Project Report on

Face Detection using Mtcnn

Submitted to: Md. Imran Hussain (26819)

Submitted By :

Nikhil Chaurasiya (11813436)

Department of computer science and engineering
Lovely Professional University
Phagwara Punjab

Github link:
https://github.com/nik9327/face_detection_using_mtcnn

About the project

Introduction

Face detection -- also called facial detection -- is an artificial intelligence (AI) based computer technology used to find and identify human faces in digital images. Face detection technology can be applied to various fields -- including security, biometrics, law enforcement, entertainment and personal safety -- to provide surveillance and tracking of people in real time.

Face detection has progressed from rudimentary computer vision techniques to advances in machine learning (ML) to increasingly sophisticated artificial neural networks (ANN) and related technologies; the result has been continuous performance improvements. It now plays an important role as the first step in many key applications -- including face tracking, face analysis and facial recognition. Face detection has a significant effect on how sequential operations will perform in the application.

How face detection works

Face detection applications use algorithms and ML to find human faces within larger images, which often incorporate other non-face objects such as landscapes, buildings and other human body parts like feet or hands. Face detection algorithms typically start by searching for human eyes -- one of the easiest features to detect. The algorithm might then attempt to detect eyebrows, the mouth, nose, nostrils and the iris. Once the algorithm concludes that it has found a facial region, it applies additional tests to confirm that it has, in fact, detected a face.

To help ensure accuracy, the algorithms need to be trained on large data sets incorporating hundreds of thousands of positive and negative images. The training improves the algorithms' ability to determine whether there are faces in an image and where they are.

The methods used in face detection can be knowledge-based, feature-based, template matching or appearance-based. Each has advantages and disadvantages:

- Knowledge-based, or rule-based methods, describe a face based on rules. The challenge of this approach is the difficulty of coming up with well-defined rules.

- Feature invariant methods -- which use features such as a person's eyes or nose to detect a face -- can be negatively affected by noise and light.
- Template-matching methods are based on comparing images with standard face patterns or features that have been stored previously and correlating the two to detect a face. Unfortunately these methods do not address variations in pose, scale and shape.
- Appearance-based methods employ statistical analysis and machine learning to find the relevant characteristics of face images. This method, also used in feature extraction for face recognition, is divided into sub-methods.

Some of the more specific techniques used in face detection include:

- Removing the background. For example, if an image has a plain, mono-color background or a pre-defined, static background, then removing the background can help reveal the face boundaries.
- In color images, sometimes skin color can be used to find faces; however, this may not work with all complexions.
- Using motion to find faces is another option. In real-time video, a face is almost always moving, so users of this method must calculate the moving area. One drawback of this method is the risk of confusion with other objects moving in the background.
- A combination of the strategies listed above can provide a comprehensive face detection method.

What is MTCNN

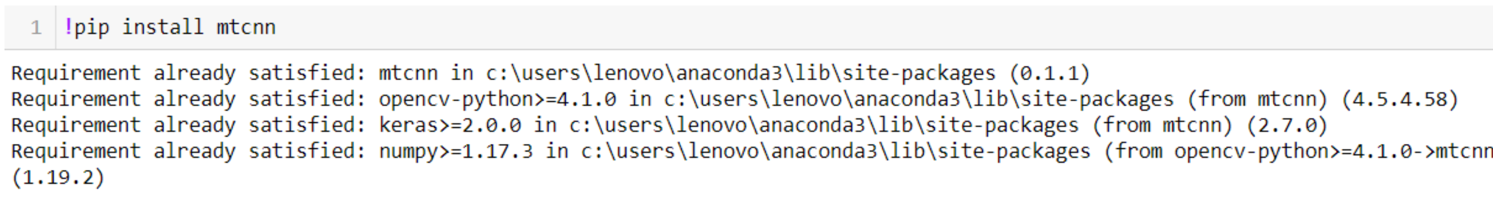
MTCNN is a python (pip) library written by Github user ipacz

In this paper, they propose a deep cascaded multi-task framework using different features of “sub-models” to each boost their correlating strengths.

