Code wins arguments

Face Recognition with Dlib in Python

July 11, 2020 / Machine Learning

Support me on Patreon

Haven't you subscribe my channel yet?

Follow me on Twitter

Follow @serengil

Dlib is a powerful library having a wide adoption in image processing community similar to OpenCV. Researchers mostly use its face detection and alignment module. Beyond this, dlib offers a strong out-of-the-box face recognition module as well. Even though it is written in c++, it has a python interface as well. In this post, we will mention how to apply face recognition with Dlib in Python.

Person of interest (2011)

Face recognition pipeline

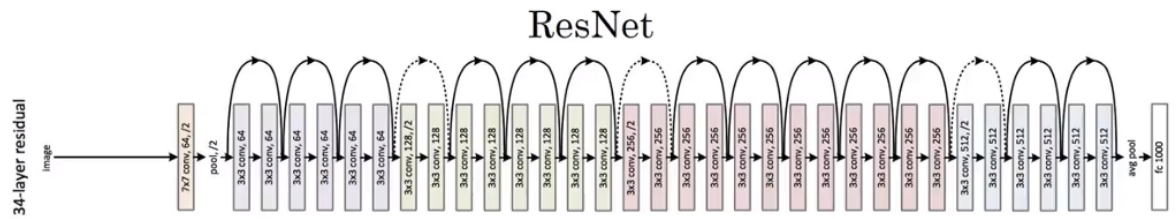
A modern face recognition pipeline consists of 4 common stages: detect, align, represent and verify. Supportively, all of those stages are covered in dlib's implementation.

Vlog

The following video explains how to apply face recognition within dlib. You can either watch the video or follow the blog post.

Model

Dlib is mainly inspired from a ResNet-34 model. Davis E. King modified the regular ResNet structure and dropped some layers and re-build a neural networks consisting of 29 convolution layers. It expexts 150x150x3 sized inputs and represent face images as 128 dimensional vectors.



ResNet-34

He then re-trained the model for various data sets including FaceScrub and VGGFace2. In other words, it learns how to find face representations with 3M samples. Then, he tested the built for labeled faces in the wild (LFW) data set which is accepted as a baseline for face recognition researches. He got 99.38% accuracy. On the other hand, human beings hardly have 97.53% score on same dataset. This means that dlib face recognition model can compete with the other state-of-the-art face recognition models and human beings as well.

Prerequisites

Dlib requires a facial landmark detector and resnet model files. You can manually download the source files and decompress them. Alternatively, the following code block will download and unzip these required files if they doesn't exist in your current directory.

```
def unzip_bz2_file(zipped_file_name):
zipfile = bz2.BZ2File(zipped_file_name)
data = zipfile.read()
newfilepath = output[:-4] #discard .bz2 extension
open(newfilepath, 'wb').write(data)

def download_file(url):
output = url.split("/")[-1]
gdown.download(url, output, quiet=False)

if os.path.isfile('shape_predictor_5_face_landmarks.dat') != True:
print("shape_predictor_5_face_landmarks.dat is going to be downloaded"
url = "http://dlib.net/files/shape_predictor_5_face_landmarks.dat.bz2"
download_file(url)
unzip_bz2_file(output)

if os.path.isfile('dlib_face_recognition_resnet_model_v1.dat') != True
print("dlib_face_recognition_resnet_model_v1.dat is going to be downlo
url = "http://dlib.net/files/dlib_face_recognition_resnet_model_v1.dat
download_file(url)
unzip_bz2_file(output)
```

Loading pre-trained models

We've downloaded the prerequisite files in the previous block. Now, we need to build pre-trained models.

```
import dlib
detector = dlib.get_frontal_face_detector()
sp = dlib.shape_predictor("shape_predictor_5_face_landmarks.dat")
facerec = dlib.face_recognition_model_v1("dlib_face_recognition_resnet_
```

Face detection and alignment

The following code block handles loading, detection and alignment stages. Aligned faces will be in shape of (150, 150, 3).

```
#load images
img1 = dlib.load_rgb_image("img1.jpg")
img2 = dlib.load_rgb_image("img2.jpg")

#detection
img1_detection = detector(img1, 1)
img2_detection = detector(img2, 1)

img1_shape = sp(img1, img1_detection[0])
img2_shape = sp(img2, img2_detection[0])

#alignment
img1_aligned = dlib.get_face_chip(img1, img1_shape)
img2_aligned = dlib.get_face_chip(img2, img2_shape)
```

On the other hand, we don't have to apply face detection within dlib because it is not the best solution in the open source solutions.

