Fixed layer Convolutional Neural Network

Alexandros Kyrloglou - s1712683
Supervisors:
dr. C.G. Zeinstra
dr.ir. L.J. Spreeuwers
dr.ir. A.J. Annema

Saturday 30th June, 2018

***Abstract*— Fully trained convolutional neural networks are being used nowadays in various applications. But, can we get an understanding on how they actually work? A small step in that direction is taken in this paper. The filters of the first layers seem to be fairly straightforward and mostly are known and recognizable. If one makes a network with fixed weights on the filters, how is its performance compared with a fully trained one? And how is the training time influenced? This paper, answers this question by experimenting in a verification style CNN put in three different situations: first a fixed layer network, second a set layer network and third a fully trained CNN. It is shown that although similar results can be achieved with the three networks, a fully trained one still has superior performance; however, training time is increased as the number of fixed layers are increased.**

I. Introduction

At present times, biometric recognition, like face recognition, is being used as the main security barrier in various projects, making the need for it to be as fast and accurate as possible obvious. The state of the art methods are using a trained convolutional neural network (CNN) on a huge data set in order to make it efficient and precise. The problem introduced, is that the output of the training is not something that can be analyzed and its workings understood, but has to be taken as always true. As this is a really important piece in the security process, understanding how it actually works is crucial. In this paper a step towards that will be attempted.

If one looks at the first layers of a trained CNN a lot of the times very common filters can be recognized. Usually, these layers are made by filters like Gaussian, Gabor, edge detection or other low order filters. Observing this, the idea of replacing the trained filters with ones that can easily be designed comes to mind. But, *how much will this affect the performance of the network? And how does this affect the training time?* These are the questions raised in this paper and in order for them to be answered the performance and training time of such a network will be compared with a more classical fully trained CNN.

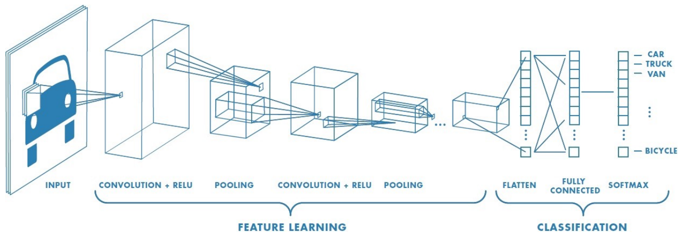

Fig. 1. Common architecture of CNN's. [10]

II. Background Theory

In this part of the paper a basic introduction in the workings of CNN will be made. For a more detailed introduction the lectures of the CS231n course from Stanford University on Convolutional Neural Networks, provide a clear explanation of convolutional networks and practice [7].

A. Architecture of CNN's

In this paper the simplest form of CNN's is used, meaning sequential models. These networks are an assembly of multiple layers put one after the other. The input of the network is put at the start and a form of classification is the output of the network. The most common layers that these networks are made of are 2D convolution, max-pooling and dense or fully connected layers.

1) 2D-Convolution layer: In a convolution layer a set of small filters, usually 3x3 to 5x5, are set. Each one of these filters is convolved with the input and the outputs are set as the channels of the output feature map, which in turn is the output of the layer. Other than the weights, there are multiple parameters that need to be set in these layers like the padding, size of filters and the stride. These will all affect the output of the layer.

2) Max-pooling layer: In a CNN max-pooling layers are used to provide a form of rotation and translation invariance. It reduces the dimensionality of the intermediate layers and therefore the number of parameters to be trained. Generally, other types of pooling can also be used but the most common

This work was supported by the Faculty of Electrical Engineering, Mathematics & Computer Science (EWI), Datamanagement & Biometrics (DMB) University of Twente and special thanks to:
dr. C.G. Zeinstra (Chris) is with Faculty of Electrical Engineering, Mathematics & Computer Science (EWI), Datamanagement & Biometrics (DMB), University of Twente, 7500 AE Enschede, The Netherlands
dr.ir. L.J. Spreeuwers (Luuk) is with Faculty of Electrical Engineering, Mathematics & Computer Science (EWI), Datamanagement & Biometrics (DMB) University of Twente, 7500 AE Enschede, The Netherlands
ing. G.J. Laanstra (Geert Jan) is with Faculty of Electrical Engineering, Mathematics & Computer Science (EWI), Datamanagement & Biometrics (DMB) University of Twente, 7500 AE Enschede, The Netherlands

one is max pooling, meaning a layer that down-samples the input taking the max from a select region, usually 2x2 to 4x4.

