## Command_Instructions

Step 1: Open VS Code and navigate to your project directory:
cd "E:\Projects\QR Code Authentication Model"

Step 2: Create and activate your virtual environment:
venv\Scripts\activate

Step 3: Upgrade pip and install required packages:
pip install --upgrade pip
pip install -r requirements.txt

Step 4: Run the data_exploration.ipynb notebook to inspect and visualize the dataset.

Step 5: Execute the traditional pipeline script to build an SVM classifier using HOG features.
python src/traditional_pipeline.py

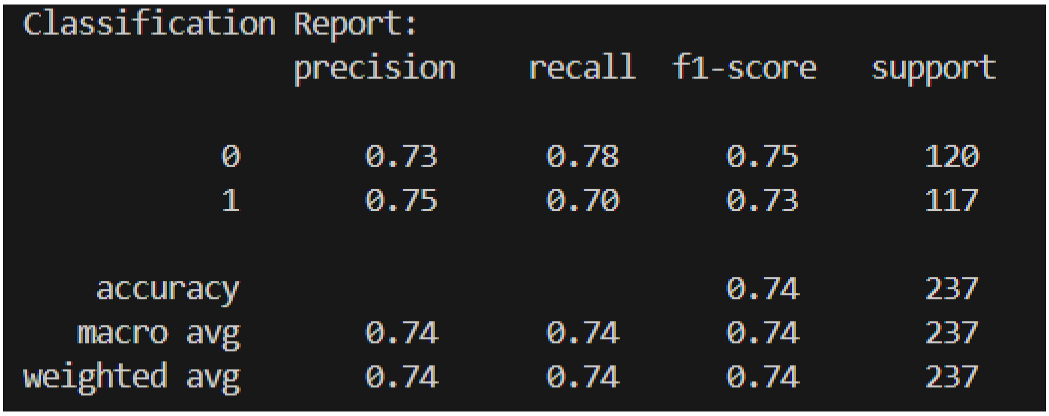
Step 6: Execute the CNN training script.
python src/cnn_model.py

Step 7: Run the cnn_training.ipynb notebook for interactive visualization of CNN training metrics.

## Terminal_Outputs

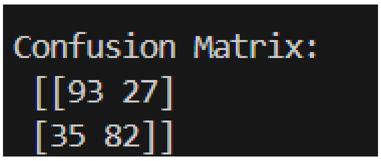
```
(venv) PS E:\Projects\QR Code Authentication Model> python src/traditional_pipeline.py
>>
[INFO] Loading dataset...
[INFO] Training SVM classifier...
```

Classification Report:



```
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.78      0.75       120
           1       0.75      0.70      0.73       117

    accuracy                           0.74       237
   macro avg       0.74      0.74      0.74       237
weighted avg       0.74      0.74      0.74       237
```

