Project

Report on

Side Face Identification System

Post Graduate Diploma in Artificial Intelligence

From C-DAC, ACTS (Pune)

Guided by:

Mr. Sankalp Varshney.

Presented by:

Abhishek Akkewar 190840128002

Harshvardhan Sahu 190840128013

Rishabh Kumar 190840128028

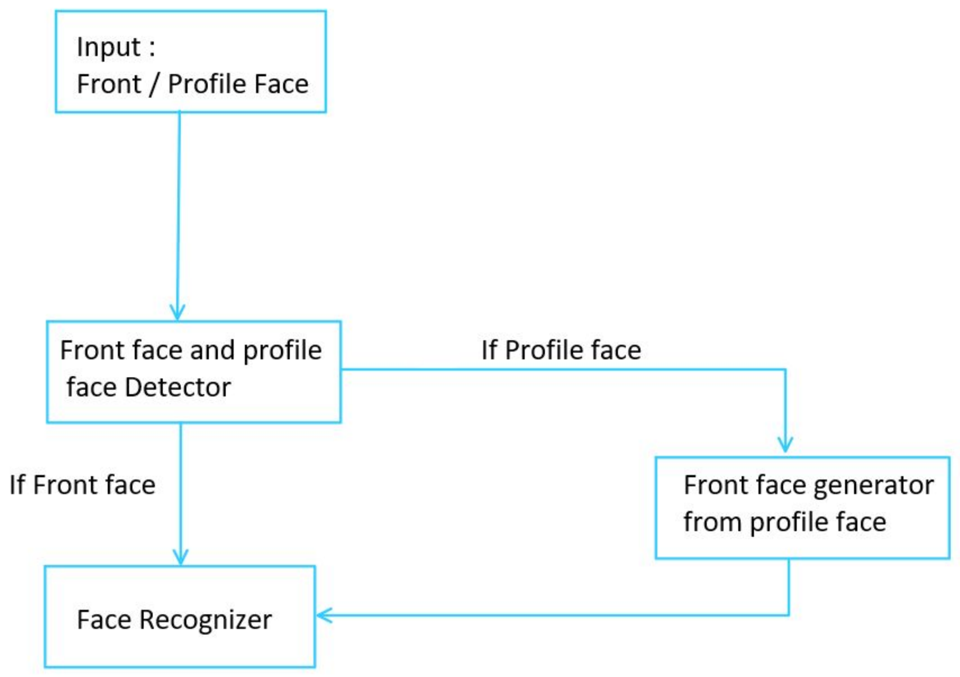
Shivam Pradhan 190840128031

Centre of Development of Advanced Computing (C-DAC), Pune.

Abstract:

Face detection and recognition have gained widespread application in versatile fields like security, emotion detection, attendance tracking etc. But many algorithms do not work well with images of a side view of the faces. Through this project we aim to address this problem. We aim to fix the lack of front view of the face during face detection applications through CCTVs and other cameras. Apart from running face detection algorithms over images of the front view of the face, we hope to expand the effectiveness of current algorithms by using the side view of a person's face to generate a frontal image which can then be used in Face Recognition algorithms. This will then allow better recognition of people from all angles and poses than current algorithms.

Index :



Input :
Front / Profile Face
Front face and profile face Detector
If Profile face
Front face generator from profile face
If Front face
Face Recognizer

Fig : Flowchart

Front Face and Side Face Detection

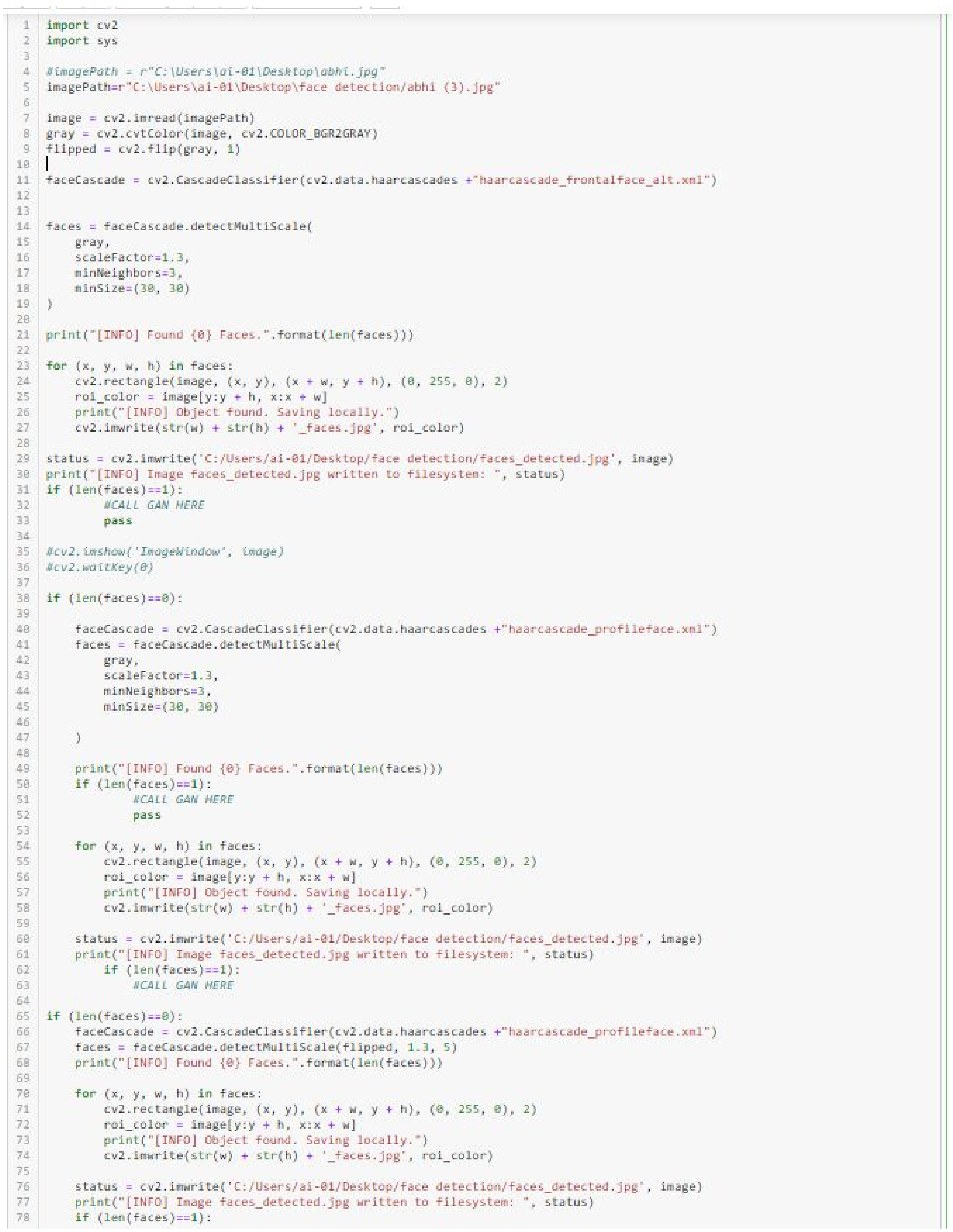
OpenCV:

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high-resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

Harcascade classifier:

Haar Cascade classifiers are an effective way for face detection. This method was proposed by Paul Viola and Michael Jones in their paper Rapid Face Detection using a Boosted Cascade of Simple Features. Haar Cascade is a machine learning-based approach where a lot of positive and negative images are used to train the classifier.

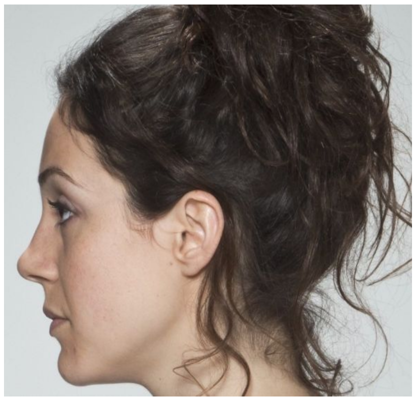
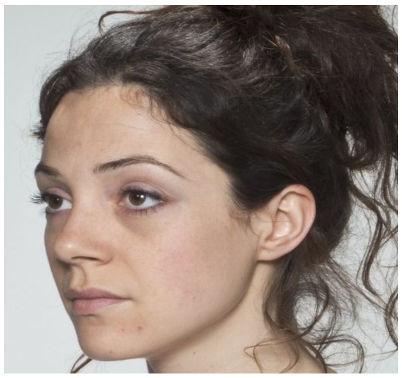
- **Positive images –** These images contain the images which we want our classifier to identify.
- **Negative Images –** Images of everything else, which do not contain the object we want to detect.



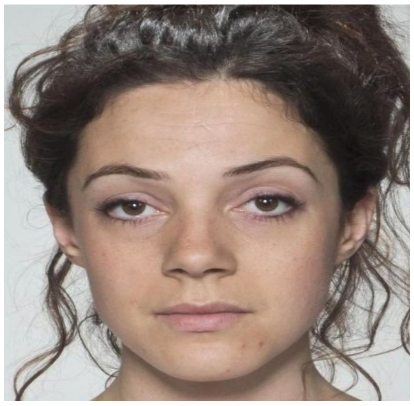
```python
import cv2
import sys

#imagePath = r"C:\Users\ai-01\Desktop\abhi.jpg"
imagePath=r"C:\Users\ai-01\Desktop\face detection/abhi (3).jpg"

image = cv2.imread(imagePath)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
flipped = cv2.flip(gray, 1)

faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +"haarcascade_frontalface_alt.xml")


faces = faceCascade.detectMultiScale(
    gray,
    scaleFactor=1.3,
    minNeighbors=3,
    minSize=(30, 30)
)

print("[INFO] Found {0} Faces.".format(len(faces)))

for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    roi_color = image[y:y + h, x:x + w]
    print("[INFO] Object found. Saving locally.")
    cv2.imwrite(str(w) + str(h) + '_faces.jpg', roi_color)

status = cv2.imwrite('C:/Users/ai-01/Desktop/face detection/faces_detected.jpg', image)
print("[INFO] Image faces_detected.jpg written to filesystem: ", status)
if (len(faces)==1):
        #CALL GAN HERE
        pass

#cv2.imshow('ImageWindow', image)
#cv2.waitKey(0)

if (len(faces)==0):

    faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +"haarcascade_profileface.xml")
    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.3,
        minNeighbors=3,
        minSize=(30, 30)

    )

    print("[INFO] Found {0} Faces.".format(len(faces)))
    if (len(faces)==1):
            #CALL GAN HERE
            pass

    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        roi_color = image[y:y + h, x:x + w]
        print("[INFO] Object found. Saving locally.")
        cv2.imwrite(str(w) + str(h) + '_faces.jpg', roi_color)

    status = cv2.imwrite('C:/Users/ai-01/Desktop/face detection/faces_detected.jpg', image)
    print("[INFO] Image faces_detected.jpg written to filesystem: ", status)
        if (len(faces)==1):
            #CALL GAN HERE

if (len(faces)==0):
    faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades +"haarcascade_profileface.xml")
    faces = faceCascade.detectMultiScale(flipped, 1.3, 5)
    print("[INFO] Found {0} Faces.".format(len(faces)))

    for (x, y, w, h) in faces:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
        roi_color = image[y:y + h, x:x + w]
        print("[INFO] Object found. Saving locally.")
        cv2.imwrite(str(w) + str(h) + '_faces.jpg', roi_color)

    status = cv2.imwrite('C:/Users/ai-01/Desktop/face detection/faces_detected.jpg', image)
    print("[INFO] Image faces_detected.jpg written to filesystem: ", status)
    if (len(faces)==1):
```

Side Face to Front Face Generation

Module - Creation of front face through the side-view image of the same person. We aim to give our algorithm an image of a side view of the face (at 90 degree or 45 degree face image to the front face image)

And we aim to generate the front face of the same person. E.g for the above example we want to get



For this task, we aim to deploy a GAN model for training and generating the image.

Generative Adversarial Network (GAN)

Overview -

A **generative adversarial network** (**GAN**) is a class of machine learning systems which given a training set, learns to generate new data with the same statistics as the training set. For example, a GAN trained with images of handbags will then be able to generate new pictures of handbags.

In short, GAN deals with a form of random variable generation problem. E.g.
The problem of generating a new image of dog can be rephrased into a problem of generating a random vector in the N-dimensional vector space that follows the "dog probability distribution"

To train (optimize) the network to express the right transform function, we will use an indirect method of training. The direct training method consists in comparing the true and the generated probability distributions and backpropagating the difference (the error) through the network. For the indirect training method, we do not directly compare the true and generated distributions. Instead in GAN, we indirectly train the network through a downstream task where we make two models compete - one model generates images and the other models try to distinguish between real images and the images created by the other model. Both models train themselves by attempting to fool the other network and adjusting their parameters accordingly.

A GAN is made up of two different learning models working in tandem with each other:-

1. **Generator** - This model learns to map from a latent space to a data distribution of interest. The generator is a neural network that models a transform function. It takes as input a simple random variable and must return, once trained, a random variable that follows the targeted distribution. In short, the generator deals with a form of random variable generation problem. his model learns to extract the most important features

2. **Discriminator** - As the function to predict the random variable is very complicated, we decide to use another model - the discriminator with another neural network to model a discriminative function. It takes as input a point (for input like a picture this could be an N-dimensional vector) and returns as output the probability of this point to be a "true" one. So in a way it tells whether the data

generated can belong to the input training set data or not (i.e if the generated image is similar enough to image in our training dataset).

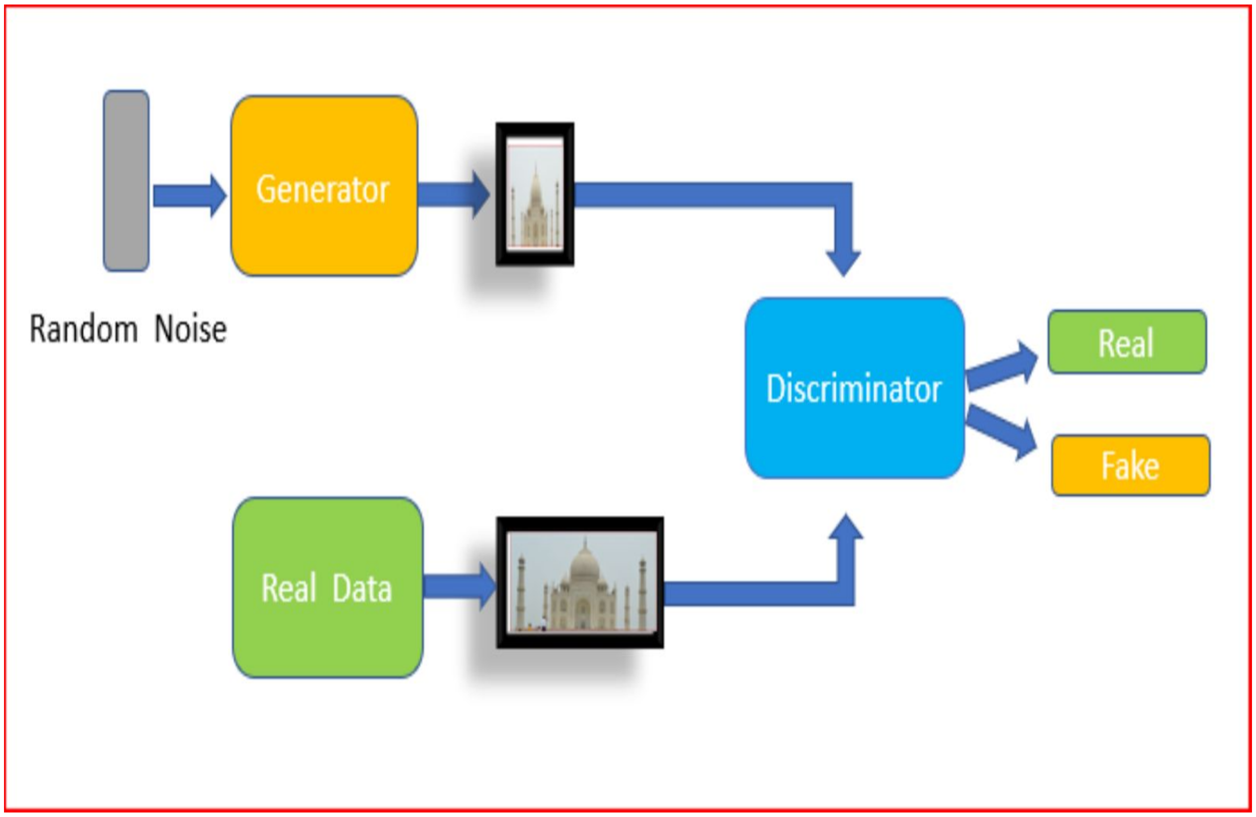
So if we have trained the network on a dataset of faces, then our generator network will learn to extract major features of faces from the images compressing the size of the image and then decompress it - i.e reconstruct the image using those features as a base and the parameters learned through training.
The Discriminator, in turn, will try to tell whether the image generated is a real(i.e that the generated image is a face) or is fake (is not a face).

