Face Recognition using Keras and OpenCV

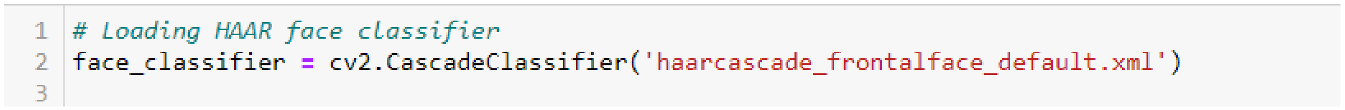
Step 1:

Collection of Images:

<u>Haar Classifier:</u>

The face_collection file is a face capture automation that uses **haarcascade_frontalface_default.xml** that captures the front portion of the face only and no side faces.

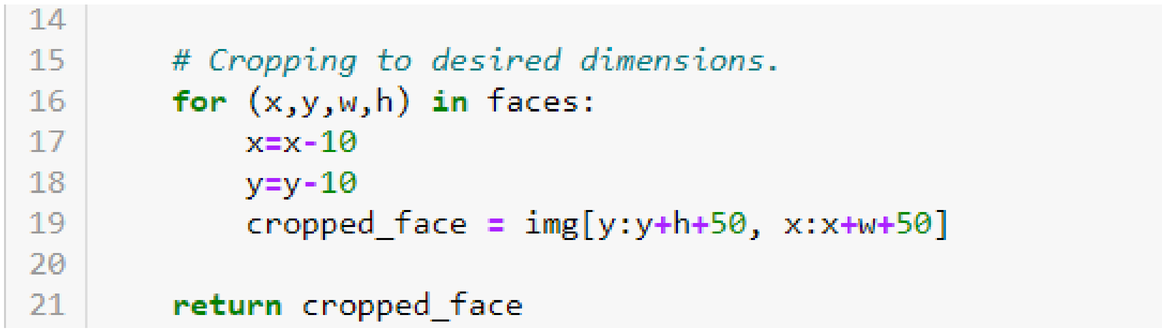
haarcascade_frontalface_default.xml this is a pre-trained classifier provided by OpenCv for capturing front faces, **it does not capture side-face.**



```python
# Loading HAAR face classifier
face_classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

```

Cropping the faces:

After the Haar cascade is loaded then the next step is to crop the images on the camera frame to the desired dimension. The dimensions can be adjusted in the code itself. If the images are in the same dimensions then it is easy for a deep neural network to perform a smooth operation.

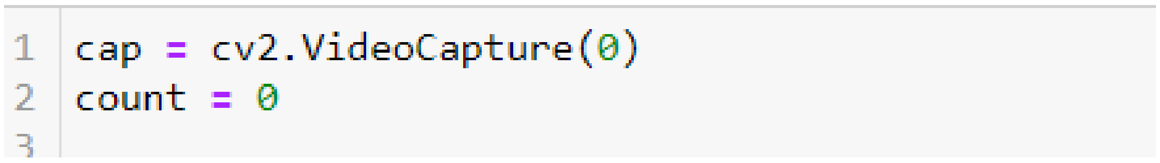


```python

    # Cropping to desired dimensions.
    for (x,y,w,h) in faces:
        x=x-10
        y=y-10
        cropped_face = img[y:y+h+50, x:x+w+50]

    return cropped_face
```

Notations: [y:y+h] → (**start** coordinate of **y** and **end** coordinate of **y**)
[x:x+w]→ (**start** coordinate of **x** and **end** coordinate of **x**)

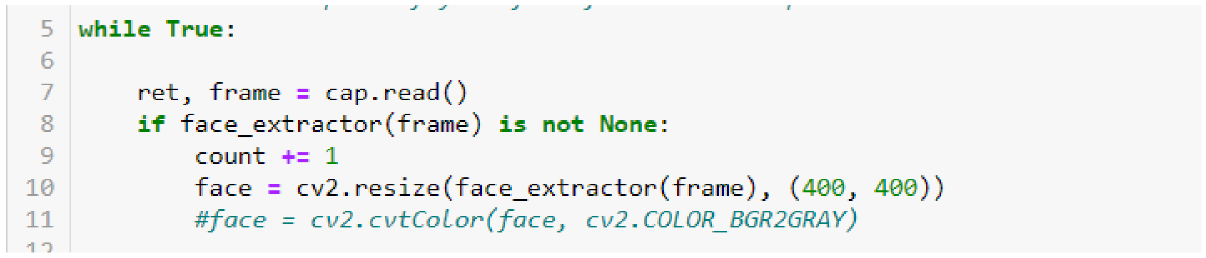
Initialization of the webcam:

