Deep Learning - Assignment 2

Introduction

In this assignment we requested to implement a Convolutional Neural Network (CNN), to solve the task of facial recognition. The specific task can be explained as such: given 2 images, determine if the same person is appearing in both images. In addition to the recognition task, we requested to integrate useful tools to improve the network training process.

Data

We used our network to classify Labelled Faces in the Wild (LFW) dataset - aligned grayscale version. There are 13233 images, image size is (105,105). In the data, of 5749 different people, 1680 of them with 2 or more images. We used the predefined division between train and test sets. Training set includes 2200 samples, first 1100 are matched pairs and another 1100 mismatched, given by the person's name and 2 input images. The test set, same columns structure, contains 1000 samples, 500 matched pairs and 500 mismatched.

Train\Validation split challenge
We splitted the original train data to train data and validation (hold out). In the progress of our work in this section we had a challenge:

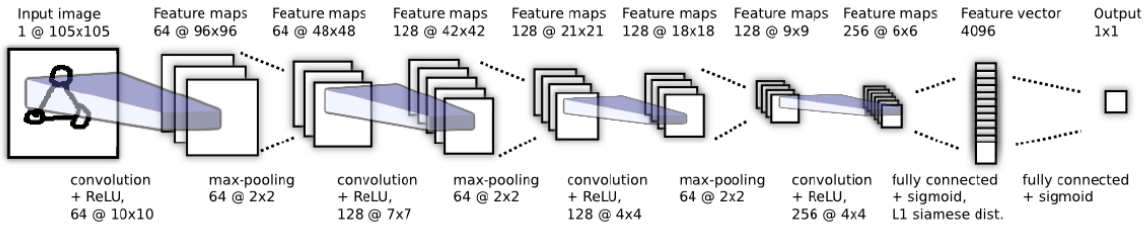
Originally the split was performed with regard only to the samples, but we noticed this resulted in a **data contamination**; samples of the same person can appear on both the train and on the validation. This is in contrast to the original train\test split which was splitted by persons. In order to alleviate this issue we split the data according to the split mentioned in the dataset webpage; it explains that the data is splitted into 10 subsets. The subsets are disjoint also in terms of persons as well as the samples in them.

We use 2 subsets which stand for 20% of the training data in order to choose our optimal configuration. We chose 20% because it is common in recent practices and from what we learned in previous courses.

Experimental setup

Our main goals are defining most efficient hyperparameters and analyzing the network performance (memory and running time) by training our network in the following configuration choices:

1. Architecture: we adopted completely our architecture from the paper; weight decay, number of layers, dims, filters. The base architecture is as follows:



   We initialized all network weights in the convolutional layers similarly to what appears in Kaiming He et al.(2015). In Our convolutional layers we used L2 regularizers like in the paper. In total our architecture consists of 38,951,745 parameters.

2. Batchsize - we adapted the batchsize from the paper (64)

3. Loss Function - we used Cross Entropy loss function as to the paper.

4. Epoches - we trained our base model with 200 iterations.

5. Optimization - we chose Adam optimizer; it is the most commonly used in the recent papers. we configured its initialization to a learning rate of 0.001.

The motivations for the configurations are: **Accuracy, loss and runtime.**

Our method is different in each experiment. We chose our configuration aggregately; on each stage of the experiment we added a factor over the optimal configuration we had up to that point. We evaluated for the following factors:

1. Convolutional stride - we hypothesize that decreasing the stride in the first layer will provide the network hidden layers with more information. This is based on that smaller stride is increasing the representation size by increasing the intersection between “windows” of the convolutional layer. We tried to modify it to 1 in the first convolutional layer.

2. Early stopping and model checkpointing - In order to prevent overfitting induced by training the model for too long on the training data we applied an early stopping mechanism. It evaluates the loss on the validation, If the model was not improved for 5 consecutive epochs we stopped the training procedure (model patience). Because the final model is bound to be worse than the one 5 epochs ago, we do not get the optimal model directly from the training procedure. In order to get the optimal model, we use a checkpoint mechanism; we will save the model each epoch then load the best one after the training procedure was stopped by the early stopping.

3. Batch normalization - Based on what we learned in class and recent literature, it is unclear what is the optimal placement for Batch normalization (batchnorm). Szegedy et al.(2015) suggested this method to minimize the internal covariate by applying the batch normalization between linear layers and activation layers. Later on, it was found that applying batchnorm after activation or after pooling layer is also beneficial. We tried: (1) both after activation and after pooling (2) only after activation (3) right after linear.