Normally GANs are unsupervised learning algorithms that use a supervised loss as part of the training so images/data generated by GAN network tries to follow the training data distribution but not target a specific class or identity i.e. a GAN generating a face does not try to generate an image of a specific person. It is only concerned with generating random faces.

In our problem, we not only need to generate a face but we need that face to belong to the same person whose face side view image was given to it. So normal GAN would not be enough.

For our project, we are going to use conditional GAN i.e where we impose an extra condition for training in the discriminator.
In our model of GAN, in addition to sending the generated front face image, we will send the target image i.e the real front face of the person. So our discriminator we calculate two things -whether the image sent by the generator is real(a face) or not and also how similar is the image sent by the generator to the target image.

Discriminator takes two sets of input, one input comes from the training dataset(real data) and the other input is the dataset generated by Generator.



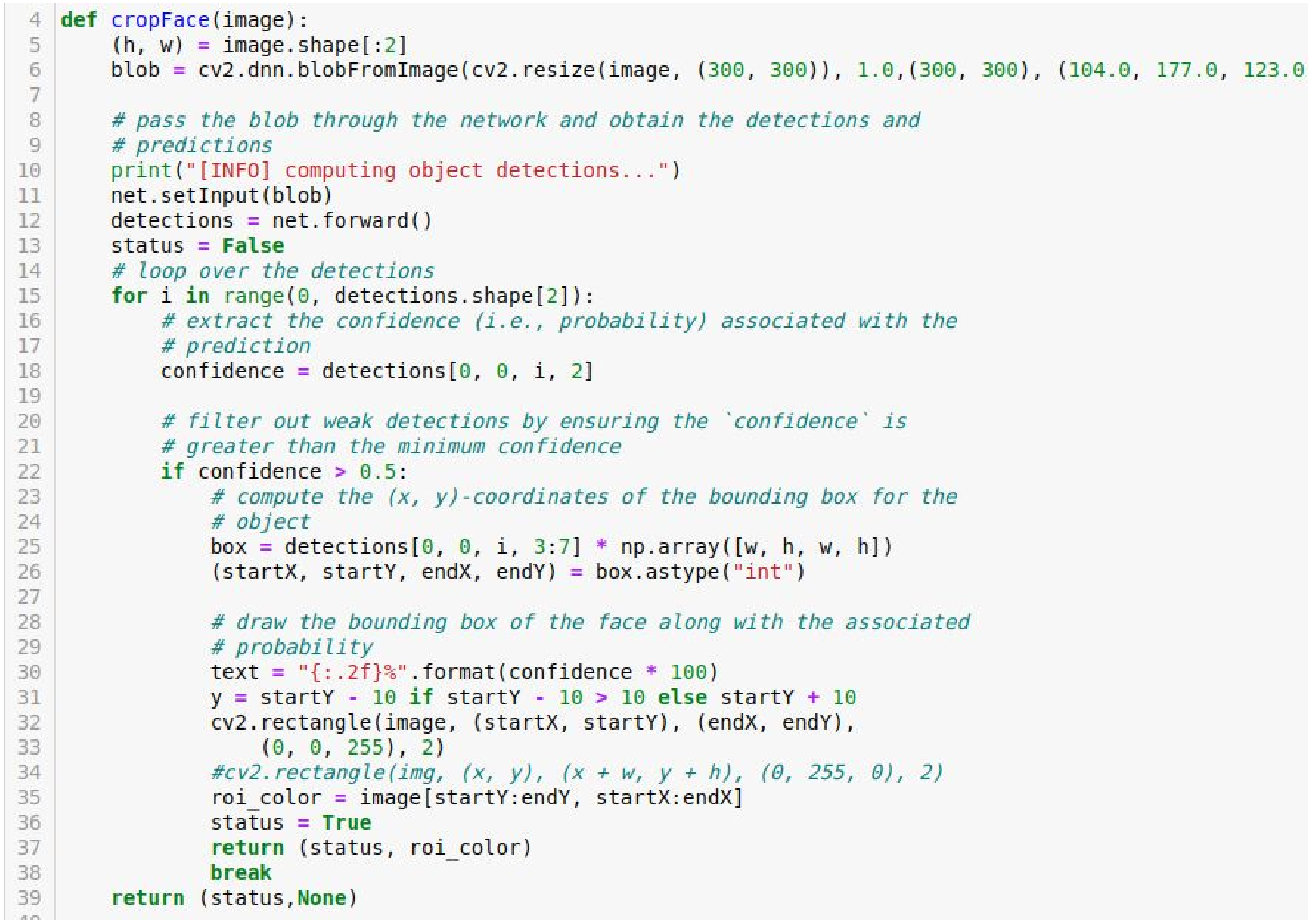
Code Analysis -

For our project, our data is arranged in a folder called ‘Cset

```
├── Cset
│   ├── Abhishek Akkewar [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Abhishek Tomar [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Aditya [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Anish [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
……….
```

This way there are folders for 21 different people with each folder holding about 5-6 images.
But all these photos are full body portrait clicks - e.g.

So we need to normalize the data by preparing to preprocess the data. The reason we perform this normalization is due to the fact that many facial recognition algorithms as it removes unwanted noise (anything except the face area) from the picture so that GAN is not influenced by stray objects etc in the picture. To normalize our data, we will need to crop the face area from the picture. To do that we will use the Facial detection code we have developed in previous modules and develop our own operation 'cropFace()'



```python
 4 def cropFace(image):
 5     (h, w) = image.shape[:2]
 6     blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,(300, 300), (104.0, 177.0, 123.0
 7 
 8     # pass the blob through the network and obtain the detections and
 9     # predictions
10     print("[INFO] computing object detections...")
11     net.setInput(blob)
12     detections = net.forward()
13     status = False
14     # loop over the detections
15     for i in range(0, detections.shape[2]):
16         # extract the confidence (i.e., probability) associated with the
17         # prediction
18         confidence = detections[0, 0, i, 2]
19 
20         # filter out weak detections by ensuring the `confidence` is
21         # greater than the minimum confidence
22         if confidence > 0.5:
23             # compute the (x, y)-coordinates of the bounding box for the
24             # object
25             box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
26             (startX, startY, endX, endY) = box.astype("int")
27 
28             # draw the bounding box of the face along with the associated
29             # probability
30             text = "{:.2f}%".format(confidence * 100)
31             y = startY - 10 if startY - 10 > 10 else startY + 10
32             cv2.rectangle(image, (startX, startY), (endX, endY),
33                 (0, 0, 255), 2)
34             #cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
35             roi_color = image[startY:endY, startX:endX]
36             status = True
37             return (status, roi_color)
38             break
39     return (status,None)
```

In the above code , from line 6-12, we are using our face detection module to detect all the faces in the photo.

Then in the line 16-38 we are calculating the rectangle over the first face detected by our previous module with confidence>0.5.The coordinates of the rectangle (starting X, Y ad ending X,Y coordinates are extracted in line 25-26.) Then using the coordinates of this rectangle, we draw a box over the face in line 31-32 and from line 35 we are storing the region of the picture enclosed by the box in the variable 'roi_color'. So this variable has now the picture of the only the face of the box which is returned. 'None' object is returned along with status= False if no face detected.

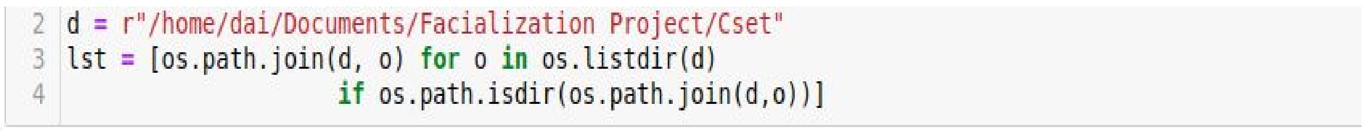
Output image from cropface function is something like this for our above example -

Once we have the function, now we have to run this function on the whole dataset to create a new dataset of cropped dataset.

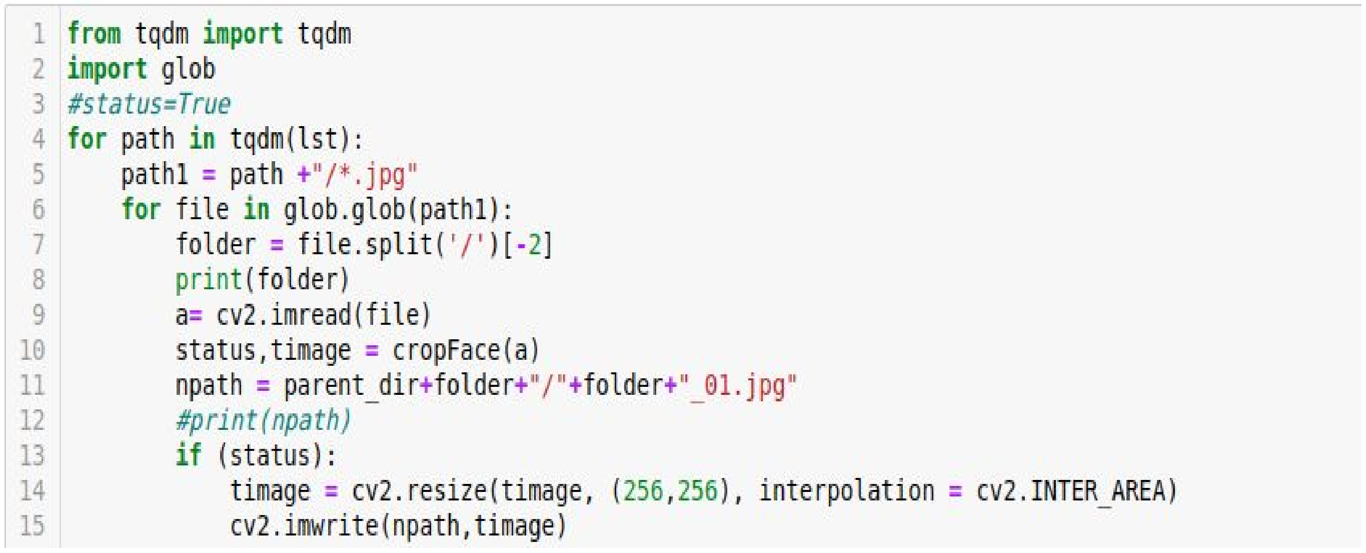
In the above code - we create a new folder named ‘CropCset’ with the same structure as our original dataset.

```
├── Cset
│   ├── Abhishek Akkewar [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Abhishek Tomar [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Aditya [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
│   ├── Anish [1 front face image ]
│       ├── side [4 side img-left 90 degree,left 45 deg,right 90 deg, right 45 deg ]
……….
```

Now we will run our function ‘cropFace() on each image and store them in the corresponding folder location(with similar names) in the new dataset folder.



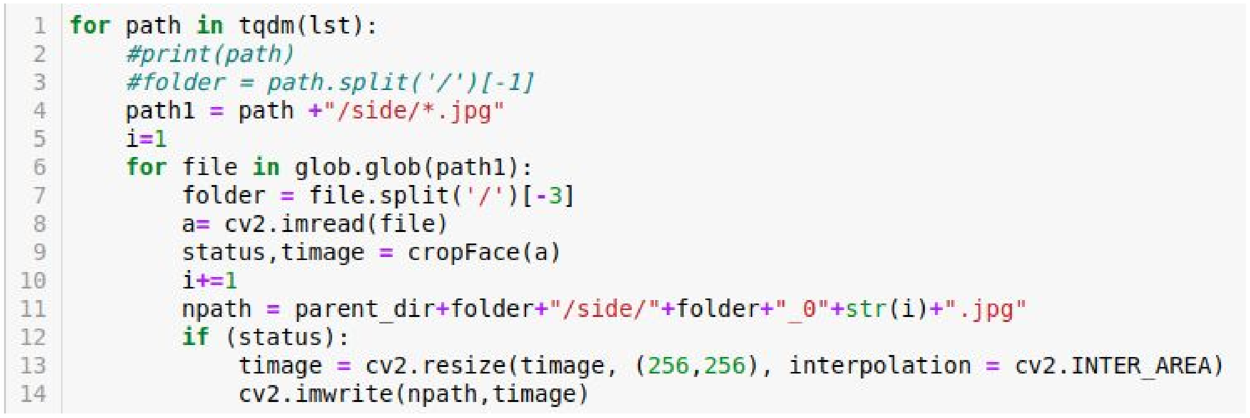
```
2 d = r"/home/dai/Documents/Facialization Project/Cset"
3 lst = [os.path.join(d, o) for o in os.listdir(d)
4                     if os.path.isdir(os.path.join(d,o))]
```



```
 1 from tqdm import tqdm
 2 import glob
 3 #status=True
 4 for path in tqdm(lst):
 5     path1 = path +"/*.jpg"
 6     for file in glob.glob(path1):
 7         folder = file.split('/')[-2]
 8         print(folder)
 9         a= cv2.imread(file)
10         status,timage = cropFace(a)
11         npath = parent_dir+folder+"/"+folder+"_01.jpg"
12         #print(npath)
13         if (status):
14             timage = cv2.resize(timage, (256,256), interpolation = cv2.INTER_AREA)
15             cv2.imwrite(npath,timage)
```

Here in line 2-4 in the above cell we create a variable list in which we have extracted all the folder paths inside the 'Cset' folder. Then in the next cell, line 4 we are running code for each folder path -

