Wiederholungsübung – in-depth Deep Learning

Dataset: Labeled Faces in the Wild (additional csv-files for data loading in ZIP file)
Coded in Google Collab. Jupyter file and data is also on Github: https://github.com/saskiah98/MLE-Abgabe

The following resources were used while working with the dataset Labeled Faces in the Wild:

Dataset: http://vis-www.cs.umass.edu/lfw/

Research: http://vis-www.cs.umass.edu/lfw/
https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14
https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset/notebook
https://www.researchgate.net/publication/305552545_Labeled_Faces_in_the_Wild_A_Survey
(Learned-Miller et al. (2016). Labeled Faces in the Wild: A Survey. p. 229, 237)
http://vis-www.cs.umass.edu/lfw/lfw.pdf
https://datagen.tech/guides/image-datasets/lfw-dataset/
https://arxiv.org/pdf/2006.13026v2.pdf (Chrysos et al., Deep Polynomioal Neural Networks, p. 9)

Transfer Learning: https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/
https://www.atmosera.com/blog/facial-recognition-with-cnns/
https://neptune.ai/blog/transfer-learning-guide-examples-for-images-and-text-in-keras

Data Augmentation:
https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/

Frequently used architectures: https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset/notebook#Detecting-Faces:-Using-a-CNN-model-to-Classify-Unseen-Faces-in-the-'Labelled-Faces-in-the-Wild'-Dataset
https://paperswithcode.com/sota/face-recognition-on-lfw
https://openaccess.thecvf.com/content/ACCV2020/papers/Kim_DiscFace_Minimum_Discrepancy_Learning_for_Deep_Face_Recognition_ACCV_2020_paper.pdf

Confusion Matrix: https://github.com/emanuelfakh/Face-Recognition/blob/master/FR_Final.ipynb

Architecture 1: https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset/notebook#4.-Model-Construction
https://openaccess.thecvf.com/content/ACCV2020/papers/Kim_DiscFace_Minimum_Discrepancy_Learning_for_Deep_Face_Recognition_ACCV_2020_paper.pdf
https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/
https://pyimagesearch.com/2018/12/24/how-to-use-keras-fit-and-fit_generator-a-hands-on-tutorial/

Architecture 2: https://github.com/emanuelfakh/Face-Recognition

Architecture 3: https://github.com/FRI-Deep-Learning/lfw-cnn/blob/master/train_model.py

Architecture 4: https://github.com/StackAbuse/sa-image-recognition-keras/blob/main/SA-ImageRecognition.ipynb

Research about the dataset

The dataset Labeled Faces in the Wild is a database of photographs designed to study the problem of face recognition and to explore, create and test image recognition algorithms. It contains more than 13 000 images from 5749 different people. The images are 250x250 Pixels of the type of JPEG. The images are labeled with the name of the person in the picture. 1680 of the people have two or more distinct pictures in the dataset. Currently, there are six incorrectly labeled matched pairs in the dataset.

The faces were detected by the Viola-Jones face detector, which is an object detection framework from 2001 that creates multiple concepts for fast and accurate objection detection. Because of the use of the Viola-Jones face detector, the pictures are already roughly centered and scaled. The default slice of the picture removes most of the background.

The data in the dataset was labeled by hand and duplicates were removed. In the dataset a lot of groups, like children, babies, people over 80, women and different ethnicities, have little to no representation. Therefore, the dataset should not be used for commercial purposes, because the results may differ with different pictures. It is also possible to have a bias in the trained model, due to the provided data.

Since a lot of people only have 1-2 pictures in the dataset, the subset is often restricted to using people with more than a certain number of pictures in the dataset. The loader in scikit learn is often used with min_faces_per_person=70. So only people with more than 70 pictures in the dataset are used. This leads to a small subset with 7 people. The subset is also unintentionally biased, since all 7 people are politicians and male. This makes sense, since there are often more pictures from politicians compared to most normal people. Also, most politicians are male, so the gender bias also makes sense. This bias can influence the quality of the trained model.

For the results, the accuracy seems to be pretty high for most methods used, but a lot of papers don't specify any details about the subset used for the specific architecture. Therefore, it is difficult to directly compare it, since the subset used isn't always known for the achieved accuracy. As Learned-Miller et al. (2016) mention in their survey about the dataset labeled faces in the wild, the highest reported accuracy for face recognition in a peer reviewed publication in 2016 stands at 99.63% +/- 0.09% by Schroff et al., where only 22 errors were reported. Chrysos et al. achieve a verification performance of 99.7% with ResNet50 and 99.8% with Prodpoly-ResNet50 in 2020.

The most used architectures were:

- **Classic CNN**: A very commonly used architecture for a face recognition task with the labeled faces in the wild dataset is a classical CNN. CNNs have shown great results for face recognition tasks.
  With a subset of the 6 most frequently appearing people (so a bit smaller subset than min_faces_per_person=70) the precision in the final model averaged around 0.8 for all classes except Tony Blair
  Precision and recall were used as metrics, since Accuracy only measures performance without taking into account the costs of false positives and false negatives. Accuracy also performs badly when there is a large class imbalance, which is the case in this dataset.
  Though the subset only uses a sample of 75 pictures per person in the code.
- **ProdPoly-ResNet50:** According to https://paperswithcode.com/sota/face-recognition-on-lfw the Architecture Prodpoly has the highest accuracy of 99.8% for this dataset and has rank #1 for face recognition on the dataset labeled faces in the wild. There was no information, if a

subset of the dataset was used and I couldn't find enough detail about the architecture in the paper to use the architecture in the following steps.

- **ResNet50**: ResNet 50 has the second highest accuracy of 99.7% for this dataset. The architecture was proposed in the same paper as ProdPoly-ResNet50. There was also no information if a subset was used and because there was not much detail about the architecture in the paper, the architecture will also not be used in the following steps.
- **Cos-DiscFace**: The architecture achieves an accuracy of 99.6% on the labeled faces in the wild dataset. There was no information on the subset used. Since there was not much detail about the architecture in the paper, the architecture will also not be used in the following steps.

More architectures that came up frequently were: FaceNet, Center Loss, CosFace, Regular Face, OE-CNNs, Adaptive Face. All these architectures achieved accuracies over 99% on the labeled faces in the wild dataset.

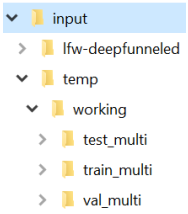
So, the state of the art for face recognition with the dataset labeled faces in the wild is that the highest achieved accuracy is 99.8% with the architecture ProdPoly-ResNet50. A lot of papers gave no detailed information about the subset that was used, but the most common subsets I found were either the top 6 most common people in the dataset, so the 6 people with the most pictures in the dataset, or people with more than 70 pictures in the dataset, so the top 7 most common people in the dataset. Since the proposed subset for the scikit loader was to use people with more than 70 pictures in the dataset, this subset will also be used in the following steps, to train the models, so the results are comparable.

Since most of the most commonly used architectures were not described in enough detail to re-build the architecture according to them, I only used the classic CNN from the most commonly used architectures in the following steps. For the next steps I used different architectures, that were used on the labeled faces in the wild dataset or for face recognition tasks with other datasets but were less common than the ones described above.