Face detection can be done with many solutions such as OpenCV, Dlib or MTCNN. OpenCV offers haar cascade, single shot multibox detector (SSD). Dlib offers Histogram of Oriented Gradients (HOG) and Max-Margin Object Detection (MMOD). Finally, MTCNN is a popular solution in the open source community as well. Herein, SSD, MMOD and MTCNN are modern deep learning based approaches whereas haar cascade and HoG are legacy methods. Besides, SSD is the fastest one. You can monitor the detection performance of those methods in the following video.

Here, you can watch how to use different face detectors in Python.

You can find out the math behind alignment more on the following video:

Besides, face detectors detect faces in a rectangle area. So, detected faces come with some noise such as background color. We can find 68 different landmarks of a face with dlib. In this way, we can get rid of any noise of a facial image.

Here, retinaface is the cutting-edge face detection technology. It can even detect faces in the crowd and it finds facial landmarks including eye coordinates. That's why, its alignment score is very high.

More sensitive way

Face detection does not have to be applied for rectangle areas. We can do it more sensitive with the facial landmark detection with Dlib. It can find 68 facial landmark points on the face including jaw and chin, eyes and eyebrows, inner and outer area of lips and nose.

Here, you can find a deeply explained tutorial about facial landmarks detection with dlib.

Represention

We will feed the aligned faces to the ResNet model and it represent faces 128 dimensional vector.

```
img1_representation = facerec.compute_face_descriptor(img1_aligned)
img2_representation = facerec.compute_face_descriptor(img2_aligned)
```

Even though dlib finds representations in dlib.vector type, we can convert it to numpy easily to find the distance easily in the following step.

```
img1_representation = np.array(img1_representation)
img2_representation = np.array(img2_representation)
```

Euclidean distance

Davis King proposes to use Euclidean distance to verify faces because he found the tuned threshold.

```
def findEuclideanDistance(source_representation, test_representation):
euclidean_distance = source_representation - test_representation
euclidean_distance = np.sum(np.multiply(euclidean_distance, euclidean_d
euclidean_distance = np.sqrt(euclidean_distance)
return euclidean_distance
```

Verification

We already have the representations of pairs. We also know how to find the distance between these vectors. King shared the tuned threshold as well.

