face-recognisation-by-transfer-learning

In this project, I have created a Face-Recognition model using the concept of **Feature Tuning**.

Step 1: We start of by collecting our dataset. For this, I have used *Haarcascade FrontalFace*. I have collected 200 images of mine & my friend for training the model & 100 images each for testing the model. You can use the following code to collect the images and prepare the dataset.

```python
#MY TRAIN IMAGES
import cv2
import numpy as np

# Load HAAR face classifier
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

# Load functions
def face_extractor(img):
    # Function detects faces and returns the cropped face
    # If no face detected, it returns the input image

    #gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(img, 1.3, 5)

    if faces is ():
        return None

    # Crop all faces found
    for (x,y,w,h) in faces:
        cropped_face = img[y:y+h, x:x+w]

    return cropped_face

# Initialize Webcam
cap = cv2.VideoCapture(0)
count = 0

# Collect 100 samples of your face from webcam input
while True:

    ret, frame = cap.read()
    if face_extractor(frame) is not None:
        count += 1
        face = cv2.resize(face_extractor(frame), (224, 224))
        #face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)

        # Save file in specified directory with unique name
        file_name_path = 'C://Users//Anuddeeph Nalla//Desktop//MLOPS-WS//dataset//train//Anudeep//anudeep' + str(count) + '.jpg'
        cv2.imwrite(file_name_path, face)

        # Put count on images and display live count
        cv2.putText(face, str(count), (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
        cv2.imshow('Face Cropper', face)

    else:
        print("Face not found")
        pass

    if cv2.waitKey(1) == 13 or count == 200: #13 is the Enter Key
        break

cap.release()
cv2.destroyAllWindows()
print("Collecting Samples Complete")
```

I have used the same block of code multiple times to collect all the training & testing images of me & my friend. You can collect images of more people as per your requirement. For more details check the code

Step 2: Now, we import pre-created MobileNet model from keras.applications. We freeze the already trained layers by *layer.trainable= False.*

```
In [5]: from keras.applications import MobileNet

        # MobileNet was designed to work on 224 x 224 pixel input images sizes
        img_rows, img_cols = 224, 224

        # Re-loads the MobileNet model without the top or FC layers
        MobileNet = MobileNet(weights = 'imagenet',
                         include_top = False,
                         input_shape = (img_rows, img_cols, 3))

        # Here we freeze the last 4 layers
        # Layers are set to trainable as True by default
        for layer in MobileNet.layers:
            layer.trainable = False

        # Let's print our layers
        for (i,layer) in enumerate(MobileNet.layers):
            print(str(i) + " "+ layer.__class__.__name__, layer.trainable)
```

```
Using TensorFlow backend.
```

```
0 InputLayer False
1 ZeroPadding2D False
2 Conv2D False
3 BatchNormalization False
4 ReLU False
5 DepthwiseConv2D False
6 BatchNormalization False
7 ReLU False
8 Conv2D False
9 BatchNormalization False
10 ReLU False
11 ZeroPadding2D False
12 DepthwiseConv2D False
13 BatchNormalization False
14 ReLU False
15 Conv2D False
16 BatchNormalization False
17 ReLU False
18 DepthwiseConv2D False
19 BatchNormalization False
20 ReLU False
21 Conv2D False
22 BatchNormalization False
23 ReLU False
24 ZeroPadding2D False
25 DepthwiseConv2D False
26 BatchNormalization False
27 ReLU False
28 Conv2D False
29 BatchNormalization False
30 ReLU False
31 DepthwiseConv2D False
32 BatchNormalization False
33 ReLU False
34 Conv2D False
35 BatchNormalization False
36 ReLU False
```

Step 3: We add layers as per our requirement. Here, I have used *Softmax* activation function.

In [6]:
```
def lw(bottom_model, num_classes):
    """creates the top or head of the model that will be
    placed ontop of the bottom layers"""

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(num_classes,activation='softmax')(top_model)
    return top_model
```

In [7]:
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers.normalization import BatchNormalization
from keras.models import Model

