



Sefik Ilkin Serengil

Acumino

Upskilling Robots

× 广告

OPEN

Code wins arguments

Menu ▾

Face Recognition with OpenFace in Keras

July 21, 2019 / Machine Learning

Support me on Patreon

 **BECOME A PATRON**

Haven't you subscribe my channel yet?

Follow me on Twitter


Follow @serengil

广告 X

Microsoft Azure

代码。
实验。
构建。

使用 Azure 不断学习
新技能和进行实验



免费试用 Azure >

[OpenFace](#) is a **lightweight** and **minimalist** model for face recognition. Similar to [Facenet](#), its [license](#) is free and allowing commercial purposes. On the other hand, [VGG-Face](#) is restricted for commercial use. In this post, we will mention how to adapt OpenFace for your face recognition tasks in Python with Keras.




Jason Bourne

Objective

Face recognition is a combination of [CNN](#), [Autoencoders](#) and [Transfer Learning](#) studies. I strongly recommend you to read [How Face Recognition Works](#) post to understand what a face recognition pipeline exactly is.


Online Course

The course cover features a blue background with a network of white nodes and connecting lines, resembling a neural network or a complex data structure.

Neural Networks for Machine Learning From Scratch

Develop your own deep learning framework from zero in Python and gain hands on experience from scratch

Sefik Ilkin Serengil

The Udemy logo, consisting of a red stylized 'U' followed by the word 'Udemy' in a sans-serif font.

Impediments

OpenFace model including model structure and weights is public but it is built with Lua Torch. Currently, PyTorch does not support to load Torch7 (.t7) models anymore after its 1.0 version released. Even though previous PyTorch versions (e.g. 0.4.1) continues to [support](#), published OpenFace Torch7 model cannot be converted because it is a really old Torch [model](#). Some researchers propose to load this model with a new Lua Torch version and save it back. However, I do not a Lua environment.

After this painful process, I found this repository – [Keras-OpenFace](#). OpenFace weights are converted to Keras already here. The whole model including structure and weights saved a standalone file here. Unfortunately, this causes trouble if you would load the model with different Keras version. Weights and structure should be separated to be compatible with all Keras environments. Weights can be shared as binary h5 extension but structure should be constructed by hand. Loading the model structure with a json file might cause “bad marshal data” exceptions similar to loading standalone model.

Luckily, converted OpenFace weights in [csv files](#) are already stored in this repository. **Your friendly neighbour blogger** will convert OpenFace model including structure and weights separately to Keras. This [notebook](#) helped me to convert weights.

CNN Model

OpenFace model expects (96×96) RGB images as input. Its has a 128 dimensional output. The model is built on **Inception Resnet V1**. I already build the CNN model for Keras. You can find the built model [here](#) as json file. Even though the model seems complex, number of parameters are much less than VGG-Face.

```
1 | import tensorflow as tf
2 | model = model_from_json(open("openface_structure.json", "r").read(), cu
```

You might have some troubles when you loading the model from json file because Keras expects you to use same environment with the publisher. In this case, you can build your model manually. [Here](#), you can find the manual model building.

Next, you should load the [pre-trained weights](#). It is size of 14 MB. That's why, I stored it in my Google Drive.

```
1 | #Pre-trained OpenFace weights: https://bit.ly/2Y34cB8
2 | model.load_weights("openface_weights.h5")
```

Vector representation

Similar to [VGG-Face](#) and [Facenet](#), we will apply one shot learning. CNN model finds the vector representations of faces and expresses faces as embeddings.

```
1 | p1 = 'openface-samples/img-1.jpg'
2 | p2 = 'openface-samples/img-2.jpg'
3 |
4 | img1_representation = model.predict(preprocess_image(p1))[0,:]
5 | img2_representation = model.predict(preprocess_image(p2))[0,:]
```

