Pattern Recognition Team Projects Report

Hanna, Milos, Saskia

2020

1 Organization

Organization was mainly carried out through regular meetings on Discord to dispatch subtasks and synchronize progress. Although sometimes communication was difficult given the fact that we were not able to meet, we feel like we were still globally able to work well together given the circumstances. We did have a bit of an emergency situation with the MLP and CNN tasks because of run times, last minute changes, and file overwriting mistakes, but we were able to adapt and learn from these problems for the rest of the tasks.

2 Tasks

All of our implementations and results can be found on the following GitHub repository: `https://github.com/hannaglad/patternrec_2020-samh`.

2.1 Support Vector Machine (SVM)

A markdown file was created at the end of our script as our small report containing the average accuracies during cross-validation and the accuracy on the test set with the optimized parameter values.
Linear, polynomial and RBF kernels were investigated with different gamma and C values using the `GridSearchCV` cross-validation function from `sklearn`.
As optimized parameter values, we got `C = 10`, `gamma = 1e-07` with the RBF kernel.
One particular struggle was figuring out the right parameters for the RBF kernel. We can see from the results that in order to get a certain accuracy with the RBF kernel (more than 90%), we needed a C parameter above 0.1, but more importantly a gamma parameter below 0.000001. When both of these parameters are outside of that range, the accuracy with the RBF kernel drops to 0.1135 which caused us some confusion in the beginning.

2.2 Mutlilayer Perceptron (MLP)

We implemented our own version of a multilayer perceptron based on the online update mode presented in the lecture slides. We chose this method over batch as it was easier to implement and has one less hyperparameter to optimize without loosing much in terms of accuracy and performance. For the activation function, we used the sigmoid function and for the loss we used the mean square error. Due to problems with sigmoid saturation, we normalized the data to 0 to 1 scale instead of 0 to 255. For weight initialization, Xavier initialization was used to keep weight variance consistency, break symmetry and avoid them blowing up or vanishing, biases were initialized to 0 [1]. Optimal performance was obtained by starting with a learning rate of 0.1 and dropping it by half every 10 epochs. We repeated random initialization several times. Best results for the normal data can be seen in Figure 1.

These graphs are completely identical which is surprising but we realized that we used the given csv file for the training set, while we generated the validation set from the pngs which we assume is a subset of the csv train set and thus would be expected to perform the same way as the set it was taken from. On the test data accuracy achieved is: 97.93%.

[1] `https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79`

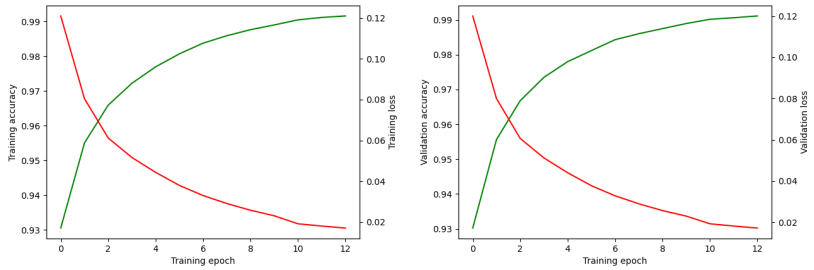

Figure 1: Accuracy and loss graph for the training and validation set.

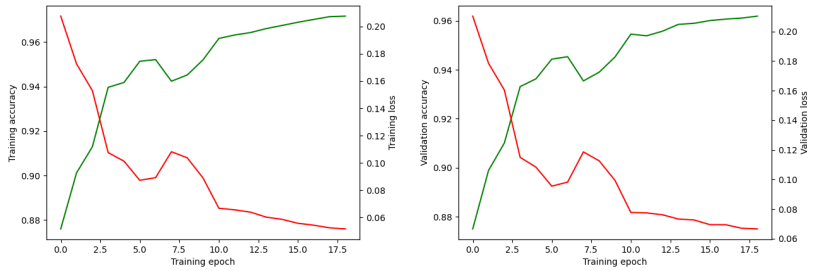

Figure 2: Accuracy and loss graph for the training and validation set of the permuted data.

The same process was performed on the permuted set and best results can be seen Figure 2.The graphs show similar trends, with the validation data having slightly less accuracy, overall demonstrating satisfactory and consistent performance. On the test data accuracy achieved is: 96.71%.

We can see that the permuted and normal data-set show approximately the same performance, while the small difference can be attributed to using the csv trainset on the normal data. This is to be expected as the main underlying mechanism of the MLP is computing dot-products which is invariant to change in the order within the data vectors, as long as it is done in the same way for both sets. Changes in the curves shapes can be attributed to the fact that we permuted the train-set order every trial. This means that samples were visited in different order every time, and since we used the online mode, this is to be expected.

