import numpy as np

import cv2

import pandas as pd

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.pipeline import Pipeline

import dlib

import librosa

import warnings

warnings.filterwarnings('ignore')

class InfantEmotionDetector:

def \_\_init\_\_(self):

"""

Initialize the Infant Emotion Detection System using SVM

"""

self.face\_detector = dlib.get\_frontal\_face\_detector()

self.landmark\_predictor = None # Load with dlib.shape\_predictor('shape\_predictor\_68\_face\_landmarks.dat')

self.scaler = StandardScaler()

self.label\_encoder = LabelEncoder()

self.svm\_model = None

self.emotion\_labels = ['calm', 'happy', 'sad', 'angry', 'surprised', 'fear']

def extract\_facial\_features(self, image):

"""

Extract facial landmark features from infant face

"""

try:

# Convert to grayscale

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Detect faces

faces = self.face\_detector(gray)

if len(faces) == 0:

return None

# Get the largest face (assuming it's the infant)

face = max(faces, key=lambda rect: rect.width() \* rect.height())

# Extract facial landmarks (requires dlib predictor model)

if self.landmark\_predictor is not None:

landmarks = self.landmark\_predictor(gray, face)

# Convert landmarks to numpy array

points = np.array([[p.x, p.y] for p in landmarks.parts()])

# Calculate geometric features

features = self.\_calculate\_geometric\_features(points)

else:

# Basic geometric features from face rectangle

features = self.\_basic\_face\_features(face, gray.shape)

return features

except Exception as e:

print(f"Error in facial feature extraction: {e}")

return None

def \_calculate\_geometric\_features(self, landmarks):

"""

Calculate geometric features from facial landmarks

"""

features = []

# Eye aspect ratio (EAR) - indicates eye openness

left\_eye = landmarks[36:42]

right\_eye = landmarks[42:48]

left\_ear = self.\_eye\_aspect\_ratio(left\_eye)

right\_ear = self.\_eye\_aspect\_ratio(right\_eye)

features.extend([left\_ear, right\_ear, (left\_ear + right\_ear) / 2])

# Mouth aspect ratio (MAR) - indicates mouth openness

mouth = landmarks[48:68]

mar = self.\_mouth\_aspect\_ratio(mouth)

features.append(mar)

# Eyebrow position relative to eyes

left\_eyebrow = landmarks[17:22]

right\_eyebrow = landmarks[22:27]

left\_brow\_height = np.mean(left\_eyebrow[:, 1]) - np.mean(left\_eye[:, 1])

right\_brow\_height = np.mean(right\_eyebrow[:, 1]) - np.mean(right\_eye[:, 1])

features.extend([left\_brow\_height, right\_brow\_height])

# Mouth curvature (smile detection)

mouth\_corners = [landmarks[48], landmarks[54]] # Left and right mouth corners

mouth\_center = landmarks[51] # Upper lip center

curvature = (mouth\_corners[0][1] + mouth\_corners[1][1]) / 2 - mouth\_center[1]

features.append(curvature)

# Face symmetry

face\_center\_x = np.mean(landmarks[:, 0])

left\_side = landmarks[landmarks[:, 0] < face\_center\_x]

right\_side = landmarks[landmarks[:, 0] >= face\_center\_x]

symmetry = np.abs(np.mean(left\_side[:, 0]) - np.mean(right\_side[:, 0]))

features.append(symmetry)

return np.array(features)

def \_basic\_face\_features(self, face\_rect, image\_shape):

"""

Extract basic features when landmark detection is not available

"""

features = []

# Face dimensions relative to image

face\_width = face\_rect.width() / image\_shape[1]

face\_height = face\_rect.height() / image\_shape[0]

face\_area = face\_width \* face\_height

# Face position

face\_center\_x = (face\_rect.left() + face\_rect.width()/2) / image\_shape[1]

face\_center\_y = (face\_rect.top() + face\_rect.height()/2) / image\_shape[0]

features.extend([face\_width, face\_height, face\_area, face\_center\_x, face\_center\_y])

return np.array(features)

def \_eye\_aspect\_ratio(self, eye\_points):

