Name: Sushant Bagul

QR Code Authentication

1. Introduction

QR codes are widely used for authentication, data sharing, and security purposes. However, different prints of the same QR code may exhibit print artifacts, resolution differences, and degradation, affecting authentication accuracy. This project aims to analyze these variations using computer vision techniques, feature extraction methods, and deep learning models (CNNs).

2. Problem Statement

The objective is to distinguish between two versions of printed QR codes (First Print and Second Print) using traditional image processing (LBP, edge detection, FFT, and blur detection) and deep learning approaches (CNNs). This will help understand how QR codes degrade over multiple prints and improve authentication reliability

3. Solution

We can implement machine learning classifier models to classify between original and counterfeit images

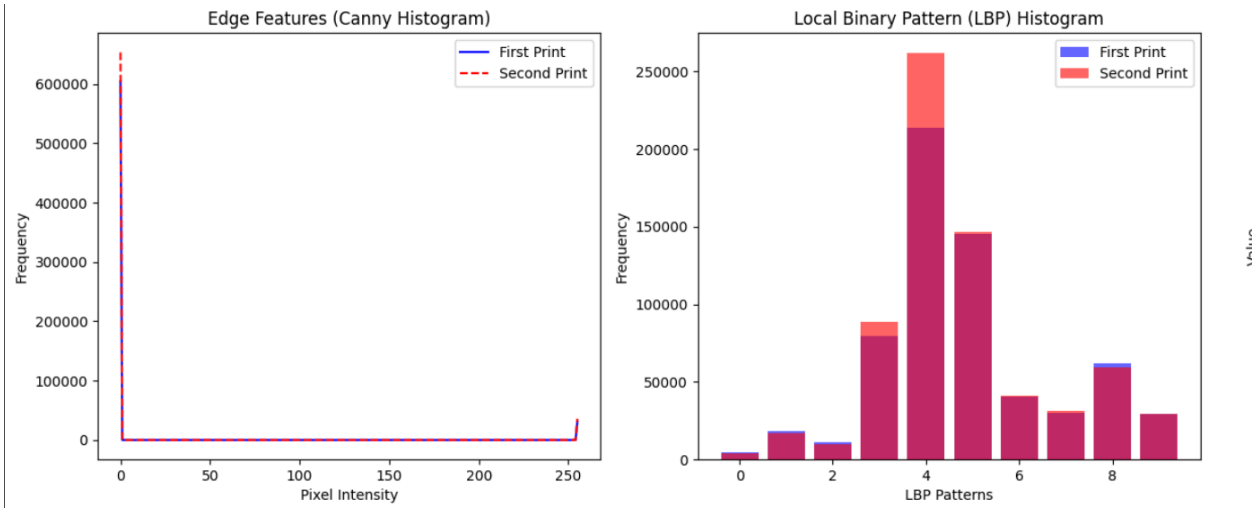
4) Step 1: Data exploration and feature extraction