4. Augmentation - To extend our training data we applied several augmentation techniques on it. Note that the faces in the data are placed exactly at the center of the image in both the train set and in the test set. This dictates the use of augmentations that do not shift the center of the image too much. We applied random zoom, small random shear and random brightness adjustments.

Original image:

Augmented image:

5. Label clipping - We will apply a mechanism that was taught in class to prevent overfitting to binary labeled data. we used the formula that was presented in class:

$$[0,1] \rightarrow \left[\frac{\varepsilon}{k}, 1 - \frac{k-1}{k}\varepsilon\right]$$

where k is the number of classes and epsilon = 0.03 and epsilon = 0.01.

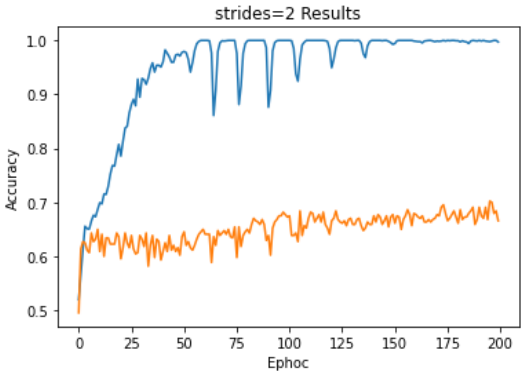
6. Learning rate scheduling - We will apply a mechanism that was taught in class to improve the optimization process.

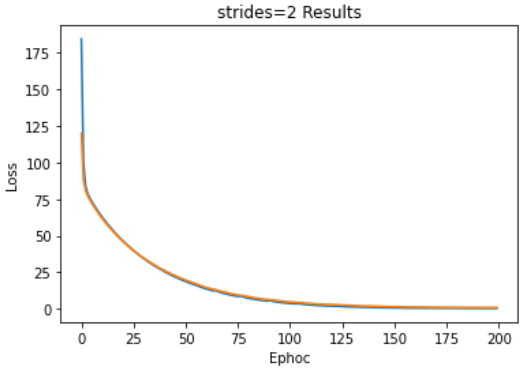
Results and Discussion

The output of the evaluation returns

- Accuracy and loss of training data
- Accuracy and loss of validation data
- Accuracy of test data
- Convergence time
- Final loss

First try - result of our base model



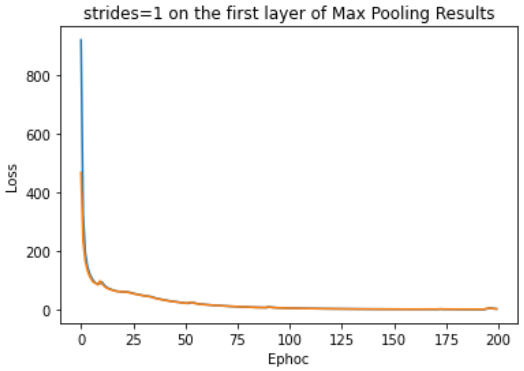


Accuracy: 0.6770
Loss: 1.0066
Number of epochs: 200

Training time per epoch:
1.8277259612083434 sec
Total training time:
365.5451922416687 sec

Convolutional stride

First convolutional layer stride modified to 1 (instead of 2): this resulted in a very slow compared to 2; 3 times slower than what it was with stride of 1. Also it provided lower accuracy, it looks by the graph the network didn't learn very well, possibly needed more iterations. Because of the lack of computational resources we could not continue all our experiments with this configuration so the rest of the experiments are with a stride of 2 in all convolutional layers.

