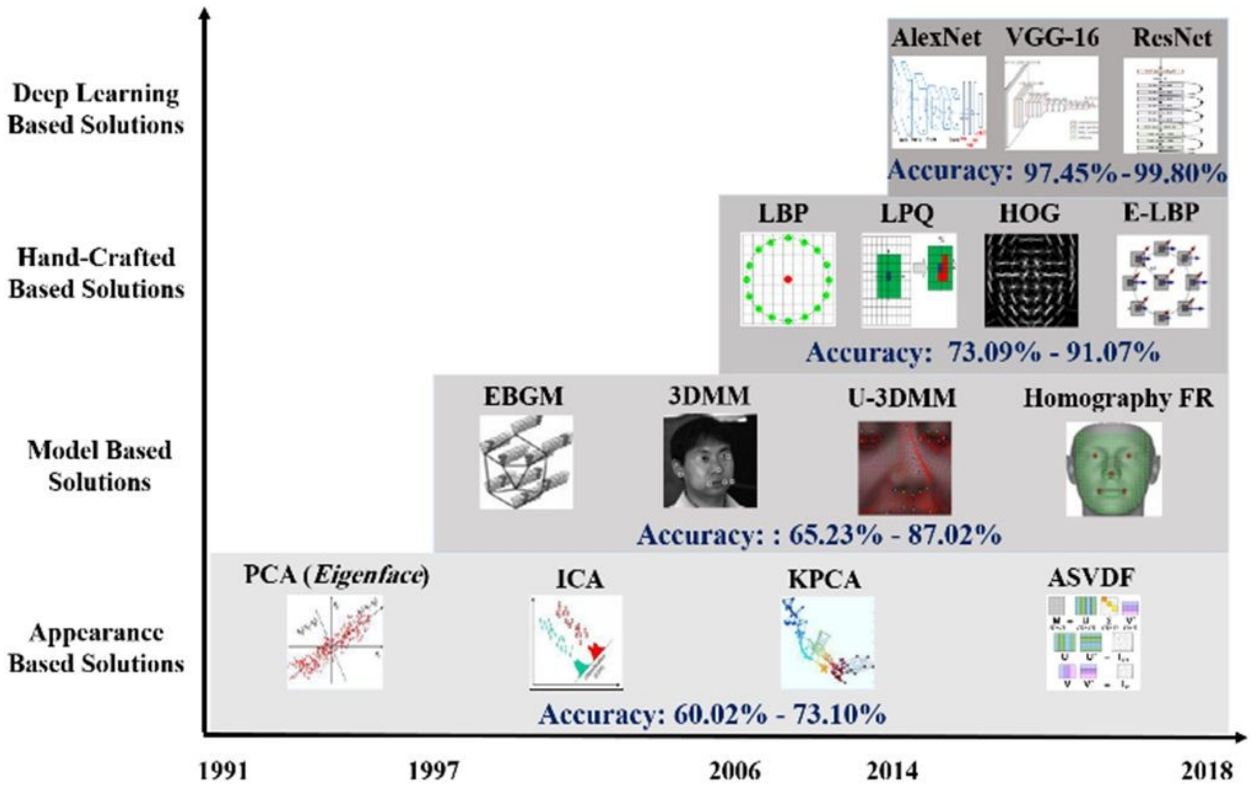
Face Recognition Project Final Report
Giray Eryılmaz

1) Problem definition and a brief survey of the recent literature

Face recognition is the problem of recognizing people from their face images, given that some images of the same people are already registered in the system. Or more broadly, given a scene image with peoples faces in it, face recognition is identifying or verifying one or more people in the scene by comparing their face images with faces already registered to the system (stored in a database).

The concept of face recognition dates back to nineteen eighties[1] but most satisfying results were obtained much recently with advances on learning based methods, primarily deep learning. The schema below is a brief summary of the history[2].


