## Machine Learning Report

# Exercise 3 – 3.8 Deep Learning

For this exercise the task 3.8 Deep Learning was chosen, to compare traditional feature extraction with CNN architectures. As datasets the suggested datasets “CarData” and “FIDS30” were chosen.

Unfortunately, I had to finish this Exercise alone. After losing a group member throughout the course, the third team member didn’t answer any of my messages, although we communicated at the beginning of the exercise. Besides that, he did not contribute anything to this.

## Traditional Feature Extraction

For traditional feature extraction the suggested way from TUWEL was chosen, based on color histograms etc. Additionally, a feature extractor based on SIFT and subsequent bag of visual words (BOVW) was implemented. The implementation was done in Jupyter notebooks, one per dataset and one per traditional feature extraction approach.

## 2.1. CarData

For the cars dataset *Figure 1* shows the performance of different classifiers based on different way of constructing the training data. The classifiers were used with the scikit-learn default parameters. *MLP\_OPT* and *RandomForest\_OPT* indicate runs with optimal parameter settings obtained using a exhaustive grid search. The different raining sets are the same, as they are constructed and named in the respective TUWEL example notebook. Every experiment was conducted using 10-fold cross validation.

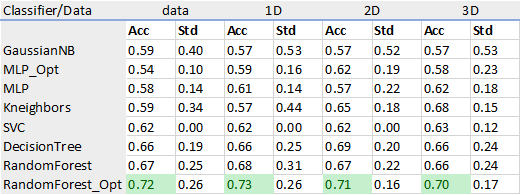


Figure 1: Performance per Classifier and training set

*Figure 2* therefore shows the first 15 results ranked by mean accuracy achieved with Random Forest Classifier. The optimal parameter settings therefore are default settings (*max-depth=None, max-features=”auto”*) and settings *n\_estimators* to 300. As *Figure 1* shows the performance can be slightly improved with optimal parameter settings (~+5% accuracy). We have chosen RandomForest and MLP for parameter optimization, as RandomForest showed the best performance without any hyperparameter optimization. MLP was chosen to compare it to the CNN performance later on, as it is a feed-forward fully connected neural network, in contrast to the convolution-based architectures in the second part of the report.



Figure 2: Results of grid search for RandomForest (first 15 ranked)

*Figure 3* shows the first 15 results ranked by mean accuracy achieved with MLP. However, when using these optimal parameter settings back in the original experiment shown above (*Figure 1*) leads to worse performance, than using default parameters, although the result of using the default settings is not residing in the top 15.



Figure 3: Results of grid search for MLP (first 15 ranked)

The performance per training set is shown in *Figure 4,* whereas *Figure 5* shows the performance per classifiers, grouped by training set. As already indicated by *Figure 1* the best performance was achieved using Random Forest with optimal parameter settings. Surprisingly not only GaussianNB, but also MLP shows the worst performance. However, we expected that the best result would be achieved using the “3D” training data, as it incorporates the most information. Although as *Figure 4* and *Figure 5* show, “3D” actually performs the worst, followed by “*data”,* whereas “2D” and “1D” yield the best results.

The comparison of runtimes of the relevant classifiers and training data combinations also yield interesting results. The “data” training set almost always takes the most time (except for SVC), whereas “1D” and “3D” are faster in comparison. Additionally, MLP is by far the slowest classifier, needing almost 10 times more time for the 10-fold cross validation when compared to the best performing classifier, Random Forest. Therefore the choice of classifiers also seems crucial here, longer training does not necessary yields better results.

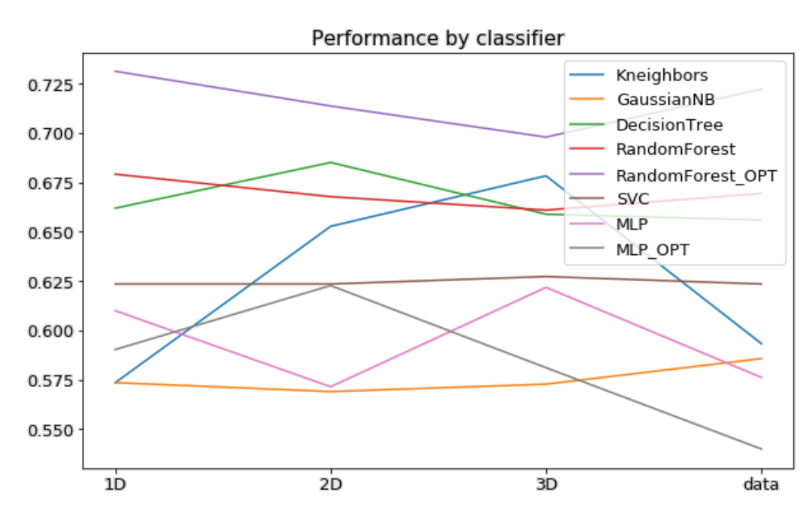
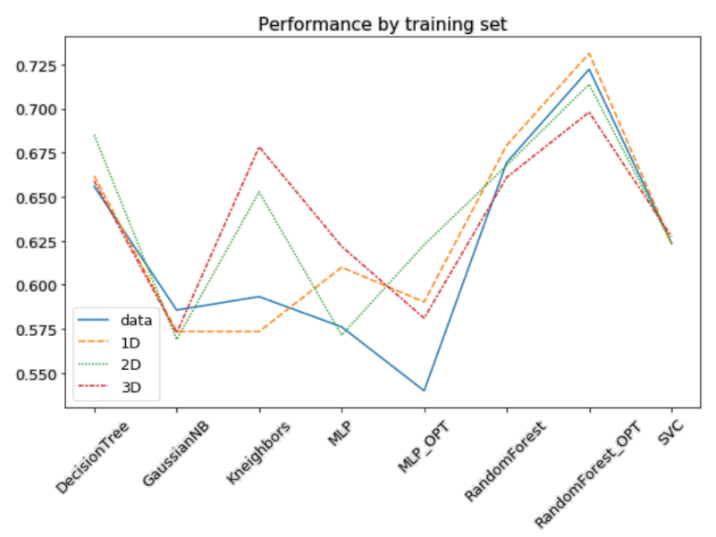