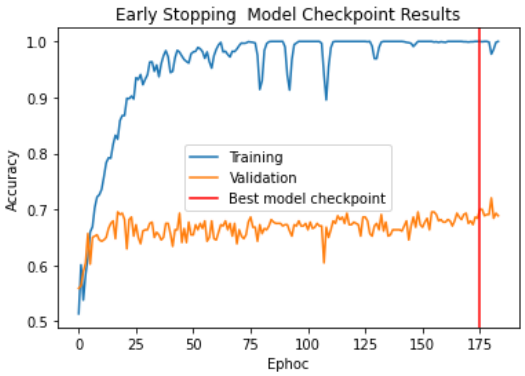


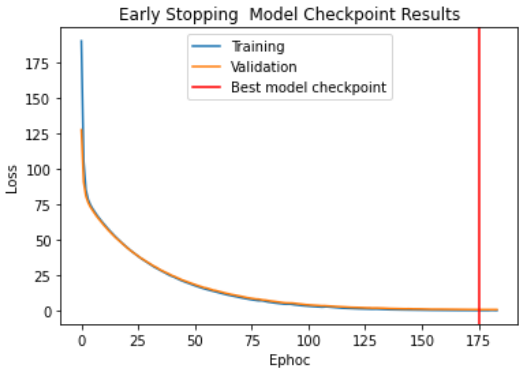
Accuracy: **0.5730**
Loss: 2.0834
Number of epochs: 200

Training time per epoch:
6.972641676664352 sec
Total training time:
1394.5283353328705 sec

Early stopping and model checkpointing

We tried to save each epoch but the platform we worked on was limited to small disk space so saved each 5 epochs instead. This mechanism improved each of the stages we will present below. Without early stopping and model checkpointing (ES and MC) the saved model would be the one represented by the last reported results in the graphs, using ES and MC we are able to take the optimal instead. The accuracy result was higher, the total time was longer but not significantly, apparently because of saving the models in the process.





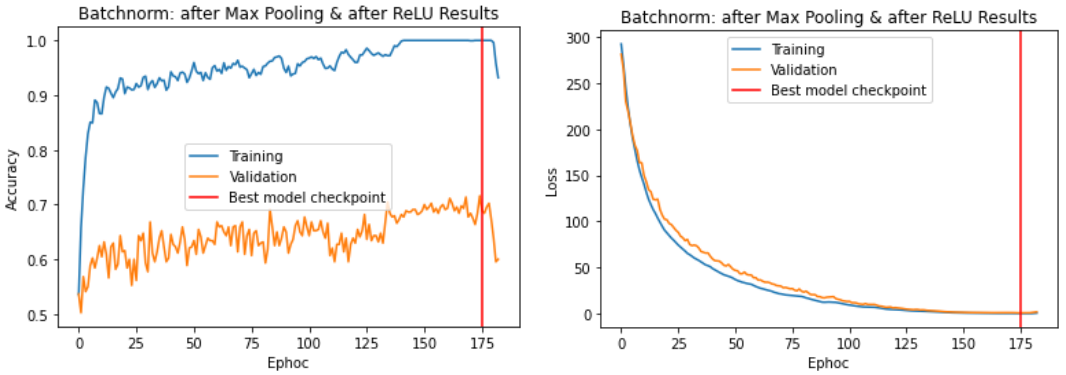
Accuracy: **0.6940**
Loss: 2.0834
Number of epochs: 184 (out of 200)
Best model epoch = 175

Training time per epoch:
2.0702033237270685 sec
Total training time:
380.91741156578064 sec

Batch normalization

In order to check all possible variations we tested all the common approaches today: (1) both after activation and after pooling (2) only after activation (3) right after linear. We saw that Batch normalization makes the network a little slower. This aligns with other analysis from recent literature and in Brock et al.(2021). But we saw that the loss and accuracy results were better. The best placement was found to be right before the relu (3) achieving **0.7010 Accuracy**. There was another good placement - right after the relu with 0.7030 accuracy and even lower loss than the (2) option, the training process got a little bit slower and required raising the model patience. The option of both after activation and after pooling batchnorm (1) wasn't good because of lower accuracy and the training took longer.

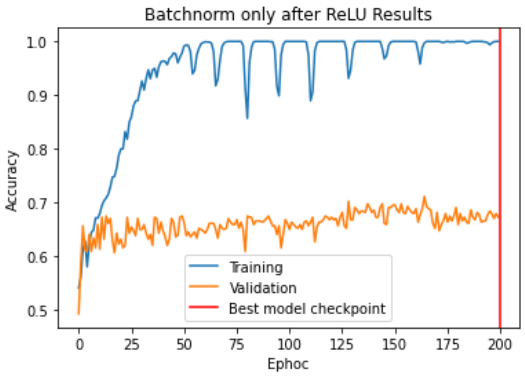
(1) Both after activation and after pooling