Data Preparation

The data was loaded manually. I used the following tutorial as basis for data loading: https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset/notebook#1.-Library-installation-and-data-read-in. The architecture from the tutorial was also used as the first architecture. The additional csv-files for loading the data can be found on github or in the ZIP folder. The images can be downloaded from: http://vis-www.cs.umass.edu/lfw/ (deep-funneled version).

The following file-structure is necessary for loading the data. The paths can also changed in the code, to fit a different file structure.



As described in the previous chapter a subset was used, where only people with more than 70 pictures in the dataset were added to the subset. To ensure more balanced classes I also used a sample of 70 pictures per class, instead of all available pictures for the 7 people.

```
The following people have more than 70 pictures in the dataset:
George_W_Bush        530
Colin_Powell         236
Tony_Blair           144
Donald_Rumsfeld      121
Gerhard_Schroeder    109
Ariel_Sharon          77
Hugo_Chavez           71
```

Since the faces are already centered, face detection is not needed for this dataset.

Data Augmentation

As data augmentation the ImageDataGenerator class from keras is used for all architectures. Keras ImageDataGenerator takes the original training data, applies transformation to each image in the batch and then replaces the original image batch with the augmented data, to train the model on the augmented data. The goal is to increase the generalizability of the model and that the model learns more robust features. The class rotates, scales, shears and flips the images, to create slightly modified versions of the input data. The ImageDataGenerator augments the data at training time.

The following parameters were used:
rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True

Architecture 1

https://datagen.tech/guides/image-datasets/lfw-dataset/

https://www.kaggle.com/code/jake126/face-detection-using-cnn-with-the-lfw-dataset/notebook#5.-Results-Analysis

The following architectures should build a model on the training data that can predict the name of the person in the picture. The training set should include multiple images of the people, to properly train the model. Then the model will predict the identity of the person in the picture.

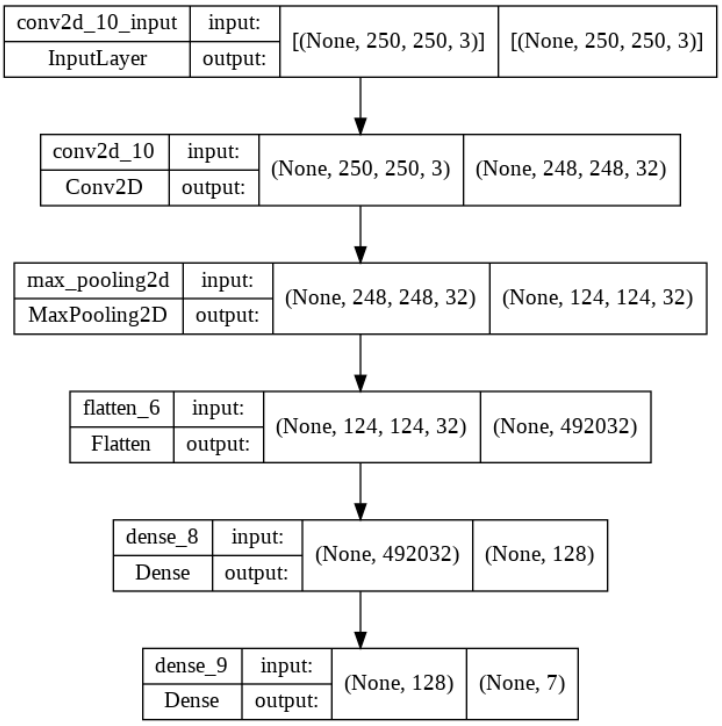
The data will be split into a training and test split. Common splits are 80%/20%, 67%/33% or 50%/50%. For the following architecture we will use an 80%/20% split, were 20% is the test size. Afterwards there is another split performed to get validation data.

The data in the LFW dataset has been preprocessed, so the face is in a similar portion of the picture, as seen in the picture below. The CNN model also considers the whole image, so differences in angle and alignment play less of a role and it is less important to detect the face in the image.

There will be performed 4 key-steps in this CNN:

1. Convolution: convolves the image with a feature detector to obtain a feature map
2. Non-Linearity: use an activation function to create non-linearities in the data
3. Pooling: reduces the dimension of the feature maps through using average convolved values
4. Connect Layers: organize the previous aspects of the model into a fully connected network

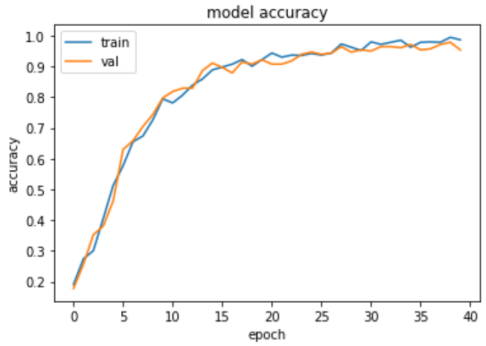
Model



The model consists of 1 Convolution Layer, 1 Max Pooling Layer, 1 Flatten Layer and 2 Dense Layers. All parameters are trainable.

Training with 40 Epochs

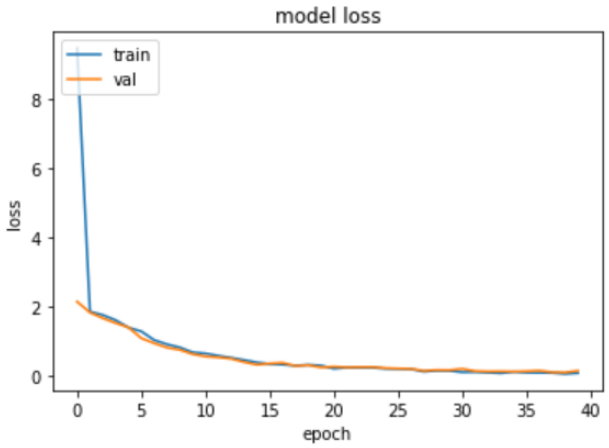
The accuracy of the CNN is 98,6%. The accuracy on the validation set is 95,3%, so not much lower than the training set. As seen in the following picture, accuracy is relatively stable during the last epochs. The model doesn't seem to overfit. Training data and validation data show similar results.




```
Epoch 38/40
23/23 [==============================] - 48s 2s/step - loss: 0.0748 - accuracy: 0.9785
Epoch 39/40
23/23 [==============================] - 46s 2s/step - loss: 0.0452 - accuracy: 0.9946
Epoch 40/40
23/23 [==============================] - 47s 2s/step - loss: 0.0692 - accuracy: 0.9866
```

Since the accuracy still changed about 1% between epoch 39 and 40 it could be good to train a bit longer by adding a few more epochs, to check if accuracy changes more or still stabilizes at 98%. Due to the long runtime of the architectures and the fact that it was already relatively stable during the last epochs, I didn't optimize this architecture further.

Loss changes very drastic during the first epochs but stabilizes after epoch 20. While it falls from 9.5 to 1.8 between epoch 1 and 2, loss decreases less during the following epochs.



Precision and Recall were both relatively high for this model.

