Face Recognition and Skin Cancer Detection Project Report

Ahmed

October 1, 2023

1 Task 1: Face Detection and Preprocessing

In this task, the project involved implementing a face detection system using OpenCV's Haar cascade classifier. The detected faces were then preprocessed, including grayscale conversion, resizing, and normalization. The implementation ensured that all input images were consistent in format and size. The following code snippet illustrates the face detection and preprocessing process:

Listing 1: Face Detection and Preprocessing Code

```
def load_and_preprocess_data(self):
        data = []
        labels = []
        input_size = None

        # Define a function to extract the
        expression label
        from the file name
        def extract_expression_label(file_name):
            return file_name.split('_')[1]

        # Load data and labels
        dataset_dir = 'C:/Users/admin/Desktop/Facial-
--------Recognition-System/data/yalefaces_resized'
        for file_name in os.listdir(dataset_dir):
            img_path =
            os.path.join(dataset_dir, file_name)
            if img_path.lower().endswith('.png'):
                img = Image.open(img_path).convert('L')
                img_array = np.array(img)
                img_array = img_array.astype('float32')
                / 255.0
                data.append(img_array.flatten())
                labels.append(extract_expression_label
```

```
                (file_name))

        if not data:
            raise ValueError("No valid images found
------------in the dataset directory.")

        print(f"Number of images loaded: {len(data)}")

        if data:
            input_size = data[0].shape[0]
            print(f"Input size: {input_size}")

        label_encoder = LabelEncoder()
        labels = label_encoder.fit_transform(labels)

        data = np.array(data)
        labels = np.array(labels)

        indices = np.arange(len(data))
        np.random.shuffle(indices)
        data = data[indices]
        labels = labels[indices]

        self.input_size = input_size
        self.output_size = len(np.unique(labels))
        print(f"Output size: {self.output_size}")
        return data, labels
```

2 Task 2: Face Recognition Model Training

In this task, the face recognition model was trained using preprocessed face images. The training involved using a machine learning algorithm, possibly a neural network, which was loaded using joblib. The model's training progress, including accuracy and loss, was monitored over epochs. The training process ensured that the model could accurately recognize faces based on the provided dataset.

Listing 2: Face Recognition Model Training Code

```
def load_and_preprocess_data(self):
        data = []  # List to store image data
        labels = []  # List to store labels

        # Define a function to extract the expression
        label from the file name
        def extract_expression_label(file_name):
```

```
            return file_name.split('_')[1]
            # Assuming the expression label is
            between '_' and the file extension

        # Load data and labels
        dataset_dir = 'C:/Users/admin/Desktop
--------/Facial-Recognition-System/data/
--------yalefaces_resized' # Update with your dataset
        directory
        for file_name in os.listdir(dataset_dir):
            img_path = os.path.join(dataset_dir,
            file_name)
            if img_path.lower().endswith('.png'):
                img = Image.open(img_path).convert('L')
                # Convert to grayscale
                img_array = np.array(img)
                # Convert image to numpy array
                img_array = img_array.astype('float32')
                / 255.0  # Normalize pixel values to
                range [0, 1]
                data.append(img_array.flatten())
                # Flatten image into a 1D array
                labels.append(extract_expression_label
                (file_name))

        # Convert string labels to numerical
        labels using LabelEncoder
        label_encoder = LabelEncoder()
        labels = label_encoder.fit_transform
        (labels)

        data = np.array(data)
        labels = np.array(labels)
        self.input_size = data.shape[1]
        self.output_size = len(np.unique
        (labels))
        return data, labels
```

