FACE DETECTION AND EMOTION CLASSIFICATION

CONTENT

INTRODUCTION

The project makes use of various APIs to detect face and classify emotion in real-time. It uses OpenCV for face detection and fastai for emotion detection. HaarCascade Classifier was used to detect face in real-time, which was preprocessed and tested on the trained model using fastai.

FastAI:

fastai is a deep learning library which provides practitioners with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains, and provides researchers with low-level components that can be mixed and matched to build new approaches. It aims to do both things without substantial compromises in ease of use, flexibility, or performance.

Link: https://www.fast.ai/

OpenCV:

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license.

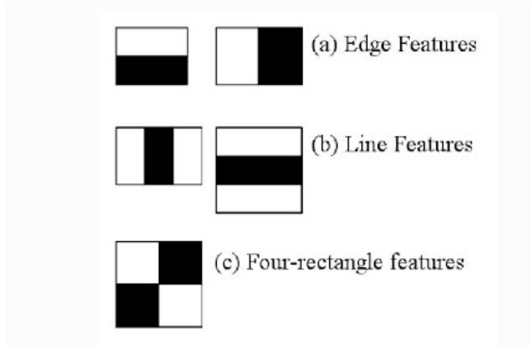
Link: https://opencv.org/

How Haarcascade Classifier works?

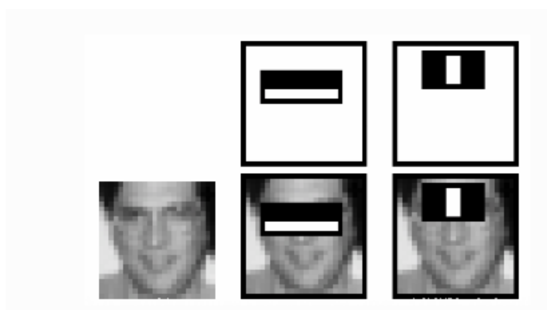
1. 'Haar features' extraction

After the tremendous amount of training data (in the form of images) is fed into the system, the classifier begins by extracting Haar features from each image. Haar

Features are kind of convolution kernels which primarily detect whether a suitable feature is present on an image or not. Some examples of Haar features are mentioned below:



These Haar Features are like windows and are placed upon images to compute a single feature. The feature is essentially a single value obtained by subtracting the sum of the pixels under the white region and that under the black. The process can be easily visualized in the example below.

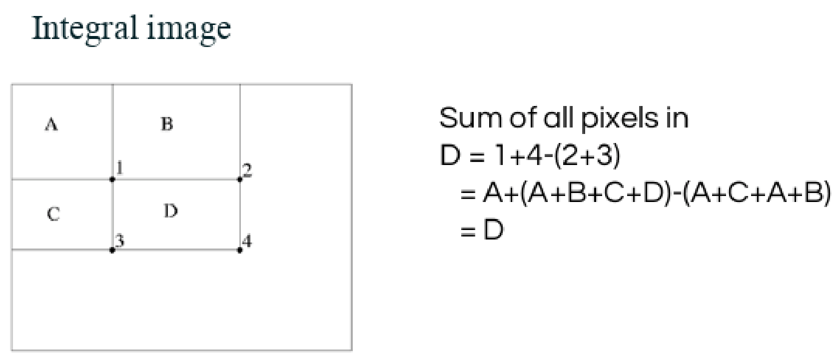


For demonstration purpose, let's say we are only extracting two features, hence we have only two windows here. The first feature relies on the point that the eye region is darker than the adjacent cheeks and nose region. The second feature focuses on the fact that eyes are kind of darker as compared to the bridge of the nose. Thus, when the feature window moves over the eyes, it will calculate a single value. This value will then be compared to some threshold and if it passes that it will conclude that there is an edge here or some positive feature.

