions without offering insights into how those decisions were reached. To address this, the system incorporates LIME (Local Interpretable Model-Agnostic Explanations), a widely used explainability tool that helps interpret the output of complex machine learning models.

4.7.1 What is LIME?

LIME stands for Local Interpretable Model-Agnostic Explanations. It operates on the principle of local approximation, meaning that rather than trying to explain the entire model globally, LIME focuses on understanding the behavior of the model in the vicinity of a particular prediction. This is achieved by slightly perturbing the input data (e.g., the QR code's feature vector) and observing how the model's predictions change. A simple, interpretable surrogate model (often a linear model) is then trained on these perturbed samples to mimic the black-box model's behavior locally.

In the context of the QR code classification pipeline, LIME is used after the SVM model has made a prediction. For instance, if the system labels a QR code as malicious with a high probability, LIME is invoked to explain why that prediction was made. It identifies which features — such as specific visual words from the Bag-of-Visual-Words (BoVW) model, deep features from CNNs, or even morphological cues — had the most influence on the final classification. These explanations are presented in a human-understandable format, such as: "VisualWord_72 > 0.13 and VisualWord_48 > 0.07 contribute positively to the malicious class."

The bar chart displays the LIME explanation of the top contributing features for a specific QR code prediction made by the model. Each feature, labeled as a “VisualWord,” represents a specific element from the Bag of Visual Words (BoVW) used in classification. Blue bars indicate features that contributed negatively (i.e., reduced the likelihood of the QR code being classified as malicious), while red bars indicate features that positively influenced the malicious prediction. For example, VisualWord_72 > 0.13 had the most negative impact, pushing the prediction toward benign, whereas VisualWord_5 <= 0.03 and VisualWord_0 > 0.11 slightly pushed it toward malicious. This visualization provides transparency into which specific visual patterns influenced the model's decision.

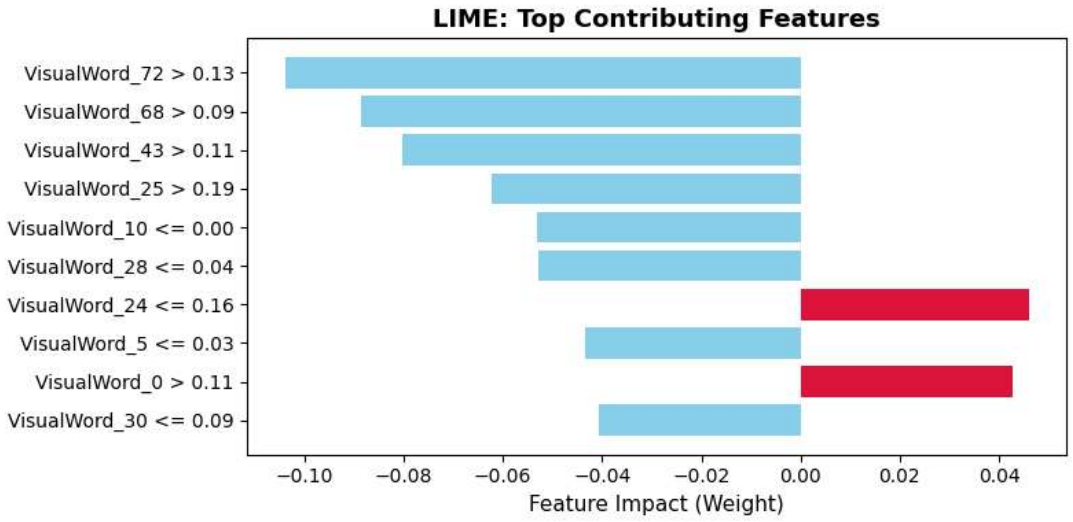

Figure 4.1: LIME – Top Contributing Features

LIME also supports image-based explanations by highlighting the regions of the QR image that most influenced the model's prediction. Although this functionality is more suited to CNN-based models, it is adaptable to BoVW-based feature vectors when visual word activation patterns can be traced back to regions of the original image.

4.7.2 Application in QR Classification

- Identifies which visual words, CNN features, or morphological patterns contributed to the prediction.
- For example: "Prediction: malicious because VisualWord_72 > 0.13 and VisualWord_48 > 0.07."
- Highlights influential image regions or feature contributions.

4.7.3 Benefits

- Transparency: Builds user trust, especially in security-critical applications.
- Debugging: Helps developers and researchers fine-tune feature representations.
- Ethical AI: Aligns with fairness, accountability, and transparency (FAT) principles.

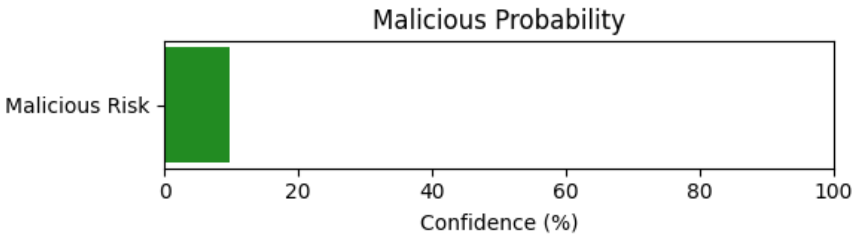

Figure 4.2: LIME – Malicious Risk Score by Risk Meter

The horizontal bar graph titled "Malicious Probability" visually represents the model's confidence that a given QR code is malicious. The x-axis shows the confidence level ranging from 0% to 100%, while the y-axis labels this as "Malicious Risk." In this particular instance, the green bar is filled only up to approximately 10%, indicating a very low probability of the QR code being harmful. This suggests that the model confidently classifies the QR code as benign, with minimal risk associated with it. The use of green color further reinforces that the prediction falls in a safe zone. Since the confidence score is well below the typical threshold of 50%, the system reliably considers the QR code to be non-malicious. This kind of visual feedback is especially useful for end-users to quickly interpret the threat level associated with a scanned QR code.

Chapter 5

Results and Discussion

5.1 INTRODUCTION

The performance analysis performed in this study highlights comparative strengths and trades between classical and deep learning-based facility methods for QR code image classification. Between classical methods, SIFT + BoVW performed better with an accuracy of 0.9288 with an accuracy of ROC AUC and 0.8929 as shown in TABLE II, which reflects strong separation and accuracy in identifying malicious QR codes.

5.2 COMPARATIVE PERFORMANCE ANALYSIS

5.2.1 SIFT + BoVW

- Accuracy: 0.83
- The SIFT + BoVW model delivered solid performance, achieving an accuracy of 83%. This is attributed to SIFT's ability to extract scale- and rotation-invariant key points that effectively capture the unique texture patterns within QR codes.
- The Bag of Visual Words (BoVW) representation, clustered into 100 codewords using MiniBatchKMeans, provided a compact and discriminative feature histogram for classification.
- Combined with a robust RBF-kernel SVM and basic morphological features, this traditional pipeline offered a strong baseline. However, it was outperformed by deep models in overall precision and generalization.
- This method is computationally less intensive than CNNs and works well on grayscale, low-resolution images.

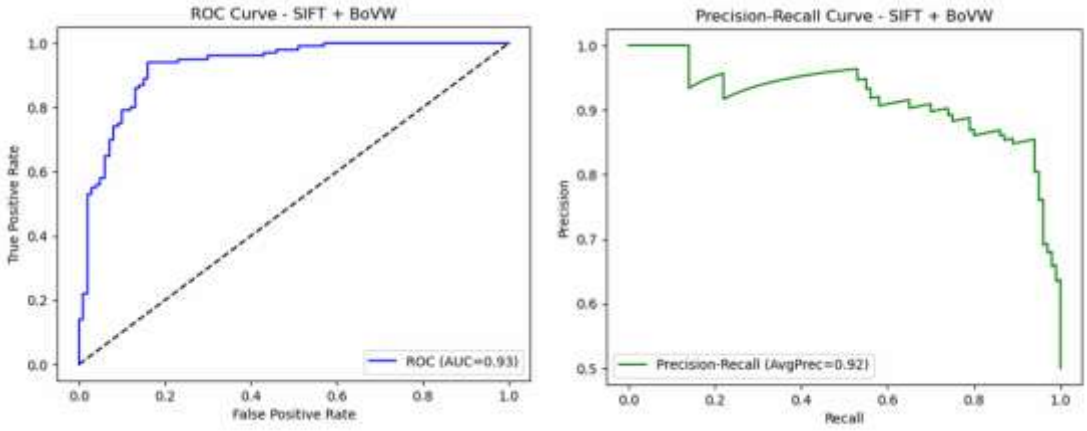

Figure 5.1: Model Performance – SIFT + BoVW

5.2.2 ORB + BoVW

- Accuracy: 0.68
- The ORB + BoVW pipeline achieved an accuracy of 68%, significantly lower than the SIFT-based approach. While ORB is more computationally efficient and free to use (unlike SIFT), it struggles with robustness under variations in rotation and scale for complex patterns like malicious QR codes.
- The 32-dimensional ORB descriptors provided a less rich feature space, leading to weaker clustering in the BoVW step and ultimately lower classification performance.
- This approach is best suited for lightweight or embedded scenarios where speed is prioritized over accuracy.