Different photos of same person should have a low distance whereas different faces should have a high distance. Euclidean distance or [cosine](#) can be the metric here. I mostly do not prefer to use l2 normalization in calculations of euclidean distance.

```
1 | def findCosineDistance(source_representation, test_representation):
2 |     a = np.matmul(np.transpose(source_representation), test_representation)
3 |     b = np.sum(np.multiply(source_representation, source_representation))
4 |     c = np.sum(np.multiply(test_representation, test_representation))
5 |     return 1 - (a / (np.sqrt(b) * np.sqrt(c)))
6 |
7 | def l2_normalize(x, axis=-1, epsilon=1e-10):
8 |     output = x / np.sqrt(np.maximum(np.sum(np.square(x), axis=axis, keepdi
9 |     return output
10 |
11 | def findEuclideanDistance(source_representation, test_representation):
12 |     euclidean_distance = source_representation - test_representation
13 |     euclidean_distance = np.sum(np.multiply(euclidean_distance, euclidean_
14 |     euclidean_distance = np.sqrt(euclidean_distance)
15 |     #euclidean_distance = l2_normalize(euclidean_distance )
16 |     return euclidean_distance
17 |
18 | cosine = findCosineDistance(img1_representation, img2_representation)
19 | euclidean = findEuclideanDistance(img1_representation, img2_representa
```

My own best practices that I've gathered from my personal experiences and experiments on OpenFace show that distance threshold for the model should be 0.02 for cosine and 0.20 for euclidean distance.

If you wonder how to determine the threshold value for this face recognition model, then [this blog post](#) explains it deeply.

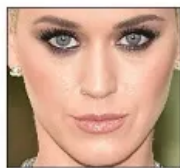
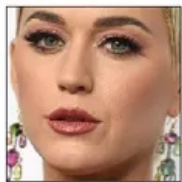
```

1  if cosine <= 0.02:
2  print("these are same")
3  else:
4  print("these are different")
5
6  """if euclidean <= 0.20:
7  print("these are same")
8  else:
9  print("these are different")"""

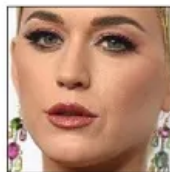
```

Testings

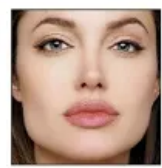
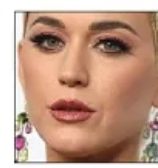
I feed some photos of Katy Perry, Miley Cyrus and Angelina Jolie to OpenFace model and check the identity of some combinations. Results are satisfactory.



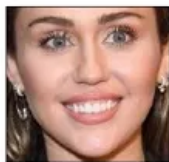
Cosine distance: 0.019822120666503906
Euclidean distance: 0.19910881
these are same



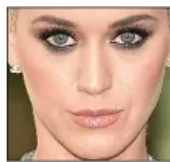
Cosine distance: 0.0255013108253479
Euclidean distance: 0.22583763
these are different



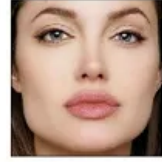
Cosine distance: 0.0390501022388672
Euclidean distance: 0.27946445
these are different



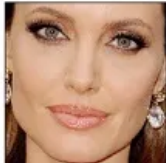
Cosine distance: 0.014958083629608154
Euclidean distance: 0.17296319
these are same



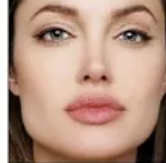
Cosine distance: 0.0324356559387207
Euclidean distance: 0.2546986
these are different



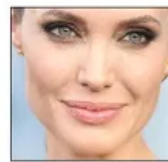
Cosine distance: 0.023996949195861816
Euclidean distance: 0.21907528
these are different



Cosine distance: 0.011376440525054932
Euclidean distance: 0.1508414
these are same



Cosine distance: 0.02035599946975708
Euclidean distance: 0.2017724
these are different




Cosine distance: 0.02776157855987549
Euclidean distance: 0.23563409
these are different

Testings for OpenFace

Real time

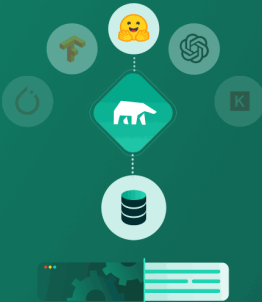
We can run OpenFace implementation for real time as well. OpenCV's haarcascade module handles detecting faces and we feed detected face to OpenFace model. You can find the source code of the following video [here](#). This solution is much more faster than VGG-Face and Facenet.