Deep Learning Based Solutions
AlexNet
VGG-16
ResNet
Accuracy: 97.45% - 99.80%
Hand-Crafted Based Solutions
LBP
LPQ
HOG
E-LBP
Accuracy: 73.09% - 91.07%
Model Based Solutions
EBGM
3DMM
U-3DMM
Homography FR
Accuracy: : 65.23% - 87.02%
Appearance Based Solutions
PCA (Eigenface)
ICA
KPCA
ASVDF
Accuracy: 60.02% - 73.10%
1991
1997
2006
2014
2018

Two examples of the state of the art are VGGnet and ResNet.

VGGNet was developed by Simonyan and Zisserman . It has 16 convolutional layers and has a uniform architecture. It has only 3x3 convolutions and lots of filters which are supposed to learn different features. VGGNet was trained on 4 GPUs for 2–3 weeks. The weight configuration of the VGGNet is publicly available thus is quite popular among the deep learning community for transfer learning tasks. However having 138 million parameters it is quite hard to train or even fine tune.

Residual Neural Network (ResNet) by Kaiming He et al was designed to tackle the problem of diminishing gradients. It uses some shortcuts to “skip connections” and heavy batch normalization. Thanks to this technique, despite having 152 layers it was able to learn properly and won at the ILSVRC 2015.

These above mentioned architectures are not the most recent advancements but they are more like keystones.

Apart from those, SphereFace[9] proposed a new loss for extracting features called Angular Softmax. They have showed that current loss methods (triplet, Contrastive, softmax) can be further improved.

CosFace[10] is another loss function aiming to improve face discrimination power of deep CNN architectures. They proposed a . Large Margin Cosine Loss with L2 normalization. They have reported some improvement on state of the art.

In my opinion the reason researchers are focusing on loss functions is because the deep learning architectures for classification are relatively well established. But a near-perfect loss function for minimizing intra class distance and maximizing inter class distance is yet to be found.

Zhihua Xie et al proposed[11] a fusion of LBPH and HOG. Their work is on infrared face recognition. They focused on finding an optimal fusion of these two methods and they reported that their fusion method performs clearly better than either methods. They did not make comparisons with learning based methods though.

2)Detailed explanation of the studied solution methods

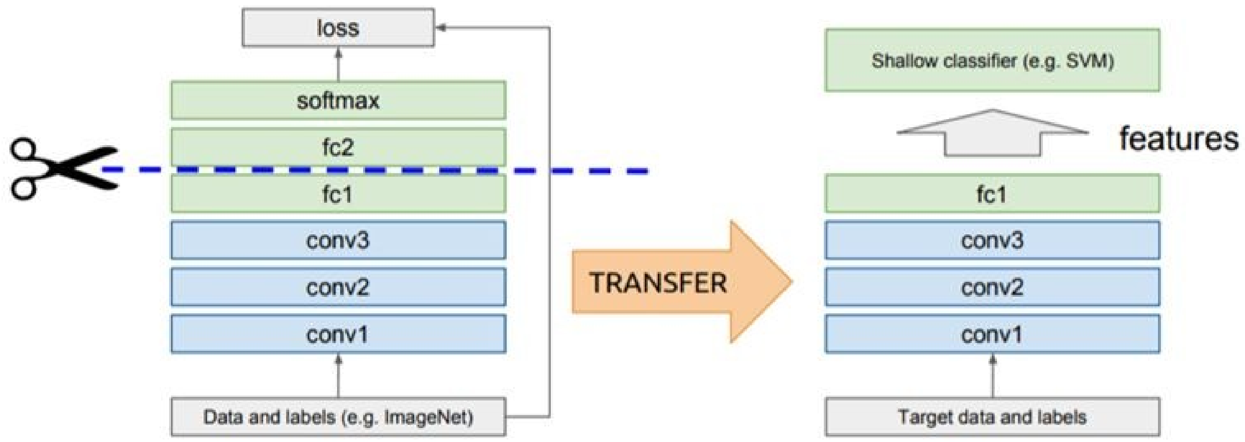
I have tested 2 methods:

1. A Deep Neural Network solution:

The state of the art solutions are convolutional deep neural networks with sometimes millions of parameters. Which require long time of training with powerful cpus/gpus or clusters of them.

