Face recognition with deep learning for mobile applications

Johannes Brändle [*1], Christian Eppler [†1] and Stefan Möbius [‡1]

[1]University of Applied Sciences, Ravensburg-Weingarten, Germany

October 29, 2015

Abstract

There is a lack of mobile open source applications to search and find selected persons in image libraries. Especially if you do not want to give away your data to cloud based services this paper shows a novel approach to use deep learning to train a feature detector in a desktop environment and integrate the results in an Android application. With our approach all privacy critical processing steps are running only locally and stored only on the mobile device. Also the performance of different neural network models are discussed.

Keywords: deep learning, supervised learning, Caffe, face recognition, CNN, Android, mobile applications

1 Introduction

State of the Art There already exist very good approaches for face recognition on desktop systems. For example Facebook recently developed DeepFace, a nine-layer deep neural network with more than 120 million parameters and an accuracy of 97.35 on the Labeled Faces in the Wild (LFW) dataset [27]. Another work of scientists at the Chinese University of Hong Kong estimate the positions of facial keypoints with three-level convolutional networks [26]. These methods have in common that they use deep learning. This is state of the art in the field of face recognition.

There are a number of frameworks available that can be used to create deep learning networks. Older frameworks like FANN (Fast Artificial Neural Network) and the multilayer perceptron implementation of OpenCV are based on CPU only calculations and lack of new approaches like drop out and rectified linear units [19][7]. In contrast to Google, which can afford using 16000 cores, we have very limited CPU computation power and deep learning requires a lot of it [12]. So these CPU-only based frameworks were no option for us. Therefore we had a closer look on the GPU supported frameworks cuda-convnet2 and Caffe. The framework Caffe, developed by the Berkeley Vision and Learning Center (BVLC) proved as best option for our needs [9]. There are some reasons for this. It supports the NVIDIA cuDNN (GPU Accelerated Deep Learning) library, which emphasizes performance, ease-of-use, and low memory overhead [4]. So the abilities of modern parallel computing hardware can be used without a lot of extra performance tuning effort. Models are defined without coding and can be easily changed. It also has the ability to switch between CPU and GPU, whereas cuda-convnet2 seems to require a NVIDIA GPU [11]. As we want to use the classifier on mobile devices, we need this CPU support. There already exists an Android port with the name caffe-android-lib, which itself is based on a compact version of Caffe named caffe-compact. Models can be trained on GPU and later be deployed to mobile devices. Currently also a lot of researchers in the field of deep neural networks use this framework.

[*]mail@joh-braendle.de
[†]home@silentchris.de
[‡]mail4stefm@gmail.com

2 Theory and Method

We use a deep convolutional neural network architecture which is explained in detail in 2.1. For training we use a computer with a high performance GPU. The classification can be done on mobile devices with the CPU, because it needs less calculation power. Therefore a special approach is needed though, which does not require training on the mobile device.

To run and test different models on desktop and mobile devices, we created a classifier module. This was done in C++, because Caffe itself is written in C++ as well. On the desktop system we use this classifier in a console application to test different models. The same classifier is used in the Android application to do the image classification. As it is written in C++, the Android NDK toolset is required for compilation. For training and testing our neural network we created a dataset from the FaceScrub [16] database as described in 3.2. We selected this dataset as it contains a huge number of images and faces with different poses.

2.1 Design of the neural network

We designed the neural network to have 6 layers with learnable weights. Figure 1 shows the structure of the network. It has an input, 2 convolutional, 3 hidden and an output layer. After each convolution local response