Figure 4: Performance by training set Figure 5: Performance by classifier

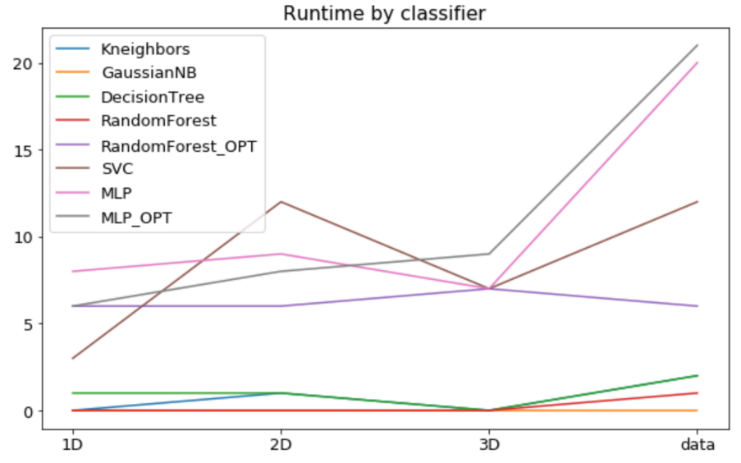


Figure 6: Runtime by classifier

As suggested by the assignment description, a different approach regarding feature extraction was implemented, mainly based on resources found online for implementing SIFT and subsequent bag of visual words [[1]](#footnote-1) [[2]](#footnote-2) [[3]](#footnote-3). However, we encountered some problems with OpenCV here. Due to licensing problems SIFT and a couple of other algorithms were moved out of the vanilla OpenCV distribution, into a opencv-contrib package. Furthermore, the latest version is not working as expected, making it necessary to compile it from source. Therefore, after researching on GitHub and similar pages, and following some workarounds, we simply used an older version of the package, namely 3.4.2.17.

Based on this version we implemented SIFT and subsequent bag of visual words, which worked quite nicely in conjunction with MiniBatchKMeans. Although we rather early experienced that this feature extraction approach is much slower than the one before. The run of MiniBatchKMeans alone sometimes took more than 15Minutes, without the processing steps needed before and after it.

The subsequent analysis pipeline was constructed similar to the one before. We conducted an exhaustive GridSearch (from scikit-learn) to determine optimal parameter settings for RandomForest and MLP. *Figure 7* and *Figure 8* contain the 15 best results for the different parameter settings. For RandomForest, again, it performs well with default parameters, and is only tuned by using a higher number of estimators.



Figure 7: Results of grid search for SIFT for RandomForest (first 15 ranked)

MLP performs better with a higher number of max iterations (500) and with a larger learning rate. Although the structure of the network (*hidden\_layer\_sizes)* seems to have a much smaller effect, as the different settings in the top 15 differ by slightly more than 2% overall.



Figure 8: Results of grid search for SIFT for MLP (first 15 ranked)

Repeating the same experiment as previously, we see that the performance can be significantly improved with this feature extraction approach. Expect *KNeighbors* and *SVC* every Classifier tops the 80% line of accuracy, which is quite an improvement to the 73% obtained for *RandomForest* in the previous setting. Furthermore the 10-fold cross validation itself was only slightly slower for every classifier expect *MLP* compared to the previous setting.

However, one has to consider the additional time needed for the more complex feature extraction, which is numerous times more time consumptive than the simpler approach used previously. Therefore, one has to consider this trade-off between runtime and accuracy here and use the respective appropriate feature extraction approach.