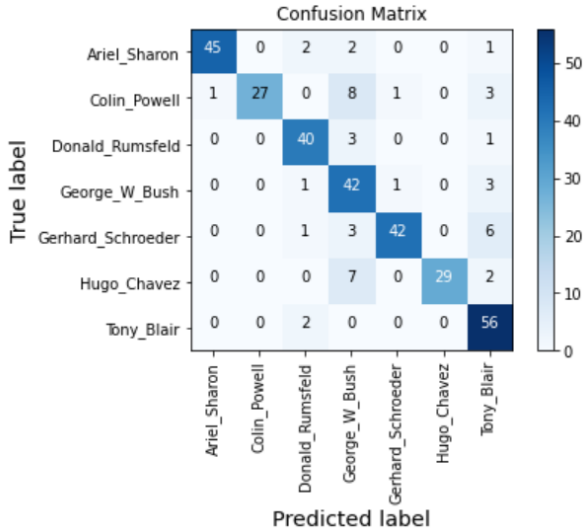
Precision ranged from 1.0 for Colin Powell and Hugo Chavez to 0.64 for George W. Bush.

Recall was highest for Tony Blair, with 0.96 and lowest for 0.67 for Colin Powel.

```
Precision:[0.9782608695652174, 1.0, 0.8695652173913043, 0.6461538461538462, 0.9545454545454546, 1.0, 0.7777777777777778]
Recall:[0.9, 0.675, 0.9090909090909091, 0.8936170212765957, 0.8076923076923077, 0.7631578947368421, 0.9655172413793104]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Hugo_Chavez', 'Tony_Blair']
```

So Colin Powell was never predicted as a label for another class, but not all images of Colin Powell were correctly identified (only 67% of the images of Colin Powell were labeled as Colin Powell).

Confusion matrix



The confusion matrix shows that Ariel Sharon was predicted correctly 45 times. He was predicted 46 times by the model, so only 1 time the label was predicted incorrect, and the algorithm mislabeled Colin Powell and predicted the label Ariel Sharon. Tony Blair was predicted correct 56 times, so even more times than Ariel Sharon. But he was predicted by the model 72 times in total, so he was

predicted incorrect more often, which is why precision was lower for him. Hugo Chavez was predicted less often, with only 29 predictions, but all of these predictions were correct (which is why his precision is 1.0). George W. Bush was labeled incorrect multiple times, from 65 predictions for Bush only 42 were correct. George W. Bush also has the lowest precision.

The confusion matrix also shows clearly the typical diagonal, that indicates that the model is doing a relatively good job at predicting the classes. Still there were a few mistakes, especially for George W. Bush and Tony Blair. These 2 classes had quite a few mistakes, compared to the other classes who barely had any mispredicted labels or even no incorrect predicted labels like Hugo Chavez.

Results

The CNN gets similar results as the results found online. With an accuracy of 98.6% the model is only a bit below the compared literature, which has an accuracy of >99%. Precision and recall were quite similar as in literature. Precision ranged between 1.0 and 0.64 for this CNN model. In literature it ranged between 1.0 and 0.77, so a bit higher. Recall had also a bit of a difference, in literature recall ranged from 0.79 to 1.0. In my code recall ranged between 0.675 to 0.96, so recall was a bit lower than in literature.

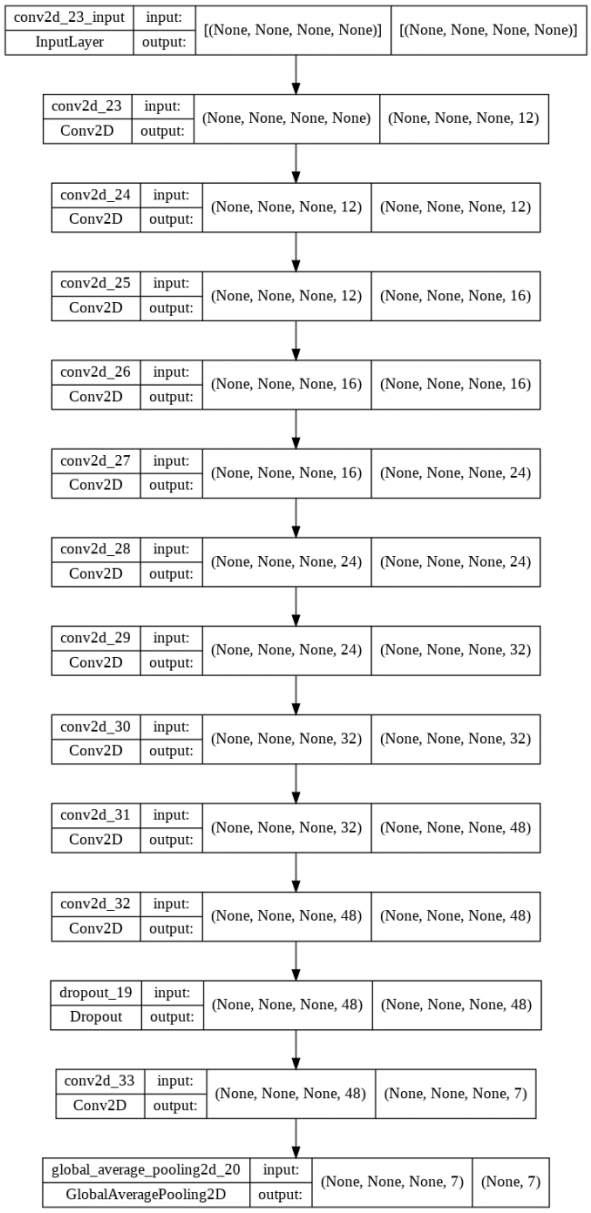
The subset in the original code was a bit different, which explains the small difference in accuracy, precision and recall. In the original code the 6 most common people in the dataset were used as subset, while I used the people with more than 70 pictures in the dataset. Instead of 75 pictures as sample I only used 70, since all 7 people had that many pictures in the dataset. The subset from other architectures in literature are not always known, but for most accuracy was >99%.

Architecture 2

https://github.com/emanuelfakh/Face-Recognition

The data augmentation, subset and data split were identical to the first architecture.

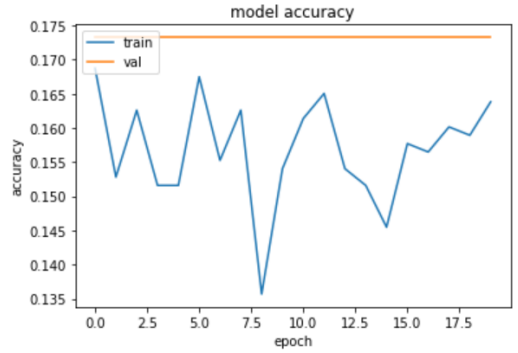
Model

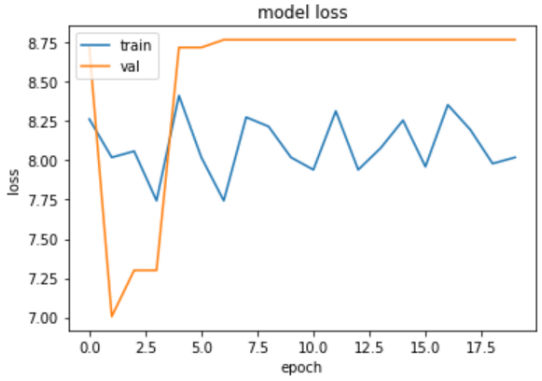


The architecture consists of multiple Convolution Layers, 1 Dropout Layer, another Convolution Layer and a GlobalAveragePooling Layer. All parameters are trainable.

Training with 20 Epochs

The accuracy of the model was generally extremely low. During the final run I achieved an accuracy of 16.3%. During previous runs the accuracy was even lower, at only 13.3% and 6.7%. Interesting was, that the accuracy of the validation data is always higher than on the training data and during the final run also completely stable. On the training data accuracy is less stable. I tried running the model with more epochs, but the results don't seem to get much better.