MTCNN performs quite fast on a CPU, even though S3FD is still quicker running on a GPU

Installing MTCNN Library

MTCNN is available as a pip package, meaning we can easily install it using

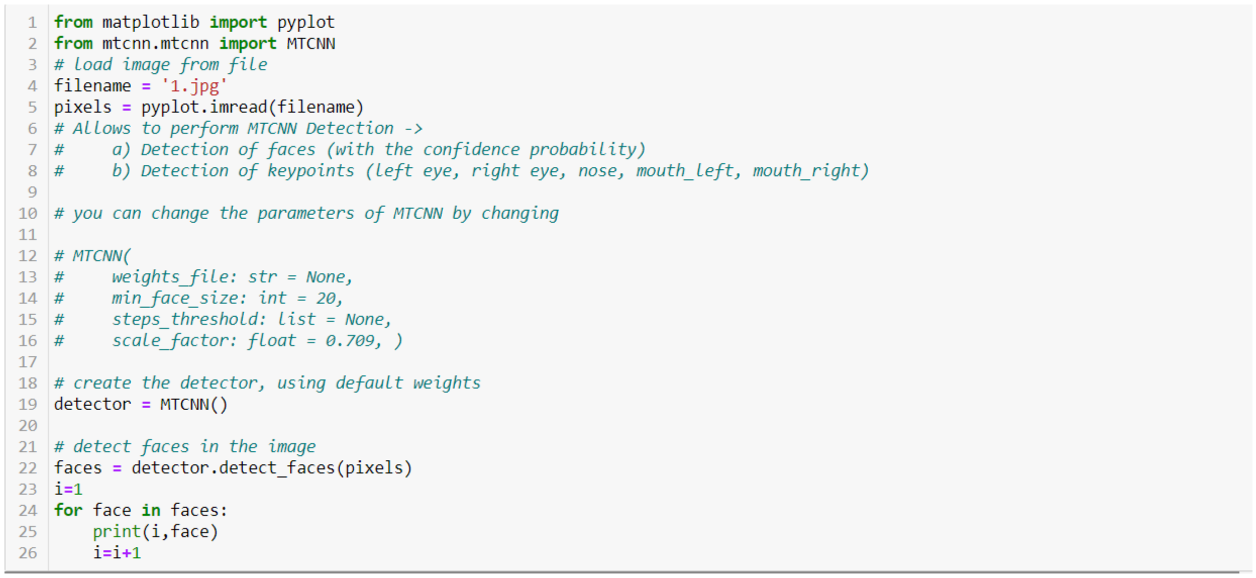


```
!pip install mtcnn
```

```
Requirement already satisfied: mtcnn in c:\users\lenovo\anaconda3\lib\site-packages (0.1.1)
Requirement already satisfied: opencv-python>=4.1.0 in c:\users\lenovo\anaconda3\lib\site-packages (from mtcnn) (4.5.4.58)
Requirement already satisfied: keras>=2.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from mtcnn) (2.7.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\lenovo\anaconda3\lib\site-packages (from opencv-python>=4.1.0->mtcnn
(1.19.2)
```