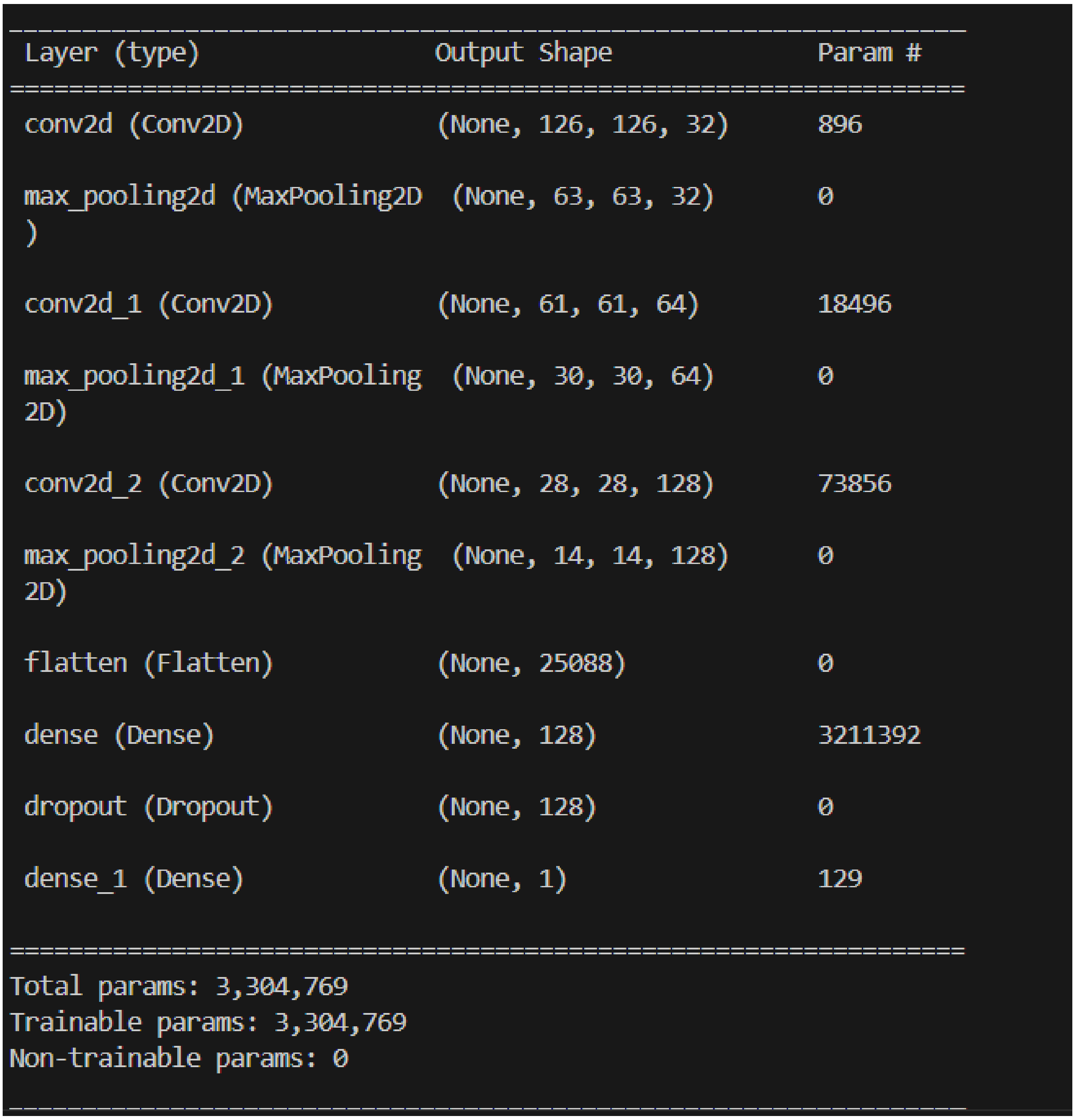
Confusion Matrix:



```
Confusion Matrix:
 [[93 27]
 [35 82]]
```

[INFO] Model saved as svm_qr_classifier.joblib

```
(venv) PS E:\Projects\QR Code Authentication Model> python src/cnn_model.py
Found 946 images belonging to 2 classes.
Found 235 images belonging to 2 classes.
2025-03-27 15:37:17.247767: I tensorflow/core/platform/cpu_feature_guard.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2025-03-27 15:37:18.464978: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device /job:localhost/replica:0/task:0/device:GPU:0 with 2135 MB memory:  -> device: 0, name: NVIDIA GeForce GTX 1650, pci bus id: 0000:01:00.0, compute capability: 7.5
Model: "sequential"
```



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2D  (None, 63, 63, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 28, 28, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 14, 14, 128)      0
 2D)

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 128)               3211392

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 3,304,769
Trainable params: 3,304,769
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/30
2025-03-27 15:37:21.733169: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384] Loaded cuDNN version 8101
29/29 [==============================] - ETA: 0s - loss: 0.7154 - accuracy: 0.5263
```