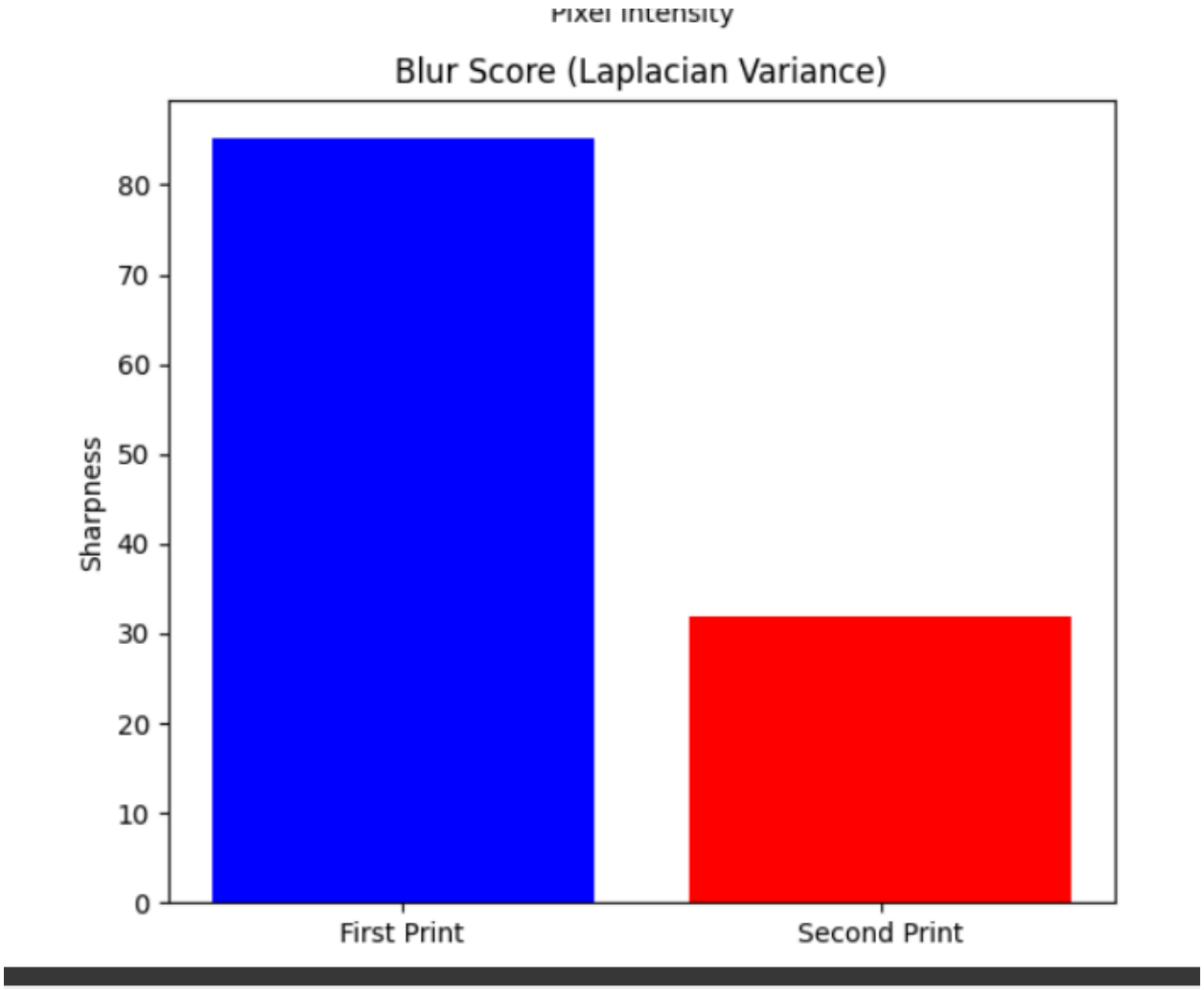
- **Edge Detection:** Edge detection is used to analyze the sharpness and structure of the QR code by detecting significant changes in pixel intensity. The Canny Edge Detector is an advanced method that identifies edges by computing gradients and removing noise.

  It works by applying Gaussian smoothing to reduce noise, detecting gradients to locate edges, and using non-maximum suppression to retain only the strongest edges. By setting double threshold values, weak and strong edges are distinguished, ensuring that only meaningful edges are detected. In the case of QR codes, the First Print typically has sharper and well-defined edges, whereas the Second Print may exhibit broken, faded, or distorted edges due to multiple scanning and printing processes.

  By analyzing the edge histograms, we can quantify the extent of degradation between prints.