"""

Calculate eye aspect ratio

"""

# Vertical eye distances

A = np.linalg.norm(eye\_points[1] - eye\_points[5])

B = np.linalg.norm(eye\_points[2] - eye\_points[4])

# Horizontal eye distance

C = np.linalg.norm(eye\_points[0] - eye\_points[3])

# Eye aspect ratio

ear = (A + B) / (2.0 \* C)

return ear

def \_mouth\_aspect\_ratio(self, mouth\_points):

"""

Calculate mouth aspect ratio

"""

# Vertical mouth distances

A = np.linalg.norm(mouth\_points[2] - mouth\_points[10]) # 50-58

B = np.linalg.norm(mouth\_points[4] - mouth\_points[8]) # 52-56

# Horizontal mouth distance

C = np.linalg.norm(mouth\_points[0] - mouth\_points[6]) # 48-54

# Mouth aspect ratio

mar = (A + B) / (2.0 \* C)

return mar

def extract\_audio\_features(self, audio\_file):

"""

Extract audio features from infant vocalizations/crying

"""

try:

# Load audio file

y, sr = librosa.load(audio\_file, sr=22050)

# Extract audio features

features = []

# Spectral features

mfccs = librosa.feature.mfcc(y=y, sr=sr, n\_mfcc=13)

features.extend(np.mean(mfccs, axis=1))

features.extend(np.std(mfccs, axis=1))

# Pitch features

pitches, magnitudes = librosa.piptrack(y=y, sr=sr)

pitch\_mean = np.mean(pitches[pitches > 0]) if np.any(pitches > 0) else 0

pitch\_std = np.std(pitches[pitches > 0]) if np.any(pitches > 0) else 0

features.extend([pitch\_mean, pitch\_std])

# Energy features

rms = librosa.feature.rms(y=y)[0]

features.extend([np.mean(rms), np.std(rms)])

# Spectral centroid

spectral\_centroids = librosa.feature.spectral\_centroid(y=y, sr=sr)[0]

features.extend([np.mean(spectral\_centroids), np.std(spectral\_centroids)])

# Zero crossing rate

zcr = librosa.feature.zero\_crossing\_rate(y)[0]

features.extend([np.mean(zcr), np.std(zcr)])

return np.array(features)

except Exception as e:

print(f"Error in audio feature extraction: {e}")

return None

def prepare\_training\_data(self, image\_paths, audio\_paths, labels):

"""

Prepare training data from images, audio files, and labels

"""

features\_list = []

valid\_labels = []

for i, (img\_path, audio\_path, label) in enumerate(zip(image\_paths, audio\_paths, labels)):

try:

# Extract visual features

image = cv2.imread(img\_path)

if image is None:

continue

visual\_features = self.extract\_facial\_features(image)

if visual\_features is None:

continue

# Extract audio features (if available)

audio\_features = None

if audio\_path and audio\_path != '':

audio\_features = self.extract\_audio\_features(audio\_path)

# Combine features

if audio\_features is not None:

combined\_features = np.concatenate([visual\_features, audio\_features])

else:

combined\_features = visual\_features

features\_list.append(combined\_features)

valid\_labels.append(label)

except Exception as e:

print(f"Error processing sample {i}: {e}")

continue

return np.array(features\_list), np.array(valid\_labels)

def train\_model(self, X, y, test\_size=0.2):

"""

Train the SVM model for infant emotion detection

"""

# Encode labels

y\_encoded = self.label\_encoder.fit\_transform(y)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y\_encoded, test\_size=test\_size, random\_state=42, stratify=y\_encoded

)

# Create pipeline with scaling and SVM

pipeline = Pipeline([

('scaler', StandardScaler()),

('svm', SVC(random\_state=42))