At this point a very promising solution is transfer learning. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned[3]. Due to enormous computational complexity of training very deep neural networks with millions of parameters, transfer learning is applied whenever possible, especially on computer vision and natural language processing tasks.

Transfer learning in deep learning: A prior model is well trained on a large dataset with enough resources. Weights learned in the prior network are used to start the new model. Depending on the problems, different layers may be borrowed, for example lower feature extraction layers.
A very nice example[4] that uses feature extraction layers of another classifier is below.



The new network may be trying to solve a problem similar to but different than the prior problem. If the problems differ too much then transfer learning may not be a good fit. If the problems are alike, than the new network may use some lower layers of the prior network and train some new layers on top. A nice example is taking an image classifier with say 1000 labels, dropping final fully connected layer with softmax and training a new fc layer with softmax with desired number of labels, for example 50.

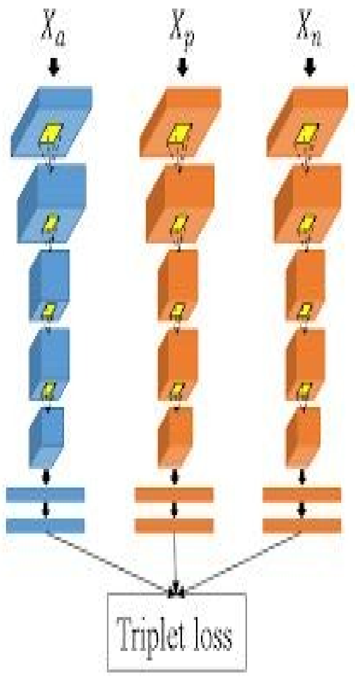
I have used weights from VGGNet for transfer learning. Since it is already trained on faces (ie not on a less related task like object classification) I was able to use all the feature extraction layers. I have used all layers, except last softmax layer, for feature extraction as a face descriptor.

Although the current version worked fine and it did not need any more layers or similar tweaks, for the sake of learning and experiment I have tried to fine tune the last layer. Although the loss dropped, it overfitted and did not perform better on tests. I could not use all of my training data because it took hours to do one epoch, this caused overfitting. The last layer contains approximately 10 million weights and intuitively it can not be trained without tens of thousands of images. I was hopeful about fine tuning but could not produce better results.

I also tried adding a lower dimension layer so that there would be less parameters to train. I added a dense layer of size 1000 effectively doing dimension reduction but again overfitted. Reducing dimension further would cause an information loss so I decided to keep going with the original weights which were already trained on millions of face images as I wanted.

In my experiments I have built a siamese network with triplet loss (triplet network) using keras. I have used VGG architecture as the base network which is described in the paper[Deep Face Recognition]. Triplet loss is not present as a built-in in keras so I have implemented it as a custom loss function.

The Siamese Network with triplet loss



Siamese network is actually multiple networks with the same weights, implemented using multiple forward passes on the same network.
The Siamese Network is good for generating embeddings and one shot learning. It effectively groups same labeled data together in one place and tries to put different labeled data away from each other. In other words what it does is finding a mapping from data points (images) to a space where the same labeled images are clustered together and different clusters are far away from each other.

Triplet network takes three images at once; one anchor image: image of some person (say person 1), a positive image: another image of the same person (person 1 again) and a negative image: an image of

another person (say person 2). All three images are passed through the network and resulting representations of these images are ***a***, ***p*** and ***n***.
We want the distance between ***a*** and ***p*** to be small and distance between ***a*** and ***n*** to be large. So we can define our loss as

