Code wins arguments

Menu

Face Recognition with DeepID in Keras

June 16, 2020 / Machine Learning

Support me on Patreon

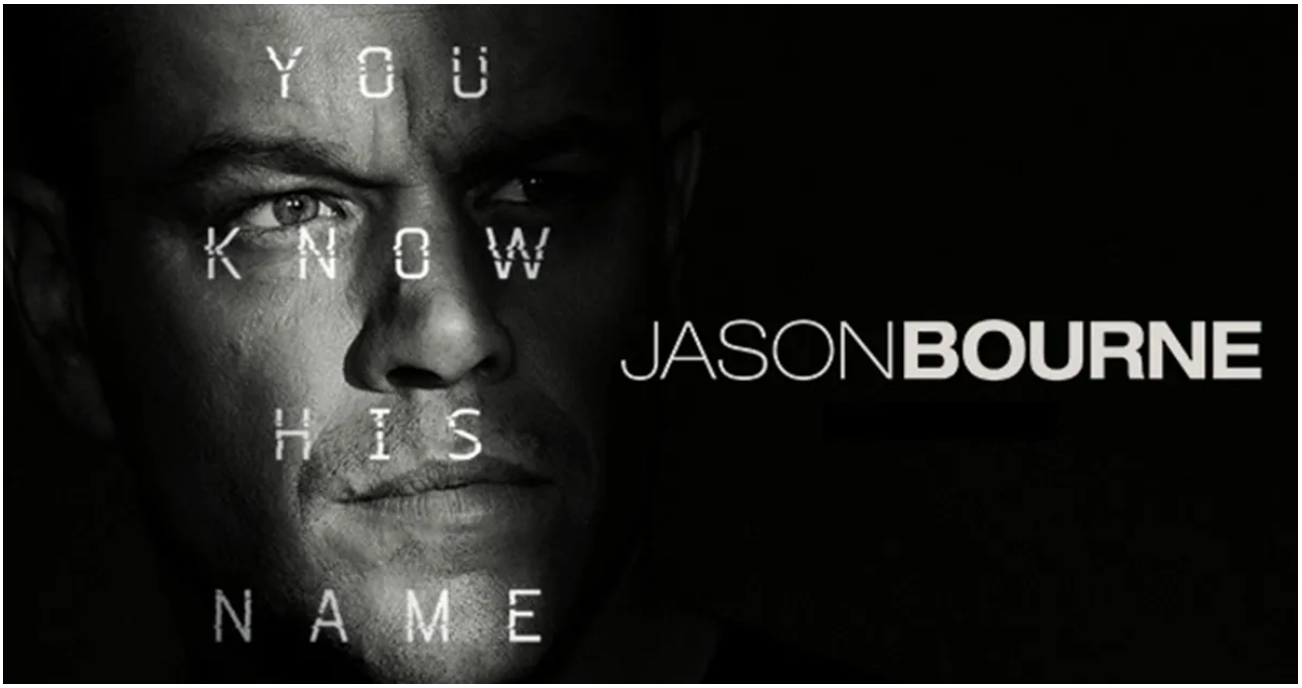
BECOME A PATRON

Haven't you subscribe my channel yet?

Follow me on Twitter

Follow @serengil

Face recognition researches are emerged from the tech giants such as Facebook and Google to the top universities in the world such as Oxford University and Carnegie Mellon University. Notice that US based models are built by commercial companies whereas UK based models are built by universities. Herein, China involved in the face recognition competition with its prestigious academic institution as well. Researchers of the Chinese University of Hong Kong announced two different version of DeepID model for face recognition tasks.



Jason Bourne (2016)

