Face Recognition with Triplet Loss in Keras

Szilárd Kósa
BME
Budapest, Hungary
szilard.kosa97@gmail.com

Lajos Bödő
BME
Budapest, Hungary
ifj.bodo.lajos@gmail.com

***Abstract*—In 2015 FaceNet introduced a new method for face recognition achieving a new record accuracy at that time. The essence of the idea is to map the face images into a 128 dimensional embedding on a unit hypersphere. The relation between two pictures can be determined from the distance of their embeddings. If two embeddings are close to each other that means the persons on the pictures look similar. Our goal is to create an implementation to this solution in Keras, and to generate visualization for the 128th dimensional representation of the face images using the newly released UMAP algorithm. To put our trained network into practice we made a python application using the Kivy library.**

I. Introduction

Two commonly mentioned part of face recognition is face verification and face recognition. In face verification the goal is to verify whether the given image is the claimed person or not. Sometimes it is called the '1:1' problem. Face recognition on the other hand is when we want to determine the identity of the person. For example we have a database of faces and we want to determine who the given face belongs to. Also mentioned as the '1:N' problem.

One solution to the face recognition problem is to get a lot of pictures from the persons who we want to recognize, and build a convnet that has an output for each person and one output if the person is not in the dataset. This way it could be solved as a classification problem. One of the big disadvantages of this solution is that we need to have lots of pictures from the persons we wish to recognize. Also if we want to add new persons to the database we would have to add more outputs to the model and retrain the network every time.

This problem can be solved with One-shot image recognition. This means that instead of having hundreds and thousands of pictures from one subject, we carry out the recognition using only one image. To make a network that can solve this problem we have to redefine the purpose of the model. Instead of building a network that can tell exactly who that person is, we have to build a network that can tell if the given two pictures portray the same person or not. When we train our network for this purpose, the face recognition task can be executed by comparing the given image with each image in the database. If the similarities between two images are above a threshold value then the output is that they are the same person. This way it is easy to use for both face recognition and face verification.

A. Related Work

This can be achieved by a Siamese network [6]. This idea comes from the DeepFace paper released at 2014 [2]. Their method reached an accuracy of 97.35% on the Labeled Faces in the Wild (LFW) dataset [9], [10]. In 2015 FaceNet continued this gain by reaching a new record accuracy of 99.63% on the same dataset [1]. Their new approach was in the training process. They trained their network with the triplet loss function.

One of the most well known open source implementation if the FaceNet is called OpenFace [12]. This was created using python and torch. With their latest release in 2016 their best model had an accuracy of 92.9%, improving their latest one from 76.1%.

When researching this topic we found that there aren't many representation of the FaceNet in Keras [1]. The ones we have found were either in Tensorflow or their purpose were only to load in a pretrained model. So we decided to create the whole training process in Keras.

II. Method

A. Triplet Loss

The triplet loss function requires three images. Two image from the same person and a third one from a different person. The first picture is called the anchor image, the second is the positive image and the last is the negative.

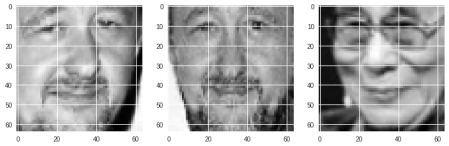

Fig. 1. An example for a triplet

Their embeddings are 128th dimensional vectors that are constrained to live on the unit hypersphere. The task of the loss function is to make the distance between the anchor and the positive images small and the distance between the anchor and the negative big. Thus the goal is to achieve the following inequality:

$$\|f(x_i^a) - f(x_i^p)\|^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|^2 \quad (1)$$

Where the alpha is a margin between the positive and the negative distances. This alpha is necessary because otherwise the network would tend to put the embeddings at the same point causing the distances to become zero. The value that we want to minimalize is:

$$L = \sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|^2 - \|f(x_i^a) - f(x_i^n)\|^2 + \alpha\right] \quad (2)$$

The problem is that if these triplets are generated randomly, after a short period of time the network will learn to distinguish them sufficiently. Meaning the network can't learn from these triplets anymore. The solution to this problem is to teach it on triplets that are hard to distinguish.

