## 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.

## 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.

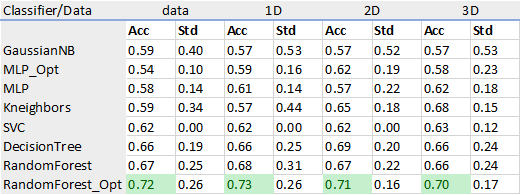


Figure 1: Performance per Classifier and training set

*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.

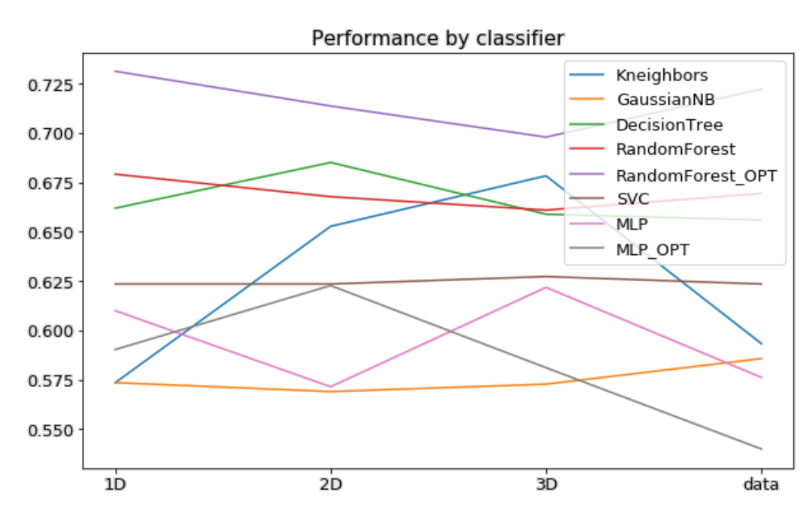
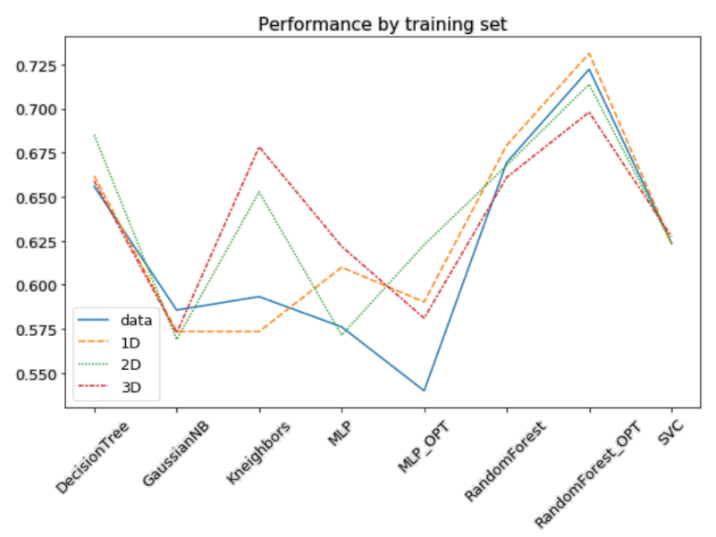


Figure 4: Performance by training set Figure 5: Performance by classifier

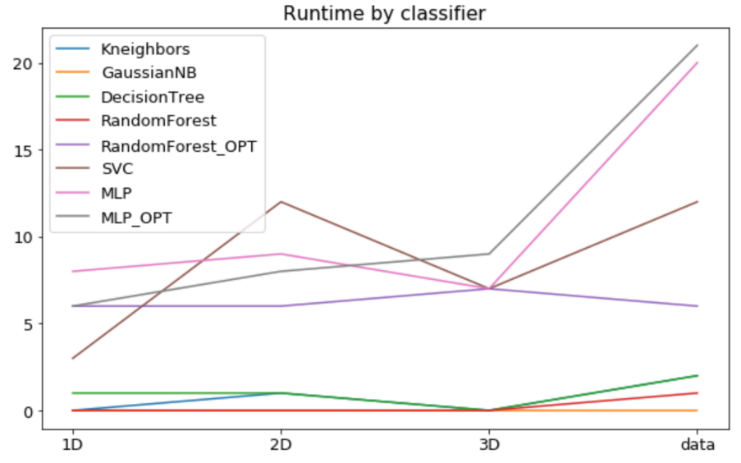


Figure 6: Runtime by classifier

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 has 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.