3) Flatten layer: Flatten layers are quite simplistic and are used in order to extract a feature vector from the output of the other layers. These layers stack all pixels of all the channels in a vector and present them as the output.

4) Fully connected / Dense layer: After multiple convolution and max-pooling layers, in order to be able to learn non-linear combinations of the features, one or more fully-connected layers are added. In these layers all of the inputs are connected to all of the outputs. The last layer of a CNN is also usually such a layer but with a different activation function. Depending on the classification that the network has to perform, this activation function is set. Most common activation functions are sigmoid or soft-max. As last layers of a CNN, other type of classifiers can be added, for example more classical classifiers like random forest or nearest neighbor.

5) Training: When the structure of the CNN is made, a set of images is passed through it and the weights of all the filters are changed in order for the input to be classified correctly. This is done for a varied amount of times, called epochs and the number of images per epoch is the batch size.

B. Common Structure

The most common structure for a CNN is to have multiple pairs of convolution and max-pooling layers at the start followed by a dense layer and some activation function like soft-max (figure 1). These networks are usually used for multi-class classification or binary classification. This means that the network is trained in order to get to a result either by saying which object was the input or identifying the image as one of two classes.

C. Verification CNN's

A CNN tuned for verification has a slightly different structure than that of a classical one. In this setting the network has an input of two stacked images, or two inputs, and is trained towards a single output which is the verification score. This network structure can be seen in figure 2. The verification score identifies whether the two images are of the same or different type.

III. Method

Here, a model is created using python, and more specifically Keras with Back-end Theano. Using these tools a verification style network is created and different experiments are run on it. The tutorial that was mostly followed to create these experiments was [5]; further documentation can be found in [4]. The code that was used in this paper can be found in VII-A.

A. Experimental setup

The goal of this work is to research the effect of setting the filters of a CNN, in training time and performance. In order to do this three experiments are created. However first the experimental setup needs to be explained. Figure 3 shows this setup. This network has the structure explained in section II-C; more specifically it is composed by four 2D-convolution, four max-pooling, one flatten and one Dense with a soft-max activation layers. This size of network was chosen in order to have a fairly simple experimental setup that can be easily created and reproduced. A more complicated setup would require substantially more time and effort in order to be correctly trained. On the other hand, a much simpler network could give results that might not be directly related to real life uses of CNN's where the network structures are far from simplistic. The number of filters per layer was chosen to be as such, 20-40-60-80 for each layer respectively, in order to be able to compare the results of this work with the results shown by F.H.J. Hillerström [1]. Her paper also was on similar topics and showed the difference between different types of CNNs and the effect of the training dataset size.

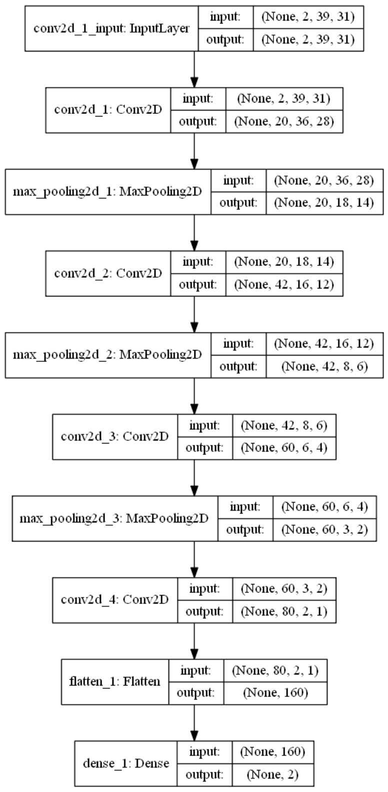

Fig. 3. Network architecture used for all three experiments in this paper. Size of all inputs and outputs of the layers are shown.

