Code wins arguments

Menu

Deep Face Recognition with Keras

August 6, 2018 / Machine Learning

Support me on Patreon

BECOME A PATRON

Haven't you subscribe my channel yet?

Follow me on Twitter

Follow @serengil

Face recognition has always been challenging topic for both science and fiction. A woman has her hair dyed or worn a hat to to disguise. Deep learning tasks usually expect to be fed multiple instances of a custom class to learn (e.g. lots of pictures of someone). This makes face recognition task satisfactory because training should be handled with limited number of instances – mostly one shot of a person exists. Moreover, adding new classes should not require reproducing the model. In this post, we'll create a deep face recognition model from scratch with Keras based on the recent researches.

Dramatic transformation of Katy Perry

Oxford visual geometry group announced its deep face recognition architecture. We have been familiar with VGG in imagenet challenge. We can recognize hundreds of images just applying transfer learning. Also, we have used same model for style transfer.

Objective

Face recognition is a combination of CNN, auto-encoder and transfer learning studies. I strongly recommend you to read How Face Recognition Works post to understand what a face recognition pipeline is.

Network configuration

Even though research paper is named Deep Face, researchers give VGG-Face name to the model. This might be because Facebook researchers also called their face recognition system DeepFace – without blank. VGG-Face is deeper than Facebook's Deep Face, it has 22 layers and 37 deep units.

The structure of the VGG-Face model is demonstrated below. Only output layer is different than the imagenet version – you might compare.

VGG-Face model

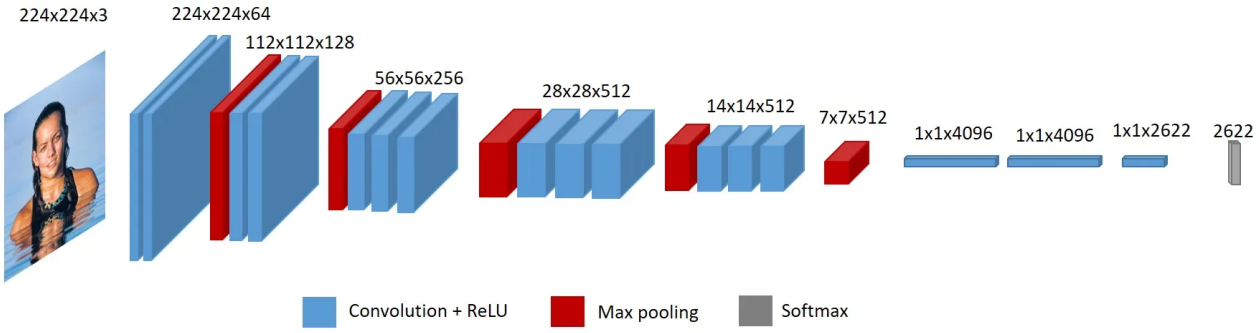
Research paper denotes the layer structre as shown below.

layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
type	input	conv	relu	conv	relu	mpool	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv
name	–	conv1_1	relu1_1	conv1_2	relu1_2	pool1	conv2_1	relu2_1	conv2_2	relu2_2	pool2	conv3_1	relu3_1	conv3_2	relu3_2	conv3_3	relu3_3	pool3	conv4_1
support	–	3	1	3	1	2	3	1	3	1	2	3	1	3	1	3	1	2	3
filt dim	–	3	–	64	–	–	64	–	128	–	–	128	–	256	–	256	–	–	256
num filts	–	64	–	64	–	–	128	–	128	–	–	256	–	256	–	256	–	–	512
stride	–	1	1	1	1	2	1	1	1	1	2	1	1	1	1	1	1	2	1
pad	–	1	0	1	0	0	1	0	1	0	0	1	0	1	0	1	0	0	1