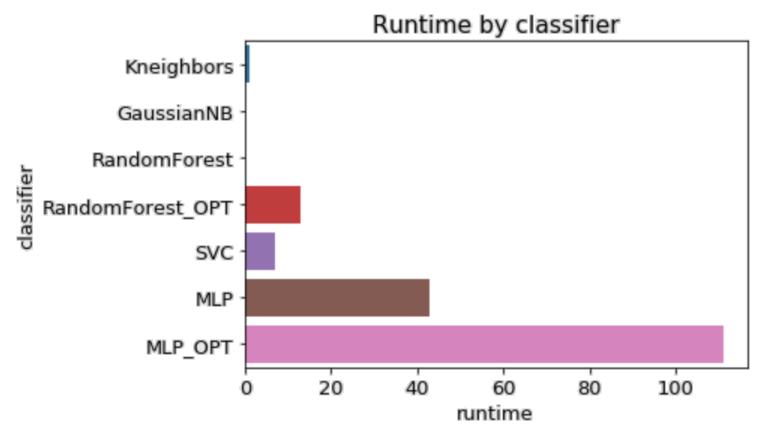
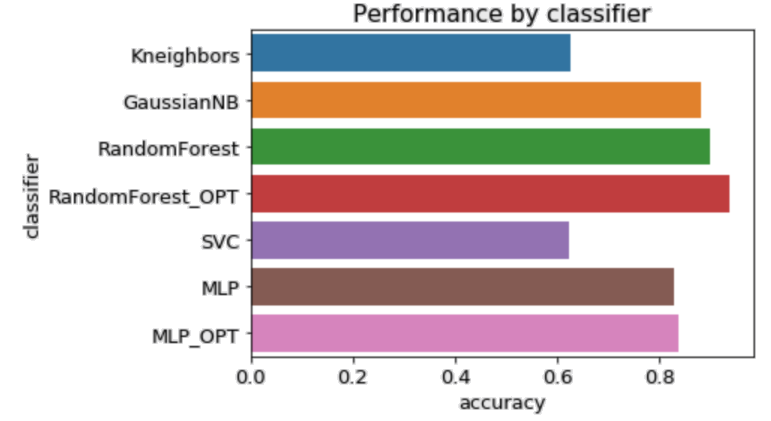


Figure 9: Runtime by classifier (SIFT, 10-fold CV) Figure 10:Performance by classifier (SIFT, 10-fold CV)

## 2.1. FIDS30

Using this dataset, containing 30 different kind of fruits, we performed the same experiments as with the previous CarData dataset. However, this dataset consists of RGB images, and the dataset is much smaller. It consists of 30 different classes, but there are less than 30 images available per class, effectively making it a much harder classification problem.

*Figure 11* summarizes the final results obtained for the simple feature extraction approach. As expected, the results get better with the number of dimensions or channels. Therefore, the worst performance is achieved with “data”, whereas the best performance is achieved with “3D” and *RandomForest* with an accuracy of 55%.

The exact results of the parameter tuning using scikit-learn’s GridSearch can be found in the relevant notebooks, to no use too many pages in the report. However, hyperparameter tuning improved the performance of *RandomForest* by 13% in the case of the “3D” training set. Whereas MLP could not be improved by much, and other classifiers also delivered much poorer performance.

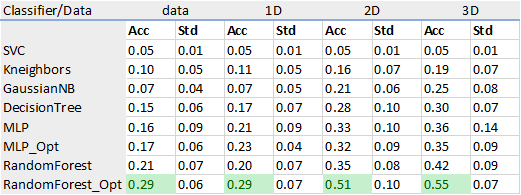


Figure 11: Performance per classifier and training set

*Figure 12* and *Figure 13* show the obtained results grouped by classifier or by training set. It clearly shows that all classifiers benefit from higher dimensions in the training data (except SVC, used with default parameters). Also, *RandomForest* in general performed better than any other classifier, throughout every setting.

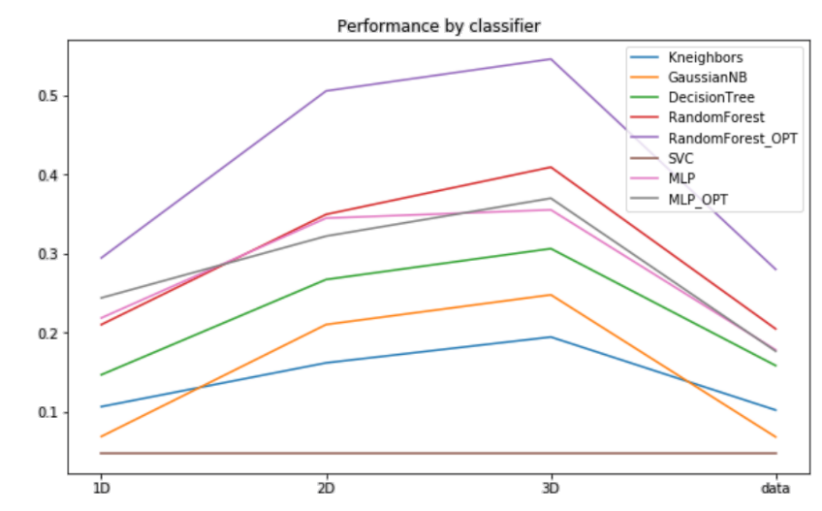
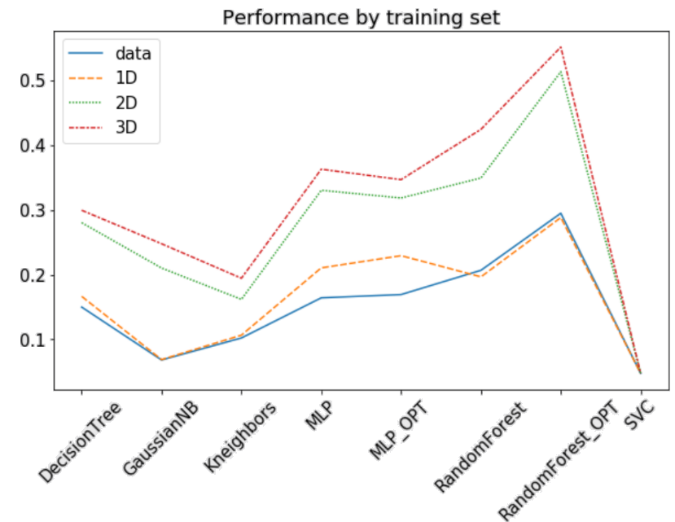


Figure 12: Performance by classifier Figure 13: Performance by training set

*Figure 14* shows the runtime needed per classifier for a complete 10-fold cross validation per training set. As already seen in the previous dataset, *MLP* is by far the slowest, but is intercepted by *RandomForest* in certain cases, due to the higher number of estimators (500).