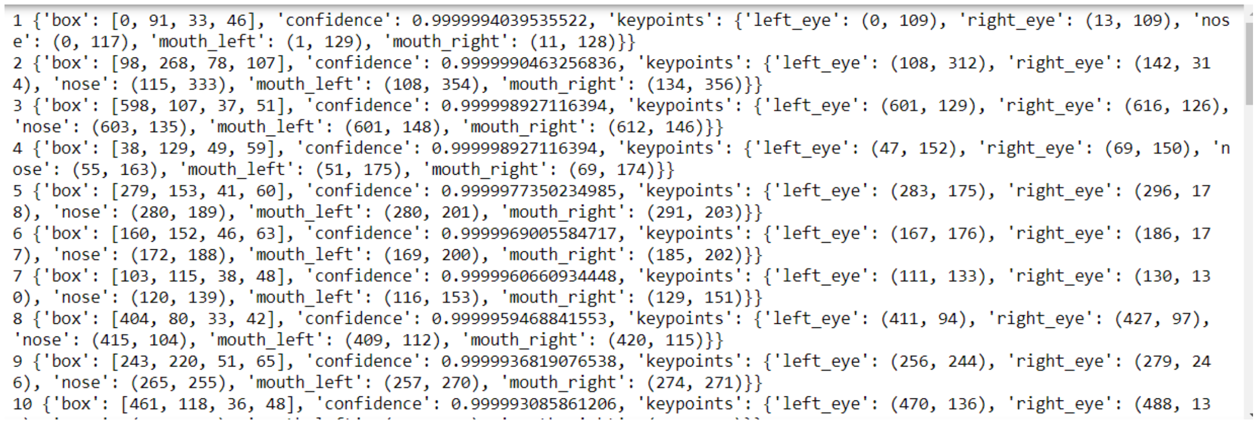
Now switching to Python/Jupyter Notebook we can check the installation with an import and quick verification:

Afterwards, we are ready to load out test image using the matplotlib imread function.



```python
from matplotlib import pyplot
from mtcnn.mtcnn import MTCNN
# load image from file
filename = '1.jpg'
pixels = pyplot.imread(filename)
# Allows to perform MTCNN Detection ->
#     a) Detection of faces (with the confidence probability)
#     b) Detection of keypoints (left eye, right eye, nose, mouth_left, mouth_right)

# you can change the parameters of MTCNN by changing

# MTCNN(
#     weights_file: str = None,
#     min_face_size: int = 20,
#     steps_threshold: list = None,
#     scale_factor: float = 0.709, )

# create the detector, using default weights
detector = MTCNN()

# detect faces in the image
faces = detector.detect_faces(pixels)
i=1
for face in faces:
    print(i,face)
    i=i+1
```

Now your output will look a lot like this:



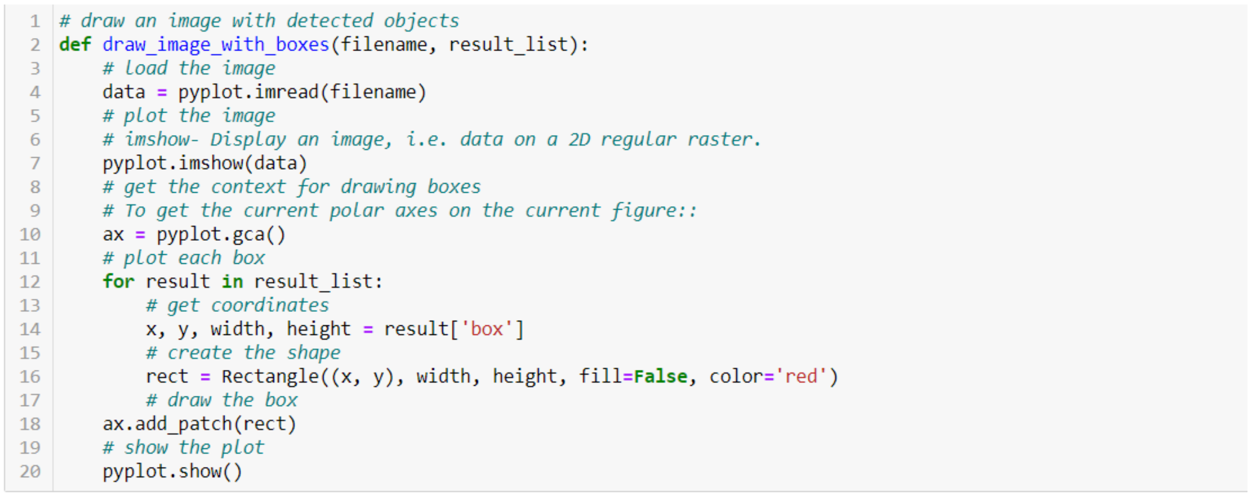
```
1 {'box': [0, 91, 33, 46], 'confidence': 0.9999994039535522, 'keypoints': {'left_eye': (0, 109), 'right_eye': (13, 109), 'nos
e': (0, 117), 'mouth_left': (1, 129), 'mouth_right': (11, 128)}}
2 {'box': [98, 268, 78, 107], 'confidence': 0.9999990463256836, 'keypoints': {'left_eye': (108, 312), 'right_eye': (142, 31
4), 'nose': (115, 333), 'mouth_left': (108, 354), 'mouth_right': (134, 356)}}
3 {'box': [598, 107, 37, 51], 'confidence': 0.999998927116394, 'keypoints': {'left_eye': (601, 129), 'right_eye': (616, 126),
'nose': (603, 135), 'mouth_left': (601, 148), 'mouth_right': (612, 146)}}
4 {'box': [38, 129, 49, 59], 'confidence': 0.999998927116394, 'keypoints': {'left_eye': (47, 152), 'right_eye': (69, 150), 'n
ose': (55, 163), 'mouth_left': (51, 175), 'mouth_right': (69, 174)}}
5 {'box': [279, 153, 41, 60], 'confidence': 0.9999977350234985, 'keypoints': {'left_eye': (283, 175), 'right_eye': (296, 17
8), 'nose': (280, 189), 'mouth_left': (280, 201), 'mouth_right': (291, 203)}}
6 {'box': [160, 152, 46, 63], 'confidence': 0.9999969005584717, 'keypoints': {'left_eye': (167, 176), 'right_eye': (186, 17
7), 'nose': (172, 188), 'mouth_left': (169, 200), 'mouth_right': (185, 202)}}
7 {'box': [103, 115, 38, 48], 'confidence': 0.9999960660934448, 'keypoints': {'left_eye': (111, 133), 'right_eye': (130, 13
0), 'nose': (120, 139), 'mouth_left': (116, 153), 'mouth_right': (129, 151)}}
8 {'box': [404, 80, 33, 42], 'confidence': 0.9999959468841553, 'keypoints': {'left_eye': (411, 94), 'right_eye': (427, 97),
'nose': (415, 104), 'mouth_left': (409, 112), 'mouth_right': (420, 115)}}
9 {'box': [243, 220, 51, 65], 'confidence': 0.9999936819076538, 'keypoints': {'left_eye': (256, 244), 'right_eye': (279, 24
6), 'nose': (265, 255), 'mouth_left': (257, 270), 'mouth_right': (274, 271)}}
10 {'box': [461, 118, 36, 48], 'confidence': 0.999993085861206, 'keypoints': {'left_eye': (470, 136), 'right_eye': (488, 13
```

What does this tell us? A lot of it is self-explanatory, but it basically returns coordinates, or the pixel values of a rectangle where the MTCNN algorithm detected faces. The “box” value above returns the location of the whole face, followed by a “confidence” level.

