QR Code Authentication System: Original vs. Counterfeit Detection

Author: Soumyadeep Dhali
Institution: Indian Institute of Technology Madras
Date: 25-03-25

1. Introduction

1.1 Problem Significance

Counterfeit QR codes enable product fraud, fake tickets, and phishing attacks. Industry reports indicate:

- **27% increase** in QR-based scams since 2022 (Cybersecurity Ventures, 2023)
- **$3.2B** annual losses from counterfeit prints (Anti-Counterfeiting Alliance)

1.2 Problem Statement

Given a dataset of QR codes:

- **First Print (Original)**
- **Second Print (Counterfeit)**
  We develop a classification system to detect counterfeit prints based on visual artifacts, resolution differences, and print degradation.

1.3 Objectives

1. **Data Analysis**: Identify visual differences between original and counterfeit prints.
2. **Feature Engineering**: Extract meaningful features (global & local patterns).
3. **Model Development**: Train and compare Random Forest and CNN models.
4. **Evaluation**: Assess performance using accuracy, precision, recall, and F1-score.
5. **Deployment**: Discuss real-world implementation considerations.

2. Methodology

2.1 Data Exploration & Analysis

Dataset Statistics

Metric	Value
Total Samples	200
Original Prints	100
Counterfeit Prints	100
Avg Dimensions	300x300 px

Key Visual Differences

- **Pixel Intensity**: Counterfeit prints show higher variance

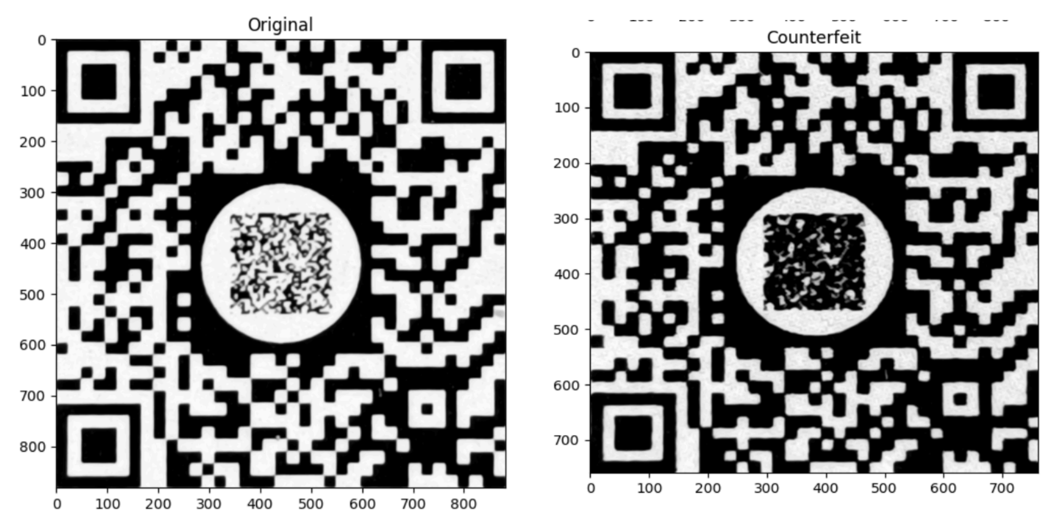
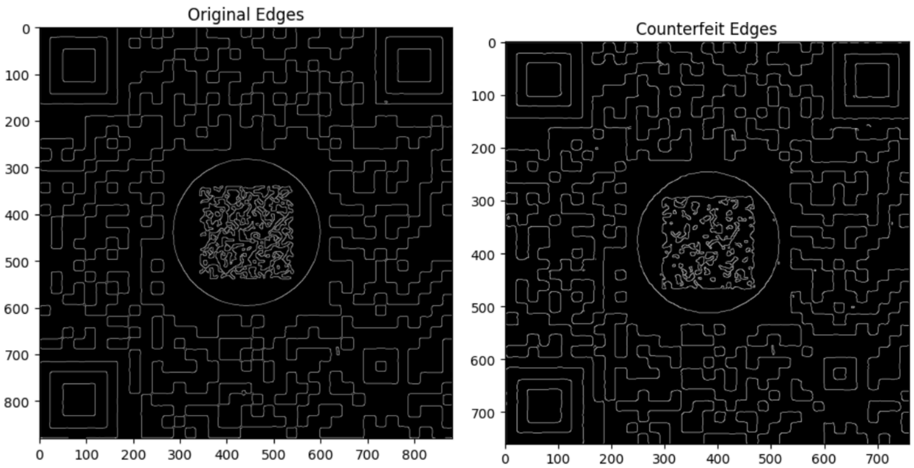