# Set our class number to 3 (Young, Middle, Old)
num_classes = 2

FC_Head = lw(MobileNet, num_classes)

model = Model(inputs = MobileNet.input, outputs = FC_Head)

print(model.summary())
```

```
conv_dw_12 (DepthwiseConv2D) (None, 7, 7, 512)         4608
_________________________________________________________________
conv_dw_12_bn (BatchNormaliz (None, 7, 7, 512)         2048
_________________________________________________________________
conv_dw_12_relu (ReLU)       (None, 7, 7, 512)         0
_________________________________________________________________
conv_pw_12 (Conv2D)          (None, 7, 7, 1024)        524288
_________________________________________________________________
conv_pw_12_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_pw_12_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_dw_13 (DepthwiseConv2D) (None, 7, 7, 1024)        9216
_________________________________________________________________
conv_dw_13_bn (BatchNormaliz (None, 7, 7, 1024)        4096
_________________________________________________________________
conv_dw_13_relu (ReLU)       (None, 7, 7, 1024)        0
_________________________________________________________________
conv_pw_13 (Conv2D)          (None, 7, 7, 1024)        1048576
```

Step 4: Next, we load our dataset. We have used the augmentation technique to increase our dataset since the size of original dataset is too small for a good accuracy.

In [8]:
```
from keras_preprocessing.image import ImageDataGenerator
```

In [10]:
```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
test_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=45,
      width_shift_range=0.3,
      height_shift_range=0.3,
      horizontal_flip=True,
      fill_mode='nearest')
train_generator = train_datagen.flow_from_directory(
        'dataset/train',
        target_size=(img_rows, img_cols),
        batch_size=64,
        class_mode='categorical')
validation_generator = test_datagen.flow_from_directory(
        'dataset/test/',
        target_size=(img_rows, img_cols),
        batch_size=32,
        class_mode='categorical')
```

```
Found 400 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
```

Step 5: Now, we begin training our model.

```
In [11]: from keras.optimizers import RMSprop
         from keras.callbacks import ModelCheckpoint, EarlyStopping
         from keras.models import load_model


         checkpoint = ModelCheckpoint("facer.h5",
                                      monitor="val_loss",
                                      mode="min",
                                      save_best_only = True,
                                      verbose=1)

         earlystop = EarlyStopping(monitor = 'val_loss',
                                   min_delta = 0,
                                   patience = 3,
                                   verbose = 1,
                                   restore_best_weights = True)

         # we put our call backs into a callback list
         callbacks = [earlystop, checkpoint]

         # We use a very small learning rate
         model.compile(loss = 'categorical_crossentropy',
                       optimizer = RMSprop(lr = 0.001),
                       metrics = ['accuracy'])
```

```
In [12]: # Enter the number of training and validation samples here
         nb_train_samples = 544
         nb_validation_samples = 200

         # We only train 10 EPOCHS
         epochs = 5
         batch_size = 64

         history = model.fit_generator(
             train_generator,
             steps_per_epoch = nb_train_samples // batch_size,
             epochs = epochs,
             callbacks = callbacks,
             validation_data = validation_generator,
             validation_steps = nb_validation_samples // batch_size)

         classifier = load_model('facer.h5')
```

```
Epoch 1/5
8/8 [==============================] - 115s 14s/step - loss: 4.1117 - accuracy: 0.5797 - val_loss: 0.0930 - val_accuracy: 0.989
6

