FACIAL EMOTION RECOGNITION SYSTEM

PROJECT CODE:

CREATE DATASET:

```
import numpy as np

import imutils

import time

import cv2

import os

import math


# system libraries

import os

import sys

from threading import Timer

import shutil

import time



def create_dataset_folders(dataset_path, labels):

      for label in labels:

      dataset_folder = dataset_path + "\\" + label

      if not os.path.exists(dataset_folder):

            os.makedirs(dataset_folder)
```

```
def detect_face(frame, faceNet, threshold=0.5):
      # grab the dimensions of the frame and then construct a blob
      # from it
      global detections
      (h, w) = frame.shape[:2]
      blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104.0,
177.0, 123.0))

      # pass the blob through the network and obtain the face
detections
      faceNet.setInput(blob)
      detections = faceNet.forward()

      # initialize our list of faces, their corresponding locations,
      # and the list of predictions from our face mask network
      locs = []
      # loop over the detections
      for i in range(0, detections.shape[2]):
      # extract the confidence (i.e., probability) associated with
      confidence = detections[0, 0, i, 2]

      # filter out weak detections by ensuring the confidence is
      # greater than the minimum confidence
      if confidence > threshold:
            # compute the (x, y)-coordinates of the bounding box for
```

```
            # the object
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # ensure the bounding boxes fall within the dimensions of
            # the frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # add the face and bounding boxes to their respective
            # lists
            locs.append((startX, startY, endX, endY))

      return (locs)



def capture_face_expression(face_expression, label, dataset_path):
      if len(face_expression) != 0:
      dataset_folder = dataset_path + "\\" + label
      number_files = len(os.listdir(dataset_folder))  # dir is your
directory path
      image_path = "%s\\%s_%d.jpg" % (dataset_folder, label,
number_files)
      cv2.imwrite(image_path, face_expression)
```

```
# define constant

dataset_path = os.getcwd() + "\\dataset"

face_model_path = os.getcwd() + "\\face_detector"

labels = ["neutral", "happy", "sad"]


# load our serialized face detector model from disk

print("[INFO] loading face detector model...")

prototxtPath = os.path.sep.join([face_model_path, "deploy.prototxt"])

weightsPath = os.path.sep.join([face_model_path,
"res10_300x300_ssd_iter_140000.caffemodel"])

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)


print("[INFO] Creating dataset folders...")

create_dataset_folders(dataset_path, labels)


cap = cv2.VideoCapture(0)


while (True):

      # Capture frame-by-frame

      ret, frame = cap.read()


      # Convert into gray scale

      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)


      # Detect faces
```

```
    locs = detect_face(frame, faceNet, threshold=0.5)

    face_expression = None

    for box in locs:

    # unpack the bounding box and predictions

    (startX, startY, endX, endY) = box

    face_expression = gray[startY:endY, startX:endX].copy()

    cv2.rectangle(gray, (startX, startY), (endX, endY), (255, 255,
255), 2)



    # show video stream

    cv2.putText(gray, "N - Neutral", (10, 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 2)

    cv2.putText(gray, "H - Happy", (10, 35),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 2)

    cv2.putText(gray, "S - Sad", (10, 55),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 2)

    cv2.putText(gray, "Q - Quit", (10, 75),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 2)

    cv2.imshow('frame', gray)



    # wait for key press

    key = cv2.waitKey(1)

    if key == ord('q'):

    break

    elif key == ord('n'):

       capture_face_expression(face_expression, "neutral",
dataset_path)
```

```
      print("[INFO] Neutral")

      elif key == ord('h'):

         capture_face_expression(face_expression, "happy",
dataset_path)

      print("[INFO] Happy")

      elif key == ord('s'):

         capture_face_expression(face_expression, "sad", dataset_path)

      print("[INFO] Sad")


# When everything done, release the capture

cap.release()

cv2.destroyAllWindows()
```

TRAIN EMOTION:

```
# import the necessary packages

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.layers import AveragePooling2D

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Input

from tensorflow.keras.models import Model
```

```
from tensorflow.keras.optimizers import Adam

from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.preprocessing.image import load_img

from tensorflow.keras.utils import to_categorical

from sklearn.preprocessing import LabelBinarizer

from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report

from imutils import paths

import matplotlib.pyplot as plt

import numpy as np

import argparse

import os


print(os.getcwd())

# define constant

dataset_path=os.getcwd()+"//dataset"

model_path=os.getcwd()+"//model//emotion_model"

plot_path=os.getcwd()+"//plot"



# initialize the initial learning rate, number of epochs to train
for,

# and batch size

INIT_LR = 1e-4
```

```
EPOCHS = 50

BS = 32


# grab the list of images in our dataset directory, then initialize

# the list of data (i.e., images) and class images

print("[INFO] loading images...")