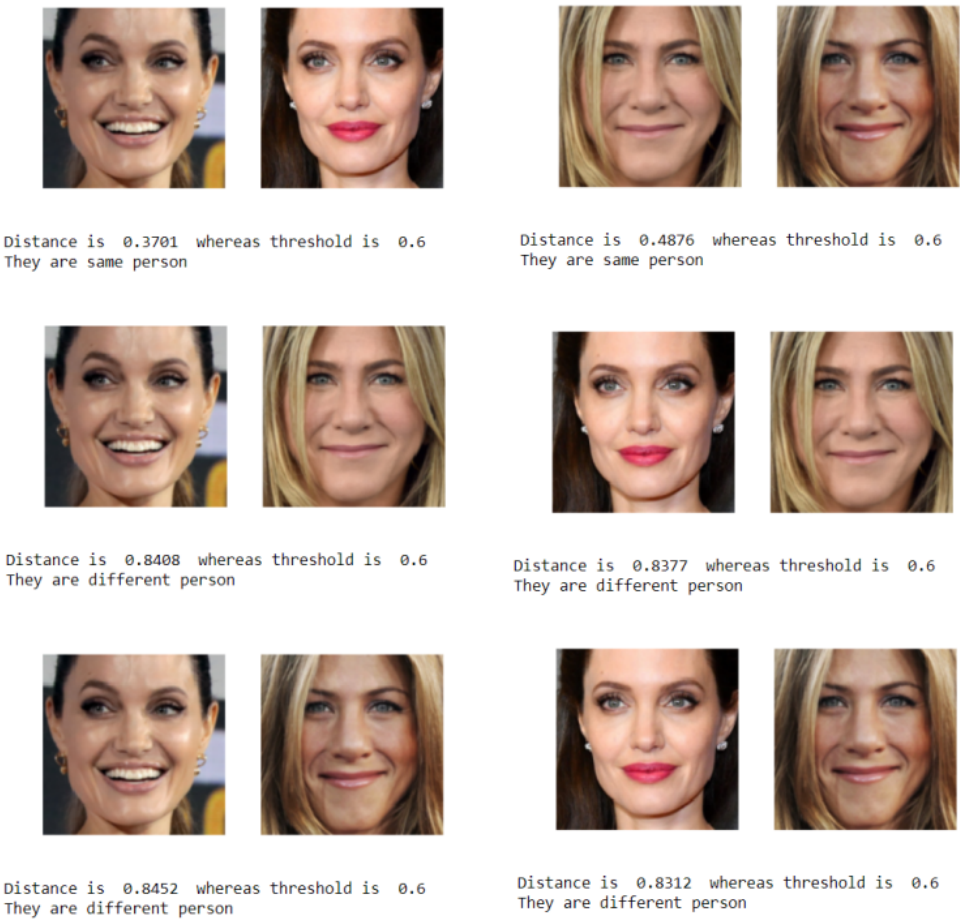
```
distance = findEuclideanDistance(img1_representation, img2_representati
threshold = 0.6 #distance threshold declared in dlib docs for 99.38% co

if distance &amp;amp;amp;amp;amp;amp;lt; threshold: print("they are
else: print("they are different")
```

Tests

I've tested the face recognition module of dlib for several pairs. The following code block will plot pairs side by side. I've used the some unit test images of deepface.

```
def plotPairs(img1, img2):
fig = plt.figure()
ax1 = fig.add_subplot(1,2,1)
plt.imshow(img1);plt.axis('off')
ax1 = fig.add_subplot(1,2,2)
plt.imshow(img2); plt.axis('off')
plt.show()
```

Results seem very satisfactory.