Epoch 00001: val_loss improved from inf to 0.09295, saving model to facer.h5
Epoch 2/5
8/8 [==============================] - 98s 12s/step - loss: 0.0152 - accuracy: 0.9914 - val_loss: 0.0980 - val_accuracy: 0.9896
```

The model has been effectively trained and ready to use. You can use this model for prediction.

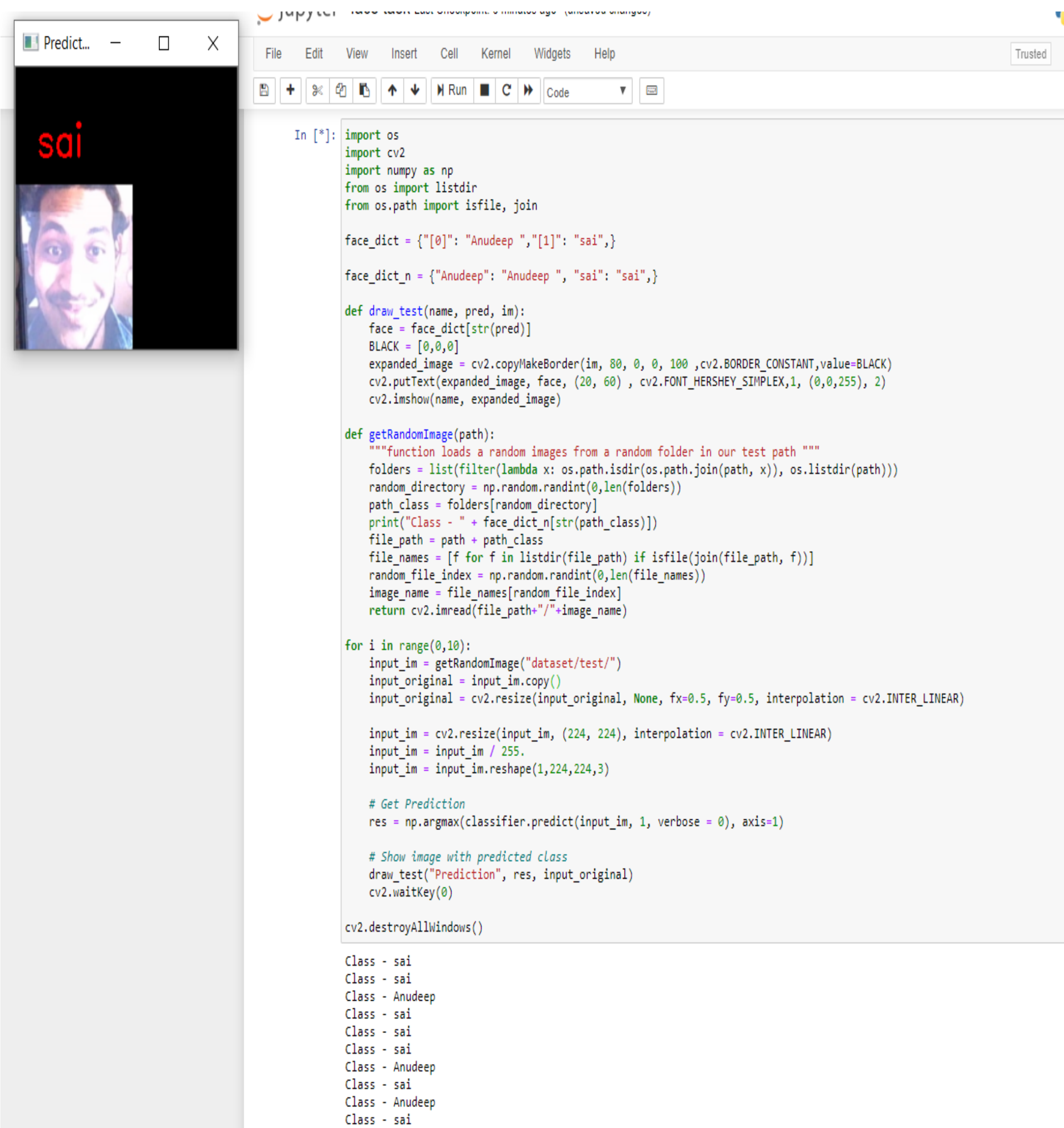
In this model, I got 99% accuracy, because the data was very less.

Step 6: Now, I have loaded the created model for prediction, and predicted mine and my friend face.

```
In [13]: from keras.models import load_model

         classifier = load_model('facer.h5')
```

The output of predicted model:




```
In [*]: import os
import cv2
import numpy as np
from os import listdir
from os.path import isfile, join

face_dict = {"[0]": "Anudeep ","[1]": "sai",}

face_dict_n = {"Anudeep": "Anudeep ", "sai": "sai",}

def draw_test(name, pred, im):
    face = face_dict[str(pred)]
    BLACK = [0,0,0]
    expanded_image = cv2.copyMakeBorder(im, 80, 0, 0, 100 ,cv2.BORDER_CONSTANT,value=BLACK)
    cv2.putText(expanded_image, face, (20, 60) , cv2.FONT_HERSHEY_SIMPLEX,1, (0,0,255), 2)
    cv2.imshow(name, expanded_image)

def getRandomImage(path):
    """function loads a random images from a random folder in our test path """
    folders = list(filter(lambda x: os.path.isdir(os.path.join(path, x)), os.listdir(path)))
    random_directory = np.random.randint(0,len(folders))
    path_class = folders[random_directory]
    print("Class - " + face_dict_n[str(path_class)])
    file_path = path + path_class
    file_names = [f for f in listdir(file_path) if isfile(join(file_path, f))]
    random_file_index = np.random.randint(0,len(file_names))
    image_name = file_names[random_file_index]
    return cv2.imread(file_path+"/"+image_name)

for i in range(0,10):
    input_im = getRandomImage("dataset/test/")
    input_original = input_im.copy()
    input_original = cv2.resize(input_original, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)

    input_im = cv2.resize(input_im, (224, 224), interpolation = cv2.INTER_LINEAR)
    input_im = input_im / 255.
    input_im = input_im.reshape(1,224,224,3)

    # Get Prediction
    res = np.argmax(classifier.predict(input_im, 1, verbose = 0), axis=1)

    # Show image with predicted class
    draw_test("Prediction", res, input_original)
    cv2.waitKey(0)

cv2.destroyAllWindows()

Class - sai
Class - sai
Class - Anudeep
Class - sai
Class - sai
Class - sai
Class - Anudeep
Class - sai
Class - Anudeep
Class - sai
```