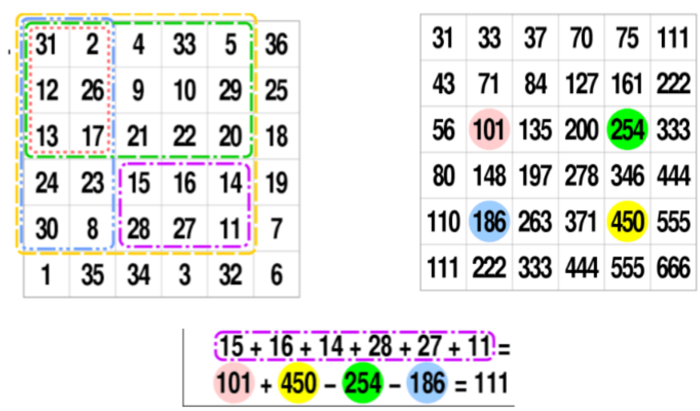
2. 'Integral Images' concept

The algorithm proposed by Viola Jones uses a 24X24 base window size, and that would result in more than 180,000 features being calculated in this window. Imagine calculating the pixel difference for all the features? The solution devised for this computationally intensive process is to go for the **Integral Image** concept.

The integral image means that to find the sum of all pixels under any rectangle, we simply need the four corner values.

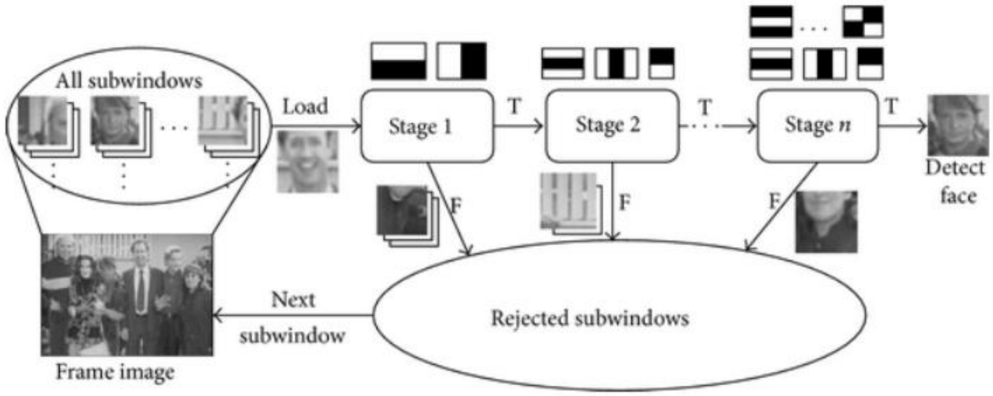


This means, to calculate the sum of pixels in any feature window, we do not need to sum them up individually. All we need is to calculate the integral image using the 4 corner values. The example below will make the process transparent.



3. Using 'Cascade of Classifiers'

Another way by which Viola Jones ensured that the algorithm performs fast is by employing a **cascade of classifiers**. The cascade classifier essentially consists of stages where each stage consists of a strong classifier. This is beneficial since it eliminates the need to apply all features at once on a window. Rather, it groups the features into separate sub-windows and the classifier at each stage determines whether or not the sub-window is a face. In case it is not, the sub-window is discarded along with the features in that window. If the sub-window moves past the classifier, it continues to the next stage where the second stage of features is applied. The process can be understood with the help of the diagram below.



Cascade structure for Haar classifiers.

DOMAIN

- Computer Vision
- Machine Learning
- Deep Learning
- Neural Networks

REQUIREMENTS

Software:

Install the following dependencies before proceeding:

- opencv-python
- PIL
- numpy
- pandas
- fastai
- pytorch
- torchvision
- matplotlib
- seaborn
- CUDA 10
- Game Ready Drivers(NVIDIA)
- Cudnn

Hardware:

- High end GPUs

ALGORITHM

For Face Detection:

OpenCV with FrontalFaceHaarCascadeClassifier:

Import required APIs

```
import cv2
from PIL import Image as PImage

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load the necessary classifier:

Here we are using frontalfaceClassifier for face detection

```
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default
.xml')
```

Load the image to be tested:

```
test_image = cv2.imread('data/face.jpg')
```

Loaded test image

Convert to grayscale:

```
test_image_gray = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)
```

Converted image to grayscale

Function to draw boundary on detection of face:

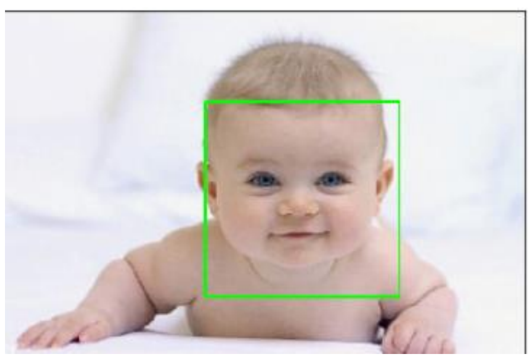
```
def draw_boundary(img, classifier, scaleFactor, minNeighbors, color,
 text):

    ## converts color image to grayimage
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ## detects image using classifier
    features = classifier.detectMultiScale(gray_img, scaleFactor, mi
nNeighbors)
    coords = []
    for (x, y, w, h) in features:
        cv2.rectangle(img, (x, y), (x +w, y+h), color, 2)
```

```
        cv2.putText(img, text, (x, y - 4), cv2.FONT_HERSHEY_SIMPLEX,
 0.8, color, 1, cv2.LINE_AA)
        coords = [x, y, w, h]

    return coords
```

Function to draw boundary on detect on face:

```
def detect(img, faceCascade, eyesCascade, text):
    color = {'blue':(255,0,0), 'red':(0,0,255), 'green':(0,255,0), '
white':(255,255,255)}
    coords = draw_boundary(img, faceCascade, 1.1, 10, color['green']
, text=text)

      return img
```

On calling function:



Result

For emotion detection:

Importing dependencies

==

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)


import os
from fastai.vision import *
from fastai import *
import matplotlib.pyplot as plt
import seaborn as sns
from functools import partial
from tqdm.notebook import tqdm
import gc
from pylab import imread,subplot,imshow,show
%matplotlib inline
```

Setting path to the data set

==

```
path_to_dataset = r'Enter directory name where dataset is present'
path = Path(path_to_dataset)
print(path)
```

Loading data from the folder

==

```
size = 224
bs = 64
data = ImageDataBunch.from_folder(path,
                                  ds_tfms=get_transforms(max_rotate=0.1,max_lighting=0.15),
                                  valid_pct=0.2,
                                  size=size,
                                  bs=bs)
```

Display batches

===

```
data.show_batch(rows=5, figsize=(7, 8))
```

Types of classes found

===

```
data.classes
```

```
['anger', 'disgust', 'fear', 'happiness', 'neutral', 'sadness', 'surprise']
```

Select model

===

```
arch = models.resnet18
```

```
learn = cnn_learner(data, arch, metrics=[accuracy], model_dir = Path("."),path = Path("."))
```

Find the learning rate

==

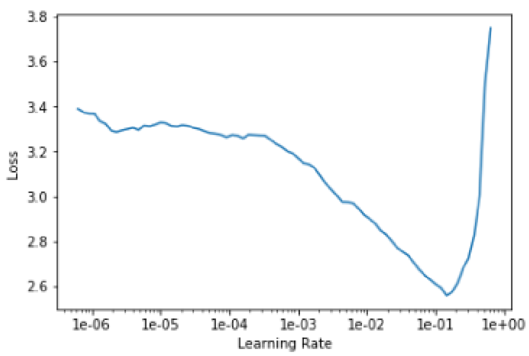
```
learn.lr_find()
learn.recorder.plot(suggestions=True)
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

52.94% [90/170 01:37<01:26 8.1062]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



Set the range for learning rate

==