imagePaths = list(paths.list_images(dataset_path))

imagePaths = [imagePath.replace("\\","//",-1) for imagePath in
imagePaths]

data = []

labels = []


# loop over the image paths

for imagePath in imagePaths:

      # extract the class label from the filename

      label = imagePath.split("//")[-2]


      # load the input image (224x224) and preprocess it

      image = load_img(imagePath, target_size=(224, 224))

      image = img_to_array(image)

      image = preprocess_input(image)


      # update the data and labels lists, respectively

      data.append(image)

      labels.append(label)
```

```
# convert the data and labels to NumPy arrays

data = np.array(data, dtype="float32")

labels = np.array(labels)


# perform one-hot encoding on the labels

lb = LabelBinarizer()

labels = lb.fit_transform(labels)


# partition the data into training and testing splits using 75% of

# the data for training and the remaining 25% for testing

(trainX, testX, trainY, testY) = train_test_split(data,
labels,test_size=0.20, stratify=labels, random_state=42)


# construct the training image generator for data augmentation

aug = ImageDataGenerator(

      zoom_range=0.15,

      width_shift_range=0.2,

      height_shift_range=0.2,

      shear_range=0.15,

      fill_mode="nearest")


# load the MobileNetV2 network, ensuring the head FC layer sets are

# left off

baseModel = MobileNetV2(weights="imagenet",
include_top=False,input_tensor=Input(shape=(224, 224, 3)))
```

```
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
# there is 3 types of expression: happy, neutral and sad
headModel = Dense(3, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)

# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
	layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
	metrics=["accuracy"])
```

```
# train the head of the network

print("[INFO] training head...")

H = model.fit(aug.flow(trainX, trainY,
batch_size=BS),steps_per_epoch=len(trainX) //
BS,validation_data=(testX, testY),validation_steps=len(testX) //
BS,epochs=EPOCHS)


# make predictions on the testing set

print("[INFO] evaluating network...")

predIdxs = model.predict(testX, batch_size=BS)


# for each image in the testing set we need to find the index of the

# label with corresponding largest predicted probability

predIdxs = np.argmax(predIdxs, axis=1)


# show a nicely formatted classification report

print(classification_report(testY.argmax(axis=1), predIdxs,

     target_names=lb.classes_))


# serialize the model to disk

#print("[INFO] saving mask detector model...")

model.save(model_path+".h5")


# plot the training loss and accuracy

N = EPOCHS
```

```
plt.style.use("ggplot")

plt.figure()

plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")

plt.plot(np.arange(0, N), H.history.get("val_loss"),
label="val_loss")

plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")

plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")

plt.title("Training Loss and Accuracy")

plt.xlabel("Epoch #")

plt.ylabel("Loss/Accuracy")

plt.legend(loc="lower left")

plt.savefig(plot_path)

print("Training Completed")
```

DETECT EMOTION:

```
# import the necessary packages

from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input

from tensorflow.keras.preprocessing.image import img_to_array

from tensorflow.keras.models import load_model

from imutils.video import VideoStream

# from pygame import mixer

import numpy as np

import imutils

import time
```

```
import cv2

import os

import math

from PIL import Image


# system libraries

import os

import sys

from threading import Timer

import shutil

import time


detections = None

count = 0




def detect_and_predict_mask(frame, faceNet, maskNet, threshold):

      # grab the dimensions of the frame and then construct a blob

      # from it

      global detections

      (h, w) = frame.shape[:2]

      blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104.0,
177.0, 123.0))


      # pass the blob through the network and obtain the face
detections
```

```
	faceNet.setInput(blob)
	detections = faceNet.forward()

	# initialize our list of faces, their corresponding locations,
	# and the list of predictions from our face mask network
	faces = []
	locs = []
	preds = []
	# loop over the detections
	for i in range(0, detections.shape[2]):
	# extract the confidence (i.e., probability) associated with
	confidence = detections[0, 0, i, 2]