Thresholds are tuned for real time implementation. It is now 0.45 for cosine and 0.95 for euclidean without l2 normalization.



Get AI into your apps. Simply.

[Star on GitHub](#)



Build AI apps with "Smart SQL"

Machine Learning has never been so simple. Use your SQL skills to build AI-powered apps

MindsDB

[Learn More >](#)

广告 X

Face alignment

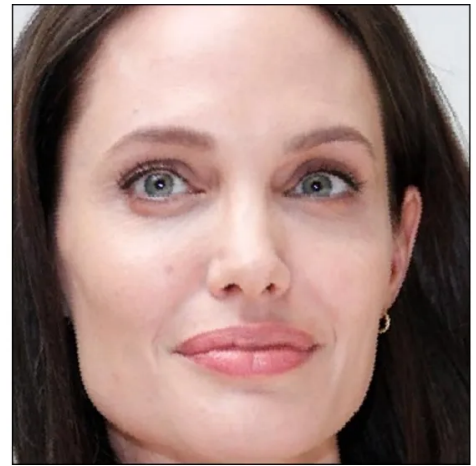
Modern face recognition pipelines consist of 4 stages: detect, align, represent and classify / verify. We've ignored the face detection and face alignment steps not to make this post so complex. However, it is really important for face recognition tasks.

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.

For instance, Google declared that face alignment increases its face recognition model [FaceNet](#) from 98.87% to 99.63%. This is almost 1% accuracy improvement which means a lot for engineering studies. [Here](#), you can find a detailed tutorial for face alignment in Python within OpenCV.



Face alignment

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](#).

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.

Conclusion

OpenFace is a lightweight face recognition model. It is not the best but it is a strong alternative to stronger ones such as [VGG-Face](#) or [Facenet](#). It has 3.7M trainable parameters. This was 145M in VGG-Face and 22.7M in Facenet. Besides, weights of OpenFace is 14MB. Notice that VGG-Face weights was 566 MB and Facenet weights was 90 MB. This comes with the speed. That's why, adoption of OpenFace is very high. You can deploy it even in a mobile device.

To be honest, this model is not perfect. It can fail some obvious testings. You should adopt VGG-Face if you do not have a tolerance for errors. On the other hand, speed of your implementation is your first benchmark, OpenFace would be a pretty solution.

Besides, OpenFace is developed by researchers of Carnegie Mellon University. VGG-Face was developed by Oxford University researchers. Notice that both studies are based on United Kingdom universities. **We owe much these UK based researches!**


Source code

I pushed source code of this blog post to GitHub as [notebook](#). Besides, model including [structure](#) and [pre-trained weights](#) in Keras format is shared as well. If model building step seems too complex, you can also load it with a single line code by referencing [model in JSON format](#). You can support this work just by starring [the repository](#).

Python Library

Herein, [deepface](#) is a lightweight face recognition framework for Python. It currently supports the most common face recognition models including [VGG-Face](#), [Facenet](#) and [OpenFace](#), [DeepID](#).

It handles model building, loading pre-trained weights, finding vector embedding of faces and applying similarity metrics to recognize faces in the background. You can verify faces with a just few lines of code. It is available on [PyPI](#). You should run the command “**pip install deepface**” to have it. Its code is also open-sourced in [GitHub](#). [GitHub repo](#) has a detailed documentation for developers. BTW, you can support this project by starring the repo.



```
#!/pip install deepface
from deepface import DeepFace

img1 = "img1.jpg"; img2 = "img2.jpg"
result = DeepFace.verify(img1, img2)

if result[0] == True:
    print(img1, " and ",img2," are same person")
else:
    print(img1, " and ",img2," are different person")
```

deepface

Here, you can watch the how to video for deepface.