```
Epoch 1: val_accuracy improved from -inf to 0.52679, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 15s 265ms/step - loss: 0.7154 - accuracy: 0.5263 - val_loss: 0.6824 - val_accuracy: 0.5268
Epoch 2/30
29/29 [==============================] - ETA: 0s - loss: 0.6904 - accuracy: 0.5317
Epoch 2: val_accuracy did not improve from 0.52679
29/29 [==============================] - 6s 212ms/step - loss: 0.6904 - accuracy: 0.5317 - val_loss: 0.6731 - val_accuracy: 0.5268
Epoch 3/30
29/29 [==============================] - ETA: 0s - loss: 0.6727 - accuracy: 0.5853
Epoch 3: val_accuracy did not improve from 0.52679
29/29 [==============================] - 6s 202ms/step - loss: 0.6727 - accuracy: 0.5853 - val_loss: 0.6557 - val_accuracy: 0.4955
Epoch 4/30
29/29 [==============================] - ETA: 0s - loss: 0.6445 - accuracy: 0.6105
Epoch 4: val_accuracy improved from 0.52679 to 0.72768, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 204ms/step - loss: 0.6445 - accuracy: 0.6105 - val_loss: 0.5987 - val_accuracy: 0.7277
Epoch 5/30
29/29 [==============================] - ETA: 0s - loss: 0.5866 - accuracy: 0.7254
Epoch 5: val_accuracy did not improve from 0.72768
29/29 [==============================] - 6s 203ms/step - loss: 0.5866 - accuracy: 0.7254 - val_loss: 0.5542 - val_accuracy: 0.6830
Epoch 6/30
29/29 [==============================] - ETA: 0s - loss: 0.5190 - accuracy: 0.7615
Epoch 6: val_accuracy improved from 0.72768 to 0.91071, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 209ms/step - loss: 0.5190 - accuracy: 0.7615 - val_loss: 0.4387 - val_accuracy: 0.9107
Epoch 7/30
29/29 [==============================] - ETA: 0s - loss: 0.4616 - accuracy: 0.8239
Epoch 7: val_accuracy did not improve from 0.91071
29/29 [==============================] - 6s 201ms/step - loss: 0.4616 - accuracy: 0.8239 - val_loss: 0.4226 - val_accuracy: 0.8348
Epoch 8/30
29/29 [==============================] - ETA: 0s - loss: 0.4239 - accuracy: 0.8063
Epoch 8: val_accuracy did not improve from 0.91071
29/29 [==============================] - 6s 203ms/step - loss: 0.4239 - accuracy: 0.8063 - val_loss: 0.3484 - val_accuracy: 0.8929
Epoch 9/30
29/29 [==============================] - ETA: 0s - loss: 0.3917 - accuracy: 0.8523
Epoch 9: val_accuracy improved from 0.91071 to 0.92411, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 208ms/step - loss: 0.3917 - accuracy: 0.8523 - val_loss: 0.2931 - val_accuracy: 0.9241
Epoch 10/30
29/29 [==============================] - ETA: 0s - loss: 0.3462 - accuracy: 0.8665
Epoch 10: val_accuracy did not improve from 0.92411
29/29 [==============================] - 6s 205ms/step - loss: 0.3462 - accuracy: 0.8665 - val_loss: 0.3005 - val_accuracy: 0.8884
Epoch 11/30
29/29 [==============================] - ETA: 0s - loss: 0.3560 - accuracy: 0.8468
Epoch 11: val_accuracy did not improve from 0.92411
```