B. Triplet Generator

There are three kinds of triplets:

- easy negatives: $d(a,p) + \alpha < d(a,n)$
- semi-hard negatives: $d(a,p) < d(a,n) < d(a,p) + \alpha$
- hard negatives: $d(a,n) < d(a,p)$

Easy negatives are the ones that are already satisfying the inequation. Thus there would be no point to train on easy triplets because their loss value would be 0. We can train our network on semi-hard and on hard triplets. Although it is important to know that using hard triplets make the network more sensitive for bad data, for example mislabeled pictures. The generator's task is to generate this triplets. There are two ways this can be done:

- offline triplet mining: generating triplets every n steps, using the most recent network weights on a subset of data
- online triplet mining: selecting the triplets from within a mini-batch.

We used online triplet mining and mini-batches consisting of n people each with k pictures. Then we created all the possible combination for each person. And for all the possible pairs we chose one hard or semi-hard negative. Resulting in $k \cdot (k-1)/2 \cdot n$ triplet. If we couldn't find a right negative then that pair was discarded.

C. Network

For input size we chose 64x64 size grayscale images. The main motivation behind the smaller image size is to make the training process faster, and to allow our model to work well even on bad quality pictures.

For the model we used a deep convolutional network [11] with inception modules [3]. It is based on the OpenFace's nn4.small2.v1 network, which is a reduced version of the FaceNet nn4 model [1], [12]. We halved the filter numbers in the model significantly reducing it's size. We also changed the pooling layers in the inception modules to match our input dimensions. And we had to remove all batch normalization layers because they caused the training to collapse. The training loss went down to 0 meanwhile the validation loss started to converge to the margin value meaning all the distances became zero. By adjusting it's hyperparameters it didn't showed any improvement so we decided to remove them.

The model has three input and three output, for the triplet loss function. This architecture was done by the Keras functional API.

TABLE I
THE FINAL NETWORK

layer	input size	output size	parameters
conv 7x7	64x64x1	32x32x32	1600
maxpool 3x3	32x32x32	16x16x32	0
conv 1x1	16x16x32	16x16x32	1056
conv 3x3	16x16x32	16x16x96	27744
maxpool 3x3	16x16x96	8x8x96	0
inception3a	8x8x96	8x8x128	41016
inception3b	8x8x128	8x8x160	57056
inception3c	8x8x160	4x4x320	99568
inception4a	4x4x320	4x4x320	136576
inception4e	4x4x320	2x2x512	179504
inception5a	2x2x512	2x2x368	198048
inception5b	2x2x368	2x2x368	165792
average pool 2x2	2x2x368	1x1x368	0
flatten	1x1x368	368	0
fully connected	368	128	47232
normalizer	128	128	0
total	0	0	955192

III. EXPERIMENTS

A. Datasets

We used the **VGGFace2** dataset as our training database [8]. The VGGFace2 is a large-scale face recognition dataset, which contains 3.31 million images of 9131 subjects. The subjects have a great variation in pose, illumination, age and ethnicity.

We downloaded the "Train data_v1" and the "Test Data_v1" files from their website. We used the "picture_hunter.py" program to process the data into a hdf5 file. The hdf5 extension was chosen because it provides a relatively fast way to access it's content and it does not take up too much space on the hard disk. We decided that we will merge the train and test data, and later break it up into train and validation sets. For testing purposes we chose another database, The Labeled Faces in the Wild (LFW) [9], [10].

When we first created the dataset for the training, we used 40 image from each person. The face recognition was done with the OpenCV python module, based on the Haar Cascades. The "haarcascade_frontalface_default.xml" file was used as the cascade classifier. Those subjects, who were not recognized on their images at least 40 times were not used furthermore. The images were turned into grayscale images, and resized to 64x64 pixel size.