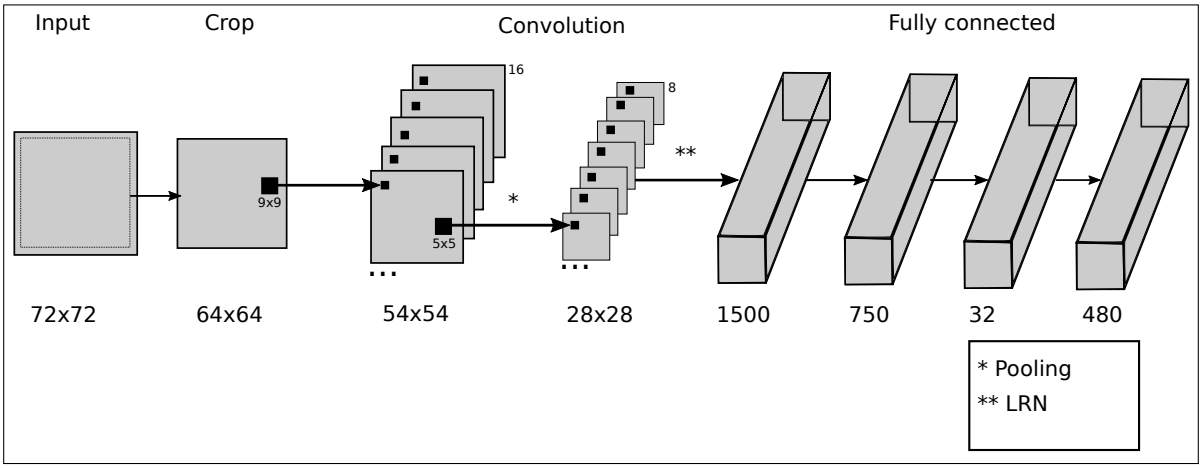


Figure 1: Neural network trained with Caffe (medium size)

normalization and max-pooling[1] are performed. Max-pooling is done in a 2x2 neighborhood. This results in better translation invariance. The first convolutional layer use a 9x9 filter, the second a 5x5 filter. The remaining layers are fully connected. Convolutional layers are utilized because they are successfully used in many image recognition tasks.

For all layers we use dropout to reduce overfitting. We use dropout instead of prelearning (e.g. with denoising auto encoders). This results in a simpler procedure. However more samples are needed for training. All layers use the rectified linear unit as activation function [6]. In contrast to the sigmoid function this helps preventing that the gradient gets near to zero for the upper layers in deep neural networks [13]. The last hidden layer reduces the information to a small amount like 32 float point values. The is done to get more general features which helps to distinguish faces of different untrained persons. We use 72x72 pixel sized colored images which are extracted from the original image by a function provided in the OpenCV library. As input for the neural network we use several cropped parts of this 72x72 sized image with a size of 64x64 pixels in BGR format. For the training phase Caffe uses randomly selected parts of the size 64x64 from the 72x72 image [2]. The advantage of this is that Caffe can produce multiple training patterns out of one image which is an efficient form of data augmentation to pseudo increase the dataset. For the classification we use five different versions of this image by cropping to 64x64 pixels each. These five parts are upper-left, upper-right, lower-left, lower-right and the middle of the image. The results of the neural network are averaged by weighting the middle twice, because this should contain only parts of the face without background and therefore is more important for our task.

2.1.1 Why not compare two faces directly

A much simpler approach than ours would be to use two different faces as input of the network. Then we would require only one output neuron which determines if the two faces show the same person or not. However this was not an option for us because of the performance issues this would cause. With this approach we would need to compare the face which is used as search input with all other images on the mobile device. For each comparison it would be necessary to calculate the result of the neural network. As the neural network can be fairly large this would require much processing power. With our approach the output of the neural network can be precalculated once for each face while the indexing phase. The indexing phase is not as time critical as a search request of the user. After the indexing phase is finished, the mobile device only has to calculate the difference in the already stored output neurons. The number of 32 output neurons in our case is very small, therefore less calculation is needed to compute the difference.

2.2 Proposed Face distance optimization

In this paper we propose a way to make it easier to distinguish between data showing the same and data showing different persons. Therefore we train an additional layer with a different loss function (see formula figure 2). The activations of the last hidden layer are used as input features of the new layer. Normally neural networks use the difference (commonly the L2-norm) between the target and the calculated output values of **one sample** as loss function. We instead use the distance between **two samples** as loss function. The idea is the following: If the samples are from the same person, they show the same face and the distance should be zero. Therefore we can follow the negative gradient to reduce the distance. However in the case the samples are not from the same person the distance should be maximized (to make distinction easier). In this case we can follow the positive gradient to increase the distance. We use the sigmoid function as activation function. This limits the length of the output vector. If the rectified linear unit would be used it could become very large. The used partial derivative of the loss function above is shown in formula figure 3.

