Design Implementation of DeepFace CNN

Swayam Swaroop Mishra
ID - 013725595
Computer Engineering Department
San José State University (SJSU)
San José, CA, USA
Email: swayam.mishra@sjsu.edu

***Abstract*—DeepFace is a deep learning facial recognition system created by a research group in Facebook Labs. It identifies human faces in digital images. It employs a nine-layer neural network with over 120 million connection weights and was trained on four million images uploaded by Facebook users. This paper shows the implementation of DeepFace and evaluate its performance on images, video and live stream from USB camera.**

***Index Terms*—DeepFace, OpenCV, Python 3.7**

I. Introduction

While face recognition has been around in one form or another since the 1960s, recent technological developments have led to a wide proliferation of this technology. This technology is no longer seen as something out of science fiction movies like Minority Report. With the release of the iPhone X, millions of people now literally have face recognition technology in the palms of their hands, protecting their data and personal information. While mobile phone access control might be the most recognizable way face recognition that is being used, it is implemented in many fields including preventing crime, protecting events and making air travel more convenient.

Some example of face recognition currently being used in today's world are:

- Prevent Retail Crime: Face recognition is currently being used to instantly identify when known shoplifters, organized retail criminals or people with a history of fraud enter retail establishments. According data, face recognition reduces external shrink by 34% and, more importantly, reduces violent incidents in retail stores by up to 91%.
- Find Missing Persons: Face recognition can be used to find missing children and victims of human trafficking. 3000 missing children were discovered in just four days of using face recognition in India.
- Protect Law Enforcement: Mobile face recognition apps, like the one offered by FaceFirst, are already helping police officers by helping them instantly identify individuals in the field from a safe distance. This can help by giving them contextual data that tells them who they are dealing with and whether they need to proceed with caution.
- Diagnose Diseases: Face recognition can be used to diagnose diseases that cause detectable changes in appearance. As an example, the National Human Genome Research Institute, uses face recognition to detect a rare disease called DiGeorge syndrome, in which there is a portion of the 22nd chromosome missing. Face recognition has helped to diagnose the disease in 96% of cases.

This paper exhibits DeepFace implementation using Python 3.7 and OpenCv for detecting people from still frame or images, video clips and from USB camera for real time detection.