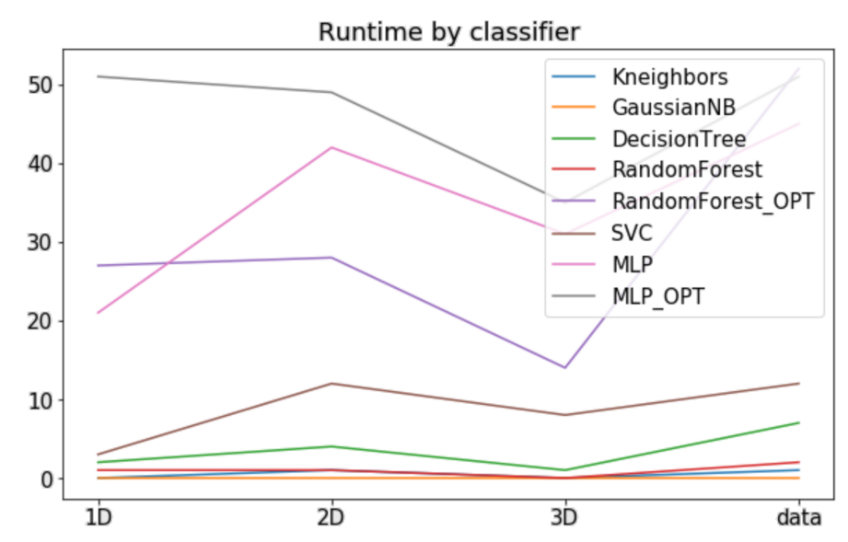


Figure 14: Runtime by classifier

*Figure 15* shows the confusion matrix obtained for the simple feature extraction approach with optimized RandomForest and 3D as training set. With an accuracy of 55%, the most confused classes are lemons and bananas, grapefruits and peaches, and acerolas and tomatoes. Which is quite reasonable from a human point of view, at least for the last two. As they have the same colour, and also forms. That lemons and bananas are confused by the classifier gives a hint that spatial properties are not considered very well if colour is not a distinctive property.

