Modern Face Recognition with Deep Learning

PROJECT DONE BY-
NISHANTH(15I227)
RAHUL(16I442)
SUBASH.G(16I444)

ABSTRACT

Face recognition is widely used in computer vision and in many other biometric applications where security is a major concern. The most common problem in recognizing a face arises due to pose variations, different illumination conditions and so on. The main focus of this paper is to recognize whether a given face input corresponds to a registered person in the database. Face recognition is done using Histogram of Oriented Gradients (HOG) technique with an inclusion of a real time subject to evaluate the performance of the algorithm. The feature vectors generated by HOG descriptor are used to train Support Vector Machines (SVM) and results are verified against a given test input. The proposed method checks whether a test image in different pose and lighting conditions is matched correctly with trained images of the facial database. The results of the proposed approach will detect the faces in an image and recognise and say who they are.

INTRODUCTION

Human has special skill in analyzing and interpreting faces, and so face analyzing has an important role in man machine relationship and different research areas has been opened in this way. Face detection is considered to be the first task performed while processing scenes for varied purposes and its results are important for subsequent steps of automated human face recognition. Therefore the whole process should work predictably and quite reliably. The increased need of security in the country, both in Unconstrained and constrained areas and the rapid developments in the field of computer vision have paved much progress in face detection and face recognition system. These systems designs can be employed in surveillance and monitoring, biometrics, traffic assistance, health care, etc. Face detection is differentiating the face from any other objects (inter-class variability). Face recognition differentiating ones face from the other (intra-class variability). Face detection and recognition poses challenging task as detecting any other objects and face has various facial features and color varies dynamically. Processing of a face in real time with occlusions, background structure and camera position adds to the existing challenges.

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

The hardware components we used to develop this project are as follows:

Processor	: Intel core i7 and above version
RAM	: Minimum of 4GB
Graphics card	: Nvidia 650 and above
Hard Disk	: 10GB
Camera	

SOFTWARE REQUIREMENTS

The software components we used to develop this project as follows:

Back end	: Python3
Operating system	: Linux

Thus these are the various software specifications that are required for the application to function properly.

Installation Options:

Installing on Mac or Linux

First, make sure you have dlib already installed with Python bindings:

For a full list of apt packages required, check out the Dockerfile and copy what's installed there.

```
# This is a sample Dockerfile you can modify to deploy your own app based on face_recognition
FROM python:3.6-slim-stretch

RUN apt-get -y update
RUN apt-get install -y --fix-missing \
  build-essential \
  cmake \
  gfortran \
  git \
  wget \
  curl \
  graphicsmagick \
  libgraphicsmagick1-dev \
  libatlas-dev \
  libavcodec-dev \
  libavformat-dev \
  libgtk2.0-dev \
  libjpeg-dev \
  liblapack-dev \
  libswscale-dev \
```

```
    pkg-config \
    python3-dev \
    python3-numpy \
    software-properties-common \
    zip \
    && apt-get clean && rm -rf /tmp/* /var/tmp/*

RUN cd ~ && \
    mkdir -p dlib && \
    git clone -b 'v19.9' --single-branch https://github.com/davisking/dlib.git dlib/ && \
    cd  dlib/ && \
    python3 setup.py install --yes USE_AVX_INSTRUCTIONS


# The rest of this file just runs an example script.

# If you wanted to use this Dockerfile to run your own app instead, maybe you would do
this:
# COPY . /root/your_app_or_whatever
# RUN cd /root/your_app_or_whatever && \
#     pip3 install -r requirements.txt
# RUN whatever_command_you_run_to_start_your_app

COPY . /root/face_recognition
RUN cd /root/face_recognition && \
    pip3 install -r requirements.txt && \
    python3 setup.py install

CMD cd /root/face_recognition/examples && \
    python3 recognize_faces_in_pictures.py
```

These instructions assume you are using Ubuntu 16.04 or newer.

Clone the code from github:

```
git clone https://github.com/davisking/dlib.git
```

Then, install this module from pypi using pip3 (or pip2 for Python 2):

```
pip3 install face_recognition
```

Build the main dlib library (optional if you just want to use Python):

```
cd dlib
mkdir build; cd build; cmake .. -DDLIB_USE_CUDA=0
-DUSE_AVX_INSTRUCTIONS=1;          cmake --build .
```

Build and install the Python extensions:

```
cd ..
python3 setup.py install --yes USE_AVX_INSTRUCTIONS --no
DLIB_USE_CUDA
```

At this point, you should be able to run python3 and type import dlib successfully.

MODULES

Module 1 - Finding all the faces in an image

To find faces in an image we use Histogram of Oriented Gradients algorithm is a feature descriptor used for the purpose of object detection. This technique counts occurrences of gradient orientation in localized portions of an image.