layer	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
type	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	relu	mpool	conv	relu	conv	relu	conv	softmx
name	relu4_1	conv4_2	relu4_2	conv4_3	relu4_3	pool4	conv5_1	relu5_1	conv5_2	relu5_2	conv5_3	relu5_3	pool5	fc6	relu6	fc7	relu7	fc8	prob
support	1	3	1	3	1	2	3	1	3	1	3	1	2	7	1	1	1	1	1
filt dim	–	512	–	512	–	–	512	–	512	–	512	–	–	512	–	4096	–	4096	–
num filts	–	512	–	512	–	–	512	–	512	–	512	–	–	4096	–	4096	–	2622	–
stride	1	1	1	1	1	2	1	1	1	1	1	1	2	1	1	1	1	1	1
pad	0	1	0	1	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0

VGG-Face layers from original paper

I visualize the VGG-Face architure to be understood clear



Visualization of VGG-Face

Let's construct the VGG Face model in Keras

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(224,224, 3)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(4096, (7, 7), activation='relu'))
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation='relu'))
model.add(Dropout(0.5))
```

```
model.add(Convolution2D(2622, (1, 1)))
model.add(Flatten())
model.add(Activation('softmax'))
```

Learning outcomes

Research group shared pre-trained weights on the group page under the path vgg_face_matconvnet / data / vgg_face.mat, but it is matlab compatible. Here, **your friendly neighborhood blogger** has already transformed pre-trained weights for Keras. If you wonder how matlab weights converted in Keras, you can read this article. Due to weight file is 500 MB, and GitHub enforces to upload files smaller than 25 MB, I had to upload pre-trained weights in Google Drive. You can find the pre-trained weights here.

```
from keras.models import model_from_json
model.load_weights('vgg_face_weights.h5')
```

Finally, we'll use previous layer of the output layer for representation. The following usage will give output of that layer.

```
vgg_face_descriptor = Model(inputs=model.layers[0].input
, outputs=model.layers[-2].output)
```

Representation

In this way, we can represent images as 2622 dimensional vector as illustarted below.

```
img1_representation = vgg_face_descriptor.predict(preprocess_image('1.j
img2_representation = vgg_face_descriptor.predict(preprocess_image('2.j
```

Notice that VGG model expects 224x224x3 sized input images. Here, 3rd dimension refers to number of channels or RGB colors. Besides, preprocess_input function normalizes input in scale of [-1, +1].

```
def preprocess_image(image_path):
img = load_img(image_path, target_size=(224, 224))
img = img_to_array(img)
img = np.expand_dims(img, axis=0)
img = preprocess_input(img)
return img
```

Vector Similarity

We've represented input images as vectors. We will decide both pictures are same person or not based on comparing these vector representations. Now, we need to find the distance of these vectors. There are two common ways to find the distance of two vectors: cosine distance and euclidean distance. Cosine distance is equal to 1 minus cosine similarity. No matter which measurement we adapt, they all serve for finding similarities between vectors.

```
def findCosineDistance(source_representation, test_representation):
a = np.matmul(np.transpose(source_representation), test_representation
b = np.sum(np.multiply(source_representation, source_representation))
c = np.sum(np.multiply(test_representation, test_representation))
return 1 - (a / (np.sqrt(b) * np.sqrt(c)))

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

Recognizing images

We've represented images as vectors and find the similarity measures of two vectors. If both images are same person, then measurement should be small. Otherwise, the measurement should be large if two images are different person. Here, epsilon value states threshold.

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

```
epsilon = 0.40 #cosine similarity
#epsilon = 120 #euclidean distance

def verifyFace(img1, img2):
img1_representation = vgg_face_descriptor.predict(preprocess_image(img
img2_representation = vgg_face_descriptor.predict(preprocess_image(img

cosine_similarity = findCosineSimilarity(img1_representation, img2_rep
euclidean_distance = findEuclideanDistance(img1_representation, img2_r

if(cosine_similarity < epsilon):
print("verified... they are same person")
else:
print("unverified! they are not same person!")
```

Cosine similarity should be less than 0.40 or euclidean distance should be less than 120 based on my observations. Thresholds might be tuned based on your problem. It is all up to you to choose the similarity measurement.