```
lr1 = 1e-3
lr2 = 1e-1
```

Start training

===

```
learn.fit_one_cycle(14,slice(lr1,lr2))
```

epoch	train_loss	valid_loss	accuracy	time
0	1.011371	0.899082	0.713241	02:59
1	0.743597	0.597746	0.806876	01:52
2	0.865933	0.888465	0.747257	01:52
3	0.920816	0.834745	0.812363	01:54
4	0.881279	0.854846	0.787125	01:53
5	0.880715	1.122557	0.782004	01:53
6	0.948977	0.729618	0.817849	01:53
7	0.621264	0.473688	0.844916	01:53
8	0.578504	0.575723	0.826993	01:53
9	0.530720	0.623550	0.817849	01:53
10	0.476236	0.424067	0.859546	01:53
11	0.440829	0.409345	0.868691	01:53
12	0.407143	0.426177	0.871982	01:53
13	0.382377	0.375116	0.872714	01:53

Export the model

===

```
learn.export()
```

For emotion detection in real-time:

emotion_detection.py

```
import os
from fastai.vision import *
from fastai import *
import matplotlib.pyplot as plt
import seaborn as sns
from functools import partial
from tqdm.notebook import tqdm
import gc
from pylab import imread,subplot,imshow,show
```

```
model_path = r'Enter path to your model'

def load_model(model_path):
    learn = load_learner(model_path)

    return learn

def predict_emotion(img):
    learn = load_model(model_path)
    pred_class, pred_idx, outputs = learn.predict(img)

    return pred_class, pred_idx, outputs
```

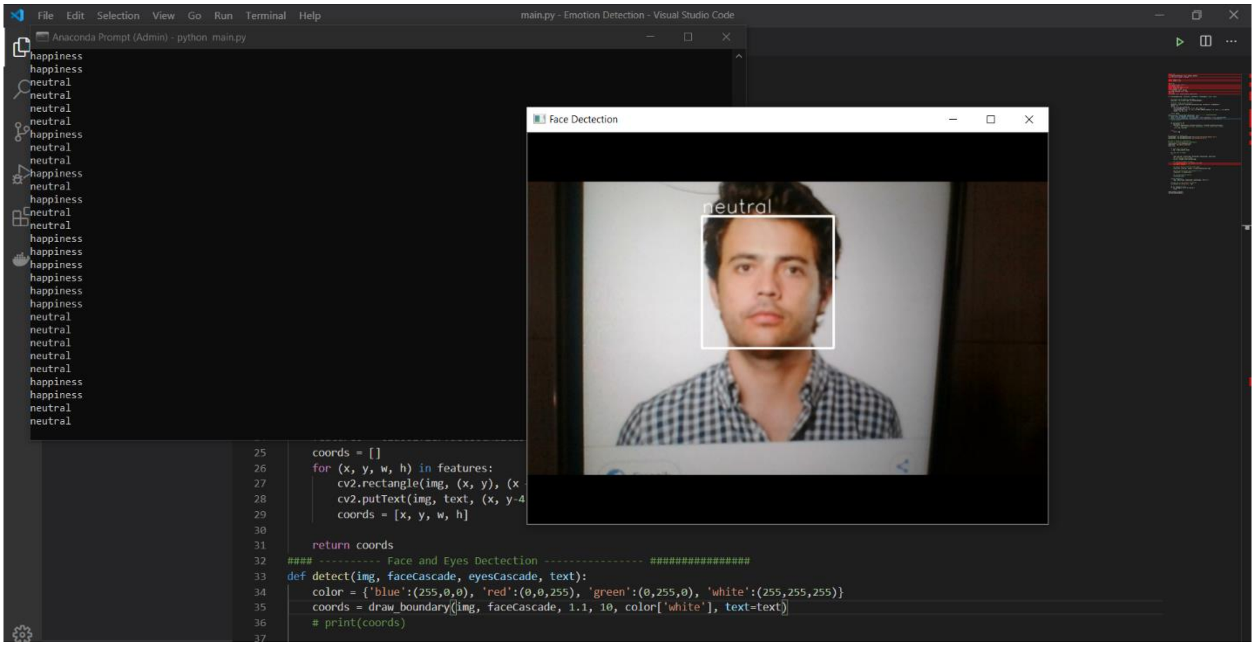
main.py