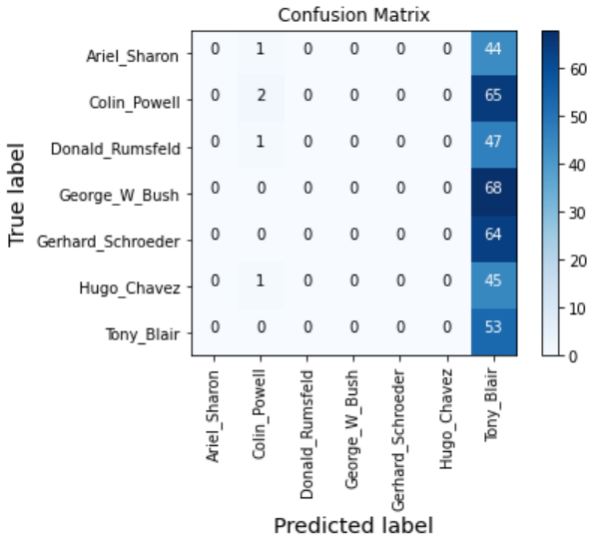


Loss is very high for this model during the whole training time. Loss stays around 8 during most of the epochs, while the lowest loss rate is at 7.7 during epochs 4 and 7. Loss is similar on training and validation data, though the curve looks like loss changes a lot, because the range is so small. After Epoch 5 loss on validation data is higher than on training data. Loss never gets low, as it is supposed to during training. It seems like the model doesn't learn much during training, since the accuracy doesn't rise much, and loss doesn't fall but stays high during most of training.

```
Precision:[0, 0.4]
Recall:[0.0, 0.029850746268656716]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Hugo_Chavez', 'Tony_Blair']
```

Precision and recall were bad for the model. Precision ranged from 0 to 0.4 and for most of the classes it wasn't even calculated (or calculated as 0, but not printed). Recall ranged from 0.0 to 0.02, but was also only printed for 2 of the classes.

Confusion Matrix



The confusion matrix also shows that the model doesn't do a good job on predicting the data correctly. Most predicted labels are incorrect, since the model labels most of the images as Tony Blair, while all other classes except Colin Powell were never predicted. Therefore, most predictions were wrong, since the model never seems to learn to differentiate the classes properly, as the confusion matrix shows.

There is also no visible diagonal in the model, which already indicates on first glance that the model isn't working as it is supposed to.

Results

The accuracy of this model was very low with only 16.3% during the last run. The results could be improved a bit from the previous runs, since accuracy was as low as 13.3% and 6.7% before. Loss stayed very high during the entire training between 8.4 and 7.7. Interesting was, that accuracy was higher on validation data and loss was also for the most part a bit higher on validation data, compared to the training data.

So, while I was able to optimize the results a bit, the architecture doesn't seem to fit well for the task and the available training data. One of the reasons might be, that the architecture consists mainly of Convolution Layers.

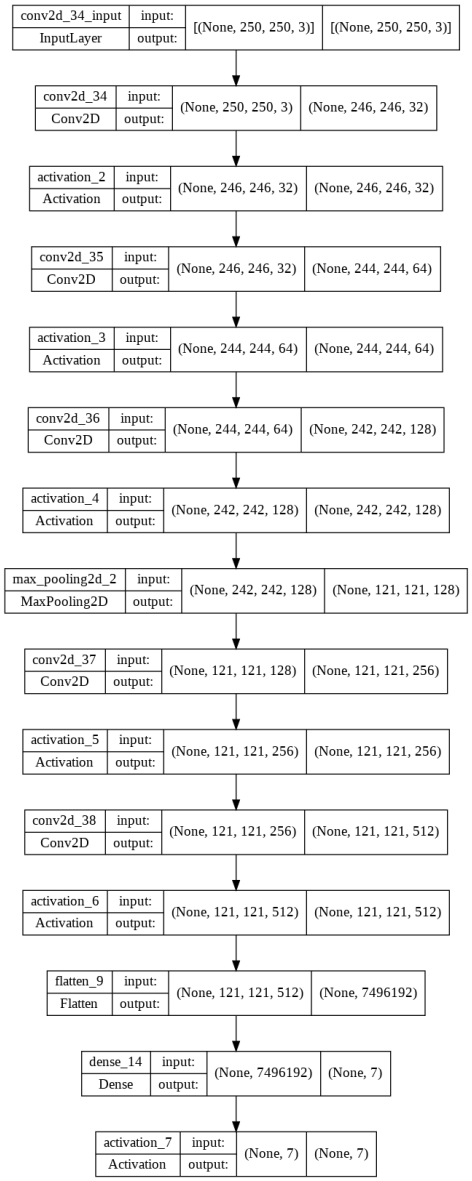
And while there would be ways to get better results through adapting the architecture, it would probably be best to change to a different architecture, instead of trying to optimize this specific architecture. Considering how low accuracy, how high loss and how bad the prediction results were for this model, using another model would probably be a better choice than optimizing this architecture.

Architecture 3

https://github.com/FRI-Deep-Learning/lfw-cnn/blob/master/train_model.py

The data augmentation, subset and data split are identical to the first architecture.

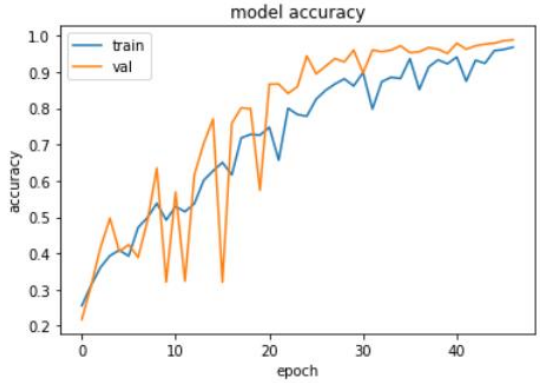
Model

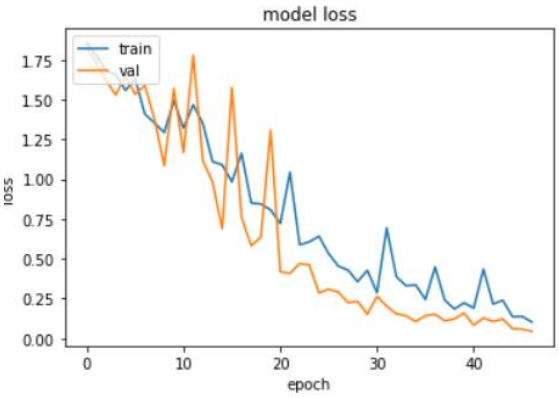


The architecture consists of multiple Convolution and Activation Layers and has 1 Flatten, 1 Dense and 1 Activation Layer at the top. All parameters are trainable.

Training with 47 Epochs

The accuracy of this model is 96.8%. As the graph for accuracy shows accuracy is relatively stable during the last epochs, the changes between the epochs at the end of training were <1%. So, the training time seems to be alright for this model. Accuracy of validation data and training data is close, so the model doesn't seem to overfit. But the accuracy is quite unstable during training, especially on validation data with changes of ~9% between epochs 13 and 14.



