AI Face Recognition System

1. Basic Face Recognition System using Face Recognition Library

```
# requirements.txt
"""
face-recognition==1.3.0
opencv-python==4.8.1.78
numpy==1.24.3
pillow==10.0.0
"""

import face_recognition
import cv2
import numpy as np
import os
from datetime import datetime
import pickle

class FaceRecognitionSystem:
    def __init__(self):
        self.known_face_encodings = []
        self.known_face_names = []
        self.load_known_faces()

    def load_known_faces(self, faces_dir="known_faces"):
```

```
"""Load known faces from directory"""
if not os.path.exists(faces_dir):
    os.makedirs(faces_dir)
    print(f"Created {faces_dir} directory. Add face images here.")
    return

# Try to load saved encodings
if os.path.exists("face_encodings.pkl"):
    with open("face_encodings.pkl", "rb") as f:
        data = pickle.load(f)
        self.known_face_encodings = data["encodings"]
        self.known_face_names = data["names"]
    print(f"Loaded {len(self.known_face_names)} known faces from file")
    return

# Load faces from images
for filename in os.listdir(faces_dir):
    if filename.endswith((".jpg", ".jpeg", ".png")):
        image_path = os.path.join(faces_dir, filename)
        image = face_recognition.load_image_file(image_path)

        # Find face encodings
        face_encodings = face_recognition.face_encodings(image)

        if len(face_encodings) > 0:
            self.known_face_encodings.append(face_encodings[0])
            name = os.path.splitext(filename)[0]
            self.known_face_names.append(name)
            print(f"Loaded {name}")
```

```
    # Save encodings for future use
    if self.known_face_encodings:
        self.save_encodings()

def save_encodings(self):
    """Save face encodings to file"""
    data = {
        "encodings": self.known_face_encodings,
        "names": self.known_face_names
    }
    with open("face_encodings.pkl", "wb") as f:
        pickle.dump(data, f)

def add_new_face(self, image_path, name):
    """Add a new face to the system"""
    image = face_recognition.load_image_file(image_path)
    face_encodings = face_recognition.face_encodings(image)

    if len(face_encodings) > 0:
        self.known_face_encodings.append(face_encodings[0])
        self.known_face_names.append(name)
        self.save_encodings()

        # Also save the original image
        faces_dir = "known_faces"
        if not os.path.exists(faces_dir):
            os.makedirs(faces_dir)
```

```
        # Save with timestamp
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"{name}_{timestamp}.jpg"
        cv2.imwrite(os.path.join(faces_dir, filename),
                    cv2.cvtColor(image, cv2.COLOR_RGB2BGR))

        print(f"Added {name} to known faces")
        return True
    return False

def recognize_faces_in_image(self, image_path):
    """Recognize faces in an image file"""
    # Load image
    image = face_recognition.load_image_file(image_path)

    # Find all face locations and encodings
    face_locations = face_recognition.face_locations(image)
    face_encodings = face_recognition.face_encodings(image, face_locations)

    results = []

    for face_encoding, face_location in zip(face_encodings, face_locations):
        # Compare with known faces
        matches = face_recognition.compare_faces(
            self.known_face_encodings,
            face_encoding,
            tolerance=0.6
        )
```

```
        name = "Unknown"
        confidence = 0

        if True in matches:
            # Find distances to all known faces
            face_distances = face_recognition.face_distance(
                self.known_face_encodings,
                face_encoding
            )
            best_match_index = np.argmin(face_distances)

            if matches[best_match_index]:
                name = self.known_face_names[best_match_index]
                confidence = 1 - face_distances[best_match_index]

        results.append({
            "name": name,
            "confidence": float(confidence),
            "location": face_location  # (top, right, bottom, left)
        })

    return results

def real_time_recognition(self):
    """Real-time face recognition from webcam"""
    video_capture = cv2.VideoCapture(0)

    # Set camera resolution
    video_capture.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
```

```
video_capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

print("Starting real-time recognition. Press 'q' to quit.")