Besides, you can run deepface in real time with your webcam as well.

Large scale face recognition

Large scale face recognition requires to apply face verification several times. However, we can store the representations of ones in our database. In this way, we just need to find the representation of target image. Finding distances between representations can be handled very fast. So, we can find an identity in a large scale data set in just seconds.

Deepface offers an out-of-the-box function to handle large scale face recognition as well.

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.

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.

The Best Single Model

There are a few state-of-the-art face recognition models: [VGG-Face](#), [FaceNet](#), [OpenFace](#) and [DeepFace](#). Some are designed by tech giant companies such as Google and Facebook whereas some are designed by the top universities in the world such as University of Oxford and Carnegie Mellon University. We will have a discussion about the best single face recognition model in this video.

Like this blog? Support me on Patreon

 **BECOME A PATRON**

[#face recognition](#), [#facenet](#), [#one shot learning](#), [#openface](#), [#python](#), [#vgg-face](#)

« Previous / Next »

22 Comments

Alan

September 3, 2019 at 8:12 pm

Hello. How do I use more of one image per person? Thank you.

[^ Reply](#)

 **Sefik Serengil**

September 3, 2019 at 8:37 pm

This is the one shot learning but you can manipulate the system as described below. Feed a multiple images of a person. For example, alex_1.jpg, alex_2.jpg and alex_3.jpg. Suppose that you are going to find who is x.jpg. Compare x and Alex. To do this, you need to find the distance x -alex_1, x -alex_2 and x -alex-3. Find the average value of all of these comparisons. If average value is less than the threshold, then x is Alex.

I hope this explains well

[^ Reply](#)

 **Alan**

September 3, 2019 at 9:08 pm

Yeah, great explanation. Thank you (again) for sharing knowledge and doing education.

[^ Reply](#)

Talha Anwar

September 27, 2019 at 6:21 pm

thanks for such a great tutorial. I have tried it, but its not labelling me correct. Either it is labelling me unknown or the other guys. I have also increased my picture, as you suggested in above pics. and remove the guy which the algo is predicting me. but now it still predicting me as unknown or the other guy left in data. How can i improve it?

[^ Reply](#)

 **Sefik Serengil**

September 27, 2019 at 6:23 pm

your reference picture is cropped?

[^ Reply](#)

 **Talha Anwar**

December 6, 2019 at 4:38 pm

yes, they are. i have one more question, at which phase, this model is being trained for our dataset? i mean in transfer learning we trained last layers according to our data. i

want to know, how this model know about our data

[^ Reply](#)

 **Sefik Serengil**

December 6, 2019 at 4:56 pm

It is tuned for find face embeddings. We would not train it anymore.

[^ Reply](#)

Michael

November 3, 2019 at 11:41 pm

Thank you for the tutorial! I tried out your real time code from GitHub and I'm getting strange results. All the source images ("database" folder) are cropped, the cv2 face detection works correctly. For instance when I try the net on Angelina I'm getting the best match result for Brad. With a photo of my own the detections are inconsistent – I'm being labeled as myself and then as someone else. Maybe you could share the dataset you have used for the YT video (testing dataset)? Otherwise, what network should I use for more usable results and also for the constrained devices?

[^ Reply](#)

 **Sefik Serengil**

November 4, 2019 at 4:12 am

This is interesting. Could you crop testing images in database folder with opencv face detection? I could find these testing images randomly.

[^ Reply](#)

 **Michael**

November 5, 2019 at 7:18 pm

Actually the opencv detection algorithm crops the images differently than they are cropped in your samples. But it changes nothing. It is noticeable that OpenFace has a very low margin of error – most wrong/correct classifications are just near the threshold.