The experiments made on these networks involve setting the weights of some of the filters of the network. For this paper only the weights of the first two convolution layers are set. This is done for various reasons. First, by observing the fact that starting layers of CNN's usually have recognizable and simplistic filters, setting these is quite straightforward. Concurrently the last layer's filters are usually higher order, which would make designing filters for these troublesome. Secondly, in most applications of CNNs the first layers stay approximately the same whereas the last layers are very dependent on the use of the CNN. Finally, the number of

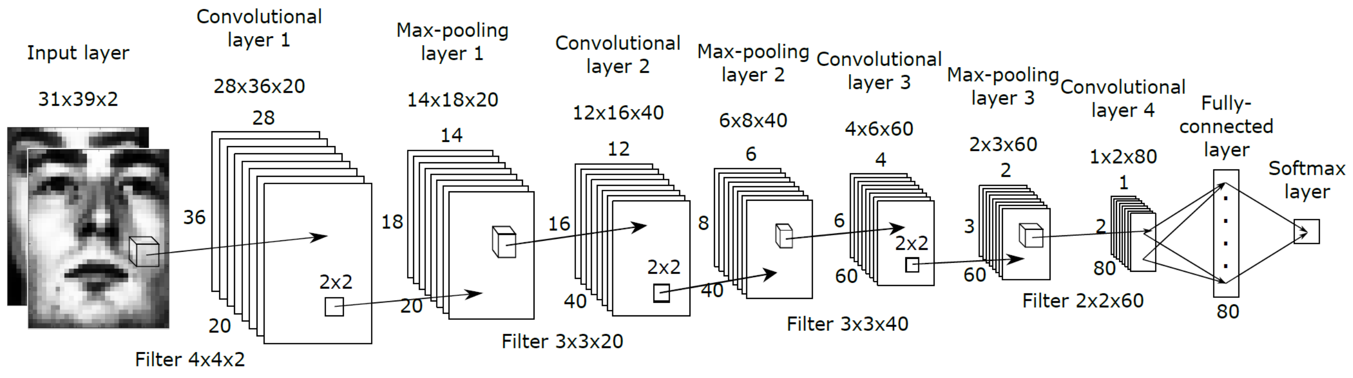

Fig. 2. Deep Verification Learning convolutional network. At the input two gray-scale images are presented, each in a separate channel. The network is directly trained towards a verification score. Network based on the topology proposed by Sun et al. [13] and worked on by F.H.J. Hillerström [1]. Face images are preprocessed images from the FRGC dataset

filters per layer highly increases from layer to layer making it very hard to design the amount of filters needed for the last layers, for instance in the network used the first layer had 20 filters while the last had 80 filters.

B. Experiments

Three different experiments are done each with different initial weights on the first layers of the same network. From these, metrics to measure the performance and time to train will be saved.

Ex. A: Fixed filter CNN: In this experiment the first and second layers' filters are fixed. The filters set in these layers can be seen in figures 4 and 6. After these filters are set, the rest of the network, layer 3 and 4, are normally trained while the first two layers remain intact throughout the training (they are fixed layers).

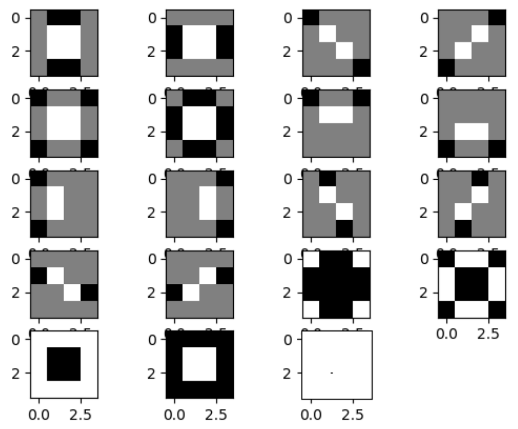

Fig. 4. First layer of *experiment A*: Fixed filters, edge detection filters. In this image numbers from -1 to 1 are mapped to colors from white to black, zero being gray one being white and -1 being black.

In the first layer mostly edge detection filters are used; this is done to extract important features of the face like the shape of the eyes, nose and mouth. Other important characteristics of a person like the placement of eyes or nose or the existence of unique features like moles or scars can also be found by using edge detection filters. Two of the channels keep the input intact to the next layer. In figure 5 an example of the output of the first layer is shown.