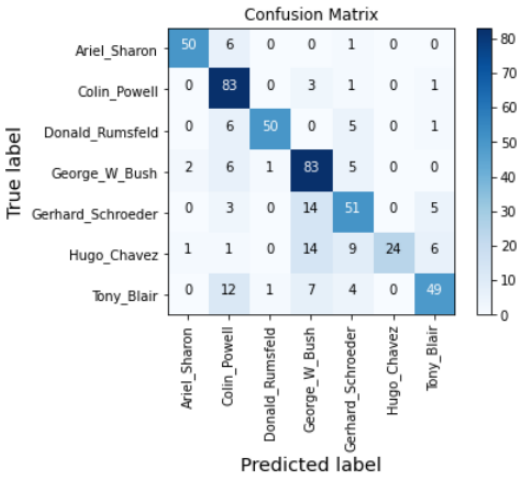


Loss is also unstable like accuracy. But it falls relatively low to 0.1 at the end of training on the training dataset and even lower on validation data with 0.04.

```
Precision:[0.9433962264150944, 0.7094017094017094, 0.9615384615384616, 0.6859504132231405, 0.6710526315789473, 1.0, 0.7903225806451613]
Recall:[0.8771929824561403, 0.9431818181818182, 0.8064516129032258, 0.8556701030927835, 0.6986301369863014, 0.43636363636363634, 0.6712328767123288]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Hugo_Chavez', 'Tony_Blair']
```

Precision is relatively high for all classes, ranging from 1.0 for Hugo Chavez to 0.67 for Gerhard Schroeder. Recall seems pretty high too, but not quite as high as precision, ranging from 0.94 for Colin Powell to 0.43 for Hugo Chavez. So, while Hugo Chavez has a precision of 1.0, recall tells another story about the predictions of this class.

Confusion Matrix



The confusion matrix shows the typical diagonal line quite well, with especially high numbers for George W. Bush and Colin Powell. But especially for George W. Bush there were quite a few wrong predicted labels, 14 for Hugo Chavez as well as Gerhard Schroeder.

The lower precision for George W. Bush is also visible in the confusion matrix. His label was predicted quite a few times for images of other people. Also, not all images of George W. Bush were predicted as such and therefore found by the model, but recall was still higher than precision for him (recall 0.85 vs. precision 0.68).

Results

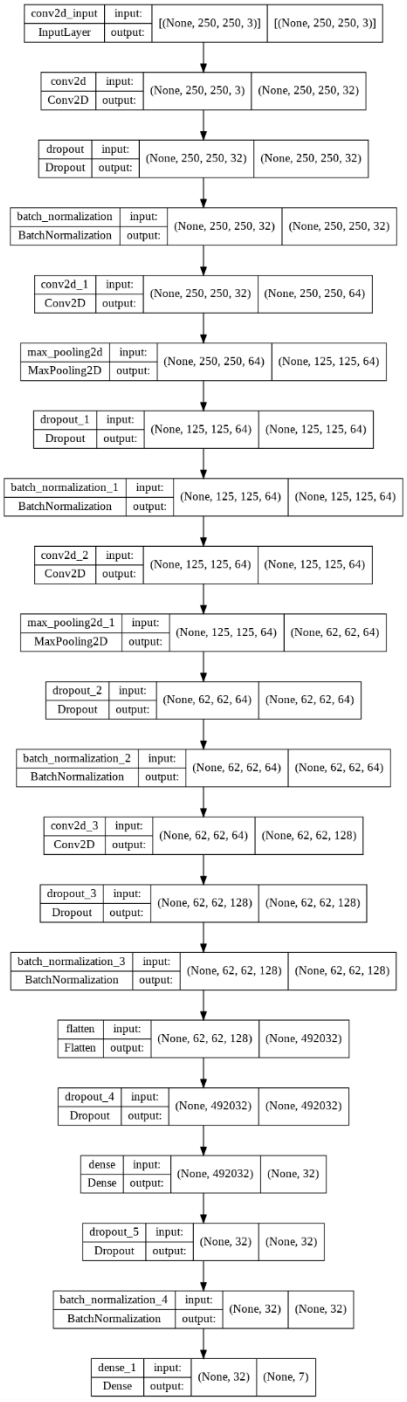
With an accuracy of 96.8% the model is doing quite well. It doesn't seem to overfit, since training and validation data both have a high accuracy. But both are a bit unstable during training, especially between epochs 8 and 20. Loss starts out quite high at 1.8 for training data, but falls to 0.1 at the end of training. Loss is even lower for validation data with only 0.04 loss at the end of training. Precision is relatively high for most classes, but the confusion matrix shows still quite a few errors, especially for George W. Bush.

Architecture 4

https://github.com/StackAbuse/sa-image-recognition-keras/blob/main/SA-ImageRecognition.ipynb

The data augmentation, subset and data split are identical to the first architecture.

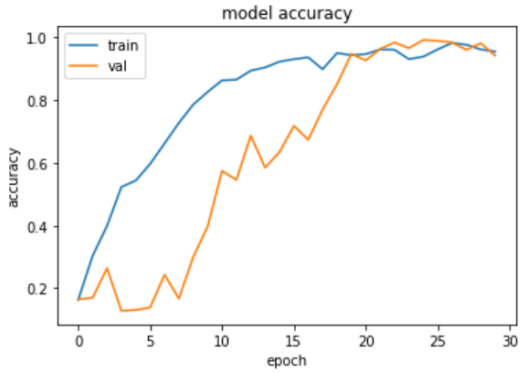
Model

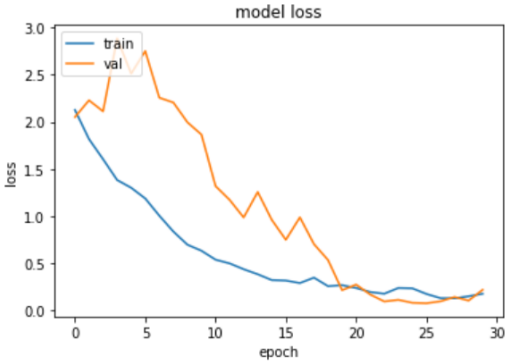


The architecture combines 1 Block with Convolution Layer, Drop out and Batch Normalization followed by 2 blocks with 1 Convolution Layer, 1 Max Pooling Layer, 1 Drop out Layer and Batch Normalization. Then follows again 1 Block with Convolution Layer, Drop out and Batch Normalization. The convolution layers have ReLu activation functions. The last block consists of 1 Flatten Layer, 1 Dropout, 1 Dense and another Dropout Layer and 1 Softmax Dense Layer at the top. There are 640 non-trainable Parameters in the model.

Training with 30 Epochs

The model achieves an accuracy of 95.5%. In the graph it looks like the model is overfitting during the first half of epochs, but the accuracy of validation data rises after that and aligns with training data. So, it seems that the model doesn't do a good job at generalizing, but since validation accuracy and training accuracy align more during the last epochs, the long training time seems to help and in the graph it looks like the model doesn't overfit anymore.





The loss graph verifies this. While in the beginning loss is a lot higher on validation data than on training data, it aligns more at the end of the training and both lines decrease a lot. So, while the loss is not as constant as it is in an optimal model, it improves a lot during training. Loss is at 0.17 at the end of training for training data and 0.21 for validation data.