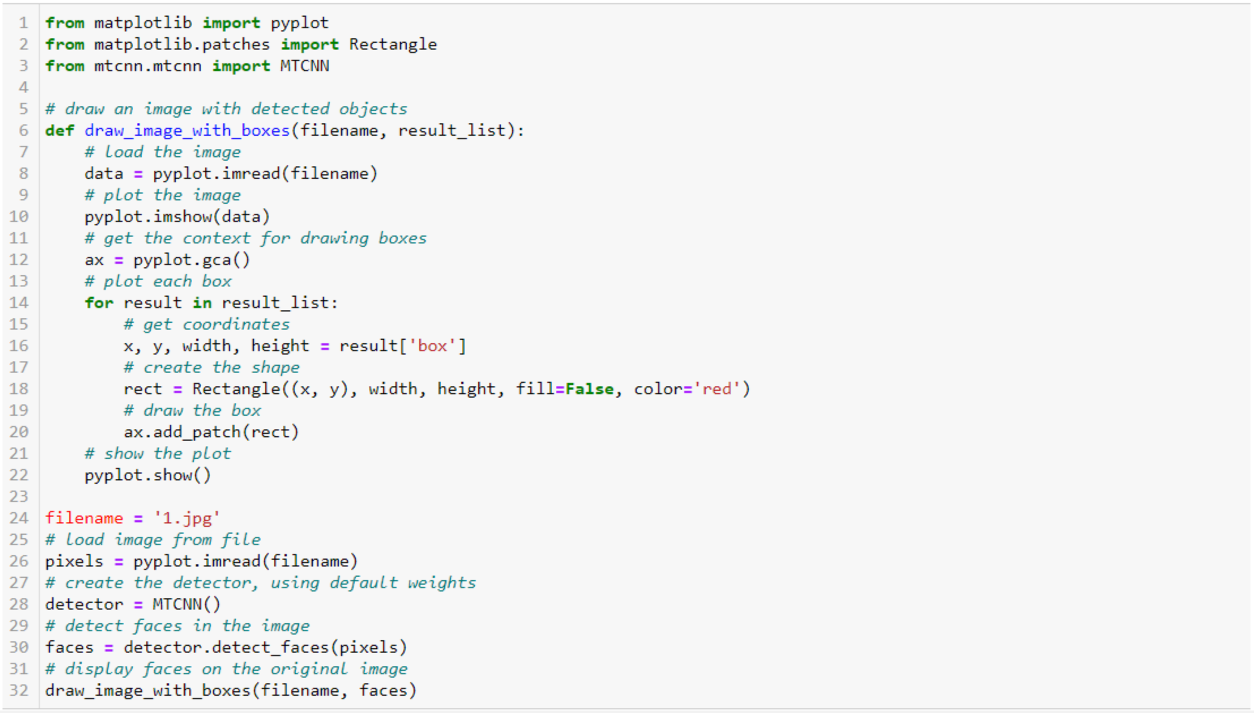
If you want to do more advanced extractions or algorithms, you will have access to other facial landmarks, called “keypoints” as well. Namely the MTCNN model located the eyes, mouth and nose as well!

Drawing a box around faces

To demonstrate this even better let us draw a box around the face using matplotlib:

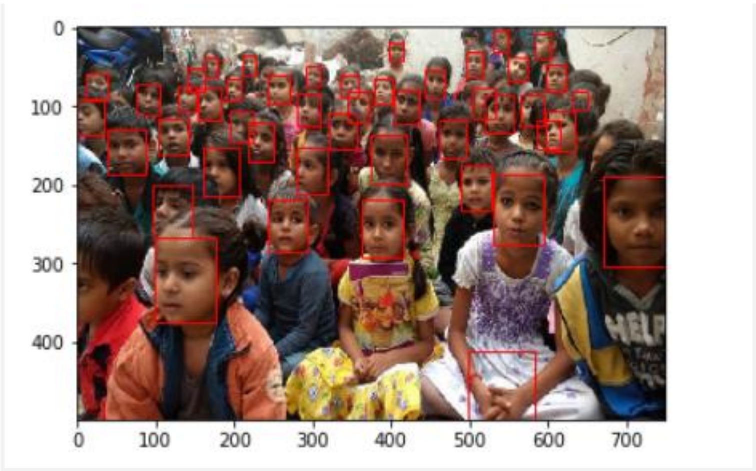


```python
# draw an image with detected objects
def draw_image_with_boxes(filename, result_list):
    # load the image
    data = pyplot.imread(filename)
    # plot the image
    # imshow- Display an image, i.e. data on a 2D regular raster.
    pyplot.imshow(data)
    # get the context for drawing boxes
    # To get the current polar axes on the current figure::
    ax = pyplot.gca()
    # plot each box
    for result in result_list:
        # get coordinates
        x, y, width, height = result['box']
        # create the shape
        rect = Rectangle((x, y), width, height, fill=False, color='red')
        # draw the box
    ax.add_patch(rect)
    # show the plot
    pyplot.show()
```



```python
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mtcnn.mtcnn import MTCNN

# draw an image with detected objects
def draw_image_with_boxes(filename, result_list):
    # load the image
    data = pyplot.imread(filename)
    # plot the image
    pyplot.imshow(data)
    # get the context for drawing boxes
    ax = pyplot.gca()
    # plot each box
    for result in result_list:
        # get coordinates
        x, y, width, height = result['box']
        # create the shape
        rect = Rectangle((x, y), width, height, fill=False, color='red')
        # draw the box
        ax.add_patch(rect)
    # show the plot
    pyplot.show()

filename = '1.jpg'
# load image from file
pixels = pyplot.imread(filename)
# create the detector, using default weights
detector = MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_image_with_boxes(filename, faces)
```