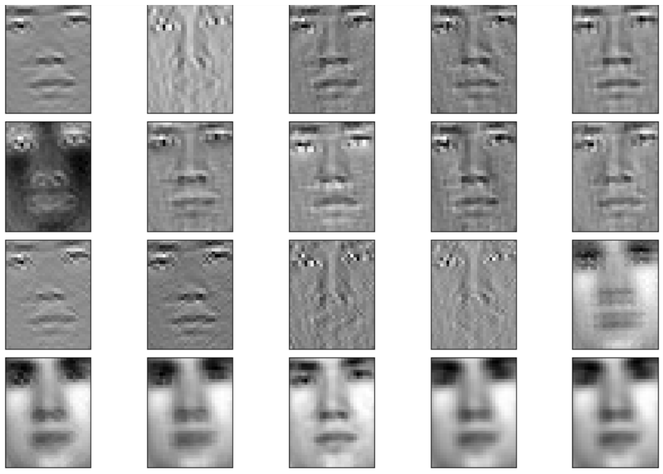

Fig. 5. Example of input image passed through filters of the first layer in *experiment A*. In this image numbers from -1 to 1 are mapped to colors from white to black, zero being gray one being white and -1 being black.

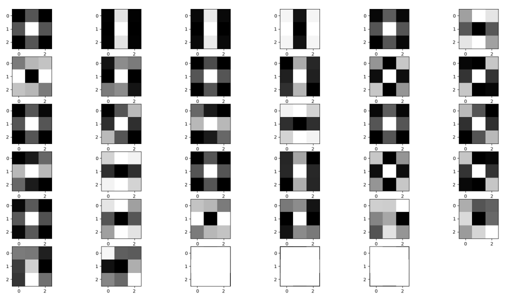

Fig. 6. Second layer of *experiment A:* Fixed filters, Gabor filters. In this image numbers from -1 to 1 are mapped to colors from white to black, zero being gray one being white and -1 being black.

In the second layer Gabor filters are mostly used. Additionally some Gaussian filters are added and some of the filters are again set to pass the input as is. Gabor filters are used in order to get the texture of the skin or the edges - [8]. The code used to create these filters is based on in [9]

To show how complicated the filters of the final layers can be, figure 7 shows only the third layer of the network and still the filters are quite complicated even though being only 3x3.

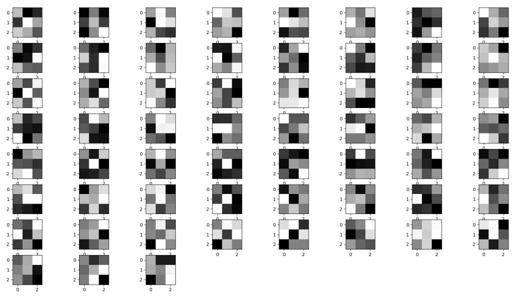

Fig. 7. Filters of the 3rd layer, *experiment A*, after training the network. In this image numbers from -1 to 1 are mapped to colors from white to black, zero being gray one being white and -1 being black.

Ex. B Set filter CNN: In this experiment the first and second layers are set and then trained. When set the same filter weights are used as in experiment *A* and then all four layer weights are left to freely train on the dataset. Comparing figure 8,to figure 4 shows easily that the majority of the structure of the filters are kept the same with only fine-tuning.

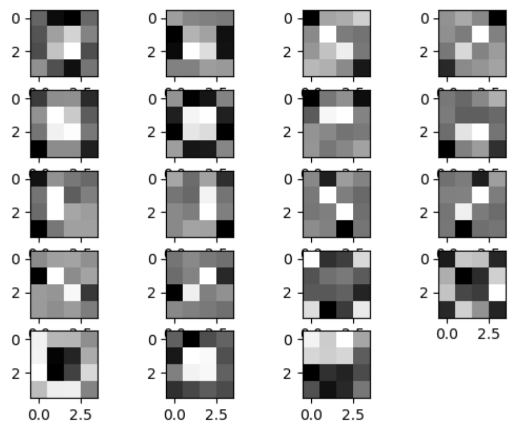

Fig. 8. First layer of *experiment B:* set layers, edge detection filter after training. In this image numbers from -1 to 1 are mapped to colors from white to black, zero being gray one being white and -1 being black.

Ex. C Random filters CNN: In the last experiment, the weights of all four layers are set randomly, as usually done in most CNNs; after that all the weights are changed during the training procedure.

