Performance Evaluation of Face Recognition Algorithms

COMP 482 - Computer Vision Final Project

Date: 31.05.2022
Student: Nadide Beyza Dokur
Instructor: Muhittin Gökmen
Institution: MEF University

1. Introduction

A method of detecting or validating a person's identification by glancing at their face is known as facial recognition. It uses face recognition technology to identify people in photographs, videos, and real-time. Facial recognition is included in biometric security. Biometric software also includes voice recognition, fingerprint recognition, and so on. Security and law enforcement are the primary applications for the technology. Other uses, on the other hand, are gaining popularity.

It's becoming more difficult to keep track of everything at the same time as the population and data grows. These issues can be solved with the use of computer vision. The following fields rely heavily on computer vision technology. A scanner recognizes and identifies text in documents using optical character recognition (OCR), iris patterns applied in vision biometrics to identify people who have gone missing, object Recognition: Ideal for finding products in real-time. Based on an image or scan in the retail and fashion industries, motion capture and shape capture, and any film that uses CGI, are examples of special effects, 3-D printing and image capture used in films, architectural structures, and other applications, use of Computer vision in sports to track the field and the players, anything with a story that allows you to wear something on your face on social media is acceptable, smart cars can recognize objects and people using computer vision, medical imaging: 3D imaging and image-guided surgery.

In this project performed a performance evaluation of the three face recognition models. The first model is implemented using MTCNN by me(https://colab.research.google.com/drive/1oGlIL_9TgYrrMBkJiLYbOmL-3Da3sOVE#scrollTo=6yhqJXD_gtL3). The second model is implemented with Haar Cascading using OpenCV by Dev Dash(https://colab.research.google.com/drive/1qcN9g1oZI76ua9QscJs3rm3MZI64BSC3#scrollTo=I5fba1ec2DhC). Finally the third model is implemented using a pre-trained model by Adrian Rosebrock(https://colab.research.google.com/drive/1vU_ITQS0m5o8uX5Owsw_NgeTe2qwCRop?hl=tr). Next section explains the models' structures.

MTCNN is a python (pip) library created by Github user ipacz that implements Zhang, Kaipeng, et alarticle. .'s "Multitask Cascaded Convolutional Networks for Joint Face Detection and Alignment." IEEE Signal Processing Letters, vol. 23, no. 10 (October 2016), pp. 1499–1503. Crossref. I prefer to use MTCNN because it is an extremely accurate and reliable algorithm. Even with varying sizes, lighting, and severe rotations, it correctly recognises faces. Face recognition with OpenCV is quick and accurate thanks to a pre-trained deep learning face detector model included with the library. OpenCV comes with pre-trained Haar cascades that may be used for face detection right out of the box. Since OpenCV 3.3, a deep learning-based face detector has been included in the library.

2. Models

2.1. Face Recognition Algorithm - 1

The algorithm developed face detection using MTCNN. The algorithm has five steps. These are:

1. Some Basic Steps

```
!pip install mtcnn
import cv2
import mtcnn
from mtcnn.mtcnn import MTCNN
import matplotlib.pyplot as plt

image_path = "omar-lopez-1qfy-jDc_jo-unsplash.jpg"
print(mtcnn.__version__)
```

2. Define method to detect face in an image using MTCNN

```
def detect_face(image):
    detector = MTCNN()
    bounding_boxes = detector.detect_faces(image)
    return bounding_boxes
```

3. Define method to draw bounding box around the face in an image

```
def draw_bounding_boxes(image, bboxes):
    for box in bboxes:
        x1, y1, w, h = box['box']
        cv2.rectangle(image, (x1, y1), (x1+w,y1+h),
(0,255,0), 2)
```

4. Define method to mark key points on face

```
def mark_key_point(image, keypoint):
    cv2.circle(image, (keypoint), 1, (0,255,0), 2)
```

