



Sefik Ilkin Serengil

Open LHS

Model Any Disease or Condition

OPEN

× 广告

Code wins arguments

Menu ▾

Deep Face Recognition with ArcFace in Keras and Python

December 14, 2020 / Machine Learning

Support me on Patreon

 **BECOME A PATRON**

Haven't you subscribe my channel yet?

Follow me on Twitter

Follow @serengil

ArcFace is developed by the researchers of Imperial College London. It is a module of InsightFace face analysis toolbox. The original study is based on MXNet and Python. However, we will run its third part re-implementation on Keras. The original study got 99.83% accuracy score on LFW data set whereas Keras re-implementation got 99.40% accuracy. So, re-implementation seems robust as well. Besides, MXNet model is seen to be reproducible with Keras. So, we will build ArcFace model from scratch in this post.




Apple Face Id launch

Face recognition pipeline

A pipeline consists of 4 common stages: detect, align, represent and verify. Herein, ArcFace is a regular face recognition algorithm responsible for representation.


Online Course



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



Source code

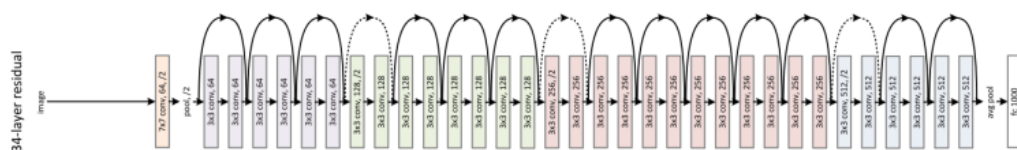
The source code of this study is [pushed](#) to GitHub as a jupyter notebook. You should clone it before starting this tutorial. Besides, you should clone [ResNet architecture building](#) dependency and [pre-trained weights](#). BTW, you can support this study★ if you star the repo.

Pre-trained model

Keras re-implementation of ArcFace shared the pre-trained model in its repo. However, it is saved as monolithic. I mean that model structure and pre-trained weights are stored in a single h5 file [here](#). However, model was saved in tensorflow 2 and it might cause troubles if you try load the model in different tensorflow versions. That's why, I prefer to build model structure in the code manually and save just pre-trained weights to avoid version problems.

Model structure

ArcFace is mainly based on ResNet34 model. The following illustration explains the model. It has a very complex architecture.



ResNet34 Architecture

I've already designed the network architecture from scratch. You should download ArcFace.py [here](#), and then call its load model function as demonstrated below. Notice that

the following program should be in the same directory with ArcFace.py.

```
1 | #https://github.com/serengil/deepface/blob/master/deepface/basemodels/ArcFace.py
2 | import ArcFace
3 | ArcFace.loadModel()
```

ArcFace model expects (112, 112, 3) shaped inputs whereas it returns 512 dimensional vector representations.

Pre-trained weights

Your friendly neighbor blogger saved just pre-trained weights and share in [Google Drive](#). Its size is 133 MB. We have already built the model structure in the previous step. Now, we can load the pre-trained weights as shown below.

```
1 | #Google Drive Link: https://drive.google.com/uc?id=1LVB3CdVejpgmGHM28Bpg
2 | model.load_weights("arcface_weights.h5")
```

Early stages of pipeline

ArcFace is responsible for the representation stage of a face recognition pipeline whereas detection and alignment are early stages. Luckily, [deepface](#) can handle those early stages. It wraps opencv, ssd, mtcnn and dlib for face detection.

My experiments show that MTCNN is the most robust detector but it is the slowest. SSD is the fastest one but its alignment is not as good as mtcnn.

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.