- **Local Binary Patterns (LBP) -** Texture AnalysisLocal Binary Patterns (LBP) is a feature extraction technique that focuses on texture variations in an image. It encodes pixel intensity differences between neighboring pixels, capturing fine details in textures. Each pixel is compared with its surrounding neighbors, and a binary pattern is generated based on whether the neighboring pixels have a higher or lower intensity. This method is particularly useful for identifying print artifacts, ink spreading, and noise in QR codes. The First Print generally has a more uniform texture, while the Second Print may show irregular patterns due to ink diffusion, distortion, and minor printing inconsistencies. LBP histograms provide a statistical representation of texture differences, making it easier to distinguish between different print versions.
- **Fast Fourier Transform (FFT)** is used to analyze an image in the frequency domain, revealing patterns that are not always visible in the spatial domain. It decomposes an image into low and high-frequency components, where low frequencies represent smooth regions, and high frequencies capture sharp transitions such as edges. QR codes rely on well-defined black and white modules, making frequency analysis crucial in detecting print distortions. The First Print generally maintains a consistent frequency distribution, while the Second Print may exhibit blurring and loss of high-frequency details due to scanning artifacts. By calculating the mean and variance of the frequency components, we can measure the loss of resolution and quality between print generations.
- **Blur detection** is crucial for assessing the sharpness of printed QR codes. The Laplacian Variance method measures image sharpness by computing the variance of the second-order derivatives. A higher variance indicates a sharp image, while a lower variance signifies blurriness. Since QR codes rely on precise black-and-white contrast for scanning accuracy, any loss in sharpness can affect their readability. The First Print is expected to have a higher variance (less blur), while the Second Print may appear blurred due to multiple scanning and printing cycles. The reduction in sharpness can impact QR code authentication, making blur detection a critical factor in assessing print quality.





5) Step 2: Model Training

5.1Random Forest Classifier:

Before training the model, we extract essential features from the QR code images using various image processing techniques:

- Edge Features (Canny Edge Detection Histogram)
- Local Binary Pattern (LBP Histogram for texture analysis)
- Fast Fourier Transform (FFT - Frequency domain analysis)
- Blur Detection (Laplacian Variance to measure sharpness degradation)

Each QR code image is transformed into a feature vector, which represents the extracted numerical values. The dataset is then split into training and testing sets (typically **an 80-20 split) to ensure the model generalizes well to unseen data.**

Creating the Random Forest Model

Once the dataset is ready, we initialize the Random Forest Classifier with a specified number of decision trees (n_estimators). Each tree will independently learn from a different subset of the training data. The key hyperparameters include:

Training
Bootstrapping: Random subsets of the training data are sampled (with replacement) to train each tree. This ensures diversity among the trees.

Feature Subset Selection: At each decision node within a tree, a random subset of features is selected to find the best split. This reduces overfitting and ensures different trees focus on different patterns.

Recursive Splitting: Each tree grows by recursively splitting data based on the most important feature at each step, creating a decision boundary for classification.

