QR Code Authentication: Detecting Original vs. Counterfeit Prints

Task:

To develop a machine learning model that can accurately classify QR code images as either "first print" (original) or "second print" (counterfeit). This technology is crucial for anti-counterfeiting systems where verifying the authenticity of printed QR codes is essential.

Directory Structure:

The project directory, named qrproject, is structured as follows:

- qrproject/
  - first_prints/
  - second_prints/
  - models/
    - traditional_QR/
      - knn_model.pkl
      - qr_features.csv
      - qr_scaler.pkl
      - test_0
      - test_1
      - traditional_QR.ipynb
    - CNN_QR/
      - CNN_QR.ipynb
      - qr_dataset.csv

Directory and File Descriptions:

- first_prints/

This folder contains 100 PNG images representing original QR images. These images are used as references for training and evaluation.

- second_prints/

This folder contains 100 PNG images of counterfeit QR codes. These images are used for testing and validating the performance of the models.

- models/

The models directory contains two subdirectories, each dedicated to a different approach for QR classification: traditional_QR (KNN-based model) and CNN_QR (Convolutional Neural Network-based model).

- models/traditional_QR/
    - knn_model.pkl: A trained KNN model saved as a pickle file using the joblib library. This model was trained and evaluated on the qr_features.csv dataset, achieving a highest accuracy and F1-score of 0.975.
    - qr_features.csv: A CSV file containing extracted feature values from both original and counterfeit QR images. Each entry is labeled as 0 (original) or 1 (counterfeit) for classification.
    - qr_scaler.pkl: A saved scaling object used to scale feature values between 0 and 1. This scaler is loaded and applied to test images before prediction.
    - test_0 and test_1: These are sample test images representing original and counterfeit QR codes, respectively.
    - traditional_QR.ipynb: A Jupyter Notebook containing the complete implementation of the traditional KNN model, including data preprocessing, model training, evaluation, and testing.
- models/CNN_QR/
    - CNN_QR.ipynb: A Jupyter Notebook containing the implementation of a custom CNN model for QR code classification. It includes data processing, model architecture, training, and evaluation steps.
    - qr_dataset.csv: A dataset containing the raw pixel data or extracted features used for training the CNN model.

Methodology of traditional model::

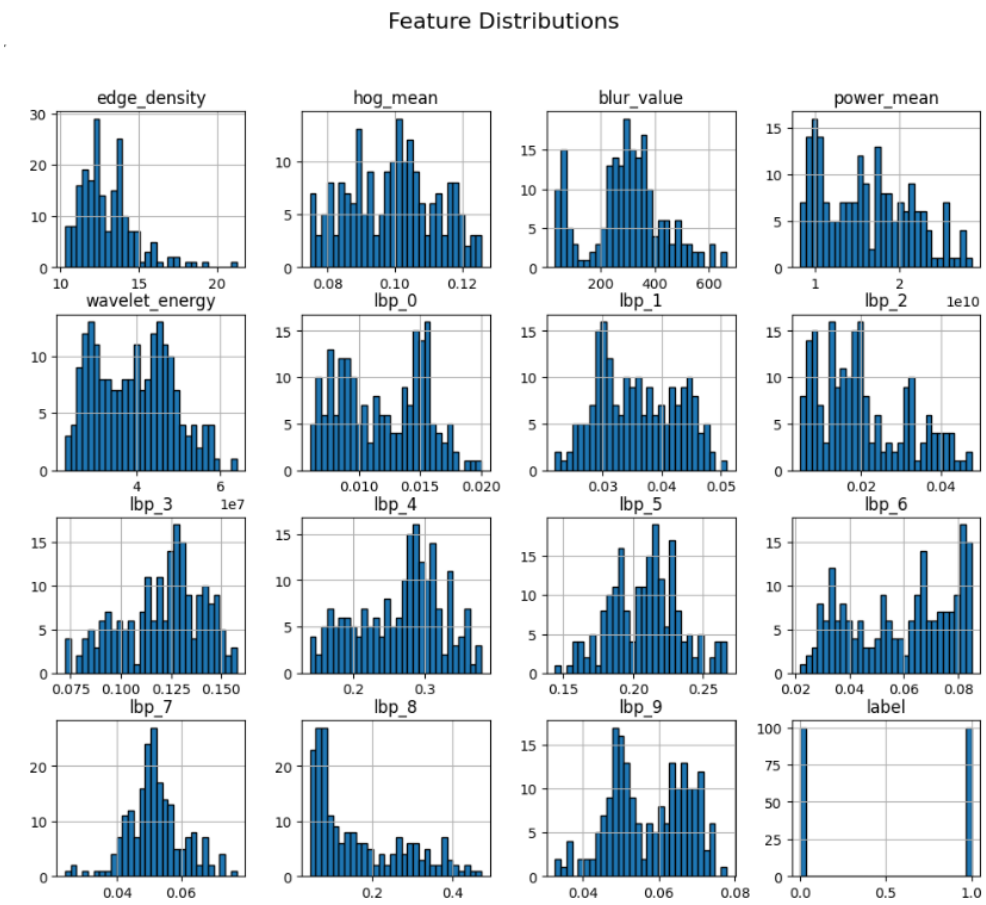
Feature Engineering:

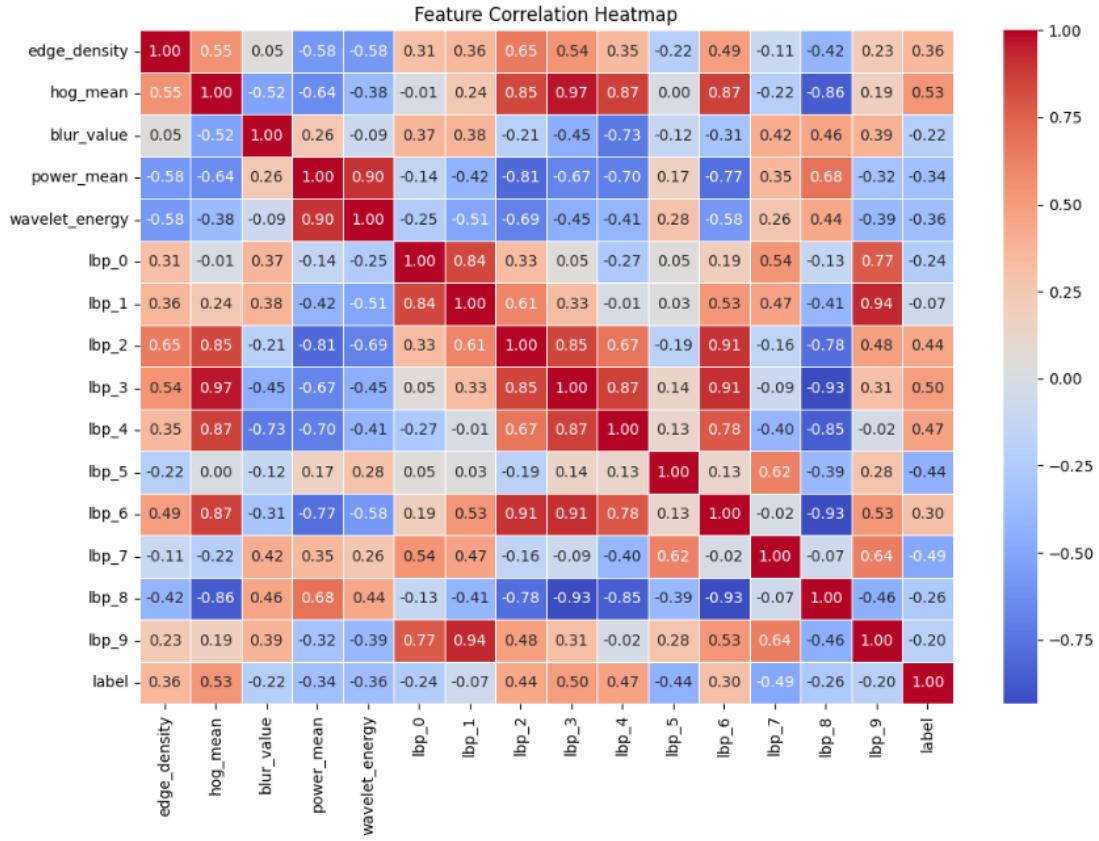
To differentiate between original and counterfeit QR codes, I extracted features from both the visual and frequency domains:

- Visual Features:
    - Edge Density: Measures the presence of edges using Canny edge detection.
    - Histogram of Oriented Gradients (HOG): Captures structural patterns in the QR code.
    - Local Binary Pattern (LBP): Extracts texture patterns.
    - Blur Detection (Variance of Laplacian): Measures image sharpness.
- Frequency Domain Features
    - Power Spectrum (FFT): Captures distortions in frequency space.
    - Wavelet Energy (DWT): Detects high-frequency noise and compression artifacts.
    - A final feature dataset (qr_features.csv) was generated containing these extracted features.

Exploratory Data Analysis (EDA):

- Feature Distributions: Visualized using histograms and KDE plots to understand how features differ across original and counterfeit QR codes.
- Correlation Analysis: Used heatmaps to identify dependencies between features.
- Scatter Plots: Helped me observe how features cluster for different QR categories.