To initialize the web cam we use an opencv command. cv2.VideoCapture(0), here (0) denoted the default camera of the system.



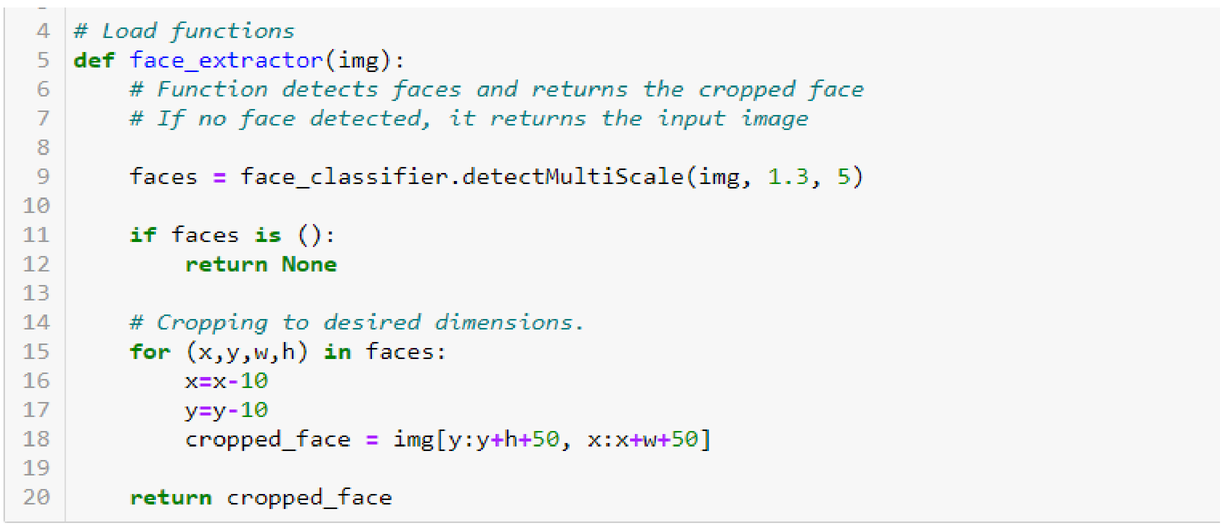
```python
cap = cv2.VideoCapture(0)
count = 0
```

After initialization the webcam starts to capture the (frame), these frames are then resized before storing it in the disk. For resizing the frame a function is defined which takes the frames captured from the camera in real time and then resize them according to the given dimensions as said above.



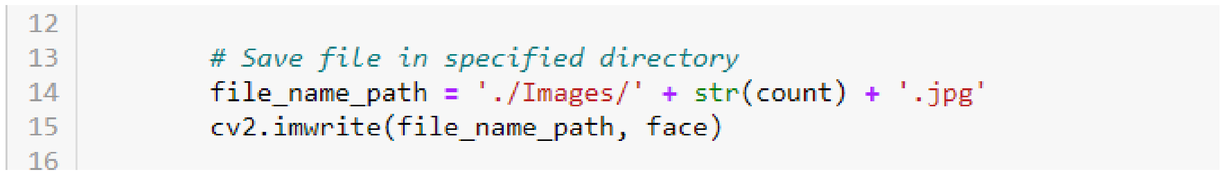
```python
while True:

    ret, frame = cap.read()
    if face_extractor(frame) is not None:
        count += 1
        face = cv2.resize(face_extractor(frame), (400, 400))
        #face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
```

In the above snippet the captured frame is passed to the face_extractor function, where resizing takes place.



