Detecting Faces Using Webcam

```
pip install opencv-python
```

```
Collecting opencv-python
  Using cached opencv_python-4.6.0.66-cp36-abi3-win_amd64.whl (35.6
MB)
Requirement already satisfied: numpy>=1.14.5 in c:\users\badda\
anaconda3\lib\site-packages (from opencv-python) (1.20.3)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.6.0.66
Note: you may need to restart the kernel to use updated packages.
```

```
import cv2

# for face detection
face_cascade =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

# resolution of the webcam
screen_width = 1280
screen_height = 720

# default webcam
stream = cv2.VideoCapture(0)

while(True):
    # capture frame-by-frame
    (grabbed, frame) = stream.read()
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # try to detect faces in the webcam
    faces = face_cascade.detectMultiScale(rgb, scaleFactor=1.3,
minNeighbors=5)

    # for each faces found
    for (x, y, w, h) in faces:
        # Draw a rectangle around the face
        color = (0, 255, 255) # in BGR
        stroke = 5
        cv2.rectangle(frame, (x, y), (x + w, y + h), color, stroke)

    # show the frame
    cv2.imshow("Image", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):    # Press q to break out
        break                  # of the loop
```

```
# cleanup
stream.release()
cv2.waitKey(1)
cv2.destroyAllWindows()
cv2.waitKey(1)
```

```
-1
```