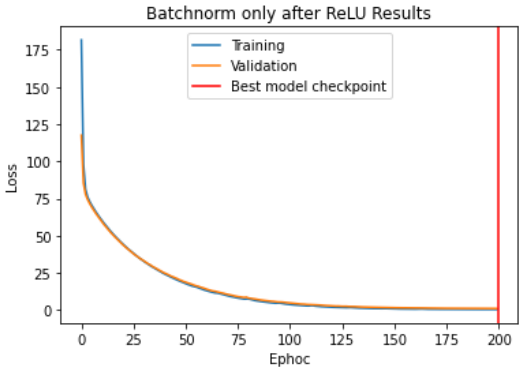


Accuracy: **0.6740**
Loss: 0.9671
Number of epochs: 183 (out of 200)
Best model epoch = 175

Training time per epoch:
2.1923255586624144 sec
Total training time:
438.4651117324829 sec

(2) Only after activation - as you can see, there were 2 trials. The first was while the model patience was 5, and the second was 10 because we saw the best model saved was on the last epoch.

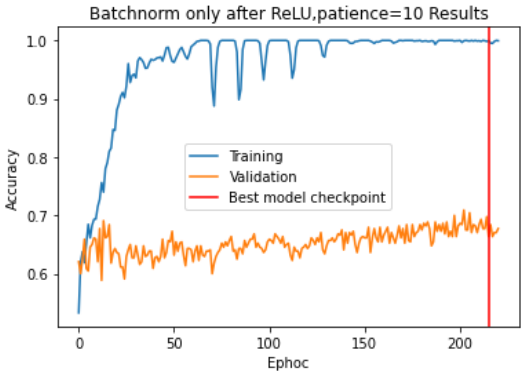


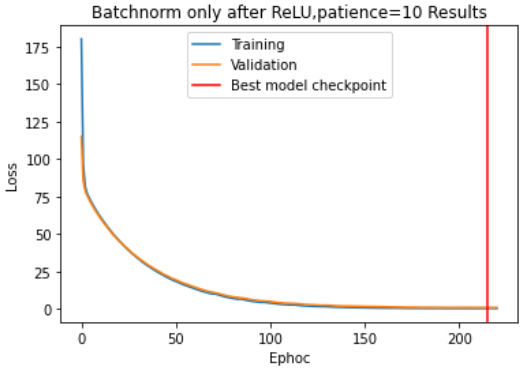


Accuracy: **0.6820**
Loss: 0.9811
Number of epochs: 200 (out of 200)
Best model epoch = 200

Training time per epoch:
2.261530832104061 sec
Total training time:
416.1216731071472 sec

After changing the patience of the model -

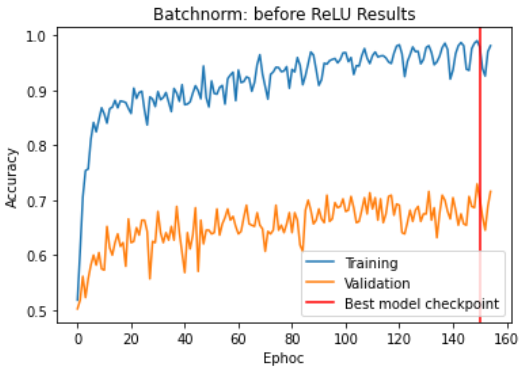


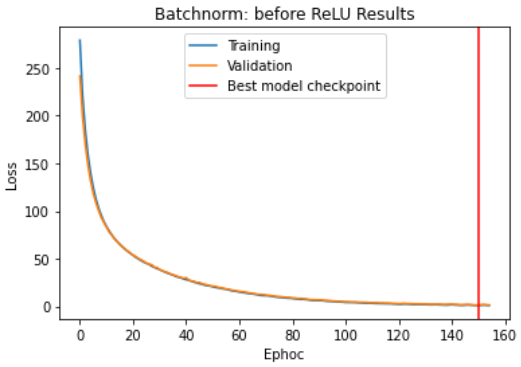


Accuracy: **0.7030**
Loss: 0.9642
Number of epochs: 221 (out of 300)
Best model epoch = 215

Training time per epoch:
2.0542405374449304 sec
Total training time:
453.9871587753296 sec

(3) After linear, before activation





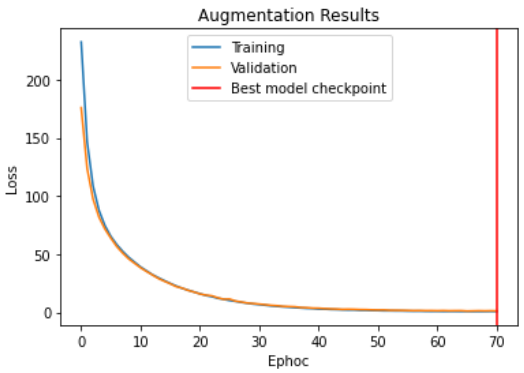
Accuracy: **0.7010**
Loss: 0.9642
Number of epochs: 155 (out of 200)
Best model epoch = 150