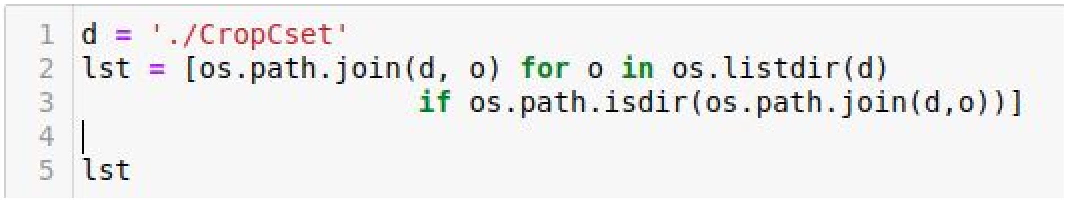
In each loop, it reads the file path all the jpg files with any names inside the folder. Since we have only one image (front face) directly inside, for each outer loop, the inner loop will run once (line 6). In line 7, the function is splitting the file path and extracting the folder name (which is the name of the person who the photo belongs to). Then using this value it creates a new path, filename for the cropped photo - folderName_01.jpg. (line 11). In line 9,10 we read the image from the original dataset and run our 'cropFace()' function on it to get a cropped image in the variable 'image'. From lines 13 -15 we write the cropped image to its new location with an appropriate name. (For the front face, the image has name personName_01.jpg). We are also resizing the image to 256 * 256 pixels to make all images of the same size.



```
 1 for path in tqdm(lst):
 2     #print(path)
 3     #folder = path.split('/')[-1]
 4     path1 = path +"/side/*.jpg"
 5     i=1
 6     for file in glob.glob(path1):
 7         folder = file.split('/')[-3]
 8         a= cv2.imread(file)
 9         status,timage = cropFace(a)
10         i+=1
11         npath = parent_dir+folder+"/side/"+folder+"_0"+str(i)+".jpg"
12         if (status):
13             timage = cv2.resize(timage, (256,256), interpolation = cv2.INTER_AREA)
14             cv2.imwrite(npath,timage)
```

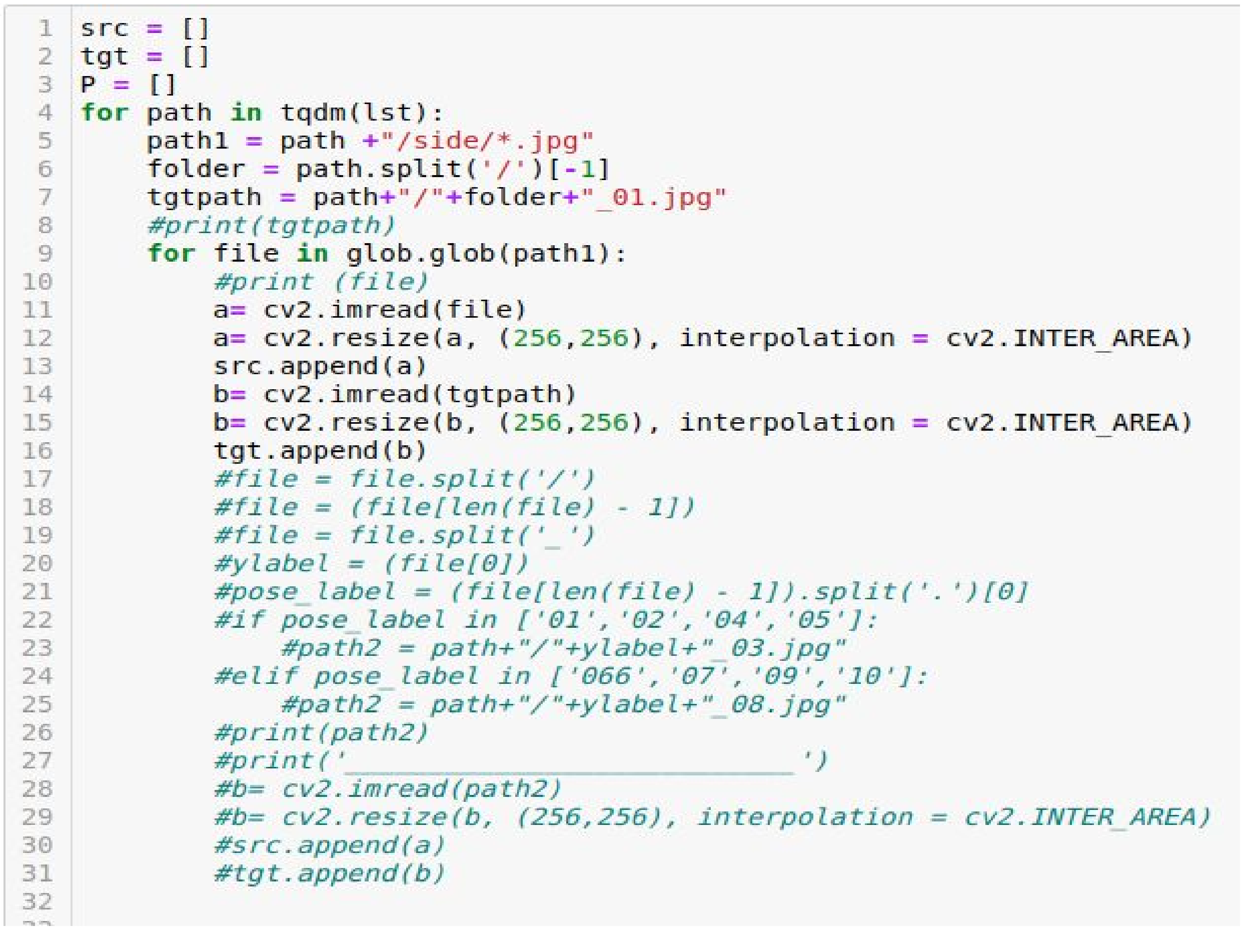
We run a similar code as above, but add 'side' in the folder path to access 'side' folder in each person's folder. Each inner loop(line 6) will run for 4-5 times (once for each side photo), read the image, crop it, resize it and store it in the location - CropCset->personFolder->side->personname_02. Jpg (number prefix run from 2 to 5 for each file)

Now that we have normalized the data, we are ready to load it for training.



```
1 d = './CropCset'
2 lst = [os.path.join(d, o) for o in os.listdir(d)
3                         if os.path.isdir(os.path.join(d,o))]
4 
5 lst
```

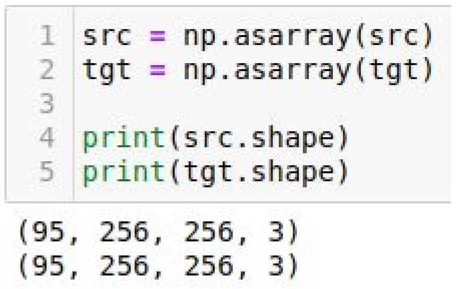
Here in line 2-4 in the above cell we create a variable lst in which we have extracted all the folder paths inside the 'CropCset' folder



```
src = []
tgt = []
P = []
for path in tqdm(lst):
    path1 = path +"/side/*.jpg"
    folder = path.split('/')[-1]
    tgtpath = path+"/"+folder+"_01.jpg"
    #print(tgtpath)
    for file in glob.glob(path1):
        #print (file)
        a= cv2.imread(file)
        a= cv2.resize(a, (256,256), interpolation = cv2.INTER_AREA)
        src.append(a)
        b= cv2.imread(tgtpath)
        b= cv2.resize(b, (256,256), interpolation = cv2.INTER_AREA)
        tgt.append(b)
        #file = file.split('/')
        #file = (file[len(file) - 1])
        #file = file.split('_')
        #ylabel = (file[0])
        #pose_label = (file[len(file) - 1]).split('.')[0]
        #if pose_label in ['01','02','04','05']:
            #path2 = path+"/"+ylabel+"_03.jpg"
        #elif pose_label in ['066','07','09','10']:
            #path2 = path+"/"+ylabel+"_08.jpg"
        #print(path2)
        #print('____________________________')
        #b= cv2.imread(path2)
        #b= cv2.resize(b, (256,256), interpolation = cv2.INTER_AREA)
        #src.append(a)
        #tgt.append(b)

```

In the above code, we create two empty list - ‘src’ and ‘tgt’. Then we run a loop over each folder path in the ‘lst’ variable. For each folder path - we create two paths -
Path1 - CropCset ->personFolder->side
Tgtpath - CropCset -> personFolder ->personname_01.jpg

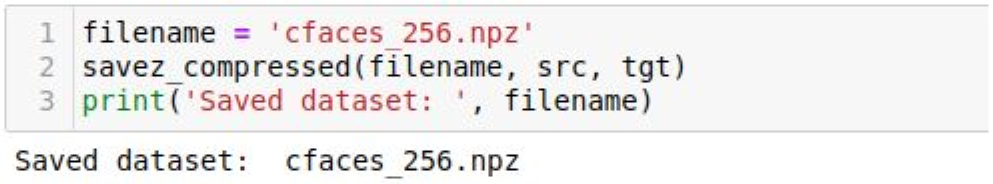
We run an inner loop to detect all jpg files inside the ‘side’ folder. It reads the side image into variable ‘a’. Then it directly reads the image with the path in tgtpath (which is the path of the corresponding front face) and store it in ‘b’. We add the side face in the ‘src’ list and the corresponding front face in ‘tgt’ list.



```
src = np.asarray(src)
tgt = np.asarray(tgt)

print(src.shape)
print(tgt.shape)
```

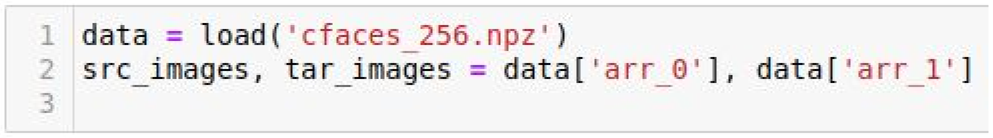
```
(95, 256, 256, 3)
(95, 256, 256, 3)
```

Now we convert the src and the list into NumPy arrays. We can see that there are 95 pairs of side face and front face images.



```
filename = 'cfaces_256.npz'
savez_compressed(filename, src, tgt)
print('Saved dataset: ', filename)
```

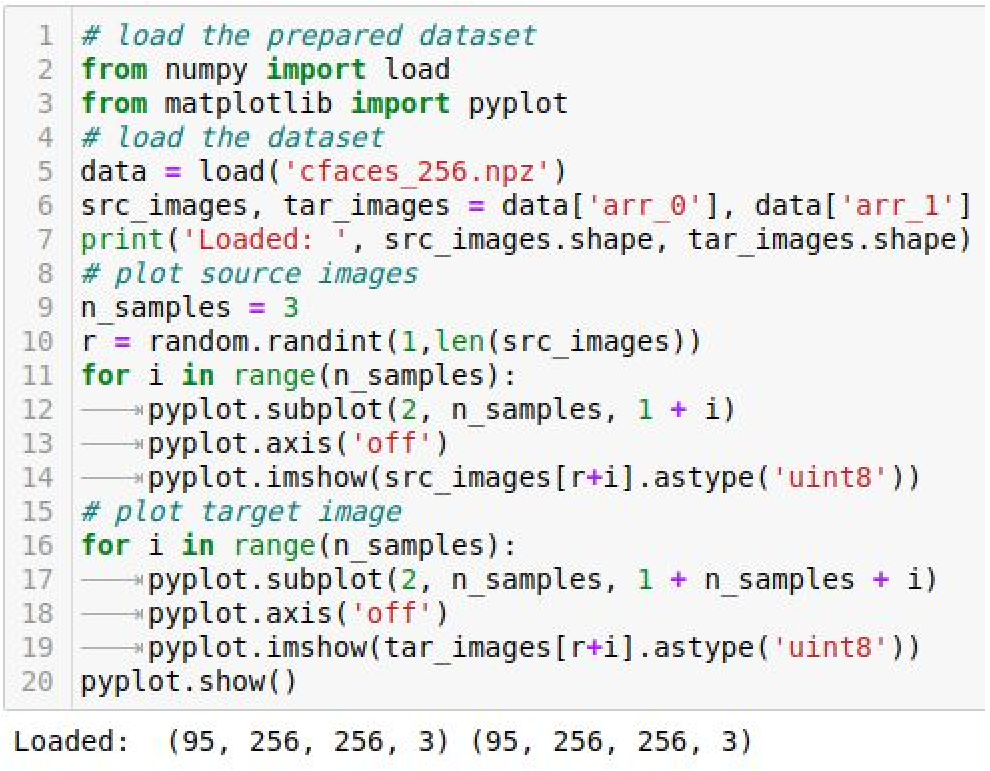
```
Saved dataset:  cfaces_256.npz
```



```
data = load('cfaces_256.npz')
src_images, tar_images = data['arr_0'], data['arr_1']

```

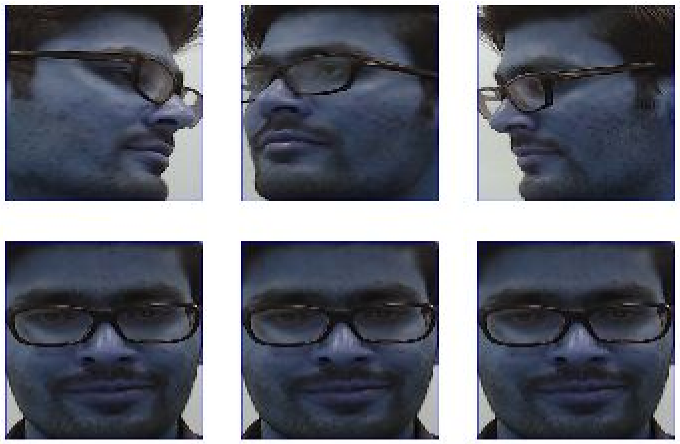
For redundancy purposes and not to waste time in future loading the datasets through code, we store the two lists together in a paired way using .npz format. **NPZ** is a **file format** by NumPy that provides storage of array data using gzip compression.

Then we load the data into src_images NumPy array and 'tar_images' NumPy array.