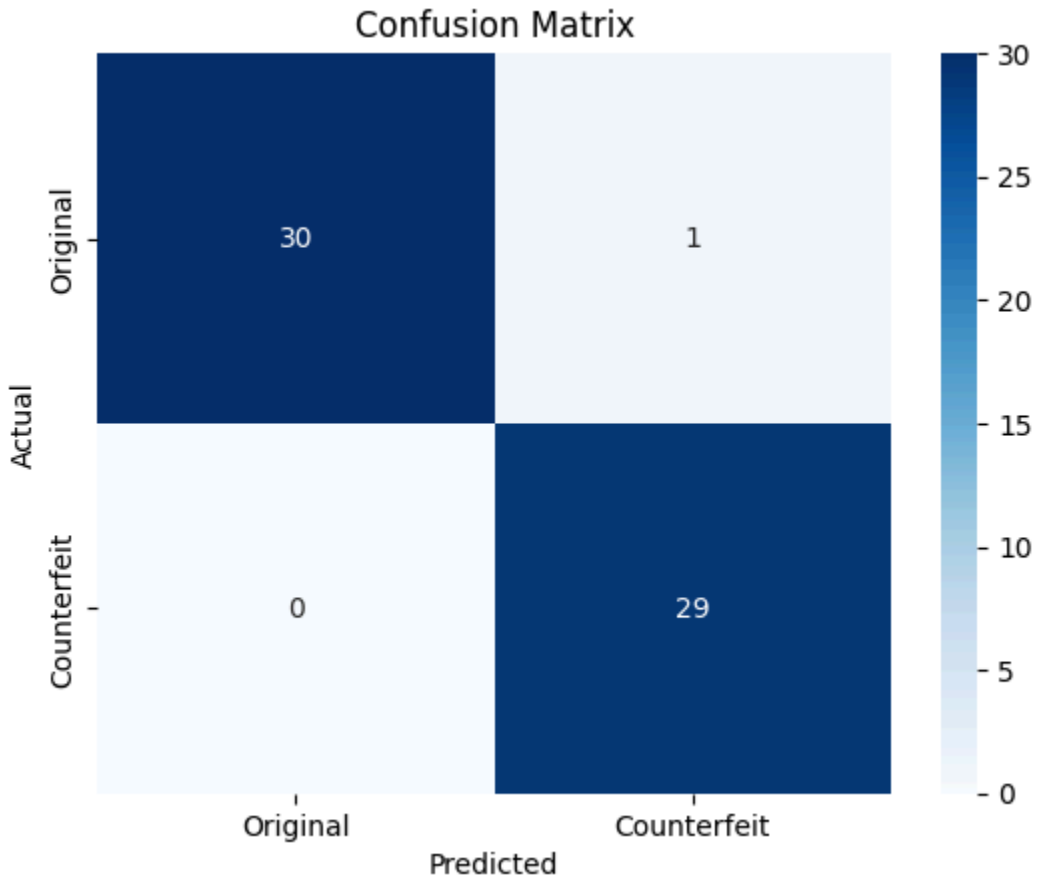
Model evaluation

Once training is complete, the model is evaluated on the test dataset using several performance metrics:

- Accuracy: Measures the percentage of correctly classified images.
- Precision: Evaluates how many predicted Second Print QR codes were correct.
- Recall (Sensitivity): Measures how well the model detects Second Print QR codes without missing them.

- F1-Score: Balances Precision and Recall to assess overall performance.
- Confusion Matrix: Provides insights into misclassification patterns, helping understand where the model struggles.

	precision	recall	f1-score	support
Original	1.00	0.97	0.98	31
Counterfeit	0.97	1.00	0.98	29
accuracy			0.98	60
macro avg	0.98	0.98	0.98	60
weighted avg	0.98	0.98	0.98	60



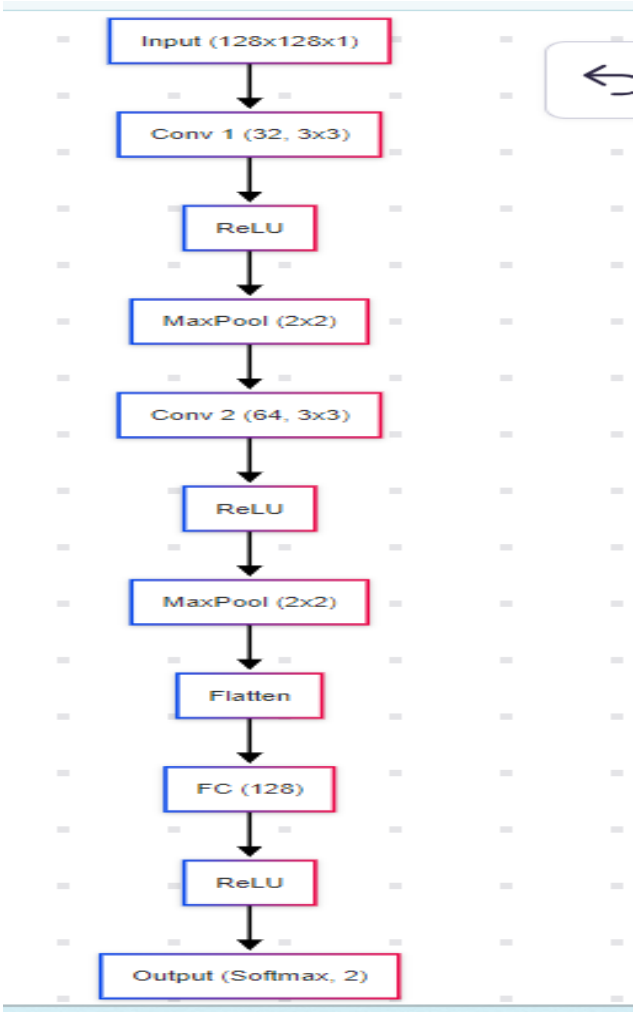
5.2) Using Convolutional Neural Networks

Step 1: Data Preprocessing and Augmentation

Before training, the QR code images must be preprocessed to ensure consistent input format for the CNN. The preprocessing steps include:

- Grayscale Conversion: Since QR codes do not contain color information, we convert them to grayscale to reduce computational complexity.
- Resizing: All images are resized to a fixed dimension (e.g., 128x128 pixels) to maintain consistency in the CNN's input layer.
- Normalization: Pixel values are scaled to a range of [0,1] or [-1,1] to improve gradient descent efficiency.