What network could I chose for more precise recognition?

[^ Reply](#)

 **Sefik Serengil**

November 5, 2019 at 7:20 pm

You should re-switch to VGG-Face

^ Reply

Yasser

March 26, 2020 at 2:34 pm

Hi ...

what is the difference between openface v1 and openface v2?

^ Reply

Yasser

March 26, 2020 at 3:20 pm

Another Question:

What is the good algorithm should I used, If I have 1000 persons (49000 images)?

^ Reply

 **Sefik Serengil**

March 27, 2020 at 8:18 am

You need to apply face recognition in real time? If yes, OpenFace is pretty. Because it has close accuracy to complex models such as VGG-Face, FaceNet and DeepFace. But if you can do it as batch, these complex models would be better.

^ Reply

 **Sefik Serengil**

March 27, 2020 at 8:19 am

Please check it here: <https://cmusatyalab.github.io/openface/models-and-accuracies/>

The both models have different design, complexity and accuracy scores.

^ Reply

 **Yasser**

March 27, 2020 at 4:01 pm

Thank you very much for the respond

What is better openface with mtcnn or dlib face recognition?

^ Reply

 **Sefik Serengil**

March 27, 2020 at 4:07 pm

I do not know them both

^ Reply

Daniel

June 8, 2021 at 9:09 am

Dear Mr. Serengil,

at first i like to say thanks for your amazing work.

I have a important question about this.

Im using your framework but i dont know why its not giving me the same distance (similarity-scores) than the original openface framework.

My code is:

```
OpenFaceModel = DeepFace.build_model("OpenFace")
result = DeepFace.verify(lennon1,clapton2,"OpenFace"
,"euclidean_l2",OpenFaceModel,detector_backend="dlib",enforce_detection=True)
print(result['distance'])
```

And i wanted to check if i get the same result like here:

<https://cmusatyalab.github.io/openface/demo-2-comparison/>

The verification between lennon1-image and clapton2-image gives me a distance of 0.6152904504327942. But if you check the openface-website, the distance should be 1.145.

Why is this so different?

^ Reply

 **Sefik Serengil**

June 9, 2021 at 7:01 pm

Hi, this covers a tensorflow re-implementation of openface. Even though, it has a same structure, its weights are different.

If you need identical model, please read this tutorial:

<https://sefiks.com/2020/07/24/face-recognition-with-opencv-dnn-in-python/>

^ Reply

 **Daniel**

June 10, 2021 at 1:39 pm

Thanks for your advise.

I managed to run the Code from <https://sefiks.com/2020/07/24/face-recognition-with-opencv-dnn-in-python/> but it still doesnt have the same distance scores between the images.

For lennon1 and lennon2 image comparison i get these scores:

Euclidean distance: 0.6823

EuclideanL2 distance: 0.6823

Cosine distance: 0.2328

For lennon1 and claptopn2 image comparison i get these scores:

Euclidean distance: 0.5396

EuclideanL2 distance: 0.5396

Cosine distance: 0.1456

Its using the identical model though, but the scores are still different.

Could it be that the difference is happening because of the different detection methods? In your example you are using opencv whereas openface uses dlib.

Thank you for your help.

Im looking for a way to use deepface with facenet and openface, where i get the same scores like the original frameworks.

[^ Reply](#)

 **Sefik Serengil**

June 10, 2021 at 1:40 pm

The reason is that detection methods are different.

[^ 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.](#)

Licensed under a [Creative Commons Attribution 4.0 International License](#).



You can use any content of this blog just to the extent that you cite or reference

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

```
@misc{sefiks13489,  
  author = {Serengil, Sefik Ilkin},
```

```
title = { Face Recognition with OpenFace in Keras
},
howpublished = {
https://sefiks.com/2019/07/21/face-recognition-
with-openface-in-keras/ },
year = { 2019 },
note = "[Online; accessed 2023-01-13]"
}
```