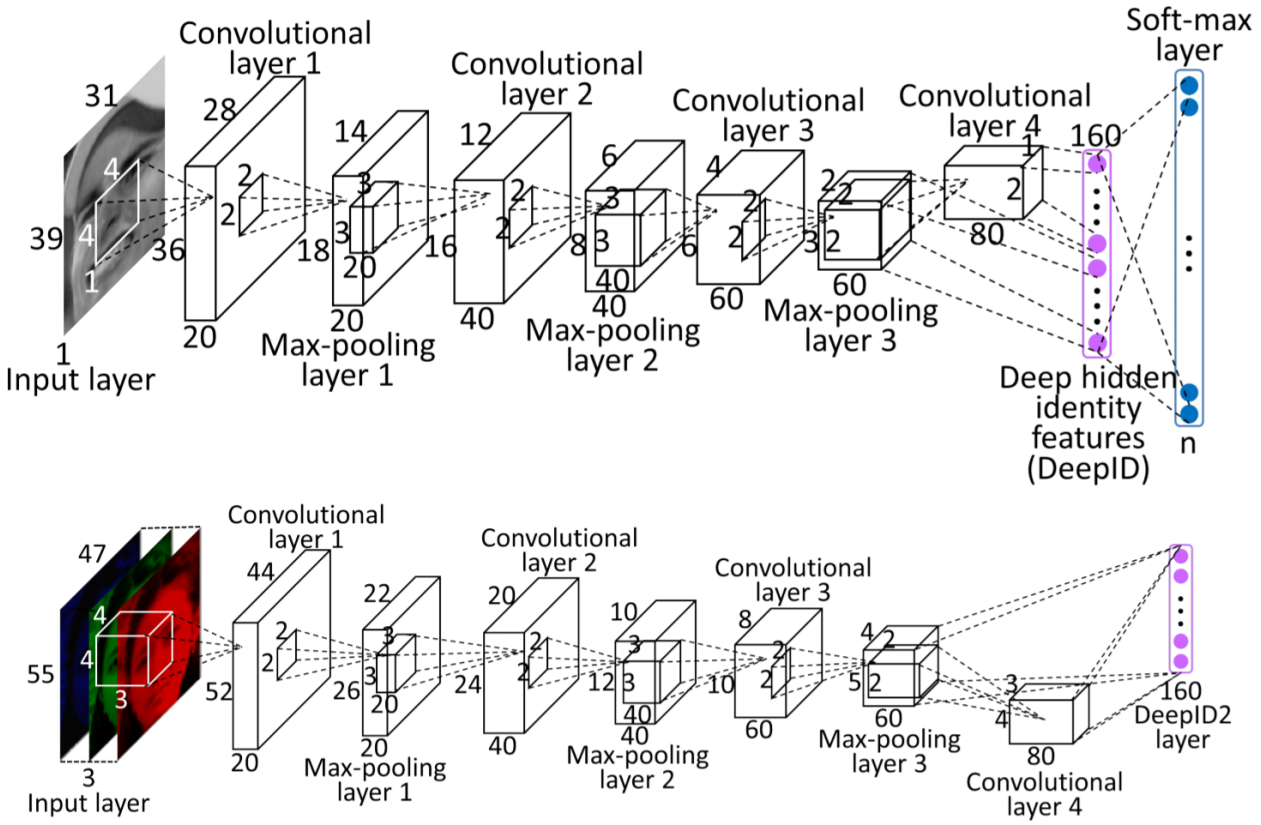
Pipeline

You should remember the common stages of a modern face recognition pipeline and know how face recognition works before reading this post.

Model structure

The both 1st and 2nd generation of DeepID models are almost same as seen. The 1st generation expect 39×31 sized 1 channel input whereas 2nd generation expects 55×47 sized 3 channel (RGB) input images. The 2nd generation is named DeepID2 as well. In this post, we will mention DeepID2 model.

There are 4 convolution layers and one fully connected layer in DeepID models. Researchers trained the model as a regular classification task to classify n identities initially. Then, they removed the final classification softmax layer when training is over and they use an early fully connected layer to represent inputs as 160 dimensional vectors. In this way, the model can represent faces it haven't seen before.



Model structures for DeepID and DeepID2

As a state-of-the-art design 3rd convolution layer is connected to the both 4th convolution layer and fully connected layer whereas 4th convolution layer is connected to fully connected layer as well. Fully connected layer adds the receiving signal from 3rd and 4th convolution layers in DeepID2 whereas 1st generation DeepID appends receiving signals from those layers.

We can build DeepID model in Keras as illutrated below.

```
from keras.models import Model
from keras.layers import Conv2D, Activation, Input, Add
from keras.layers.core import Dense, Flatten, Dropout
from keras.layers.pooling import MaxPooling2D
myInput = Input(shape=(55, 47, 3))

x = Conv2D(20, (4, 4), name='Conv1', activation='relu', input_shape=(5
x = MaxPooling2D(pool_size=2, strides=2, name='Pool1')(x)
x = Dropout(rate=1, name='D1')(x)

x = Conv2D(40, (3, 3), name='Conv2', activation='relu')(x)
x = MaxPooling2D(pool_size=2, strides=2, name='Pool2')(x)
x = Dropout(rate=1, name='D2')(x)

x = Conv2D(60, (3, 3), name='Conv3', activation='relu')(x)
x = MaxPooling2D(pool_size=2, strides=2, name='Pool3')(x)
x = Dropout(rate=1, name='D3')(x)

x1 = Flatten()(x)
fc11 = Dense(160, name = 'fc11')(x1)

x2 = Conv2D(80, (2, 2), name='Conv4', activation='relu')(x)
```

```
x2 = Flatten()(x2)
fc12 = Dense(160, name = 'fc12')(x2)

y = Add()([fc11, fc12])
y = Activation('relu', name = 'deepid')(y)

model = Model(inputs=[myInput], outputs=y)
```