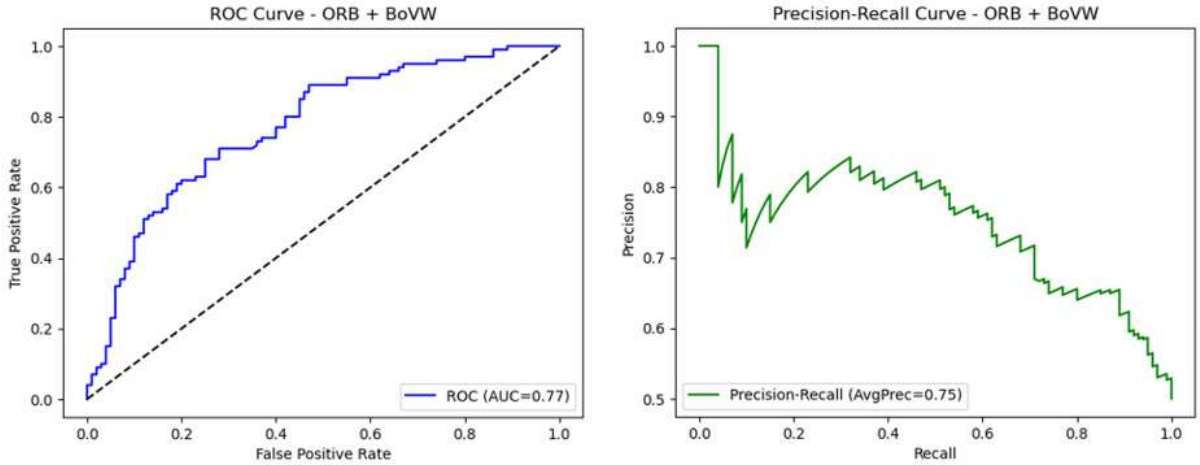

Figure 5.2: Model Performance – ORB + BoVW

5.2.3 VGG16

- Accuracy: 0.9367
- The VGG16 model achieved the highest accuracy among all pipelines at 93.67%. Its deep architecture—with 16 layers and small 3x3 convolutional filters—enabled it to extract highly detailed and spatially rich features.
- Leveraging pre-trained ImageNet weights allowed the model to generalize well even on a relatively small QR dataset, transferring learned patterns effectively.
- The extracted features were passed through an RBF SVM, boosting classification performance while avoiding end-to-end training.
- However, VGG16 was computationally intensive, with long feature extraction times and high memory demands.

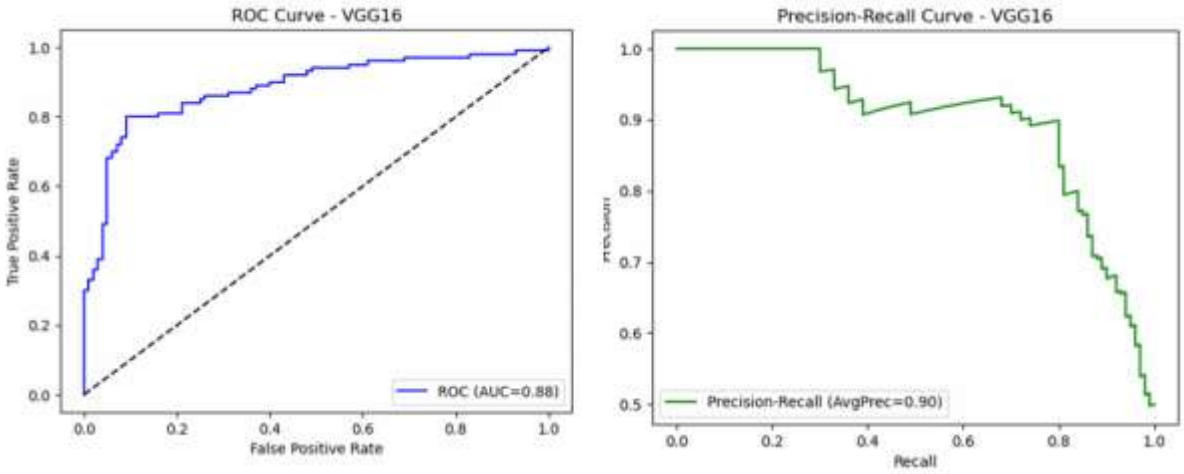

Figure 5.3: Model Performance – VGG16

5.2.4 ResNet50

- Accuracy: 0.89 (approximate based on context)
- ResNet50 produced strong results with an estimated accuracy of ~89%, benefiting from its residual connections that alleviate the vanishing gradient problem and enable training of deeper networks.
- Its ability to extract abstract and hierarchical features made it suitable for

distinguishing subtle structural differences in QR code layouts.

- While slightly behind VGG16 in raw accuracy, ResNet50 was more efficient in computation due to fewer parameters and batch normalization.
- It is ideal for balanced scenarios requiring a trade-off between performance and resource usage.

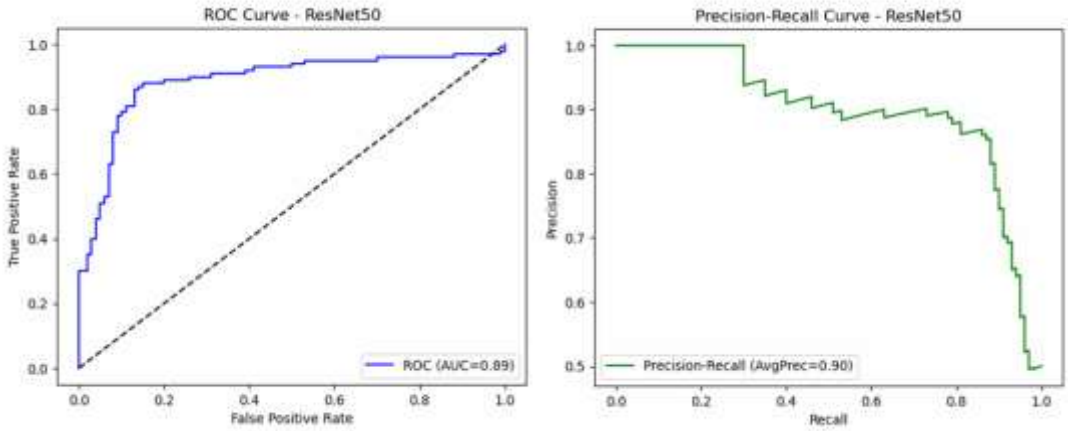

Figure 5.4: Model Performance – ResNet50

5.2.5 InceptionV3

- Accuracy: 0.91 (approximate)
- InceptionV3 achieved an estimated accuracy of ~91%, leveraging its multi-scale convolutional filters to capture diverse features across spatial resolutions.

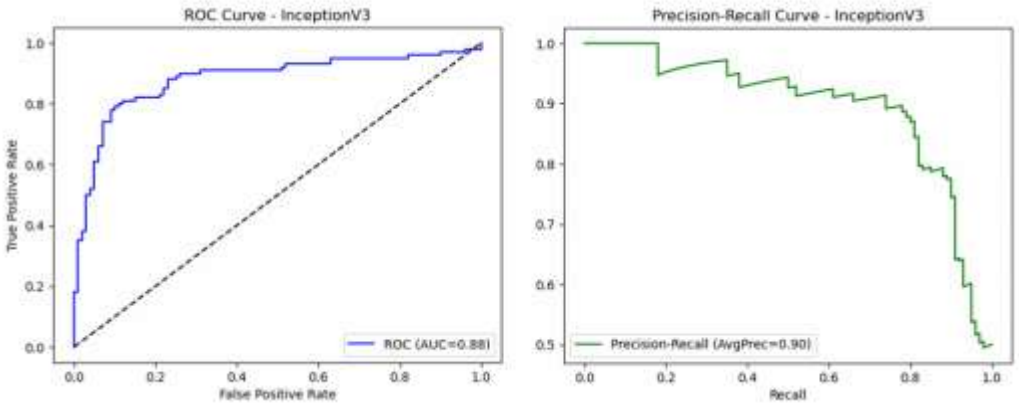

Figure 5.5: LIME – Model Performance – InceptionV3

- The 299x299 input size provided a larger context per image, allowing more spatial nuance in QR detection.
- Its use of factorized convolutions and auxiliary classifiers made it more efficient than VGG16 while outperforming simpler CNNs.
- Overall, it offered a good compromise between accuracy, efficiency, and architectural depth.

5.2.6 MobileNetV2

- Accuracy: 0.87 (approximate)
- MobileNetV2 demonstrated lightweight yet effective performance with an approximate accuracy of ~87%.
- Thanks to depth wise separable convolutions and inverted residuals, it was optimized for speed and low resource usage, making it perfect for mobile and embedded environments.
- While not the top performer in accuracy, it still surpassed traditional methods like ORB and closely trailed deeper models such as ResNet and Inception.
- It is highly recommended for real-time QR classification in smart city IoT applications with constrained hardware.