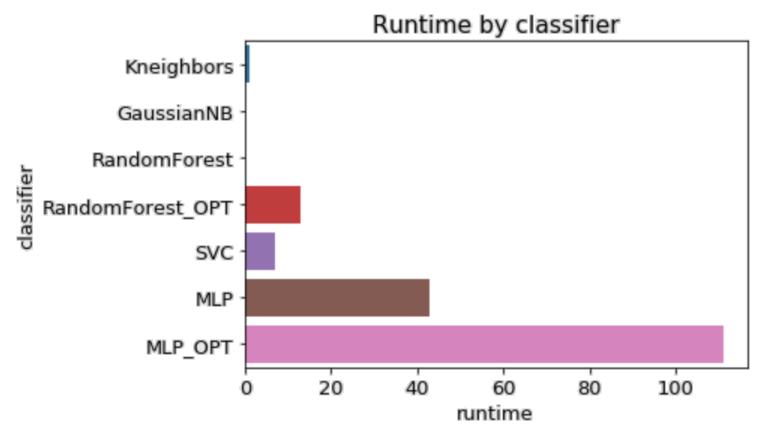
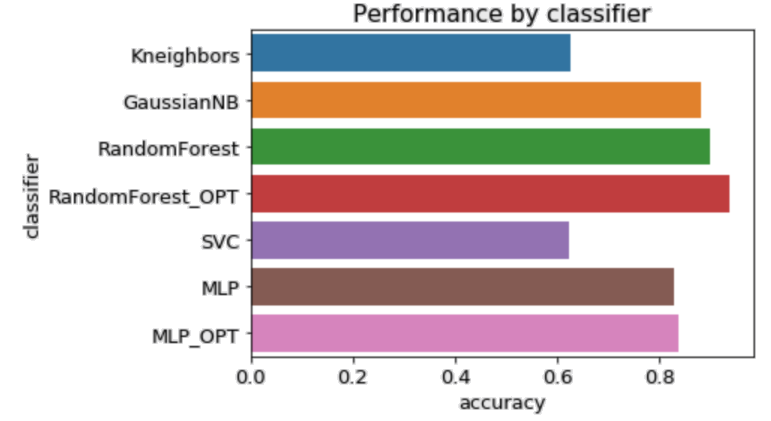


Figure 9: Runtime by classifier (SIFT, 10-fold CV) Figure 10:Performance by classifier (SIFT, 10-fold CV)

## 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.

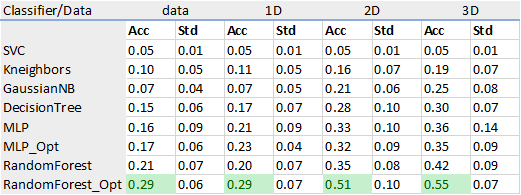


Figure 11: Performance per classifier and training set

*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.

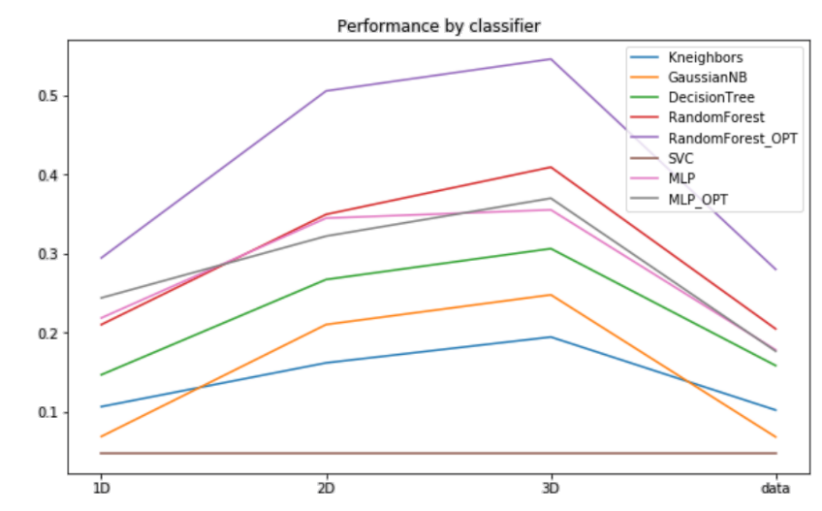
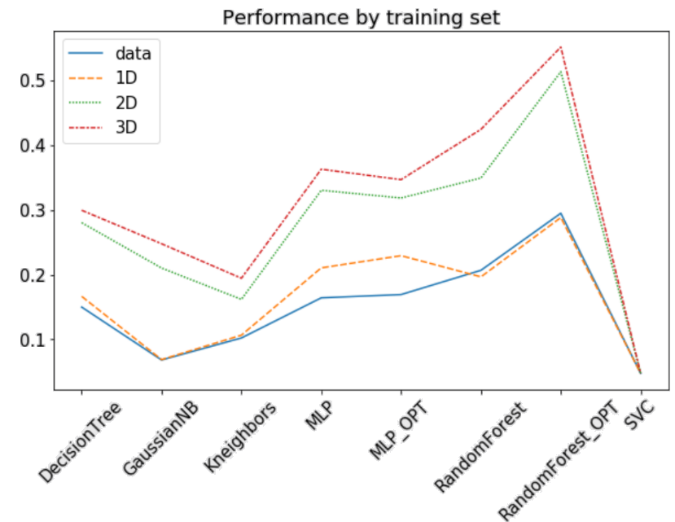