This is **one shot learning** process. I mean that we would not feed multiple images of a person to network. Suppose that we store a picture of a person on our database, and we would take a photo of that one in the entrance of building and verify him. This process can be called **face verification** instead of face recognition.

Some researchers call finding distances of represented two images as **Siamese networks**.

Testing

I tested the developed model with variations of Angelina Jolie and Jennifer Aniston. Surprisingly, the model can verify all instances I fed. For example, Angelina Jolie is either blonde or brunette in the following test set. She even wears a hat in a image.

Detecting Angelina Jolie

The mode is very successful for true negative cases. Descriptor can overwhelmingly detect Angelina Jolie and Jennifer Aniston.

Detecting that Jennifer Aniston and Angelina Jolie are different person

True positive results for Jennifer Aniston fascinate me. I might not detect the 3rd one (2nd row, 1st column). Jennifer is at least 10 years young in that photo.

True positive cases for Jennifer Aniston

I think the revolutionary testing was on Katy Perry. The face recognition model can recognize her even for dramatic appearance change.

```
Cosine similarity:  0.31454116106033325
Euclidean distance:  103.37808
verified... they are same person
```

Dramatic transformation of Katy Perry can be detected

Of course, I can test the model for limited number of instances. The model got 98.78% accuracy for labeled faces in the wild dataset. The dataset contains 13K images of 5K people. BTW, researchers fed 2.6 M images to tune the model weights.

Having the hair dyed or wearing hat just like in movies do not work against AI systems. Movie producers should find more creative solutions.

Angelina Jolie has her hair dyed in Salt (2010)

Real Time Deep Face Recognition

We can apply deep face recognition in real time as well. Face pictures in database represented as 2622 dimensional vector at program initialization once. Luckily, opencv can handle face detection. Then, we will represent detected face and check similarities.

Early stages of face recognition pipeline

Modern face recognition pipelines consist of 4 stages: detect, align, represent and classify / verify. We've ignored the face detection and face alignment step 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.

Moreover, 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

So, we can recognize faces easily on-premise by combining transfer learning and auto-encoder concepts. Additionally, some linear algebra ideas such as cosine similarity contribute the decision. We've fed frontal images to the model directly. Finally, I pushed

the source code of project on my GitHub profile. BTW, I run the code for TensorFlow backend.