CODE- To find faces in an image

```
from PIL import Image
import face_recognition

# Load the jpg file into a numpy array
image = face_recognition.load_image_file("sachin.jpg")

# Find all the faces in the image using the default HOG-based model.
# This method is fairly accurate, but not as accurate as the CNN model and not GPU
accelerated.
# See also: find_faces_in_picture_cnn.py
face_locations = face_recognition.face_locations(image)

print("I found {} face(s) in this photograph.".format(len(face_locations)))

for face_location in face_locations:

  # Print the location of each face in this image
  top, right, bottom, left = face_location
  print("A face is located at pixel location Top: {}, Left: {}, Bottom: {}, Right:
{}".format(top, left, bottom, right))

  # You can access the actual face itself like this:
  face_image = image[top:bottom, left:right]
  pil_image = Image.fromarray(face_image)
  pil_image.show()
```

OUTPUT-

Module 2 - Extracting facial features in an image

To extracting facial features in an image we use an algorithm called face landmark estimation. Face landmark estimation will come up with 68 specific points that exist on every face - the top of the chin, the outside edge of each eye, the inner edge of

each eyebrow, etc., The extracted features are encoded.
Code to extracting facial features in an image

```
from PIL import Image, ImageDraw
import face_recognition

# Load the jpg file into a numpy array
image = face_recognition.load_image_file("sachin.jpg")

# Find all facial features in all the faces in the image
face_landmarks_list = face_recognition.face_landmarks(image)

print("I found {} face(s) in this photograph.".format(len(face_landmarks_list)))

for face_landmarks in face_landmarks_list:

  # Print the location of each facial feature in this image
  facial_features = [
      'chin',
      'left_eyebrow',
      'right_eyebrow',
      'nose_bridge',
      'nose_tip',
      'left_eye',
      'right_eye',
      'top_lip',
      'bottom_lip'
  ]

  for facial_feature in facial_features:
      print("The {} in this face has the following points: {}".format(facial_feature,
face_landmarks[facial_feature]))

  # Let's trace out each facial feature in the image with a line!
  pil_image = Image.fromarray(image)
  d = ImageDraw.Draw(pil_image)

  for facial_feature in facial_features:
      d.line(face_landmarks[facial_feature], width=5)

  pil_image.show()
```

OUTPUT-

Module 3 - Identify faces in pictures

To identify the faces in picture we use support vector machines models with associated learning algorithms that analyse data used for classification and regression analysis. Train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. The result of the classifier is the name of the person.

Code-

```
import face_recognition

# Load the jpg files into numpy arrays
sachin_image = face_recognition.load_image_file("sachin.jpg")

unknown_image = face_recognition.load_image_file("sac.jpg")

# Get the face encodings for each face in each image file
# Since there could be more than one face in each image, it returns a list of encodings.
# But since I know each image only has one face, I only care about the first encoding in each image, so I grab index 0.
try:
  sachin_face_encoding = face_recognition.face_encodings(sachin_image)[0]
  unknown_face_encoding = face_recognition.face_encodings(unknown_image)[0]
except IndexError:
  print("I wasn't able to locate any faces in at least one of the images. Check the image files. Aborting...")
  quit()

known_faces = [
  sachin_face_encoding
]

# results is an array of True/False telling if the unknown face matched anyone in the known_faces array
results = face_recognition.compare_faces(known_faces, unknown_face_encoding)

print("Is the unknown face a picture of sachin? {}".format(results[0]))
print("Is the unknown face a new person that we've never seen before? {}".format(not True in results))
```

OUTPUT-

Module 4 Integrating the modules

Integrating the above modules together with webcam.

Code-

```
import cv2
import face_recognition
```

```
# This is a demo of running face recognition on live video from your webcam. It's a little more complicated than the
# other example, but it includes some basic performance tweaks to make things run a lot faster:
#   1. Process each video frame at 1/4 resolution (though still display it at full resolution)
#   2. Only detect faces in every other frame of video.

# PLEASE NOTE: This example requires OpenCV (the `cv2` library) to be installed only to read from your webcam.
# OpenCV is *not* required to use the face_recognition library. It's only required if you want to run this
# specific demo. If you have trouble installing it, try any of the other demos that don't require it instead.

# Get a reference to webcam #0 (the default one)
video_capture = cv2.VideoCapture(0)

# Load a sample picture and learn how to recognize it.
Rubaesh_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Rubaesh.jpg")
Rubaesh_face_encoding = face_recognition.face_encodings(Rubaesh_image)[0]

Nagarajan_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Nagarajan.jpg")
Nagarajan_face_encoding = face_recognition.face_encodings(Nagarajan_image)[0]

Bala_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Bala.jpg")
Bala_face_encoding = face_recognition.face_encodings(Bala_image)[0]

Subash_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Subash.jpg")
Subash_face_encoding = face_recognition.face_encodings(Subash_image)[0]

Venkat_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Venkat.jpg")
Venkat_face_encoding = face_recognition.face_encodings(Venkat_image)[0]

Sachin_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Sachin.jpg")
Sachin_face_encoding = face_recognition.face_encodings(Sachin_image)[0]
Python_mam_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Python_mam.jpg")
```

```
Python_mam_face_encoding =
face_recognition.face_encodings(Python_mam_image)[0]

Rahul_image =
face_recognition.load_image_file("/home/rubaesh/Phase2/Train/Rahul.jpg")
Rahul_face_encoding = face_recognition.face_encodings(Rahul_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    Rubaesh_face_encoding,
    Venkat_face_encoding,
    Bala_face_encoding,
    Subash_face_encoding,
    Nagarajan_face_encoding,
    Sachin_face_encoding,
    Python_mam_face_encoding,
    Rahul_face_encoding

]
known_face_names = [
    "Rubaesh",
    "Venkat",
    "Bala",
    "Subash",
    "Nagarajan",
    "Sachin",
    "Python_mam",
    "Rahul"

]

# Initialize some variables
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    # Grab a single frame of video
    ret, frame = video_capture.read()

    # Resize frame of video to 1/4 size for faster face recognition processing
    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB color (which
```

```
face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame,
face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings,
face_encoding,tolerance=0.5)
            name = "Unknown"

            # If a match was found in known_face_encodings, just use the first one.
            if True in matches:
                first_match_index = matches.index(True)
                name = known_face_names[first_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame


    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale back up face locations since the frame we detected in was scaled to 1/4
size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        if name== "Unknown":
            cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
        else:
            cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

        # Draw a label with a name below the face
        if name == "Unknown":
            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        else:
```