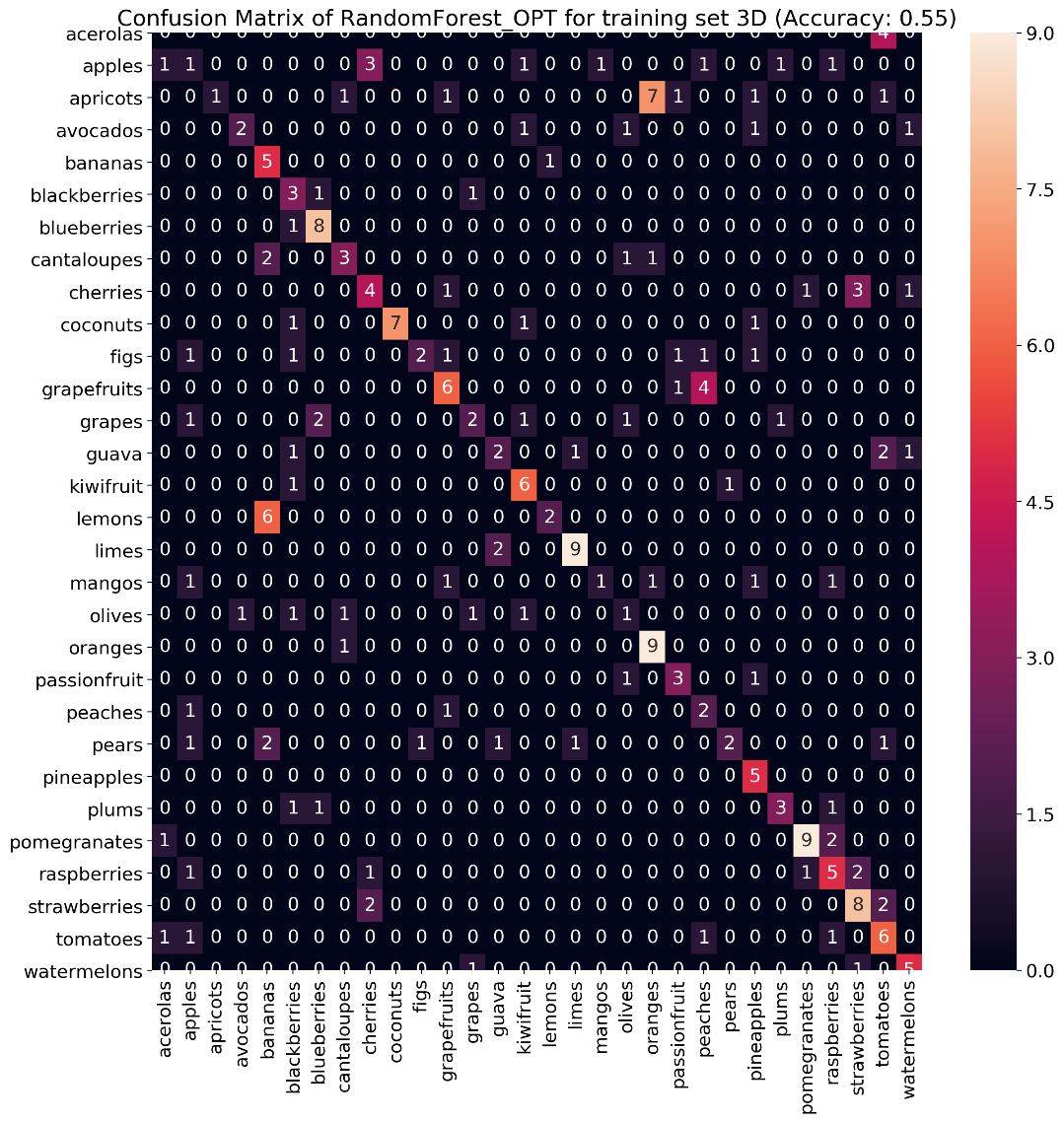


Figure 15: Confusion Matrix of RandomForest\_OPT/3D

The feature extraction pipeline based on SIFT for FIDS30 was built similar to the one for the CarData dataset. However, the performance could not be improved in this case, only the rank of the different classifiers changes. *MLP\_OPT* delivers the same performance in this setting as *RandomForest\_OPT* in the simpler feature extraction approach, whereas *RandomForest* gets significantly worse.In addition to the time needed for the more complex feature extraction using *MiniBatchKMeans* and SIFT, the final classification also takes significantly more time.

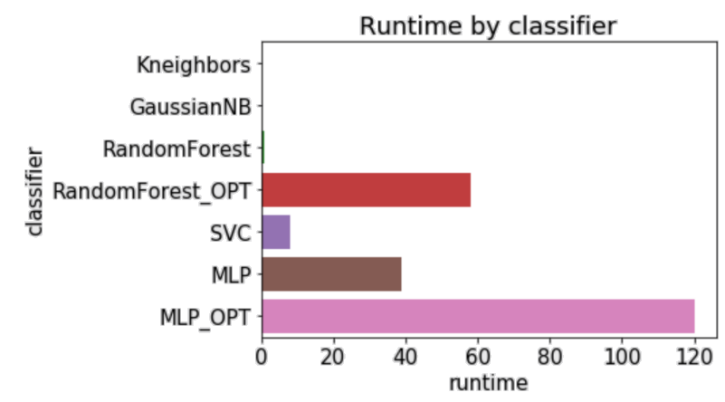
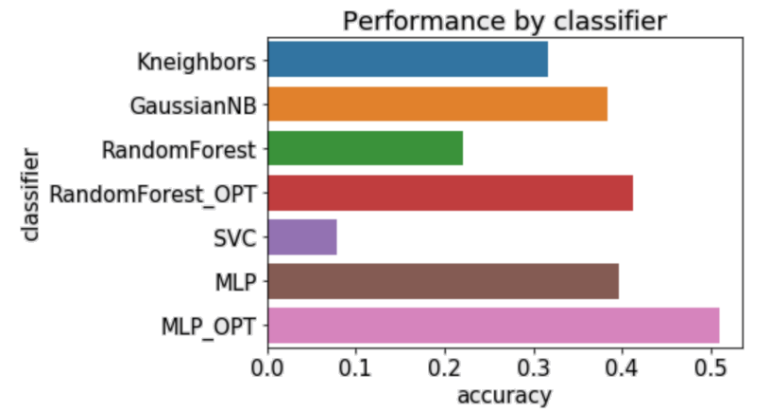


Figure 16: Performance by classifier (SIFT) Figure 17: Performance by classifier (SIFT)