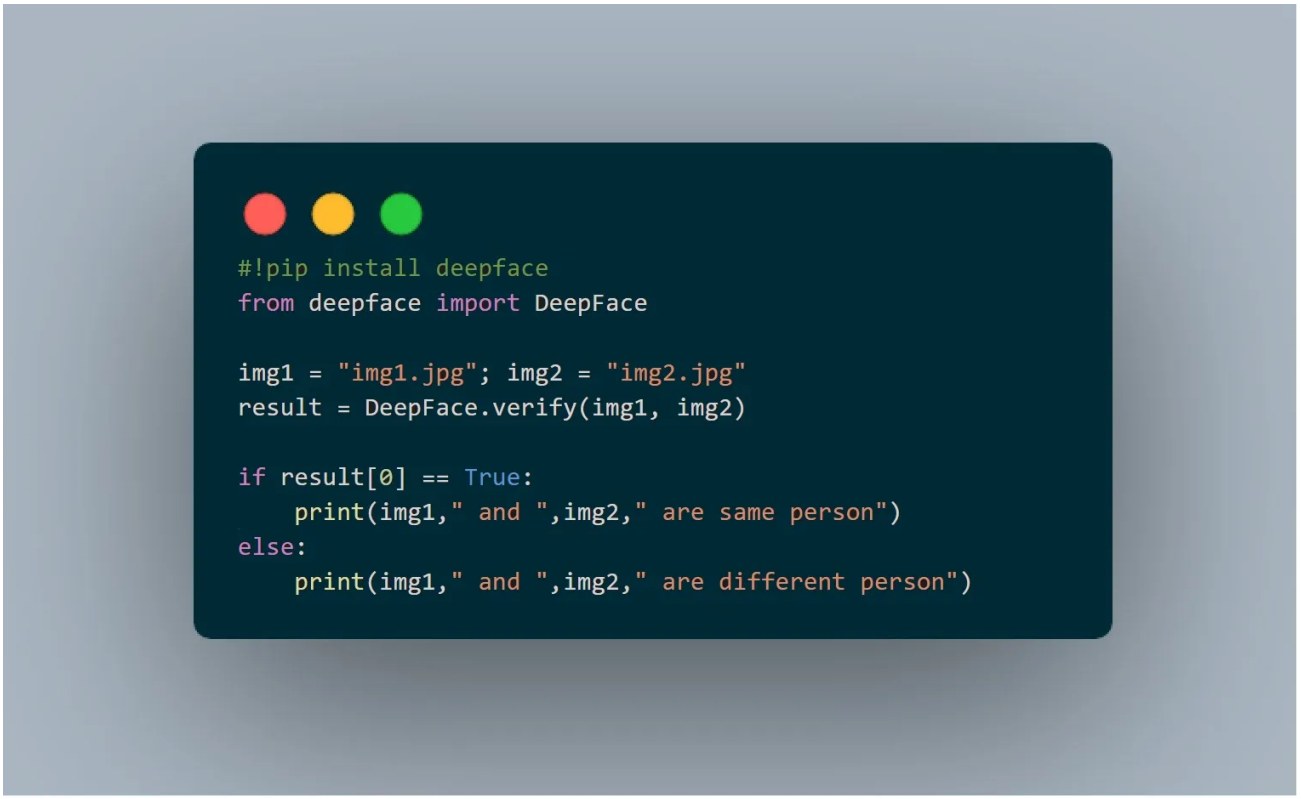
Finally, Google has Facenet, Carnegie Mellon University has OpenFace and Facebook has DeepFace face recognition models as an alternative to VGG-Face.

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, OpenFace, Facebook DeepFace and DeepID.

It handles building pre-designed model, loading pre-trained weights, finding vector embedding of faces and finding similarity to recognize faces in the background. So, you can verify faces with a just few lines of code.

It is available on PyPI as well. You should run the command "**pip install deepface**" to have it. Its code is also fully open-sourced in GitHub. Repository has a detailed documentation for developers. BTW, you can support this project by starring the repo.



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

#autoencoder, #face recognition, #face verification, #transfer learning, #vgg

« *Previous* / *Next* »

66 Comments

Hani
March 9, 2019 at 10:07 am

Good day.

First of all, thank you for this! It is very helpful.

However, I cant seem to import my images to the model. I am getting the error:

"ValueError: operands could not be broadcast together with remapped shapes [original->remapped]: (64,226,226)->(64,newaxis,newaxis) (226,226,64)->(226,newaxis,newaxis) and requested shape (226,64)"

I manually cropped my dataset and have resized them to 224×224 but I still keep getting this error. Do you have an idea as to why I get this error? Your input will be greatly appreciated!

Reply

Sefik Serengil
March 9, 2019 at 11:30 am

You are using Tensorflow backend?

Reply

Hani
March 9, 2019 at 2:55 pm

I was using tf.keras.layers for the layers when it should've been keras.layers only. But thank you for your fast reply!

Reply

Hani
March 9, 2019 at 2:54 pm

I have found and fixed the problem hehe

Reply

gaurav
May 22, 2019 at 7:02 pm

What's the error u found?

Reply

ali
March 17, 2019 at 3:45 pm

How can I feed both classes with multiple images to get better accuracy ?