```
import cv2
from emotion_detection import predict_emotion
from PIL import Image as PImage

import numpy as np
import pandas as pd

import os
from fastai.vision import *
from fastai import *
import matplotlib.pyplot as plt
import seaborn as sns
from functools import partial
from tqdm.notebook import tqdm
import gc
from pylab import imread,subplot,imshow,show

def draw_boundary(img, classifier, scaleFactor, minNeighbors, color, text)
:

    ## converts color image to grayimage
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    ## detects image using classifier
```

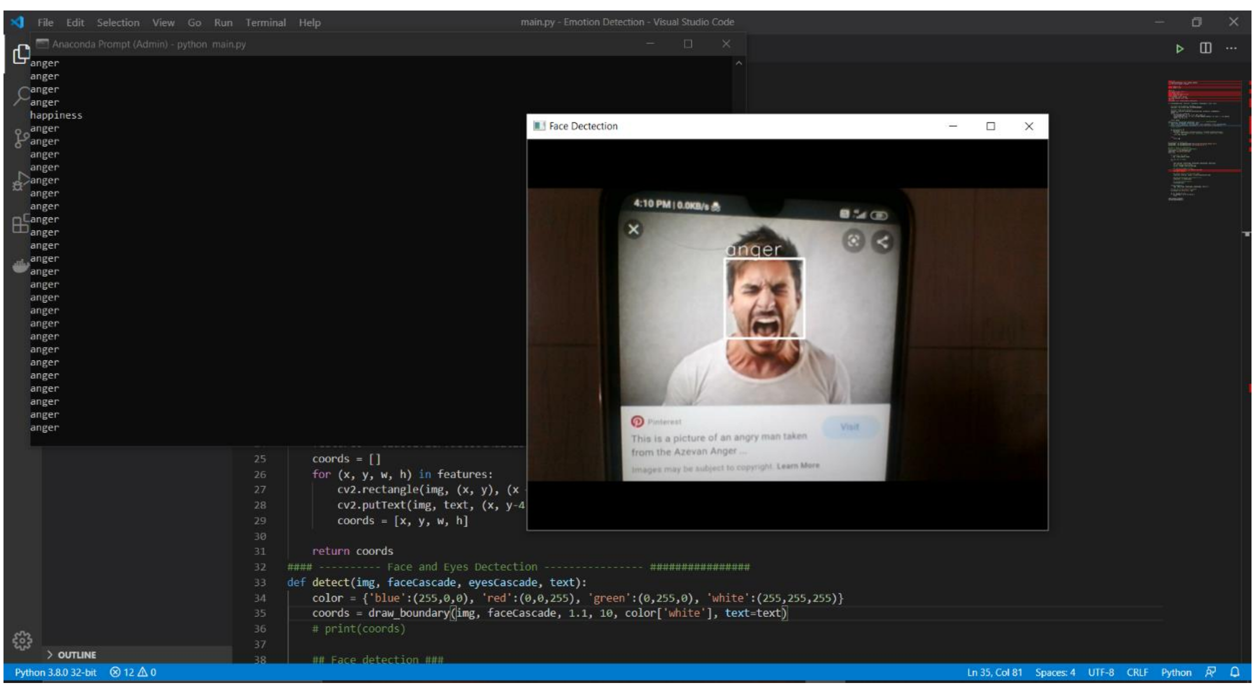
```
    features = classifier.detectMultiScale(gray_img, scaleFactor, minNeigh
bors)
    coords = []
    for (x, y, w, h) in features:
        cv2.rectangle(img, (x, y), (x +w, y+h), color, 2)
        cv2.putText(img, text, (x, y-
4), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 1, cv2.LINE_AA)
        coords = [x, y, w, h]

    return coords
#### ---------- Face and Eyes Dectection ---------------- ################
def detect(img, faceCascade, eyesCascade, text):
    color = {'blue':(255,0,0), 'red':(0,0,255), 'green':(0,255,0), 'white'
:(255,255,255)}
    coords = draw_boundary(img, faceCascade, 1.1, 10, color['white'], text
=text)
    # print(coords)