C. Data & Preparation

The training of these networks is done on images taken from the FRGC dataset [11] and more specifically the FRGC_v2.0 from fall 2003, spring 2003 and spring 2004. In this dataset pictures are taken from the same people in these times; this dataset has pictures with various facial expressions and backgrounds. For these experiments only pictures in controlled environment and neutral expressions are chosen. After this selection the face of the each person is cropped from the image using the function crop-face [12]. This function uses the positions of the eyes and the output size in order to get the face of the person. From this dataset, pairs of images are made either with the same person or two different ones. The total number of pairs used in the training is 5659, this size of dataset is chosen in order to get a network that is trained on a medium to small dataset; this is where setting the filters would make most sense as, if a very large amount of data is provided, there would be no need to lessen the training or to speed the process. The number of ”same” classified images is slightly bigger than that of the ”different” ones being 52% of the dataset. This set of images is split in two parts: a training set and a testing one. The training set in 80% of the whole, while the testing set is 20%. This split is done in a way that ensures the fact that there are around 50% ”same” and 50% ”different” pairs in both the training and the testing set.

D. Expectations

By doing these experiments and looking into the performance of the out-coming networks as well as the time and number of epochs they need to train, some conclusions can be drawn. The expectations for these are that the performance of experiment C will be best followed by B and then by A. This is due to the fact that the third experiment has more freedom to train on the specific dataset and get the best results as it has more weights to train. However, this means that it is also expected to take a longer training time.

E. Performance Metrics

In order to test the workings of a CNN, a number of metrics are used. The most common ones are accuracy, loss, receiver operating characteristic(ROC curve) and the Area under the ROC curve.

Accuracy of a CNN is the percentage of times it classifies an image, or in this case a pair, correctly. This is calculated throughout the training of the CNN and presented in a graph. The accuracy of both the training and the testing dataset is calculated. Naturally the accuracy of the training dataset is going to be much higher as the network is trying to classify images that it has been directly trained on. When training the network, its accuracy needs to be maximized for both the training and the validation dataset.

Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets. Loss shows how well the network is doing in recognizing images from the specific datasets, testing/training. This is a general idea what loss is, as the more detailed mathematical explanation is out of the scope of this paper. Of most importance is the fact that the loss needs to be kept as low as possible during the training.

When training and choosing when to stop the training, the loss and accuracy are mostly taken into account as the best time to stop is when the loss is as low as possible and accuracy is maximized. -[3]

The ROC curve shows how well a binary classifier can distinguish between the two choices. In a classifier there will always be false positives and false negatives. A threshold needs to be decided on, in order to choose whether an image will be classified as ”same” or ”different”. A ROC curve is made by varying this threshold and plotting the true positive rate against the false positive rate. For this plot a completely random classifier would get a line through the middle of the graph. Generally, the further towards the top left the graph is, the better the CNN. This is easily shown with the AUC of the graph which is a calculation of the area under the ROC curve - so the higher the AUC the better the CNN. The AUC is an easy way to tell which CNNs are better at classifying a dataset, as the higher it is the better the CNN with a natural maximum of 1. -[2]

IV. Experimental data and results

Ex. A: Fixed filter CNN

Here the results of *experiment A*, fixed 1st and 2nd layer CNN, are shown. In figure 9 the accuracy of this network for the 500 epochs is shown. Figure 10 depicts the loss of the same network. The optimal time to stop the training is around the 150th epoch to be certain that the accuracy is at its highest while the loss is kept low. The lowest ROC curve of figure 15 is taken from this network at that epoch. The AUC of that curve is approximately 0.9599.