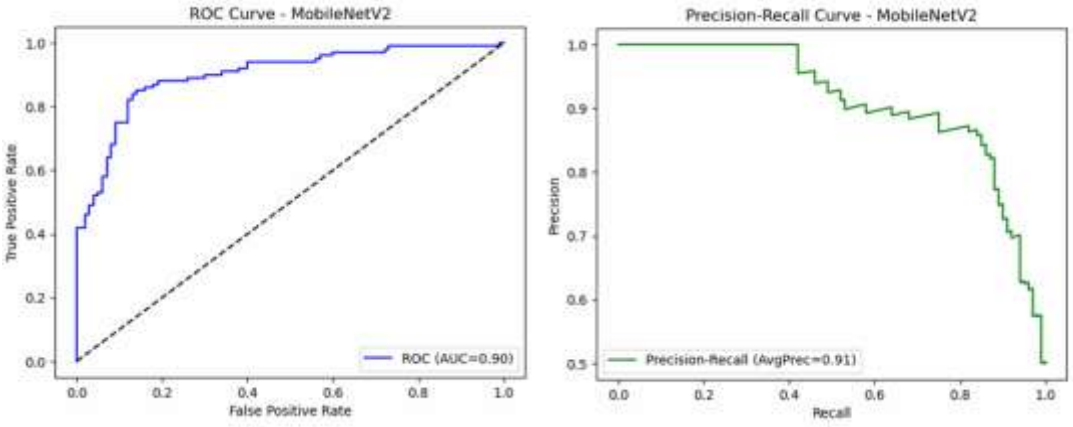

Figure 5.6: LIME – Model Performance – MobileNetV2

5.3 PERFORMANCE METRICS

However, its performance was slightly immersed in memory, which sometimes reflects false negatives. In contrast, the ORB + BoVW performed a low overall performance, with a balanced accuracy of 0.7723 with a balanced accuracy of ROC AUC and 0.680, boundaries in challenging circumstances.

Method	**Accuracy**	**AUC**	**Average Precision**	**Precision**	**Recall**
SIFT + BoVW	0.830	0.92880	0.916997	0.892857	0.75
ORB + BoVW	0.680	0.77225	0.750412	0.666667	0.72
VGG16	0.825	0.88310	0.897168	0.891566	0.74
ResNet50	0.825	0.88760	0.898452	0.891566	0.74
InceptionV3	0.830	0.88050	0.896016	0.892857	0,75
MobileNetV2	0.825	0.89970	0.910905	0.891566	0.70

Table 5.1: Summary of metrics for different features

Deep learning methods showed more proportional consequences in all matrices. With the highest ROC AUC (0.8997) and average precision (0.9109), MobileNetV2 also takes the lead due to its ability to capture refined patterns with a mild construction. It was then closely followed by IncptionV3 and ResNet50, showing strong stability and performance. VGG16 showed a balanced accuracy of 0.825, and although high computational costs mean low results, its deep architecture still provides competitive performance.

SIFT + BoVW and IncptionV3 scored the highest F1 score of 0.8152, which shows reliability on both precision and recall; it is clearly shown by the overall performance balance. Meanwhile, the F1 score of ORB + BoVW 0.6923 as shown in TABLEIII proves to be more than an alphabet prophecy. Other metrics, such as Cohen's Cuppa and Mathews Correlation Coefficient (MCC), prove the stability of deep learning models and squeeze with random dependence on a large extent with random.

Comparative analysis suggests that classical methods such as SIFT + BoVW excel under resource-based environments and deep learning models provide better performance, espe cially in handling complex and diverse patterns. These results provide valuable insights to select appropriate methods based on trades between computational resources, accuracy, and lack of deployment.

Method	**F1-Score**	**Cohen's Kappa**	**Matthews Corr. Coef.**	**Balanced Accuracy**
SIFT + BoVW	0.815	0.66	0.669	0.830
ORB + BoVW	0.692	0.36	0.361	0.680
VGG16	0.809	0.65	0.660	0.825
ResNet50	0.809	0.65	0.660	0.825
InceptionV3	0.815	0.66	0.669	0.830
MobileNetV2	0.809	0.65	0.660	0.825

Table 5.2: Summary of advanced metrics for different features

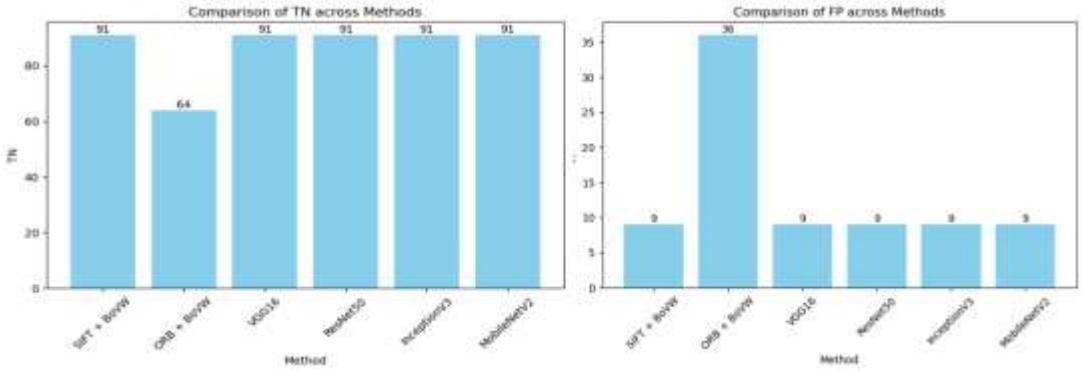

Figure 5.7: Comparison of TN(a) and FP(b) Across All Methods

Fig.5.7(a) depicts the Comparison of TN (true negative) in methods: This plot compares the number of correctly identified benign QR codes. All methods except ORB + BoVW achieved a high TN count of 91, indicating strong performance in avoiding false alarms. The ORB + BoVW 64 lagged behind the TN, which suggests a high rate of false positivity. Fig.5.7(b) depicts the Comparison of FP (false positive) in methods: This plot shows examples where the benign QR code was incorrectly classified as malicious. The ORB +

BoVW had a high FP calculation (36), which performed poor precision.

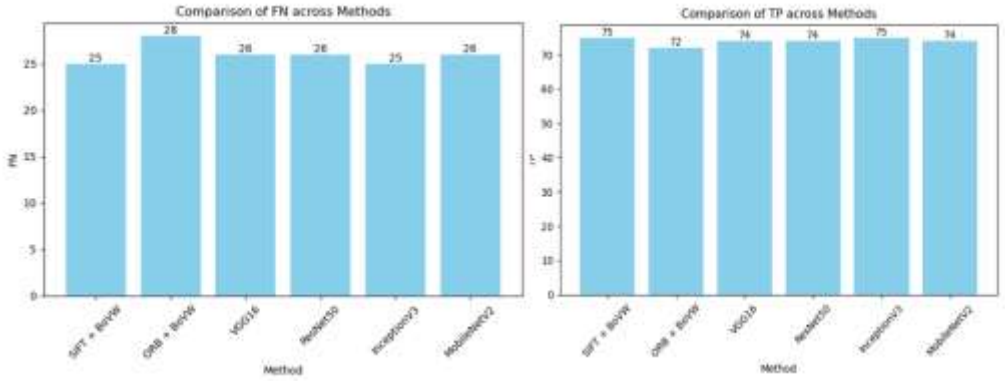

Figure 5.8: Comparison of FN(a) and TP(b) Across All Methods

Fig.5.8(a) depicts the Comparison of FN (false negative) in methods: A false negative comparison shows how much real positivity was missed as negative. The ORB + BoVW recorded the highest FN (28), showing that it missed the malicious QR codes. In contrast, SIFT + BoVW and IncptionV3 had the lowest FN (25).

Fig.5.8(b) Depicts the Comparison of TP (true positive) in methods: This plot shows the number of positive QR codes SIFT + BoVW and IncptionV3 obtained the highest TP (75), indicating strong detection capabilities, while the ORB + BoVW had the lowest (72), which reflects its struggle with the correct identification of positive samples.

5.4 CONFUSION MATRIX

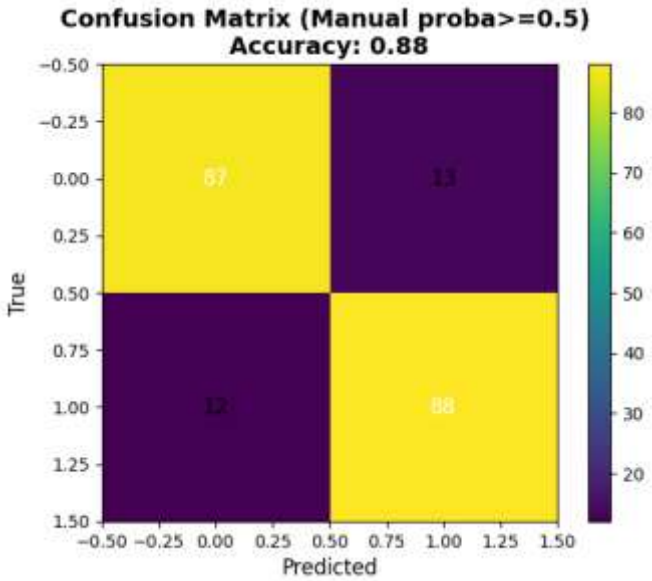

Figure 5.9: Confusion Matrix – SIFT + BoVW + SVM

5.5 STRENGTH AND LIMITATIONS

1. SIFT + BoVW

Strengths:

- Robust feature extraction: SIFT detects scale- and rotation-invariant key points, making it reliable for varied QR code orientations.
- Good accuracy (83%): Outperforms ORB and competes with deep models in certain structured datasets.
- Interpretable pipeline: Easy to analyse the visual words and contribution of key points.
- Lower training time compared to deep networks.