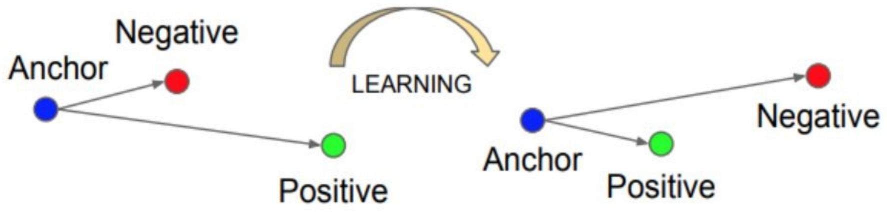
$$L = dist(a-p) - dist(a-n)$$

But this loss drops to zero if all the images are mapped to the same point which is undesirable but can happen in practice. To overcome this we introduce a margin, we want the distance between anchor and negative to be at least “margin amount of” larger than the distance between the anchor and the positive. Then the loss becomes:

$$L = dist(a-p) - dist(a-n) + margin$$

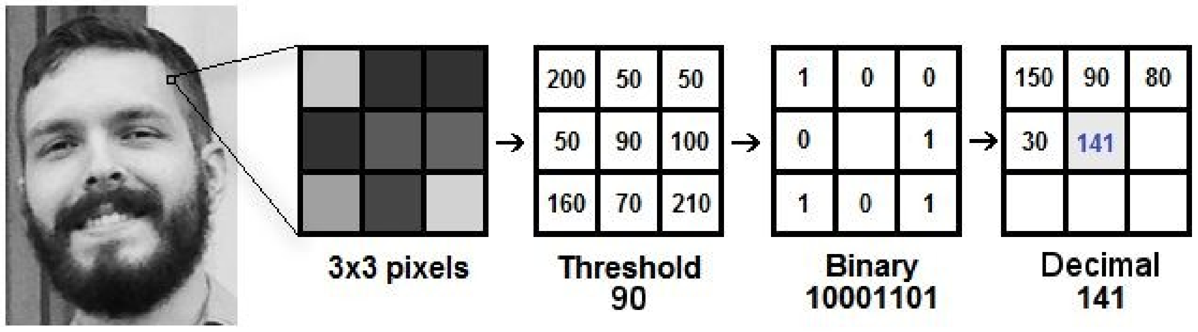
Since this can practically be negative the real loss is ***L*** if it is positive and 0 if otherwise. Note that in that case we may need to change margin or other changes may be needed elsewhere depending on the underlying reason.

What we would like to happen is depicted below[6].



2. LBP based solution:

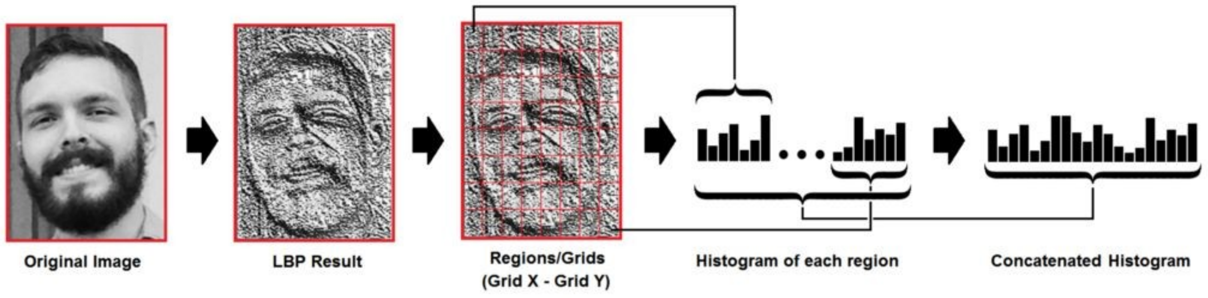
LBP is a visual descriptor which is less affected by changes in illumination by its nature. It is especially good for texture classification. LBP operation is simply comparing each pixel with its neighbours and assigning a 1 if the neighbour is greater and 0 if otherwise. Concatenating these values clockwise (or counterclockwise, does not matter as long as it is consistent) to produce a binary number, this number is assigned to the central pixel. The procedure is better understood with an example[7]:



The resulting image is a very nice descriptor of patterns and very illumination invariant. This process can be extended to larger radiuses. Then it is called extended LBP or circular LBP.