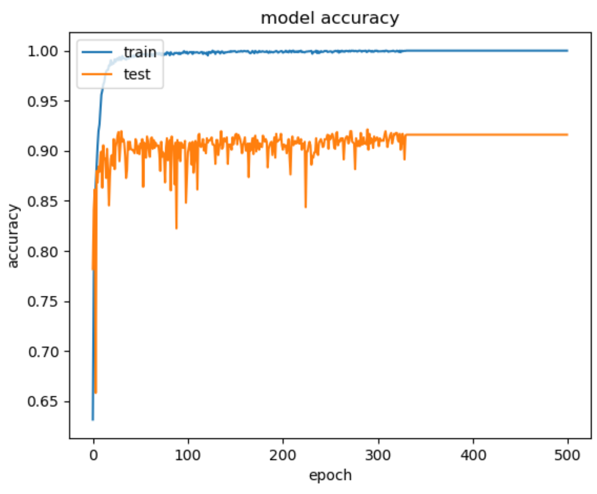

Fig. 9. Accuracy of *Experiment A:* Trained Fixed layer CNN

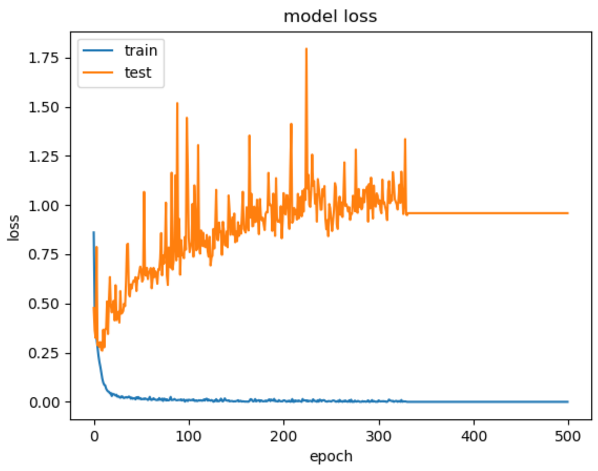

Fig. 10. Loss of *Experiment A:* Trained Fixed layer CNN

Ex. B Set filter CNN

Here the results of *experiment B*: set weights of 1st and 2nd layer and then fully train the CNN, are shown. In figure 11 the accuracy of this network for the 500 epochs is shown and figure 12 depicts the loss of the same network. The optimal time to stop the training is around the 75th epoch to be certain that the accuracy is at its highest while the loss is kept low. The ROC curve of figure 15 is taken from that epoch, and in that plot it is the middle curve. The AUC in this case is 0.9641

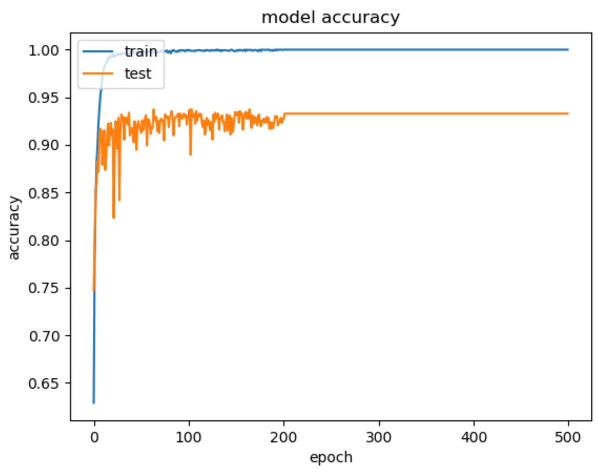

Fig. 11. Accuracy of *Experiment B:* Trained set layer CNN

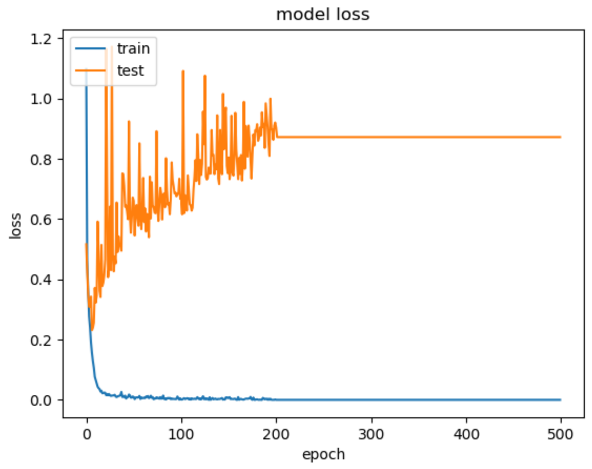

Fig. 12. Loss of *Experiment B:* Trained set layer CNN

Ex. C Random filters CNN

Finally here the results of *experiment C*, randomly set filters and then fully trained CNN, are shown. In figure 13 the accuracy of this network for the 500 epochs is shown and figure 14 shows the loss of the same network. The optimal time to stop the training is probably around the 50th epoch to be more certain that the accuracy is at its highest while the loss is kept low. The ROC curve of figure 15 is taken from that epoch, and in that plot is the top curve. The AUC here is 0.9747