```
29/29 [==============================] - 6s 202ms/step - loss: 0.3560 - accuracy: 0.8468 - val_loss:
0.3268 - val_accuracy: 0.8527
Epoch 12/30
29/29 [==============================] - ETA: 0s - loss: 0.3355 - accuracy: 0.8446
Epoch 12: val_accuracy improved from 0.92411 to 0.94196, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 211ms/step - loss: 0.3355 - accuracy: 0.8446 - val_loss:
0.2317 - val_accuracy: 0.9420
Epoch 13/30
29/29 [==============================] - ETA: 0s - loss: 0.3245 - accuracy: 0.8731
Epoch 13: val_accuracy did not improve from 0.94196
29/29 [==============================] - 6s 206ms/step - loss: 0.3245 - accuracy: 0.8731 - val_loss:
0.2685 - val_accuracy: 0.9018
Epoch 14/30
29/29 [==============================] - ETA: 0s - loss: 0.3087 - accuracy: 0.8807
Epoch 14: val_accuracy improved from 0.94196 to 0.95089, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 206ms/step - loss: 0.3087 - accuracy: 0.8807 - val_loss:
0.2142 - val_accuracy: 0.9509
Epoch 15/30
29/29 [==============================] - ETA: 0s - loss: 0.2668 - accuracy: 0.8933
Epoch 15: val_accuracy did not improve from 0.95089
29/29 [==============================] - 6s 208ms/step - loss: 0.2668 - accuracy: 0.8933 - val_loss:
0.2332 - val_accuracy: 0.9152
Epoch 16/30
29/29 [==============================] - ETA: 0s - loss: 0.2543 - accuracy: 0.9037
Epoch 16: val_accuracy did not improve from 0.95089
29/29 [==============================] - 6s 202ms/step - loss: 0.2543 - accuracy: 0.9037 - val_loss:
0.2035 - val_accuracy: 0.9286
Epoch 17/30
29/29 [==============================] - ETA: 0s - loss: 0.2654 - accuracy: 0.8939
Epoch 17: val_accuracy improved from 0.95089 to 0.95536, saving model to cnn_qr_classifier.h5
29/29 [==============================] - 6s 205ms/step - loss: 0.2654 - accuracy: 0.8939 - val_loss:
0.1962 - val_accuracy: 0.9554
Epoch 18/30
29/29 [==============================] - ETA: 0s - loss: 0.2651 - accuracy: 0.8961
Epoch 18: val_accuracy did not improve from 0.95536
29/29 [==============================] - 6s 214ms/step - loss: 0.2651 - accuracy: 0.8961 - val_loss:
0.2254 - val_accuracy: 0.9152
Epoch 19/30
29/29 [==============================] - ETA: 0s - loss: 0.2681 - accuracy: 0.8917
Epoch 19: val_accuracy did not improve from 0.95536
29/29 [==============================] - 5s 180ms/step - loss: 0.2681 - accuracy: 0.8917 - val_loss:
0.2366 - val_accuracy: 0.8973
Epoch 20/30
29/29 [==============================] - ETA: 0s - loss: 0.2666 - accuracy: 0.8851
Epoch 20: val_accuracy did not improve from 0.95536
29/29 [==============================] - 5s 169ms/step - loss: 0.2666 - accuracy: 0.8851 - val_loss:
0.2482 - val_accuracy: 0.9018
Epoch 21/30
29/29 [==============================] - ETA: 0s - loss: 0.2532 - accuracy: 0.8873
Epoch 21: val_accuracy did not improve from 0.95536
29/29 [==============================] - 5s 168ms/step - loss: 0.2532 - accuracy: 0.8873 - val_loss:
0.1757 - val_accuracy: 0.9554
```

```
Epoch 22/30
29/29 [==============================] - ETA: 0s - loss: 0.2199 - accuracy: 0.9081
Epoch 22: val_accuracy did not improve from 0.95536
29/29 [==============================] - 5s 170ms/step - loss: 0.2199 - accuracy: 0.9081 - val_loss:
0.2327 - val_accuracy: 0.9018
Epoch 22: early stopping
[INFO] Training history saved as notebooks/cnn_history.pkl
8/8 [==============================] - 1s 107ms/step - loss: 0.2653 - accuracy: 0.8894
Validation Accuracy: 88.94%
(venv) PS E:\Projects\QR Code Authentication Model>
```

## Codes and Results

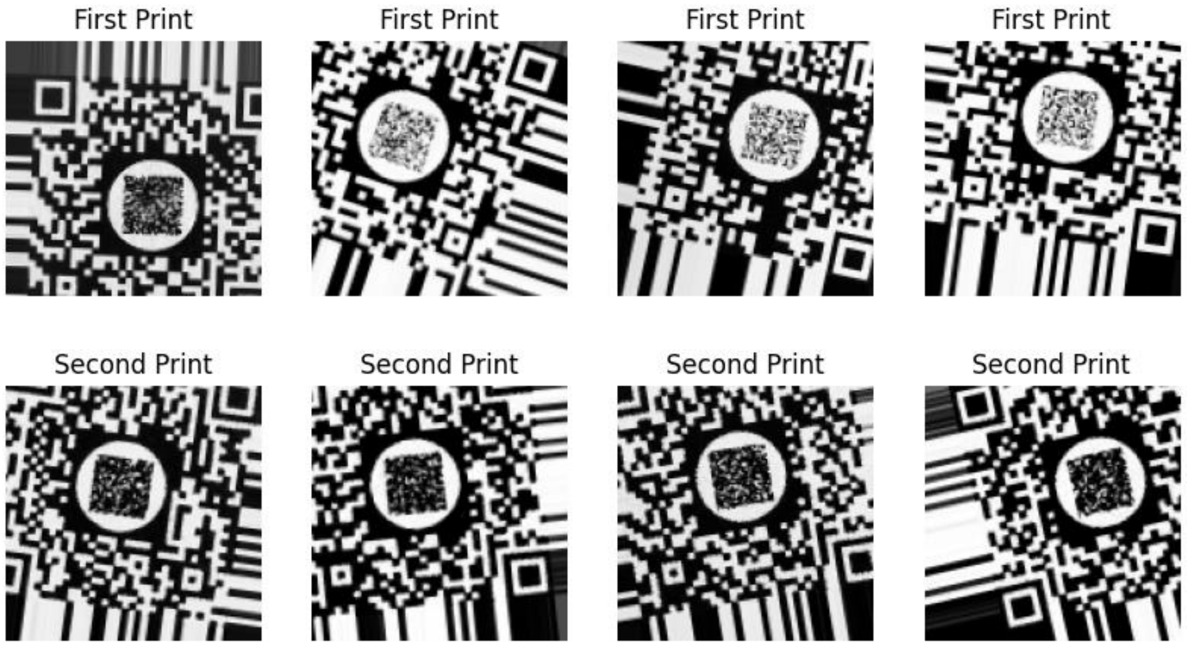
### Data Exploration ipynb