Later we remade the dataset to improve the learning process. The images with text on them were filtered out,

aligned in a way that placed the inner eyes and the bottom lip on the same place on every picture. We also filtered out the blurry pictures and those, which contained too much empty pixels. Our final dataset contains 7754 persons.

The Labeled Faces in the Wild (LFW) dataset was used as our test data [9], [10]. The LFW dataset contains 13233 images and 5749 people. The LFW data was pre-processed similarly to the training data, but if a picture could not be processed we used another instance of the picture from the LFWcrop Face Dataset [13]. The images from this dataset was already cropped and resized to 64x64, so it was a good alternative.

B. Training

The training process was a very time consuming process, so to get the best out of it, we made a lot of testing before training the final model. We tested several architectures. First, we had a small model with only four 3x3 convolutional layers and two max pooling layers after every second convolutional layer. Finally it contained the fully connected layers to produce the 128 embedding. We used this model to comparison.

The next tests was to add inception modules to the network. Finally we decided to use Openface's nn4.small2.v1 network with some modifications, as we found that this network produced the best results [12]. The biggest modification was to halve the number of filters in every layer making it significantly smaller. This caused the training time to go down substantially with only a very little performance drop.

For optimizer we decided next to the Adam optimizer with the learning rate set to 0.0001. Our loss function was the triplet loss function. It doesn't have an implementation in Keras so we had to define it as a custom loss function.

For the final training we had two hyper parameters that were still not tested:

- alpha: the margin in the triplet loss function
- negative triplet types: the triplet generation could be done in two ways, the generator could generate semi-hard or hard triplets

Using these hyperparameters, we could check all four combinations, and after the training, choose which one was the best model. The best validation loss was achieved by the model with 0.2 margin and semi-hard triplet generation.

TABLE II
TEST RESULTS

model	Precision	Recall	Accuracy	AUC
Semihard 02	0.8620	0.7972	0.8326	0.9085
Semihard 04	0.8198	0.8108	0.8127	0.8896
Hard 02	0.8533	0.8189	0.8365	0.9079
Hard 04	0.8301	0.7771	0.8064	0.8841

Since the project's goal was to create a good model for face recognition, the precision and accuracy values are both important. The "Semihard 02" model has the best precision rating, while it's accuracy only marginally worse than the "Hard 02" model. Also the AUC rating is also better for the "Semihard 02" model. So we chose the "Semihard 02" as our best model.

C. Testing

The testing was made with the LFW dataset [9], [10]. There were 10 sets of matched pairs and mismatched pairs. We determined a threshold value for a set using 10-fold cross validation. Using the determined threshold, we calculated the number of true positives, true negatives, false positives and false negatives based on our model's prediction. From this we gained an accuracy rating on the given set. The final accuracy rating was the average of the 10 set's accuracy ratings.

We also generated an ROC curve for each set, and an average ROC curve as well. We also calculated the AUC values, which provided us another feedback.

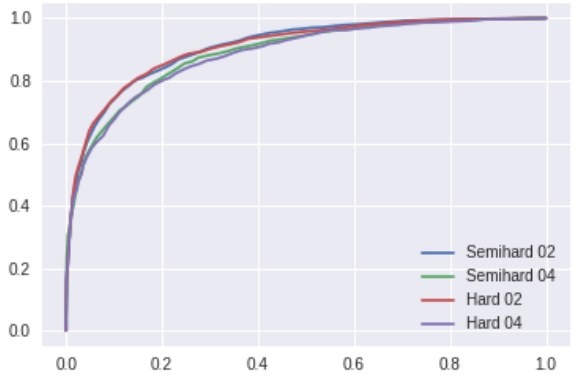

Fig. 2. The ROC curves of the models

We made a python application, which enables us to test our model in real time. The application ("webcam_app.py") records a person's face with the laptop's camera, then it gives the loaded model the live feed and the recorded image or images. The model decides if the picture from the live feed belongs to the recorded person. The application is capable of recording multiple people, and then recognizing them.