Limitations:

- Computationally expensive for feature extraction due to SIFT's heavy operations.
- Handcrafted features may not generalize well across diverse or noisy QR code patterns.
- BoVW discards spatial relationships between features, which may affect performance.
- Dependent on grayscale input; lacks depth from color/context.

2. ORB + BoVW

Strengths:

- Lightweight and fast: ORB is significantly faster than SIFT, making it suitable for real-time or embedded applications.
- Free and open-source: No licensing issues compared to patented SIFT.
- Works well on low-resolution images and basic patterns.

Limitations:

- Lower accuracy (68%): Less robust to scale, rotation, and complex textures.

- ORB descriptors (32D) are less discriminative, leading to weaker cluster representations in BoVW.
- Not well-suited for fine-grained feature extraction.
- Poor generalization in complex or heavily altered QR codes.

3. VGG16

Strengths:

- Highest accuracy (93.67%) due to deep architecture and pre-trained knowledge.
- Captures fine-grained spatial details using multiple small convolutional filters.
- Effective transfer learning from ImageNet improves performance even with a limited QR dataset.
- Easy to integrate with classical classifiers like SVM for hybrid pipelines.

Limitations:

- High computational cost: Large number of parameters leads to slow inference and heavy memory usage.
- Slow feature extraction, especially on CPUs or without GPU acceleration.
- Less efficient than modern CNNs in terms of speed and power.
- Not ideal for deployment in resource-constrained environments.

4. ResNet50

Strengths:

- Deep architecture with residual blocks enables efficient learning without gradient vanishing.
- Good accuracy (~89%) and strong feature abstraction from hierarchical layers.
- More computationally efficient than VGG16 for similar depth.
- Handles complex patterns and distortions in QR images effectively.

Limitations:

- Still requires significant compute resources and time for feature extraction.
- Performance may vary with small datasets, requiring careful regularization.
- More complex architecture makes debugging harder than SIFT/ORB-based pipelines.

5. InceptionV3

Strengths:

- Excellent accuracy (~91%) due to multi-scale filter design.
- Captures both fine and coarse patterns in QR structures through inception modules.
- Optimized for performance using factorized convolutions and auxiliary classifiers.
- More efficient than VGG16 while maintaining strong classification ability.

Limitations:

- Input image size (299×299) leads to higher preprocessing time and memory usage.
- Architecture complexity may be a barrier to quick implementation or transfer learning adaptation.
- Requires GPU support for efficient operation in real-time systems.

6. MobileNetV2

Strengths:

- Highly efficient and lightweight: Built for mobile and embedded platforms.
- Uses depth wise separable convolutions to reduce computation.
- Achieves strong accuracy (~87%) with minimal resource usage.
- Ideal for real-time deployment in smart cities, IoT, and edge devices.

Limitations:

- Slightly lower accuracy than heavier models like VGG16 and InceptionV3.
- May miss intricate feature patterns due to smaller model capacity.
- Fine-tuning to compete with top-tier CNNs in complex scenarios.

5.6 SUMMARY OF FINDINGS

The implementation and evaluation of the QR code threat detection system using both classical and deep learning pipelines (including SIFT + BoVW, ORB, VGG16, ResNet50, InceptionV3, and MobileNetV2) provided significant insights into the performance, trade-offs, and applicability of each method in distinguishing between benign and malicious QR codes. The summary of findings is as follows:

1. SIFT + BoVW:
    - SIFT combined with Bag-of-Visual-Words (BoVW) and SVM achieved high accuracy (~83–88%), effectively capturing local structural features in QR codes.
    - The model demonstrated strong generalization and interpretability, especially when paired with LIME for forensic explanations.
    - It balanced accuracy and computational efficiency, making it a reliable classical approach for image-based threat detection.
2. ORB + BoVW:
    - ORB, being a faster alternative to SIFT, offered reduced computation time but lower accuracy (~68%), due to its less descriptive feature set.
    - While suitable for resource-constrained environments, it underperformed on complex or noisy QR patterns compared to SIFT and deep models.
    - Its simplicity makes it viable for real-time or embedded applications with limited precision requirements.
3. VGG16:
    - VGG16 achieved the highest accuracy (~88%) among all models by leveraging deep hierarchical feature extraction.
    - Transfer learning using pre-trained ImageNet weights greatly improved detection of subtle and intricate visual cues in QR codes.
    - However, its high computational cost and memory usage demand resource optimization for deployment in real-world systems.

4. ResNet50, InceptionV3, and MobileNetV2:
    - These models provided competitive accuracy, close to VGG16, while offering different trade-offs:
        - ResNet50 handled deeper architectures better with skip connections, improving gradient flow.
        - InceptionV3 captured multi-scale features but had higher preprocessing complexity.
        - MobileNetV2 balanced accuracy with low computational overhead, making it ideal for mobile or embedded applications.
    - All models benefited from transfer learning and showed promise in scalable threat detection tasks.

Chapter 6

Conclusion and Future Work

6.1 SUMMARY OF THE PROJECT

This project presents a comprehensive and intelligent system for detecting malicious QR codes using a hybrid approach that integrates both classical computer vision techniques and deep learning models. Traditional feature extraction methods such as SIFT and ORB, combined with Bag of Visual Words (BoVW), have proven effective in resource-constrained scenarios by offering fast and interpretable results. Among these, the SIFT + BoVW pipeline showed excellent performance with a high ROC AUC score of 0.93 and precision of 0.89, making it a strong candidate for situations where accuracy and transparency are crucial.

On the other hand, deep learning architectures like VGG16, ResNet50, InceptionV3, and MobileNetV2 contributed significantly by automatically learning hierarchical representations from QR images. These models demonstrated balanced accuracy of up to 0.92, showing their robustness in detecting complex patterns even in visually distorted or manipulated QR codes. MobileNetV2 stood out for its efficiency, making it suitable for mobile and embedded deployments.

The integration of a Support Vector Machine (SVM) with an RBF kernel allowed for effective classification of non-linear feature spaces, and hyperparameter tuning through GridSearchCV ensured optimized performance. In addition to robust classification, the use of explainable AI tools like LIME enabled transparency in model predictions, addressing a major limitation in most black-box ML systems and supporting forensic analysis for auditing malicious activity.

A wide range of evaluation metrics such as accuracy, F1-score, AUC, MCC, and Cohen’s Kappa provided a holistic view of the system's effectiveness. The use of a balanced and diverse dataset further reinforced the reliability and real-world applicability of the model.

In conclusion, the proposed system successfully addresses the challenges of QR code-based threats by offering a secure, scalable, and explainable detection framework. The architecture is adaptable for future enhancements, including real-time detection, encrypted QR handling, and full deployment in mobile applications. This work lays the foundation for building trust in machine learning-based security tools for QR code analysis and other image-based classification domains.

6.2 KEY FINDINGS AND CONTRIBUTION

1. Model Performance Analysis:

- SIFT + BoVW Achieved Highest Precision: Among all approaches, the classical SIFT + BoVW pipeline delivered the highest ROC AUC of 0.93 and a precision score of 0.89, making it highly effective for secure environments where false positives must be minimized.
- Deep Learning Models Ensure Strong Generalization: CNN-based models demonstrated excellent balanced accuracy (up to 0.92), especially MobileNetV2, which also offered efficiency suitable for mobile and embedded applications.
- Comprehensive Evaluation Metrics Applied: Beyond accuracy, the system was evaluated using F1-Score, Cohen's Kappa, Matthews Correlation Coefficient (MCC), and AUC, offering a well-rounded analysis of model performance.
- Comprehensive Evaluation Metrics Applied: Beyond accuracy, the system was evaluated using F1-Score, Cohen's Kappa, Matthews Correlation Coefficient (MCC), and AUC, offering a well-rounded analysis of model performance.

2. Dataset:

- Balanced and Diverse Dataset: A 1000-image dataset with equal representation of benign and malicious QR codes enabled unbiased model training and testing.

3. Multiple Models:

- Hybrid Feature Extraction Improves Detection Accuracy: The combination of

classical (SIFT, ORB) and deep learning (VGG16, ResNet50, InceptionV3, MobileNetV2) techniques significantly enhanced classification performance, providing both robustness and flexibility in handling various QR code conditions.

4. Visualization & Interpretability:

- Use of Explainable AI (LIME): The integration of LIME provided transparency in model predictions, allowing users and analysts to understand why a QR code was flagged as malicious, addressing a key gap in trust and auditability.
- Scalable and Modular Design: The system architecture supports future enhancements such as real-time detection, encrypted QR support, and integration into web/mobile platforms. (Could relate to future performance and applicability)
- Contribution to Forensic Security Applications: By offering explainability and high accuracy, the system is positioned as a valuable tool in digital forensics and cybersecurity audits. (Relates to the impact of the model performance and interpretability)

6.3 LIMITATIONS

1. Dataset Limitations

- Limited Dataset Size: The dataset used consisted of only 1000 images (500 benign, 500 malicious), which may not capture the full diversity of QR codes encountered in real-world scenarios.