Reply

Sefik Serengil
March 17, 2019 at 3:47 pm

You should not! You can have hundreds of pictures of 10 people and design a simple CNN. What if 11th member joined in your team? You need to re-design your network first because previously your network had 10 outputs, not it should be 11. Instead of designing a regular CNN, this approach is adopted. You design a network once, and you use it to find the representation of an image. Two different images of same person should have close similarity whereas different people should have distant similarity.

Reply

Ray
May 22, 2019 at 11:37 am

What if I took average vector of multiple predictions of one person? would that make this vector more accurate than using just one picture?

Reply

Sefik Serengil
May 22, 2019 at 11:46 am

This is not applicable. Suppose that you have 10 pictures of someone in your data set. Detected face on a web cam should have low distance for each 10 image in your data set if they are same. Calculating wouldn't overperform.

Reply

Ray
May 22, 2019 at 12:32 pm

I just tried the following on the LFW dataset on people with more than 1 picture, took predictions of each persons _0001 image and put it on an array, then ran loop trough the dataset and chose random person and random image which is not 0001, then using cosine simularity tried to find which row in array it is. accuracy of finding the right person was about 31%. then I took all images of every person, ran prediction on them and took the average description vector which was then saved in the array, afterwards again took random persons image from dataset and ran loop trough the array calculating the cosine distance and finding the closest match, now the results were 95.8% correct. both of these experiments are using 1680 people(9164 files) and 1000 random pictures to find match with. Results of course are affected by the fact that I used all images of each person to create this average description vector and used random images from the same 9164 images. But still this accuracy bump is impressive in my opinion

Sefik Serengil
May 22, 2019 at 2:03 pm

Good work

Amin
August 28, 2019 at 6:51 am

What if train a binary classifier such as logistic regression or SVM? Can it help us to save the time when we have large number of people in the team (10,000 for example)?

Reply

Sefik Serengil
August 28, 2019 at 12:21 pm

You still should not! Re-designing a model is not the best practice.

Reply

mohamad

December 2, 2019 at 9:16 pm

Hi Sefik, Thanks for your very nice Blog,
But let's say I have 10 picture for each person in my 1000 person collection and I now that these persons won't change,(no add no remove) in this case can I use CNN with 1000 output?
also is there any sample for this scenario?

Reply

Sefik Serengil
December 3, 2019 at 5:23 am

You can build a custom CNN in that scenario. However, daily applications require to add and remove entities, that's why, I could not see any application similar to your explanation.

Reply

Pasquale
May 17, 2019 at 6:37 am

Great article.
I'd like to make two question:
1) Did you add droput layers before last two convolutions or the original model from Oxford researchers had them ?
2) Do you think cosine distance is better than euclidean distance to recognize if two faces as the same person ?
Thank you.

Reply

Sefik Serengil
May 17, 2019 at 7:19 am

Thank you for your kindly response
1- Exactly the same model with Oxford researchers
2- Cosine distance performs better in VGG-Face model, whereas euclidean distance overperforms in Google's Facenet model. Researchers mentioned these metrics in their papers. (BTW, you can find facenet here: https://sefiks.com/2018/09/03/face-recognition-with-facenet-in-keras/)

Notice that VGG-Face has 2622 dimensional output vector and Facenet has 128 dimensional output vector. In my opinion, cosine overperforms when dimensions increase. But this is a heuristic answer. It is not based on experiments.

Reply

jh.lam

July 15, 2021 at 12:07 pm

yep, I found that even if same model with different versions(say vgg-face.caffemodel vs vgg-face keras) will result in significant difference in euclid but cosine.

Reply

Parth

May 17, 2019 at 7:09 pm

Hey Sefik, Great post!!
I have a couple of questions.
1. Can i use this model for recognizing multiple people in a real time videocapture?If so,how effective will this model be in that situation?
2. Is there a way to calculate the accuracy for this model, like a confusion matrix or something?

Thanks in advance

Reply

Sefik Serengil

May 18, 2019 at 8:34 am

Hi,
1- processing multiple faces is fine as is. My experiments didn't have any performance issue.
2- sure, you can create a confusion matrix.