The live test is particularly useful seeing the trained models results in constantly changing environments.

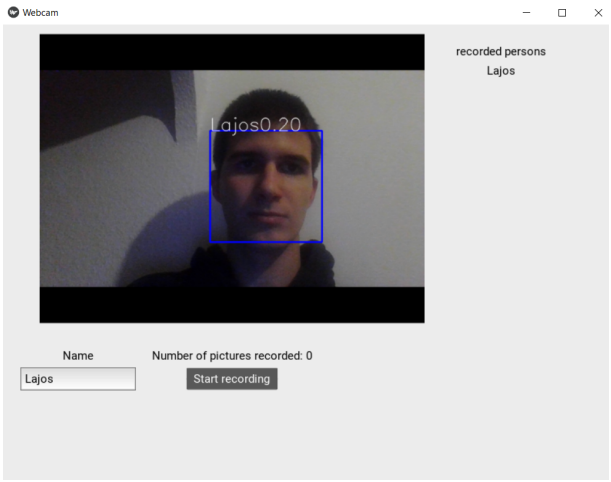

Fig. 3. A screenshot of the "webcam_app.py". The application frames the detected faces and writes out the calculated distance values from the closest recorded people. If a value is smaller than a predetermined threshold, the application recognizes the person. In the shown image the threshold was set to 0.7

D. Visualization

To visualize the training process we used the newly released UMAP algorithm [7]. At the current time this is the most powerful dimension reduction tool that we know of. As the output of the model is a 128th dimensional vector it makes it suitable for this type of representation. We compress it to the 2D to make it easy to examine. For this purpose we created a dataset from the LFW dataset containing 20 pictures from 20 people [9], [10]. If their embeddings are close to each other then the training were successful as the model sufficiently recognized the similarities in the pictures. First we created a diagram before the training to have comparison.

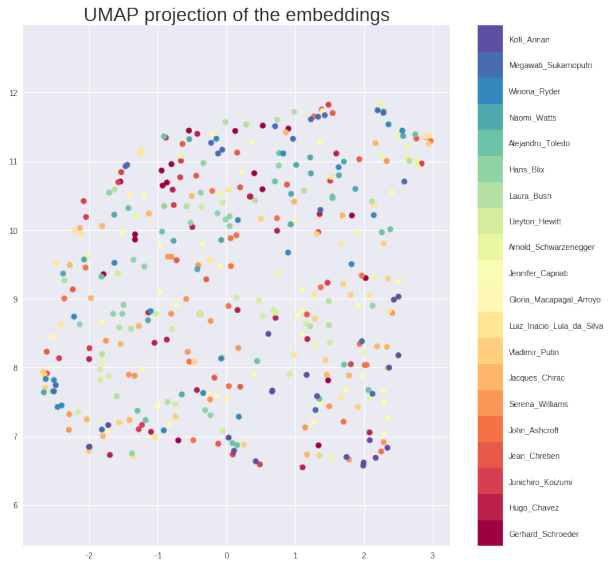

Fig. 4. The dimension reduction before the training

Finally here is how the dimension reduction looks like with our best model.

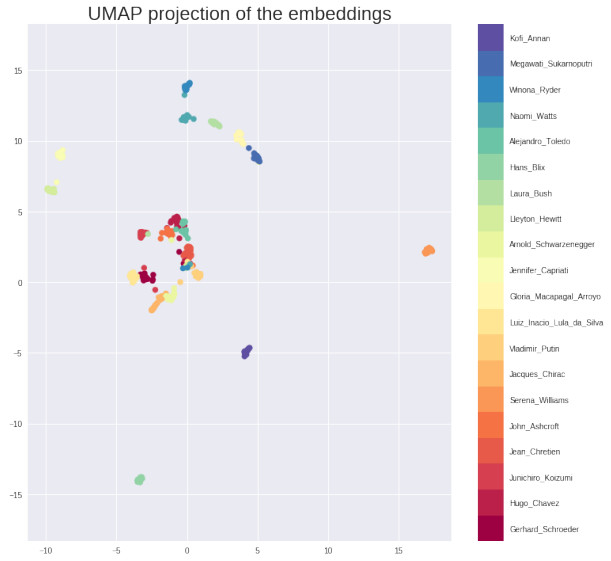

Fig. 5. The dimension reduction after the training

To make the figures more interactive we used the bokeh library to create plots with mouseover tooltips of the images.