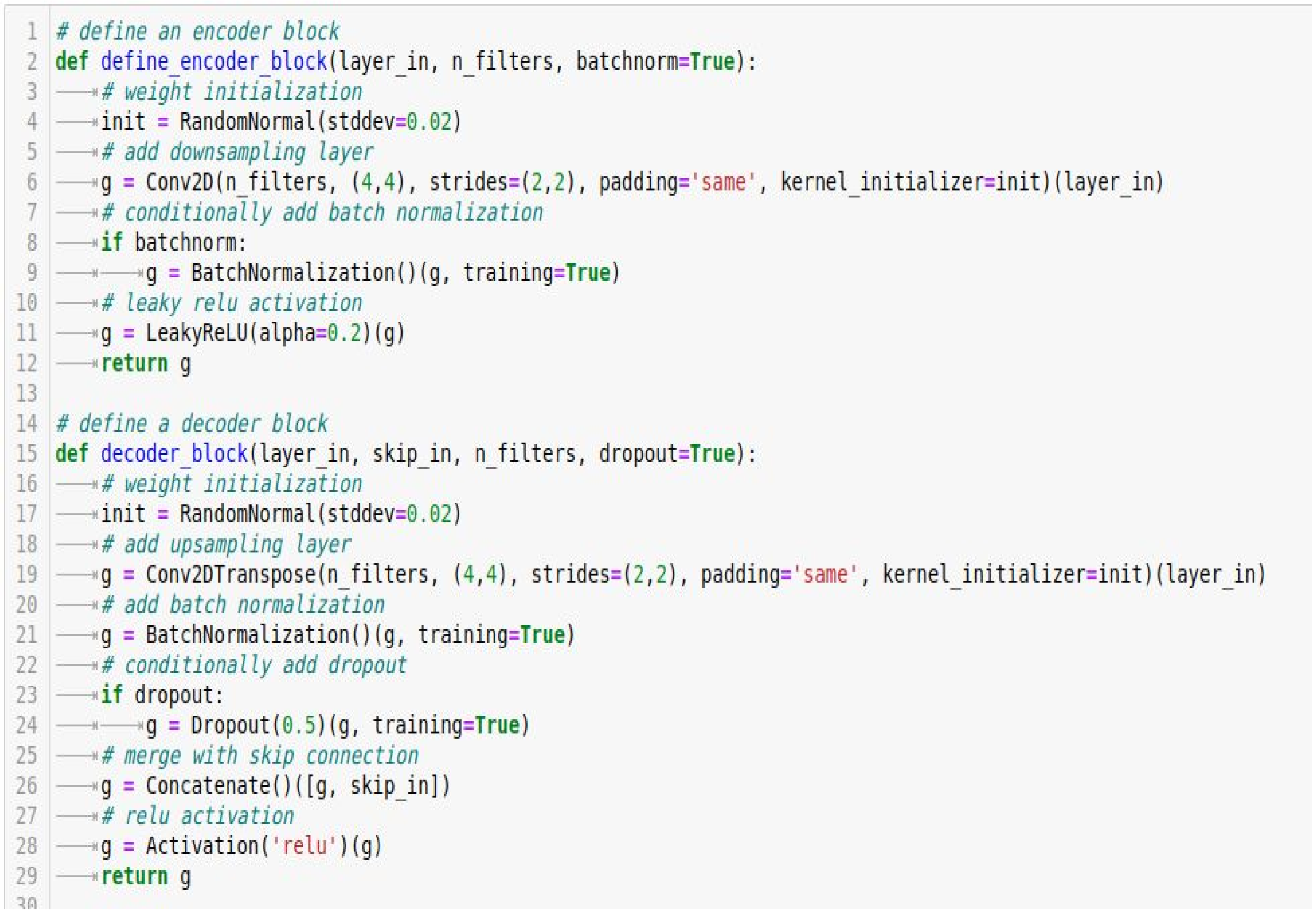
```python
# load the prepared dataset
from numpy import load
from matplotlib import pyplot
# load the dataset
data = load('cfaces_256.npz')
src_images, tar_images = data['arr_0'], data['arr_1']
print('Loaded: ', src_images.shape, tar_images.shape)
# plot source images
n_samples = 3
r = random.randint(1,len(src_images))
for i in range(n_samples):
    pyplot.subplot(2, n_samples, 1 + i)
    pyplot.axis('off')
    pyplot.imshow(src_images[r+i].astype('uint8'))
# plot target image
for i in range(n_samples):
    pyplot.subplot(2, n_samples, 1 + n_samples + i)
    pyplot.axis('off')
    pyplot.imshow(tar_images[r+i].astype('uint8'))
pyplot.show()
```

```
Loaded:  (95, 256, 256, 3) (95, 256, 256, 3)
```



We plot random three images through pyplot to see if image pairs have been loaded correctly - i.e side face and front face are of the same person at the same index in the NumPy arrays. As we can see from above, it is so. (Note since OpenCv reads images in BGR format while pyplot assumes RGB format, the pyplot of images is showing bluish)

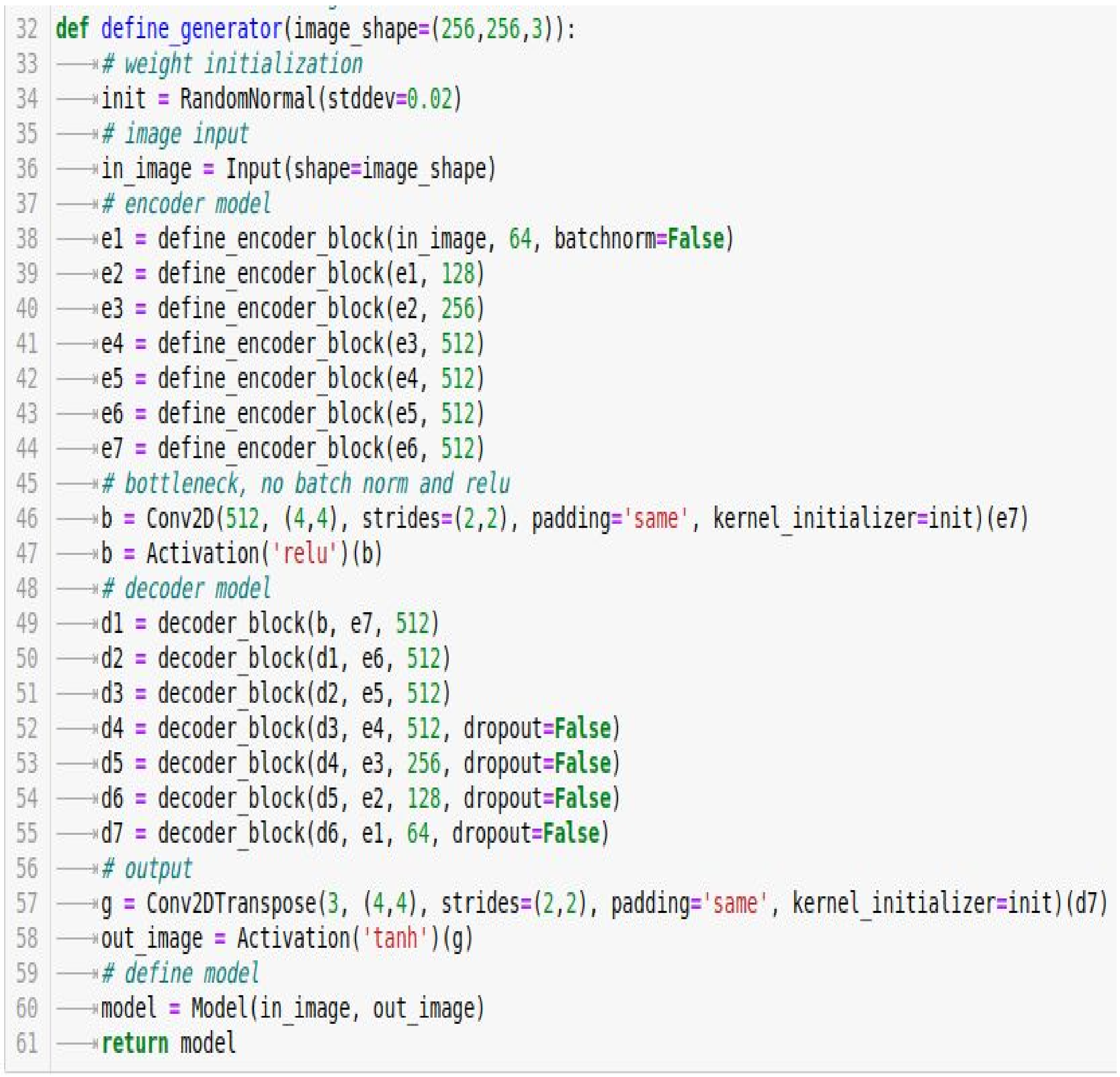
Now we define the architecture of our GAN model -



```python
# define an encoder block
def define_encoder_block(layer_in, n_filters, batchnorm=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add downsampling layer
    g = Conv2D(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # conditionally add batch normalization
    if batchnorm:
        g = BatchNormalization()(g, training=True)
    # leaky relu activation
    g = LeakyReLU(alpha=0.2)(g)
    return g

# define a decoder block
def decoder_block(layer_in, skip_in, n_filters, dropout=True):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # add upsampling layer
    g = Conv2DTranspose(n_filters, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(layer_in)
    # add batch normalization
    g = BatchNormalization()(g, training=True)
    # conditionally add dropout
    if dropout:
        g = Dropout(0.5)(g, training=True)
    # merge with skip connection
    g = Concatenate()([g, skip_in])
    # relu activation
    g = Activation('relu')(g)
    return g

```

Here we are defining layer set/block for encoding and decoding. In encoding we are running a convolutional layer (activation function used is leaky relu as we will be normalizing the pixel values to scale of -1 to 1 so we need a function that has slope for negative values) to extract important features of the images through the use of filters thereby reducing the size of the image but increasing the number of output channels.

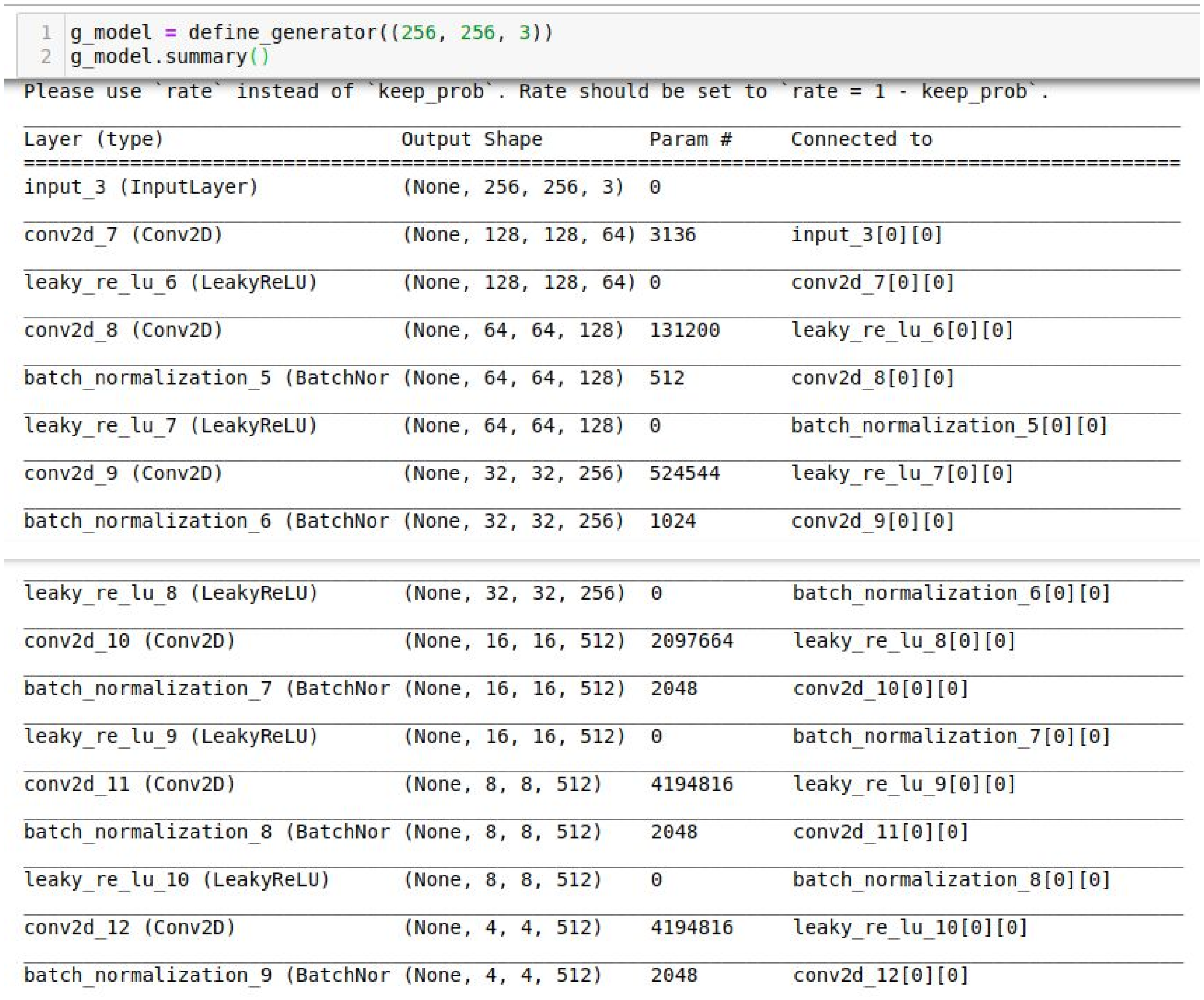
In the decoder block we are doing the reverse of what we did in encoding. We are using Conv2DTranspse to try to guess and fill in the pixel values lost during the encoding process of the images creating image/data of higher dimension from feature values extracted in encoding.



```python
def define_generator(image_shape=(256,256,3)):
    # weight initialization
    init = RandomNormal(stddev=0.02)
    # image input
    in_image = Input(shape=image_shape)
    # encoder model
    e1 = define_encoder_block(in_image, 64, batchnorm=False)
    e2 = define_encoder_block(e1, 128)
    e3 = define_encoder_block(e2, 256)
    e4 = define_encoder_block(e3, 512)
    e5 = define_encoder_block(e4, 512)
    e6 = define_encoder_block(e5, 512)
    e7 = define_encoder_block(e6, 512)
    # bottleneck, no batch norm and relu
    b = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(e7)
    b = Activation('relu')(b)
    # decoder model
    d1 = decoder_block(b, e7, 512)
    d2 = decoder_block(d1, e6, 512)
    d3 = decoder_block(d2, e5, 512)
    d4 = decoder_block(d3, e4, 512, dropout=False)
    d5 = decoder_block(d4, e3, 256, dropout=False)
    d6 = decoder_block(d5, e2, 128, dropout=False)
    d7 = decoder_block(d6, e1, 64, dropout=False)
    # output
    g = Conv2DTranspose(3, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d7)
    out_image = Activation('tanh')(g)
    # define model
    model = Model(in_image, out_image)
    return model
```

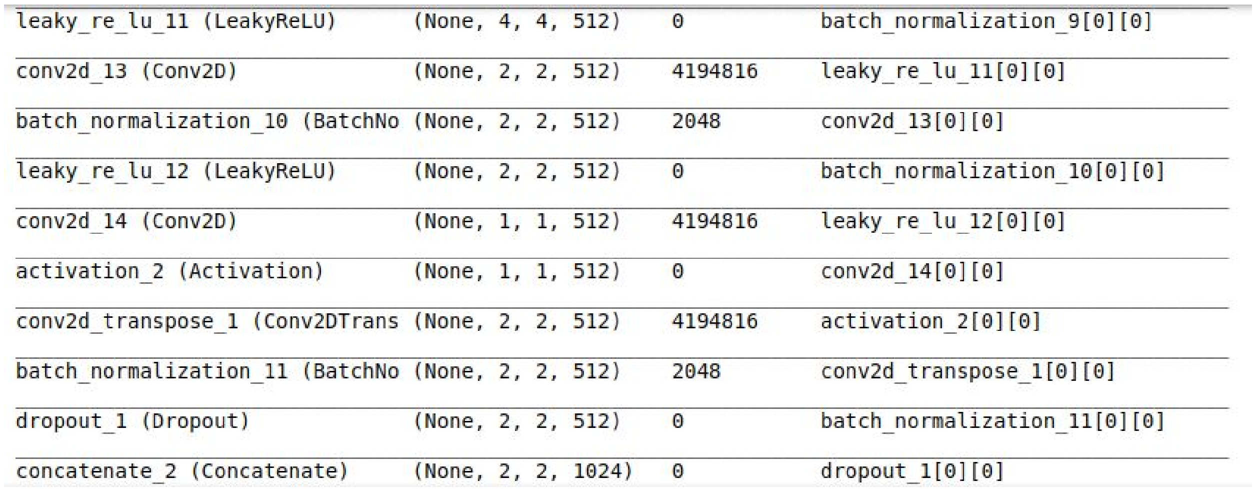
So the architecture of the Generator model is composed of 7 layers of encoders. Please note that initially we are doubling the number of filters applied from 64->128->256->512 to get that number of different output channels. By the end of the encoding part - we have 512 different outputs of size 1 * 1. So you have 512 learned parameters or important features at the end of encoding.