Face detection using CNN(Deep Learning Model)

```
# Deep Learning CNN model to recognize face
'''This script uses a database of images and creates CNN model on top
of it to test
   if the given image is recognized correctly or not'''

'''####### IMAGE PRE-PROCESSING for TRAINING and TESTING data
#######'''

# Specifying the folder where images are present
TrainingImagePath='C:/Users/badda/Downloads/Face Images/Final Training
Images'

from keras.preprocessing.image import ImageDataGenerator


# Defining pre-processing transformations on raw images of training
data
# These hyper parameters helps to generate slightly twisted versions
# of the original image, which leads to a better model, since it
learns
# on the good and bad mix of images
train_datagen = ImageDataGenerator(
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True)

# Defining pre-processing transformations on raw images of testing
data
# No transformations are done on the testing images
test_datagen = ImageDataGenerator()

# Generating the Training Data
training_set = train_datagen.flow_from_directory(
        TrainingImagePath,
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')
```

```
# Generating the Testing Data
test_set = test_datagen.flow_from_directory(
        TrainingImagePath,
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')

# Printing class labels for each face
test_set.class_indices
```

```
Found 253 images belonging to 17 classes.
Found 253 images belonging to 17 classes.
```

```
{'ShahrukhKhan': 0,
 'face1': 1,
 'face10': 2,
 'face11': 3,
 'face12': 4,
 'face13': 5,
 'face14': 6,
 'face15': 7,
 'face16': 8,
 'face2': 9,
 'face3': 10,
 'face4': 11,
 'face5': 12,
 'face6': 13,
 'face7': 14,
 'face8': 15,
 'face9': 16}
```

```
'''############ Creating lookup table for all faces ############'''
# class_indices have the numeric tag for each face
TrainClasses=training_set.class_indices

# Storing the face and the numeric tag for future reference
ResultMap={}
for faceValue,faceName in
zip(TrainClasses.values(),TrainClasses.keys()):
    ResultMap[faceValue]=faceName

# Saving the face map for future reference
import pickle
with open("ResultsMap.pkl", 'wb') as fileWriteStream:
    pickle.dump(ResultMap, fileWriteStream)

# The model will give answer as a numeric tag
# This mapping will help to get the corresponding face name for it
print("Mapping of Face and its ID",ResultMap)
```

```
# The number of neurons for the output layer is equal to the number of
faces
OutputNeurons=len(ResultMap)
print('\n The Number of output neurons: ', OutputNeurons)
```

```
Mapping of Face and its ID {0: 'ShahrukhKhan', 1: 'face1', 2:
'face10', 3: 'face11', 4: 'face12', 5: 'face13', 6: 'face14', 7:
'face15', 8: 'face16', 9: 'face2', 10: 'face3', 11: 'face4', 12:
'face5', 13: 'face6', 14: 'face7', 15: 'face8', 16: 'face9'}

 The Number of output neurons:  17
```

```
'''####################### Create CNN deep learning model
########################'''
from tensorflow.keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPool2D
from keras.layers import Flatten
from keras.layers import Dense

'''Initializing the Convolutional Neural Network'''
classifier= Sequential()

''' STEP--1 Convolution
# Adding the first layer of CNN
# we are using the format (64,64,3) because we are using TensorFlow
backend
# It means 3 matrix of size (64X64) pixels representing Red, Green and
Blue components of pixels
'''
classifier.add(Convolution2D(32, kernel_size=(5, 5), strides=(1, 1),
input_shape=(64,64,3), activation='relu'))

'''# STEP--2 MAX Pooling'''
classifier.add(MaxPool2D(pool_size=(2,2)))

'''############## ADDITIONAL LAYER of CONVOLUTION for better accuracy
#################'''
classifier.add(Convolution2D(64, kernel_size=(5, 5), strides=(1, 1),
activation='relu'))

classifier.add(MaxPool2D(pool_size=(2,2)))

'''# STEP--3 FLattening'''
classifier.add(Flatten())

'''# STEP--4 Fully Connected Neural Network'''
classifier.add(Dense(64, activation='relu'))

classifier.add(Dense(OutputNeurons, activation='softmax'))
```

```
'''# Compiling the CNN'''
#classifier.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
classifier.compile(loss='categorical_crossentropy', optimizer =
'adam', metrics=["accuracy"])

###########################################################
import time
# Measuring the time taken by the model to train
StartTime=time.time()

# Starting the model training
classifier.fit_generator(
                    training_set,
                    steps_per_epoch=7,
                    epochs=10,
                    validation_data=test_set,
                    validation_steps=10)

EndTime=time.time()
print("###### Total Time Taken: ", round((EndTime-StartTime)/60),
'Minutes ######')
```

```
Epoch 1/10
```

```
C:\Users\badda\AppData\Local\Temp/ipykernel_86716/3785646586.py:44:
UserWarning: `Model.fit_generator` is deprecated and will be removed
in a future version. Please use `Model.fit`, which supports
generators.
  classifier.fit_generator(
```

```
7/7 [==============================] - ETA: 0s - loss: 127.4526 -
accuracy: 0.0588WARNING:tensorflow:Your input ran out of data;
interrupting training. Make sure that your dataset or generator can
generate at least `steps_per_epoch * epochs` batches (in this case, 10
batches). You may need to use the repeat() function when building your
dataset.
7/7 [==============================] - 1s 170ms/step - loss: 127.4526
- accuracy: 0.0588 - val_loss: 13.6670 - val_accuracy: 0.0711
Epoch 2/10
7/7 [==============================] - 1s 93ms/step - loss: 5.9477 -
accuracy: 0.0498
Epoch 3/10
7/7 [==============================] - 1s 89ms/step - loss: 3.0159 -
accuracy: 0.0804
Epoch 4/10
7/7 [==============================] - 1s 90ms/step - loss: 2.6509 -
accuracy: 0.1357
Epoch 5/10
```

```
7/7 [==============================] - 1s 89ms/step - loss: 2.4509 -
accuracy: 0.1810
Epoch 6/10
7/7 [==============================] - 1s 97ms/step - loss: 2.2791 -
accuracy: 0.2217
Epoch 7/10
7/7 [==============================] - 1s 92ms/step - loss: 2.1294 -
accuracy: 0.2443
Epoch 8/10
7/7 [==============================] - 1s 91ms/step - loss: 1.8665 -
accuracy: 0.4253
Epoch 9/10
7/7 [==============================] - 1s 92ms/step - loss: 1.6520 -
accuracy: 0.4389
Epoch 10/10
7/7 [==============================] - 1s 93ms/step - loss: 1.8383 -
accuracy: 0.3575
###### Total Time Taken:  0 Minutes ######
```

```python
'''########### Making single predictions ###########'''
import numpy as np
from tensorflow.keras.preprocessing import image

ImagePath='C:/Users/badda/Downloads/Face Images/Final Testing
Images/ShahrukhKhan/test.webp'
test_image=image.load_img(ImagePath,target_size=(64, 64))
test_image=image.img_to_array(test_image)

test_image=np.expand_dims(test_image,axis=0)

result=classifier.predict(test_image,verbose=0)
#print(training_set.class_indices)

print('####'*10)
print('Prediction is: ',ResultMap[np.argmax(result)])
```

```
########################################
Prediction is:  ShahrukhKhan
```

Tasks

Task 1: Run the above code with given dataset.

Task 2: What did you analyze in the above code. (Include in the PDF)

Task 3: Write what could be the requirement, specification, and environment for the face detection model by taking the below example. (Include in the PDF)

EXAMPLE: LANE ASSISTANCE

REQ: The vehicle must be prevented from veering off the lane.

SPEC: Lane detector accurately identifies lane markings in the input image; the controller generates correct steering commands

ENV: Sensors are providing accurate information about the lane; driver responses when given warning; steering wheel is functional

Task 4: Write analysis on whether our face detection model is satisfying all three things. (Include in the PDF)

Task 5:

Choose one of the problems such as face detection, and vehicle detection.

Write what could be the requirement, specifications, and environment for that problem. (Include in the PDF)

Now create and test the model.

Write analysis on whether the written requirement is feasible or not, environment and specification are correct or not, etc. (Include in the PDF)