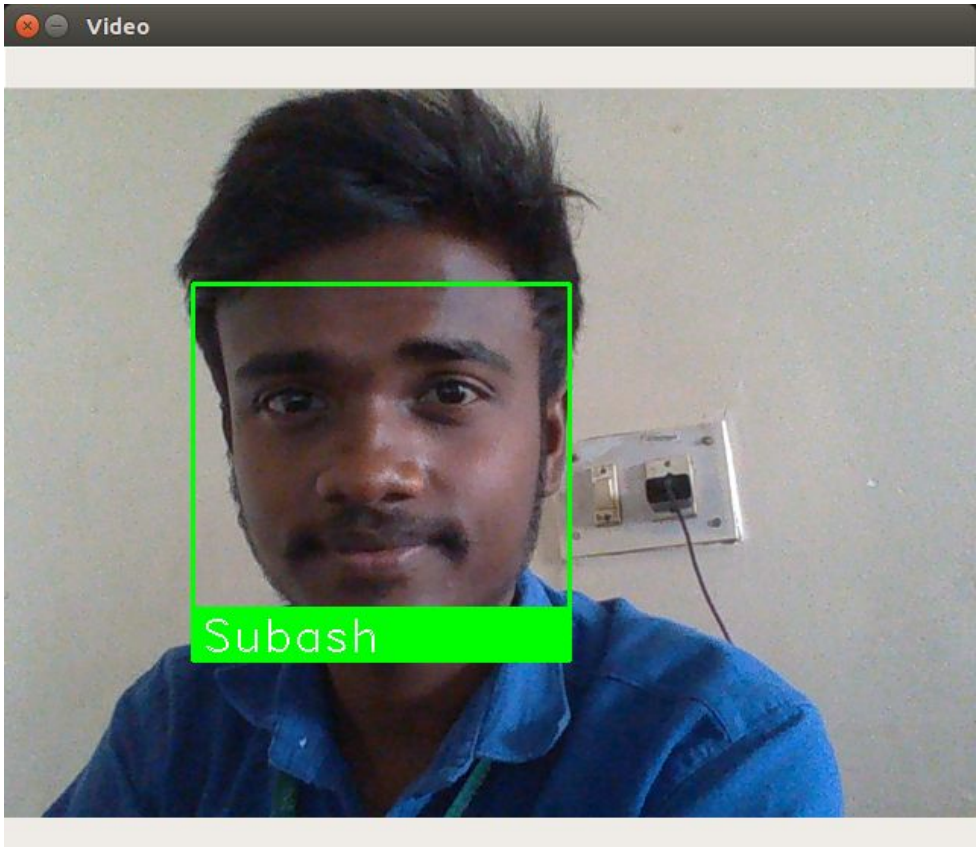
```
            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 255, 0), cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

        # Display the resulting image
        cv2.imshow('Video', frame)

        # Hit 'q' on the keyboard to quit!
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release handle to the webcam
video_capture.release()
cv2.destroyAllWindows()
```

OUTPUT-



APPLICATIONS

Face Detection technology may be used as a core component (basis) for a great number of applications with a wide sphere of usage.

Smart captcha

Webcam based energy/power saver.

Time tracking service.

Outdoor surveillance camera service.

Video chat service.

CONCLUSION

Face detection developed based on HOG descriptor is working well in real time. Face visualization models implemented based on HOG descriptor is able to visualize a face in any orientation from +900 to -900 and detection rate varies in few seconds depending on the faces that contained in the video frame. In future this method is helpful in face recognition systems.

REFERENCES

1.https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

2. https://github.com/ageitgey/face_recognition