



L LOVELY
P ROFESSIONAL
U NIVERSITY

Transforming Education Transforming India

INT 404 – PROJECT OF ARTIFICIAL INTELLIGENT

Group No. - 03

Project Name – Face Recognition System

Submitted By -

Name - Rochan Joshi

Roll no. - 09

Registration no. - 11810809

Name – Saurabh Srivastava

Roll No.- 30

Registration No.- 11804430

Name - Yuvraj Singh

Roll No. - 18

Registration no. - 11804443

Name – Vijaya Shree Kanwal

Roll No. - 36

Registration No. - 11813692

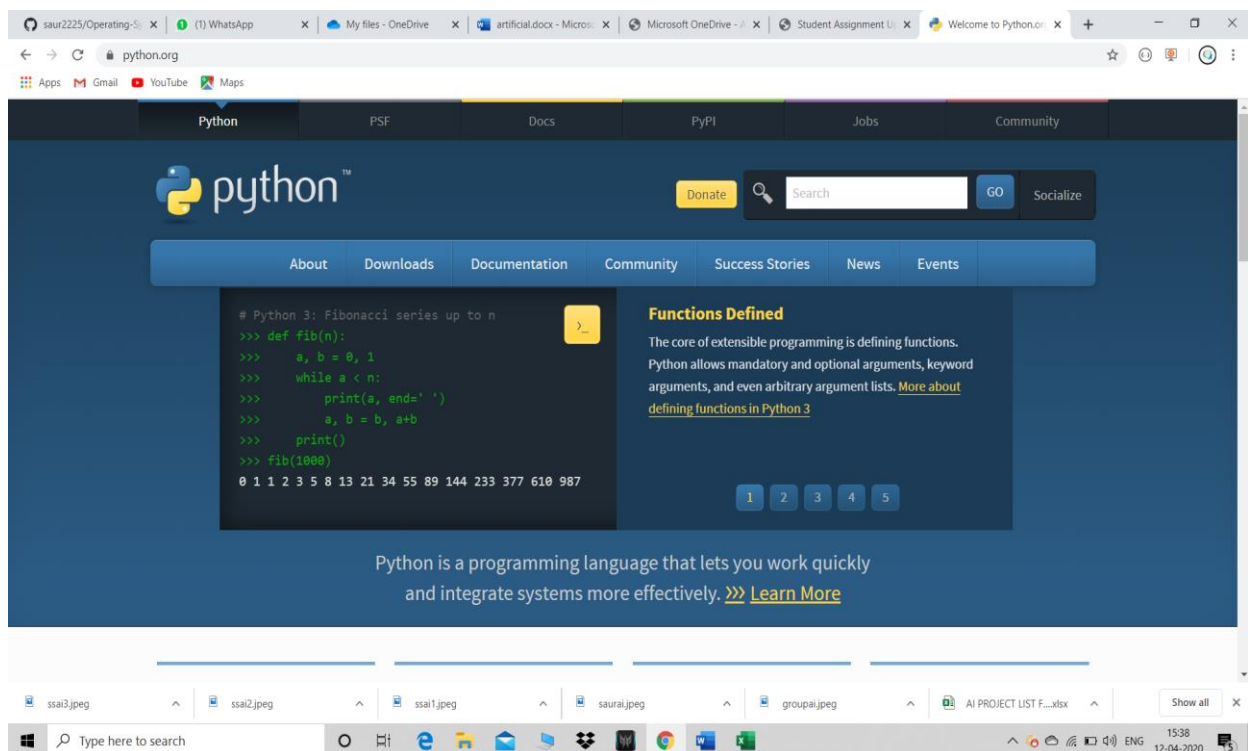
Here for the Artificial intelligence project we have made a face recognition system using OpenCV in python.

For the completion of the project we followed a systematic approach.

- We first installed python in our system using the following steps:

Step 1: Download Python 3.8

- To start, go to python.org/downloads and then click on the button to download the latest version of Python



Step 2: Run the .exe file

- Next, run the .exe file that you just downloaded

Step 3: Install Python 3.8

- You can now start the installation of Python by clicking on Install Now

The installation starts after these steps and is ready for use

From here python is installed but we need a platform to use it; like in anaconda; we get a platform to use all the functionalities. Here we install PyCharm, which is an IDE, for which to write the code in python we require python which was installed previously. PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. If we talk about its functionalities then

- it provides coding assistance and analysis, with code completion, syntax and error highlighting, linter integration, and quick fixes
- Project and code navigation: specialized project views, file structure views and quick jumping between files, classes, methods and usages

For PyCharm installation the following steps were followed:

Step 1)

- To download PyCharm visit the website <https://www.jetbrains.com/pycharm/download/> and Click the "DOWNLOAD" link under the Community Section.