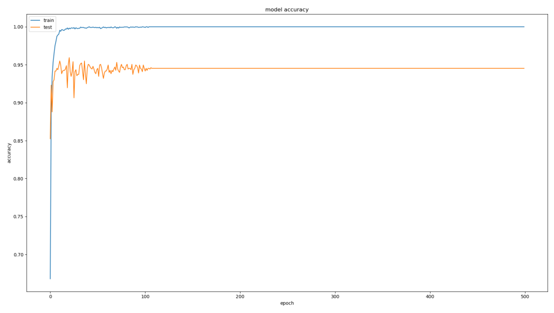

Fig. 13. Accuracy of *Experiment C:* Fully trained CNN

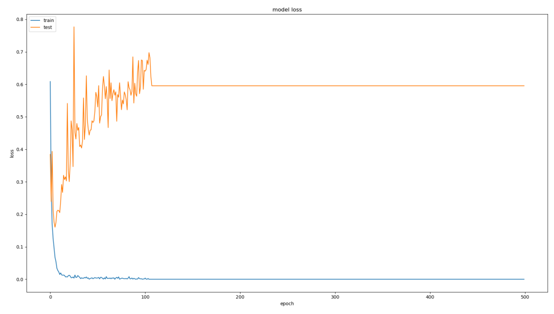

Fig. 14. Loss of *Experiment C:* Fully trained layer CNN

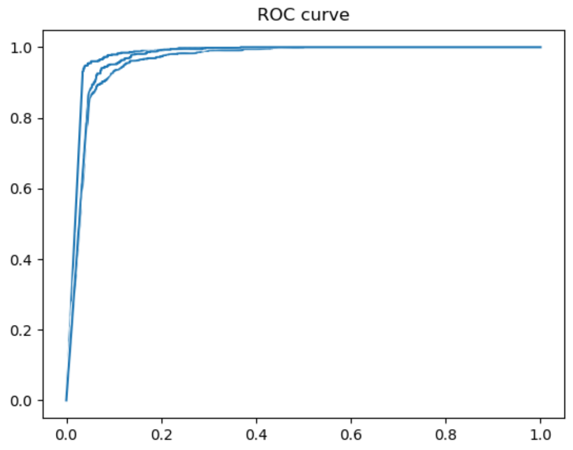

Fig. 15. ROC curve of all three experiments. Top one is *Experiment C:* fully trained, middle one *Experiment B:* is set layer and bottom is *Experiment A:* fixed layer CNN

V. Discussion

From the results shown in chapter III-E, it is easily seen that the fully trained CNN, *experiment C*, gives the best classification of the dataset when compared with the other two. At the same time, it takes less epochs to train and has superior performance. This can easily be recognized by comparing the AUC of the three experiments and the stop epoch for each one. Namely the difference in AUC between the first and third experiment is 0.0148 and it takes approximately three times longer to train.

However even the fixed layer CNN, *experiment A*, gives very good results on its own, having very high accuracy. This shows that indeed the first layers of CNNs are made by easily designed filters. If more time had be spent designing more filters and tuning them, in order to get the best features out of the fixed layer CNN, then even closer performance could probably be achieved. This can be easily seen from the fact that in *experiment B* the filters were not fundamentally changed but remained very similar to the fixed layers filters.

When looking at the training time the fact that the fixed layer takes more epochs to get to a good result seems questionable. Since it has less weights to train one would expect it to take less time to train, though as seen that is not the case. This result can be explained by the fact that because it has less weights to train these weights need more training time in order for them to be perfectly tuned to give the same result. As such the fixed layer network will need an increasing amount of time to train.

VI. Conclusion

Concluding, indeed the layers of a CNN can fixed be one after another in order to finally get to a network with only set filters that are fully understood, without compromising its performance. This would require a lot of testing and research into what kind of filters are needed to get the optimal features out of each layer. The training time would increase as the number of fixed layers is increased. If such a network is designed it would be a network that we can fully interpret, giving some understanding in the workings of CNNs as a whole.

VII. Appendix

A. Code

All the code used to get these results can be found in the git: `https://github.com/AlKyrl/BA_Ass-Fixed-layer-CNN`