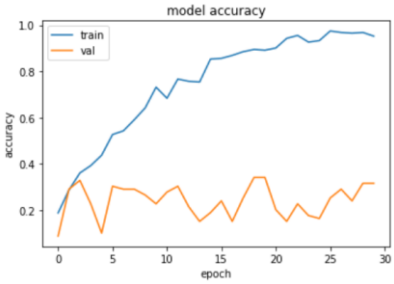
```
Precision:[1.0, 1.0, 1.0, 0.8604651162790697, 0.8064516129032258, 0.3111111111111111, 0.3765432098765432]
Recall:[0.4117647058823529, 0.5714285714285714, 0.20754716981132076, 0.4457831325301205, 0.36231884057971014, 0.84, 0.953125]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Hugo_Chavez', 'Tony_Blair']
```

Precision ranges from 1.0 for Ariel Sharon, Colin Powell and Donald Rumsfeld to 0.31 for Hugo Chavez. So, it is very different for the individual classes. Hugo Chavez and Tony Blair have a way lower precision than the other classes.

Recall ranges from 0.95 for Tony Blair to 0.2 for Donald Rumsfeld. So, the range is also very big and the recall is very different for the individual classes.

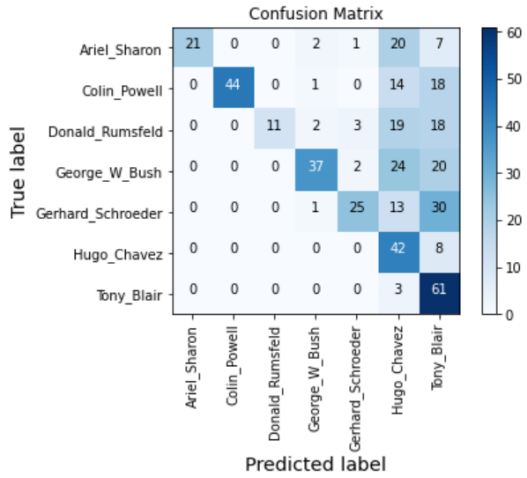
Previous run

Interesting was, that during a previous run the model showed heavy signs of overfitting without getting better at the end of training, as shown in the following graph of the previous run:



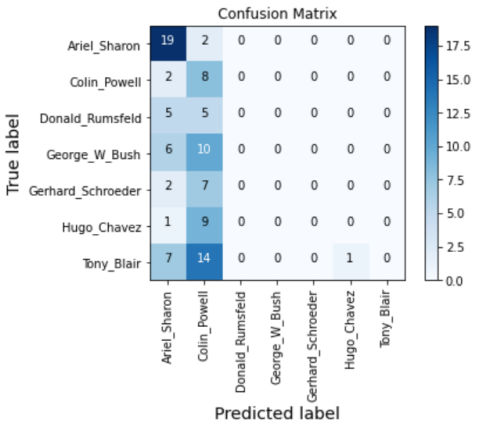
The graph shows the model is heavily overfitting and doesn't do a good job at generalizing the model. While the accuracy is also high on the training data (similar to the final run), the validation data reaches a way lower accuracy of only 31.6%. So, during the previous run it looked like the model is not a good fit for the task and the training data. But it improved a lot during the final run and was less overfitting.

Confusion Matrix



The confusion matrix shows that while the typical diagonal line of the confusion matrix is visible, most labels were predicted as Tony Blair and Hugo Chavez. So, while there were some correct predictions, a lot of the time the model predicts Tony Blair and Hugo Chavez, but the labels were incorrect. So while on the first glance the model doesn't seem to be doing too bad, the confusion matrix looks different. Despite the visible diagonal line, a lot of images were incorrectly predicted as Hugo Chavez and Tony Blair.

Previous run



The difference between the 2 runs is extremely visible in the confusion matrix. During the previous run the model predicts most pictures as Colin Powell and Ariel Sharon. Only 1 time another person (Hugo Chavez) was predicted by the model. During the second run mostly Hugo Chavez and Tony Blair were predicted, but there was also a visible diagonal line, so a lot of other labels were also labeled correctly.

I couldn't figure out the reason for the extreme difference in the results, since I didn't adapt the model or epoch count between these two runs. In the first run the model is heavily overfitting, in the second run it starts out like it is also overfitting, but it looks like the overfitting corrects itself after half of the training time and the model starts to generalize better. Though I don't understand how the model could correct itself, but it looks like it in the graph.

Results

One the first glance the results of this model seem to be pretty good with an accuracy of 95.5%. So, the accuracy of the model is relatively close to the accuracy of the literature and the state-of-the-art architectures. The graph shows that the model is heavily overfitting in the beginning and doesn't do a good job at generalizing. But after epoch 17 accuracy of validation data stabilizes and aligns with accuracy of training data. So, it seems the overtraining fixes itself during training time, though I don't understand how that can be the case.

Due to the extreme difference in the results of the 2 runs and the overfitting in the previous run, it might be the case that the model is too complex for the small dataset. And while the results in the final run were a lot better compared to the previous run, it seems like a different model is better suited for this task.

To make sure the model generalizes better and to avoid overfitting, like it happens during the previous run, there are two options:

- Training the model on more examples, by making the subset used for training bigger
- Changing the complexity of the model

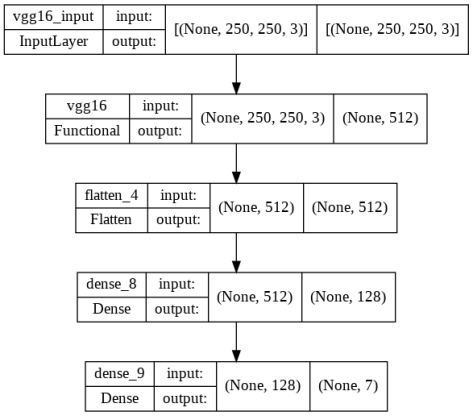
Since the 2 runs had such different results, it didn't make much sense to me to optimize the results further, especially since I can't figure out why the runs had such different results. And to make the results comparable I didn't want to change the subset of the data, since this would make the analysis difficult.

Transfer Learning

The data augmentation, subset and data split are identical to the first architecture.

Model

As pre-trained model the VGG16 model was used as base. The VGG16 model is one of the three most popular models for transfer learning and can be used for images.



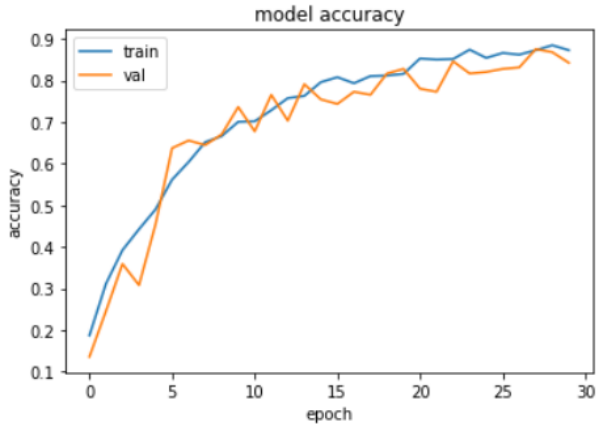
The architecture consists of the pre-trained VGG16 model, with 1 Flatten and 2 Dense Layers at the top.