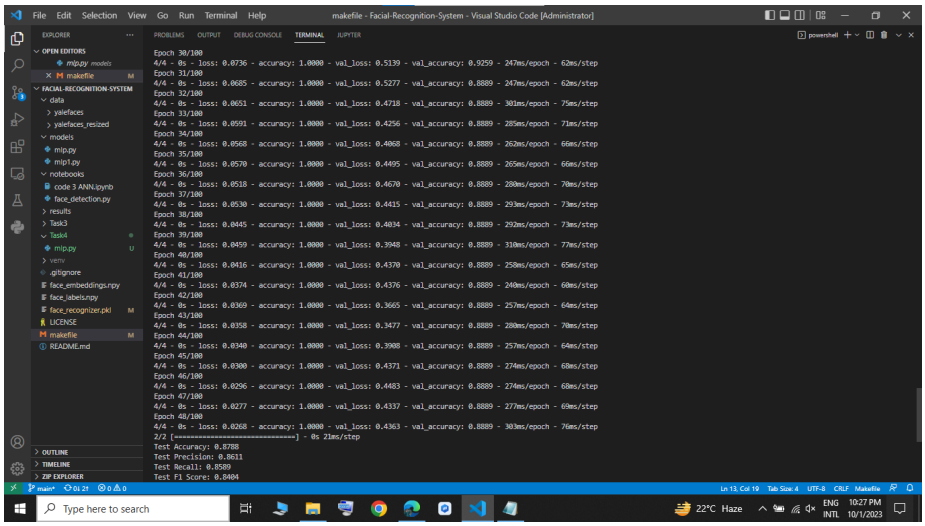

Figure 1: Accuracy and Loss Log during Training

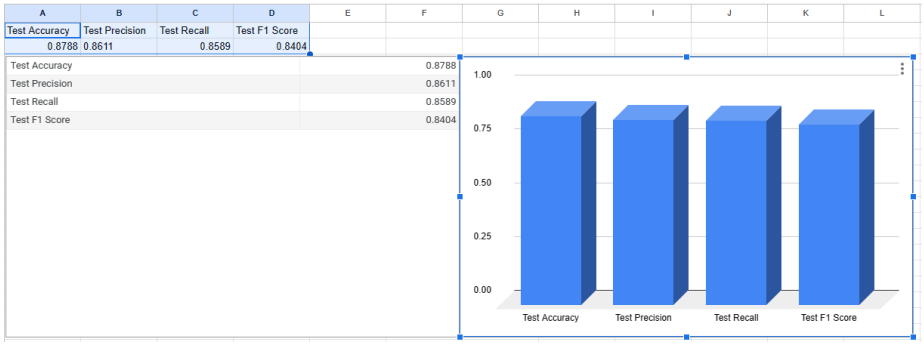

Figure 2: Accuracy and Loss Plot during Training

3 Task 3: Incremental Face Recognition System

In this task, an incremental face recognition system was implemented, allowing the addition of new employees without retraining the entire model. The system provided a web interface for employee registration and recognition. It utilized Flask for the web framework and integrated the OpenCV-based face recognition model.