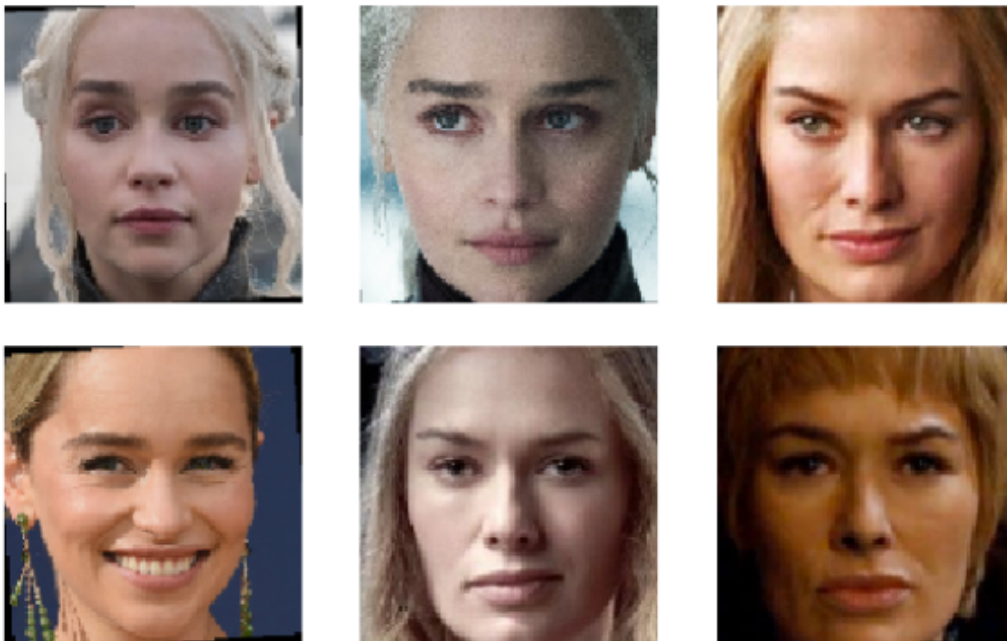
In addition, MediaPipe can find 468 landmarks. Please see its real time implementation in the following video. Recommended tutorials: [Deep Face Detection with MediaPipe](#), [Zoom Style Virtual Background Setup with MediaPipe](#).

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.

Preprocessing

```
1  #!pip install deepface
2  from deepface.commons import functions
3
4  img1_path = "img1.jpg"
5  img2_path = "img2.jpg"
6
7  img1 = functions.preprocess_face(img1_path, target_size = (112, 112))
8  img2 = functions.preprocess_face(img2_path, target_size = (112, 112))
```

I'm going to test the model for the two iconic characters of Game of Thrones: Emilia Clarke (Daenerys Targaryen) and Lena Headey (Cersei Lannister). In particular, Emilia's daily appearance and her role look very different. I even might not recognize her even if I didn't know her. Similarly, Lena looks very different with her short and long hair.



Game of Thrones ladies

Representation

We already applied detection and alignment to facial images and also resize them to the expected size. Now, we can feed those preprocessed facial images to the ArcFace.

```
1 | img1_embedding = model.predict(img1)[0]
2 | img2_embedding = model.predict(img2)[0]
```

Verification

We fed facial images to a CNN model and it represents facial image pairs to 512 dimensional vectors. Here, we expect that distance between the image pair representations should be low for same person whereas it should be higher for different persons.

```
1 | from deepface.commons import distance as dst
2 |
3 | metric = 'euclidean'
```