Training time per epoch:
2.0202520154212995 sec
Total training time:
369.7061188220978 sec

Augmentations

The augmentations extended the train set variation. In our first try, the augmentation results showed lower accuracy because the model stopped too early, probably because the new data that make the model straggle in the learning process. We raised the model patience and tested the best batchnorm option from the last experiments. batchnorm before activation achieved high accuracy but the training time was significantly higher, almost double. At last, using batchnorm after activation together with augmentations gave best results, it made the network take longer to converge but resulted in a better accuracy. It achieved **0.7120 Accuracy.**

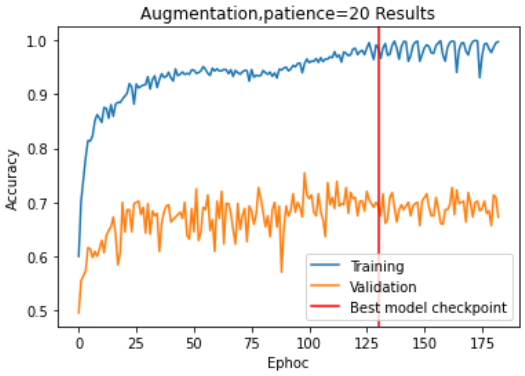
Results before raising the patience -

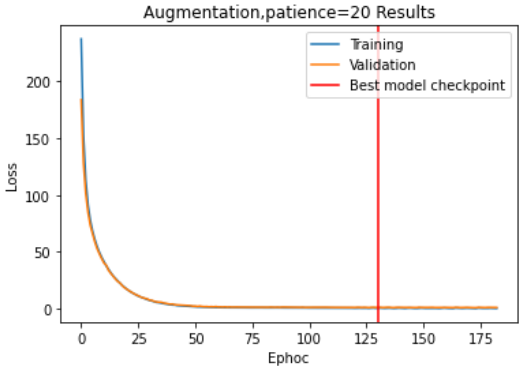


Accuracy: **0.6470**
Loss: 1.2955
Number of epochs: 71 (out of 200)
Best model epoch = 70

Training time per epoch:
2.836649196378646 sec
Total training time:
439.6806254386902 sec

Results after raising the patience with batchnorm before activation function -

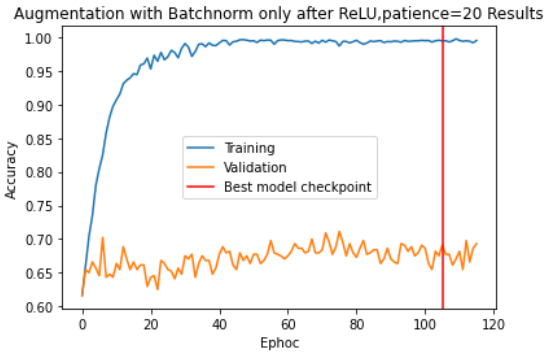


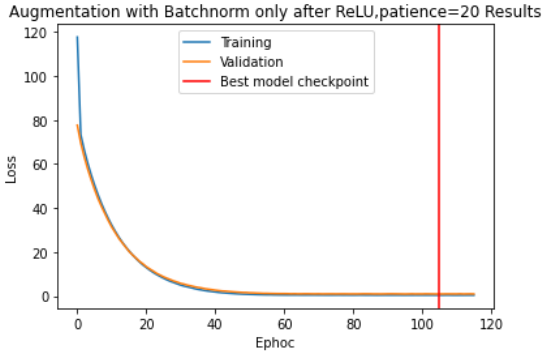


Accuracy: **0.7120**
Loss: 0.9142
Number of epochs: 183 (out of 200)
Best model epoch = 130

Training time per epoch:
6.099552842437244 sec
Total training time:
1116.2181701660156 sec