Reply

Pasquale

May 18, 2019 at 8:27 am

Thank you for your kindly response.
I was wondering why VGG-Facenet has 2622 dimensional descriptor vector and I found that they had a dataset containing the faces of 2622 celebrities. In the previous layer I found a 4096 dimensional output vector.
In this presentation (slide 17)
http://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/presentation.pptx
researchers talk about "Learning a projection from 4096 to 1024 dimensions". Do you know how to obtain this 1024 dimensional output vector ?

Reply

Sefik Serengil

May 18, 2019 at 9:28 am

It might be a miswriting because it is obvious that 2622 output nodes in slide number 15 and 16. This is mentioned as “2622 way multi class criterion (soft max)”. Besides, researchers shared pre-trained weights and I transform weights to Keras format. This model represents faces as 2622 dimensional vectors, too.

Reply

Pasquale

May 18, 2019 at 10:23 am

According to their poster
http://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/poster.pdf
Oxford researchers developed a model:
– 2622 way classification
– 4096d descriptor
Since I’m not interested in classification but I want to predict if two images represent the same person, I think it would be better to take the output of the previous layer (4096 dimension).

Reply

Sefik Serengil

May 18, 2019 at 12:58 pm

You should try, it might have higher accuracy. Sincerely, all my experiments are based on 2622 dimensional vector.

Reply

Pasquale

May 20, 2019 at 11:08 am

Hi Sefik
I tried to use 4096d vector cutting a layer, but I got worse results.
On the other hand I found another your post dealing with Google FaceNet whose model uses a 128d descriptor vector only. But I wasn’t able to get that model working. I obtained crazy results.

Reply

Sefik Serengil

May 20, 2019 at 11:09 am

VGG-Face is better than Facenet 🙂

Reply

Sibi

June 23, 2019 at 3:14 pm

Hey I'm trying to adopt this CNN for a live feed and when I tried to test it on my workplace (~60 employees), I found that the accuracy was really low (~55%). This is probably a problem with the way I'm classifying the embeddings. I use a KNN with euclidian distance and uniform weights, I tried messing around with other distances and weights as well but similar results. Any idea on what other methods I could try?

*ignore previous comment

Reply

Sefik Serengil

June 23, 2019 at 3:16 pm

You cropped the for 60 employees? CNN model expects cropped faces as inputs.

Reply

Sibi

June 23, 2019 at 7:46 pm

Yup cropped the faces, tried with padding(margin) of 1-10 px, tried without as well. Not sure why the distance between embeddings is so low.

Reply

Sefik Serengil

June 24, 2019 at 5:42 am

Could you share some samples?

Reply

Sibi

July 6, 2019 at 10:08 am

Hi Sefik,

Sorry it took me a bit to reply, I fixed the issue (pre-processing wasn't happening correctly). But now that I run the test (currently in India), I see that the distance between people is only around 70-80 instead of the 120+ I see when I run the test on

Caucasian faces. I'm guessing this has something to do with the pretrained weights aka the image set used for tuning. Any idea on how I can retrain the model/weights on a different dataset?

Reply

Sefik Serengil
July 6, 2019 at 10:09 am

You should study siamase networks

Reply

Amanda
July 2, 2019 at 12:29 am

Hi,

I loved this article and it worked very well for me. I was just wondering if there was a way that I'd could see exactly which deep features are being extracted.

I'd also like to know if the classifier being used is a deep classifier or is it not considered a deep classifier because it represents the images as vectors and just finds a similarity score between them.

Reply

Sefik Serengil
July 2, 2019 at 3:34 am

It is a deep representer. You can see still deep features with some subsidiary tool such as shap or lime. Yesterday, I posted a post about shap but it has a bug on this network. I opened an issue about that.

Reply

Ugonna
December 4, 2020 at 4:15 pm

Please can this approach work when u apply it to a scramble or inverted face to be recognition and how do i introduce 5 fold validation to model because i have small dataset of 1k

Reply

Sefik Serengil

December 6, 2020 at 6:14 pm

Training is not performed in this approach. We just used the pre-trained weights.