Step 2)

- Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.

Step 3)

- On the next screen, Change the installation path if required. Click “Next”.

Step 4)

- On the next screen, you can create a desktop shortcut if you want and click on “Next”.

Step 5)

- Choose the start menu folder. Keep selected JetBrains and click on “Install”.

Step 6)

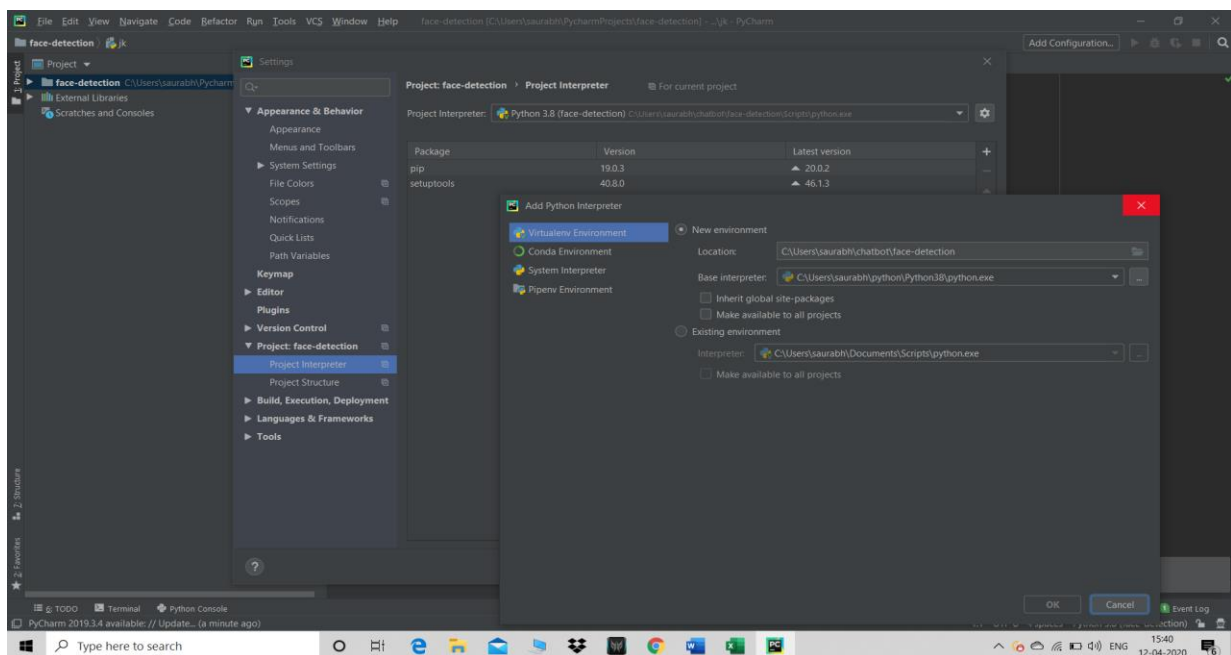
- Wait for the installation to finish.

Step 7)

- Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.