Local Binary Pattern Histograms (LBPH)

are used for comparing LBP images. The LBP transformed image is divided into bins and in each bin there are values between 0 and 256. These values, form a histogram and concatenating these histograms form a feature vector for the original image. It is depicted nicely below[8]:

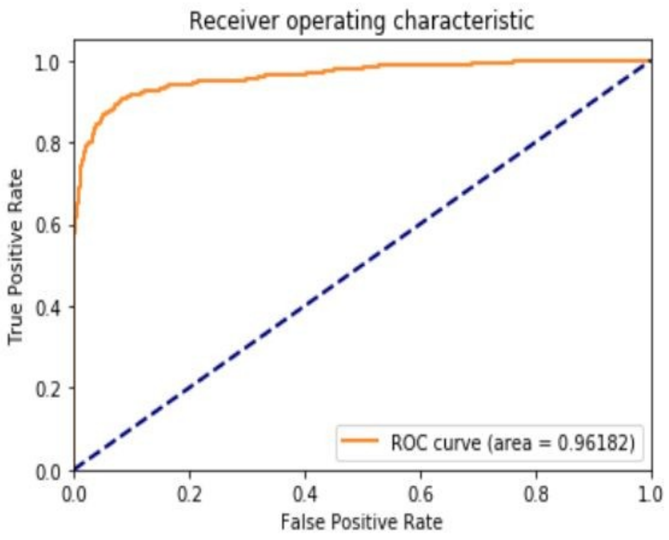


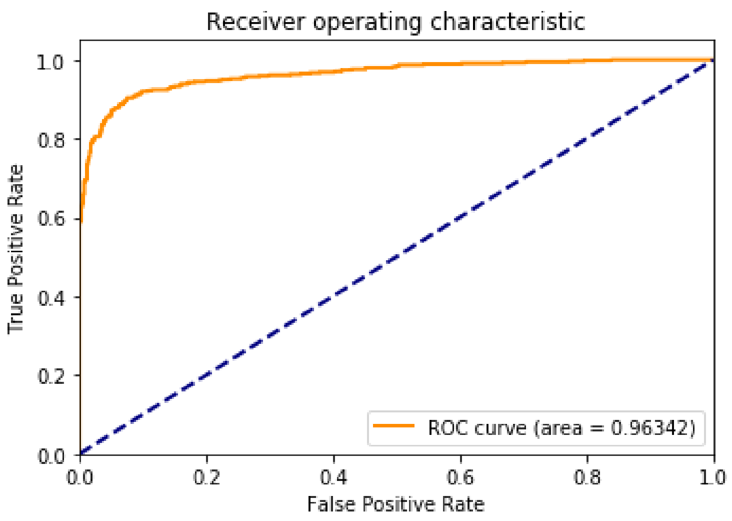
3) Experimental results

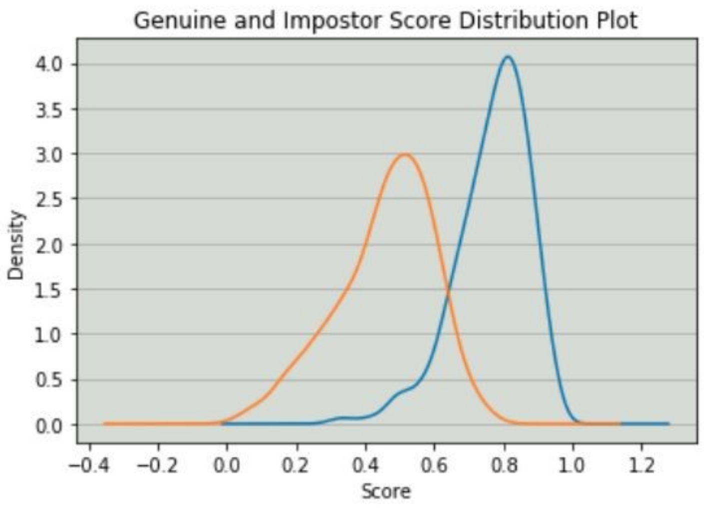
Transfer Learning Method:

As I mentioned before I have tried to fine tune the model. For that I have used about 74k triplets for training and about 1100 images for testing. I have used identification based metrics for comparison such as roc curve and genuine - imposter distribution plots, hoping that these will give me better insight for understanding the systems capabilities. I have tested different configurations but none of my trials resulted in making any noticeable improvement. Generally the model overfitted. Even only the last layer has more than 1 million parameters so training it is not easy. On validation I could not make any improvements. Here are the best score I could get from validation for both the original and modified versions:

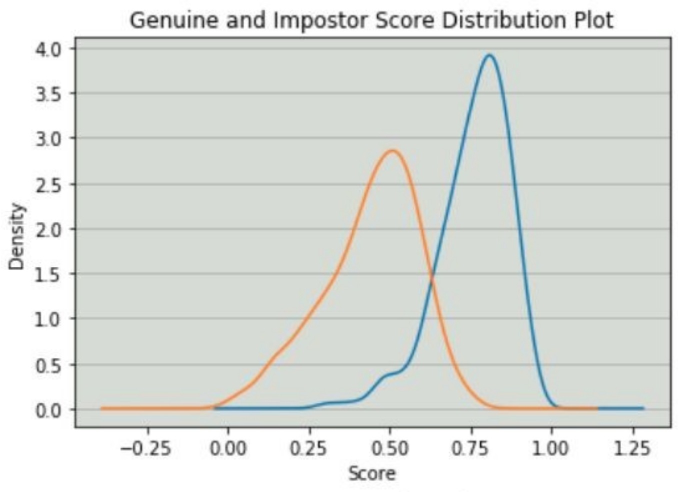
Modified on the left, original on the right







```
eer: 0.08968219416630381, thresh: 0.62541463611
```



```
eer: 0.08968110291045812, thresh: 0.616280162663135
```

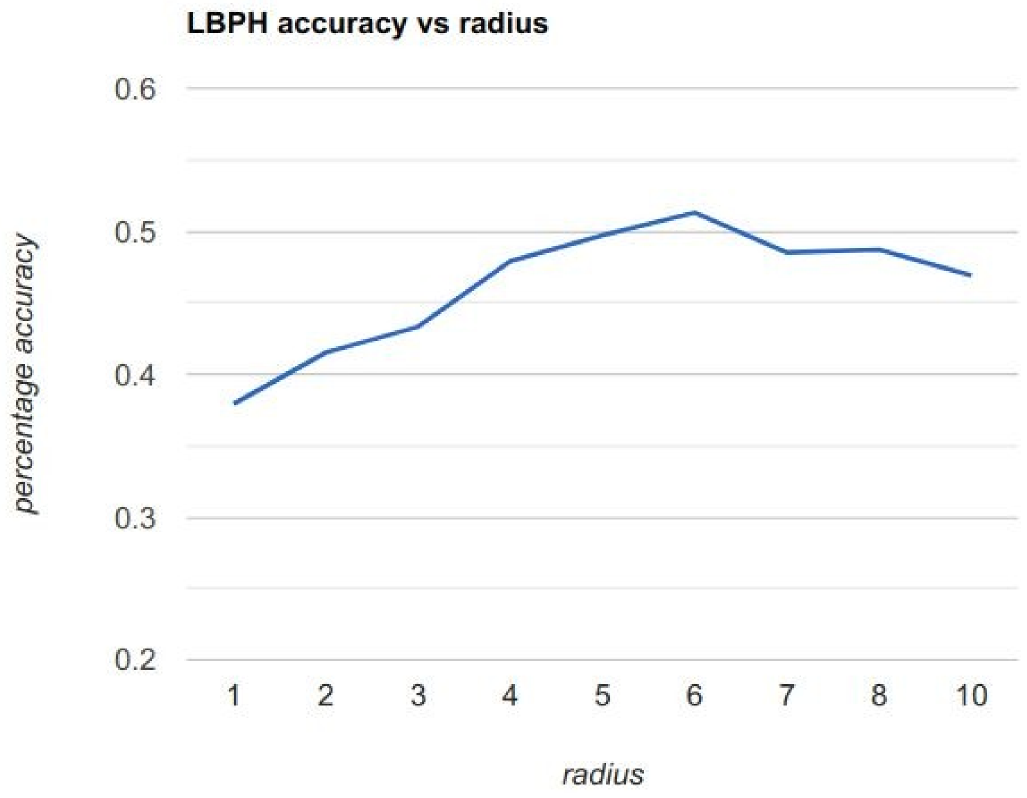
After these experiments I have decided to continue with the original weights.