Now in the decoding part, we use 7 decoder block (note the number of filters is reverse of the order in the encoder part of the code i.e. 512->512->512->512->256->128->64 to get 256 * 256 NumPy array

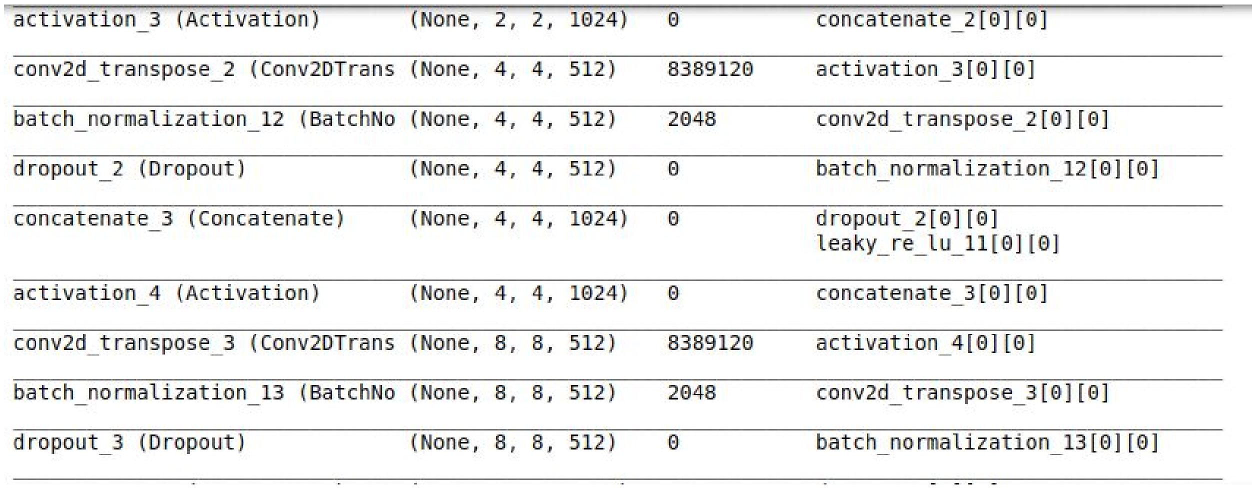


```
g_model = define_generator((256, 256, 3))
g_model.summary()
```

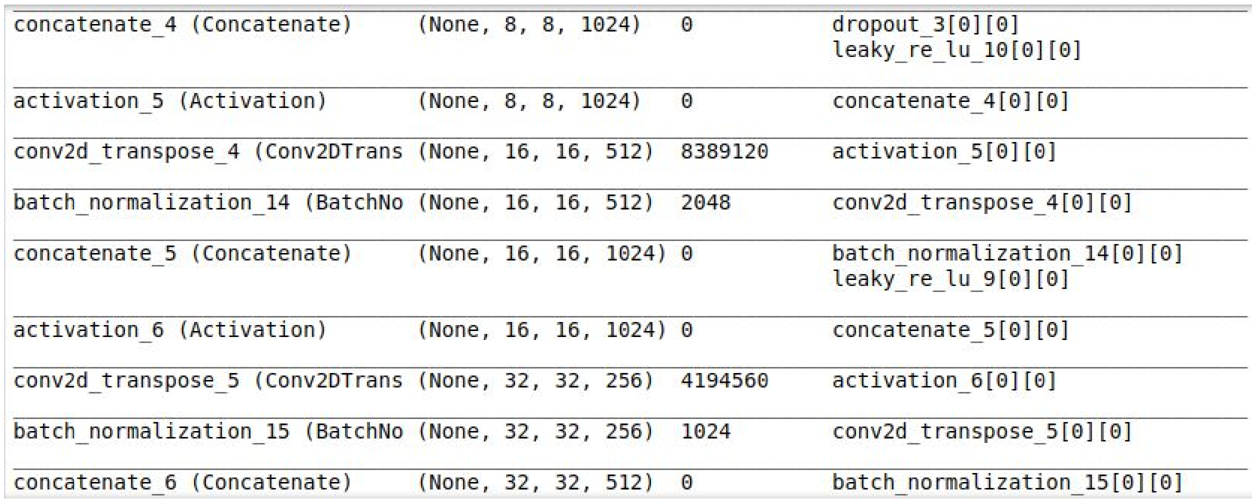
```
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.

Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            (None, 256, 256, 3)  0

conv2d_7 (Conv2D)               (None, 128, 128, 64) 3136        input_3[0][0]

leaky_re_lu_6 (LeakyReLU)       (None, 128, 128, 64) 0           conv2d_7[0][0]

conv2d_8 (Conv2D)               (None, 64, 64, 128)  131200      leaky_re_lu_6[0][0]

batch_normalization_5 (BatchNor (None, 64, 64, 128)  512         conv2d_8[0][0]

leaky_re_lu_7 (LeakyReLU)       (None, 64, 64, 128)  0           batch_normalization_5[0][0]

conv2d_9 (Conv2D)               (None, 32, 32, 256)  524544      leaky_re_lu_7[0][0]

batch_normalization_6 (BatchNor (None, 32, 32, 256)  1024        conv2d_9[0][0]

leaky_re_lu_8 (LeakyReLU)       (None, 32, 32, 256)  0           batch_normalization_6[0][0]

conv2d_10 (Conv2D)              (None, 16, 16, 512)  2097664     leaky_re_lu_8[0][0]

batch_normalization_7 (BatchNor (None, 16, 16, 512)  2048        conv2d_10[0][0]

leaky_re_lu_9 (LeakyReLU)       (None, 16, 16, 512)  0           batch_normalization_7[0][0]

conv2d_11 (Conv2D)              (None, 8, 8, 512)    4194816     leaky_re_lu_9[0][0]

batch_normalization_8 (BatchNor (None, 8, 8, 512)    2048        conv2d_11[0][0]

leaky_re_lu_10 (LeakyReLU)      (None, 8, 8, 512)    0           batch_normalization_8[0][0]

conv2d_12 (Conv2D)              (None, 4, 4, 512)    4194816     leaky_re_lu_10[0][0]

batch_normalization_9 (BatchNor (None, 4, 4, 512)    2048        conv2d_12[0][0]
```



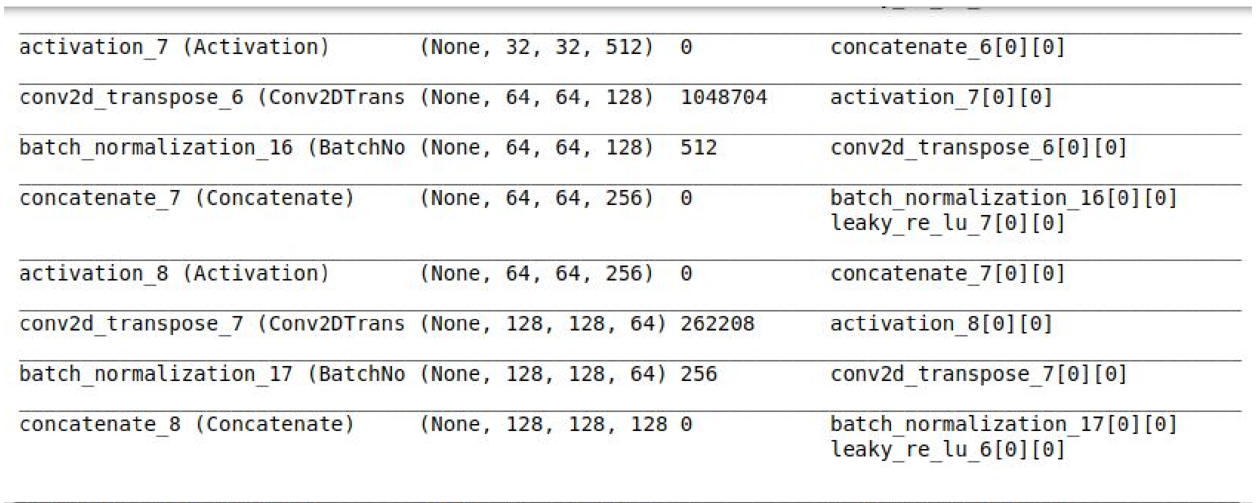
```
leaky_re_lu_11 (LeakyReLU)      (None, 4, 4, 512)    0           batch_normalization_9[0][0]

conv2d_13 (Conv2D)              (None, 2, 2, 512)    4194816     leaky_re_lu_11[0][0]

batch_normalization_10 (BatchNo (None, 2, 2, 512)    2048        conv2d_13[0][0]

leaky_re_lu_12 (LeakyReLU)      (None, 2, 2, 512)    0           batch_normalization_10[0][0]

conv2d_14 (Conv2D)              (None, 1, 1, 512)    4194816     leaky_re_lu_12[0][0]

activation_2 (Activation)       (None, 1, 1, 512)    0           conv2d_14[0][0]

conv2d_transpose_1 (Conv2DTrans (None, 2, 2, 512)    4194816     activation_2[0][0]

batch_normalization_11 (BatchNo (None, 2, 2, 512)    2048        conv2d_transpose_1[0][0]

dropout_1 (Dropout)             (None, 2, 2, 512)    0           batch_normalization_11[0][0]

concatenate_2 (Concatenate)     (None, 2, 2, 1024)   0           dropout_1[0][0]
```



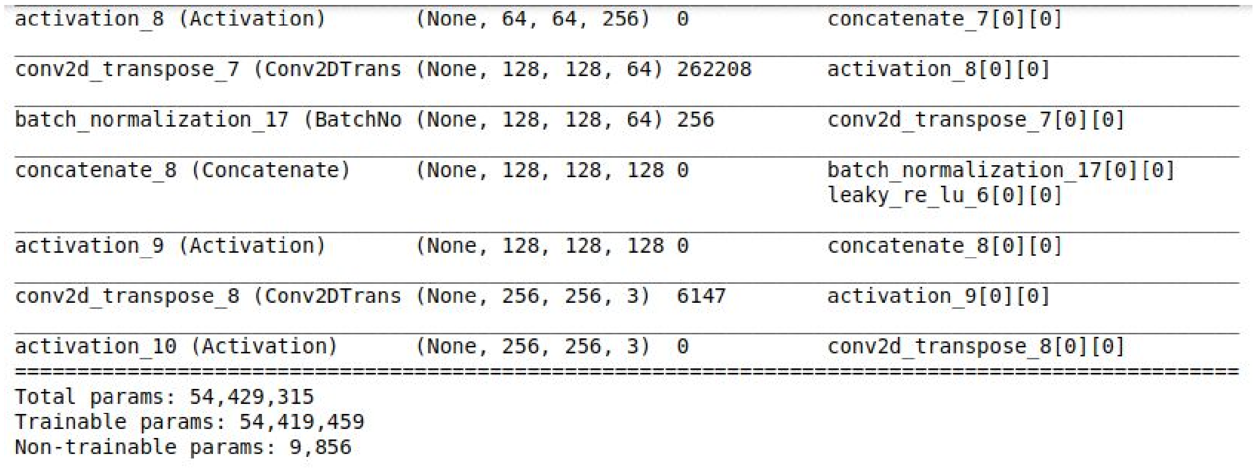
```
activation_3 (Activation)       (None, 2, 2, 1024)   0           concatenate_2[0][0]

conv2d_transpose_2 (Conv2DTrans (None, 4, 4, 512)    8389120     activation_3[0][0]

batch_normalization_12 (BatchNo (None, 4, 4, 512)    2048        conv2d_transpose_2[0][0]

dropout_2 (Dropout)             (None, 4, 4, 512)    0           batch_normalization_12[0][0]

concatenate_3 (Concatenate)     (None, 4, 4, 1024)   0           dropout_2[0][0]
                                                                 leaky_re_lu_11[0][0]

activation_4 (Activation)       (None, 4, 4, 1024)   0           concatenate_3[0][0]

conv2d_transpose_3 (Conv2DTrans (None, 8, 8, 512)    8389120     activation_4[0][0]

batch_normalization_13 (BatchNo (None, 8, 8, 512)    2048        conv2d_transpose_3[0][0]

dropout_3 (Dropout)             (None, 8, 8, 512)    0           batch_normalization_13[0][0]
```



```
concatenate_4 (Concatenate)     (None, 8, 8, 1024)   0           dropout_3[0][0]
                                                                 leaky_re_lu_10[0][0]

activation_5 (Activation)       (None, 8, 8, 1024)   0           concatenate_4[0][0]

conv2d_transpose_4 (Conv2DTrans (None, 16, 16, 512)  8389120     activation_5[0][0]

batch_normalization_14 (BatchNo (None, 16, 16, 512)  2048        conv2d_transpose_4[0][0]

concatenate_5 (Concatenate)     (None, 16, 16, 1024) 0           batch_normalization_14[0][0]
                                                                 leaky_re_lu_9[0][0]

activation_6 (Activation)       (None, 16, 16, 1024) 0           concatenate_5[0][0]

conv2d_transpose_5 (Conv2DTrans (None, 32, 32, 256)  4194560     activation_6[0][0]

batch_normalization_15 (BatchNo (None, 32, 32, 256)  1024        conv2d_transpose_5[0][0]

concatenate_6 (Concatenate)     (None, 32, 32, 512)  0           batch_normalization_15[0][0]
```