II. Methodology

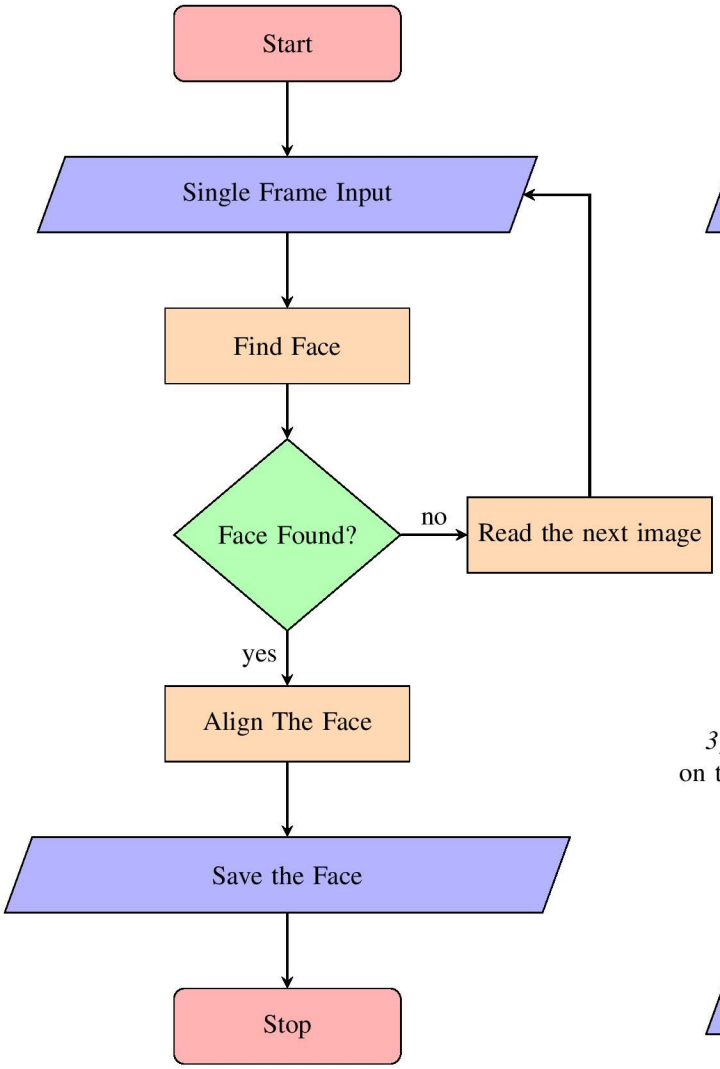
A. Algorithm Description

- Create a directory called dataset and create a sub-directory by the persons name and store their images in it.
- Import pre-trained face detector and face aligner.
- Iterate over every image and try to find the face using the face detector.
- If a face is found, align it and normalize the image to 228x228 image.
- Embed the normalize image to a pickle file. Train the support vector machine learning model and then store the model to another pickle file.
- For testing either provide a still image, a video or live video stream from a USB camera/ web camera. From a single frame extract the face. Normalize the face and pass that to the training model for prediction.
- If the face is recognised, display the person's name or else display unknown.

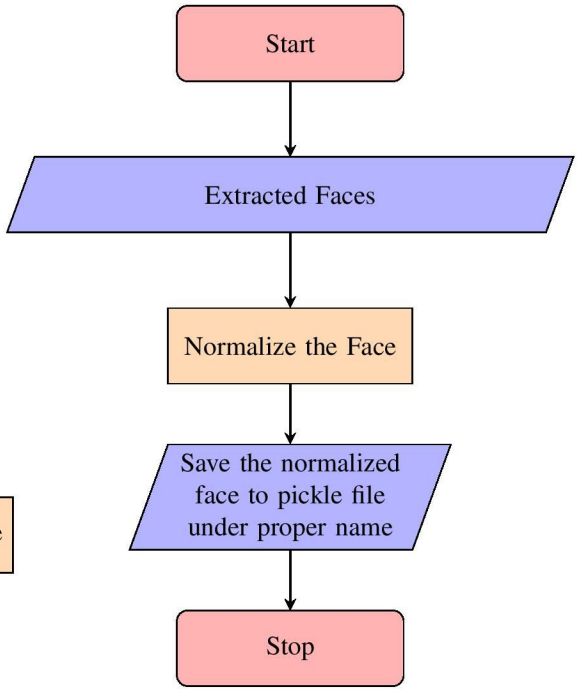
B. Flow Charts

The flow charts of the algorithm are illustrated below:

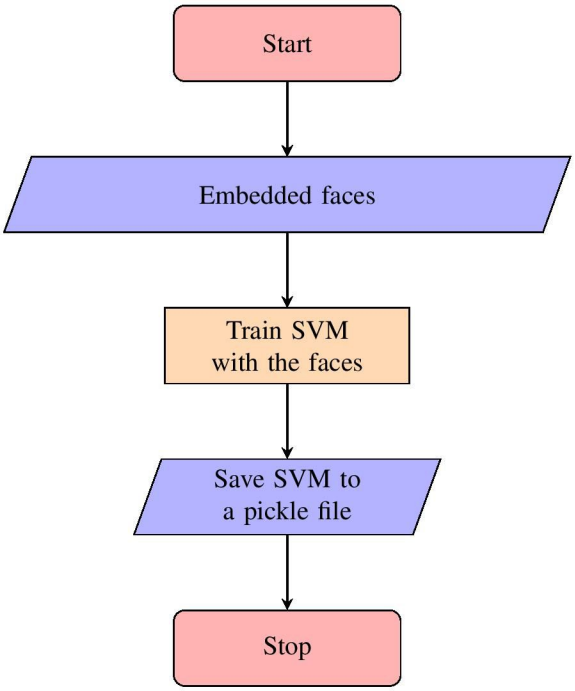
1) Face Tracker1: Face detection and saving region of interest(face) - faceTracker1.py.



2) Face Tracker2: Embedding the detected the face to a pickle file - faceTracker2.py.