process_frame = True
attendance_log = set()  # Track recognized people in this session

while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()

    if not ret:
        break

    # Process every other frame to improve performance
    if process_frame:
        # Convert BGR (OpenCV) to RGB (face_recognition)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

        # Find faces
        face_locations = face_recognition.face_locations(rgb_frame)
        face_encodings = face_recognition.face_encodings(
            rgb_frame,
            face_locations
        )

        for face_encoding, (top, right, bottom, left) in zip(
            face_encodings, face_locations
        ):
```

```
matches = face_recognition.compare_faces(
    self.known_face_encodings,
    face_encoding
)

name = "Unknown"
color = (0, 0, 255)  # Red for unknown

if True in matches:
    face_distances = face_recognition.face_distance(
        self.known_face_encodings,
        face_encoding
    )
    best_match_index = np.argmin(face_distances)

    if matches[best_match_index]:
        name = self.known_face_names[best_match_index]
        color = (0, 255, 0)  # Green for known
        confidence = 1 - face_distances[best_match_index]

        # Log attendance
        if name not in attendance_log:
            attendance_log.add(name)
            self.log_attendance(name)

# Draw rectangle around face
cv2.rectangle(frame, (left, top), (right, bottom), color, 2)

# Draw label
```

```
            cv2.rectangle(
                frame,
                (left, bottom - 35),
                (right, bottom),
                color,
                cv2.FILLED
            )
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(
                frame,
                name,
                (left + 6, bottom - 6),
                font,
                0.5,
                (255, 255, 255),
                1
            )

        process_frame = not process_frame

        # Display the resulting frame
        cv2.imshow('Face Recognition System', frame)

        # Break loop on 'q' press
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release resources
    video_capture.release()
```

```
        cv2.destroyAllWindows()

    def log_attendance(self, name):
        """Log attendance to file"""
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        with open("attendance.csv", "a") as f:
            f.write(f"{name},{timestamp}\n")
        print(f"Logged attendance for {name} at {timestamp}")

# Example usage
if __name__ == "__main__":
    # Initialize the system
    face_system = FaceRecognitionSystem()

    # Menu for different operations
    while True:
        print("\n=== Face Recognition System ===")
        print("1. Real-time recognition from webcam")
        print("2. Recognize faces in an image")
        print("3. Add new face")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            face_system.real_time_recognition()

        elif choice == "2":
            image_path = input("Enter image path: ")
```

```
    if os.path.exists(image_path):
        results = face_system.recognize_faces_in_image(image_path)
        print(f"\nFound {len(results)} face(s):")
        for i, result in enumerate(results, 1):
            print(f"{i}. {result['name']} (Confidence: {result['confidence']:.2%})")
    else:
        print("Image not found!")

elif choice == "3":
    image_path = input("Enter image path: ")
    name = input("Enter person's name: ")
    if os.path.exists(image_path):
        success = face_system.add_new_face(image_path, name)
        if success:
            print(f"Successfully added {name}")
        else:
            print("No face detected in the image!")
    else:
        print("Image not found!")

elif choice == "4":
    print("Exiting...")
    break

else:
    print("Invalid choice!")
```

2. Alternative Using DeepFace (More Advanced)

```
# requirements.txt for DeepFace version
"""
deepface==0.0.89
opencv-python==4.8.1.78
pandas==2.0.3
tensorflow==2.13.0  # or tensorflow-cpu
"""

from deepface import DeepFace
import cv2
import pandas as pd
from datetime import datetime
import os

class DeepFaceRecognition:
    def __init__(self, db_path="face_database"):
        self.db_path = db_path
        if not os.path.exists(db_path):
            os.makedirs(db_path)

    def register_face(self, image_path, person_name):
        """Register a new face in the database"""
        # Create person's directory
        person_dir = os.path.join(self.db_path, person_name)
        if not os.path.exists(person_dir):
            os.makedirs(person_dir)
```

```
    # Save the image
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"{timestamp}.jpg"
    save_path = os.path.join(person_dir, filename)

    # Read and save image
    img = cv2.imread(image_path)
    if img is not None:
        cv2.imwrite(save_path, img)
        print(f"Registered {person_name} successfully!")