5. Test with an image to detect a face

```
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

bboxes = detect_face(image)
print("Output of MTCNN detector is...\n",bboxes)
from google.colab import drive
drive.mount('/content/drive')
# draw bounding box around the detected face and mark
facial keypoints
draw_bounding_boxes(image, bboxes)
mark_key_point(image, bboxes[0]['keypoints']['left_eye'])
mark_key_point(image,
bboxes[0]['keypoints']['right_eye'])
mark_key_point(image, bboxes[0]['keypoints']['nose'])
mark_key_point(image,
bboxes[0]['keypoints']['mouth_left'])
mark_key_point(image,
bboxes[0]['keypoints']['mouth_right'])

# display the image
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.xticks([])
plt.yticks([])
plt.show()
```

2.2. Face Recognition Algorithm - 2

The algorithm developed face detection with Face Recognition Library and Haar Cascading using OpenCV.

The algorithm has four steps. These are:

1. Load Libraries

```
#@title Load libraries
!pip install -q face_recognition
```

```
!pip install -q fer
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import matplotlib.patches as patches
%matplotlib inline
import face_recognition
import numpy as np
from PIL import ImageDraw
import PIL.Image
from io import BytesIO
from fer import FER
import cv2
import operator
import time
from google.colab import files
from google.colab.patches import cv2_imshow
detector = FER()
```

2. Upload ımages with Faces

```
#@title Upload image with faces
      uploaded = files.upload()
```

3. Use Face Recognition Library and Draw a Red Box Around the Faces as Well As Predicted Emotion

```
#@title Use 'Face Recognition' library and draw a red box
around the faces as well as predicted emotion
start = time.time()
face_locations = face_recognition.face_locations(image)
fig,ax = plt.subplots(figsize=(20,height))
for face_location in face_locations:
   top, right, bottom, left = face_location
   ax.imshow(im,aspect='auto')
   rect = patches.Rectangle((left,top), (right - left),
(bottom-top),linewidth=3,edgecolor='r',facecolor='none')
   ax.add_patch(rect)
   face_image = image[top-boundary:bottom+boundary,
left-boundary:right+boundary]
   attribute = detector.detect_emotions(face_image)
   if (not attribute) == False:
```

```
    emotion =
max(attribute[0]['emotions'].items(),key=operator.itemget
ter(1))[0]
    #print(emotion)
    plt.text(left, top, emotion, fontsize=8,
bbox=dict(fill=True, edgecolor='blue', linewidth=1))
end = time.time()
print("Number of faces:",len(face_locations))
print("Time taken:",end-start," seconds")
```

4. Use Haar Cascades and Draw A Red Box Around The Faces

```
#@title Use Haar Cascades and draw a red box around the
faces
start = time.time()
#Load the cascade
face_cascade =
cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

#Read the input image
img = cv2.imread(list(uploaded.keys())[0])

#Convert into GrayScale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#Detect Faces
faces = face_cascade.detectMultiScale(gray, 1.3, 4)
#Draw rectangle around each faces
for (x, y, w, h) in faces:
   cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0,
255), 3)
#Display the output
ims = cv2.resize(img,(im.size[0],im.size[1]))
cv2_imshow(ims)
cv2.waitKey()
end = time.time()
print("Time taken:",end-start," seconds")
print("Number of faces:",len(faces))
```

2.3. Face Recognition Algorithm - 3

The algorithm developed face detection using a pre-trained model.

The algorithm has ten steps. These are:

1. Import required Python libraries

```
import imutils
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode
```

2. Start webcam

```
def take_photo(filename='photo.jpg', quality=0.8):
 js = Javascript('''
   async function takePhoto(quality) {
     const div = document.createElement('div');
     const capture = document.createElement('button');
     capture.textContent = 'Capture';
     div.appendChild(capture);

     const video = document.createElement('video');
     video.style.display = 'block';
     const stream = await
navigator.mediaDevices.getUserMedia({video: true});

     document.body.appendChild(div);
     div.appendChild(video);
     video.srcObject = stream;
     await video.play();

     // Resize the output to fit the video element.
```

```
google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
 display(js)
 data = eval_js('takePhoto({})'.format(quality))
 binary = b64decode(data.split(',')[1])
 with open(filename, 'wb') as f:
   f.write(binary)
 return filename
```

