Introduction to Machine Learning (SS 2023) Programming Project

Author 1
Last name: Gasser
First name: Georg
Matrikel Nr.: 12105791

Author 2
Last name: Cetin
First name: Remzi
Matrikel Nr.: 12106676

I. Introduction

This project is about classification of grayscale images with a size of 128x128 pixels that resemble a letter a-z or a number 0-9 in sign language. This is a multiclass classification problem with 36 classes and we have a data set that consists of 9680 images to train our model on.

II. Implementation / ML Process

We began by reading through the instructions and then discussed about what is best applicable. We decided to use a Support Vector Machine (SVM) and a Neural Network (NN). Since we are working in a team we created a GitHub repository to share our code and work together effectively. The first thing was to label our data. We wrote a class to construct a data set useable for training. This was necessary because the images we were given were all in one folder with an additional csv with the labels for the images.
The first method we implemented was the Neural Network and we got some inspiration by a Youtube video that introduced us to pytorch and how we can implement a NN. Our Classifier is consists of 8 layers in total. We used two convolutional layers, then a Rectified Linear Unit activation function after each one aswell as a max pooling layer that reduces the spatial dimensions of the input by a factor of two. Then we flattened the tensor to a 1-dimensional tensor to use it for our final layer which is a fully connected layer with 36 output units to match our 36 classes. In the training loop our NN makes a prediction and calculates the loss between the predicted outputs and the actual labels. Then it computes the gradients of the loss with respect to the model parameters using backpropagation. When choosing our hyperparameters we did simple trial and error, we ended up at a learning rate of 0.000096. Batch size of 32 is a common value that balances between fast training but using a lot of memory with a large size and better generalization but slow training with a smaller size. Finally Cross-Entropy Loss is commonly used for multi-class classification problems, where each input sample belongs to one of several classes A Neural Network was our first thought when we read the instructions of our project. Because Neural Networks can effectively capture complexities and learn to differentiate between different signs. They are a common used machine learning method for image classification. Neural Networks can handle our large data set with almost 10,000 images and can benefit from techniques such as mini-batch training and regularization to handle the data set's size effectively.
The second method we implemented was a Support Vector Machine with the help of sklearn. We trained the classifier with grid search and cross-validation. The function "classifier.fit" takes the training data and labels as input and trains the classifier using all possible combinations of hyperparameters specified in the parameter grid for the SVM classifier. Then our scikit-learn GridSearchCV function performs a search over the specified parameter values and evaluates the performance of the classifier for each combination of parameter values. Finally we obtain the best estimator and added a accuracy function. Finally we added PCA and specified that it should retain 95% of the variance in the data to begin with and decided to stick with it since training time and accuracy were pretty good. We looked for another popular approach besides a convolutional Neural Network and a fully connected NN did not seem suitable for this problem so we came accross a SVM. Sign language images can often suffer from variations in lighting conditions, hand orientations, and background clutter. We started off with more hyperparameters but quickly saw that our training script is not terminating so we began by reducing the parameters which means a less deep grid search and added PCA to achieve faster training speed. SVMs can handle these variation and noisy data well by focusing on maximizing the margin between classes and achieve good generalization performance so we decided to go with it.

III. Results

As we can see on the graph for "Training and Validation Accuracy" for CNN, the training accuracy is higher at each epoch. But the values are more far apart the compared to the results of the SVM-method. At the beginning, the difference between the training and validation accuracy was even higher but with some methods like regularization we treated the so called phenomenon "overfitting" and got these results for accuracy using CNN. After 10 epochs of training we get an

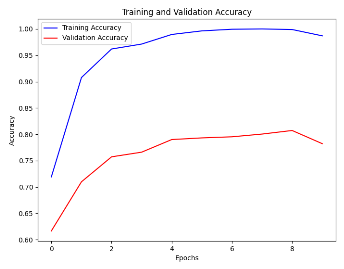

Fig. 1. CNN accuracy

accuracy of approximately 80 percent for the validation set. Another interesting aspect is that the losses during training are higher than during validation for the first epochs.
The results are better if we use SVM as we can see on the training and validation accuracies.