$$L = \frac{1}{2}\left\|o^{(t_1)} - o^{(t_2)}\right\|_2^2 = \frac{1}{2}\sum_i \left[Sig\left(\sum_j w_{i,j}\cdot x_j^{(t_1)}\right) - Sig\left(\sum_j w_{i,j}\cdot x_j^{(t_2)}\right)\right]^2$$

Figure 2: Loss function

$$\frac{\partial L}{\partial w_{k,l}} = \left[Sig\left(\sum_j w_{k,j}\cdot x_j^{(t_1)}\right) - Sig\left(\sum_j w_{k,j}\cdot x_j^{(t_2)}\right)\right] \cdot \left[Sig'\left(\sum_j w_{k,j}\cdot x_j^{(t_1)}\right)\cdot x_l^{(t_1)} - Sig'\left(\sum_j w_{k,j}\cdot x_j^{(t_2)}\right)\cdot x_l^{(t_2)}\right]$$

Figure 3: Derivate of the loss function for weight with index k,l

2.3 Optimizations in the Android application

On the mobile device we have less calculation power. For this reason we have to do some performance tweaks in the application. We split the work on the mobile device into an indexing and a search phase. The user initiates the indexing process after the installation of the application is finished. This should be done while the mobile phone has power supply because this task needs time and has huge CPU usage. After this step all results are stored in a SQLite database. Images where no face is found are marked in the database and will not be considered during the search. All images that have not changed do not have to be processed a second time. Only new images and changed images will be processed. When the user initiates a search request only the selected image are processed. All other images are precalculated and do not need to be processed. Finally the distances of the searched image to all indexed images is calculated. One disadvantage of this method is that the indexing process should be started every time new images are placed on the mobile phone.

3 Experiments

We tried neural network models with different sizes for training. Table 1 shows the three different network sizes that we used. It shows the number of neurons for each layer. These network sizes showed to be a good compromise between size and accuracy. The large model has a size of 65 MB, the medium 29 MB and the small 9 MB. The large network is at the border to be feasible for mobile devices (RAM, processing power, disk usage and size of the Android App).

The ROC curve in figure 4 shows the accuracy over the number of iterations. We started with a learning rate of 0.02 and gradually reduced it by multiplying it with a factor of 0.99975 every 200 iterations. After 5m iterations the learning rate is about $4 \cdot 10^{-5}$. As momentum we use 0.1 and $9 \cdot 10^{-7}$ for weight decay. The solver file of Caffe is shown in figure 10.

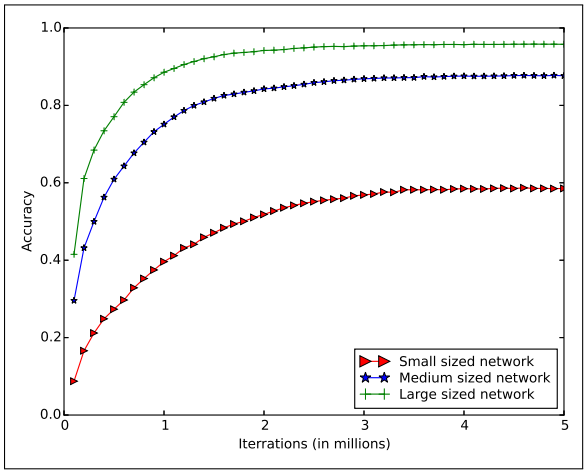


Figure 4: Accuracy over time

Table 1: Sizes of the layers

	Convolutional		Fully connected		
	1	2	1	2	3
Small	16	8	500	250	32
Medium	16	8	1500	750	32
Large	16	16	1800	1000	64

3.1 Different distance metrics

Our goal is to create a classifier which can decide, if two images shows the same or a different person. The approach should work for persons who the neural network was not trained for. Another very important goal is to reduce the amount of work for the mobile device as much as possible. Therefore we decided to do no training at all on the mobile device.