```
# notebooks/data_exploration.ipynb
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Define dataset path
dataset_path = r"E:\Projects\QR Code Authentication Model\dataset"
first_print_path = os.path.join(dataset_path, "First_Print")
second_print_path = os.path.join(dataset_path, "Second_Print")

def load_image_paths(folder):
    return [os.path.join(folder, file) for file in os.listdir(folder) if file.lower().endswith(('.png', '.jpg', '.jpeg'))]

first_images = load_image_paths(first_print_path)
second_images = load_image_paths(second_print_path)

print("Number of First Print images:", len(first_images))
print("Number of Second Print images:", len(second_images))

# Function to display sample images
def show_samples(image_paths, title, num_samples=4):
    plt.figure(figsize=(10, 10))
    for i, img_path in enumerate(image_paths[:num_samples]):
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        plt.subplot(1, num_samples, i+1)
        plt.imshow(img)
        plt.title(title)
        plt.axis('off')
    plt.show()

show_samples(first_images, "First Print")
show_samples(second_images, "Second Print")
```

```
# Analyze image dimensions
def get_image_dims(image_paths):
    dims = []
    for path in image_paths:
        img = cv2.imread(path)
        dims.append(img.shape[:2])
    return dims

dims_first = get_image_dims(first_images)
dims_second = get_image_dims(second_images)

# Plot height distributions for both categories
h_first, w_first = zip(*dims_first)
h_second, w_second = zip(*dims_second)

sns.kdeplot(h_first, label="First Print Height", shade=True)
sns.kdeplot(h_second, label="Second Print Height", shade=True)
plt.legend()
plt.title("Distribution of Image Heights")
plt.show()
```

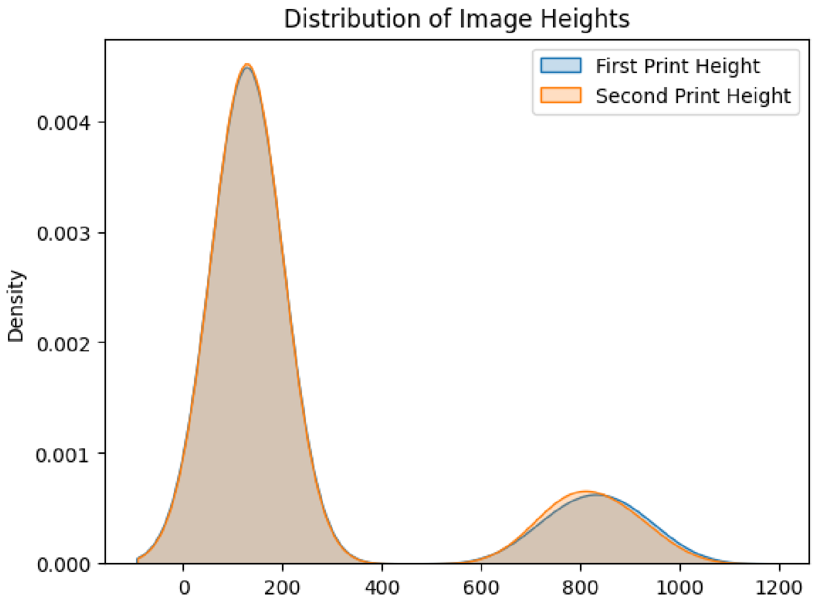
**Output:**

```
Number of First Print images: 593
Number of Second Print images: 588
```



```
C:\Users\samsa\AppData\Local\Temp\ipykernel_14172\3967304756.py:52: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(h_first, label="First Print Height", shade=True)
C:\Users\samsa\AppData\Local\Temp\ipykernel_14172\3967304756.py:53: FutureWarning:
```