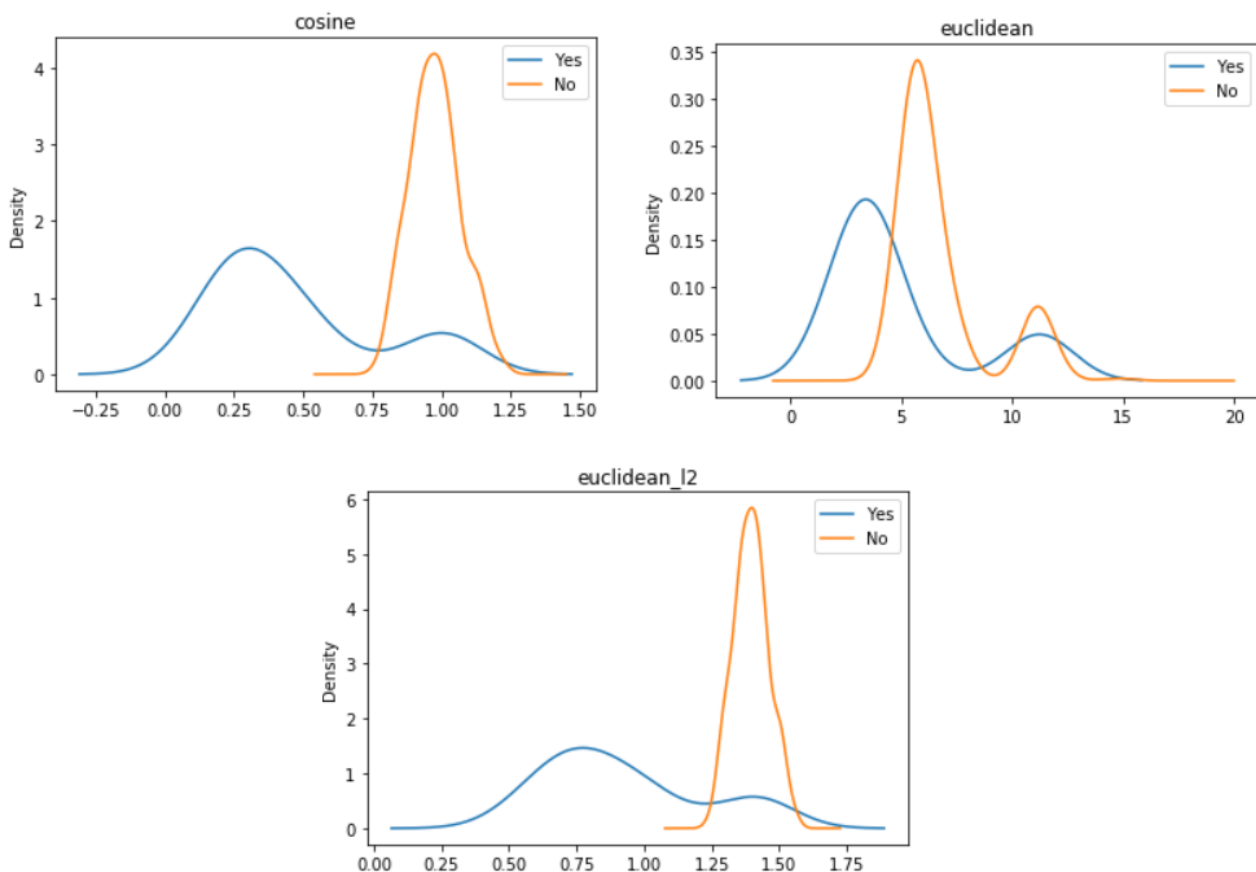
```

4
5 if metric == 'cosine':
6     distance = dst.findCosineDistance(img1_embedding, img2_embedding)
7 elif metric == 'euclidean':
8     distance = dst.findEuclideanDistance(img1_embedding, img2_embedding)
9 elif metric == 'euclidean_l2':
10    distance = dst.findEuclideanDistance(dst.l2_normalize(img1_embedding),

```

Threshold

We have distance values but how to determine a distance is low or high? The easiest way to determine it is to feed lots of positive and negative instances. Then, decision tree algorithms can find the best split point. Here, you can find a detailed tutorial: [Fine Tuning the Threshold in Face Recognition](#). I fed the [unit test images](#) of deepface. Here, master.csv stores the image pairs and they are same person or not. There are 37 positive; 239 negative; 276 total instances here.



Distributions

It seems that yes and no classes are distributed discretely. That's good. Target labels are unbalanced. That's why, no classes seem higher values in the y-axis.

When I feed distance values and target classes to the [C4.5 algorithm](#), it finds the best threshold values when information gain maximizes.

```

1 def findThreshold(metric):
2     if metric == 'cosine':
3         return 0.6871912959056619
4     elif metric == 'euclidean':

```

5
6
7

Accuracy

As I mentioned before, this keras re-implementation got 99.40% accuracy on [LFW data set](#). I also fed the unit test instances of deepface and find the best thresholds. Here are the accuracy, precision and recall values for each metric based on found thresholds.

| metric | precision | recall | f1-score | accuracy |
|--------------|-----------|--------|----------|----------|
| cosine | 0.98 | 0.89 | 0.93 | 0.97 |
| euclidean | 0.98 | 0.86 | 0.91 | 0.96 |
| euclidean l2 | 0.98 | 0.89 | 0.93 | 0.97 |

Decision

We have the both distance and threshold values for each distance metric. Now, let's find the decision.