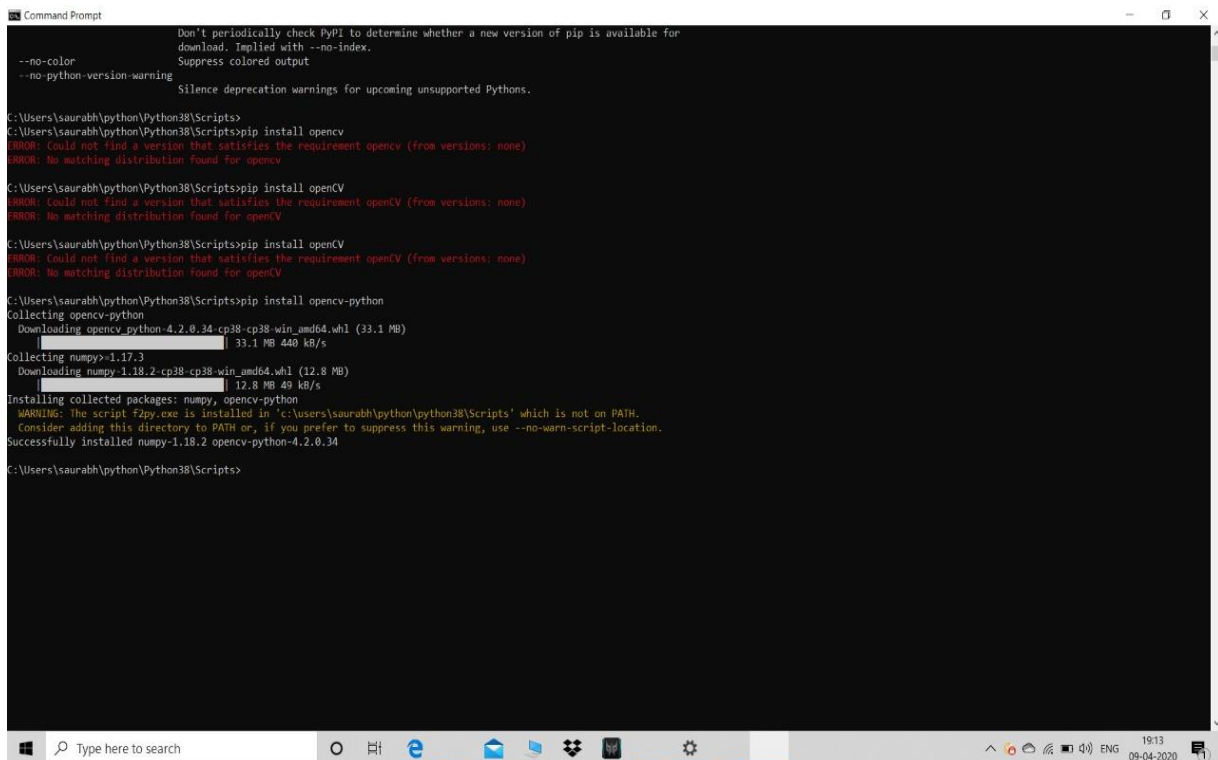
Step 8)

- After you click on "Finish," the Following screen will appear



Now when we install python PIP automatically gets installed. What is pip? PIP is a package manager for Python packages, or modules we can say. PIP is the standard package manager for Python. It allows you to install and manage additional packages that are not part of the Python standard library. We use the **command** PIP install “whatever the library name is that we intend to include” on command prompt or terminal then automatically whatever was the desired module or library gets installed.

Now in PyCharm we install the module OpenCV by the **command** PyCharm install OpenCV-python.



```
Command Prompt

Don't periodically check PyPI to determine whether a new version of pip is available for
download. Implied with --no-index.
--no-color                Suppress colored output
--no-python-version-warning Silence deprecation warnings for upcoming unsupported Pythons.

C:\Users\saurabh\python\Python38\Scripts>pip install opencv
ERROR: Could not find a version that satisfies the requirement opencv (from versions: none)
ERROR: No matching distribution found for opencv

C:\Users\saurabh\python\Python38\Scripts>pip install openCV
ERROR: Could not find a version that satisfies the requirement openCV (from versions: none)
ERROR: No matching distribution found for openCV

C:\Users\saurabh\python\Python38\Scripts>pip install openCV
ERROR: Could not find a version that satisfies the requirement openCV (from versions: none)
ERROR: No matching distribution found for openCV

C:\Users\saurabh\python\Python38\Scripts>pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.2.0.34-cp38-cp38-win_umd64.whl (33.1 MB)
    33.1 MB 440 kB/s
Collecting numpy>=1.17.3
  Downloading numpy-1.18.2-cp38-cp38-win_umd64.whl (12.8 MB)
    12.8 MB 49 kB/s
Installing collected packages: numpy, opencv-python
WARNING: The script f2py.exe is installed in 'c:\users\saurabh\python\python38\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed numpy-1.18.2 opencv-python-4.2.0.34

C:\Users\saurabh\python\Python38\Scripts>
```

Here in case of our project we have upgraded PIP to install OpenCV

OpenCV is the most popular library for computer vision. Originally written in C/C++, it now provides bindings for Python.

OpenCV uses machine learning algorithms to search for faces within a picture. Because faces are so complicated, there isn't one simple test that will tell you if it found a face or not. Instead, there are thousands of small patterns and features that must be matched. The algorithms break the task of identifying the face into thousands of smaller, bite-sized tasks, each of which is easy to solve. These tasks are also called classifiers.

For something like a face, you might have 6,000 or more classifiers, all of which must match for a face to be detected (within error limits, of course). But therein lies the problem: for face detection, the algorithm starts at the top left of a picture and moves down across small blocks of data, looking at each block, constantly asking, "Is this a face? ... Is this a face? ... Is this a face?" Since there are 6,000 or more tests per block, you might have millions of calculations to do, which will grind your computer to a halt.