```
`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.
  sns.kdeplot(h_second, label="Second Print Height", shade=True)
```



## Traditional_Pipeline.py

```
# src/traditional_pipeline.py

import os

import cv2

import numpy as np

from imutils import paths

from skimage.feature import hog

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import classification_report, confusion_matrix

import joblib


# Set dataset paths

dataset_path = r"E:\Projects\QR Code Authentication Model\dataset"

first_print_path = os.path.join(dataset_path, "First_Print")

second_print_path = os.path.join(dataset_path, "Second_Print")


def load_dataset():

    imagePaths = []

    labels = []
    for imgPath in paths.list_images(first_print_path):
        imagePaths.append(imgPath)
        labels.append(0)  # label 0 for First Print (original)
    for imgPath in paths.list_images(second_print_path):
        imagePaths.append(imgPath)
        labels.append(1)  # label 1 for Second Print (counterfeit)
    return imagePaths, labels

def extract_features(imagePath):
    image = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize(image, (128, 128))
    features, _ = hog(image, orientations=9, pixels_per_cell=(8, 8),
                      cells_per_block=(2, 2), block_norm='L2-Hys',
                      visualize=True)
    return features
def main():
    print("[INFO] Loading dataset...")
    imagePaths, labels = load_dataset()
    data = []
    for path in imagePaths:
        features = extract_features(path)
        data.append(features)
    data = np.array(data)
    labels = np.array(labels)

    # Split the dataset into training and testing
    (trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.2, random_state=42)

    # Train the SVM classifier
    print("[INFO] Training SVM classifier...")
    model = SVC(kernel='linear', probability=True, random_state=42)
    model.fit(trainX, trainY)
    # Evaluate the model

    preds = model.predict(testX)

    print("Classification Report:\n", classification_report(testY, preds))

    print("Confusion Matrix:\n", confusion_matrix(testY, preds))


    # Save the trained model

    joblib.dump(model, "svm_qr_classifier.joblib")

    print("[INFO] Model saved as svm_qr_classifier.joblib")


if __name__ == "__main__":

    main()
```

### cnn_model.py

```
# src/cnn_model.py
import os
import pickle
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

# Define dataset path and image parameters
dataset_path = r"E:\Projects\QR Code Authentication Model\dataset"
img_height, img_width = 128, 128
batch_size = 32
epochs = 30

# Data augmentation using ImageDataGenerator with a validation split of 20%
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # 20% for validation
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1
)

train_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='training',
    shuffle=True
)

validation_generator = train_datagen.flow_from_directory(
    dataset_path,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='binary',
    subset='validation',
    shuffle=False
)

def build_model():
    model = models.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', input_shape=(img_height, img_width, 3)),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(64, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(128, (3,3), activation='relu'),
        layers.MaxPooling2D((2,2)),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(1, activation='sigmoid')
    ])

    model.compile(
        optimizer=optimizers.Adam(learning_rate=1e-4),
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

model = build_model()
model.summary()

# Callbacks for saving the best model and early stopping
checkpoint = ModelCheckpoint("cnn_qr_classifier.h5", monitor='val_accuracy', save_best_only=True,
verbose=1)
earlystop = EarlyStopping(monitor='val_accuracy', patience=5, verbose=1)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size,
    epochs=epochs,
    callbacks=[checkpoint, earlystop]
)

# Save training history for later visualization into the 'notebooks' folder
with open("notebooks/cnn_history.pkl", "wb") as f:
    pickle.dump(history.history, f)
print("[INFO] Training history saved as notebooks/cnn_history.pkl")