3. Click 'Capture' to take a photo using your webcam.

```
image_file = take_photo()
```

4. Read, resize and display the image.

```
#image = cv2.imread(image_file, cv2.IMREAD_UNCHANGED)
image = cv2.imread(image_file)

# resize it to have a maximum width of 400 pixels
image = imutils.resize(image, width=400)
(h, w) = image.shape[:2]
print(w,h)
cv2_imshow(image)
```

5. OpenCV's deep learning face detector is based on the Single Shot Detector (SSD) framework with a ResNet base network. The network is defined and trained using the Caffe Deep Learning framework

   Download the pre-trained face detection model, consisting of two files:

   The network definition (deploy.prototxt)

   The learned weights (res10_300x300_ssd_iter_140000.caffemodel)

```
!wget -N https://raw.githubusercontent.com/opencv/opencv/master/samples/dnn/face_detector/deploy.prototxt
!wget -N https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20170830/res10_300x300_ssd_iter_140000.caffemodel
```

6. Load the pre-trained face detection network model from disk

```
print("[INFO] loading model...")
prototxt = 'deploy.prototxt'
model = 'res10_300x300_ssd_iter_140000.caffemodel'
net = cv2.dnn.readNetFromCaffe(prototxt, model)
```

7. Use the dnn.blobFromImage function to construct an input blob by resizing the image to a fixed 300x300 pixels and then normalizing it.

```
# resize it to have a maximum width of 400 pixels
image = imutils.resize(image, width=400)
blob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0, (300, 300), (104.0, 177.0, 123.0))
```

8. Pass the blob through the neural network and obtain the detections and predictions.

```
print("[INFO] computing object detections...")
net.setInput(blob)
```

```
detections = net.forward()
```

9. Loop over the detections and draw boxes around the detected faces

```
for i in range(0, detections.shape[2]):
 # extract the confidence (i.e., probability) associated
with the prediction
 confidence = detections[0, 0, i, 2]

 # filter out weak detections by ensuring the
`confidence` is
 # greater than the minimum confidence threshold
 if confidence > 0.5:
    # compute the (x, y)-coordinates of the bounding box
for the object
    box = detections[0, 0, i, 3:7] * np.array([w, h, w,
h])
    (startX, startY, endX, endY) = box.astype("int")
    # draw the bounding box of the face along with the
associated probability
    text = "{:.2f}%".format(confidence * 100)
    y = startY - 10 if startY - 10 > 10 else startY + 10
    cv2.rectangle(image, (startX, startY), (endX, endY), (0,
0, 255), 2)
    cv2.putText(image, text, (startX, y),
      cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
```

10. Show the resulting image

```
cv2_imshow(image)
```

3. Outputs

In this project the given input for the three models. The input image consists of 26 people as shown in Figure 1.

Figure 1. Input of the project.

Here is the output of the first model which is implemented using MTCNN. The model detected 27 faces with 1 wrong detection in 30.9 seconds as shown in Figure 2.

Figure 2. Output of the Model-1.

Here is the output of the second model which is implemented using Face Recognition Library. The model detected 24 faces with 2 wrong detections in 2.43 seconds as shown in Figure 3.

Figure 3. Output of the Model-2.

Here is the output of the second model which is implemented with Haar Cascading using OpenCV. The model detected 25 faces with 4 wrong detections in 1.95 seconds as shown in Figure 4.

Figure 4. Output of the Model-2.

Here is the output of the third model which is implemented using a pre-trained model. The model couldn't recognize faces from the original input so that the original input zoomed. Then the model could detect a face from 26 people as shown in Figure 5.

Figure 5. Output of the Model-3.