To get around this, OpenCV uses cascades. What's a cascade? The best answer can be found in the dictionary: "a waterfall or series of waterfalls."

Like a series of waterfalls, the OpenCV cascade breaks the problem of detecting faces into multiple stages. For each block, it does a very rough and quick test. If that passes, it does a slightly more detailed test, and so on. The algorithm may have 30 to 50 of these stages or cascades, and it will only detect a face if all stages pass. The cascades themselves are just a bunch of XML files that contain OpenCV data used to detect objects. You initialize your code with the cascade you want, and then it does the work for you.

Since face detection is such a common case, OpenCV comes with a number of built-in cascades for detecting everything from faces to eyes to hands to legs

In this project we have used the cascade approach. Face detection using Haar cascades is a machine learning based approach where a cascade function is trained with a set of input data. OpenCV already contains many pre-trained classifiers for face, eyes, smiles, etc. Today we will be using the face classifier.

To make it clearer let us dig a little deeper, Cascading is a particular case of ensemble learning based on the concatenation of several classifiers. Cascading classifiers are trained with several hundred "positive" sample views of a particular object that we intend to work on and arbitrary "negative" images of the same size. After the classifier is trained it can be applied to a region of an image and detect the object in question. So here we had to detect face so we had to train the classifier with the images of faces so these, so the displayed/inputted image will be called positive image if it would contain face and negative if it does not. Talking about the classifier's response to it, it gives a value 1 if the image is of the respective interest and a 0 if otherwise.

Here the benefit is that OpenCV comes with a trainer and identifier which is used to apply the concept of positive and negative image or one can access a trained OpenCV directly from GitHub as we have done in our project.

- You need to download the trained classifier XML file (haarcascade_frontalface_default.xml), which is available in OpenCV's GitHub repository. Then to detect a face we need to provide an image
- The detection works only on grayscale images. So, it is important to convert the colour image to grayscale using **cv2.cvtColor()** function
- **detectMultiScale** function is used to detect the faces. It takes 3 arguments —

the input image, scaleFactor and minNeighbours. scaleFactor specifies how much the image size is reduced with each

scale. minNeighbours specifies how many neighbors each candidate rectangle should have to retain it. You can read about it in detail [here](#). You may have to tweak these values to get the best results.

- faces contain a list of coordinates for the rectangular regions where faces were found. We use these coordinates to draw the rectangles in our image.

Similarly, we can detect faces in videos. As you know videos are basically made up of frames, which are still images. So we perform the face detection for each frame in a video

Here is the code that we have made to accomplish the project allotted

```
# OpenCV program to detect face in real time
```

```
# import libraries of python OpenCV
```

```
# where its functionality resides
```

```
import cv2
```

```
# load the required trained XML classifiers
```

```
# https://github.com/Itseez/opencv/blob/master/
```

```
# data/haarcascades/haarcascade_frontalface_default.xml
```

```
# Trained XML classifiers describes some features of some
```

```
# object we want to detect a cascade function is trained
```

```
# from a lot of positive(faces) and negative(non-faces)
```

```
# images.
```



```
face_cascade =  
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')  
  
# https://github.com/Itseez/opencv/blob/master  
# /data/haarcascades/haarcascade_eye.xml  
# Trained XML file for detecting eyes  
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')  
  
# capture frames from a camera  
cap = cv2.VideoCapture(0)  
  
# loop runs if capturing has been initialized.  
while 1:  
  
    # reads frames from a camera  
    ret, img = cap.read()  
  
    # convert to gray scale of each frames  
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  
  
    # Detects faces of different sizes in the input image  
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
```

```

for (x,y,w,h) in faces:

    # To draw a rectangle in a face

    cv2.rectangle(img,(x,y),(x+w,y+h),(255,255,0),2)

    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]


    # Detects eyes of different sizes in the input image

    eyes = eye_cascade.detectMultiScale(roi_gray)


    #To draw a rectangle in eyes

    for (ex,ey,ew,eh) in eyes:

cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,127,255),2)


    # Display an image in a window

    cv2.imshow('img',img)


    # Wait for Esc key to stop

    k = cv2.waitKey(30) & 0xff

    if k == 27:

        break

```

Close the window

```
cap.release()
```

De-allocate any associated memory usage

```
cv2.destroyAllWindows()
```

Output on uploading the image to our system -

