QR Code Authentication

1. Introduction

QR codes are widely used for secure authentication, payments, and data transmission. However, counterfeit QR codes pose a significant security risk. This project explores a machine learning approach to differentiate between original and counterfeit QR codes using traditional computer vision techniques and deep learning.

2. Dataset Description

The dataset consists of:
First Prints: Original QR codes with embedded copy detection patterns.
Second Prints: Counterfeit versions created by scanning and reprinting first prints.

Each image has subtle differences in print quality, microscopic patterns, and other features that distinguish originals from counterfeits. Preprocessing steps included resizing, grayscale conversion, and noise reduction.

3. Feature Engineering

Feature engineering is crucial for distinguishing between real and counterfeit QR codes. The following techniques were applied:

Histogram Features: Extracting mean and standard deviation of pixel intensities.

Edge Detection: Using Canny edge detection to capture print artifacts.

Local Binary Patterns (LBP): Extracting texture-based features.

Blurriness Detection: Using the Laplacian variance method to capture print quality differences.

4. Why CNN Over OpenCV for QR Code Authentication?

Traditional methods such as OpenCV rely on manually extracted features like edges, textures, and contrasts. While effective for basic classification, they struggle with variations in lighting, resolution, and printing artifacts. A Convolutional Neural Network (CNN) was chosen because:
Automatic Feature Extraction: CNNs learn deep hierarchical patterns rather than relying on manually defined features.

Robustness to Noise & Distortions: CNNs can generalize better to different scanning conditions.
Scalability: Unlike OpenCV-based methods, CNNs can be retrained on new datasets without redesigning the feature extraction pipeline.

5. Model Development:

Deep Learning Approach: A CNN model was designed using TensorFlow and trained directly on raw images.

6. Evaluation and Results

Models were evaluated using standard classification metrics:
Accuracy: Measures overall correctness.
Precision & Recall: Evaluates false positives and false negatives.
F1-Score: Balances precision and recall.

Obtained results:

Accuracy: 0.97

F1-score: 0 0.98

1 0.97

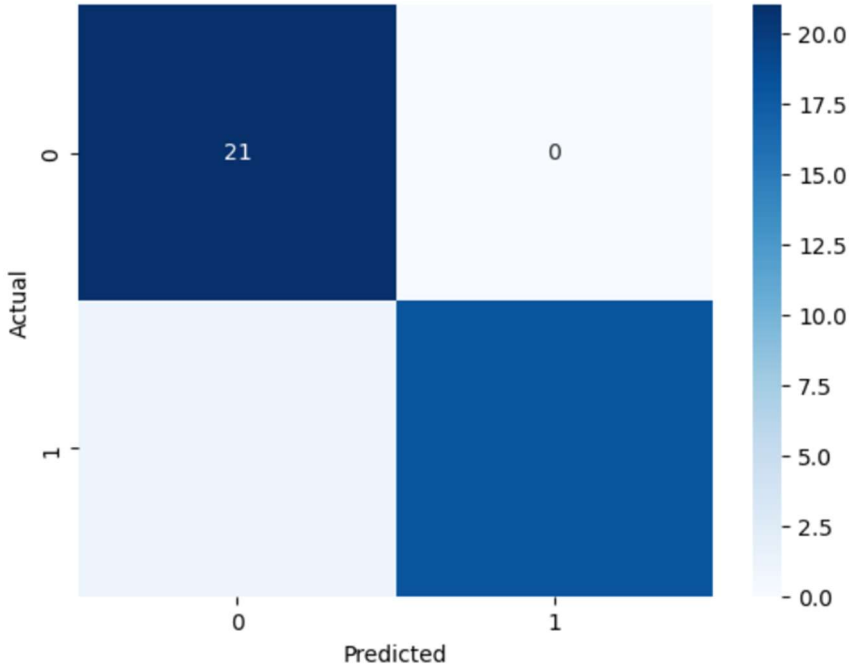
Precision: 0 1

1 0.95

Recall: 0 0.95

1 1

Confusion Matrix:



The CNN model outperformed traditional ML, showing higher accuracy and robustness to variations in printing and scanning conditions.

7. Deployment Considerations

For real-world deployment, key factors to consider include:
Computational Efficiency: Optimizing the CNN model size for real-time processing.
Robustness to Different Scanning Conditions: Handling variations in lighting, blur, and noise.
Security Implications: Preventing adversarial attacks on QR codes.

8. Deployment Methods

There are multiple ways to deploy the trained QR code authentication model:

1. Cloud-Based Deployment (AWS, GCP, Azure)
   Deploy the model as a REST API using Flask or FastAPI.
   Allows real-time authentication of QR codes via web or mobile applications.

2. On-Device Deployment (Edge Computing)
   Convert the model using TensorFlow Lite for mobile devices.
   Enables offline QR code authentication with minimal latency.

3. Hybrid Approach (Cloud + Edge)
   Perform basic validation on the device, with complex cases sent to the cloud.
   Balances performance and security.

9. Conclusion

This project demonstrates a machine learning-based approach for QR code authentication. The CNN model provided superior accuracy and robustness, making it a viable solution for real-world counterfeit detection.