        return True
    return False

def verify_face(self, image_path, person_name):
    """Verify if the face matches a registered person"""
    person_dir = os.path.join(self.db_path, person_name)

    if not os.path.exists(person_dir):
        print(f"No registered faces for {person_name}")
        return False

    # Get all images for this person
    reference_images = [
        os.path.join(person_dir, img)
        for img in os.listdir(person_dir)
        if img.endswith((".jpg", ".png", ".jpeg"))
    ]

    if not reference_images:
```

```
            print(f"No reference images for {person_name}")
            return False

        try:
            # Verify against first reference image
            result = DeepFace.verify(
                img1_path=image_path,
                img2_path=reference_images[0],
                model_name="Facenet",
                detector_backend="opencv"
            )

            if result["verified"]:
                print(f"✅ Face verified as {person_name}")
                print(f"Distance: {result['distance']}")
                return True
            else:
                print(f"❌ Face does not match {person_name}")
                return False

        except Exception as e:
            print(f"Error during verification: {e}")
            return False

    def identify_face(self, image_path):
        """Identify who the face belongs to"""
        try:
            dfs = DeepFace.find(
                img_path=image_path,
```

```
            db_path=self.db_path,
            model_name="Facenet",
            detector_backend="opencv",
            enforce_detection=False,
            silent=True
        )

        if dfs and len(dfs) > 0:
            df = dfs[0]
            if len(df) > 0:
                # Get best match
                identity = df.iloc[0]["identity"]
                distance = df.iloc[0]["distance"]

                # Extract person name from path
                person_name = os.path.basename(os.path.dirname(identity))

                # Convert distance to confidence (lower distance = higher confidence)
                confidence = max(0, 1 - (distance / 4))  # Normalize

                return {
                    "name": person_name,
                    "confidence": confidence,
                    "distance": distance
                }

        return {"name": "Unknown", "confidence": 0, "distance": None}

    except Exception as e:
```

```
        print(f"Error during identification: {e}")
        return {"name": "Unknown", "confidence": 0, "distance": None}

    def real_time_identification(self):
        """Real-time face identification"""
        cap = cv2.VideoCapture(0)
        print("Starting real-time identification. Press 'q' to quit.")

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            # Save temporary image
            temp_path = "temp_frame.jpg"
            cv2.imwrite(temp_path, frame)

            # Identify faces
            result = self.identify_face(temp_path)

            # Display result on frame
            label = f"{result['name']} ({result['confidence']:.2%})"
            cv2.putText(
                frame,
                label,
                (20, 40),
                cv2.FONT_HERSHEY_SIMPLEX,
                1,
                (0, 255, 0) if result['name'] != "Unknown" else (0, 0, 255),
```

```
            2
        )

        cv2.imshow("DeepFace Recognition", frame)

        # Remove temp file
        if os.path.exists(temp_path):
            os.remove(temp_path)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

# Example usage for DeepFace
if __name__ == "__main__":
    recognizer = DeepFaceRecognition()

    while True:
        print("\n=== DeepFace Recognition System ===")
        print("1. Register new face")
        print("2. Verify face")
        print("3. Identify face from image")
        print("4. Real-time identification")
        print("5. Exit")

        choice = input("Enter choice: ")
```

```
    if choice == "1":
        img_path = input("Enter image path: ")
        name = input("Enter person name: ")
        recognizer.register_face(img_path, name)

    elif choice == "2":
        img_path = input("Enter image path: ")
        name = input("Enter person name to verify: ")
        recognizer.verify_face(img_path, name)

    elif choice == "3":
        img_path = input("Enter image path: ")
        result = recognizer.identify_face(img_path)
        print(f"Result: {result['name']} (Confidence: {result['confidence']:.2%})")

    elif choice == "4":
        recognizer.real_time_identification()

    elif choice == "5":
        break
```

3. Simple Web Interface

```
# app.py - Flask Web Application
"""
pip install flask flask-cors
"""
```

```
from flask import Flask, render_template, request, jsonify, Response
import cv2
import numpy as np
import base64
from face_recognition_system import FaceRecognitionSystem
import os

app = Flask(__name__)
face_system = FaceRecognitionSystem()

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/recognize', methods=['POST'])
def recognize():
    try:
        # Get image from request
        image_data = request.json['image'].split(',')[1]
        image_bytes = base64.b64decode(image_data)
        nparr = np.frombuffer(image_bytes, np.uint8)
        img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        # Save temp image
        temp_path = "temp_web.jpg"
        cv2.imwrite(temp_path, img)

        # Recognize faces
        results = face_system.recognize_faces_in_image(temp_path)
```

```
        # Clean up
        if os.path.exists(temp_path):
            os.remove(temp_path)

        return jsonify({
            "success": True,
            "faces": results
        })

    except Exception as e:
        return jsonify({
            "success": False,
            "error": str(e)
        })

@app.route('/add_face', methods=['POST'])
def add_face():
    try:
        name = request.form['name']
        image_file = request.files['image']

        # Save image
        temp_path = f"temp_add_{name}.jpg"
        image_file.save(temp_path)

        # Add to system
        success = face_system.add_new_face(temp_path, name)
```

```
        # Clean up
        if os.path.exists(temp_path):
            os.remove(temp_path)

        return jsonify({
            "success": success,
            "message": f"Face added successfully for {name}" if success else "Failed to add face"
        })

    except Exception as e:
        return jsonify({
            "success": False,
            "error": str(e)
        })

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```