The challenging part is to fulfill these requirements by finding an appropriate neural network architecture. The idea is that the neural network should produce the same (or a very similar) output pattern for pictures of the same person. For different persons the output should differ as much as possible. If this would be the case we could use the quadratic distance of the two output vectors to decide if the pictures show the same or a different person. Our first idea was to train a neural network which classifies a bunch of different persons. Each person is linked to a different output neuron which should have either a activity of 1, if the input data is a picture of this person, or 0 otherwise.

To determine if two pictures of trained persons are from the same person, it is therefore only necessary to check if the index of the neuron with the highest activity is the same for both pictures. However it is not our goal to classify only learned persons. The approach should work with every person. Even for persons who are not born until now. Therefore we had the idea to check if the quadratic distance of the output neurons is smaller than a certain threshold. A person should have more similarities to certain trained persons and less to others. However we could not get the approach working very well. All the activities on the output neurons for a not trained person

seems to be very close to 0. Because of this, the quadratic distance seems to have not much expressiveness. As a new try we used the activity of the last hidden layer of our model trained with Caffe. The last hidden layer was designed to have a much smaller number of neurons than the output layer. Therefore the idea was that the neural network has to learn more general features in this layer. It should not be possible that the last hidden layer contains much special features for a certain person because then other persons could not be classified correctly. Using the activation of the last hidden layer resulted in much better expressiveness.

To further improve the results we decided to train an additional layer which uses the activations of the last hidden layer as input features. This layer is described in detail in 2.2.

3.2 Preparation of sample data

To train and test the neural network we use the FaceScrub dataset [16], which has over 100.000 images of faces. The images show faces with different orientations, not only specific, as biometrical. However we had to do some pre and post-processing of the data. First the dataset consists only of URLs to image files. Some of the images have changed or do have a different resolution now. So we only use images which still have the correct SHA256 checksum.

Normally the dataset is split into two parts. One usually larger part for training and the other for validation. However for our case three parts are needed. Before splitting the dataset in training and validation data, we separated another part from the dataset. This part will later be used to test the capability of the neural net to generalize on completely untrained persons. For this data we also made as less modifications as possible. This is needed to have comparable results. Especially we did not copy or remove images that showed the wrong person or no person at all.

The remaining part of the dataset is used for training and validation on trained data. This validation data is used to test the accuracy for images of trained persons. Note that we do not really need this kind of validation data. However it is used while training the Caffe model. There we use it to check the progress of the training and if the training is working at all. Otherwise it would be necessary to alter the Caffe-Framework for our special case.

For the training data and validation data we did some modification to increase the number of samples. First we cut the images by hand in the way that only the desired face was visible or deleted the image, if it showed another person or no person at all. We increased the number of training samples to about 275.000. Therefore we used the same source image multiple times rotated by a small degree. Also we added small amount of noise, blur, motion-blur and some other image effects. The source code is available in a small toolbox [5].

The faces are automatically cropped out of the images with the Cascade Classification API from OpenCV [21]. The android application later uses the same cropping function so the cut faces will contain the same part of the face as in the training phase. However for the training data the resulting images were checked manually, because sometimes OpenCV detects unwanted parts on the image as faces. Again we did not remove any images from the validation data with untrained persons as this would change the accuracy results.

We also added a new class of images to the training data which do not contain a face. Later this class is used to check if the detected part of an image is really a face.

4 Results

4.1 Application

As result we created a ready to use Android application[1][2] which can be used to search for faces in images with an other image. It is possible to search a person based on a photo in contacts, a photo from the filesystem or a photo taken with the camera from the mobile device. On figure 5 you can see a screenshot of the created Android application.

The application uses OpenCV to detect faces and the Caffe model to recognize faces. All result vectors are saved locally in a SQLite database. The application is designed for an easy adaption of the Caffe model without putting hands on the code. Only a small amount of files (see README of toolbox [5] for details) have to be adapted. The files can be replaced directly on the sdcard for testing purposes. So it is possible to train an own Caffe model [21]. An other possible adaptation is to exchange the xml file of the OpenCV detector [23]. By this change and by using another Caffe model the app can be modified easily to detect completely different objects. This can be done without changing a single line of code. For example detecting flowers, cars etc.