	# filter out weak detections by ensuring the confidence is
	# greater than the minimum confidence
	if confidence > threshold:
		# compute the (x, y)-coordinates of the bounding box
for
		# the object
		box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
		(startX, startY, endX, endY) = box.astype("int")

		# ensure the bounding boxes fall within the dimensions of
		# the frame
		(startX, startY) = (max(0, startX), max(0, startY))
```

```
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # extract the face ROI, convert it from BGR to RGB channel
            # ordering, resize it to 224x224, and preprocess it
            face = frame[startY:endY, startX:endX]
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            face = cv2.resize(face, (224, 224))
            face = img_to_array(face)
            face = preprocess_input(face)
            face = np.expand_dims(face, axis=0)

            # add the face and bounding boxes to their respective
            # lists
            locs.append((startX, startY, endX, endY))
            # print(maskNet.predict(face)[0].tolist())
             preds.append(maskNet.predict(face)[0].tolist())
      return (locs, preds)



# SETTINGS
MASK_MODEL_PATH = os.getcwd() + "\\model\\emotion_model.h5"
FACE_MODEL_PATH = os.getcwd() + "\\face_detector"
SOUND_PATH = os.getcwd() + "\\sounds\\alarm.wav"
THRESHOLD = 0.5
```

```
# Load Sounds

# mixer.init()

# sound = mixer.Sound(SOUND_PATH)


# load our serialized face detector model from disk

print("[INFO] loading face detector model...")

prototxtPath = os.path.sep.join([FACE_MODEL_PATH, "deploy.prototxt"])

weightsPath = os.path.sep.join([FACE_MODEL_PATH,
"res10_300x300_ssd_iter_140000.caffemodel"])

faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)


# load the face mask detector model from disk

print("[INFO] loading emotion detector model...")

maskNet = load_model(MASK_MODEL_PATH)


# initialize the video stream and allow the camera sensor to warm up

print("[INFO] starting video stream...")

vs = VideoStream(0).start()

time.sleep(2.0)


labels = ["happy", "neutral", "sad"]


# loop over the frames from the video stream

while True:

      # grab the frame from the threaded video stream and resize it
```

```
	# to have a maximum width of 400 pixels
	frame = vs.read()
	frame = imutils.resize(frame, width=400)
	original_frame = frame.copy()
	frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
	frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR)

	# detect faces in the frame and determine if they are wearing a
	# face mask or not
	(locs, preds) = detect_and_predict_mask(frame, faceNet,
maskNet, THRESHOLD)

	# loop over the detected face locations and their corresponding
	# locations
	for (box, pred) in zip(locs, preds):
	# unpack the bounding box and predictions
	(startX, startY, endX, endY) = box
	# include the probability in the label
	label = str(labels[np.argmax(pred)])
	# display the label and bounding box rectangle on the output
	# frame

	if label == "happy":
		cv2.putText(original_frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 200, 50), 2)
```

```
            cv2.rectangle(original_frame, (startX, startY), (endX,
endY), (0, 200, 50), 2)

            img = cv2.imread('happy.png')

            cv2.imshow('happy', img)

            cv2.destroyWindow('neutral')

            cv2.destroyWindow('sad')


      elif label == "neutral":

            cv2.putText(original_frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255),2)

            cv2.rectangle(original_frame, (startX, startY), (endX,
endY), (255, 255, 255), 2)

            img = cv2.imread('neutral.png')

            cv2.imshow('neutral', img)

            cv2.destroyWindow('happy')

            cv2.destroyWindow('sad')



      elif label == "sad":

            cv2.putText(original_frame, label, (startX, startY - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 50, 200), 2)

            cv2.rectangle(original_frame, (startX, startY), (endX,
endY), (0, 50, 200), 2)

            img = cv2.imread('sad.png')

            cv2.imshow('sad', img)

            cv2.destroyWindow('happy')

            cv2.destroyWindow('neutral')
```

```
        # show the output frame
        frame = cv2.resize(original_frame, (860, 490))
        cv2.imshow("Facial Expression b", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
        break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```