Results after raising the patience with batchnorm before activation function -



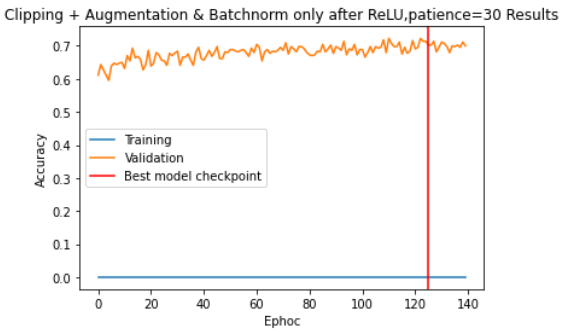


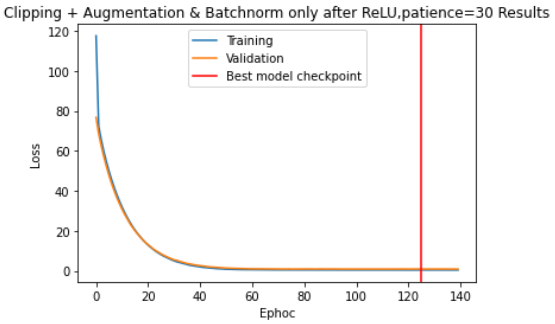
Accuracy: **0.7120**
Loss: 0.9142
Number of epochs: 116 (out of 200)
Best model epoch = 105

Training time per epoch:
5.265564908241403 sec
Total training time:
610.8055293560028 sec

Clipping

Softening the labels was supposed to prevent numerical issues caused by the sigmoid function, as we learned in class. Alas, it was found to decrease the accuracy to 0.6920. We tested for different variations of Clipping, including applying it with different parameters and only on the augmented data, both on the augmented data and on the original train data etc. It seems like higher patience on the model parameters gave higher accuracy but none of the variations provided better results.



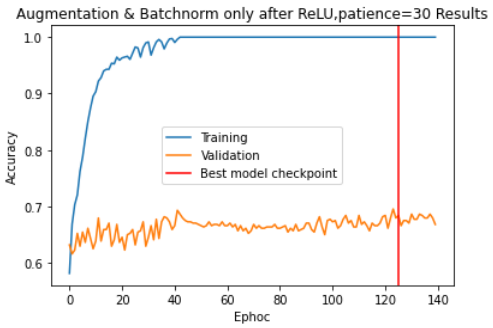


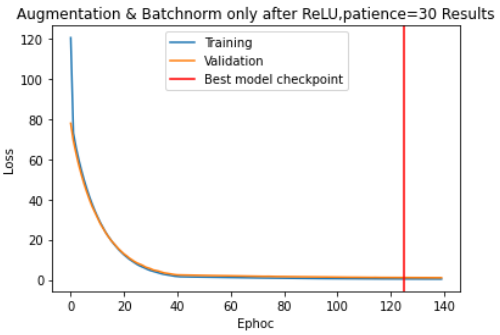
Accuracy: 0.6920
Loss: 0.8953
Number of epochs: 140 (out of 200)
Best model epoch = 123

Training time per epoch:
5.731665863309588 sec
Total training time:
802.4332208633423 sec

Scheduling

As a common practice we chose to decrease the learning rate by dividing its value in 10. From the experiments in the previous stages we deduced that the decrease in learning rate should be between epoch 40 to 60. epoch 40 seems to be the best from our experiments (it's when the training starts to slow down).





Accuracy: 0.6940
Loss: 1.0535
Number of epochs: 140 (out of 200)
Best model epoch = 132

Training time per epoch:
5.972412317139762 sec
Total training time:
836.1377243995667 sec

Examples of accurate and misclassified examples

We chose our most successful model to be examined; The model from the data augmentation stage. It was trained on augmented data but without clipping and scheduling.

Samples the network classified incorrectly:

(1)

(2)

As a human we can say that this is indeed the same person. But a small anecdotal experiment with humans showed that revealing the 2 pairs of images to humans to less than 1 second can cause them to classify this incorrectly. This was inspired by Elsayed et al.(2018), which showed that it is possible to craft samples that fool both time limited

humans and networks. As for technical difficulties: we see that the person in the first pair is at a different age of his life and is presented from a different angle in the second pair.

Samples the network classified correctly:
(1)

(2)

Here we see 2 samples classified correctly with high confidence (0.9 and 0.88). This can stem from the fact that the 2 people have prominent facial features, such as forehead and haircut. Also, the 2 pairs show images are from similar angles which we can assume as helpful. The same anecdotal experiment worked partially in this case, as our subject classified only one of the two correctly.