1
2
3
4
5
6

True positive pairs

ArcFace amazingly verifies the identities of Emilia Clarke and Lena Headey. Emilia has a very different look in her daily life and the role in GOT. Similarly, Lena's look is very different when she has a short hair and long hair. Still the ArcFace model verifies them.

they are same person
Distance is 3.9 whereas as expected max threshold is 4.16



they are same person
Distance is 4.13 whereas as expected max threshold is 4.16



they are same person
Distance is 4.02 whereas as expected max threshold is 4.16



they are same person
Distance is 2.67 whereas as expected max threshold is 4.16



they are same person
Distance is 3.61 whereas as expected max threshold is 4.16



they are same person
Distance is 3.7 whereas as expected max threshold is 4.16



True positives

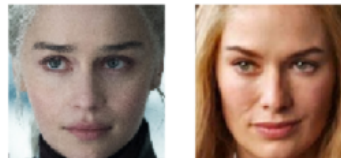
True negative pairs

ArcFace succeeded the verification for true negative pairs. That's an amazing work!

they are different persons
Distance is 5.47 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.7 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.09 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.42 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.77 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.05 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.32 whereas as expected max threshold is 4.16



they are different persons
Distance is 5.76 whereas as expected max threshold is 4.16



Running ArcFace in deepface

We have built ArcFace model from scratch and applied all stages of a pipeline step by step. On the other hand, we can build and run ArcFace model within deepface with a few lines of code.

We can apply face verification or find an identity in a database. We just set model name argument to ArcFace here.

```
1  #!pip install deepface
2  from deepface import DeepFace
3
4  #face verification
5  obj = DeepFace.verify("img1.jpg", "img2.jpg", model_name = 'ArcFace')
6  print(obj)
7
8  #face recognition
9  df = DeepFace.find("img1.jpg", db_path = "C:/my_db", model_name = 'Arc
10 print(df.head())
```

deepface also wraps [VGG-Face](#), [Google FaceNet](#), [OpenFace](#), [Facebook DeepFace](#), [DeepID](#) and [Dlib](#) models. Herein, the both FaceNet, VGG-Face and Dlib overperform than others.

Real time face recognition

It supports real time face recognition as well.

Large scale face recognition

A face recognition pipeline actually verifies an image pair is same person or different persons. Herein, [face recognition](#) requires to apply face verification several times. Deepface can find an identity in a database fast because it stores the representations of database items beforehand.

Notice that face recognition requires $O(n)$ time complexity and this becomes problematic for millions level data. Herein, you can run deepface with [Elasticsearch](#).

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.

Ensemble

deepface wraps many face recognition models and they are all amazing. Herein, deepface offers a special [boosting solution](#) to improve the model accuracy. This comes with higher accuracy but it is really slow. If your priority is the accuracy instead of speed, you might think to adopt this approach.

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

In this post we mentioned a state-of-the-art face recognition model, how to build and run it from scratch, and its performance. Besides, how to run it with a few lines of code as well.

I [pushed](#) the source code of this study to GitHub. You can support this study if you star★ the repo.

Like this blog? Support me on Patreon

 **BECOME A PATRON**

#arcface, #face recognition

« *Previous* / *Next* »

4 Comments

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

Sam

September 16, 2021 at 10:25 pm

Deepface weights take a long time to load. I am using an Intel Core i3 processor with 8 GB RAM, Windows. It takes 50 seconds to 2 minutes to load the weights.

“DeepFace.analyze” with images does not take that much time though.

Would you please suggest how I can reduce the loading time reasonably?

^ Reply

Toh Zhen Kang

September 26, 2022 at 6:56 am

Hi, a wonderful notebook to follow through, most resources are accessible and convenient.

However, you mention that you have already provided the pre-train weights for Arcface in this notebook, I would like to know the dataset used to train this. Is it all right if you share the data?

^ Reply

 **Sefik Serengil**

September 30, 2022 at 10:35 pm

I did not train the model. I just use the pre-trained weights.

[^ Reply](#)

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.](#)



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{sefiks15058,  
  author = {Serengil, Sefik Ilkin},  
  title = { Deep Face Recognition with ArcFace in  
Keras and Python },  
  howpublished = {  
https://sefiks.com/2020/12/14/deep-face-  
recognition-with-arcface-in-keras-and-python/  
},  
  year = { 2020 },  
  note = "[Online; accessed 2023-01-13]"  
}
```