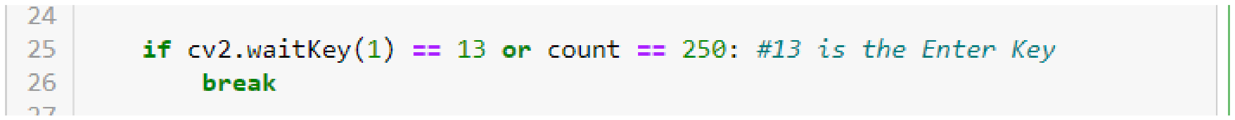
```python
# Load functions
def face_extractor(img):
    # Function detects faces and returns the cropped face
    # If no face detected, it returns the input image

    faces = face_classifier.detectMultiScale(img, 1.3, 5)

    if faces is ():
        return None

    # Cropping to desired dimensions.
    for (x,y,w,h) in faces:
        x=x-10
        y=y-10
        cropped_face = img[y:y+h+50, x:x+w+50]

    return cropped_face
```

Once cropped, the image gets stored in an image folder, that is present in the working directory in jpg format.



```
        # Save file in specified directory
        file_name_path = './Images/' + str(count) + '.jpg'
        cv2.imwrite(file_name_path, face)
```

In the below snippet we have defined the count of the frames that we want to capture. In this case 250.



```
    if cv2.waitKey(1) == 13 or count == 250: #13 is the Enter Key
        break
```

As we run the code the webcam is activated and it starts to capture images based in the count and resize each one of them and stores them.

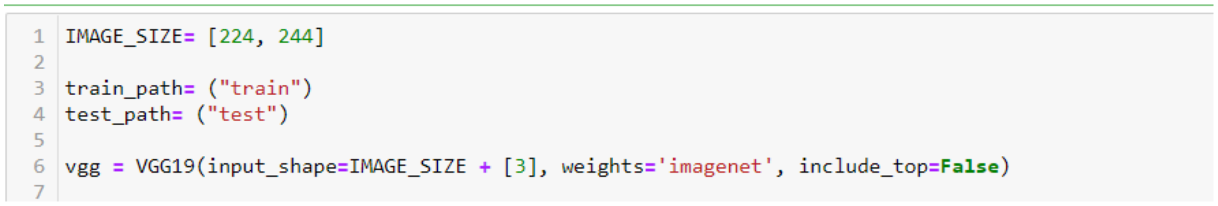
Out of the captured images the images need to be split into train and test folder. The images of different people in different folders should be stored with their names so that it can be used to train our deep learning network which in future will be able to recognize these images.

Step 2:
Building and training the model:

Here we are using keras to build the model and we are using transfer learning to train it. Keras provides various different pre-trained models for prediction and feature extraction. It has several different models with different accuracies and for different purposes.
Example: VGG16, VGG19, ResNet50, ResNet101, etc

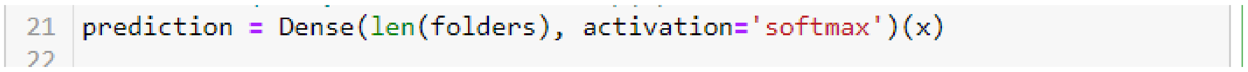
We have used **VGG16** for our model. We can use other pre-trained models as well.
VGG16 is a model that is trained on **ImageNet dataset** which is collection of 15 million high quality images belonging to different categories and is used in various image classification problems.

→The image size for our network to accept is set at [224,224]
→We then provide the path to train and test folder.
→Then the VGG16 is initialized and since we are using colored images we define input shape as [224,224,3], where 3 stands for RGB layers.
→The weights have been initialized to **'imgenet'** weights.