Pre-trained weights

Even though DeepID is designed and developed by an academic institution, researchers just share the model structure and they don't prefer to share the pre-trained weights of the model. Luckily, DeepID model is retrained by open source community. Roy Ruan pushed the pre-trained weights for TensorFlow in his GitHub repo.

I converted the tensorflow weights into Keras format. Here, you can find keras weights of DeepID2 model. If you wonder how I converted TensorFlow weights to Keras format, then this notebook will inform you.

```
#Ref: https://drive.google.com/file/d/1uRLtBCTQQAvHJ_KVrdbRJiCKxU8m5q2J
model.load_weights("deepid_keras_weights.h5")
```

This is a very minimal model. Weight file is 1.53 MB and there are 400K params in the built model. Weight file is more than 500 MB for both VGG-Face and DeepFace, 90 MB for FaceNet, and 15 MB for OpenFace models. This comes with a high speed in both building and prediction steps.

Pre-processing

Remember that a modern face recognition pipeline consists of 4 common stages: detect, align, represent and verify. The both detection and alignment stages are pre-processing steps. Besides, researches show that just alignment increases the model accuracy about 1%. Luckily, deepface package hadles those stages with a few lines of code.

```
#!pip install deepface
from deepface.commons import functions

img1_path = "img1.jpg"; img2_path = "img2.jpg"

img1 = functions.detectFace(img1_path, (47, 55))
img2 =  functions.detectFace(img2_path, (47, 55))
```

In this way, just face area of source images will be detected and they will be aligned horizontally. Besides, those images will be resized to the expected size of DeepID input layer.

On the other hand, 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.

In addition, MediaPipe can find 468 landmarks. Please see its real time implementation in the following video. Recommended tutorials: Deep Face Detection with MediaPipe, Zoom Style Virtual Background Setup with MediaPipe.

Representation

DeepID model is responsible for representing face images as vectors. We've already built the model. Feeding processed images to predict function will extract representions.

```
img1_representation = model.predict(img1)[0,:]
img2_representation = model.predict(img2)[0,:]
```

Now, we have 160 dimensional vector representation for two different face images.

Verification

We expect that face representations of same person should have a high similarity and low distance whereas representations of different people should have a low similarity and high distance. Herein, we can apply cosine or euclidean distance to verify a pair.

I will use the out-of-the-box functions of deepface package to find distance metrics as well.