# Evaluate and save the final model
loss, acc = model.evaluate(validation_generator)
print(f"Validation Accuracy: {acc*100:.2f}%")
model.save("final_cnn_qr_classifier.h5")
```

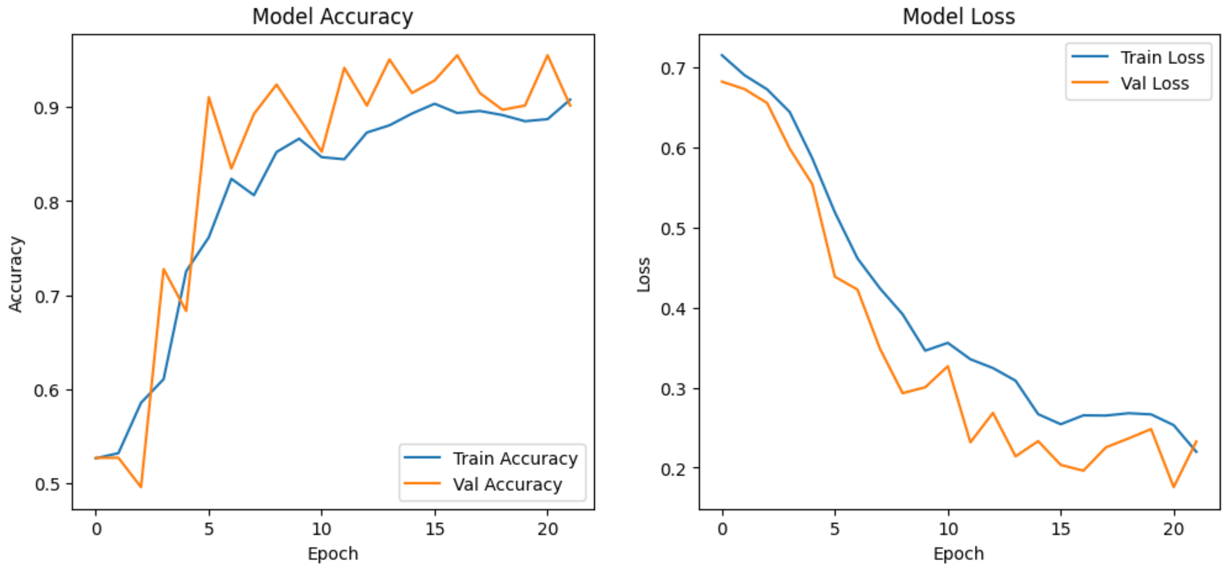
### utils.py

```
# src/utils.py
import os
import cv2
import matplotlib.pyplot as plt

def load_image_paths(folder, extensions=('.png', '.jpg', '.jpeg')):
    """
    Return a list of image paths in a given folder filtered by specified extensions.
    """
    return [os.path.join(folder, file) for file in os.listdir(folder) if file.lower().endswith(extensions)]

def load_and_preprocess_image(image_path, size=(128, 128), color_mode='grayscale'):
    """
    Load an image from disk, resize it, and convert its color space.

    :param image_path: Path to the image file.
    :param size: Tuple of (width, height) to resize the image.
    :param color_mode: 'grayscale' or 'rgb'
    :return: Preprocessed image.
    """
    if color_mode == 'grayscale':
        image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    else:
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    image = cv2.resize(image, size)
    return image

def plot_sample_images(image_paths, title="Sample Images", num_samples=4):
    """
    Plot a set of sample images from a list of image paths.
    """
    plt.figure(figsize=(10, 10))
    for i, img_path in enumerate(image_paths[:num_samples]):
        img = load_and_preprocess_image(img_path, size=(128, 128), color_mode='rgb')
        plt.subplot(1, num_samples, i+1)
        plt.imshow(img)
        plt.title(title)
        plt.axis('off')
    plt.show()
```

```
if __name__ == "__main__":
    # Example usage:
    sample_folder = r"E:\Projects\QR Code Authentication Model\dataset\First_Print"
    paths = load_image_paths(sample_folder)
    print(f"Found {len(paths)} images in {sample_folder}")
    plot_sample_images(paths, title="First Print Samples")
```

### cnn_training.ipynb

```
# notebooks/cnn_training.ipynb
import matplotlib.pyplot as plt
import pickle
# Load the training history (a dictionary, not a History object)
with open("cnn_history.pkl", "rb") as f:
    history_dict = pickle.load(f)
# Now 'history_dict' has keys like 'accuracy', 'val_accuracy', 'loss', 'val_loss'
plt.figure(figsize=(12, 5))
# Plot Accuracy
plt.subplot(1, 2, 1)
plt.plot(history_dict['accuracy'], label='Train Accuracy')
plt.plot(history_dict['val_accuracy'], label='Val Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history_dict['loss'], label='Train Loss')
plt.plot(history_dict['val_loss'], label='Val Loss')
plt.title("Model Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```


Model Accuracy
Accuracy
Epoch
Train Accuracy
Val Accuracy
Model Loss
Loss
Epoch
Train Loss
Val Loss