4.2 Analysis of learned features in first convolutional layer

The images figure 7 and figure 8 show the weights (bias unit is omitted here) and the resulting maps for a sample image. The weights can have positive or negative values. Here a gray pixel means a weight of zero. Positive weights are brighter, negative weights are darker. The images of the weights are nearly grayscale. This means that the color seems to have a marginal influence. Some training images are grayscale. This could result in more grayscale like weights. The weights are mostly bars with different orientations. However horizontal structures

[1] https://play.google.com/store/apps/details?id=de.informatikprojekt.searchmyfriends

[2] https://www.youtube.com/watch?v=wpLRmk19AmI

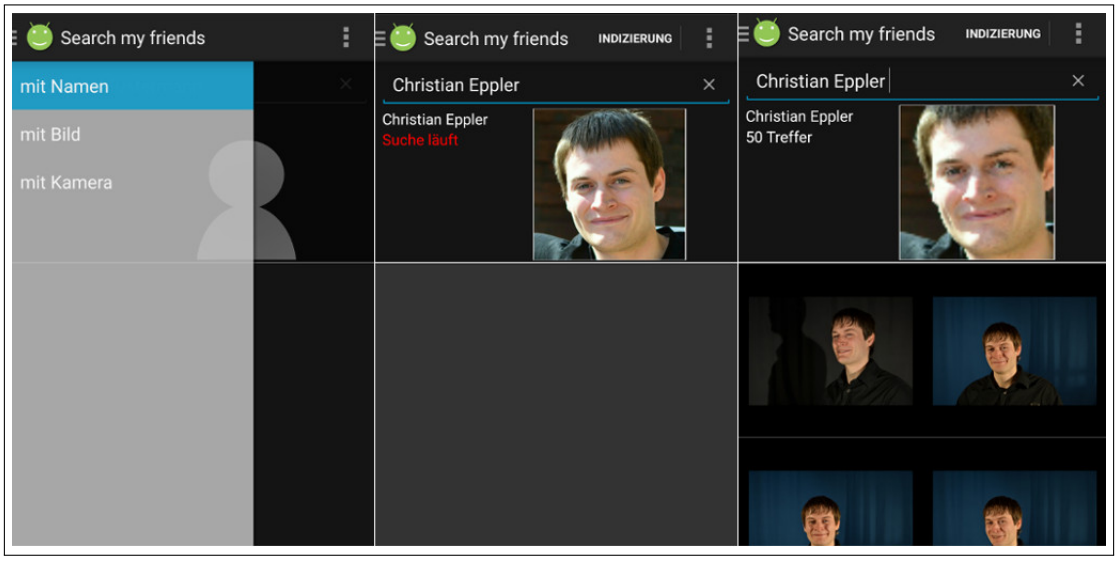


Figure 5: Application

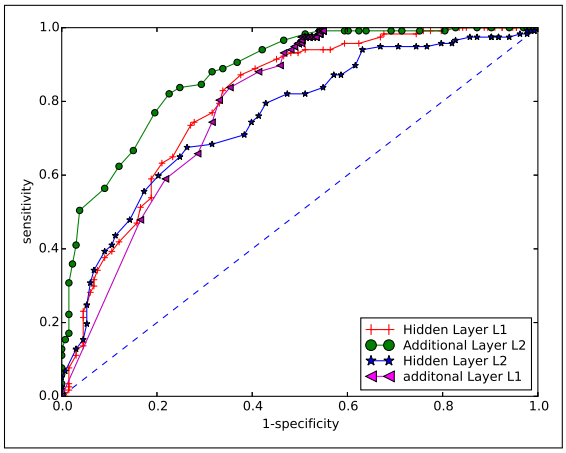


Figure 6: Performance

seem to be preferred. On figure 8 a black pixel means zero activation. The brighter the pixel in the figure the more the neuron is active (negative activations are not possible). One particular interesting result is that the 11th filter does have weights close to zero. This resulting map has also zero activation over the whole image. As the bias unit is also zero this neuron seems to fire never. However we have no explanation for this behavior.