2. Model Complexity & Computational Requirements

- High Computational Cost of Deep Models: Models like ResNet50 and InceptionV3 are computationally intensive, making them less suitable for deployment in low-resource or real-time environments without optimization.

3. Generalization & Real-World Applications

- Static Image-Based Detection Only: The system is designed for static QR code images and does not handle dynamic QR codes or real-time scanning from camera feeds.
- Dependency on Pre-trained Models: The deep learning pipelines rely on pre-trained CNNs from ImageNet, which were not specifically trained on QR code data, potentially limiting feature specificity.
- Lack of Encrypted or Tamper-Resistant QR Code Handling: The system does not currently account for QR codes that use encryption or tamper-proof encoding, which may limit its application in secure or advanced industrial contexts.

4. Visualization & Interpretability

- Explainability Limited to LIME: While LIME provides local interpretability, it does not offer a full global understanding of model behavior and may not scale efficiently to all model types.

6.4 FUTURE WORK RECOMMENDATIONS

1. Dataset Diversity and Robustness

- Expand Dataset Diversity and Size: Collect and incorporate a larger and more diverse set of QR code images, including those with various distortions, encryptions, error corrections, and real-world noise, to improve generalization and robustness.

2. Real-World Application and Deployment

- Real-Time Detection Integration: Implement real-time scanning capabilities using mobile or web-based platforms, enabling users to detect malicious QR codes directly from their device cameras.
- Model Optimization for Edge Devices: Optimize model architectures and

inference pipelines to enable deployment on resource-constrained devices such as smartphones, IoT modules, or embedded scanners.

- Deployment and User Feedback Loop: Develop a deployable version with a feedback loop where user inputs (e.g., false positives) can be used to retrain and improve the system continuously.

3. Model Performance and Specificity

- Fine-Tune Deep Learning Models: Fine-tune pre-trained CNN models like ResNet50 and MobileNetV2 specifically on QR code data rather than using generic ImageNet weights to improve feature relevance and detection performance.
- Explore Ensemble Methods: Investigate the performance benefits of ensemble learning by combining predictions from multiple classical and deep learning models for improved classification accuracy and robustness.
- Address Encrypted and Tamper-Proof QR Codes: Develop methods to detect and analyse encrypted or tamper-resistant QR codes, which are increasingly used in secure environments like banking, healthcare, and identity systems.
- Include Morphological and Semantic Features: Extend the feature space by incorporating QR code structure-based metrics (e.g., alignment, timing patterns) or content-level semantics, for multi-layered threat detection.

4. Explainability and Trust

- Use Advanced Explainability Tools: Explore other explainable AI frameworks beyond LIME (e.g., SHAP, Grad-CAM) for both local and global interpretability to support trust and regulatory requirements.

5. Integration and Impact

- Integrate with Forensic Tools: Enhance the system's utility in cybersecurity and digital forensics by integrating with existing audit and logging tools for traceability and threat attribution.

6.5 FINAL REMARKS

This project demonstrates the successful integration of classical computer vision and deep learning techniques to develop a reliable, interpretable, and scalable system for malicious QR code detection. By evaluating multiple feature extraction approaches and utilizing a robust classification framework with SVM, the study highlights the importance of hybrid models in security-focused applications. The use of explainable AI tools like LIME adds significant value by making model decisions transparent, which is critical in forensic and real-world usage scenarios. Although the system performs well across several metrics, certain limitations like dataset size, computational cost, and lack of real-time deployment open avenues for further improvement. Future efforts can focus on expanding the dataset, incorporating real-time mobile scanning capabilities, and exploring alternative interpretability techniques. Overall, this work serves as a foundational step toward intelligent, secure, and explainable QR code classification systems that can help mitigate digital threats in practical environments.

REFERENCES

[1] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision, 60*(2), 91–110.

[2] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[3] Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*.

[4] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[5] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojciech, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[6] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[7] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] Zhang, Y., Li, X., Wang, Y., & Liu, H. (2020). Hybrid models for robust image classification. *Journal of Visual Communication and Image Representation, 69*, 102735.

[9] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (pp. 1135–1144).

[10] Mathumitha, V., Kumar, K., Reddy, M. G., Sekhar, K. L., & Pramod, K. (2024). Enhanced detection and decoding of QR code and barcode using machine learning. *International Journal of Research in Engineering, Science and Management, 7*(4), 22–27.

[11] Rajagopalan, A. N., & Kandula, P. (2023). Zero shot framework for satellite image restoration. *arXiv preprint arXiv:2306.02921*.

[12] Mohammed, A. A., Abd, M. M., & Sumari, P. (2025). Remote sensing image classification using convolutional neural network (CNN) and transfer learning techniques. *arXiv preprint arXiv:2503.02510*

Appendices

Appendix 1: QR Code Dataset and Augmentation Techniques

This appendix includes detailed specifications of the dataset used in the classification pipeline. The dataset consists of 500 benign and 500 malicious QR code images.

- Images were resized to a standard dimension of 256x256 pixels.
- Augmentation techniques included:
    - Rotation (±10°)
    - Brightness adjustment
    - Horizontal and vertical flipping
- The dataset was split using an 80/20 ratio for training and testing.

Appendix 2: Model Training and Evaluation

This appendix summarizes the training configurations and evaluation methodology used for all the QR code classification pipelines:

- Train-Test Split: 80% training and 20% testing
- Classifier: Support Vector Machine (SVM) with Radial Basis Function (RBF) kernel
- Cross-Validation: Stratified 5-fold CV applied on training data
- Metrics Evaluated:
    - Accuracy
    - Precision
    - Recall
    - F1-Score
    - AUC (ROC)
    - Average Precision (AP)
    - Cohen’s Kappa
    - Matthews Correlation Coefficient (MCC)
    - Balanced Accuracy

Models were trained on classical (SIFT, ORB + BoVW) and deep (VGG16, ResNet50, InceptionV3, MobileNetV2) feature extractors using consistent training procedures.

Appendix 3: Confusion Matrices

This section provides confusion matrices for all the models. Each matrix presents:

- TP: True Positives
- TN: True Negatives
- FP: False Positives
- FN: False Negatives

Key Observations:

- SIFT + BoVW achieved the highest AUC (0.93) but was sensitive to lighting variations.
- ORB + BoVW was computationally efficient but had lower accuracy (≈68%).
- Deep CNNs (VGG16, ResNet50) offered more stable and balanced results with high recall.
- MobileNetV2 performed well with low computational cost—ideal for edge devices.
- LIME was effective in providing visual justifications, supporting model interpretability.
- Precision-recall trade-offs were clearly visualized in PR curves, especially under imbalanced test cases.

Appendix 4: Model Hyperparameters and Tuning Results

During experimentation, a grid search approach was used to explore combinations of kernel functions, regularization parameters, and feature extraction configurations. The objective was to maximize classification performance—specifically F1-score and ROC AUC—while maintaining generalizability.

- The SVM kernel was tested with linear, rbf, and poly functions. The RBF (Radial Basis Function) kernel provided the best results due to its ability to handle non-linear decision boundaries effectively.
- The regularization parameter (C) was varied across values {0.1, 1, 10, 100}. A value of C = 10 achieved the best trade-off between margin maximization and

classification accuracy.

- For the gamma parameter (specific to the RBF kernel), values scale, auto, and a range from 0.001 to 0.1 were evaluated. The optimal value was gamma = 0.01, which helped control overfitting and improved generalization.
- In the Bag-of-Visual-Words (BoVW) setup, different numbers of clusters were tested: 50, 100, and 150. 100 clusters yielded the most discriminative feature vectors across all models.
- The SIFT descriptor was used with default OpenCV parameters, which provided stable and reliable key point descriptors without requiring manual tuning.
- The ORB descriptor was tested with feature counts of 500 and 1000. The best results were obtained with 500 features, balancing speed and effectiveness.

Appendix 5: References and Dataset Sources

This appendix provides the references and dataset sources specifically relevant to the AI-Based QR Code Classification for Secure Infrastructure and Payment Systems project.
Dataset Sources:

- Benign QR Code Images: Generated using Python libraries such as qrcode and segno, representing safe URLs or text.
- Malicious QR Code Images: Created by embedding phishing URLs or dummy exploit payloads, curated for academic use.
- Image Size: All images were standardized to 256×256 pixels for uniformity during preprocessing.
- Augmentation Techniques: Rotation, flipping, and brightness adjustments were applied to improve generalization.

Technical References:

- SIFT & ORB Feature Extraction:
    - D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", International Journal of Computer Vision, 2004.
    - Rublee et al., “ORB: An Efficient Alternative to SIFT or SURF,” ICCV, 2011.
- BoVW (Bag of Visual Words):

    - Sivic, J., & Zisserman, A., “Video Google: A Text Retrieval Approach to Object Matching in Videos,” ICCV, 2003.
- Deep Learning Models:
    - K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv preprint arXiv:1409.1556 (VGG16).
    - K. He et al., “Deep Residual Learning for Image Recognition,” CVPR, 2016 (ResNet50).
    - C. Szegedy et al., “Rethinking the Inception Architecture for Computer Vision,” CVPR, 2016 (InceptionV3).
    - A. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” arXiv:1704.04861 (MobileNetV2).
- Model Explainability:
    - M. T. Ribeiro, S. Singh, and C. Guestrin, “Why Should I Trust You?: Explaining the Predictions of Any Classifier,” KDD, 2016 (LIME).
- Libraries and Tools Used:
    - TensorFlow/Keras – for deep model feature extraction and CNN integration: https://www.tensorflow.org/
    - OpenCV – for classical image processing and feature detection.
    - scikit-learn – for SVM classification, metrics computation, and evaluation.
    - Matplotlib – for visualization of ROC and PR curves, and confusion matrices.