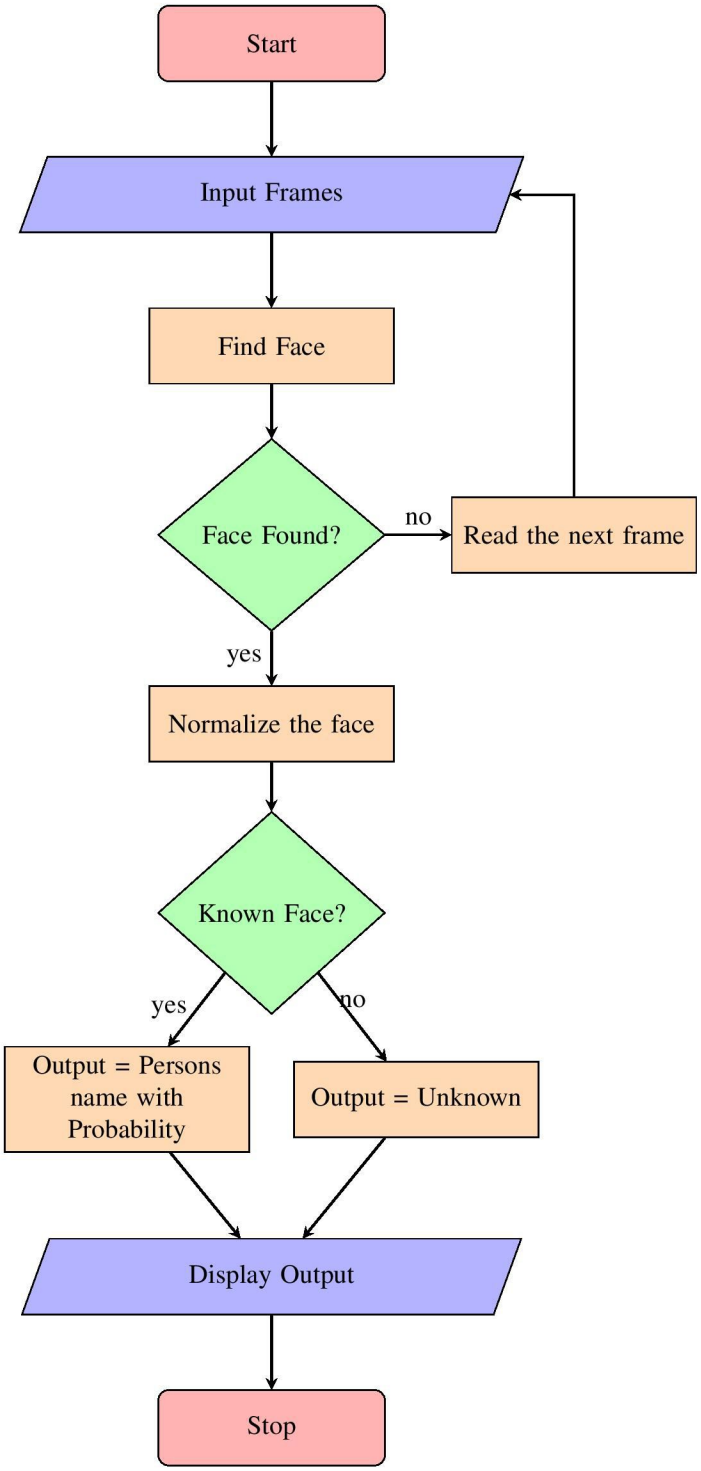


3) DeepFace Trained 1: Training support vector machine on the embedded face - deepFaceTrained1.py.



4) DeepFace Trained 2: Testing the model by giving an unlabeled face and finding what the model predicts -

- deepFaceTrained2.py - testing trained faces in images.
- deepFaceTrained2_video.py - testing trained faces in videos.
- deepFaceTrained2_webCam.py - testing trained faces in live stream.



C. Pseudo Code

1) Face Extraction and Creating Embedding::

- Initialize path to dataset.
- Load serialized face embedding model from disk
- Initialize dlib's face detector and then create the facial landmark predictor and the face aligner.
- Loop over the image paths
  - Extract name of person from image path.
  - Read the image using OpenCV.
  - Resize the image and convert it to grayscale.
  - Construct a blob from the image
  - Apply OpenCV's deep learning-based face detector to localize faces in the input image.
  - If a face is found
    - Compute the (x, y)-coordinates of the bounding box for the face
    - Extract the face ROI and grab the ROI dimensions. And ensure the face width and height are sufficiently large.
    - Detect face and return bounding box
    - Transform or align the image using Outer eyes and nose landmark indices and crop image.
    - Normalize the image to 228x228.
    - Add the name of the person + corresponding face embedding to their respective lists
    - Dump the facial embeddings + names to disk using a pickle file.

2) Training:

- Load the face embeddings.
- Encode the labels.
- Train the model using the embeddings of the face and then produce the actual face recognition.
- Write the actual face recognition model to disk using a pickle file.
- Write the label encoder to disk

3) Testing:

- Load our serialized face detector from disk.
- Load our serialized face embedding model from disk.
- Load the actual face recognition model along with the label encoder.
- Initialize the video stream.
- Start the FPS throughput estimator.
- Loop over frames from the video file stream.
  - Grab the frame from the threaded video stream.
  - Resize the frame and then grab the image dimensions.
  - Construct a blob from the image and apply OpenCV's deep learning-based face detector to localize faces in the input image blob.
  - Loop over the detections.
    - Extract the probability associated with the prediction.
    - Filter out weak detections.
    - Compute the (x, y)-coordinates of the bounding box for the face.
    - Extract the face ROI.
    - Ensure the face width and height are sufficiently large.
    - Construct a blob for the face ROI and then pass the blob through our face embedding model.
    - Perform classification to recognize the face.
    - Draw the bounding box of the face along with the associated probability.