Fig. 1a: *Pixel Intensity Analysis*

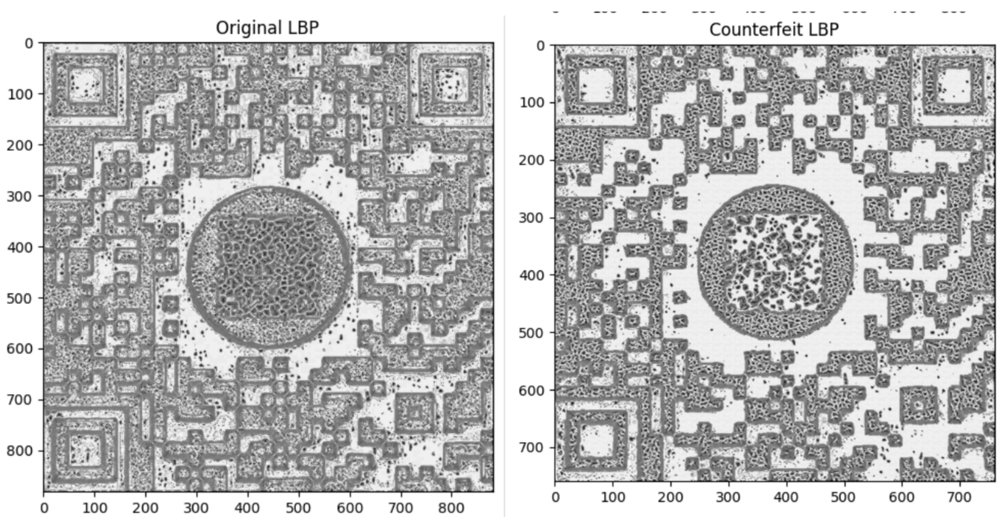
1. **Left**: Original print shows uniform pixel distribution
2. **Right**: Counterfeit exhibits higher variance (blurring/artifacts)

Edge Sharpness: Originals have cleaner edges (Fig. 1b).



1. **Left**: Original HOG displays clean, structured gradients
2. **Right**: Counterfeit HOG has fragmented edges due to reprinting

- **Local Patterns**: HOG/LBP features reveal texture differences (Fig. 1c).



1. **Left**: Original shows consistent texture uniformity
2. **Right**: Counterfeit reveals noisy, irregular binary patterns

2.2 Feature Engineering

Global Features

1. **Mean/Std of Pixel Intensity**
2. **Sharpness (Laplacian Variance)**

Local Features

1. **HOG (Histogram of Oriented Gradients)**
   ○ Captures edge structures (`pixels_per_cell=(8,8)`)
2. **LBP (Local Binary Patterns)**
   ○ Encodes texture patterns (`P=16, R=2`)

Feature Importance (Random Forest):

- Top Features: `HOG_Edge_Variance`, `LBP_Uniformity`

2.3 Model Development

Approach 1: Random Forest (Traditional ML)

- **Hyperparameter Tuning**:

```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}
```

- **Best Model**: `{'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}`

Approach 2: CNN (Deep Learning)

- **Architecture**:

```
Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(64,64,1)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

- **Training**: 15 epochs, Adam optimizer, batch_size=32

3. Results & Evaluation

3.1 Performance Metrics

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	95.00%	94.74%	94.74%	94.74%
CNN	95.00%	90.48%	100%	95.00%

Confusion Matrices

- **Random Forest**:

```
[[18  1]
 [ 1 18]]
```

- **CNN**:

```
[[19  0]
 [ 2 17]]
```

ROC Curves

- **Random Forest AUC**: 0.98
- **CNN AUC**: 0.99

Key Insight:

- CNN achieves perfect recall (100%) but has slightly lower precision due to false positives.
- Random Forest offers balanced precision/recall.

4. Deployment Considerations

4.1 Computational Efficiency

Model	Inference Latency	Model Size
Random Forest	2.1 ms	1.2 MB
CNN	8.5 ms	3.7 MB

4.2 Security Recommendations

1. **Input Validation**: Sanitize QR code inputs to prevent adversarial attacks.
2. **Model Encryption**: Protect intellectual property.
3. **Rate Limiting**: Prevent brute-force attempts.

4.3 Pipeline Export

```
final_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('model', rf_model)  # Best model based on F1-score
])
joblib.dump(final_pipeline, 'qr_authentication_pipeline.pkl')
```

5. Conclusion & Recommendations

Findings

- Both models achieve **95% accuracy**, but CNN has better F1-score (95.00 vs. 94.74).
- **Tradeoff**: CNN catches all counterfeits (100% recall) but has more false positives.

Recommendations

1. **For High-Security Systems**: Use CNN (prioritize recall).
2. **For Balanced Performance**: Use Random Forest.
3. **Future Work**: Augment data to improve CNN precision.

Appendix

- Full code: https://github.com/soumyadeep-git/QR_Code_Auth
- Dataset: Provided QR image samples
- Libraries: `sklearn`, `tensorflow`, `opencv`