    ## Face detection ###
    if len(coords) == 4:
        face_img = img[coords[1]:coords[1]+coords[3], coords[0]:coords[0]+
coords[2]]
    #     coords = draw_boundary(roi_img, eyesCascade, 1.1, 14, color['red
'], "Eyes")
        return img, face_img

    else:
        return img


## classifiers to detect face
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eyesCascade = cv2.CascadeClassifier('haarcascade_eye.xml')

## Uses 0: webcam for videocapture
## Uses -1: external drives for videocapture
video_capture = cv2.VideoCapture(0)
pred_class = "No emotion detected"
while True:

    ## reads data from webcam
    _, img = video_capture.read()

    ## crops face as images
    try:
```

```
        img, face_img = detect(img, faceCascade, eyesCascade, pred_class)
        ## converts np.array to image
        pil_im = PImage.fromarray(face_img)

        ## converts pilImage to tensor
        x = pil2tensor(pil_im ,np.float32).div_(255)
        fast_img = Image(x)

        ## predict emotion from the tensor image
        pred_class, pred_idx, outputs = predict_emotion(fast_img)

        ## type casting fastai.core.Category to str
        pred_class = str(pred_class)

        ## printing detected emotion
        print(pred_class)

    except ValueError:
        img = detect(img, faceCascade, eyesCascade, "Neutral")