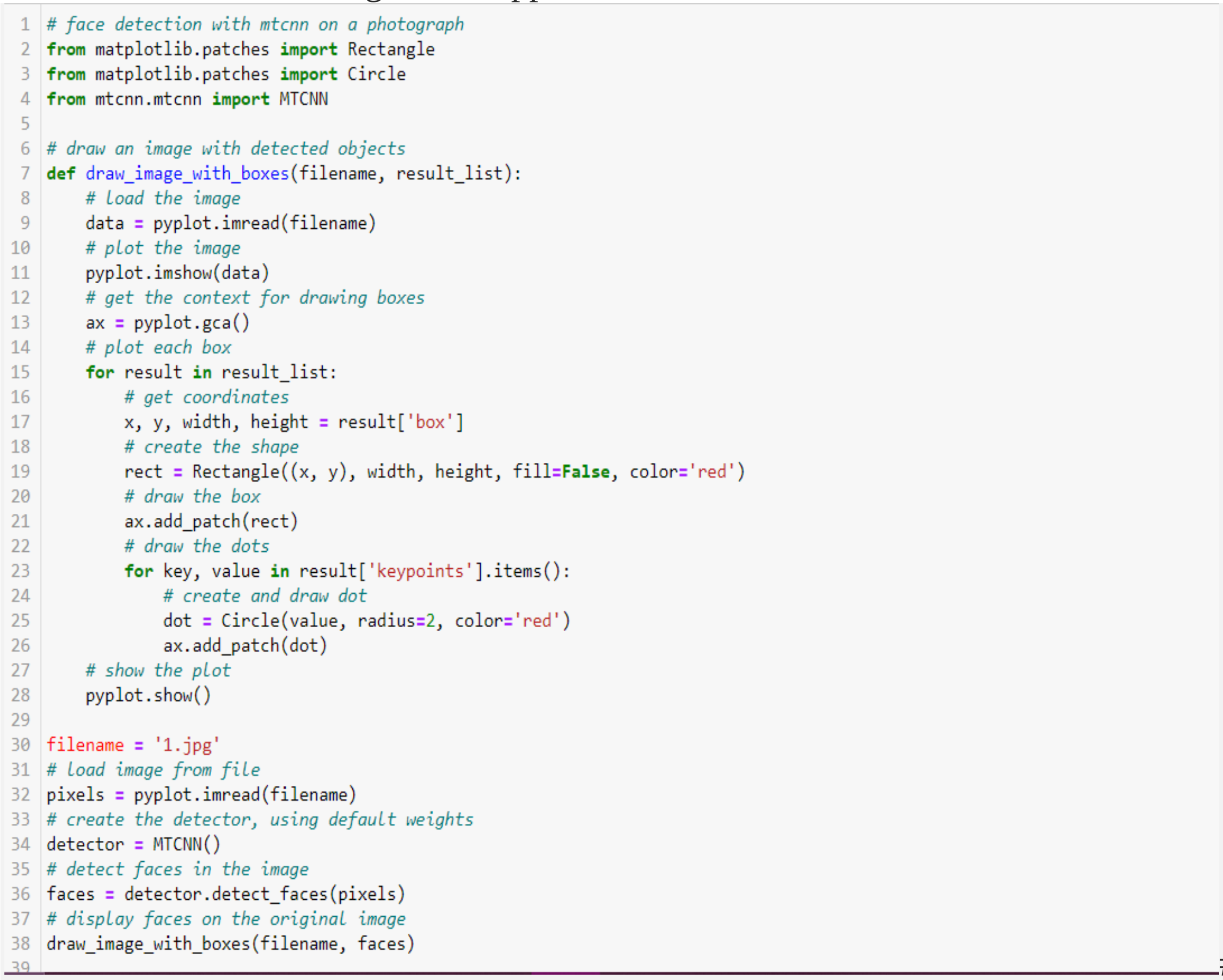
Output



Adding more features in the model to display eyes, mouth and nose around faces

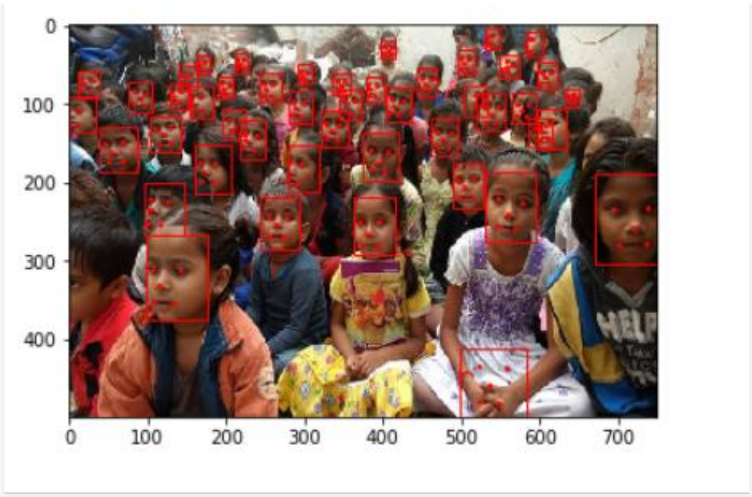
Now let us take a look at the aforementioned “keypoints” that the MTCNN model returned.

We will now use these to graph the nose, mouth and eyes as well.
We will the add following code snippet to our code above:



```python
# face detection with mtcnn on a photograph
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from mtcnn.mtcnn import MTCNN

# draw an image with detected objects
def draw_image_with_boxes(filename, result_list):
    # load the image
    data = pyplot.imread(filename)
    # plot the image
    pyplot.imshow(data)
    # get the context for drawing boxes
    ax = pyplot.gca()
    # plot each box
    for result in result_list:
        # get coordinates
        x, y, width, height = result['box']
        # create the shape
        rect = Rectangle((x, y), width, height, fill=False, color='red')
        # draw the box
        ax.add_patch(rect)
        # draw the dots
        for key, value in result['keypoints'].items():
            # create and draw dot
            dot = Circle(value, radius=2, color='red')
            ax.add_patch(dot)
    # show the plot
    pyplot.show()

filename = '1.jpg'
# load image from file
pixels = pyplot.imread(filename)
# create the detector, using default weights
detector = MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_image_with_boxes(filename, faces)
```

#



Extra Experimental analysis (in case of large dataset)

Advanced MTCNN: Speed it up (\~x100)!

Now let us come to the interesting part. If you are going to process millions of pictures you will need to speed up MTCNN, otherwise, you will either fall asleep or your CPU will burn before it will be done.

But what exactly are we talking about? If you are running the above code it will take around one second, meaning we will process around one picture per second. If you are running MTCNN on a GPU and use the sped-up version it will achieve around 60–100 pictures/frames a second. That is a boost of up to **100 times**!