Reply

Amanda

July 30, 2019 at 1:05 am

Hi,

I've implemented this model and it was working well at first; however, now that I'm trying to test multiple people against the same person and others, I'm receiving a fairly low AUC for the model. Is there anyway to finetune the model for a custom dataset?

Reply

Sefik Serengil

July 30, 2019 at 5:07 am

Faster way is tuning threshold. Better way is training your custom siamase network for your custom data set.

Reply

Amanda

July 30, 2019 at 1:05 am

Hi,

I've implemented this model and it was working well at first; however, now that I'm trying to test multiple people against the same person and others, I'm receiving a fairly low AUC for the model. Is there anyway to finetune the model for a custom dataset?

Reply

Amanda

July 30, 2019 at 12:07 pm

So, how would I be able to ise my own custom data in the program? I've looked up a bunch of stuff but none of it was really clear.

Reply

Sefik Serengil

July 30, 2019 at 1:41 pm

I plan to mention siamase networks from scratch but as you said it is a complex topic

Reply

GermanCh
August 8, 2022 at 10:05 am

Same happening to me!

Reply

Sefik Serengil
August 8, 2022 at 5:36 pm

You cannot use your own data. This is pre-trained model.

Reply

gin
November 6, 2019 at 2:04 am

i am very curious, why for each convolutional layer, the number of layers keeps increasing (64, 128, 256) however the size is constant at 3×3. usually, I apply an increasing layer number but with decreasing size.

btw good tutorial.
thanks

Reply

Sefik Serengil
November 6, 2019 at 4:18 am

It’s all state of the art of researchers. There is no answer why them construct this structure. Probably, they tried lots of model and this gave the best result in their experiments. There might be overperforming model than this.

Reply

Anabetsy Termini
March 23, 2020 at 8:11 pm

Hi there! Thank you for this. I have a question though. I tried running deep-face-real-time in Pycharm Mac OS. However, it keeps saying “employee representations retrieved successfully
qt.qpa.plugin: Could not find the Qt platform plugin “cocoa” in “”

This application failed to start because no Qt platform plugin could be initialized. Reinstalling the application may fix this problem.”
Could you please point me in the right direction to fix this? I have tried installing opencv headless as well PyQt5 but nothing seems to work on my Mac. Thanks in advance.

Reply

Sefik Serengil
April 1, 2020 at 6:03 am

Did you install opencv with the following command? pip3 install opencv-python-headless

Reply

Anabetsy Termini
April 1, 2020 at 6:07 am

Thanks for getting back to me! I fixed it since then but yes I had done that as well. I ended up creating a fresh virtual environment and it finally worked. Thanks again!

Reply

Osama Atallah
October 20, 2020 at 7:42 pm

Hello Sefik,
Thank you for your efforts building Deep Face and for this lovely article. I’m gathering more than 20000 photos for 500 middle-eastern childeren and eldery people. I want to fine-tune the last layer of the model to make it perform well for these two groups.

Please advise me on this. Also, what is the loss function I should use in model.compile()? and what is learning rate do recommend to start with?

Reply

Sefik Serengil
October 21, 2020 at 4:24 pm

Try adam and 0.01 pair.

Reply

Osama Atallah
October 20, 2020 at 7:50 pm

Cont.

Also please tell me how to train this model. Is it different than those for normal CNN classifiers?

Thank you

Reply

Sefik Serengil
October 21, 2020 at 4:25 pm

No it is a regular CNN. But you should drop the last layer when training is over. You will use pre-final layers to find representations instead of a classification task.

Reply

jh.lam
July 17, 2021 at 2:50 am

Hi @Sefik Serengil ,
I wonder a couple of questions:
1. what is the loss used in model which is stated by @Osama before? I think this should not be the categorical-crossentropy else this seems exactly a common CNN therefor how to learn face features?
2. similarly, why use the last 2nd layer ,say 'flatten layer', when in prediction? it's essentially a class output layer just before softmax.

thanks and looking forward your reply.