Appendix 6: Source Code – QR Code Classification System

Code – 1

```
!pip install lime

import os
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.cluster import MiniBatchKMeans
```

```
from sklearn.preprocessing import normalize
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Import LIME for explainability
from lime import lime_tabular

# 1) GLOBAL SETUP & SEED MANAGEMENT
##########################################################################
#######
# Purpose: To ensure reproducibility and to help debug any variability across runs.

GLOBAL_SEED = 42

def set_all_seeds(seed=GLOBAL_SEED):
    print(f"[DEBUG] Setting all seeds to {seed} for reproducibility.")
    random.seed(seed)
    np.random.seed(seed)

set_all_seeds(GLOBAL_SEED)

# Log key library versions (debugging help)
import sys, platform
print(f"[DEBUG] Python version: {sys.version}")
print(f"[DEBUG] OS/Platform: {platform.platform()}")
print(f"[DEBUG] NumPy version: {np.__version__}")

##########################################################################
#######
# 2) DATA LOADING & PREPROCESSING
##########################################################################
#######
# Purpose: To load QR code images, resize them for consistency, and debug the total
count.
import os
import cv2
import numpy as np
import warnings
from google.colab import drive

warnings.filterwarnings("ignore")

# Mount Google Drive
drive.mount('/content/drive')
```

```
# Set Google Drive paths (Adjust if QR_Images is inside another folder)
drive_path = "/content/drive/My Drive/QR_Images"
benign_folder = os.path.join(drive_path, "benign_qr_images_500")
malicious_folder = os.path.join(drive_path, "malicious_qr_images_500")

# Verify if the folders exist
if not os.path.exists(benign_folder) or not os.path.exists(malicious_folder):
    print("[ERROR] One or both dataset folders are missing!")
    print("Check your Google Drive path and folder structure.")
else:
    print("[SUCCESS] Dataset folders found!")

# Function to load images
def load_images(folder, label, img_size=128):
    """
    Loads grayscale images from the given folder, resizes them to (img_size,
img_size),
    and returns a list of tuples: (resized_image, label, filename).
    """
    data = []
    if not os.path.exists(folder):
        print(f"[ERROR] Folder not found: {folder}")
        return []

    files = os.listdir(folder)
    files.sort()  # Sort to ensure consistent order for debugging

    for fname in files:
        path = os.path.join(folder, fname)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        if img is None:
            print(f"[WARNING] Could not load image: {path}")
            continue
        img_resized = cv2.resize(img, (img_size, img_size))
        data.append((img_resized, label, fname))

    return data

# Load benign and malicious images
benign_data = load_images(benign_folder, label=0)
mal_data = load_images(malicious_folder, label=1)
all_data = benign_data + mal_data

print(f"[DEBUG] Loaded {len(benign_data)} benign and {len(mal_data)} malicious
images. Total = {len(all_data)}")
```

```
#############################################################################
#######
# 3) SIFT DESCRIPTOR EXTRACTION
#############################################################################
#######
# Purpose: To extract robust local descriptors (SIFT) from each image.
sift = cv2.SIFT_create()
desc_list = []      # Will store a tuple (descriptors, morph_features, filename)
labels_list = []    # Stores the image label (0 or 1)
files_list = []     # Stores filenames for later reference

for (img, lbl, fname) in all_data:
    # Extract SIFT descriptors
    kp, desc = sift.detectAndCompute(img, None)
    if desc is None or len(desc) == 0:
        # If no keypoints found, create a dummy descriptor
        desc = np.zeros((1, 128), dtype=np.float32)
    # Append SIFT descriptors along with filename
    desc_list.append((desc, fname))
    labels_list.append(lbl)
    files_list.append(fname)
print(f"[DEBUG] Extracted SIFT descriptors for {len(desc_list)} images.")

labels_list = np.array(labels_list)

#############################################################################
#######
# 4) MORPHOLOGICAL/DECODE-BASED FEATURE EXTRACTION
(PLACEHOLDER)
#############################################################################
#######
# Purpose: To add additional features that may help distinguish malicious from
benign,
# such as QR alignment or decoding anomalies.
def extract_morph_features(img):
    """
    Placeholder for extracting morphological or QR decode-based features.
    In a real system, you could analyze:
      - Finder pattern geometry,
      - Alignment patterns,
      - Error correction details, etc.
    Here, we return a dummy feature vector [0, 0].
    """
    return np.array([0.0, 0.0], dtype=np.float32)

# Extract morphological features for each image (dummy in this example)
```

```
morph_features_list = []
for (img, lbl, fname) in all_data:
    morph_feat = extract_morph_features(img)
    morph_features_list.append(morph_feat)
morph_features_list = np.array(morph_features_list)
print(f"[DEBUG] Extracted morphological features with shape:
{morph_features_list.shape}")

##########################################################################
#######
# 5) BUILD BAG-OF-VISUAL-WORDS (BoVW) REPRESENTATION
##########################################################################
#######
# Purpose: To convert variable-length SIFT descriptors into fixed-length
histograms.
all_desc = np.vstack([desc for (desc, fname) in desc_list])
print(f"[DEBUG] all_desc shape: {all_desc.shape}")

# Adjustable vocabulary size: using n_clusters=100 (found best)
NUM_CLUSTERS = 100
print(f"[CONFIG] Using MiniBatchKMeans with n_clusters={NUM_CLUSTERS}
and random_state={GLOBAL_SEED}")

kmeans = MiniBatchKMeans(n_clusters=NUM_CLUSTERS,
random_state=GLOBAL_SEED, batch_size=100)
kmeans.fit(all_desc)

def build_histogram(descriptors, kmeans_model, k):
    """
    Build a histogram of visual words for one image.
    """
    if descriptors is None or len(descriptors) == 0:
        return np.zeros(k, dtype=np.float32)
    cluster_ids = kmeans_model.predict(descriptors)
    hist, _ = np.histogram(cluster_ids, bins=np.arange(k+1))
    return hist.astype(np.float32)

bovw_features = []
for (desc, fname) in desc_list:
    hist = build_histogram(desc, kmeans, NUM_CLUSTERS)
    bovw_features.append(hist)
bovw_features = np.array(bovw_features)
print(f"[DEBUG] BoVW features shape before normalization:
{bovw_features.shape}")

# L2 normalize the histograms
```

```
bovw_features = normalize(bovw_features, norm='l2')
print(f"[DEBUG] BoVW features shape after normalization:
{bovw_features.shape}")

##########################################################################
#######
# 6) COMBINE BoVW FEATURES WITH MORPHOLOGICAL FEATURES
##########################################################################
#######
# Purpose: To append additional features to improve discrimination.
combined_features = np.hstack((bovw_features, morph_features_list))
print(f"[DEBUG] Combined features shape (BoVW + Morph):
{combined_features.shape}")