2.3 Convolutional Neural Network (CNN)

We implemented a convolution neural network using the `pytorch` framework provided in the exercise. After several smaller scale trials we concluded that the architecture presented in the code gives the best results given the constraints. We used SGD again as the optimizer so that the performance can be somewhat comparable to the MLP. Results for the normal dataset are shown in Figure 3.In short, similar trends are

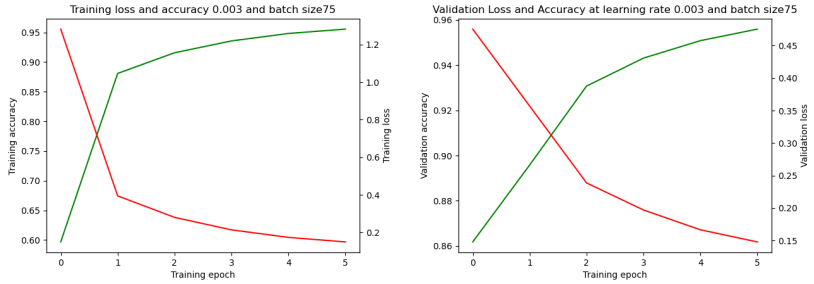

Figure 3: Accuracy and loss graph for the training and validation set.

once again observed with the validation and the training sets, with the main difference being the steepness of the curves. Final test accuracy is: 96.06%.

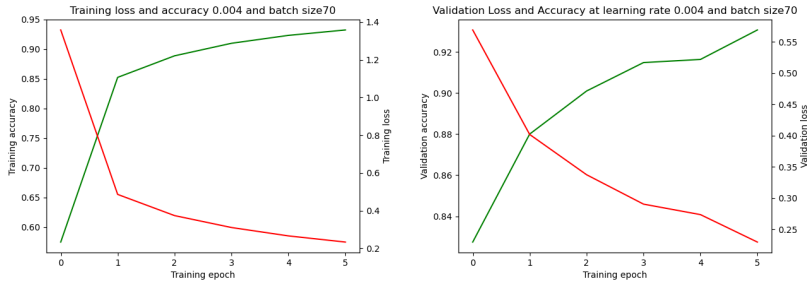

Figure 4: Accuracy and loss graph for the training and validation set of the permuted data.

We repeated the process on the permuted set and best results can be seen in Figure 4.Graphs yet again show similarities with the only difference being steepness of change. Final test acccuracy is: 0.9394.

Compared to the non-permuted dataset, CNN has a slight drop in performance. This is expected as CNN, unlike MLP, exploit spatial relationship which in this case are disrupted by permutation. The reason the performance drop is so small is that these pictures are not simple and relatively small in size. A higher drop in performance for CNN would be expected if we were to test it on more complex images, while the MLP should stay consistent.

2.4 Keyword Spotting using Dynamic Time Wraping (DTW)

Overall, the idea was to do good binarization and preprocessing so that no artifacts of the background remains to cause problems with inference. Six features described in the problem were created and then selected on the basis of whats the best combination of features that produces results of highest quality. Next, after constructing feature vector by the sliding window approach DTW was implemented as specified in the article... The main problems arose due to our misunderstanding of what was actually required for accuracy and precision.

2.5 Graph Matching Molecules

To put our data into shape, an independent script was used to create lists of graph objects, which was then saved using the `pickle` module. Next a cost matrix was specified, consisting of indels and node substitions. Deletions and insertion costs were determined by counting the number of edges and including a node indel cost. As for the substitutions, since the edges were unlabeled and indel costs were equivalent, substitution was calculated as the cost to change the node plus the difference in edge numbers.
For the Cn and Ce parameters a grid search was preformed with using KNN as the actual classifying approach and graph edit distance as the distance metric. For each parameter combination Ks were tried in the range from 1 to 20 and best result was selected.
The best accuracy of 0.996 was obtained with Cn:0.5 and Ce:1.0

3 Conclusion

Overall, these tasks were really usefull for understanding the different algorithms and allowed us to develop both coding and project management skills. It was very rewarding to see our implementations work. One comment about the tasks in particular would be for DTW, were we had a lot of trouble with image processing, given that we are all from the bioinformatics master and didn't have very much experience with this kind of thing. Obviously image processing is an important part of pattern recognition, but it sometimes felt a bit overwhelming to have to learn this on the spot with out much information available.

Figure 5: On a positive note, some of our image processing attempts turned out to be quite artistic

On a more personal note, we feel like we could have made better use of git and github. Since none of us had ever used this platform before, we weren't able to fully exploit the potential of these tools. But it has given us motivation to learn!