More zoomed, the model detected 9 faces in 9 faces with 0 wrong detections as shown in Figure 6.

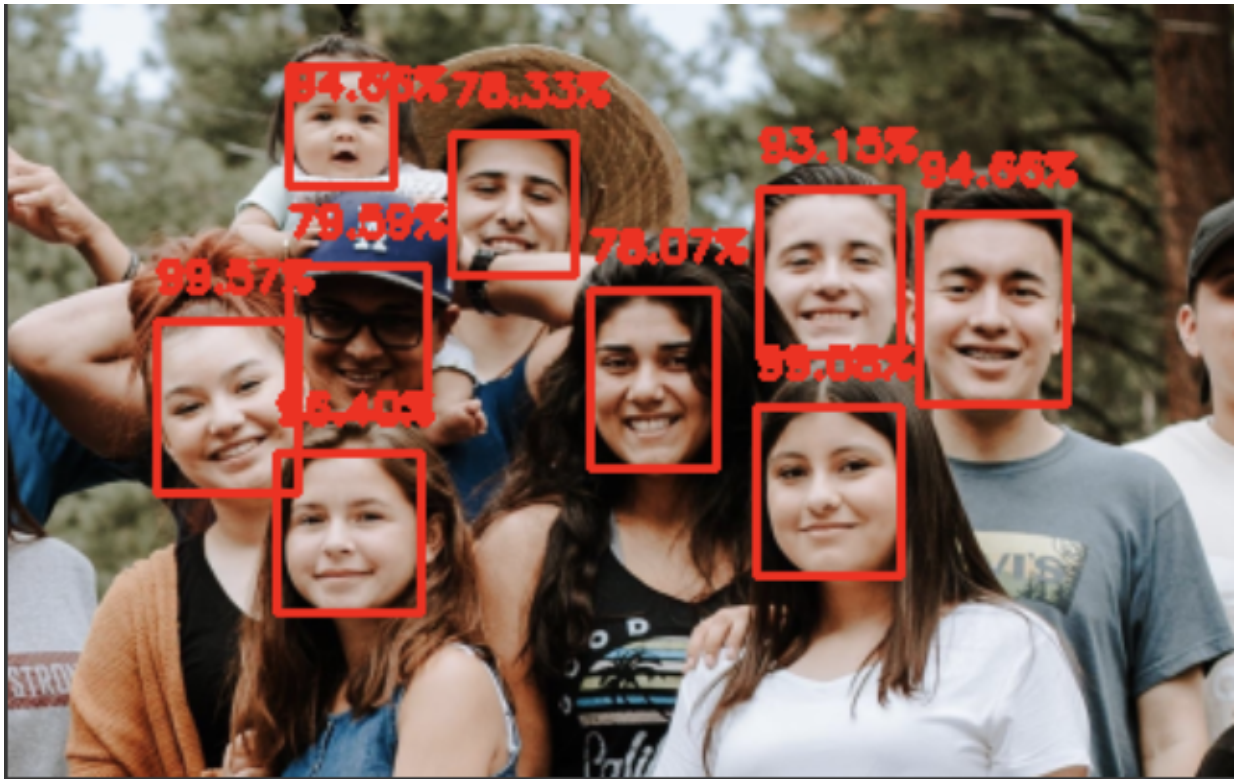

Figure 6. Output of the Model-3.

Finally more zoomed, the model detected 11 faces in 13 faces with 2 wrong detections in 1.23 seconds as shown in Figure 7.

Figure 7. Output of the Model-3.

4. Performance Evaluations & Results

Model-1 has better accuracy than Model-2. Model-3 has better accuracy than Model-2. So that Model-1 has the best accuracy rate. MTCNN identifies a false negative since it is more sensitive by default. We can quickly alter the thresholds for MTCNN with the thresholds property in such instances, which is quite beneficial. Model-2 is faster than Model-3 and Model-3 is faster than Model-1. So the fastest algorithm is Model-2.