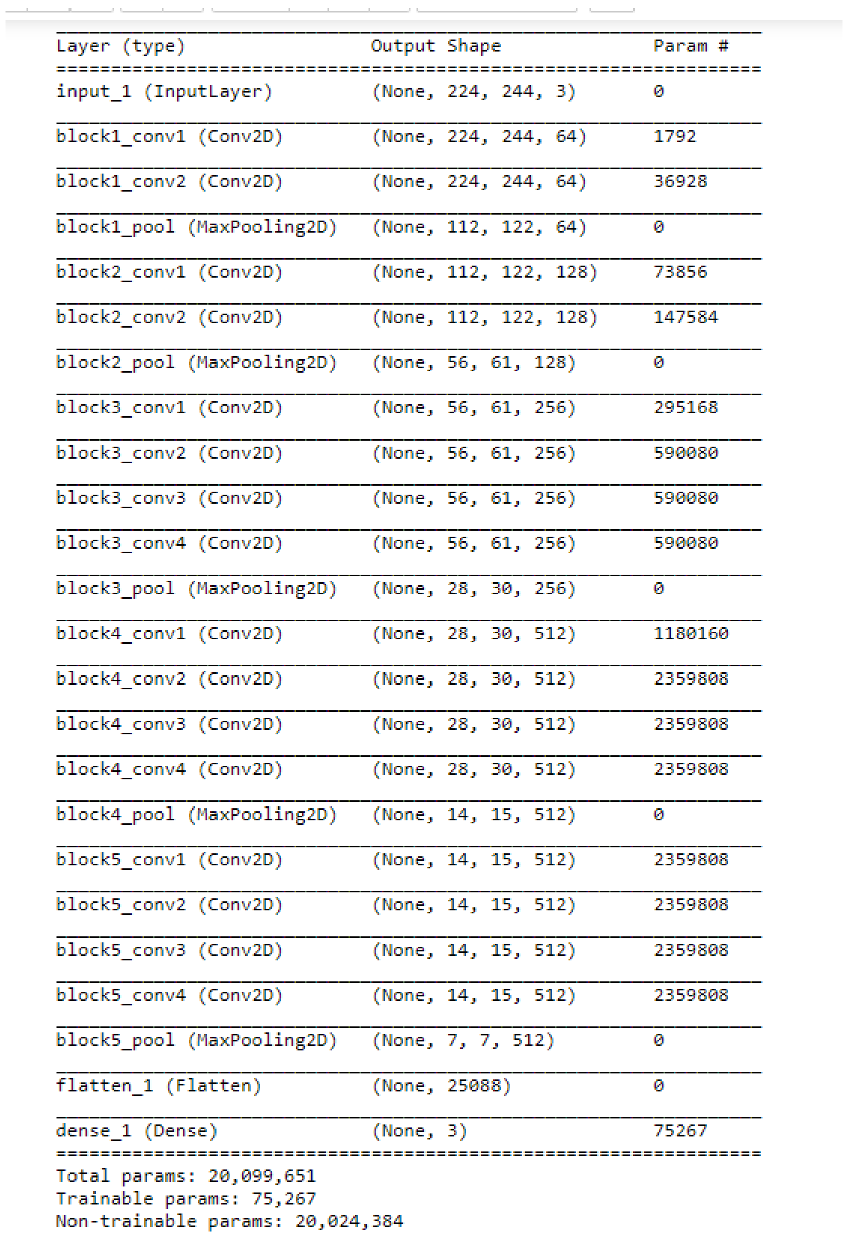
```
IMAGE_SIZE= [224, 244]

train_path= ("train")
test_path= ("test")

vgg = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

```

- ➔ We then define all the parameters of the last layer of the network, it then takes all the different classes that we want it to predict.
- ➔ Here we use activation function as **softmax** function which ranges from 0 to 1 and classified based on the probability.



```
prediction = Dense(len(folders), activation='softmax')(x)
```