IV. Future Plans

Continuing this project, we would test other models. The final model contained inception modules, so in the future we would experiment with different kind of inception modules. There are two main directions for these tests. The first is the factorization of the convolutional layers, which could possibly shorten the training time. We made one small test with a model similar to the inception V2 model style, where the 5x5 convolutional layers were factorized into two 3x3 convolutional layers [4]. The validation loss was slightly higher, than the simple model's, but we think more tests are needed for a concrete result. The second direction is to build the inception modules in a different way. An inception layer contains parallel convolutional layers and a pooling layer. We could make distinct modules with different number of convolutional layers or differently sized convolutional layers.

TABLE III
Our inception module ideas

Name	1x1	3x3	5x5	7x7	Pooling
Inception Small3	Yes	Yes	-	-	Yes
Inception Small5	Yes	-	Yes	-	Yes
Inception Medium5	Yes	Yes	Yes	-	Yes
Inception Medium7	Yes	Yes	-	Yes	Yes
Inception Large7	Yes	Yes	Yes	Yes	Yes

Another future model could be a much deeper convolutional network than the current one. This could be done

with a similar structure to the ResNet model to maintain the train-ability of the deep model [5].

We would also experiment with the margin's value for example change during the training. Changing the input images' size could also show interesting results. Of course the model would have to change a bit to accommodate the different input size. During this we could test the larger inception modules (the ones with the 7x7 convolutional layers) if the input images size would be bigger.

References

[1] Florian Schroff, Dmitry Kalenichenko and James Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering" arXiv:1503.03832v3, 2015.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. "Deepface: Closing the gap to human-level performance in face verification." In IEEE Conf. on CVPR, 2014.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." CoRR, abs/1409.4842, 2014.

[4] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision." arXiv preprint arXiv:1512.00567, 2015.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" arXiv:1602.07261v2, 2016.

[6] G. Koch, R. Zemel and R. Salakhutdinov, "Siamese Neural Networks for One-shot Image Recognition", 2015.

[7] Leland McInnes, "umap Documentation", 2018.

[8] Qiong Cao, Li Shen, Weidi Xie, Omkar M. Parkhi and Andrew Zisserman, "VGGFace2: A dataset for recognising faces across pose and age" arXiv:1710.08092v2, 2018.

[9] Gary B. Huang, Manu Ramesh, Tamara Berg and Erik Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments" University of Massachusetts, Amherst, Technical Report 07-49, October, 2007.

[10] Gary B. Huang and Erik Learned-Miller, "Labeled Faces in the Wild: Updates and New Reporting Procedures" University of Massachusetts, Amherst, Technical Report UM-CS-2014-003, May, 2014.

[11] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z. Li and Timothy Hospedales, "When Face Recognition Meets with Deep Learning: an Evaluation of Convolutional Neural Networks for Face Recognition" arXiv:1504.02351, April, 2015.

[12] Amos, Brandon and Bartosz Ludwiczuk and Satyanarayanan, Mahadev, "OpenFace: A general-purpose face recognition library with mobile applications" CMU-CS-16-118, CMU School of Computer Science, 2016.

[13] http://conradsanderson.id.au/lfwcrop/?fbclid=IwAR287yZXy_-zNN2o3nY7sBIUXq4edJX5pf7LcYNwmNbZDIYV8coCSH7YGMo 09th December 2018.