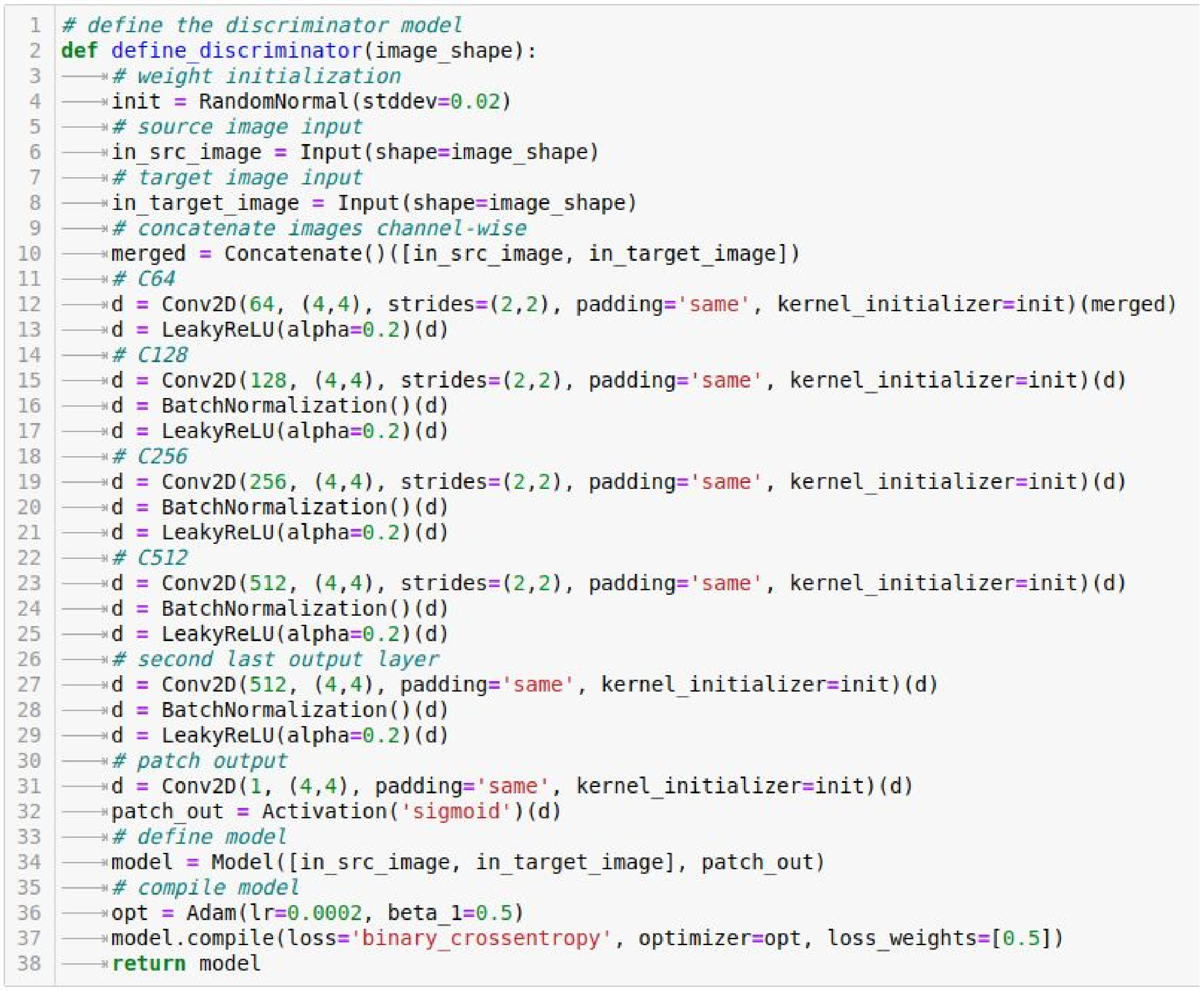


```
activation_7 (Activation)       (None, 32, 32, 512)  0           concatenate_6[0][0]
__________________________________________________________________________________________________
conv2d_transpose_6 (Conv2DTrans (None, 64, 64, 128)  1048704     activation_7[0][0]
__________________________________________________________________________________________________
batch_normalization_16 (BatchNo (None, 64, 64, 128)  512         conv2d_transpose_6[0][0]
__________________________________________________________________________________________________
concatenate_7 (Concatenate)     (None, 64, 64, 256)  0           batch_normalization_16[0][0]
                                                                 leaky_re_lu_7[0][0]
__________________________________________________________________________________________________
activation_8 (Activation)       (None, 64, 64, 256)  0           concatenate_7[0][0]
__________________________________________________________________________________________________
conv2d_transpose_7 (Conv2DTrans (None, 128, 128, 64) 262208      activation_8[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 128, 128, 64) 256         conv2d_transpose_7[0][0]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 128, 128, 128 0           batch_normalization_17[0][0]
                                                                 leaky_re_lu_6[0][0]
```



```
activation_8 (Activation)       (None, 64, 64, 256)  0           concatenate_7[0][0]
__________________________________________________________________________________________________
conv2d_transpose_7 (Conv2DTrans (None, 128, 128, 64) 262208      activation_8[0][0]
__________________________________________________________________________________________________
batch_normalization_17 (BatchNo (None, 128, 128, 64) 256         conv2d_transpose_7[0][0]
__________________________________________________________________________________________________
concatenate_8 (Concatenate)     (None, 128, 128, 128 0           batch_normalization_17[0][0]
                                                                 leaky_re_lu_6[0][0]
```

```
__________________________________________________________________________________________________
activation_9 (Activation)       (None, 128, 128, 128 0           concatenate_8[0][0]
__________________________________________________________________________________________________
conv2d_transpose_8 (Conv2DTrans (None, 256, 256, 3)  6147        activation_9[0][0]
__________________________________________________________________________________________________
activation_10 (Activation)      (None, 256, 256, 3)  0           conv2d_transpose_8[0][0]
==================================================================================================
Total params: 54,429,315
Trainable params: 54,419,459
Non-trainable params: 9,856
```

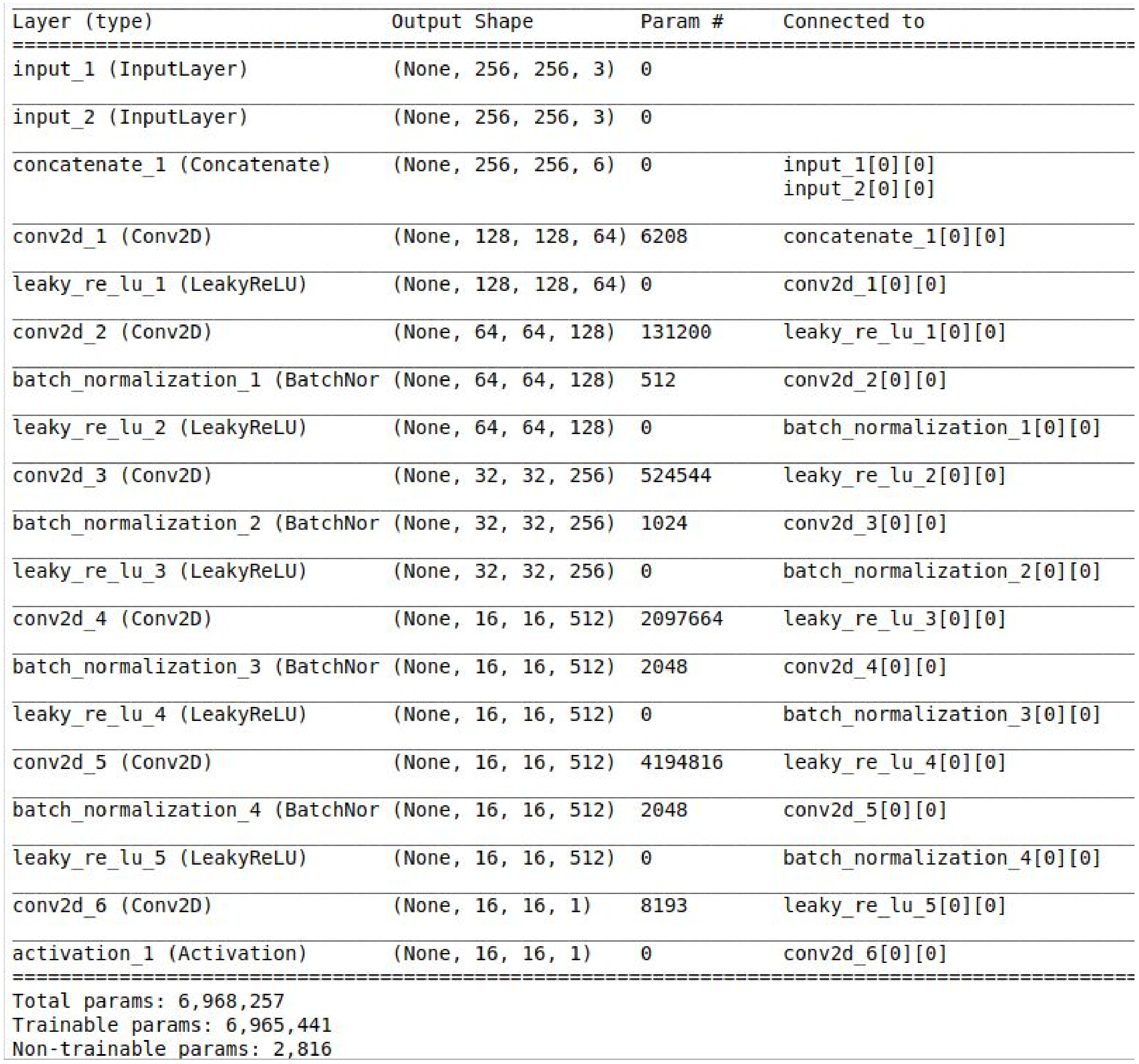


```
 1 # define the discriminator model
 2 def define_discriminator(image_shape):
 3     # weight initialization
 4     init = RandomNormal(stddev=0.02)
 5     # source image input
 6     in_src_image = Input(shape=image_shape)
 7     # target image input
 8     in_target_image = Input(shape=image_shape)
 9     # concatenate images channel-wise
10     merged = Concatenate()([in_src_image, in_target_image])
11     # C64
12     d = Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(merged)
13     d = LeakyReLU(alpha=0.2)(d)
14     # C128
15     d = Conv2D(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
16     d = BatchNormalization()(d)
17     d = LeakyReLU(alpha=0.2)(d)
18     # C256
19     d = Conv2D(256, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
20     d = BatchNormalization()(d)
21     d = LeakyReLU(alpha=0.2)(d)
22     # C512
23     d = Conv2D(512, (4,4), strides=(2,2), padding='same', kernel_initializer=init)(d)
24     d = BatchNormalization()(d)
25     d = LeakyReLU(alpha=0.2)(d)
26     # second last output layer
27     d = Conv2D(512, (4,4), padding='same', kernel_initializer=init)(d)
28     d = BatchNormalization()(d)
29     d = LeakyReLU(alpha=0.2)(d)
30     # patch output
31     d = Conv2D(1, (4,4), padding='same', kernel_initializer=init)(d)
32     patch_out = Activation('sigmoid')(d)
33     # define model
34     model = Model([in_src_image, in_target_image], patch_out)
35     # compile model
36     opt = Adam(lr=0.0002, beta_1=0.5)
37     model.compile(loss='binary_crossentropy', optimizer=opt, loss_weights=[0.5])
38     return model
```

In the discriminator model, we just run a few encoder blocks (conv2d -> Batch Normalization -> LeakyReLU activation) similar to the encoder block of the generator. Then we run an Adam optimizer and compute binary cross-entropy loss to detect how successfully our model is generating the image.

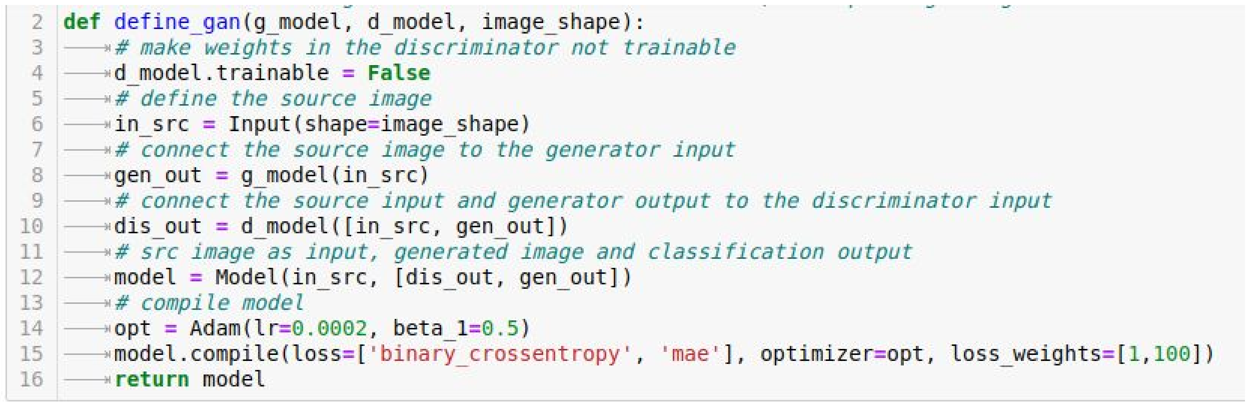
Please note how in line 34, we are passing in_src_image (inside the unified GAN model definition later, we will be passing the image that is being generated by the generator to this variable) and in_target_image (the real image of front face for comparison) to the model together.

Summary of the discriminator model -



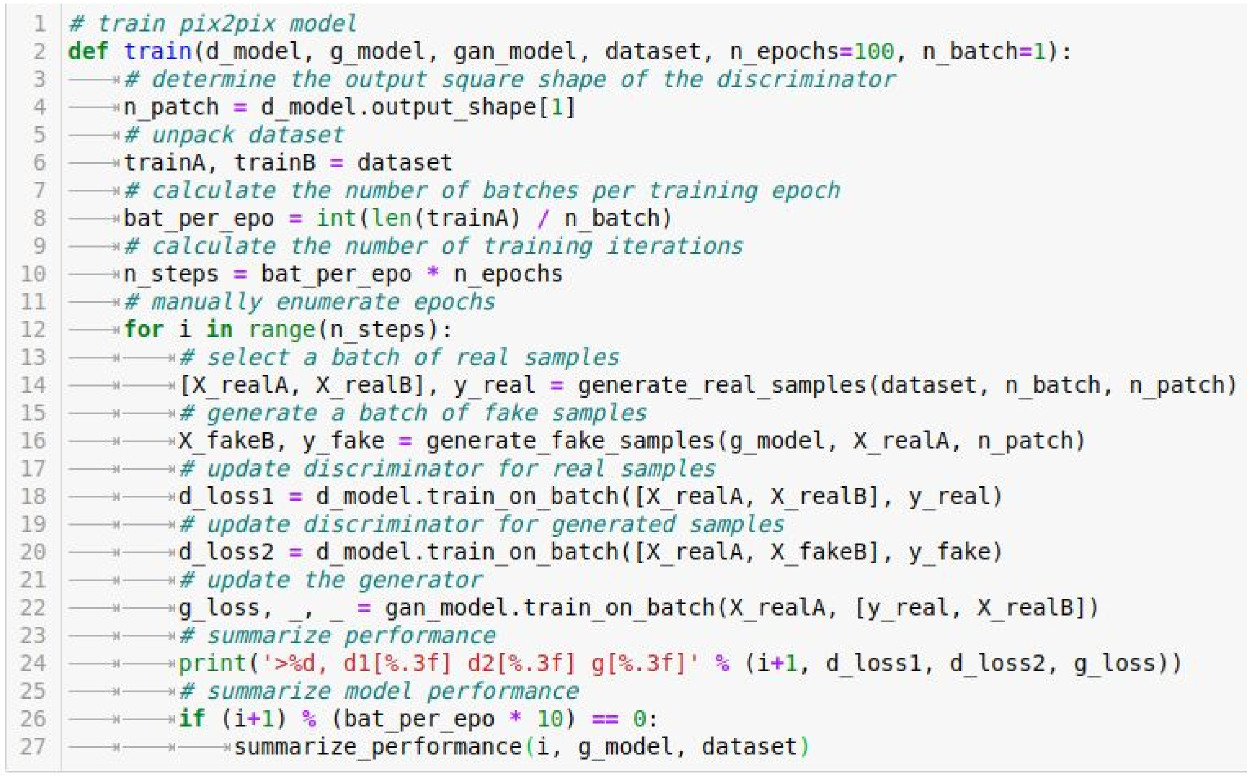
```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 256, 256, 3)  0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 256, 256, 3)  0
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 256, 256, 6)  0           input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 128, 128, 64) 6208        concatenate_1[0][0]
__________________________________________________________________________________________________
leaky_re_lu_1 (LeakyReLU)       (None, 128, 128, 64) 0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 64, 64, 128)  131200      leaky_re_lu_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 64, 64, 128)  512         conv2d_2[0][0]
__________________________________________________________________________________________________
leaky_re_lu_2 (LeakyReLU)       (None, 64, 64, 128)  0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 32, 32, 256)  524544      leaky_re_lu_2[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 32, 32, 256)  1024        conv2d_3[0][0]
__________________________________________________________________________________________________
leaky_re_lu_3 (LeakyReLU)       (None, 32, 32, 256)  0           batch_normalization_2[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 16, 16, 512)  2097664     leaky_re_lu_3[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 16, 16, 512)  2048        conv2d_4[0][0]
__________________________________________________________________________________________________
leaky_re_lu_4 (LeakyReLU)       (None, 16, 16, 512)  0           batch_normalization_3[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 16, 16, 512)  4194816     leaky_re_lu_4[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 16, 16, 512)  2048        conv2d_5[0][0]
__________________________________________________________________________________________________
leaky_re_lu_5 (LeakyReLU)       (None, 16, 16, 512)  0           batch_normalization_4[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 16, 16, 1)    8193        leaky_re_lu_5[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 16, 16, 1)    0           conv2d_6[0][0]
==================================================================================================
Total params: 6,968,257
Trainable params: 6,965,441
Non-trainable params: 2,816
__________________________________________________________________________________________________
```