Step 2: CNN Model Architecture Design



A CNN consists of several layers that automatically extract and learn features from QR code images. The architecture used includes:

1. **Convolutional Layers:**Filters (kernels) slide over the input image to detect patterns such as edges, textures, and shapes.
   The first layers detect basic features (e.g., edges), while deeper layers capture complex patterns (e.g., QR artifacts).

2. **Activation Functions:**ReLU (Rectified Linear Unit) is applied after each convolution to introduce non-linearity, allowing the network to learn complex relationships.

3. **Pooling Layers:**Max Pooling reduces the spatial size of feature maps while retaining important features, improving efficiency and reducing overfitting.

4. **Fully Connected Layers (FC):**Flattened feature maps are passed through dense layers, which classify the QR codes into First Print or Second Print.

5. **Softmax Output Layer:**The final layer applies Softmax activation, which converts outputs into probabilities for each class (First or Second Print).

Step 3: Forward Propagation & Feature Learning

Once the architecture is set up, the model is trained using forward propagation:

- The input image passes through convolutional and pooling layers, extracting relevant spatial features.
- The output from these layers is flattened and passed through fully connected layers to learn classification boundaries.
- The Softmax function assigns a probability score to each class (First Print vs. Second Print).

At this stage, the model makes an initial prediction, but it is usually incorrect due to randomly initialized weights.

Step 4: Loss Calculation and Backpropagation

To optimize the model, we calculate the difference between the predicted and actual labels using a loss function (e.g., Cross-Entropy Loss for classification tasks).

- Backpropagation is then used to adjust the CNN's weights and filters:

    1. The loss is propagated backward through the network.
    2. Gradients are computed for each parameter (filter weights, biases).
    3. The optimizer (e.g., Adam or SGD) updates the parameters to minimize loss.

This iterative process allows the CNN to improve feature extraction and classification accuracy over multiple training cycles (epochs).

Step 5: Training the Model (Epochs and Batches) Training is done in multiple epochs, where the CNN repeatedly processes the dataset to improve its predictions.

- The dataset is divided into mini-batches to efficiently update weights without requiring too much memory.
- At each epoch:The model makes predictions.Loss is calculated and backpropagated.The optimizer updates weights to minimize errors.

- Validation Accuracy & Loss: After each epoch, the model is evaluated on the validation dataset to monitor performance and detect overfitting.

Step 6: Model Evaluation and Performance Metrics

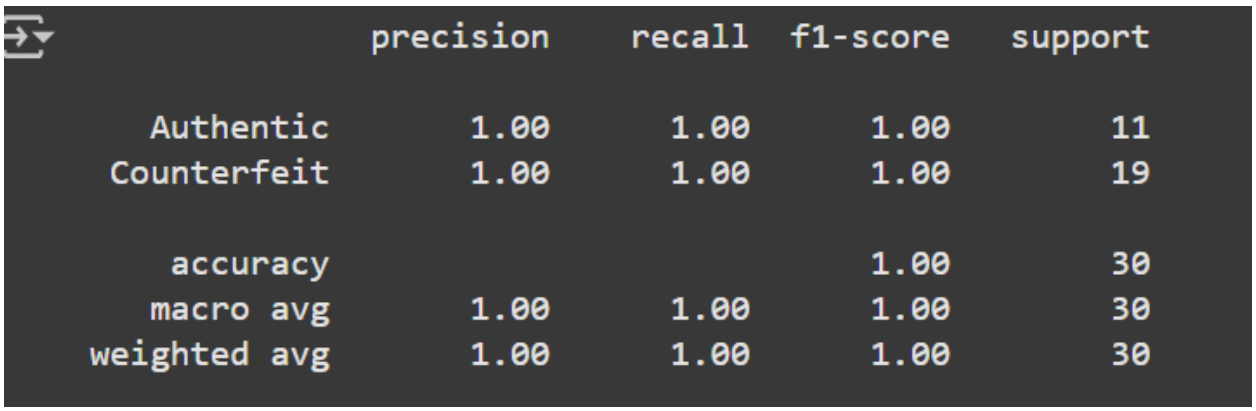
After training, the model is tested on an unseen test dataset to evaluate its generalization ability. The key performance metrics include:

- Accuracy: Percentage of correctly classified QR codes.
- Precision & Recall: Helps assess the model's ability to differentiate between the two print versions.
- F1-Score: Balances precision and recall for an overall performance measure.
- Confusion Matrix: Provides insights into false positives and false negatives.