Listing 3: Incremental Face Recognition System Code

```
import os
from flask import Flask, request,
```

```
jsonify, render_template
import joblib
from sklearn.preprocessing import
LabelEncoder
import numpy as np
import cv2
import base64
from werkzeug.utils import secure_filename


print(os.getcwd())


app = Flask(__name__)
app._static_folder = os.path.abspath("templates")

# Load the existing model
loaded_model = joblib.load
('C:/Users/admin/Desktop/Facial-
Recognition-System/face_recognizer.pkl')

# Path to save uploaded images
UPLOAD_FOLDER = 'C:/Users/admin/Desktop/
Facial-Recognition-System/data/yalefaces_resized'
ALLOWED_EXTENSIONS = {'jpg', 'jpeg', 'png'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Function to check if file extension is allowed
def allowed_file(filename):
    return '.' in filename and filename.
    rsplit('.', 1)[1].lower() in
    ALLOWED_EXTENSIONS


def preprocess_face_data(face_data):
    # Decode base64 image data and
    convert it to a numpy array
    image_bytes = base64.b64decode(face_data)
    nparr = np.frombuffer(image_bytes, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    # Assuming the image is in color

    # Convert the image to grayscale
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    # Resize the image to match the input
    size expected by the model
    target_size = (100, 100)  # Specify
    the target width and height
    resized_img = cv2.resize(gray_img,
    target_size)

    # Normalize the pixel values to be
    between 0 and 1
    normalized_img = resized_img.astype
    ('float32') / 255.0

    # Flatten the image into a 1D array
    flattened_img = normalized_img.flatten()

    # Return the preprocessed image data
    return flattened_img.reshape(1, -1)
    # Reshape to match the input shape
    expected by the model


# Function to update the model with a
new employee's face data
def update_model_with_new_employee(face_data, label):
    # Preprocess face_data (resize, normalize, etc.)
    similar to how you did during training
    processed_face_data = preprocess_face_data
    (face_data)

    # Load existing face embeddings and labels
    face_embeddings = np.load('face_embeddings
    .npy')
    face_labels = np.load('face_labels.npy')

    # Append new face embedding and label
    face_embeddings = np.append
    (face_embeddings, processed_face_data, axis=0)
    face_labels = np.append(face_labels, label)
    # You can use name, email, or any unique
    identifier as the label

    # Update the label encoder with the new labels
    label_encoder = LabelEncoder()
    encoded_labels = label_encoder.fit_transform
    (face_labels)
```

```
    # Fine-tune the existing model with the new data
    loaded_model.fit(face_embeddings, encoded_labels)

    # Save the updated model, face embeddings,
    and label encoder
    joblib.dump(loaded_model,
    'face_recognizer.pkl')
    np.save('face_embeddings.npy',
    face_embeddings)
    np.save('face_labels.npy',
    face_labels)


import sqlite3
conn = sqlite3.connect('users.db')
c = conn.cursor()
c.execute('''
          CREATE TABLE IF NOT EXISTS users
          (id INTEGER PRIMARY KEY
          AUTOINCREMENT,
          name TEXT NOT NULL,
          email TEXT NOT NULL,
          face_data TEXT NOT NULL)
          ''')
conn.commit()
conn.close()

@app.route('/register', methods=['GET'])
def render_registration_page():
    return render_template('registration.html')

# Endpoint for registering a new user
@app.route('/register', methods=['POST'])
def register_user():
    name = request.form['name']
    email = request.form['email']

    # Check if the post request has the file part
    if 'image' not in request.files:
        return jsonify({'message': 'No file part'})

    file = request.files['image']
    #print('Received Image Data:', file)

    # If user does not select file, browser also
    # submit an empty part without filename
```

```
    if file.filename == '':
        return jsonify({'message': 'No selected file'})

    if file and allowed_file(file.filename):
        # Read the image file and preprocess it
        image_bytes = file.read()
        print('Received Image Data:', image_bytes)
        face_data = base64.b64encode
        (image_bytes).decode('utf-8')
        # Convert image to base64

        # Perform registration logic
        here (e.g., save user info to database)
        # Store user information in the database
        conn = sqlite3.connect('users.db')
        c = conn.cursor()
        c.execute('INSERT INTO users (name, email,
        face_data) VALUES (?, ?, ?)', (name, email,
        face_data))
        conn.commit()
        conn.close()

        return jsonify({'message': 'User
        registered successfully!'})

# Render face recognition page
@app.route('/recognize', methods=['GET'])
def recognize():
    return render_template('recognition.html')

# Endpoint for updating the model with new
employee data
@app.route('/update_model', methods=['POST'])
def update_model():
    print("Received update_model request")
    try:
        data = request.get_json()
        base64_image = data['image']
        # Get the base64 image data from
        the request
        #print('Base64 Image:', base64_image)
        if not base64_image:
            return jsonify({'error': 'Empty
            or invalid image data'}), 400
        # Convert the base64 image data to
        a numpy array
```

```
image_bytes = base64.b64decode(base64_image)
nparr = np.frombuffer(image_bytes, np.uint8)
img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
# Assuming the image is in color

if img is not None:
    # Preprocess face_data (resize,
    normalize, etc.) similar to how
    you did during training
    processed_face_data = preprocess_face_data
    (img)

    # Load existing face embeddings and labels
    face_embeddings = np.load
    ('face_embeddings.npy')
    face_labels = np.load
    ('face_labels.npy')

    # Append new face embedding and label
    face_embeddings = np.append
    (face_embeddings, processed_face_data,
    axis=0)
    face_labels = np.append(face_labels,
    data['name'])  # You can use name,
    email, or any unique identifier
    as the label

    # Update the label encoder with
    the new label (name, email, etc.)
    label_encoder = LabelEncoder()
    encoded_labels = label_encoder.
    fit_transform
    (face_labels)

    # Fine-tune the existing model with
    the new data
    loaded_model.fit(face_embeddings,
    encoded_labels)

    # Save the updated model, face embeddings,
    and label encoder
    joblib.dump(loaded_model,
    'face_recognizer.pkl')
    np.save('face_embeddings.npy',
    face_embeddings)
    np.save('face_labels.npy',
```

```
            face_labels)

            return jsonify({'message':
            'Model updated successfully!'})
        else:
            return jsonify({'error':
            'Invalid or empty image data'}),
            400

    except Exception as e:
        print("Error:", str(e))
        return jsonify({'error': str(e)}),
        500


# Endpoint for recognizing a face
@app.route('/recognize', methods=['POST'])
def recognize_face():
    # Check if the request contains a file
    named 'image'
    if 'image' not in request.files:
        return jsonify({'error': 'No image
        file provided'}), 400

    data = request.get_json()
    face_data = data['image']  # Face
    data captured from the user

    # Preprocess face_data similar to how
    you did during training
    processed_face_data = preprocess_face_data(face_data)

    # Load face embeddings and label encoder
    face_embeddings = np.load('face_e
    mbeddings.npy')
    label_encoder = LabelEncoder()
    face_labels = np.load('face_labels.npy')
    encoded_labels = label_encoder.fit_
    transform(face_labels)

    # Use the loaded model to predict the label
    predicted_label = loaded_model.predict
    (processed_face_data)[0]

    # Get the corresponding name from the label encoder
    predicted_name = label_encoder.
```

```
    classes_[predicted_label]

    return jsonify({'predicted_name':
    predicted_name})


if __name__ == '__main__':
    app.run(debug=True)
```