    ## opens tab to show output from webcam
    cv2.imshow("Face Dectection", img)

    ## for closing the cam
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

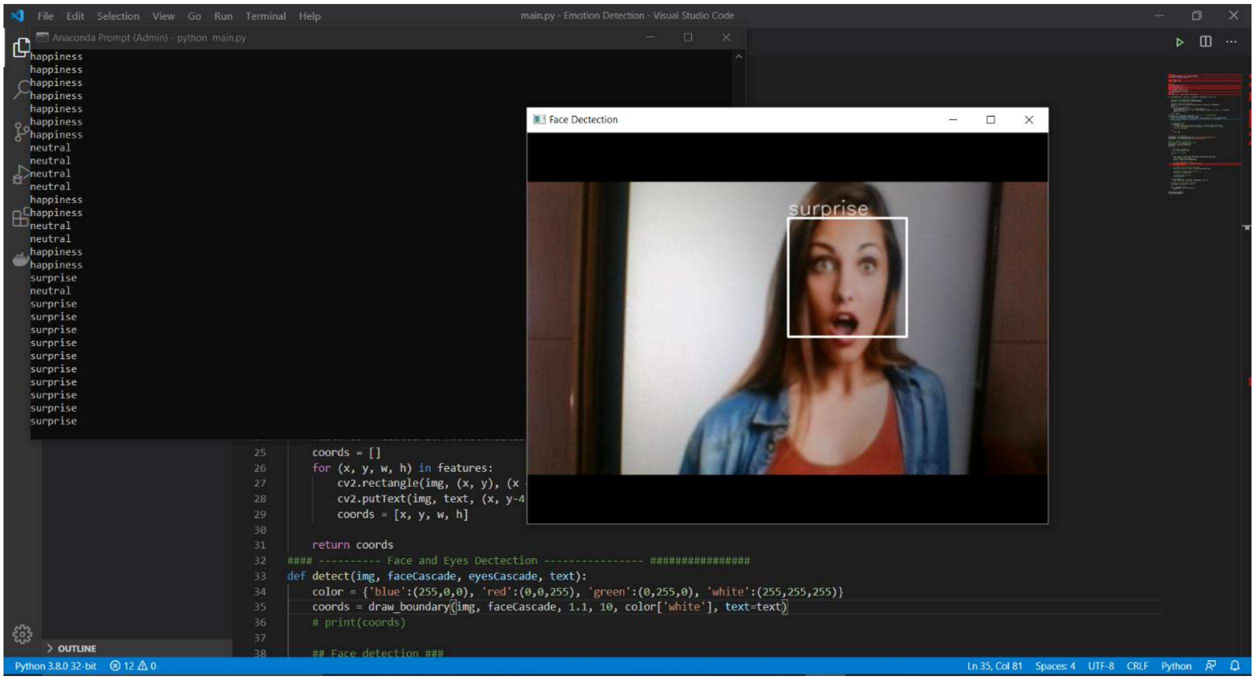
RESULTS



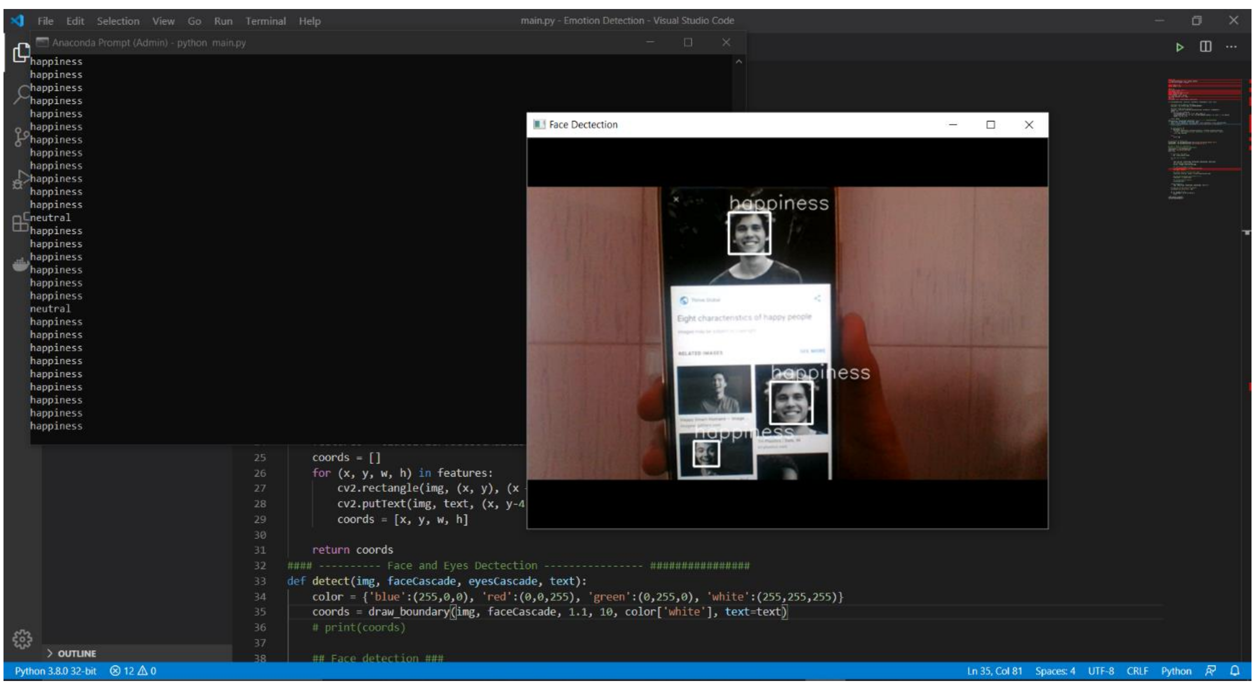
Neutral emotion



Anger emotion



Surprise emotion



Happy emotion

TEST CASE

Load the model

==

```
path_to_trained_model = r'ENter directory where trained model was saved'
learn = load_learner(path_to_trained_model)
```

Testing the model

==

```
img = open_image('happy.jpg')
type(img)
```

```
fastai.vision.image.Image
```

```
img
```

```
pred_class, pred_idx, outputs = learn.predict(img)
```

```
pred_class, pred_idx, outputs
```

```
(Category happiness,
 tensor(3),
 tensor([6.2531e-04, 1.2135e-02, 1.0862e-03, 9.6239e-01, 3.3673e-03, 1.9985e-02,
         4.1485e-04]))
```

```
pred_class
```

```
Category happiness
```