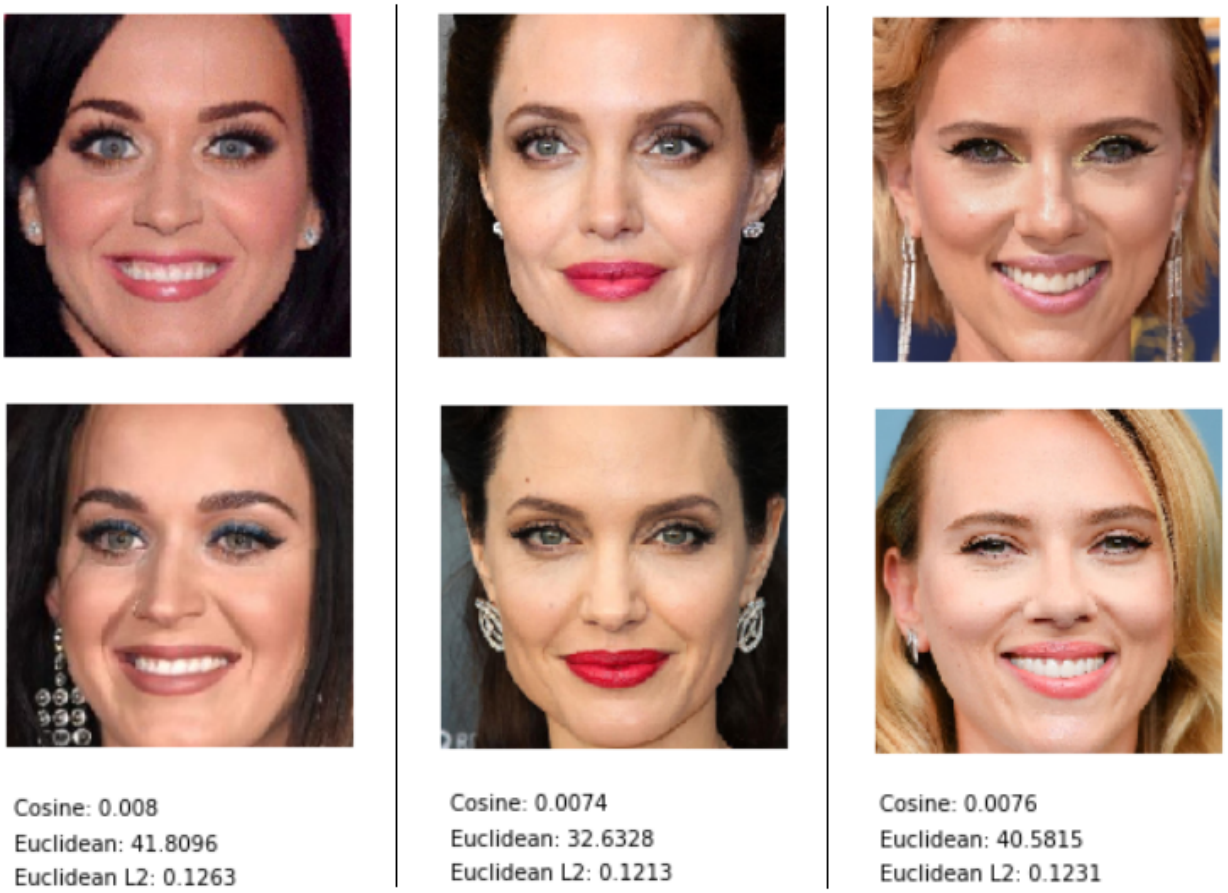
```
#!pip install deepface
from deepface.commons import distance as dst

cosine_distance = dst.findCosineDistance(img1_representation, img2_repr
euclidean_distance = dst.findEuclideanDistance(img1_representation, img
euclidean_l2_distance = dst.findEuclideanDistance(dst.l2_normalize(img1
```

My experiments show that the following threshold values perform well to verify image pairs.

```
if metric == 'cosine':
threshold = 0.015
elif metric == 'euclidean':
threshold = 45
elif metric == 'euclidean_l2':
threshold = 0.17
```