B. Confusion matrices

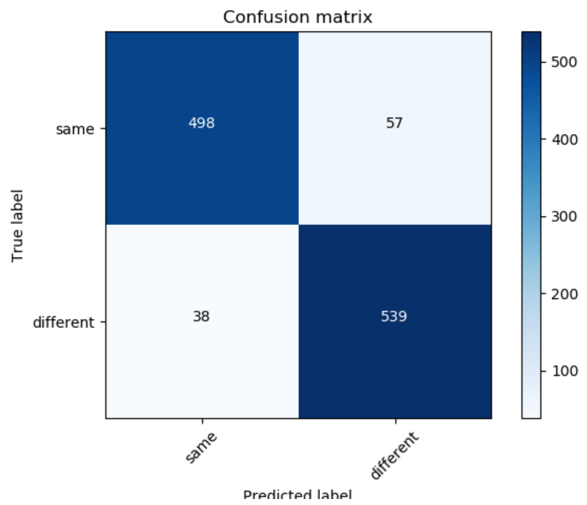

Fig. 16. Confusion matrix of Validation data for *Experiment A*

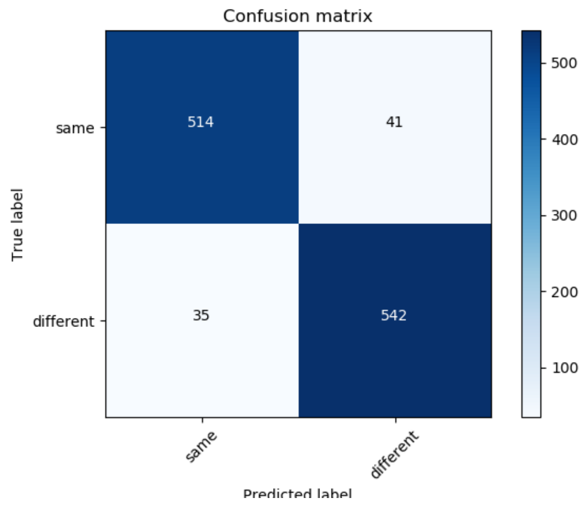

Fig. 17. Confusion matrix of Validation data for *Experiment B*

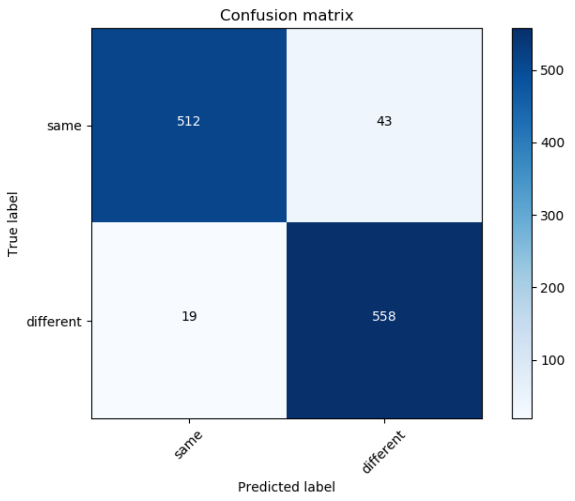

Fig. 18. Confusion matrix of Validation data for *Experiment C*

References

[1] F.H.J Hillerström, Deep Verification Learning (Master Thesis).University of Twente Department of Services, Cybersecurity and Safety 5.12.2016 .

[2] Receiver operating characteristic, `https://en.wikipedia.org/wiki/Receiver_operating_characteristic`.

[3] How to interpret loss and accuracy for a machine learning model, `https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-mach\ine-learning-model`

[4] Keras: The Python Deep Learning library, Keras Documentation, `https://keras.io/`

[5] GPU-accelerated Deep Learning on Windows 10 native (Keras/Tensorflow/CNTK/MXNet and PyTorch) ,`https://github.com/philferriere/dlwin`

[6] Kriesel 2007 Neural Networks, David Kriesel. A Brief Introduction to Neural Networks, 2007 `www.dkriesel.com`

[7] Course notes belonging to the CS231n lectures, `http://cs231n.stanford.edu/`

[8] Gabor filter, `https://en.wikipedia.org/wiki/Gabor_filter`

[9] Keras kernel initializer with gabor filter, `https://stackoverflow.com/questions/46177505/keras-kernel-initializer-with-gabor-filter`

[10] Image of Common CNN, `https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-\cnn-deep-learning-99760835f148`

[11] Face Recognition Grand Challenge (FRGC), `https://www.nist.gov/programs-projects/face-recognition-grand-challenge-frgc`

[12] Aligning face images. Philipp Wagner,`https://github.com/AlKyrl/BA_Ass-Fixed-layer-CNN/blob/master/Aligning%20face%20images.pdf`

[13] Y. Sun, X. Wang, and X. Tang, ”Hybrid deep learning for face verification,” in Proceedings of the IEEE International Conference on Computer Vision, pp. 1489-1496, 2013.