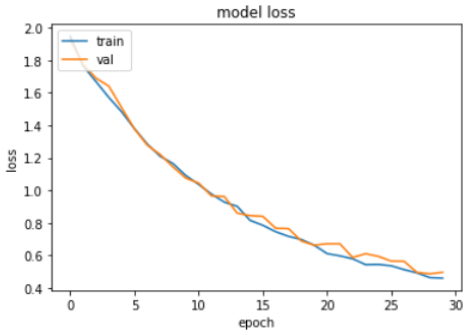
There were 14.714.688 non-trainable parameter and 66.567 trainable parameter. The non-trainable parameter are the frozen layers from the pre-trained VGG16 model, that were frozen with base_model.trainable = False, so the weights from the original VGG16 model are not updated during training.

Training with 30 Epochs

In the beginning the results of transfer learning were quite low. The accuracy during the first execution was only 30%, which was quite surprising since it was so much lower than a normal CNN and the results in literature. With optimizing of the model and parameters of the architecture I could achieve an accuracy of 87.2% during my last run. The accuracy was relatively stable during the last epochs, so training time seems okay. This is also visible in the graph.

The model doesn't seem to overfit. Training data and validation data show similar accuracy, though a bit unstable.





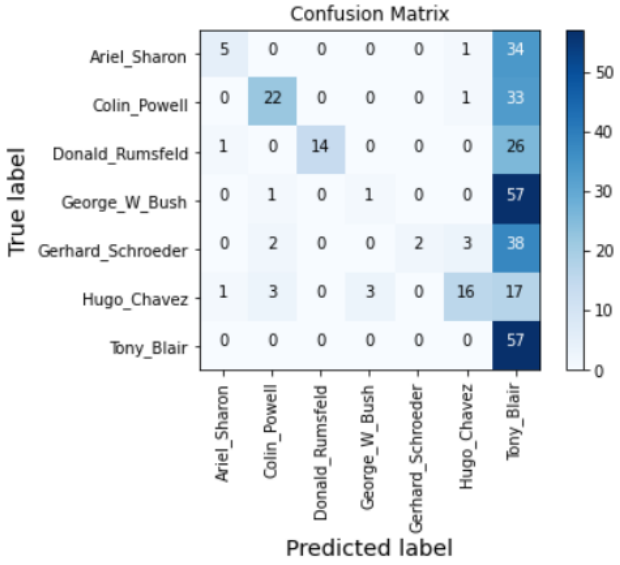
Loss seems quite good, loss decreases for both training and validation data continuously and at about the same rate.

Loss decreased from 1.9 to 0.46 in training. So, there were still some errors happening in the model, but the error rate dropped.

```
Precision:[0.7142857142857143, 0.7857142857142857, 1.0, 0.25, 1.0, 0.7619047619047619, 0.21755725190839695]
Recall:[0.125, 0.39285714285714285, 0.34146341463414637, 0.01694915254237288, 0.044444444444444446, 0.4, 1.0]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Hugo_Chavez', 'Tony_Blair']
```

Precision differed a lot for the different classes. It was 1.0 for Donald Rumsfeld and Gerhard Schroeder. For Tony Blair it was way lower with only 0.21. Recall was extremely high for Tony Blair with 1.0, the highest recall rate for all architectures. George W. Bush on the other hand had an extremely low recall with 0.01, the lowest for all architectures.

Confusion Matrix



The confusion matrix shows, that while accuracy was quite high for this model, it doesn't do a good job at predicting the actual labels correct. Most images were predicted as Tony Blair, which explains the low precision and extremely high recall rate. So all pictures of Tony Blair were correctly labeled as Tony Blair, but many pictures from other classes were also predicted as Tony Blair. So the high recall seems to be more of a coincidence, than actual good predictions.

Since most images were predicted as Tony Blair there were also lots of wrong predictions there. So, while this confusion matrix at least has a bit of a visible diagonal (unlike some of the other architectures), it is very clear that many of the images were mislabeled.

Results

The accuracy of this model was 87.2% and relatively stable at the end of training, so training time seems okay. But precision and recall tell a different story about how good the model is. While the model doesn't seem to overfit, it doesn't do a good job at correctly labeling the images either. Most predictions were for Tony Blair and most of them were incorrect. With precision ranging from 1.0 to 0.21, precision was quite different for the individual classes.

Results

Architecture	Number of epochs	Accuracy	Loss	Precision	Recall
Architecture 1	40	98.6% (train) 95.3% (val)	0.06 (train) 0.13 (val)	Between 1.0 and 0.64	Between 0.96 and 0.67
Architecture 2	20	16.3% (train) 17.3% (val)	8.0 (train) 8.7 (val)	Between 0.4 and 0.0	Between 0.02 and 0.0
Architecture 3	47	96.8% (train) 98.8% (val)	0.1 (train) 0.04 (val)	Between 1.0 and 0.67	Between 0.94 and 0.43
Architecture 4	30	95.5% (train) – overfitting in first half 94.2% (val)	0.17 (train) 0.21 (val)	Between 1.0 and 0.31	Between 0.95 and 0.2
Transfer Learning VGG16	30	87.2% (train) 84.2% (val)	0.46 (train) 0.49 (val)	Between 1.0 and 0.21	Between 1.0 and 0.01

For most of the state-of-the-art architectures accuracy was used as metric to evaluate the effectiveness of the model. And while accuracy alone is not always enough to compare how good the models actually work it will be used to compare the used architectures to the state-of-the-art architectures of the literature as a first basis. Some architectures in literature also used precision and recall, to measure how good the model performs, so they will be taken into account as well.

While the subset I used for my architectures was quite commonly used for training on many different models, I couldn't find the used subset for the architectures with the highest accuracy. So, there might be differences when it comes to the subset of the state-of-the-art architectures with the highest accuracies compared to my code. But all architectures I used are trained on the same subset, to make them comparable, even though the subset in literature might differ a bit.

The state-of-the-art for the labeled faces in the wild dataset is an accuracy of 99.8% for ProdPoly-ResNet50. Other common architectures with good results were: FaceNet, Center Loss, CosFace, Regular Face, OE-CNNs, Adaptive Face. All these architectures achieved accuracies over 99% on the labeled faces in the wild dataset.

The highest accuracies from my architectures are: 98.6% for architecture 1, 96.8% for architecture 3 and 95.5% for architecture 4. All of them had similar accuracies on validation data too. The state-of-the-art is an accuracy of 99.8% for ProdPoly-ResNet50 on the labeled faces in the wild dataset. So, while my models performed quite well according to their accuracy, the difference to the state of the art is noticeable with at least 1% less accuracy.

From the used architectures, 2 did a good job according to accuracy as well as precision: architecture 1 and 3. With accuracies of 98.6% for architecture 1 and 96.8% they come close to the state-of-the-art accuracy of 99.8% for ProdPoly-ResNet50 and their precision was very high too. Architecture 4 also had high accuracy, but for this architecture loss was a bit higher and precision was a lot lower. When looking at the confusion matrix for this architecture it also becomes clear, that the model didn't do a good job predicting the labels. The confusion matrices will be compared in more detail in the following chapter.