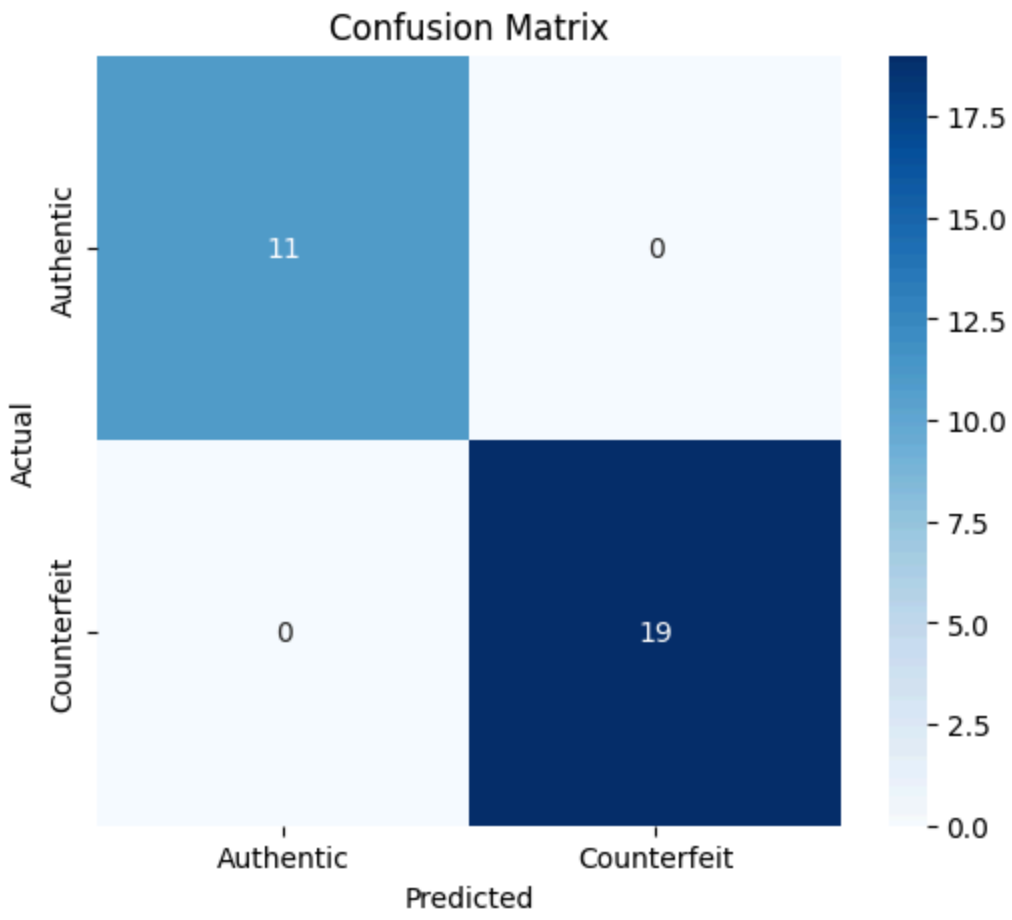
Step 7: Hyperparameter Tuning & Optimization

To further improve the CNN model's performance, several tuning techniques are applied:

- Learning Rate Adjustments: Fine-tuning the learning rate to ensure smooth convergence.



	precision	recall	f1-score	support
Authentic	1.00	1.00	1.00	11
Counterfeit	1.00	1.00	1.00	19
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30


Confusion Matrix
Authentic
Counterfeit
Actual
11
0
0
19
Authentic
Counterfeit
Predicted
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0