Figure 12: Performance by classifier Figure 13: Performance by training set

*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 war the slowest, but is intercepted by *RandomForest* in certain cases, due to the higher number of estimators (500).

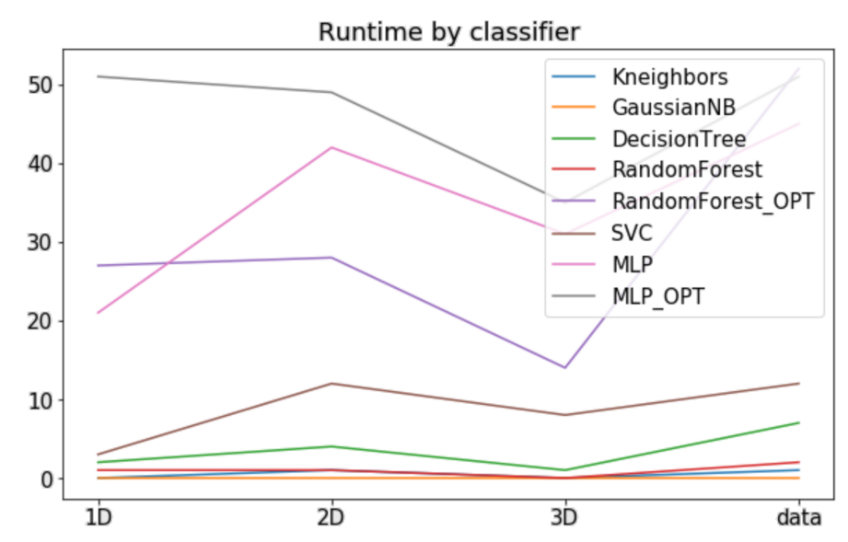


Figure 14: Runtime by classifier

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.

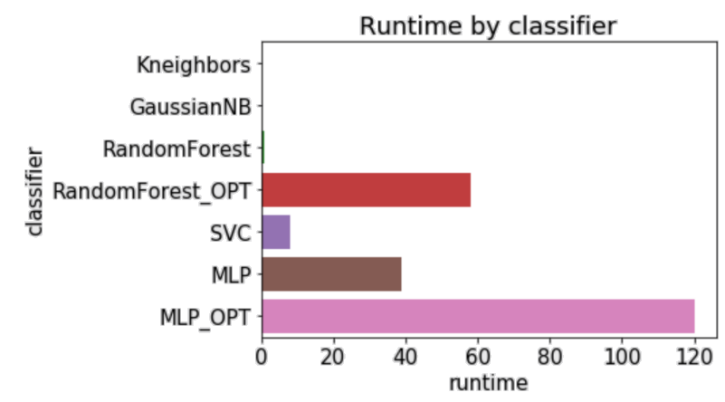
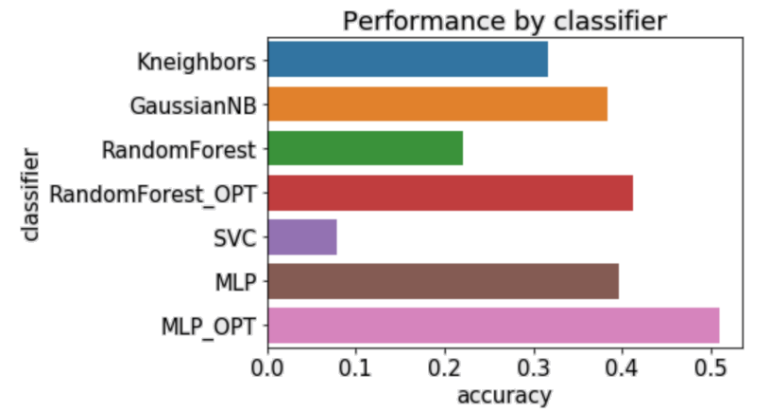


Figure 15: Performance by classifier (SIFT) Figure 16: Performance by classifier (SIFT)

## 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.

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.

## 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 trade off between runtime and achieved accuracy, if the numerous times slower pipeline is worth the improved accuracy.

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)