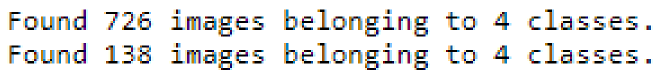
Below is my model summary:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         (None, 224, 244, 3)       0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 244, 64)      1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 244, 64)      36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 122, 64)      0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 122, 128)     73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 122, 128)     147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 61, 128)       0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 61, 256)       295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 61, 256)       590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 61, 256)       590080    
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 61, 256)       590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 30, 256)       0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 30, 512)       1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 30, 512)       2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 30, 512)       2359808   
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 30, 512)       2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 15, 512)       0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 15, 512)       2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 15, 512)       2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 15, 512)       2359808   
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 15, 512)       2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 75267     
=================================================================
Total params: 20,099,651
Trainable params: 75,267
Non-trainable params: 20,024,384
```

This snippet above just defined how many layers I am using and which layer is performing what task.

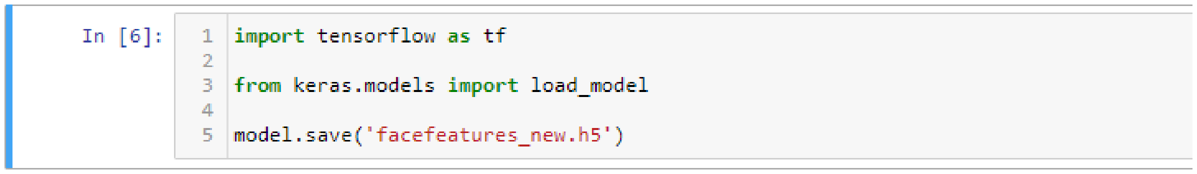
Based on the images of the different person that we have provided in the folder it finds out the number of classes that it has to identify. Here number if classes is number of different people.



```
Found 726 images belonging to 4 classes.
Found 138 images belonging to 4 classes.
```

We then fit of model with given dataset and run it and test its accuracy using test dataset.

Once it has run, we **save the model** so that we **don't have to run it again**, since it is time consuming.



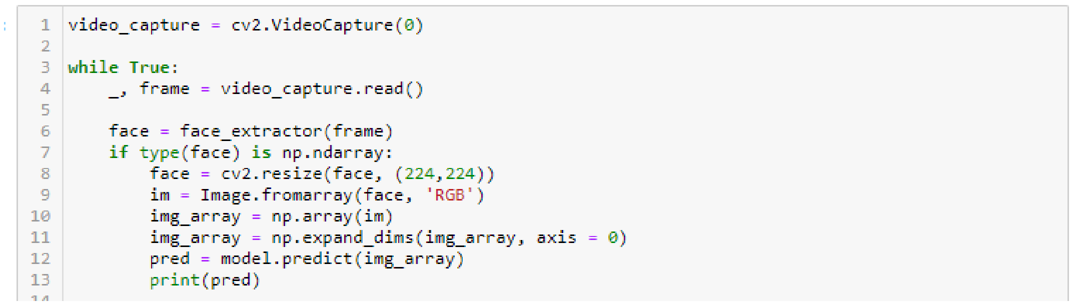
```
In [6]:
import tensorflow as tf