5. Conclusion

The first model which is implemented using MTCNN has the best accuracy rate. The model detected 27 faces with 1 wrong detection. The model can speed up x100 times actually. If MTCNN is run on a GPU and the sped-up version is used, 60–100 pictures/frames per second can be achieved. That's a 100-fold increase in power. If the program to extract all of the faces from a movie at a rate of 10 faces per second (one second of a movie has on average approximately 24 frames, so every second frame), the total number of frames would be 10 * 60 (seconds) * 120 (minutes) = 72,000. It will take 72,000 * 1 (seconds) = 72,000s / 60s = 1,200m = 20 hours to process one frame if it takes one second to process one frame. This work will take 72,000 (frames) / 100 (frames/sec) = 720 seconds = 12 minutes with the sped-up version of MTCNN.

Another option is to use Haar cascades, which has its own set of pitfalls. It misses some faces, in particular, mistaking a pant leg, ankle, and tip of a shoe for a face. However, combining Haar cascades and OpenCV is substantially faster than using the face recognition library in every iteration, about 20–30 percent faster at times. This kind of speed boost is huge, especially when dealing with hundreds of photos (e.g. video). It also refers to a procedure that is expected to be less computationally expensive and performed on less expensive technology that consumes less energy. Of course, these aren't the only two ways of detecting faces, and there are probably better ones out there. If you were to choose between the two, I don't believe there is a clear winner – one is better at detecting (face recognition), but the other is far faster (Haar cascades).

OpenCV library — When compared to OpenCV's Haar cascades, OpenCV comes with a better accurate face detector out of the box. Deep learning is used in the more accurate OpenCV face detector, which uses the Single Shot Detector (SSD) architecture with ResNet as the foundation network.

6. References

[1]A. Rosebrock, "Face detection with OpenCV and deep learning - PyImageSearch", *PyImageSearch*, 2022. [Online]. Available: https://pyimagesearch.com/2018/02/26/face-detection-with-opencv-and-deep-learning/. [Accessed: 31- May- 2022]

[2]"Two Step Facial Recognition With Colab", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/two-step-facial-recognition-with-colab-883a54faafad. [Accessed: 31- May- 2022]

[3]"Face Detection using MTCNN—a guide for face extraction with a focus on speed", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/face-detection-using-mtcnn-a-guide-for-face-extraction-with-a-focus-on-speed-c6d59f82d49. [Accessed: 31- May- 2022]

[4]"Robust face detection with MTCNN", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/robust-face-detection-with-mtcnn-400fa81adc2e#:~:text=MTCNN%20is%20very%20accurate%20and,get%20RGB%20images%20as%20input. [Accessed: 31- May- 2022]

[5]"Implementing Computer Vision - Face Detection - Analytics Vidhya", *Analytics Vidhya*, 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/12/implementing-computer-vision-face-detection/. [Accessed: 31- May- 2022]

[6]J. Brownlee, "How to Perform Face Detection with Deep Learning", *Machine Learning Mastery*, 2022. [Online]. Available: https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/. [Accessed: 31- May- 2022]

[7]"What is Face Detection and How Does It Work?", *SearchEnterpriseAI*, 2022. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/face-detection. [Accessed: 31- May- 2022]

[8]"Face Recognition for Beginners", *Medium*, 2022. [Online]. Available: https://towardsdatascience.com/face-recognition-for-beginners-a7a9bd5eb5c2. [Accessed: 31- May- 2022]

[9]"Computer Vision And Facial Recognition: Real World Applications", *Knowledgenile.com*, 2022. [Online]. Available: https://www.knowledgenile.com/blogs/computer-vision-and-facial-recognition-real-world-applications/. [Accessed: 31- May- 2022]

[10]"Face Detection in 2022: Real-time applications with deep learning (Updated) - viso.ai", *viso.ai*, 2022. [Online]. Available: https://viso.ai/deep-learning/face-detection-overview/. [Accessed: 31- May- 2022]