Surprising was, that transfer learning with a pre-trained model did not perform nearly as well as some of the other architectures. This may be the case, because the pre-trained model was too complex for the limited amount of data in the subset. Transfer learning only reached an accuracy of

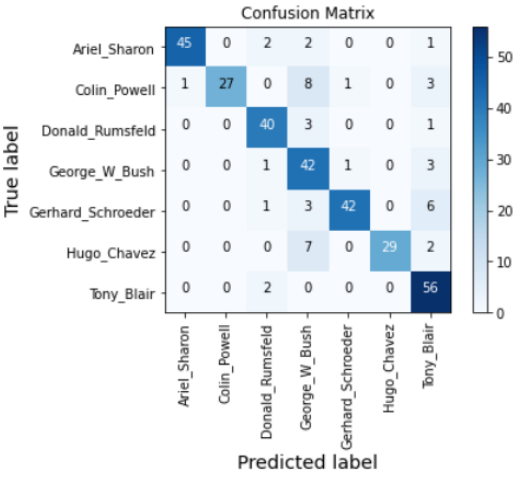
87.2% on training data and even lower on validation data with only 84.2%. Precision was extremely low for some of the classes, with the lowest precision being only 21% for Tony Blair.

By far the worst model was architecture 2. Even after optimizing the results the accuracy was only 16.3% and precision ranged between 0.0 and 0.4. Loss was also a lot higher than all other architectures with 8.0 on training data and 8.7 on validation data. All other architectures had loss values smaller than 1 and some of them even got close to 0 (architecture 1 on training data with only 0.06 and architecture 3 on validation data with 0.04).

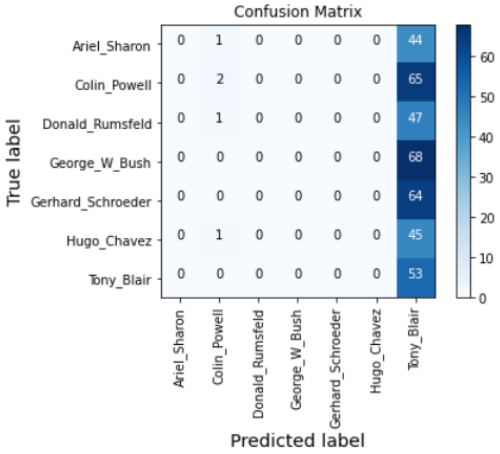
All architectures had quite a big range when it came to precision and recall.
Architecture 1, the architecture with the highest accuracy also has a high precision. The highest precision was 1.0 for Colin Powell and Hugo Chavez. Though the lowest precision of 64% for George W. Bush was a bit lower than the lowest precision of architecture 3. So, while architecture 3 has a bit lower accuracy, it performed better when it comes to precision. The lowest precision for this model was 67% for Gerhard Schroeder.
Recall was a bit lower than precision for all architectures. The lowest was by far architecture 2, with recall between 0.02 and 0.0. Architecture 1 also performed well when it comes to recall, with recall ranging from 0.96 for Tony Blair and lowest 0.67 for Colin Powell. The biggest range when it comes to recall happened with transfer learning. Transfer learning had the highest and lowest recall value of all architecture. Transfer learning was the only architecture with recall of 1.0 for Tony Blair and the lowest of 0.01 for George W. Bush.

Results - Comparing the confusion matrices

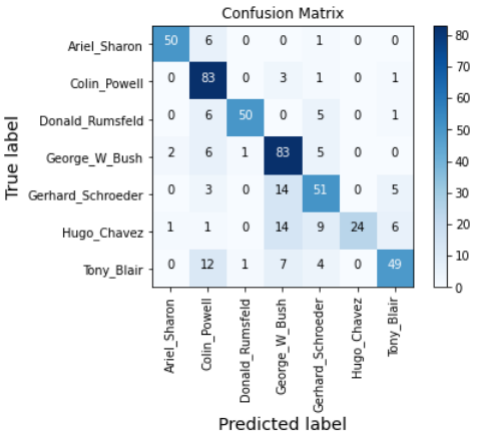
Architecture 1:



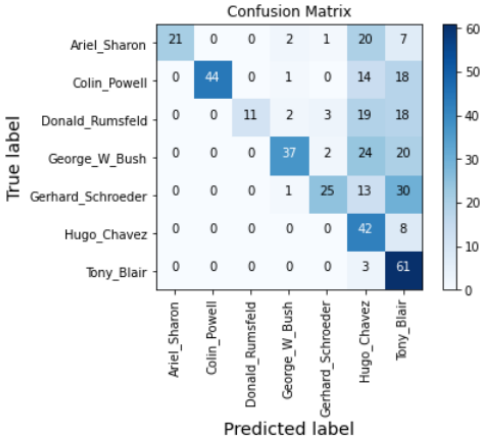
Architecture 2:



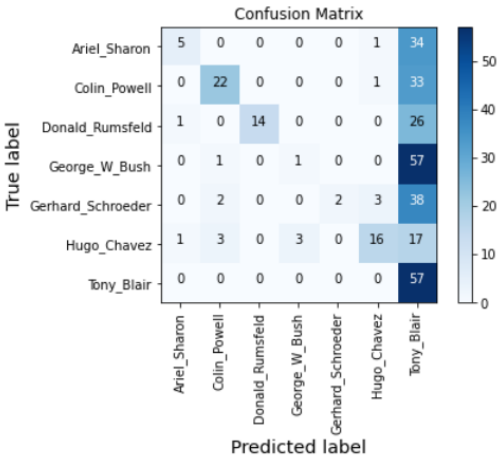
Architecture 3:



Architecture 4:



Transfer Learning:



The confusion matrices support the results from before: architecture 1 and 3 have the best results. For these architectures there is a clear diagonal line of correctly labeled images visible in the

confusion matrixes. Architecture 4 is not doing too bad, but Tony Blair and Hugo Chavez get incorrectly predicted very often for images of other people.

Conclusion

The best architecture is architecture 1, closely followed by architecture 3. This is supported not only because of high accuracies (with 98.6% and 96.8% quite close to the state-of-the-art accuracy of >99%) and the low loss rate, but also because precision and recall are very high, though some classes have a bit lower values. The confusion matrices for these architectures also show a clear diagonal line, which shows the correctly predicted classes. Though both still have a few errors.

Architecture 4 seems to overfit a bit in the beginning, though during the second half of training the model seems to be getting better and generalize better. Though the final run of this architecture had way better results than the previous runs another architecture might be better in practice, since the model doesn't seem to be reliable, considering the extreme differences between the runs.

Transfer learning did worse than all architectures except architecture 2. This might be because the pre-trained model is too complex for the small subset used. Precision and recall were still high for some of the classes, but very low for some others. So the pre-trained model doesn't seem to fit too well here.

By far the worst architecture was architecture 2. Though I could improve accuracy a bit compared to the first run the accuracy of 16.3% is still extremely low compared to the other architectures and the state-of-the-art accuracy of >99%. Optimizing this architecture further would not have made much sense, for use in practice it would be better to change to a different architecture.

For all of the architectures I managed to optimize the results quite a bit, compared to their first training runs. For architecture 2 and 4 further optimization was not useful, since the models didn't seem to fit the task and dataset well. For transfer learning the model seems to be too complex for the limited data, so further optimization probably doesn't make much sense here.

Interesting is, that all architectures seem to predict Tony Blair quite a lot. This goes for the better architectures like 1 and 3 as well as the architectures that don't do so well, so transfer learning and especially architecture 2. Architecture 2 seems to mostly predict Tony Blair with only predicting Colin Powell 5 times and never predicting any of the other labels.

For most of the state-of-the-art architectures I couldn't find much data about where the errors in the architecture were. But for the classic CNN, that I also used as the first architecture, Tony Blair also had the lowest precision (41%) and was often mis-classified.