Dlib in Python

Dlib:

Dlib is a real life problem solving library used in computer vision, which is originally written in C++, but has easy & compatible python bindings.

Dlib offers the art technique for face recognition tasks that actually used landmarks to estimate the location of 68 key-points which maps on facial points of a person's face.

Purpose of Dlib:

The salient features of face like eyes, nose, lips, eyebrows jawline etc. are localized and represented by the facial landmark detector (Dlib). This is usually used to check weather mouth is open or closed, Eyes are blinked or not.

So, this library is ultimately applied for checking:

- Face Alignment
- Blink Detection
- Head pose estimation etc.

Facial Landmarks:

For face detection problems, detection of facial landmarks is important factor. Landmark facial detectors attempt to localize the key pints of an input image (which specifies the region the interest (ROI)).

This is the two-step process for detecting facial landmarks:

1. Localize the face in an image
2. Detection of key facial structures of an input image in which ROI is specified.

There is no. of ways of face detection i.e.

- Histogram of Oriented Gradients (HOG) + Linear Support Vector Machine (SVM)
- Deep Learning based algorithms for face localization.
- Opencv's built in Haar cascades (Viola-Jones-Detector)

Pros and Cons of Viola-Jones-detector & HOG+ Linear SVM detector

Viola-Jones-Detector	HOG+ Linear SVM
• Extremely fast	• Moderate
• Makes tedious task while batch processing of larger datasets for face detection.	• There isn't such issue in this detector.

To remove the false positives in OpenCV while detecting human faces/Objects etc. it requires a lot of time. Still, there is no guarantee that exact same parameters will work from image to image.

Since, main concern is about:

- Detection of false positive objects
- Missing face detections entirely due to the selection of poor parameters per image basis.

This will make the it tedious tasks for (Viola-Jones-Detector) while batch processing of larger datasets for face detection.

Working of HOG + Linear SVM:

- Frontal descriptor consists of Histogram of Oriented Gradients (HOG) which is image descriptor and a linear support vector machine (SVM) could be used to train highly accurate object classifier or particularly face detectors.
- There are positive (contains region of interest (ROI) & negative samples (doesn't contain any ROI) extract HOG samples from both positive & negative samples.
- Then train a linear SVM on both +ve & -ve samples.
- After this applying hard-negative-sampling i.e. for each & every image with possible scale from negative training set, apply the sliding window approach & slide window from left-to-right & top-to-bottom across the images.
  Then apply HOG descriptor for each window & apply classifier. If classifier classifies the objects absolutely false positive, then record the probability of the classification along with false positive records associated with feature vector.
- Sort the false-positive samples that is found during hard-negative-mining according to their probability confidence.
  After this retrain the classifier using hard-negative samples.
- Then applied this trained dataset of classifier on test-data in the same way as it is applied during training i.e. slide window from top-to-bottom & right-to-left (for each scale of image).
  Then extract the Histogram of Oriented Gradients (HOG) descriptor of Objects & record bounding box of window.
- To overcome the overlapping issues (i.e. draw multiple bounding box around the object) apply non-maximum-suppression algorithm. The results are quite satisfactory and good as compare to other algorithms i.e. Mean shift algorithm.

Compatibility:

Ubuntu	Raspberry Pi	Windows
• GPU	• GPU	• CPU (Not working efficiently)
• OpenBLAS (for linear algebra Optimization. Allows Dlib functionality to execute faster. (Optimization significant)	• Both python 2.7 & 3 are pre-installed on Raspbi. • OpenBLAS (significant for Optimization)	• For this first installed CMake. Then CMake supports the installation of Dlib in Pycharm on python 3.

Implementation:

Installing requirements:

- OpenCV-python
- Dlib
- Google Colab (Open Source Software)

For Windows

There is an additional requirement for window 10 i.e.

- Install cmake library to install the Dlib

Step 1:

Import the libraries

```
[1]  pip install dlib

     Requirement already satisfied: dlib in /usr/local/lib/python3.6/dist-packages (19.18.0)
```

```
[4]  pip install opencv-python

     Requirement already satisfied: opencv-python in /usr/local/lib/python3.6/dist-packages (4.1.2.30)
     Requirement already satisfied: numpy>=1.11.3 in /usr/local/lib/python3.6/dist-packages (from opencv-python) (1.
```

```
[6]  import dlib
     import cv2
```

Step 2:

Load & Show the image:

```
#read the image
img= cv2.imread('/content/faceimg.PNG')
```

In Google colab there is issue while compiling cv2.imshow function because this is incompatible for Jupiter notebook. To overcome this issues first import cv.imshow from Jupiter patches as shown below screenshot:

```
from google.colab.patches import cv2_imshow
cv2_imshow(img)
#cv2.imshow(winname='Face', mat=img)
```

First just loading the image using imread() function. Then show the present the image using imshow() function.