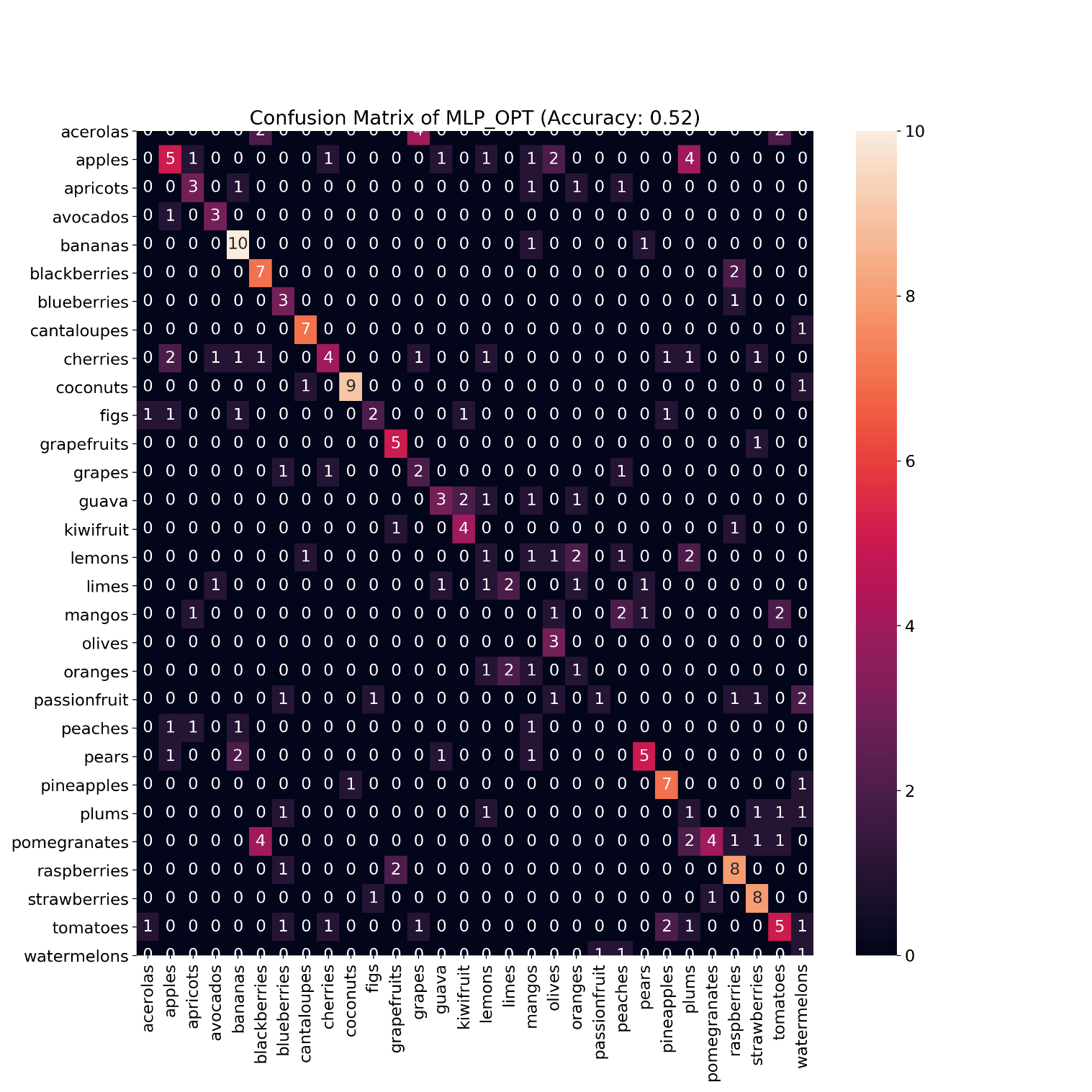


Figure 18: Confusion Matrix for MLP\_OPT/SIFT

*Figure 18* shows the confusion matrix like the one above, but for *MLP\_OPT* when used as a classifier based on SIFT and subsequent BOVW. The most confused classes are pomegranates and blackberries, acerolas and grapes, and apples and plums. Overall the confusion seems much more random than with the previous approach. Although one can guess some similarities between these 3 pairs.

## Convolutional Neural Network

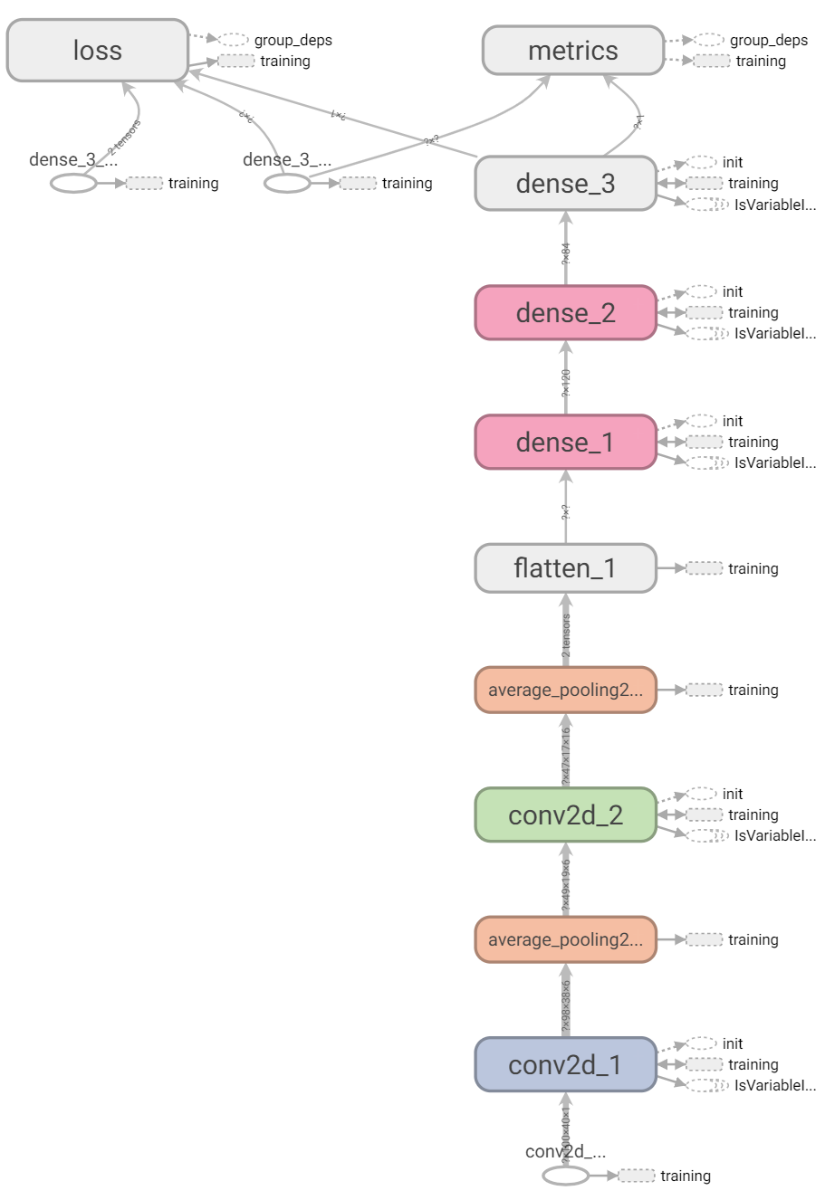
As starting point for our experiments with CNN for the 2 datasets given we have chosen the LeNet5[[4]](#footnote-4) architecture, which is quite popular but also relatively easy at the same time. For implementing our CNNs we have chosen Keras with Tensorflow backend, simply as we have some previous experience with it, and it makes it possible to quickly create a number of different layers and architecture in a very abstract way.