LBP based method:

For LBPH base solution I have implemented the algorithm in python. But I also used OpenCV' s LBPH recognizer as a benchmark to justify the validity of my results.
In the following test I gave used 1600 images from 254 people as training set, these images are registered and 500 images of the same 254 unique identity are used for test.

My implementation of LBPH with 1-NN resulted 22% accuracy.
It resulted 41% accuray with SVM.

These results, especially the result of 1-NN, seemed low to me. That's why I have used OpenCV's LBPH recognizer to validate my results.
OpenCV LBPH recognizer gave 37% accuracy with radius=1 which is better than my implementation with 1NN but worse than SVM used version. So i believe that my implementation is correct.
I have also tested to see effects of radius: It gets better for a while then decreases as expected.



4) Conclusions

I have seen that neural networks are really good at learning good, discriminative features. Training them is as hard. Even just fine tuning was hard. This is partially because the model was already fine tuned with faces. There was no need to train it for a different task. Yet I had thought I could do a little bit better. Another possible reason is not focusing on choosing hard triplets. Since the network is already well trained, it successfully maps images of different people apart. So the loss is so small that is why gradients are so small too. The model needs extra hard cases to learn further. This is probably the main difficulty of using triplet loss. One solution I have wanted to try was to first find hard triplets which would be computationally expensive but possibly fruitful then finetune the network on those triplets but I could not make it in time. At any rate, the CNN structure is quite powerful and clearly superior.

LBPH is quite fast to train and quite successful in my opinion. It is resistant to changes in illumination to some extent. The output space is fairly large which is not very desirable. One idea is to train a neural network from LBPH output to reduce output dimension. Also note that not every histogram is equally important, using a NN would also provide more effective usage of existing histograms.

5) References

[1] Francis Galton, "Personal identification and description," In Nature, pp. 173-177, June 21, 1888.
[2] Face Recognition: A Novel Multi-Level Taxonomy based Survey (Alireza Sepas-Moghaddam , Fernando Pereira, Paulo Lobato Correia).
[3] Chapter 11: Transfer Learning ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf
[4] Dipanjan Sarkar, 2018
[5] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui and J. Collomosse, "Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask"
[6] FaceNet (Florian Schroff, Dmitry Kalenichenko, James Philbin 2015)
[7][8] Kelvin Salton do Prado (Understanding LBPH algorithm, 2017)
[9] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, Le Song; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 212-220
[10] Wang, Hao et al. "CosFace: Large Margin Cosine Loss for Deep Face Recognition." 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018): 5265-5274.
[11] Z. Xie, P. Jiang and S. Zhang, "Fusion of LBP and HOG using multiple kernel learning for infrared face recognition," 2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS), Wuhan, 2017, pp. 81-84.
Deep Face Recognition, Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman (https://www.robots.ox.ac.uk)
Deep Residual Learning for Image Recognition (Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, 2015)

The Libraries and how I used them:

dlib : for loading images, face detection, cropping and alignment.
cv2 (openface) : for comparing my lbp implementation with.
Sklearn : for train-test splitting and svm classifier
matplotlib : for drawing plots
keras: for building and training the neural network
numpy and pandas for data analysis, (reading csv, making arrays etc).
PIL: I was using this to load images and cropping etc before I switched to dlib.
scipy: for distance testing different distance functions (it is more efficient than pure python)