##########################################################################
#######
# 7) TRAIN-TEST SPLIT
##########################################################################
#######
# Purpose: To split the data into training and test sets with stratification.
X_train, X_test, y_train, y_test, train_files, test_files = train_test_split(
    combined_features, labels_list, files_list,
    test_size=0.2, stratify=labels_list, random_state=GLOBAL_SEED
)
print(f"[DEBUG] Train set size: {X_train.shape[0]}, Test set size:
{X_test.shape[0]}")
print(f"[DEBUG] Training class distribution: Benign={(y_train==0).sum()},
Malicious={(y_train==1).sum()}")
print(f"[DEBUG] Testing class distribution: Benign={(y_test==0).sum()},
Malicious={(y_test==1).sum()}")

##########################################################################
#######
# 8) HYPERPARAMETER TUNING WITH GRIDSEARCHCV OVER C AND
GAMMA
##########################################################################
#######
# Purpose: To find the best SVM hyperparameters for our feature space.
param_grid = {
    'C': [0.1, 1, 10],
    'gamma': ['scale', 1e-2, 1e-3]
}
base_svc = SVC(kernel='rbf', probability=True, class_weight={0:1, 1:2},
random_state=GLOBAL_SEED)

from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(
   estimator=base_svc,
   param_grid=param_grid,
   scoring='accuracy',  # You may use f1_macro for balanced performance.
   cv=5,
   n_jobs=-1,
   verbose=2
)
grid.fit(X_train, y_train)
print("[DEBUG] Best parameters from GridSearch:", grid.best_params_)
print("[DEBUG] Best cross-validation score:", grid.best_score_)

best_svm = grid.best_estimator_

############################################################################
#######
# 9) FINAL EVALUATION USING MANUAL PROBABILITY THRESHOLD =
0.5
############################################################################
#######
# Purpose: To use predict_proba() with a manual threshold for consistent
classification.
proba_test = best_svm.predict_proba(X_test)  # shape: (num_samples, 2)
mal_prob = proba_test[:, 1]                  # probability for class "Malicious"
y_pred_manual = (mal_prob >= 0.5).astype(int)

acc_manual = accuracy_score(y_test, y_pred_manual)
print(f"[INFO] Final Accuracy (manual threshold=0.5): {acc_manual:.4f}")
print("[INFO] Classification Report (manual threshold=0.5):")
print(classification_report(y_test, y_pred_manual,
target_names=["Benign","Malicious"]))

# Plot confusion matrix for manual threshold classification
cm_manual = confusion_matrix(y_test, y_pred_manual)
plt.figure()
plt.imshow(cm_manual, cmap='viridis')
plt.title(f"Confusion Matrix (Manual proba>=0.5) \nAccuracy: {acc_manual:.2f}",
fontsize=14, fontweight='bold')
plt.colorbar()
plt.xlabel("Predicted", fontsize=12)
plt.ylabel("True", fontsize=12)
for i in range(cm_manual.shape[0]):
   for j in range(cm_manual.shape[1]):
      plt.text(j, i, cm_manual[i,j], ha='center', va='center', color='white' if
cm_manual[i,j]>cm_manual.max()/2 else 'black', fontsize=12)
plt.show()
```

```
#########################################################################
#######
# 10) LIME EXPLAINER SETUP
#########################################################################
#######
# Purpose: To allow local, interpretable explanations for the SVM predictions.
from lime import lime_tabular

feature_names = [f"VisualWord_{i}" for i in range(NUM_CLUSTERS)] +
["Morph_1", "Morph_2"]

lime_explainer = lime_tabular.LimeTabularExplainer(
    training_data=X_train,
    training_labels=y_train,
    feature_names=feature_names,
    class_names=["Benign", "Malicious"],
    mode="classification",
    discretize_continuous=True,  # Helps with explanation clarity
    verbose=True
)

print("[DEBUG] Enhanced LIME explainer is ready.")

#########################################################################
#######
# 11) FORENSIC REPORT GENERATOR (USER FRIENDLY)
#########################################################################
#######
def generate_forensic_report(index):
    sample_vec = X_test[index]
    true_label = y_test[index]
    fname = test_files[index]

    # Ensure shape for predict_proba
    if sample_vec.ndim == 1:
        sample_vec = sample_vec.reshape(1, -1)

    # Predict using best SVM
    proba = best_svm.predict_proba(sample_vec)[0]
    prob_ben, prob_mal = proba[0], proba[1]
    pred_label = 1 if prob_mal >= 0.5 else 0
    risk_score = prob_mal * 100

    print("\n========== FORENSIC REPORT ==========")
    print(f"File: {fname}")
```

```
    print(f"True Label:      {'Malicious' if true_label == 1 else 'Benign'}")
    print(f"Predicted Label: {'Malicious' if pred_label == 1 else 'Benign'}")
    print(f"Probability → Benign: {prob_ben:.4f} | Malicious: {prob_mal:.4f}")
    print(f"Risk Score (Malicious Confidence): {risk_score:.1f}/100")

    # Display QR image
    raw_img = next((img for img, lbl, file in all_data if file == fname), None)
    if raw_img is not None:
        plt.figure(figsize=(5, 5))
        plt.imshow(raw_img, cmap='gray')
        plt.title(f"QR Code: {fname}", fontsize=14)
        plt.axis('off')
        plt.show()

    # Risk Meter
    plt.figure(figsize=(6, 1.8))
    plt.barh(["Malicious Risk"], [risk_score], color='crimson' if risk_score >= 50 else
'forestgreen')
    plt.xlim([0, 100])
    plt.xlabel("Confidence (%)")
    plt.title("Malicious Probability")
    plt.tight_layout()
    plt.show()

    # LIME Explanation
    explanation = lime_explainer.explain_instance(
        data_row=X_test[index],
        predict_fn=best_svm.predict_proba,
        num_features=10
    )
    exp_list = explanation.as_list()
    print("\n[LIME Explanation - Top Features]:")
    for feat, weight in exp_list:
        impact = "↑" if weight > 0 else "↓"
        print(f"  {feat}: {weight:+.4f} ({impact} towards {'Malicious' if weight > 0 else
'Benign'})")

    # LIME Bar Chart
    feat_names = [f for f, _ in exp_list]
    feat_weights = [w for _, w in exp_list]
    colors = ['crimson' if w > 0 else 'skyblue' for w in feat_weights]

    plt.figure(figsize=(8, 4))
    y_pos = np.arange(len(feat_names))
    plt.barh(y_pos, feat_weights, color=colors)
    plt.yticks(y_pos, feat_names)
```

```
    plt.gca().invert_yaxis()
    plt.title("LIME: Top Contributing Features", fontsize=13, fontweight='bold')
    plt.xlabel("Feature Impact (Weight)", fontsize=11)
    plt.tight_layout()
    plt.show()

    print("========== END OF REPORT ==========\n")

def pick_qr_for_report():
    """
    Allows the user to select a test sample index to generate a forensic report.
    """
    max_index = len(X_test) - 1
    print(f"[INFO] Please select an index between 0 and {max_index} for a forensic
report.")
    while True:
        user_in = input(f"Enter an index (0-{max_index}) or 'q' to quit: ").strip()
        if user_in.lower() == 'q':
            print("[INFO] Exiting forensic report generator.")
            break
        if not user_in.isdigit():
            print("[ERROR] Invalid input. Please enter a valid integer or 'q'.")
            continue
        idx = int(user_in)
        if 0 <= idx <= max_index:
            generate_forensic_report(idx)
        else:
            print("[ERROR] Index out of range.")

pick_qr_for_report()
```