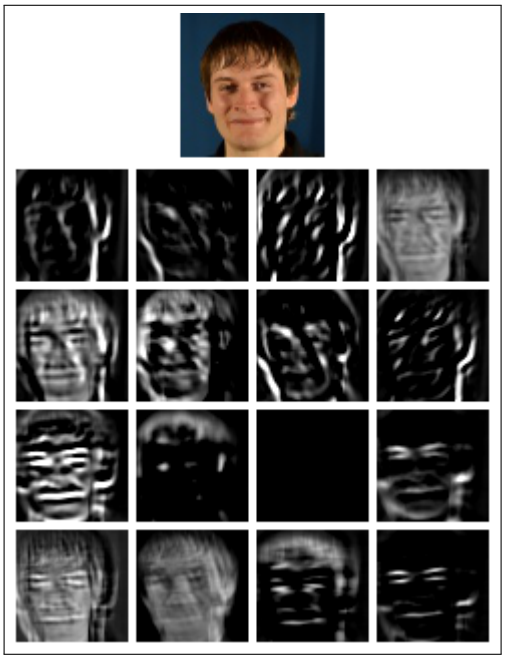


Figure 8: Resulting maps for an example image

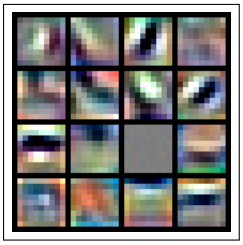


Figure 7: Weights of the first convolutional layer

4.3 Comparison classification of different layers

In the following we show a projection of the activations of different layers in the neural network. We use Sammon's mapping to project the high dimensional data to 2 dimensions, so that the results are human readable. As with PCA the distances between different samples are approximately the same. However, in contrast to PCA Sammon's mapping [24] is a nonlinear mapping.
Figure 9a shows the Sammon's mapping of the raw input images. As expected the plot shows no visible clusters for images of the same person. The reason for this is that the raw pixel values can be quite different although the same person is showed (e.g. different brightness level or orientation of the face).
Figure 9b shows the Sammon's mapping of the activations of the output layer learned with the Caffe framework. It shows visible clusters. However some clusters are overlapping.
Figure 9c shows Sammon's mapping of the activations of the last hidden layer learned with the Caffe framework. It also shows visible, but also some overlapping clusters. Further the clusters are close to each other. This means that the learned neural network cannot distinguish very well between them in these cases.
Figure 9d shows Sammon's mapping of the activations of our additional layer. In contrast to the Sammon's mapping of the output- or hidden layer the distances between the clusters are bigger in some cases. This helps to distinguish between the clusters more easily. However especially in the case where the clusters are overlapping this results in no improvement. As the additional layer is using the sigmoid function as activation function (in contrast to rectified linear unit for the hidden layer) the results seem to have less outlier. A particular interesting result in this graph is that all male persons are in the lower half and all females are in the upper half. This is shown in figure 9e. Therefore we can conclude that our model is able to separate female and male persons very well.

Table 2: Accuracy after 5m iterations

	Hidden		Extra	
	L1	L2	L1	L2
Small	0.73	0.70	0.75	0.76
Medium	0.72	0.55	0.79	0.72
Large	0.56	0.53	0.74	0.78

We could show that our method is working fairly well as shown in table 2. With the best model we got an accuracy of 0.79 for checking if two images show the same person. The calculation of the accuracy is done with faces which do not occur in the training dataset. The task to learn suitable features is not easy, because the neural network has to learn to generalize. Compared to other publicized approaches our method does not have the best accuracy. However our goal was not to reach the best accuracy. Instead we were looking for a method which works well on mobile devices. Therefore we had to accept some compromises as described in 2.1.
We tried to normalize the output values. However this did not give much better results.