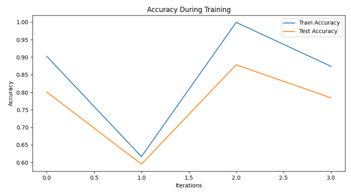

Fig. 2. SVM accuracy

Overall, we can see that the SVM-model performs less well on the training data than the CNN-model but better on the test data.
We also created a heat map to visualize the relationship between the hyper-parameters and the model performance. After performing the grid search we plotted the results to see how different hyper-parameter combinations perform. The hyper-parameter ”gamma” controls the shape of the decision boundary in the model. So, it determines the reach or influence of each training example. The regularization (hyper-)parameter ”C” is controls the trade-off between achieving a low training error and having a small-margin hyperplane. The accuracies for the test data set show that the best model has ”gamma” with 0.001 and ”C” with 1 with an accuracy of approximately 89 percent The results are getting worse if we increase both ”gamma” and ”C”, like we can see on the model with an accuracy of approximately 59 percent.

IV. Discussion

At the beginning of our project, there was a really high difference between the accuracy on the training set and the accuracy on the validation set, using CNN. At each epoch, the accuracy on the training set was nearly perfect. Already for the first epoch, the result was approximately 80 percent, whereas the validation accuracy started at approximately 60 percent. After each epoch the training accuracy increased much more than the validation accuracy. So, the model performed well on the training data set but less good on the test data set. This phenomenon is called overfitting. To overcome this, we added a dropout layer, which sets a fraction of input units to zero (50 percent in our case) during training. This can help to prevent the model from relying too heavily on specific neurons. Additionally, we reduced the learning rate to allow the model converge more slowly and potentially improve generalization, and added weight decay for regularization.
On of the reasons for higher loss during training than during validation could the fact that the regularization terms are only applied while training the model on the training set which can lead to higher losses during training. Dropout randomly freezes neurons in a layer during training. Since dropout is only applicable during the training process, it has an higher impact on the training loss which can lead to phases of lower validation loss.
The SVM-model has a better validation accuracy and a better relation between validation and training accuracies since it performs less well on the training data and it is better in generalizing to new unseen data. There are several factors why the accuracy is better using SVM. Dimensionality reduction with PCA is used where the feature space is reduced and most important information is captured. The complexity of the data is reduced which can help the SVM classifier to generalize better and avoid overfitting. Also the SVM method uses grid search which searches for the best combination of hyper-parameters to achieve the best performance. A high value for ”gamma” leads to a more complex decision boundary which leads the model to fit the training data more closely. The consequence of this is that the model overfits. A small ”gamma” makes the boundary generalized and more spread out which allows to consider points that are far away as similar. A large ”C” puts higher penalty on missclassifications. The model is much more focused to classify training data and creates a more complex decision boundary. This could cause overfitting too. But if ”C” is smaller, it allows more training errors and leads to a simpler decision boundary. So, it is important to take the best values for both ”gamma” and ”C” in combination.

V. Conclusion

The accuracy of the test-set for our CNN-method was approximately 80 percent. As assumed, it started with a low percentage at beginning but then converged to approximately 80 percent. We could see that the higher the validation loss was the lower was the accuracy, which is actually something to be expected. In the SVM-method, we can see better results if we choose smaller values for ”gamma” and regularization parameter ”C”. But of course it should be a good combination of parameters. Furthermore, the SVM-model’s accuracy on the validation set is better.
This project taught us how interesting machine learning actually is. Putting the theoretical concepts into practice makes it easier to understand the topics better, but it also required a good understanding of the theoretical part. After finding the possible methods to use to solve the task, the be-all and end-all is to find optimal hyper-parameters, layers or regularization terms. So, it is not always the first approach the best one. You have to fine tune a lot.