Figure 19: Structure of LeNet5

We also used the TensorBoard callback for Keras, which can be used to track experiments in a dedicated Web-Dashboard. This, in the end was very beneficial for this exercise, as it allows one to track the accuracy, runtimes, and validation scores obtained by different experiment runs, and it allows one to visualize the model in the form of a graph.

*Figure 19* shows such a graph generated by TensorBoard. It consists of the different layers, namely two convolutional layers with an AveragePooling layer each, followed by the flattening of the resulting feature matrix. Followed by 3 Dense layers, and the respective loss function and exported metrics.

One can vary a range of different parameters for a CNN architecture. Starting at the various convolutional layers, different numbers and sizes of filters can be chosen. To keep the architecture consistent, we did not alter these, also a filter size of 3x3 is proven in literature to be one of the better options. We only altered the image sizes, the batch size, the number of epochs and steps per epoch and the extend of data augmentation.

To implement the last one, we used Keras’ ImageDataGenerator functionality, which offers a range of different options for data augmentation. Starting from zooming and rotating, over rescaling to flipping and normalization.

As the plots generated by TensorBoard tend to get quite large, and *Figure 20* may not be quite readable, it can also be found among the submission files (*CNN\_Cars\_Tensorboard.png*). Respectively the results for the FIDS30 dataset can also be found in the file *CNN\_Fruits\_Tensorboard.png.* Generally speaking, the CNNs perform much better for CarData than for FIDS30. The example architecture (TUWEL example) is even able to top the performance achieved with SIFT or similar, as it reaches over 98% accuracy on the validation set. Although also the LeNet5 architecture can beat it reaching an accuracy of over 88%.

Regardless the architecture and the dataset, the performance can be improved by increasing the number of epochs to 100, or by using data augmentation. However, the performance can be worse if the data is augmented too much, and by vertically flipping images. As this only makes sense for certain domains. E.g. for cars it makes little sense, as images with cars lying on the roof are quite rare. Additionally, Keras offers two options namely *featurewise\_std\_normalization* and *featurewise\_center* which seem quite promising, but yield quite devastating results, cutting the accuracy in half in some cases.

Another observation was, that the performance by using *SGD* is significantly better, over when using *Adam*. Which we would not have guessed, as the choice of the optimizer was quite irrelevant when dealing with scikit-learn’s MLP.

For the FIDS30 dataset we also tried to increase the input sizes (when resizing the images), however it only increased the runtime and processing power needed but didn’t yield any significant improvement. Which, we assume, shows that enough information is already contained in a 64x64 image to draw distinctions between various kinds of fruits.