  - update our centroid tracker using the computed set of bounding box rectangles
  - loop over the tracked objects
    * draw both the ID of the object and the centroid of the object on the output frame.
  - Update the FPS counter.
  - show the output frame.
  - break from the loop if 'q' is pressed.
- Stop the timer and display FPS information.
- Do a bit of cleanup.

III. Testing and Verification

I have three python scripts, which can recognize faces from still images, recorded videos and from USB camera or web camera. The below figures represent the test that were performed while testing the performance of the model.

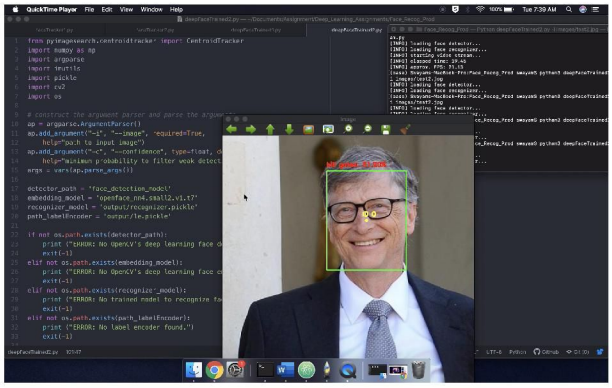


Fig. 1. Test on still image

Figure 1 shows the test on a still image. The recognised face is of Bill Gates and has a probability of 81%.

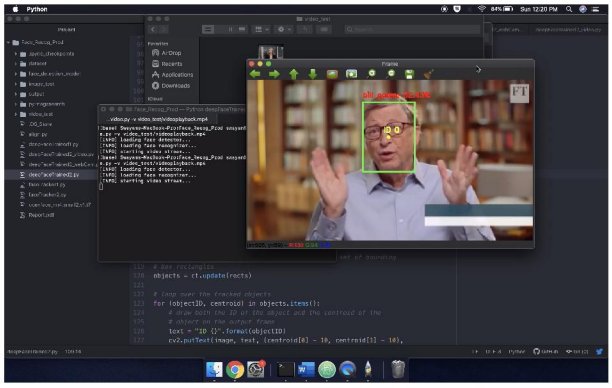


Fig. 2. Test on recorded video

Figure 2 shows the test on a recorded video. The recognised face is of bill gates and has a probability of 52%.

Figure 3 shows the test on a live stream. The recognised face is Swayam and has a probability of 48%. I got a low probability on my face because of less number of trained images.

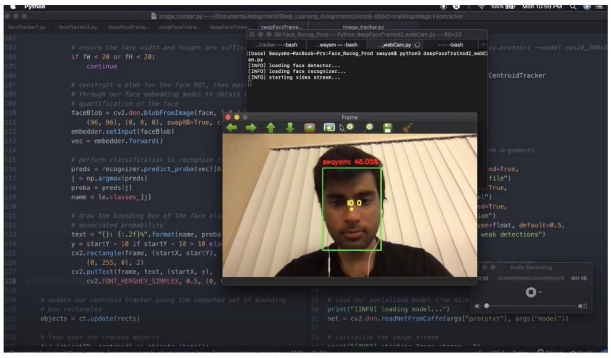


Fig. 3. Test on live video stream or web camera

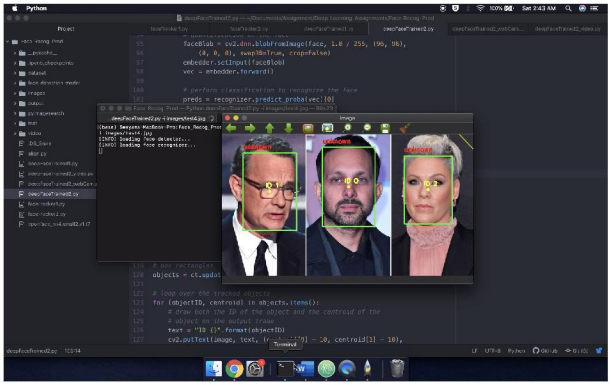


Fig. 4. Test on still image

Figure 4 shows the test on a still image. Since all the persons in the image are not trained and unknown to the model, so I get an output as unknown without any probability.

References

[1] https://github.com/hualili/opencv.
[2] https://www.facefirst.com/blog/amazing-uses-for-face-recognition-facial-recognition-use-cases/
[3] https://github.com/krasserm/face-recognition
[4] https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/
[5] https://www.pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/