4.4 Related work

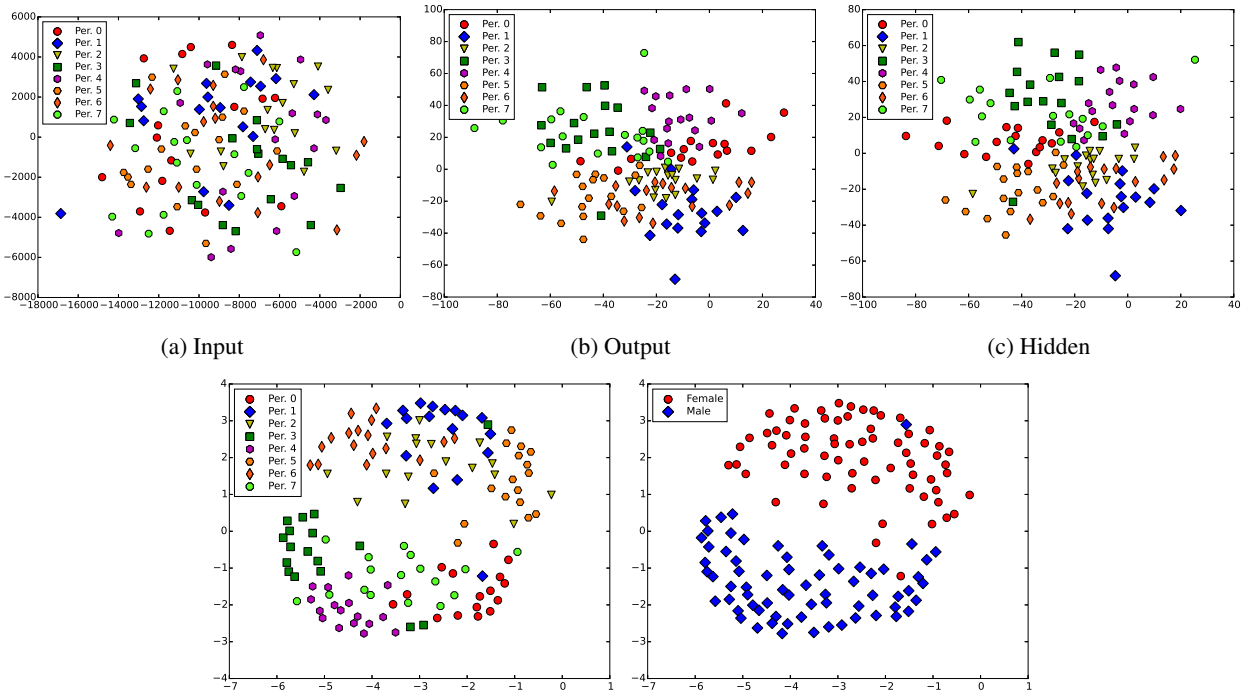
Table 3 shows a comparison of the accuracy with another convolutional neural network from [8]. The results are not calculated with the same dataset. Therefore the significance of this comparison should be treated with caution. However the images are very similar to the images in the dataset[16] that we used. Therefore we can conclude that the results should not differ too much. The other paper uses the LFW faces DB [25]. It could be possible future work to calculate the accuracy on the LFW dataset with our approach.

Table 3: Accuracy comparison with results from [8]

	Own	Other
Small	0.75	0.7828 ± 0.0046
Medium	0.79	0.7882 ± 0.0037
Large	0.74	0.7807 ± 0.0035

5 Conclusions

We have seen that the medium size neural network gives the best results for unlearned persons. It is surprising that it shows to have a better performance than the largest network. The results for unlearned persons are shown in table 2. On the other hand the results for learned person in figure 6 show other results.
We have shown that small neural networks can work fairly well even with a very small number of weights. On our experiments we have shown that the medium size network with a size of approximately 29MB gives the best results. This are good news for mobile use because less calculation time is needed. However it is still required to spend a lot of time on the PC to train the network for a long time to get good results.
We have split the work on the mobile device in an indexing and an search phase. The indexing phase can be run on the phone for example in the night when it is connected to a power supply. While the indexing phase all in advance calculable data is stored in a SQLite database. Out of this reason the user search request can be execute very quick without waiting too long, which is very crucial for the acceptance of the app.
Another alternative promising approach for a better accuracy for the classification of different faces would be to use clustering to get only one label for each person instead of 32 float values. But with this approach the clustering should be calculated every time we search for a new person to get the best results. But this can also result in far more calculation time.