Result:

Face Recognition:

```
#Load the detector
detector=dlib.get_frontal_face_detector()
```

Create an object of detector that actually use to retrieve the faces information. Each face is an object that contains the points where image can be found.

```
#Convert image into grayscale
gray=cv2.cvtColor(img, code=cv2.COLOR_BGR2GRAY)

#Use detector to find landmarks
faces=detector(gray)
```

The dlib.get_frontal_face-detector() function works only with gray scale images. For this covert the image into gray scale.

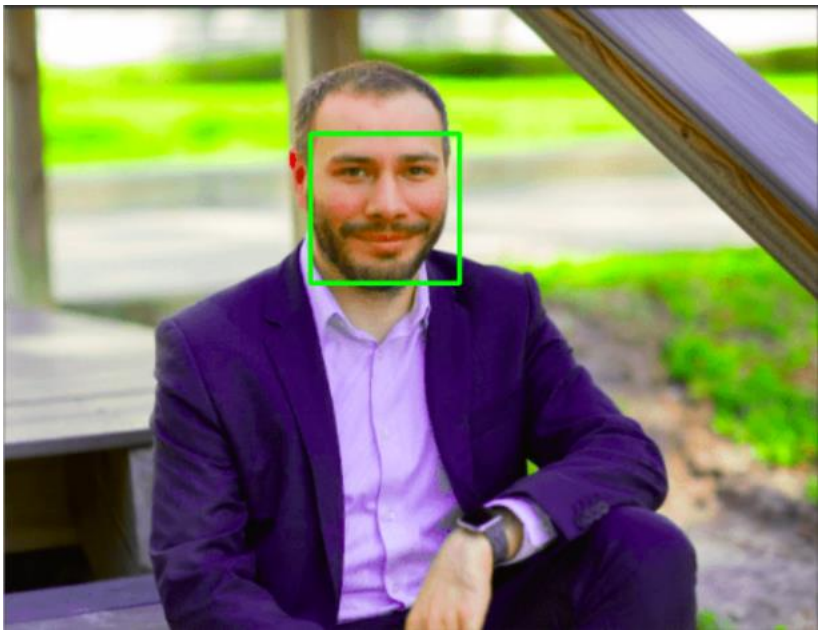
```
for face in faces:
  x1=face.left() #left points
  y1=face.top() # top points
  x2=face.right() #right points
  y2=face.bottom() #bottom points

  #Draw a rectangle
  cv2.rectangle(img=img, pt1=(x1,y1), pt2=(x2, y2), color=(0,255,0), thickness=4)

  #show the image
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

Then apply for loop for detecting the points of face. After this draw a rectangle in which define different parameters such color and thickness of rectangle which has to be drawn around the face. Then just show the rectangle around the face.

Result:



Step 3: Identifying face features:

For identifying the face features first installed the pre-trained model from github using the link mentioned below:

https://github.com/jupyter/notebook/issues/3935.

After loading it into google colab used shape_predictor() function as shown below:

```
#Identifying face features
#Load the predictor
predictor= dlib.shape_predictor("/content/shape_predictor_68_face_landmarks.dat")
```

Our predictor return an object which contains 68 points which is actually HOG+SVM in which 68 are the coordinates or landmark points. For this purpose choose a face observing it and select a point which is exactly the center of the object. In our scenario we choose 27 point landmark for both x and y coordinates that lie exactly b/w the eyes as shown below screenshot:

```
for face in faces:
  x1=face.left() #left points
  y1=face.top() # top points
  x2=face.right() #right points
  y2=face.bottom() #bottom points

  #Look for the landmarks
  landmarks= predictor(image=gray, box=face)
  x=landmarks.part(27).x
  y=landmarks.part(27).y

  #Draw a circle
  cv2.circle(img=img, center=(x,y), radius=5, color=(0, 255,0), thickness=-1)

#show the image
from google.colab.patches import cv2_imshow
cv2_imshow(img)
```

Instead of 1 point just render all the points at once:

```
#Use detector to find landmarks
faces=detector(gray)

for face in faces:
  x1=face.left() #left points
  y1=face.top() # top points
  x2=face.right() #right points
  y2=face.bottom() #bottom points

  #Look for the landmarks
  landmarks= predictor(image=gray, box=face)

  #Loop through all the points
  for n in range(0,68):
    x=landmarks.part(n).x
    y=landmarks.part(n).y

    #draw a circle
    cv2.circle(img=img, center=(x,y), radius=1, color=(0, 255,0), thickness=-1)
```

Result:

The desired features can be obtained by adjusting the range interval to get any feature specified in the code as attach in the screenshot above.

OpenCV AND Dlib are powerful libraries (that can be used for different applications like face detection, Eyes blinking and many others) working with machine learning and computer vision.

Application:

- Used for face detection i.e.
    - Attendance System
    - Person wearing a masks or not during current covid situation.
    - Used in biometric system.
    - Information security
    - Voter Verification
- Used to check Eyes are blinking or not to check whether a person in drowsiness situation to avoid the accidents.