from keras.models import load_model

model.save('facefeatures_new.h5')
```

Changing the models:

Now, if we want use a different model for training our network, we just need to replace VGG16 in the code with whatever our desired model name is.

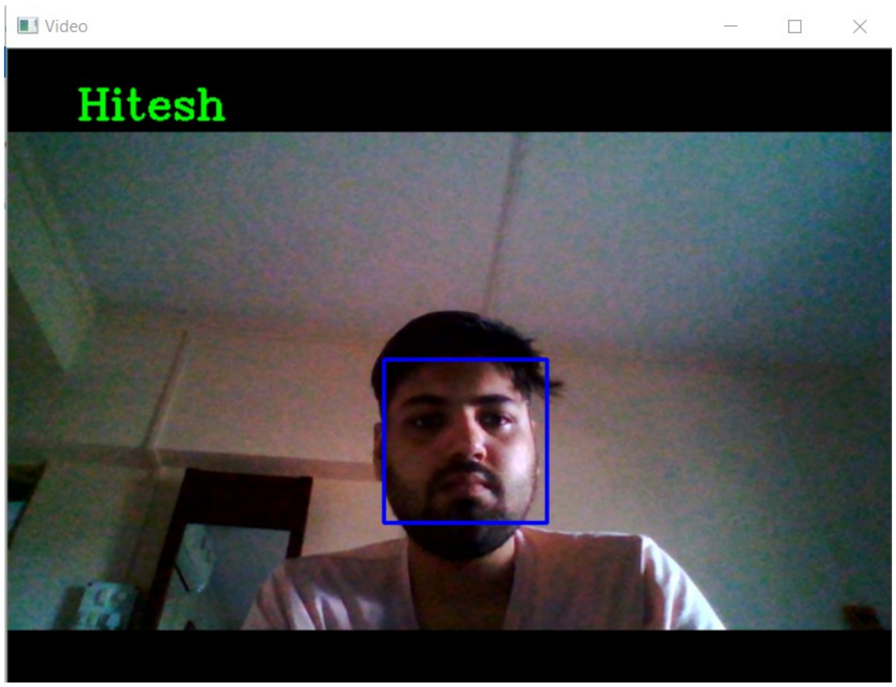
Testing on real time:

for this we use our saved model for prediction, we use the same code for to get frames from camera as we used earlier while collecting the images. But this time we feed these images to our trained model for prediction.

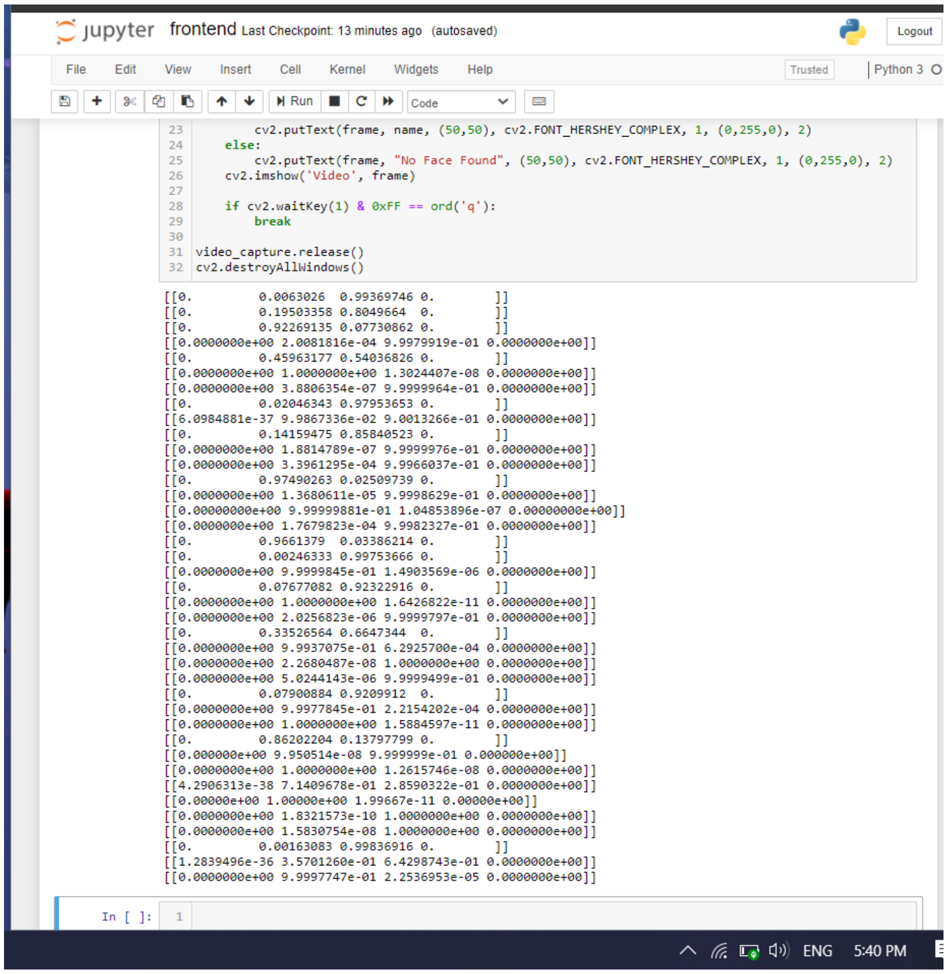