Reply

Sefik Serengil
July 17, 2021 at 5:21 pm

You should train this as a regular classification task. That's why, its loss should be categorical crossentropy.

Early layers of the CNN provide us representations. We will not use its classification result. That's why, we will drop its final layer and use its early layers to create embedding.

Flatten layer connects convolution layers to fully connected layers.

Reply

Osama Atallah

October 20, 2020 at 9:12 pm

Sorry. The dataset is for 5000 children and eldry people not 500

Reply

Priyank

October 27, 2020 at 7:51 pm

Hi Sefik,

Do you have nay blog where I can recognize face using partial face data i.e. if I pass a cropped image to detecter and then VGGFace or any other model – how can I normalize the image so that it can be detected or recognized as a face.

Thanks,
Priyank

Reply

Sefik Serengil

October 28, 2020 at 6:01 am

Read me of this repo might help you: https://github.com/serengil/deepface

Reply

Priyank

December 6, 2020 at 5:26 pm

Hi Sefik,

Is there a way we can create confusion matrix for two image comparison – i.e. if we are comparing images of Ang Jolie and Jen Aniston and calculating the cosine similarity or Euclidean distance between them – we have a value based on our threshold – either they are same person or different – can we form a confusion matrix on this result base only?

Reply

Sefik Serengil

December 6, 2020 at 6:07 pm

Yes, you can. Please read this post: https://sefiks.com/2020/08/27/labeled-faces-in-the-wild-for-face-recognition/

Reply

Priyank

December 6, 2020 at 5:27 pm

Hi Sefik,

Is there a way we can create confusion matrix for two image comparison – i.e. if we are comparing images of Ang Jolie and Jen Aniston and calculating the cosine similarity or Euclidean distance between them – we have a value based on our threshold – either they are same person or different – can we form a confusion matrix on this result base only?

Reply

Raghu Sanjeev

February 9, 2021 at 9:34 am

Hi Sefik,
1. I have a directory called Known_images where I have stored pictures of known persons. Each picture has only 1 image of the person
2. Now I have pictures in another directory called Unknwon_Images which has pictures of persons to do Facial recognition. Each picture may have multiple faces of persons. Some faces may be known and some may be unknwon.
3. How can we find out the faces of all the known people from the pciture that has multiple faces?

e.g.
– Known_Images – has pictures of Angelina Julie, Jeniffer Aniston, Britney Spears
– Uknown_Images has pictures that has 3 faces. One of them may be Angelina Julie other one is Jeniffer Aniston and third one is Michelle Obama. How can I do Face Reogntion in the above scenario?

Reply

Quique

June 13, 2021 at 8:19 am

Hi Sefik, there is something i do not understand at all. If the three last layers are Fully Connected ones, why did you put them as convolutional layers in the code and represented as such in your VGGFace's structure?

Reply

Sefik Serengil

June 13, 2021 at 8:23 am

Authors labeled those layers FC but they are actually convolution layers. You can check this in the table – "VGG-Face layers from original paper". For example, fc8 has 2622 num

filts value. Regular fully connected layers don't have filters.

Reply

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

---

Shawn Foster
May 22, 2022 at 1:15 pm

How did you calculate the accuracy for this? I'm facing troubles there

Reply

Sefik Serengil
June 2, 2022 at 9:00 am

I have not calculated that accuracy. It is mentioned in the reference paper.

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.

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

```
@misc{sefiks12474,
  author ={Serengil, Sefik Ilkin},
  title = { Deep Face Recognition with VGG-Face in
Keras | sefiks.com },
  howpublished = {
https://sefiks.com/2018/08/06/deep-face-
recognition-with-keras/ },
  year = { 2018 },
  note = "[Online; accessed 2023-01-13]"
}
```