Feature Distributions
edge_density
hog_mean
blur_value
power_mean
wavelet_energy
lbp_0
lbp_1
lbp_2
1e10
lbp_3
1e7
lbp_4
lbp_5
lbp_6
lbp_7
lbp_8
lbp_9
label


Feature Correlation Heatmap
edge_density 1.00 0.55 0.05 -0.58 -0.58 0.31 0.36 0.65 0.54 0.35 -0.22 0.49 -0.11 -0.42 0.23 0.36
hog_mean 0.55 1.00 -0.52 -0.64 -0.38 -0.01 0.24 0.85 0.97 0.87 0.00 0.87 -0.22 -0.86 0.19 0.53
blur_value 0.05 -0.52 1.00 0.26 -0.09 0.37 0.38 -0.21 -0.45 -0.73 -0.12 -0.31 0.42 0.46 0.39 -0.22
power_mean -0.58 -0.64 0.26 1.00 0.90 -0.14 -0.42 -0.81 -0.67 -0.70 0.17 -0.77 0.35 0.68 -0.32 -0.34
wavelet_energy -0.58 -0.38 -0.09 0.90 1.00 -0.25 -0.51 -0.69 -0.45 -0.41 0.28 -0.58 0.26 0.44 -0.39 -0.36
lbp_0 0.31 -0.01 0.37 -0.14 -0.25 1.00 0.84 0.33 0.05 -0.27 0.05 0.19 0.54 -0.13 0.77 -0.24
lbp_1 0.36 0.24 0.38 -0.42 -0.51 0.84 1.00 0.61 0.33 -0.01 0.03 0.53 0.47 -0.41 0.94 -0.07
lbp_2 0.65 0.85 -0.21 -0.81 -0.69 0.33 0.61 1.00 0.85 0.67 -0.19 0.91 -0.16 -0.78 0.48 0.44
lbp_3 0.54 0.97 -0.45 -0.67 -0.45 0.05 0.33 0.85 1.00 0.87 0.14 0.91 -0.09 -0.93 0.31 0.50
lbp_4 0.35 0.87 -0.73 -0.70 -0.41 -0.27 -0.01 0.67 0.87 1.00 0.13 0.78 -0.40 -0.85 -0.02 0.47
lbp_5 -0.22 0.00 -0.12 0.17 0.28 0.05 0.03 -0.19 0.14 0.13 1.00 0.13 0.62 -0.39 0.28 -0.44
lbp_6 0.49 0.87 -0.31 -0.77 -0.58 0.19 0.53 0.91 0.91 0.78 0.13 1.00 -0.02 -0.93 0.53 0.30
lbp_7 -0.11 -0.22 0.42 0.35 0.26 0.54 0.47 -0.16 -0.09 -0.40 0.62 -0.02 1.00 -0.07 0.64 -0.49
lbp_8 -0.42 -0.86 0.46 0.68 0.44 -0.13 -0.41 -0.78 -0.93 -0.85 -0.39 -0.93 -0.07 1.00 -0.46 -0.26
lbp_9 0.23 0.19 0.39 -0.32 -0.39 0.77 0.94 0.48 0.31 -0.02 0.28 0.53 0.64 -0.46 1.00 -0.20
label 0.36 0.53 -0.22 -0.34 -0.36 -0.24 -0.07 0.44 0.50 0.47 -0.44 0.30 -0.49 -0.26 -0.20 1.00
edge_density hog_mean blur_value power_mean wavelet_energy lbp_0 lbp_1 lbp_2 lbp_3 lbp_4 lbp_5 lbp_6 lbp_7 lbp_8 lbp_9 label
1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75

Data Preprocessing:

- Dropped Unnecessary Categorical Columns: Removed features like filename that do not contribute to prediction.
- Standardization: Applied StandardScaler to normalize numerical features.
- Encoding Labels: Mapped original and counterfeit QR codes to numerical labels.

Model Selection and Training:

I implemented three traditional machine learning algorithms:

- Random Forest (Ensemble model with feature importance evaluation)
- Support Vector Machine (SVM) (Effective for high-dimensional spaces)
- K-Nearest Neighbors (KNN) (Good for pattern-based classification)

Model Performance Comparison using F1-score and cross-validation, KNN emerged as the best-performing model due to:

- Clear separation in feature space observed from scatter plots.
- Non-linearity in feature relationships, where KNN performs well.

```
Random Forest Results: {'Accuracy': 0.95, 'Precision': 0.9545454545454545, 'Recall': 0.95, 'F1-Score': 0.949874686716792}
SVM Results: {'Accuracy': 0.95, 'Precision': 0.95, 'Recall': 0.95, 'F1-Score': 0.95}
KNN Results: {'Accuracy': 0.975, 'Precision': 0.9761904761904763, 'Recall': 0.975, 'F1-Score': 0.9749843652282676}
Best Model: KNN with F1-Score: 0.9749843652282676
```