```python
video_capture = cv2.VideoCapture(0)

while True:
    _, frame = video_capture.read()

    face = face_extractor(frame)
    if type(face) is np.ndarray:
        face = cv2.resize(face, (224,224))
        im = Image.fromarray(face, 'RGB')
        img_array = np.array(im)
        img_array = np.expand_dims(img_array, axis = 0)
        pred = model.predict(img_array)
        print(pred)
```

Then we predict the person based on the probability as we have used softmax.



The array shown below is a numpy array that shows up when it is predicting. This is an encoded form of the image which it compares to the trained images for matching.


jupyter frontend Last Checkpoint: 13 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
ENG
5:40 PM

```
            cv2.putText(frame, name, (50,50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
        else:
            cv2.putText(frame, "No Face Found", (50,50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
        cv2.imshow('Video', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video_capture.release()
    cv2.destroyAllWindows()
```

```
[[0.         0.0063026  0.99369746 0.        ]]
[[0.         0.19503358 0.8049664  0.        ]]
[[0.         0.92269135 0.07730862 0.        ]]
[[0.0000000e+00 2.0081816e-04 9.9979919e-01 0.0000000e+00]]
[[0.         0.45963177 0.54036826 0.        ]]
[[0.0000000e+00 1.0000000e+00 1.3024407e-08 0.0000000e+00]]
[[0.0000000e+00 3.8806354e-07 9.9999964e-01 0.0000000e+00]]
[[0.         0.02046343 0.97953653 0.        ]]
[[6.0984881e-37 9.9867336e-02 9.0013266e-01 0.0000000e+00]]
[[0.         0.14159475 0.85840523 0.        ]]
[[0.0000000e+00 1.8814789e-07 9.9999976e-01 0.0000000e+00]]
[[0.0000000e+00 3.3961295e-04 9.9966037e-01 0.0000000e+00]]
[[0.         0.97490263 0.02509739 0.        ]]
[[0.0000000e+00 1.3680611e-05 9.9998629e-01 0.0000000e+00]]
[[0.00000000e+00 9.99999881e-01 1.04853896e-07 0.00000000e+00]]
[[0.0000000e+00 1.7679823e-04 9.9982327e-01 0.0000000e+00]]
[[0.         0.9661379  0.03386214 0.        ]]
[[0.         0.00246333 0.99753666 0.        ]]
[[0.0000000e+00 9.9999845e-01 1.4903569e-06 0.0000000e+00]]
[[0.         0.07677082 0.92322916 0.        ]]
[[0.0000000e+00 1.0000000e+00 1.6426822e-11 0.0000000e+00]]
[[0.0000000e+00 2.0256823e-06 9.9999797e-01 0.0000000e+00]]
[[0.         0.33526564 0.6647344  0.        ]]
[[0.0000000e+00 9.9937075e-01 6.2925700e-04 0.0000000e+00]]
[[0.0000000e+00 2.2680487e-08 1.0000000e+00 0.0000000e+00]]
[[0.0000000e+00 5.0244143e-06 9.9999499e-01 0.0000000e+00]]
[[0.         0.07900884 0.9209912  0.        ]]
[[0.0000000e+00 9.9977845e-01 2.2154202e-04 0.0000000e+00]]
[[0.0000000e+00 1.0000000e+00 1.5884597e-11 0.0000000e+00]]
[[0.         0.86202204 0.13797799 0.        ]]
[[0.000000e+00 9.950514e-08 9.999999e-01 0.000000e+00]]
[[0.0000000e+00 1.0000000e+00 1.2615746e-08 0.0000000e+00]]
[[4.2906313e-38 7.1409678e-01 2.8590322e-01 0.0000000e+00]]
[[0.00000e+00 1.00000e+00 1.99667e-11 0.00000e+00]]
[[0.0000000e+00 1.8321573e-10 1.0000000e+00 0.0000000e+00]]
[[0.0000000e+00 1.5830754e-08 1.0000000e+00 0.0000000e+00]]
[[0.         0.00163083 0.99836916 0.        ]]
[[1.2839496e-36 3.5701260e-01 6.4298743e-01 0.0000000e+00]]
[[0.0000000e+00 9.9997747e-01 2.2536953e-05 0.0000000e+00]]
```

```
In [ ]:
```