Once we have created the generator and discriminator model, we can now then unify it into a single unit called GAN.



```python
def define_gan(g_model, d_model, image_shape):
	# make weights in the discriminator not trainable
	d_model.trainable = False
	# define the source image
	in_src = Input(shape=image_shape)
	# connect the source image to the generator input
	gen_out = g_model(in_src)
	# connect the source input and generator output to the discriminator input
	dis_out = d_model([in_src, gen_out])
	# src image as input, generated image and classification output
	model = Model(in_src, [dis_out, gen_out])
	# compile model
	opt = Adam(lr=0.0002, beta_1=0.5)
	model.compile(loss=['binary_crossentropy', 'mae'], optimizer=opt, loss_weights=[1,100])
	return model
```

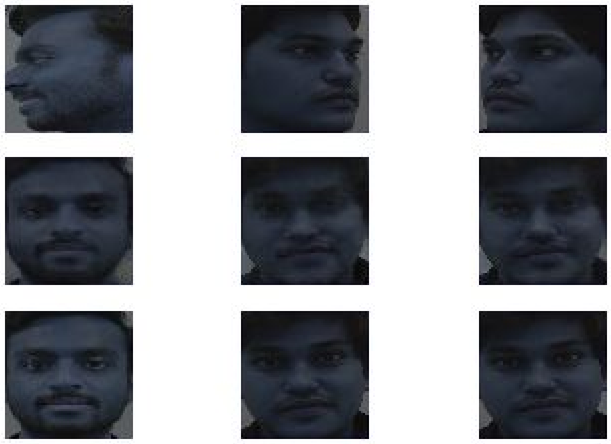
In the above model, we are using Adam optimizer. For batch size is 1 since we want our model to learn after working with each image pair. Binary Crossentropy loss is used to measure how well our model is differentiating between real and false images. While ‘mae’ loss parameter is a pixel-wise loss between pixels of the generated image and the target image. This is the modification that allows the model to learn to create the face of the same person.



```python
# train pix2pix model
def train(d_model, g_model, gan_model, dataset, n_epochs=100, n_batch=1):
	# determine the output square shape of the discriminator
	n_patch = d_model.output_shape[1]
	# unpack dataset
	trainA, trainB = dataset
	# calculate the number of batches per training epoch
	bat_per_epo = int(len(trainA) / n_batch)
	# calculate the number of training iterations
	n_steps = bat_per_epo * n_epochs
	# manually enumerate epochs
	for i in range(n_steps):
		# select a batch of real samples
		[X_realA, X_realB], y_real = generate_real_samples(dataset, n_batch, n_patch)
		# generate a batch of fake samples
		X_fakeB, y_fake = generate_fake_samples(g_model, X_realA, n_patch)
		# update discriminator for real samples
		d_loss1 = d_model.train_on_batch([X_realA, X_realB], y_real)
		# update discriminator for generated samples
		d_loss2 = d_model.train_on_batch([X_realA, X_fakeB], y_fake)
		# update the generator
		g_loss, _, _ = gan_model.train_on_batch(X_realA, [y_real, X_realB])
		# summarize performance
		print('>%d, d1[%.3f] d2[%.3f] g[%.3f]' % (i+1, d_loss1, d_loss2, g_loss))
		# summarize model performance
		if (i+1) % (bat_per_epo * 10) == 0:
			summarize_performance(i, g_model, dataset)
```

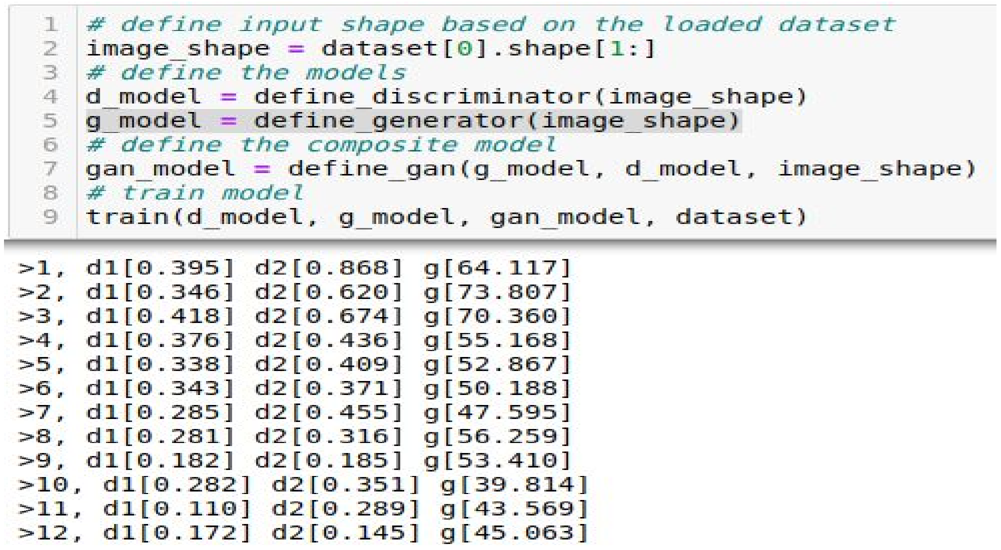
Now we run the model to train through the above function.
Some functions used in the code -

1. Generate_real_samples - take the real face images
2. Generate_fake_samples - uses the generator model's predict function to generate fake images to send to the discriminator
3. summarize _performance - saves the model and the pyplot of random three samples - three images for each sample side face, generated face by the model, the real face that was the target.



An example of the pyplot saved by the function - the middle images are generated by our model.

We run the model for the 95 image pairs



```
# define input shape based on the loaded dataset
image_shape = dataset[0].shape[1:]
# define the models
d_model = define_discriminator(image_shape)
g_model = define_generator(image_shape)
# define the composite model
gan_model = define_gan(g_model, d_model, image_shape)
# train model
train(d_model, g_model, gan_model, dataset)
```

```
>1, d1[0.395] d2[0.868] g[64.117]
>2, d1[0.346] d2[0.620] g[73.807]
>3, d1[0.418] d2[0.674] g[70.360]
>4, d1[0.376] d2[0.436] g[55.168]
>5, d1[0.338] d2[0.409] g[52.867]
>6, d1[0.343] d2[0.371] g[50.188]
>7, d1[0.285] d2[0.455] g[47.595]
>8, d1[0.281] d2[0.316] g[56.259]
>9, d1[0.182] d2[0.185] g[53.410]
>10, d1[0.282] d2[0.351] g[39.814]
>11, d1[0.110] d2[0.289] g[43.569]
>12, d1[0.172] d2[0.145] g[45.063]
```

In the above image, we can see the three-loss values -
d1 - this the binary cross-entropy loss of the discriminator
d2 - this the ‘mae’ loss for the discriminator
g - this the loss of the generator

Output at the 100th epoch for our model is shown below - the middle image is the generated image by our GAN model. As you can see - for data in the training set

Face Recognition

Face recognition is the task of identifying an already detected object as a known or unknown face.
To build our face recognition system, we'll first perform face detection, extract face embeddings from each face using deep learning, train a face recognition model on the embeddings, and then finally recognize faces in both images and video streams with OpenCV.

OpenCV Face Recognition

Well, keep in mind that the dlib face recognition post relied on two important external libraries:

1. dlib (obviously)
2. face_recognition (which is easy to use set of face recognition utilities that wrap around dlib)

While we used OpenCV to *facilitate* face recognition, OpenCV *itself* was not responsible for identifying faces.

- we'll learn how we can apply deep learning and OpenCV together (with no other libraries other than sci-kit-learn) to:

1. Detect faces
2. Compute 128-d face embeddings to quantify a face
3. Train a Support Vector Machine (SVM) on top of the embeddings
4. Recognize faces in images and video streams

All of these tasks will be accomplished with OpenCV, enabling us to obtain a "pure" OpenCV face recognition pipeline.

- In order to build our OpenCV face recognition pipeline, we'll be applying deep learning in two key steps:

1. To apply *face detection*, which detects the *presence* and location of a face in an image, but does not identify it.
2. To extract the 128-d feature vectors (called "embeddings") that *quantify* each face in an image.

- First, we input an image or video frame to our face recognition pipeline. Given the input image, we apply face detection to detect the location of a face in the image.

Face alignment, as the name suggests, is the process of (1) identifying the geometric structure of the faces and (2) attempting to obtain a canonical alignment of the face based on translation, rotation, and scale.

While optional, face alignment has been demonstrated to increase face recognition accuracy in some pipelines.

- To train a face recognition model with deep learning, each input batch of data includes three images:

1. The *anchor*
2. The *positive* image
3. The *negative* image

The anchor is our current face and has identity *A*.

The second image is our positive image — this image also contains a face of person *A*.

The negative image, on the other hand, *does not have the same identity*, and could belong to person *B*, *C*, or even *Y*!

The point is that the anchor and positive image both belong to the same person/face while the negative image does not contain the same face.

- The neural network computes the 128-d embeddings for each face and then tweaks the weights of the network (via the triplet loss function) such that:

1. The 128-d embeddings of the anchor and positive image lie closer together
2. While at the same time, pushing the embeddings for the negative image father away

3. In this manner, the network is able to learn to quantify faces and return highly robust and discriminating embeddings suitable for face recognition.

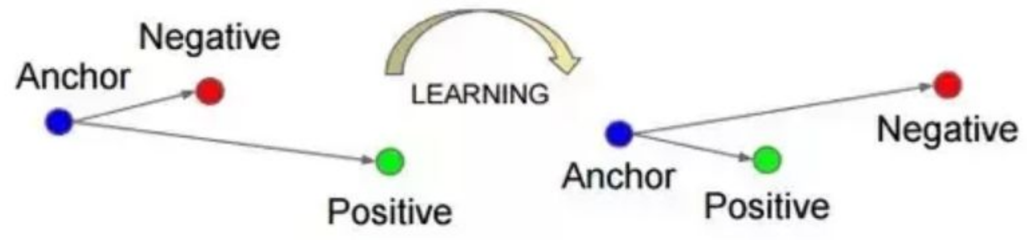


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

Our face recognition dataset

The dataset we are using today contains three people:

- Dataset consists of some of our classmates and teachers.
- Each class contains a total of six images.
- Images are different angles. To get more accuracy.

- “Unknown”, which is used to represent faces of people we do not know.

Method used

1. **face_recognition.api.face_encodings**(*face_image*, *known_face_locations=None*, *num_jitters=1*, *model='small'*)

Given an image, return the 128-dimension face encoding for each face in the image.

Parameters:

- **face_image** – The image that contains one or more faces

- **known_face_locations** – Optional - the bounding boxes of each face if you already know them.
- **num_jitters** – How many times to re-sample the face when calculating encoding. Higher is more accurate, but slower (i.e. 100 is 100x slower)
- **model** – Optional - which model to use. “large” (default) or “small” which only returns 5 points but is faster.

Returns:
A list of 128-dimensional face encodings (one for each face in the image)

2. **face_recognition.api.face_distance(*face_encodings, face_to_compare*)**

Given a list of face encodings, compare them to a known face encoding and get a Euclidean distance for each comparison face. The distance tells you how similar the faces are.

Parameters:

- **faces** – List of face encodings to compare
- **face_to_compare** – A face encoding to compare against

Returns:

A NumPy array with the distance for each face in the same order as the ‘faces’ array.

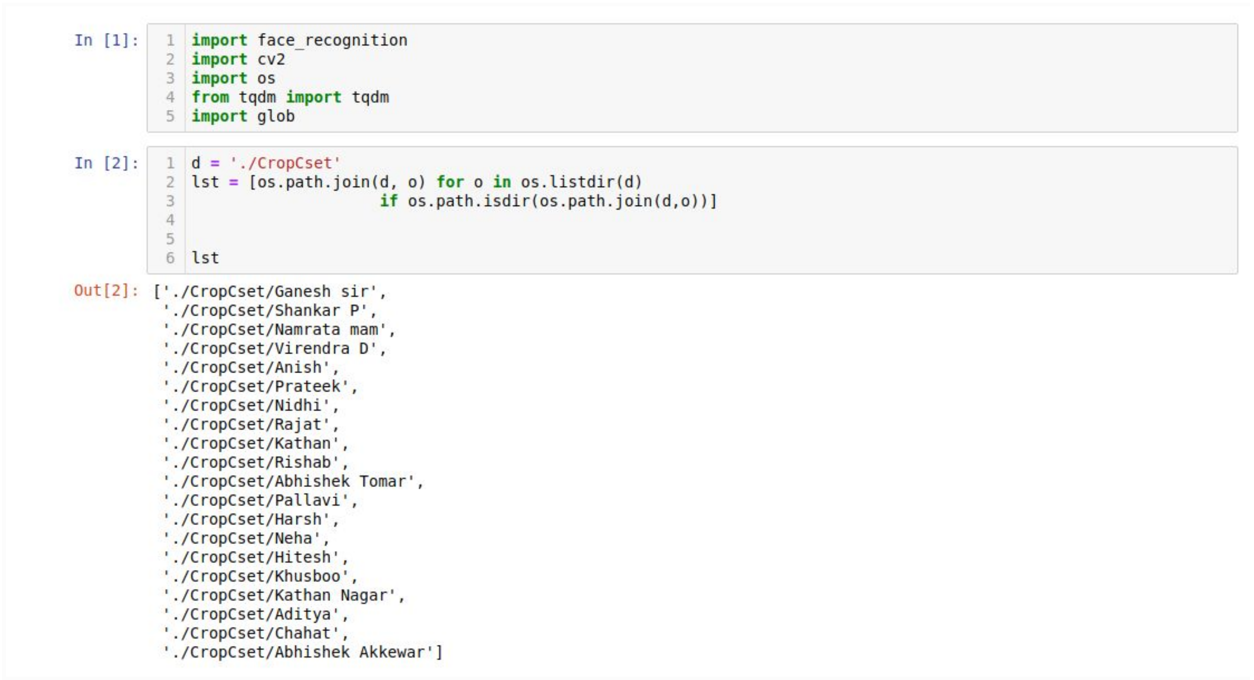
3. **face_recognition.api.compare_faces(*known_face_encodings, face_encoding_to_check, tolerance=0.6*)**

Compare a list of face encodings against a candidate encoding to see if they match.

Parameters:

- **known_face_encodings** – A list of known face encodings
- **face_encoding_to_check** – A single face encoding to compare against the list
- **tolerance** – How much distance between faces to consider it a match. Lower is more strict. 0.6 is the typical best performance.