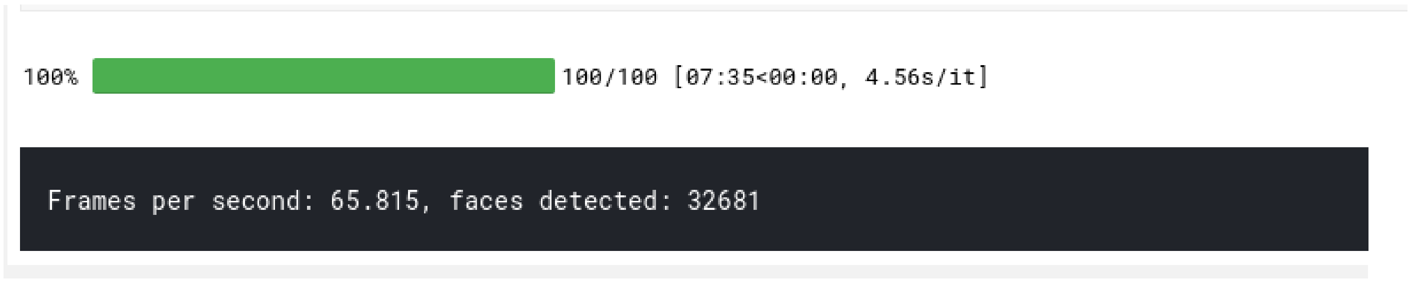
If you are for example going to extract all faces of a movie, where you will extract 10 faces per second (one second of the movie has on average around 24 frames, so every second frame) it will be 10 * 60 (seconds) * 120 (minutes) = 72,000 frames.

Meaning if it takes one second to process one frame it will take 72,000 * 1 (seconds) = 72,000s / 60s = 1,200m = **20 hours**

With the sped-up version of MTCNN this task will take 72,000 (frames) / 100 (frames/sec) = 720 seconds = **12 minutes**!

To use MTCNN on a GPU you will need to set up CUDA, cudnn, pytorch and so on

Once installed we will do the necessary imports as follows:



```
100% 100/100 [07:35<00:00, 4.56s/it]

Frames per second: 65.815, faces detected: 32681
```

The above image shows the output of the code running on an NVIDIA Tesla P100, so depending on the source material, GPU and processor you might experience better or worse performance.

Extracting all images seperately

```python
# extract and plot each detected face in a photograph
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from matplotlib.patches import Circle
from mtcnn.mtcnn import MTCNN

# draw each face separately
def draw_faces(filename, result_list):
    # load the image
    data = pyplot.imread(filename)
    # plot each face as a subplot
    for i in range(len(result_list)):
        # get coordinates
        x1, y1, width, height = result_list[i]['box']
        x2, y2 = x1 + width, y1 + height
        # define subplot
        pyplot.subplot(1, len(result_list), i+1)
        pyplot.axis('off')
        # plot face
        pyplot.imshow(data[y1:y2, x1:x2])
    # show the plot
    pyplot.show()

filename = '1.jpg'
# load image from file
pixels = pyplot.imread(filename)
# create the detector, using default weights
detector = MTCNN()
# detect faces in the image
faces = detector.detect_faces(pixels)
# display faces on the original image
draw_faces(filename, faces)
```

Conclusion

Face recognition is an emerging technology that can provide many benefits. Face recognition can save resources and time, and even generate new income streams, for companies that implement it right.

What lies ahead for this technology?

It’s difficult to be certain. Some experts predict that our faces will replace IDs, passports and credit card pin numbers. Given the fact how convenient and cost-effective this technology is, this prediction is not far-fetched.

If this prediction becomes a reality, any company that implemented the technology today might gain a competitive advantage in the future.

What the Future Holds or future scope?

The future of facial recognition technology is bright. Forecasters opine that this technology is expected to grow at a formidable rate and will generate huge revenues in the coming years. Security and surveillances are the major segments which will be deeply influenced. Other areas that are now welcoming it with open arms are private industries, public buildings, and schools. It is estimated that it will also be adopted by retailers and banking systems in coming years to keep fraud in debit/credit card purchases and payment especially the ones that are online

Reference

-Towards machine learning blogs

- Some Github repository