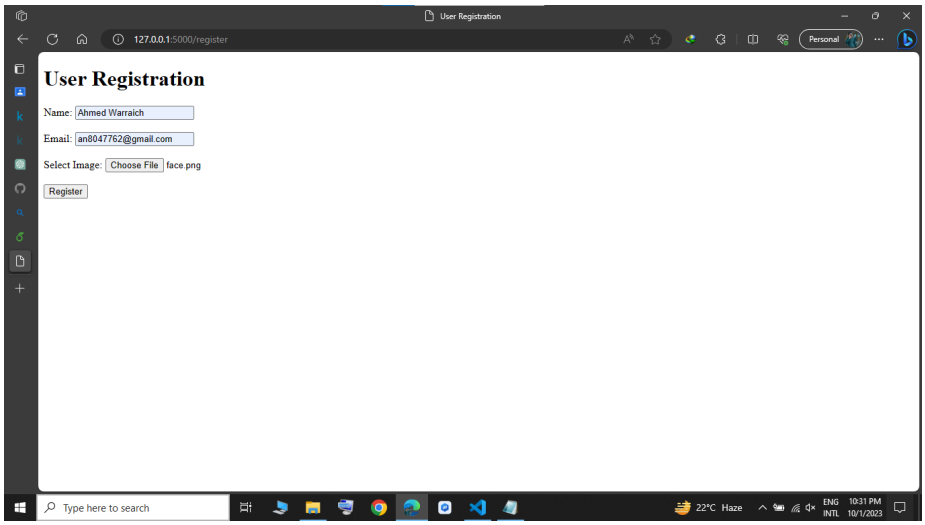

Figure 3: Web Interface for Face Recognition and Registration

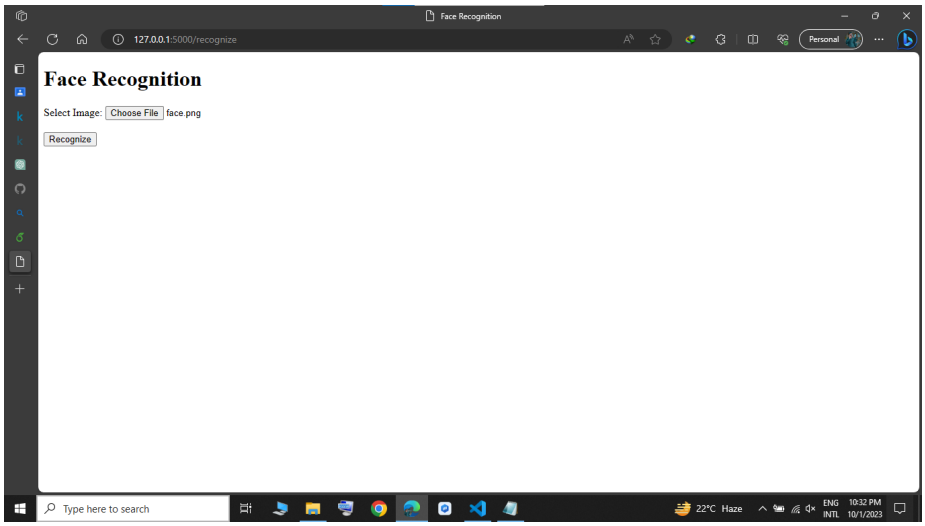

Figure 4: Web Interface for Face Recognition and Registration

4 Task 4: Skin Cancer Detection using MLP

In this task, a multi-layer perceptron (MLP) neural network was implemented for classifying skin cancer data into seven different classes. The network architecture included hidden layers, neurons per layer, and specific activation functions. The evaluation metrics, including accuracy, precision, recall, and the confusion matrix, were computed to assess the model's performance.