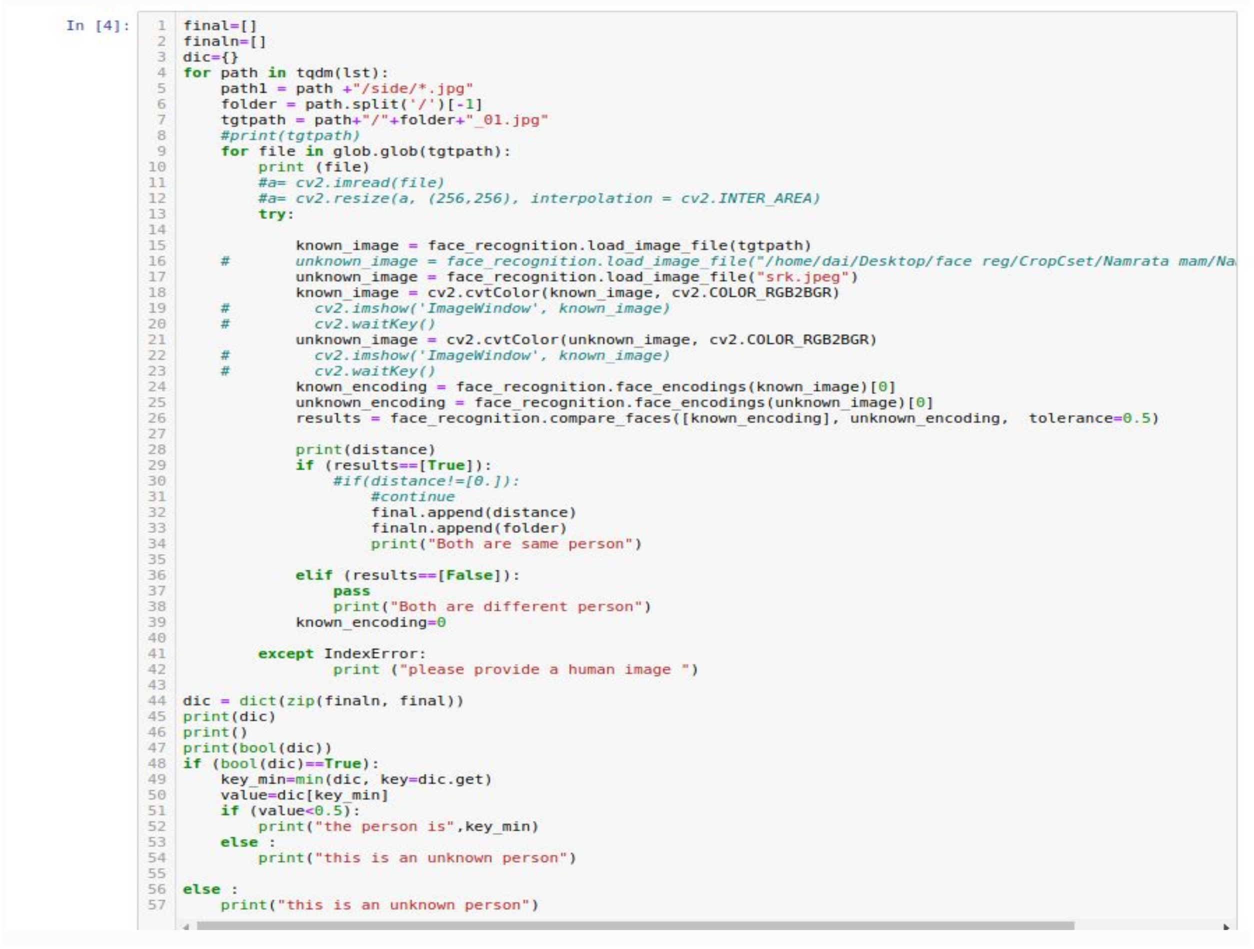
Returns:
A list of True/False values indicating which known_face_encodings match the face encoding to check.



```
In [1]:
import face_recognition
import cv2
import os
from tqdm import tqdm
import glob
```

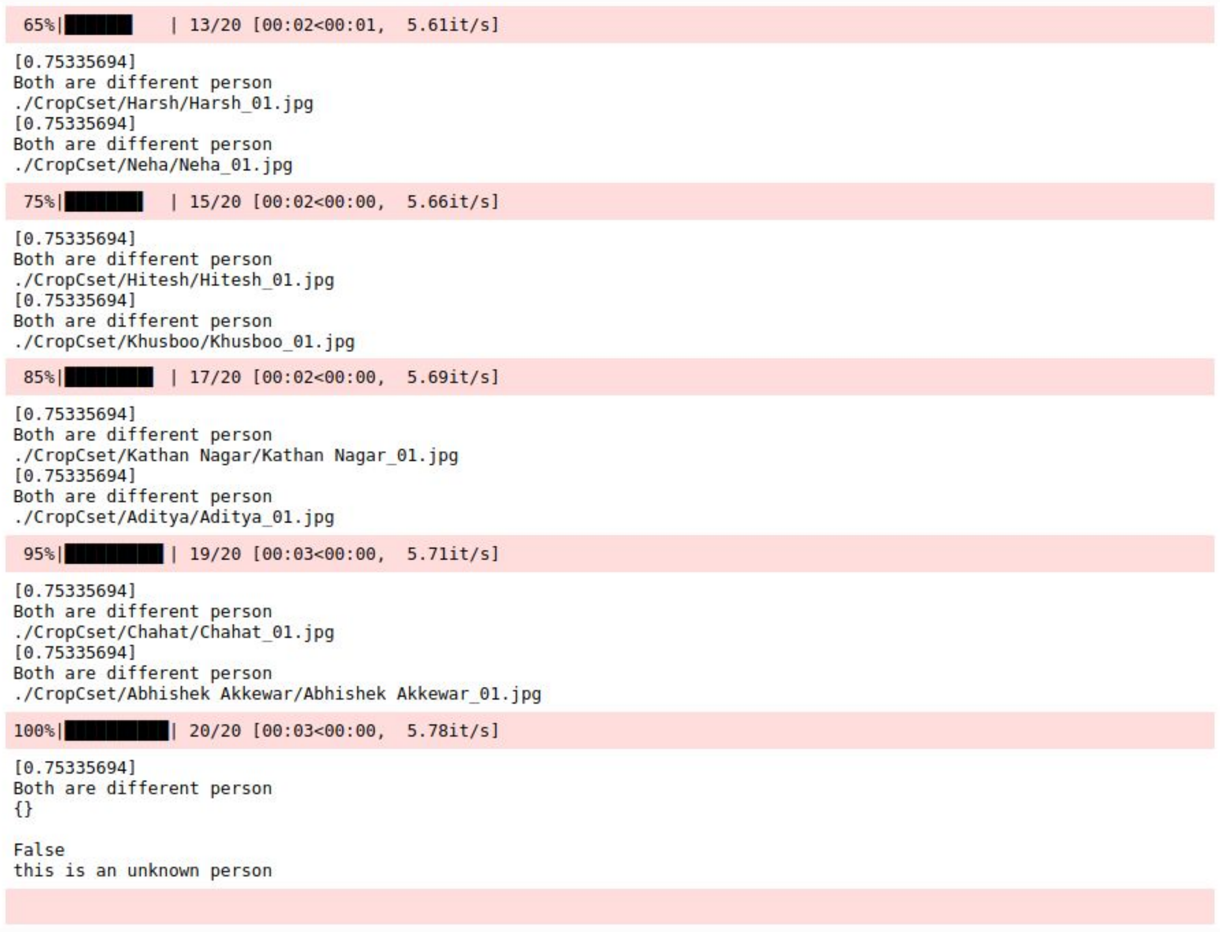
```
In [2]:
d = './CropCset'
lst = [os.path.join(d, o) for o in os.listdir(d)
                    if os.path.isdir(os.path.join(d,o))]


lst
```

```
Out[2]: ['./CropCset/Ganesh sir',
 './CropCset/Shankar P',
 './CropCset/Namrata mam',
 './CropCset/Virendra D',
 './CropCset/Anish',
 './CropCset/Prateek',
 './CropCset/Nidhi',
 './CropCset/Rajat',
 './CropCset/Kathan',
 './CropCset/Rishab',
 './CropCset/Abhishek Tomar',
 './CropCset/Pallavi',
 './CropCset/Harsh',
 './CropCset/Neha',
 './CropCset/Hitesh',
 './CropCset/Khusboo',
 './CropCset/Kathan Nagar',
 './CropCset/Aditya',
 './CropCset/Chahat',
 './CropCset/Abhishek Akkewar']
```

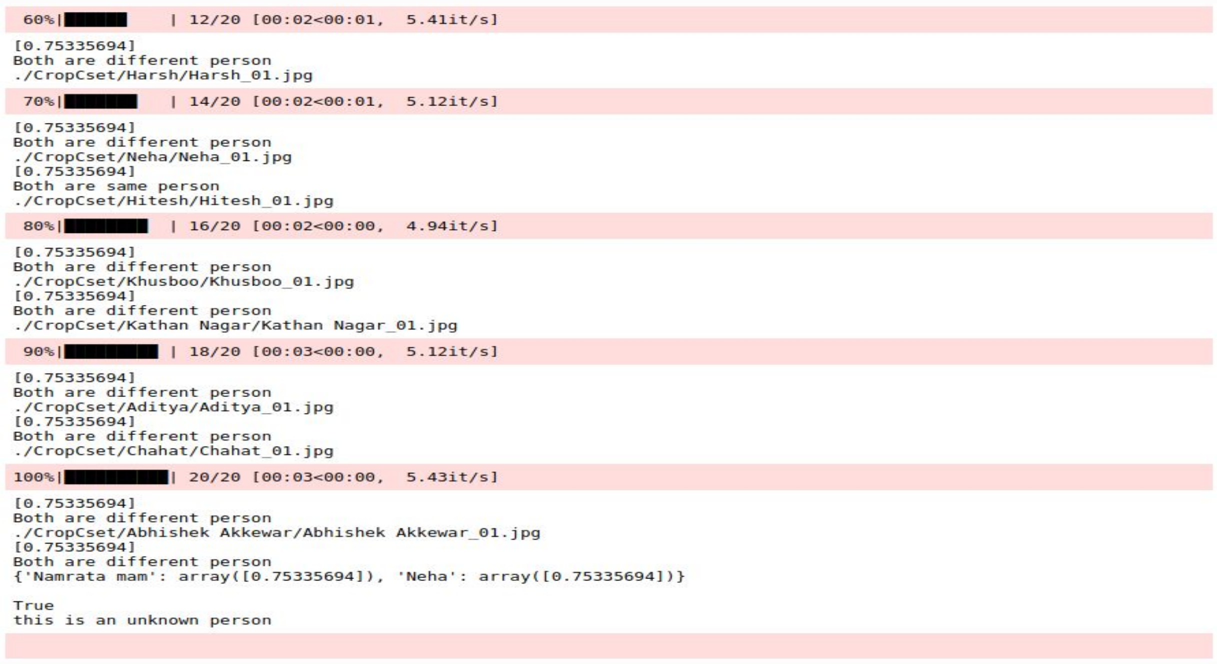


```
In [4]:
final=[]
finaln=[]
dic={}
for path in tqdm(lst):
    path1 = path +"/side/*.jpg"
    folder = path.split('/')[-1]
    tgtpath = path+"/"+folder+"_01.jpg"
    #print(tgtpath)
    for file in glob.glob(tgtpath):
        print (file)
        #a= cv2.imread(file)
        #a= cv2.resize(a, (256,256), interpolation = cv2.INTER_AREA)
        try:

            known_image = face_recognition.load_image_file(tgtpath)
    #       unknown_image = face_recognition.load_image_file("/home/dai/Desktop/face reg/CropCset/Namrata mam/Na
            unknown_image = face_recognition.load_image_file("srk.jpeg")
            known_image = cv2.cvtColor(known_image, cv2.COLOR_RGB2BGR)
    #         cv2.imshow('ImageWindow', known_image)
    #         cv2.waitKey()
            unknown_image = cv2.cvtColor(unknown_image, cv2.COLOR_RGB2BGR)
    #         cv2.imshow('ImageWindow', known_image)
    #         cv2.waitKey()
            known_encoding = face_recognition.face_encodings(known_image)[0]
            unknown_encoding = face_recognition.face_encodings(unknown_image)[0]
            results = face_recognition.compare_faces([known_encoding], unknown_encoding,  tolerance=0.5)

            print(distance)
            if (results==[True]):
                #if(distance!=[0.]):
                    #continue
                    final.append(distance)
                    finaln.append(folder)
                    print("Both are same person")

            elif (results==[False]):
                pass
                print("Both are different person")
            known_encoding=0

        except IndexError:
                print ("please provide a human image ")

dic = dict(zip(finaln, final))
print(dic)
print()
print(bool(dic))
if (bool(dic)==True):
    key_min=min(dic, key=dic.get)
    value=dic[key_min]
    if (value<0.5):
        print("the person is",key_min)
    else :
        print("this is an unknown person")

else :
    print("this is an unknown person")
```

Outputs



```
 65%|██████    | 13/20 [00:02<00:01,  5.61it/s]
[0.75335694]
Both are different person
./CropCset/Harsh/Harsh_01.jpg
[0.75335694]
Both are different person
./CropCset/Neha/Neha_01.jpg
 75%|███████▌  | 15/20 [00:02<00:00,  5.66it/s]
[0.75335694]
Both are different person
./CropCset/Hitesh/Hitesh_01.jpg
[0.75335694]
Both are different person
./CropCset/Khusboo/Khusboo_01.jpg
 85%|████████▌ | 17/20 [00:02<00:00,  5.69it/s]
[0.75335694]
Both are different person
./CropCset/Kathan Nagar/Kathan Nagar_01.jpg
[0.75335694]
Both are different person
./CropCset/Aditya/Aditya_01.jpg
 95%|█████████▌| 19/20 [00:03<00:00,  5.71it/s]
[0.75335694]
Both are different person
./CropCset/Chahat/Chahat_01.jpg
[0.75335694]
Both are different person
./CropCset/Abhishek Akkewar/Abhishek Akkewar_01.jpg
100%|██████████| 20/20 [00:03<00:00,  5.78it/s]
[0.75335694]
Both are different person
{}

False
this is an unknown person
```



```
 60%|██████    | 12/20 [00:02<00:01,  5.41it/s]
[0.75335694]
Both are different person
./CropCset/Harsh/Harsh_01.jpg
 70%|███████   | 14/20 [00:02<00:01,  5.12it/s]
[0.75335694]
Both are different person
./CropCset/Neha/Neha_01.jpg
[0.75335694]
Both are same person
./CropCset/Hitesh/Hitesh_01.jpg
 80%|████████  | 16/20 [00:02<00:00,  4.94it/s]
[0.75335694]
Both are different person
./CropCset/Khusboo/Khusboo_01.jpg
[0.75335694]
Both are different person
./CropCset/Kathan Nagar/Kathan Nagar_01.jpg
 90%|█████████ | 18/20 [00:03<00:00,  5.12it/s]
[0.75335694]
Both are different person
./CropCset/Aditya/Aditya_01.jpg
[0.75335694]
Both are different person
./CropCset/Chahat/Chahat_01.jpg
100%|██████████| 20/20 [00:03<00:00,  5.43it/s]
[0.75335694]
Both are different person
./CropCset/Abhishek Akkewar/Abhishek Akkewar_01.jpg
[0.75335694]
Both are different person
{'Namrata mam': array([0.75335694]), 'Neha': array([0.75335694])}

True
this is an unknown person
```

<u>Check this index</u>

1.4.1 Face Detection
- Find faces in a photograph
- Find faces in a photograph (using deep learning)
- Find faces in batches of images w/ GPU (using deep learning)

1.4.2 Facial Features
- Identify specific facial features in a photograph
- Apply (horribly ugly) digital make-up

1.4.3 Facial Recognition
- Find and recognize unknown faces in a photograph based on photographs of known people
- Compare faces by numeric face distance instead of only True/False matches
- Recognize faces in live video using your webcam - Simple / Slower Version (Requires OpenCV to be installed)
- Recognize faces in live video using your webcam - Faster Version (Requires OpenCV to be installed)
- Recognize faces in a video file and write out new video file (Requires OpenCV to be installed)
- Recognize faces on a Raspberry Pi w/ camera
- Run a web service to recognize faces via HTTP (Requires Flask to be installed) • Recognize faces with a K-nearest neighbors classifier