import cv2

import os

# Prompt the user to select an option: capture an image from the webcam or load an image from a file

option = input("Select one option:\n1. open webcam to capture\n2. Browse the image on your desktop\n")

if option == '1':

# If option 1 is selected, capture an image from the webcam

name = input("Enter a name: ")

# Create a directory to store the generated images, based on the user's name

directory = "image\_generated/{name}"

os.makedirs(directory, exist\_ok=True)

# Access the webcam

cap = cv2.VideoCapture(0)

# Load the face cascade classifier

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

# Counter for the captured images

count = 1

while True:

# Read a frame from the webcam

ret, frame = cap.read()

# Convert the frame to grayscale

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

# Detect faces in the grayscale frame

faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

# Draw rectangles around the detected faces in the original frame

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

# Display the frame with the rectangles

cv2.imshow("Capture", frame)

# If 's' key is pressed and at least one face is detected, save the image

if cv2.waitKey(1) & 0xFF == ord('s') and len(faces) > 0:

image\_path = f"{directory}/{count:02d}.jpg"

cv2.imwrite(image\_path, frame)

print("Saved image:", image\_path)

count += 1

# If 'q' key is pressed or the maximum number of images (100) is reached, exit the loop

if cv2.waitKey(1) & 0xFF == ord('q') or count > 100:

break

# Release the webcam and destroy the windows

cap.release()

cv2.destroyAllWindows()

elif option == '2':

# If option 2 is selected, load an image from a file

# Prompt the user to enter the image file path

file\_path = input("Enter the path for the image: ")

if os.path.isfile(file\_path):

# If the file path is valid, proceed with generating images

# Prompt the user to enter the folder name

name = input("Enter one folder name: ")

# Create a directory to store the generated images, based on the folder name

directory = f"image\_generated/{name}"

os.makedirs(directory, exist\_ok=True)

# Read the image from the file

image = cv2.imread(file\_path)

# Counter for the generated images

count = 1

while count <= 100:

# Generate image paths and save the image

image\_path = f"{directory}/{count:02d}.jpg"

cv2.imwrite(image\_path, image)

print("Image saved:", image\_path)

count += 1

else:

# If the file path is invalid, display an error message

print("File cannot be found!")

else:

# If an invalid option is selected, display an error message

print("Invalid option chosen!")

# Create a label file to store the labels of the generated images

label\_file = open("label.txt", "w")

# Get a list of folders in the "image\_generated" directory

folders = os.listdir("image\_generated")

# Write the labels (folder names) to the label file

for i, folder in enumerate(folders):

label = f"{i} {folder}\n"

label\_file.write(label)

# Close the label file

label\_file.close()

# Display a success message

print("Successfully generated label.txt!"

)

import cv2

import tensorflow as tf

from tensorflow import keras

import numpy as np

from sklearn.model\_selection import train\_test\_split

import os

data\_dir = "dataset\_images"

label\_file = "label.txt"

labels = []

# Read labels from label.txt file

with open(label\_file, "r") as file:

lines = file.readlines()

for line in lines:

label = line.strip().split(" ")[1]

labels.append(label)

data = []

labels\_encoded = []

# Load images and encode labels

for label\_idx, folder in enumerate(os.listdir(data\_dir)):

folder\_path = os.path.join(data\_dir, folder)

for filename in os.listdir(folder\_path):

img\_path = os.path.join(folder\_path, filename)

img = cv2.imread(img\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

img = cv2.resize(img, (224, 224))

img = img / 255.0

data.append(img)

labels\_encoded.append(label\_idx)

data = np.array(data)

labels\_encoded = np.array(labels\_encoded)

# Split data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels\_encoded, test\_size=0.2, random\_state=42)

model = keras.Sequential([

keras.applications.MobileNetV2(include\_top=False, input\_shape=(224, 224, 3)),

keras.layers.GlobalAveragePooling2D(),

keras.layers.Dense(len(labels), activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_data=(X\_test, y\_test))

# Save the trained model

model.save("facedetection.h5")

# Importing the required libraries

import cv2 # To utilize OpenCV for computer vision tasks, include the library in your project by importing it

import tensorflow as tf # To incorporate TensorFlow for deep learning tasks, import the library into your project

from tensorflow import keras # To facilitate the creation and training of neural networks, import the Keras module into project

import numpy as np # To perform numerical operations, import the NumPy library into your project

import datetime # To add timestamps to your code, import the datetime module into your project

attendance\_log = "attendance.txt" #Specify the file path for the attendance log

def get\_class\_name(class\_no):

"""

Function to map class numbers to class names.

:param class\_no: Integer representing the class number

:return: Corresponding class name as a string

"""

class\_names = ["alice", "pawan", "rupendra", "saugat", "sujan"]

if class\_no >= 0 and class\_no < len(class\_names):

return class\_names[class\_no]

else:

return "Unknown"

model = keras.models.load\_model('facedetection.h5') # Load the face detection model that has been trained

face\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml') # Load the face cascade classifier into the program

cap = cv2.VideoCapture(0) # Initialize and open the video capture device (webcam) for further usage.

cap.set(4, 480) # Adjust the height of the captured video to the desired value.

font = cv2.FONT\_HERSHEY\_COMPLEX #Specify the font to be used for displaying text on the image.

while True:

success, img\_original = cap.read() # Retrieve a frame from the video capture device.

gray = cv2.cvtColor(img\_original, cv2.COLOR\_BGR2GRAY) #Convert the frame to grayscale format.

faces = face\_cascade.detectMultiScale(gray, 1.3, 5) \

for (x, y, w, h) in faces:

cv2.rectangle(img\_original, (x, y), (x + w, y + h), (0, 255, 0), 2) #it Draws a square around the face when it is detected

crop\_img = img\_original[y:y + h, x:x + w]

img = cv2.resize(crop\_img, (224, 224))

img = img / 255.0

img = np.expand\_dims(img, axis=0)

prediction = model.predict(img)

class\_index = np.argmax(prediction)

class\_name = get\_class\_name(class\_index)

cv2.putText(img\_original, class\_name, (x, y + h + 20), font, 0.75, (255, 255, 255), 1, cv2.LINE\_AA) # Display the class name above the detected face

with open(attendance\_log, "a") as file:

timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Get the timestamp in current event

attendance\_entry = f"{timestamp} - {class\_name}\n" # this will create a attendance entery

file.write(attendance\_entry)

cv2.putText(img\_original, f"{class\_name} - Attendance Registered", (x, y - 10), font, 0.75, (0, 255, 0), 2, cv2.LINE\_AA)

cv2.imshow("Result", img\_original) #Display the processed frame, showcasing the results of face detection and recognition.

cv2.waitKey(1)

if len(faces) > 0:

cv2.waitKey(3000) # If at least one face is detected, wait for 3 seconds before terminating the program.

break

cap.release()

cv2.destroyAllWindows() # Closes windows