])

# Hyperparameter tuning

param\_grid = {

'svm\_\_C': [0.1, 1, 10, 100],

'svm\_\_kernel': ['rbf', 'poly', 'linear'],

'svm\_\_gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1]

}

print("Performing hyperparameter tuning...")

grid\_search = GridSearchCV(

pipeline, param\_grid, cv=5, scoring='accuracy', n\_jobs=-1, verbose=1

)

grid\_search.fit(X\_train, y\_train)

# Best model

self.svm\_model = grid\_search.best\_estimator\_

# Evaluate model

y\_pred = self.svm\_model.predict(X\_test)

print(f"\nBest Parameters: {grid\_search.best\_params\_}")

print(f"Best Cross-validation Score: {grid\_search.best\_score\_:.4f}")

print(f"Test Accuracy: {accuracy\_score(y\_test, y\_pred):.4f}")

# Detailed evaluation

print("\nClassification Report:")

print(classification\_report(

y\_test, y\_pred,

target\_names=self.label\_encoder.classes\_

))

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

return self.svm\_model

def predict\_emotion(self, image, audio\_file=None):

"""

Predict emotion from a single image and optional audio file

"""

if self.svm\_model is None:

raise ValueError("Model not trained yet. Please train the model first.")

# Extract features

visual\_features = self.extract\_facial\_features(image)

if visual\_features is None:

return None, 0.0

# Extract audio features if available

audio\_features = None

if audio\_file:

audio\_features = self.extract\_audio\_features(audio\_file)

# Combine features

if audio\_features is not None:

features = np.concatenate([visual\_features, audio\_features])

else:

features = visual\_features

# Reshape for prediction

features = features.reshape(1, -1)

# Predict

prediction = self.svm\_model.predict(features)[0]

confidence = np.max(self.svm\_model.predict\_proba(features)[0])

# Decode label

emotion = self.label\_encoder.inverse\_transform([prediction])[0]

return emotion, confidence

def real\_time\_emotion\_detection(self, camera\_id=0):

"""

Real-time emotion detection using webcam

"""

if self.svm\_model is None:

raise ValueError("Model not trained yet. Please train the model first.")

cap = cv2.VideoCapture(camera\_id)

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

while True:

ret, frame = cap.read()

if not ret:

break

# Predict emotion

emotion, confidence = self.predict\_emotion(frame)

if emotion is not None:

# Display result on frame

text = f"Emotion: {emotion} ({confidence:.2f})"

cv2.putText(frame, text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX,

1, (0, 255, 0), 2)

# Show frame

cv2.imshow('Infant Emotion Detection', frame)

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

break

cap.release()

cv2.destroyAllWindows()

# Example usage

def main():

"""

Example usage of the Infant Emotion Detection System

"""

# Initialize detector

detector = InfantEmotionDetector()

# Example training data (replace with actual data paths)

image\_paths = ['infant1.jpg', 'infant2.jpg', 'infant3.jpg'] # Add actual paths

audio\_paths = ['cry1.wav', '', 'laugh1.wav'] # Optional audio files

labels = ['happy', 'sad', 'happy'] # Corresponding emotion labels

# Prepare training data

print("Preparing training data...")

# X, y = detector.prepare\_training\_data(image\_paths, audio\_paths, labels)

# For demonstration, create dummy data

X = np.random.random((100, 15)) # 100 samples, 15 features

y = np.random.choice(['calm', 'happy', 'sad', 'angry', ’fear’, ‘surprised’], 100)

# Train model

print("Training SVM model...")

detector.train\_model(X, y)

# Example prediction

# test\_image = cv2.imread('test\_infant.jpg')

# emotion, confidence = detector.predict\_emotion(test\_image)

# print(f"Predicted emotion: {emotion} with confidence: {confidence:.2f}")

print("Training completed successfully!")

if \_\_name\_\_ == "\_\_main\_\_":

main()