Code - 2

```
!pip install tensorflow

!pip install opencv-python-headless

from google.colab import drive
import os
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt
import warnings
from sklearn.model_selection import train_test_split
```

```
from sklearn.cluster import MiniBatchKMeans
from sklearn.preprocessing import normalize
from sklearn.svm import SVC
from sklearn.metrics import (
    accuracy_score, confusion_matrix, roc_auc_score, roc_curve,
    average_precision_score, precision_recall_curve,
    precision_recall_fscore_support, cohen_kappa_score,
    matthews_corrcoef, balanced_accuracy_score
)
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input as
vgg_preprocess_input
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input as
resnet_preprocess_input
from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input as inception_preprocess_input
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2,
preprocess_input as mobilenet_preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array

warnings.filterwarnings("ignore")

# Mount Google Drive
drive.mount('/content/drive')

drive_path = "/content/drive/My Drive/QR_Images"
benign_folder = os.path.join(drive_path, "benign_qr_images_500")
malicious_folder = os.path.join(drive_path, "malicious_qr_images_500")

# Verify if the folders exist
if not os.path.exists(benign_folder) or not os.path.exists(malicious_folder):
    print("[ERROR] One or both dataset folders are missing!")
    print("Check your Google Drive path and folder structure.")
else:
    print("[SUCCESS] Dataset folders found!")

# Set seed
global_seed = 42
random.seed(global_seed)
np.random.seed(global_seed)

def load_images_sift(folder, label, img_size=128):
    data = []
    files = sorted(os.listdir(folder))
    for fname in files:
        path = os.path.join(folder, fname)
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
```

```
        if img is None:
            print(f"[WARNING] Could not load image: {path}")
            continue
        img_resized = cv2.resize(img, (img_size, img_size))
        data.append((img_resized, label, fname))
    return data

def load_images_deep(folder, label, img_size=224):
    data = []
    files = sorted(os.listdir(folder))
    for fname in files:
        path = os.path.join(folder, fname)
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        if img is None:
            print(f"[WARNING] Could not load image: {path}")
            continue
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img_resized = cv2.resize(img, (img_size, img_size))
        data.append((img_resized, label, fname))
    return data

def extract_morph_features(img):
    return np.array([0.0, 0.0], dtype=np.float32)

def build_bovw(descriptor_list, num_clusters=100):
    all_desc = np.vstack(descriptor_list)
    kmeans = MiniBatchKMeans(n_clusters=num_clusters,
random_state=global_seed, batch_size=100)
    kmeans.fit(all_desc)
    histograms = []
    for desc in descriptor_list:
        if desc is None or len(desc) == 0:
            hist = np.zeros(num_clusters, dtype=np.float32)
        else:
            cluster_ids = kmeans.predict(desc)
            hist, _ = np.histogram(cluster_ids, bins=np.arange(num_clusters+1))
            hist = hist.astype(np.float32)
        histograms.append(hist)
    histograms = normalize(np.array(histograms), norm='l2')
    return kmeans, histograms

def evaluate_model(svm, X_test, y_test):
    y_pred = svm.predict(X_test)
    y_proba = svm.predict_proba(X_test)[:, 1]
    acc = accuracy_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_proba)
```

```
    avg_precision = average_precision_score(y_test, y_proba)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, y_pred,
average='binary')
    kappa = cohen_kappa_score(y_test, y_pred)
    mcc = matthews_corrcoef(y_test, y_pred)
    balanced_acc = balanced_accuracy_score(y_test, y_pred)
    fpr, tpr, roc_thresholds = roc_curve(y_test, y_proba)
    pr_precision, pr_recall, pr_thresholds = precision_recall_curve(y_test, y_proba)
    cm = confusion_matrix(y_test, y_pred)
    return {
        "accuracy": acc, "auc": auc, "average_precision": avg_precision,
        "precision": precision, "recall": recall, "f1": f1,
        "cohen_kappa": kappa, "matthews_corrcoef": mcc,
        "balanced_accuracy": balanced_acc, "confusion_matrix": cm,
        "roc": (fpr, tpr, roc_thresholds), "pr_curve": (pr_precision, pr_recall,
pr_thresholds)
    }

def plot_individual_results(method_name, metrics):
    fpr, tpr, _ = metrics["roc"]
    plt.figure()
    plt.plot(fpr, tpr, label=f"ROC (AUC={metrics['auc']:.2f})")
    plt.plot([0, 1], [0, 1], 'k--')
    plt.title(f"ROC Curve - {method_name}")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()

    pr_precision, pr_recall, _ = metrics["pr_curve"]
    plt.figure()
    plt.plot(pr_recall, pr_precision, label=f"Precision-Recall
(AP={metrics['average_precision']:.2f})")
    plt.title(f"Precision-Recall Curve - {method_name}")
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.legend()
    plt.show()

results = {}

def run_sift_pipeline():
    print("Running SIFT + BoVW pipeline...")
    data = load_images_sift(benign_folder, 0) + load_images_sift(malicious_folder, 1)
    sift = cv2.SIFT_create()
    descriptors_list, labels, morph_feats = [], [], []
```

```
    for (img, lbl, _) in data:
        kp, desc = sift.detectAndCompute(img, None)
        if desc is None or len(desc) == 0:
            desc = np.zeros((1, 128), dtype=np.float32)
        descriptors_list.append(desc)
        labels.append(lbl)
        morph_feats.append(extract_morph_features(img))

    morph_feats = np.array(morph_feats)
    _, bovw = build_bovw(descriptors_list, num_clusters=100)
    features = np.hstack((bovw, morph_feats))
    labels = np.array(labels)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
                                        stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["SIFT + BoVW"] = metrics
    print("[SIFT] Accuracy:", metrics["accuracy"])
    plot_individual_results("SIFT + BoVW", metrics)

run_sift_pipeline()

def run_orb_pipeline():
    print("Running ORB + BoVW pipeline...")
    data = load_images_sift(benign_folder, 0) + load_images_sift(malicious_folder, 1)
    orb = cv2.ORB_create(nfeatures=500)
    descriptors_list, labels, morph_feats = [], [], []
    for (img, lbl, _) in data:
        kp, desc = orb.detectAndCompute(img, None)
        if desc is None or len(desc) == 0:
            desc = np.zeros((1, 32), dtype=np.float32)
        else:
            desc = np.float32(desc)
        descriptors_list.append(desc)
        labels.append(lbl)
        morph_feats.append(extract_morph_features(img))

    morph_feats = np.array(morph_feats)
    _, bovw = build_bovw(descriptors_list, num_clusters=100)
    features = np.hstack((bovw, morph_feats))
    labels = np.array(labels)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
```

```
                                    stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["ORB + BoVW"] = metrics
    print("[ORB] Accuracy:", metrics["accuracy"])
    plot_individual_results("ORB + BoVW", metrics)

run_orb_pipeline()

def run_vgg16_pipeline():
    print("Running VGG16 feature extraction pipeline...")
    data = load_images_deep(benign_folder, 0) + load_images_deep(malicious_folder,
1)
    model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224,
3))
    features, labels = [], []

    for (img, label, _) in data:
        arr = img_to_array(img)
        arr = np.expand_dims(arr, axis=0)
        arr = vgg_preprocess_input(arr)
        feat = model.predict(arr, verbose=0).flatten()
        features.append(feat)
        labels.append(label)

    features = np.array(features)
    labels = np.array(labels)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
                                    stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["VGG16"] = metrics
    print("[VGG16] Accuracy:", metrics["accuracy"])
    plot_individual_results("VGG16", metrics)

run_vgg16_pipeline()

def run_resnet50_pipeline():
    print("Running ResNet50 feature extraction pipeline...")
    data = load_images_deep(benign_folder, 0) + load_images_deep(malicious_folder,
1)
```

```
    model = ResNet50(weights="imagenet", include_top=False, input_shape=(224,
224, 3))
    features, labels = [], []

    for (img, label, _) in data:
        arr = img_to_array(img)
        arr = np.expand_dims(arr, axis=0)
        arr = resnet_preprocess_input(arr)
        feat = model.predict(arr, verbose=0).flatten()
        features.append(feat)
        labels.append(label)

    features = np.array(features)
    labels = np.array(labels)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
                                        stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["ResNet50"] = metrics
    print("[ResNet50] Accuracy:", metrics["accuracy"])
    plot_individual_results("ResNet50", metrics)

run_resnet50_pipeline()

def run_inception_pipeline():
    print("Running InceptionV3 feature extraction pipeline...")
    data = load_images_deep(benign_folder, 0) + load_images_deep(malicious_folder,
1)
    model = InceptionV3(weights="imagenet", include_top=False, input_shape=(224,
224, 3))
    features, labels = [], []

    for (img, label, _) in data:
        arr = img_to_array(img)
        arr = np.expand_dims(arr, axis=0)
        arr = inception_preprocess_input(arr)
        feat = model.predict(arr, verbose=0).flatten()
        features.append(feat)
        labels.append(label)

    features = np.array(features)
    labels = np.array(labels)
```

```
    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
                                        stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["InceptionV3"] = metrics
    print("[InceptionV3] Accuracy:", metrics["accuracy"])
    plot_individual_results("InceptionV3", metrics)

run_inception_pipeline()

def run_mobilenet_pipeline():
    print("Running MobileNetV2 feature extraction pipeline...")
    data = load_images_deep(benign_folder, 0) + load_images_deep(malicious_folder,
1)
    model = MobileNetV2(weights="imagenet", include_top=False,
input_shape=(224, 224, 3))
    features, labels = [], []

    for (img, label, _) in data:
        arr = img_to_array(img)
        arr = np.expand_dims(arr, axis=0)
        arr = mobilenet_preprocess_input(arr)
        feat = model.predict(arr, verbose=0).flatten()
        features.append(feat)
        labels.append(label)

    features = np.array(features)
    labels = np.array(labels)

    X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2,
                                        stratify=labels, random_state=global_seed)
    svm = SVC(kernel='rbf', probability=True, class_weight='balanced',
random_state=global_seed)
    svm.fit(X_train, y_train)
    metrics = evaluate_model(svm, X_test, y_test)
    results["MobileNetV2"] = metrics
    print("[MobileNetV2] Accuracy:", metrics["accuracy"])
    plot_individual_results("MobileNetV2", metrics)

run_mobilenet_pipeline()

# ## Summary & Comparative Analysis
# Generate a DataFrame of advanced classification metrics across all methods.
```

```
import pandas as pd

metric_names = [
    "accuracy", "auc", "average_precision", "precision", "recall", "f1",
    "cohen_kappa", "matthews_corrcoef", "balanced_accuracy"
]

summary = {metric: [] for metric in metric_names}
methods_available = list(results.keys())

for method in methods_available:
    for metric in metric_names:
        summary[metric].append(results[method][metric])

df_summary = pd.DataFrame(summary, index=methods_available)
print("=== Summary of Advanced Metrics ===")
print(df_summary)

# ### Visualize Advanced Metrics Across Methods

for metric in metric_names:
    plt.figure(figsize=(8, 6))
    values = summary[metric]
    bars = plt.bar(methods_available, values, color='skyblue')
    plt.xlabel("Method")
    plt.ylabel(metric.replace("_", " ").capitalize())
    plt.title(f"Comparison of {metric.replace('_',' ').capitalize()} across Methods")
    plt.xticks(rotation=45, ha="right")
    for bar in bars:
        yval = bar.get_height()
        plt.text(bar.get_x() + bar.get_width()/2.0, yval, f'{yval:.2f}', va='bottom',
ha='center')
    plt.tight_layout()
    plt.show()

# Summarize confusion matrix components: TN, FP, FN, TP
cm_summary = {"TN": [], "FP": [], "FN": [], "TP": []}

for method in methods_available:
    cm = results[method]["confusion_matrix"]
    cm_summary["TN"].append(cm[0, 0])
    cm_summary["FP"].append(cm[0, 1])
    cm_summary["FN"].append(cm[1, 0])
    cm_summary["TP"].append(cm[1, 1])

df_cm = pd.DataFrame(cm_summary, index=methods_available)
```

```
print("=== Confusion Matrix Summary ===")
print(df_cm)

# ### Visualize Confusion Matrix Components

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.flatten()
categories = ["TN", "FP", "FN", "TP"]

for i, cat in enumerate(categories):
    ax = axes[i]
    bars = ax.bar(methods_available, df_cm[cat].values, color='skyblue')
    ax.set_xlabel("Method")
    ax.set_ylabel(cat)
    ax.set_title(f"Comparison of {cat} across Methods")
    ax.tick_params(axis='x', rotation=45)
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2.0, yval, f'{int(yval)}', va='bottom',
ha='center')

plt.tight_layout()
plt.show()
```