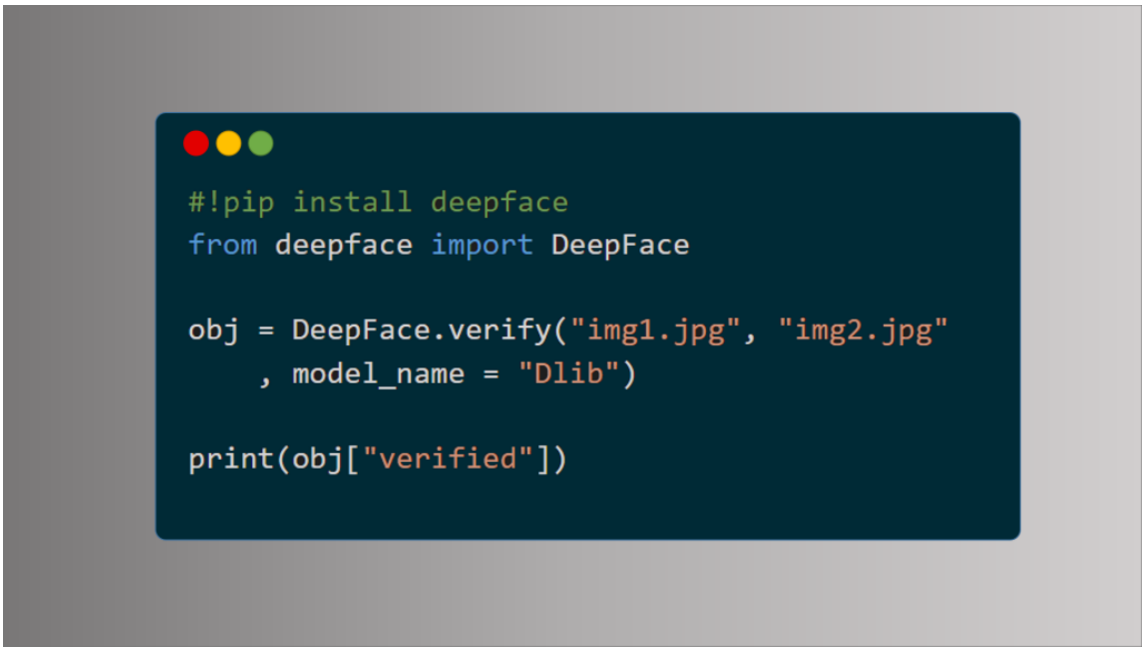


Tests

Out-of-the-box pipeline

Dlib is a spectacular library. However, it expects you to apply all common stages of a face recognition pipeline: detect, align, represent and verify. This might discourage you. Herein, DeepFace library for python handles all of those stages in the background and you can run it with a few lines of code.

It is a hybrid face recognition framework wrapping the **state-of-the-art** face recognition models including University of Oxford's VGG-Face, Google FaceNet, Carnegie Mellon University's OpenFace, Facebook DeepFace, The Chinese University of Hong Kong's DeepID and Dlib ResNet model.



Dlib ResNet model in deepface package

Here, you can find a video covering how to run deepface.

This comes with a real-time implementation as well.

Large scale face recognition

Besides, you can apply large scale face recognition.

Notice that face recognition has O(n) time complexity and this might be problematic for millions or billions level data. Herein, approximate nearest neighbor (a-nn) algorithm reduces time complexity dramatically. Spotify Annoy, Facebook Faiss and NMSLIB are amazing a-nn libraries. Besides, Elasticsearch wraps an NMSLIB and it comes with highly scalability. You should run deepface within those a-nn libraries if you have really large scale data base.

On the other hand, a-nn algorithm does not guarantee to find the closest one always. We can still apply k-nn algorithm here. Map reduce technology of big data systems might satisfy the both speed and confidence here. mongoDB, Cassandra and Hadoop are the most popular solutions for no-sql databases. Besides, if you have a powerful database such as Oracle Exadata, then RDBMS and regular sql might satisfy your concerns as well.

Tech Stack Recommendations

Face recognition is mainly based on representing facial images as vectors. Herein, storing the vector representations is a key factor for building robust facial recognition systems. I summarize the tech stack recommendations in the following video.

Conclusion

So, we've mentioned how to use out-of-the-box face recognition module of dlib library. It seems that dlib comes with a challenging face recognition service. It also covers all common stages of a modern face recognition pipeline. Just importing dlib is enough to apply face verification.

Finally, I pushed the source code of this study to GitHub. You can support this work by starring⭐ the repo.

#dlib, #face recognition

« *Previous* / *Next* »

---

3 Comments

Justin

June 14, 2021 at 7:28 pm

What is the difference between Deepface and “face_recognition” of Adam Geitgey.

When i set model and detection_backend from deepface to “dlib”, it should actually just work like the “face_recognition” framework of Adam.
The link: https://github.com/ageitgey/face_recognition

Your framework Deepface and the other framework, both are using the same dlib model and the dlib face detector. But why do i get different matching scores?

Thank you

Reply

Sefik Serengil
June 14, 2021 at 7:30 pm

I do not know what Adam did in the background.

Reply

Pingback: DeepFace - Most Popular Deep Face Recognition in 2021 (Guide) | viso.ai

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

ex: jane doe

Email *

ex: janedoe@gmail.com

Website

ex: http://janedoe.wordpress.com

Notify me of follow-up comments by email.

Notify me of new posts by email.

Submit

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Please cite this post if it helps your research. Here is an example of BibTex entry:

```
@misc{sefiks14405,
  author ={Serengil, Sefik Ilkin},
  title = { Face Recognition with Dlib in Python },
  howpublished = {
https://sefiks.com/2020/07/11/face-recognition-
with-dlib-in-python/ },
  year = { 2020 },
  note = "[Online; accessed 2023-01-13]"
}
```