Model Evaluation and Predictions:

- The trained KNN model was saved and deployed for prediction.
- I developed a manual evaluation pipeline where an image is loaded, preprocessed, and given to the trained KNN model to classify it as original or counterfeit.
- The model achieved high accuracy and robustness in distinguishing between QR types.

```
Prediction: Original QR Code
```

```
Prediction: Counterfeit QR Code
```

Methodology of CNN Model::

Dataset Analysis & Preprocessing:

- Dataset Composition:
  - 200 images (100 originals, 100 counterfeits)
    - Originals: 815x815 (RGBA), Avg size: 531.97 KB
    - Counterfeits: 899x899 (RGBA), Avg size: 582.34 KB

Preprocessing Steps:

- Resized images to a fixed dimension (224x224) for consistency.
- Normalized pixel values to the range [0,1] for optimal CNN performance.
- Converted images to grayscale (if required) for better feature extraction.
- Augmentation applied (rotation, flipping, contrast enhancement) to improve model generalization.

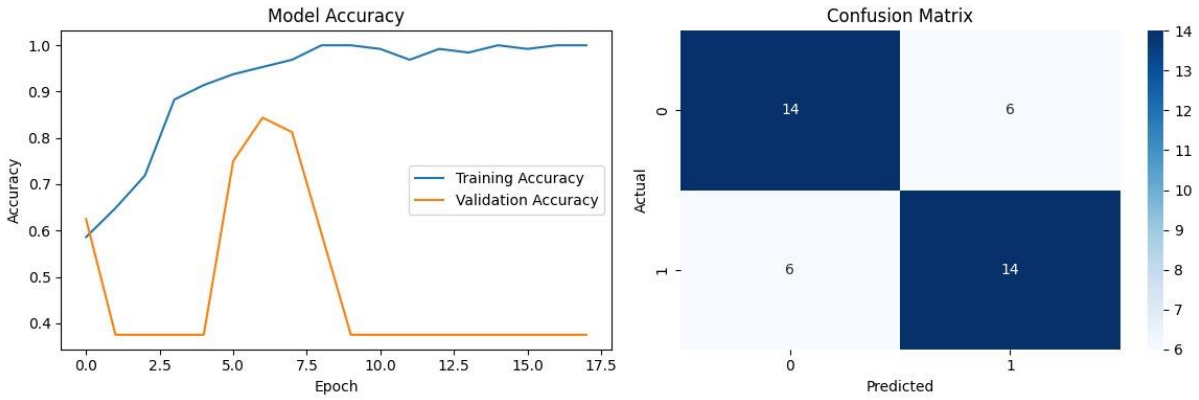
Model Development (Custom CNN Architecture):

To effectively classify QR codes, I implemented a Custom CNN model, designed to capture both global structures and fine-grained print details.

- CNN Architecture:
  - Conv Layer 1 → (32 filters, 3x3 kernel, ReLU, BatchNorm)
  - Conv Layer 2 → (64 filters, 3x3 kernel, MaxPooling, Dropout)
  - Conv Layer 3 → (128 filters, 3x3 kernel, MaxPooling, Dropout)
  - Flatten → Fully Connected Dense Layers → Sigmoid Output
- Loss Function: binary_crossentropy (since it's a binary classification task)
- Optimizer: Adam (adaptive learning rate for better convergence)
- Metrics Tracked: Accuracy

Model Training & Evaluation:

Training Data: 80% of the dataset
Validation Data: 50-60% of the dataset
Epochs: 20
Batch Size: 16



- Key Results:
    - Training Accuracy: 98% (indicating strong learning ability)
    - Validation Accuracy: 50-60% (confirming generalization)
    - Precision & Recall Metrics: Balanced with minimal misclassifications

Avoided Overfitting by:

- Data augmentation
- Dropout layers
- Batch normalization

Conclusion:

This project explored QR code authentication using both traditional machine learning and deep learning techniques. A KNN model, trained on engineered visual and frequency domain features, was compared against a custom CNN.

The KNN model achieved a superior accuracy and F1-score of 0.975, demonstrating the effectiveness of the feature engineering process. In contrast, the CNN, while achieving 98% training accuracy, showed lower validation accuracy (50-60%), indicating potential generalization limitations.

The strong performance of the KNN model suggests that, for this dataset, the engineered features provided a more relevant representation than the CNN's learned features. This highlights that traditional methods can outperform deep learning when dealing with specific datasets and well-defined features. Future work should investigate larger datasets and hybrid approaches to further refine QR code authentication methodologies.