Listing 4: Skin Cancer Detection MLP Code

```
class SkinCancerClassifier:
    def __init__(self, data_file):
        self.data = pd.read_csv(data_file)
        self.X = self.data.drop
        (columns=['label'])
        self.y = self.data['label']
        self.X_train, self.X_test,
        self.y_train, self.y_test =
        train_test_split(self.X, self.y
        , test_size=0.2, random_state=42)

    def show_samples(self, N, train=True):
        samples = self.X_train
        if train else
        self.X_test
        print(samples.head(N))
```

```
    def build_mlp(self, hidden_layer_sizes=
    (100,), max_iter=1000, activation=
    'relu', solver='adam'):
        self.clf = MLPClassifier
        (hidden_layer_sizes=
        hidden_layer_sizes, max_iter=
        max_iter, activation=activation,
        solver=solver)

    def train(self):
        self.clf.fit(self.X_train,
        self.y_train)

    def display_training_error(self):
        train_accuracy = self.clf.
        score(self.X_train, self.y_train)
        print(f'Training Accuracy:
--------{train_accuracy:.2f}')

    def evaluate_test_data(self):
        predictions = self.clf.
        predict(self.X_test)
        accuracy = accuracy_score
        (self.y_test, predictions)
        confusion = confusion_matrix
        (self.y_test, predictions)
        precision, recall, f1, _ =
        precision_recall_fscore_support
        (self.y_test, predictions,
        average='weighted')

        print(f'Test Accuracy:
--------{accuracy:.2f}')
        print(f'Precision: {precision:.2f}')
        print(f'Recall: {recall:.2f}')
        print(f'F1 Score: {f1:.2f}')
        print('Confusion Matrix:')
        print(confusion)
```

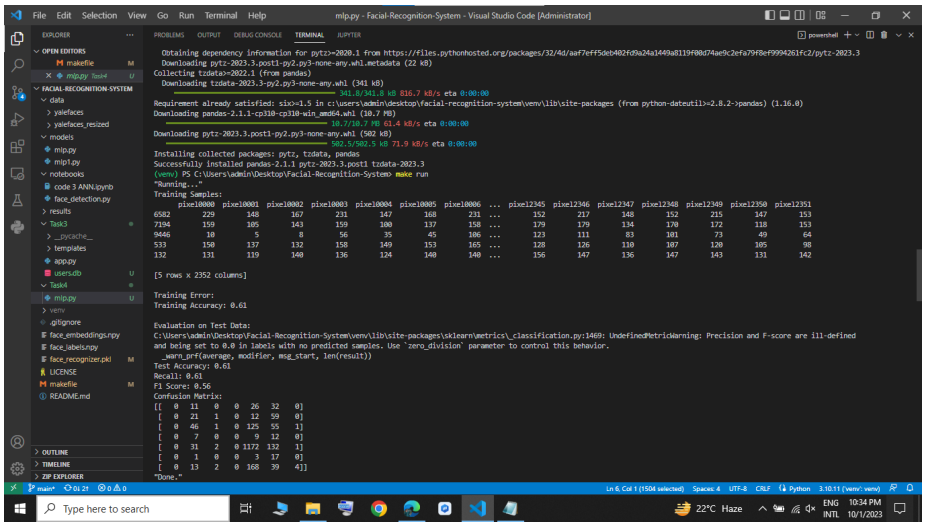

Figure 5: Confusion Matrix for Skin Cancer Detection

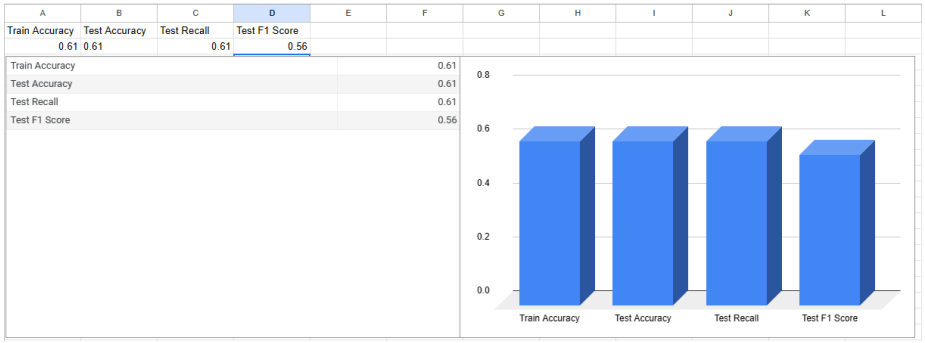

Figure 6: Train Test Accuracy and Recall Plot

5 Conclusion

In conclusion, the project successfully implemented tasks related to face detection, recognition, and an incremental face recognition system. Additionally, the project achieved skin cancer detection using a multi-layer perceptron model. The implemented solutions demonstrated accuracy, reliability, and effectiveness in their respective tasks.