(a) Input (b) Output (c) Hidden

(d) Additional (e) Additional Female/Male

Figure 9: Sammon's mapping

We didn't use a third convolutional layer because our first approach had fewer input neurons. For this network a third convolutional layer was not needed. Later we did not want to change our design from the ground because the it was already working fairly well. Adding an additional convolutional layer would require to balance the network structure again. It is a possible future task to train a network with 3 convolutional layers. Another paper indicates that for our input size it could be useful to add a third convolutional layer [8].

In this paper we proposed a new approach to learn a metric for face distances. We could show that our approach increases the distances for faces of different persons. Also our approach is very good in distinguishing between female and male persons. Another main advantage of our approach is that we running all privacy critical processing only locally on the mobile device. This aspect is more important if we have a look on the latest news of hacked cloud services and the patriot act law. Especially in the field of face recognition it can be a serious privacy issue if the data falls into the wrong hands.

6 Acknowledgments

We would like to thank Prof. Brümmer for his gracious support and feedback. We also thank Matthias Bernhard for providing a Samsung Galaxy S4 for development and testing.

```
net: "/PATH/train_val.prototxt"
test_iter: 1000
test_interval: 1000
base_lr: 0.02
lr_policy: "step"
gamma: 0.99975
stepsize: 200
display: 20
max_iter: 5000000
momentum: 0.100
weight_decay: 0.0000009
snapshot: 50000
snapshot_prefix: "/PATH/caffenet_train"
solver_mode: GPU
```

Figure 10: Caffe solver configuration file

References

[1] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 111–118, 2010.

[2] caffe.berkeleyvision.org. Layers. `http://caffe.berkeleyvision.org/tutorial/layers.html` [31.07.2015].

[3] Yuheng Chen. Caffe compact. `https://github.com/chyh1990/caffe-compact` [29.07.2015].

[4] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, and John Tran. cudnn: Efficient primitives for deep learning. `http://arxiv.org/pdf/1410.0759v3.pdf` [29.07.2015].

[5] Johannes Brändle Christian Eppler, Stefan Möbius. Toolbox for dataset. `http://informatikprojekt.de/uploads/searchMyFriends_toolbox_v0.1.tar` [19.08.2015].

[6] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[7] FANN. Fast artificial neural network library. `http://leenissen.dk/fann/wp/` [29.07.2015].

[8] 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 preprint arXiv:1504.02351*, 2015.

[9] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[10] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[11] Alexander Krizhevsky. cuda-convnet2. `https://code.google.com/p/cuda-convnet2/wiki/Compiling` [29.07.2015].

[12] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[14] Alexis Mignon and Frédéric Jurie. Pcca: A new approach for distance learning from sparse pairwise constraints. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2666–2672. IEEE, 2012.

[15] Jawad Nagi, Frederick Ducatelle, Gianni Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, Luca Maria Gambardella, et al. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*, pages 342–347. IEEE, 2011.

[16] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 343–347. IEEE, 2014.

[17] Nvidia. cudnn. `https://developer.nvidia.com/cudnn` [29.07.2015].

[18] Nvidia. https://github.com/sh1r0/caffe-android-lib. `https://github.com/sh1r0/caffe-android` [29.07.2015].

[19] OpenCV. Opencv mlp. `https://github.com/Itseez/opencv/blob/master/modules/ml/src/ann_mlp.cpp`[29.07.2015].

[20] OpenCV. Opencv mlp. `http://docs.opencv.org/modules/ml/doc/neural_networks.html`[29.07.2015].

[21] opencv.org. Cascade classifier training. `http://docs.opencv.org/doc/user_guide/ug_traincascade.html` [31.07.2015].

[22] opencv.org. opencv face recognition. `http://docs.opencv.org/2.4/modules/contrib/doc/facerec/index.html` [29.07.2015].

[23] opencv.org. opencv object recognition. `https://github.com/Itseez/opencv/tree/master/data/` [31.07.2015].

[24] John W Sammon. A nonlinear mapping for data structure analysis, 1969.

[25] Conrad Sanderson and Brian C Lovell. Multi-region probabilistic histograms for robust and scalable identity inference. In *Advances in Biometrics*, pages 199–208. Springer, 2009.

[26] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep convolutional network cascade for facial point detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3476–3483. IEEE, 2013.

[27] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lars Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.