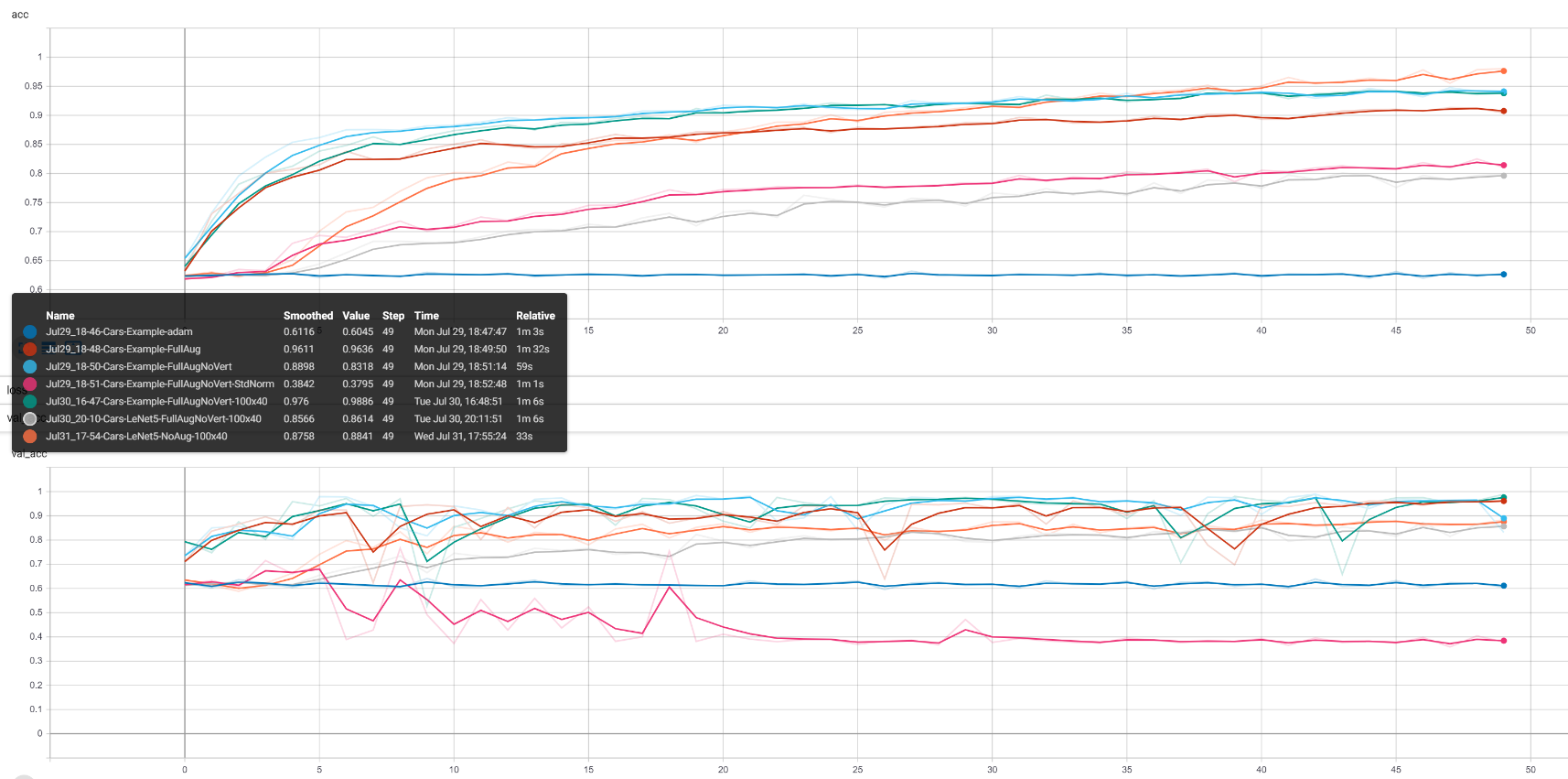


Figure 20: Experiments conducted on the CarData dataset

We then also tried to implement other CNN architectures which are more complex, and therefore seemed more promising to us, as we were not satisfied with the performance reached for the FIDS30 dataset. However, AlexNet[[5]](#footnote-5) somehow performed unrealistically bad, which more or less must be a hint, that our implementation was not correct. Also, our experiments with GoogleNet[[6]](#footnote-6) did not yield results, due to the quite cumbersome input and output structure, which yields to some problems when using Keras’ ImageDataGenerator.

To improve the performance for the FIDS30 dataset we also tried to apply transfer learning. Keras offers a range of applications for this purpose, from *ResNet50* to *VGG16*[[7]](#footnote-7). The result is shown in *Figure 21.* We tried *MobileNet, ResNet50* and *VGG16.* The best performance obtained using own CNN architectures was slightly over 60%, using the architecture from the TUWEL example and SGD optimizer.

This could be improved by both, *ResNet50* and *VGG16*, which was even able to reach more than 82% in validation accuracy. *MobileNet* although consisting of much more layers than our own implementations, was not able to reach more than 60%.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Figure 21: Experiments conducted on the FIDS30 dataset using Transfer Learning

## Conclusion

Only when finally comparing the results from the FIDS30 with the CarData dataset, we got that the usage of 3 channels for the CarData dataset is quite useless as it only consists of gray scaled images. Therefore of course every additional channel adds nothing to the information being already at hand. On the other hand, in the case of RGB images as contained in the FIDS30 dataset, every additional channel of course adds information, as the result is completely different if the fruit at hand is orange, red or blue.

We also showed with our results that the performance can be improved using more sophisticated feature extraction approaches, although it is not guaranteed, as shown with the FIDS30 dataset. Furthermore, it needs much more time, and therefore one has to consider this tradeoff between runtime and achieved accuracy, if the numerous times slower pipeline is worth the improved accuracy.

CNNs have the benefit of not being dependent on feature extraction to the same extend as traditional classification algorithms. This may save time, as one simply feeds in a 2D array with several layers per channel, and the CNN autonomously learns features etc, although improved performance is not guaranteed.

Transfer Learning on the other side, enables one to use architectures trained on millions of images, using dozens of layers. With Keras it is only necessary to import the relevant model, add a couple of Dense layers at the end, and the results are really promising, delivering the best results with the least effort of all approaches considered in this report.

1. <https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html> [↑](#footnote-ref-1)
2. <https://www.kaggle.com/pierre54/bag-of-words-model-with-sift-descriptors> [↑](#footnote-ref-2)
3. <https://kushalvyas.github.io/BOV.html> [↑](#footnote-ref-3)
4. <https://engmrk.com/lenet-5-a-classic-cnn-architecture/> [↑](#footnote-ref-4)
5. <https://www.learnopencv.com/understanding-alexnet/> [↑](#footnote-ref-5)
6. <https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7> [↑](#footnote-ref-6)
7. <https://keras.io/applications/> [↑](#footnote-ref-7)