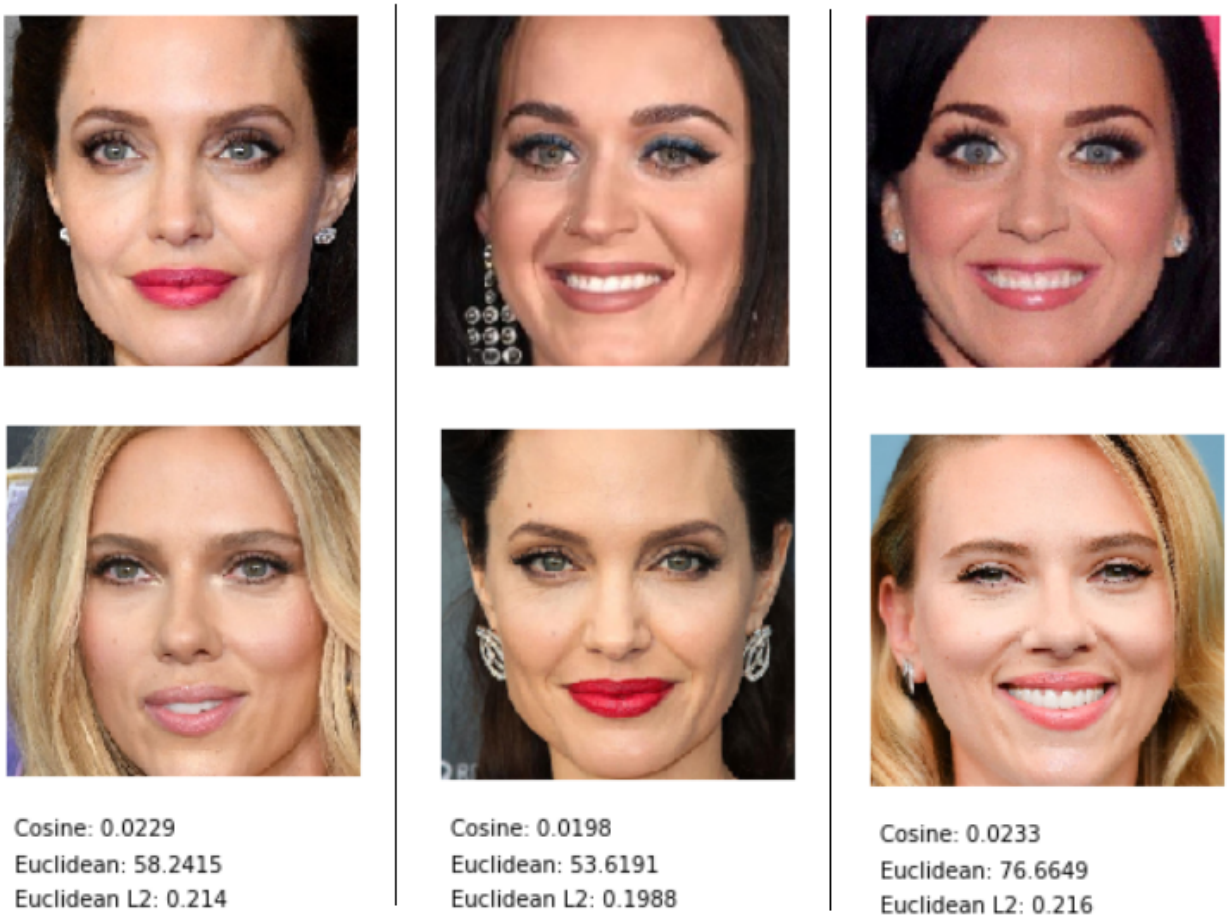
Tests

I've tested DeepID model for the unit test items of deepface package. We can verify all face pairs based on threshold values I've mentioned before.



Positive pairs

Besides, all negative pairs have a large distance values than tuned threshold.



Negative pairs

Results seem very satisfactory and they convinced me about robustness about this model.

DeepID in Python

In this post, we've mentioned the technical depth of DeepID model. DeepFace package for python offers you to use DeepID model in face recognition tasks with a few lines of code as well.

DeepID in deepface package

Here, you can watch how to use DeepID model in your face recognition tasks.

Moreover, you can run face recognition with DeepID model in real time as well.

Large scale face recognition

Finally, you can apply face recognition on a large scale data set.

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.

Ensemble method

We've mentioned just a single face recognition model. On the other hand, there are several state-of-the-art models: VGG-Face, Google FaceNet, OpenFace, Facebook DeepFace and DeepID. Even though all of those models perform well, there is no absolute better model. Still, we can apply an ensemble method to build a grandmaster model. In this approach, we will feed the predictions of those models to a boosting model. Accuracy metrics including precision, recall and f1 score increase dramatically in ensemble method whereas running time lasts longer.

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 DeepID within Keras and Python. Even though it is very minimal face recognition model, its results seem very satisfactory. Being minimal comes with a fast speed for both model building and prediction stages. That's why, it is a very strong option to adapt in real time studies.

I pushed the source code of this study to GitHub as a notebook. Besides, I shared the pre-trained weights in Google Drive as well.

#face recognition

« Previous / Next »

---

3 Comments

Alan

January 16, 2021 at 7:00 pm

Hi Sefik.
There is no detectFace function in deepface.commons
from deepface.commons import functions, distance as dst
img1 = detectFace(img1_path, (47, 55))
according to
https://github.com/serengil/deepface/blob/master/deepface/DeepFace.py
So I have tested
from deepface.DeepFace import detectFace
but error appears
ValueError: ('Valid backends are ', ['opencv', 'ssd', 'dlib', 'mtcnn'], ' but you passed ', (47, 55))
Could you help me please?
Trying
def detect_face(imagem, detector):
faces = detector.detectMultiScale(imagem, 1.13, 5)
for (x,y,w,h) in faces:
detected_face = imagem[int(y):int(y+h), int(x):int(x+w)]
detected_face = cv2.resize(detected_face, (47, 55))
img_pixels = image.img_to_array(detected_face)
print(img_pixels.shape)
img_pixels = np.expand_dims(img_pixels, axis = 0)
img_pixels /= 255 #normalize input in [0, 1]
print(img_pixels.shape)
return detected_face, img_pixels
result is bad
Thank you

Reply

Sefik Serengil
January 16, 2021 at 8:29 pm

Please modify detectFace to preprocess_face

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{sefiks14346,
author ={Serengil, Sefik Ilkin},

```
  title = { Face Recognition with DeepID in Keras },
  howpublished = {
https://sefiks.com/2020/06/16/face-recognition-
with-deepid-in-keras/ },
  year = { 2020 },
  note = "[Online; accessed 2023-01-13]"
}
```