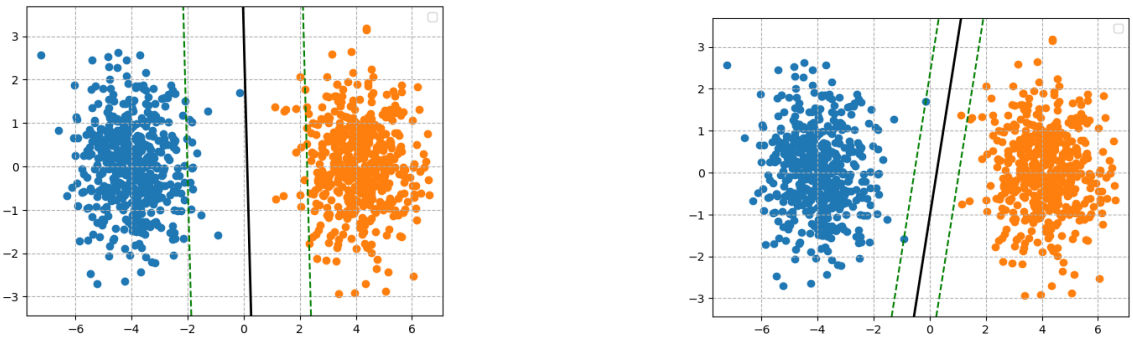
Introduction

In this assignment the main work will be on a classification task of the FashionMNIST dataset with different machine learning and deep learning models. FashionMNIST is a dataset consisting of $70,000$ data-points with each being a 28×28 grayscale image. The dataset contains 10 categories. The data-points are split in a training set, consisting of $50,000$ data-points, a validation and a test set, each of which consists out of $10,000$ data-points.

1 Part: SVM and Decision Boundaries

In this part the functionality and behavior of Support Vector Machines was studied. SVM showed a lot of useful hyperparameter that can fine-tune the model if used correctly. A very useful one that makes comparison even possible is the "random_state". Another very important one is the "kernel", it chooses the type of kernel for the given dataset. For linear tasks the linear kernel can be used, but for more complex tasks rbf is a good choice. The last one that was interesting in the experiments that were conducted was "C", this is the regularization parameter. In Figure 1 a comparison between SVM decision functions for different values of C is shown for linearly separable data. Lower C values allow more error and by thus achieve a larger margin as we can see in Figure 1a, while higher values make the model stricter which leads to smaller margins Figure 1b.

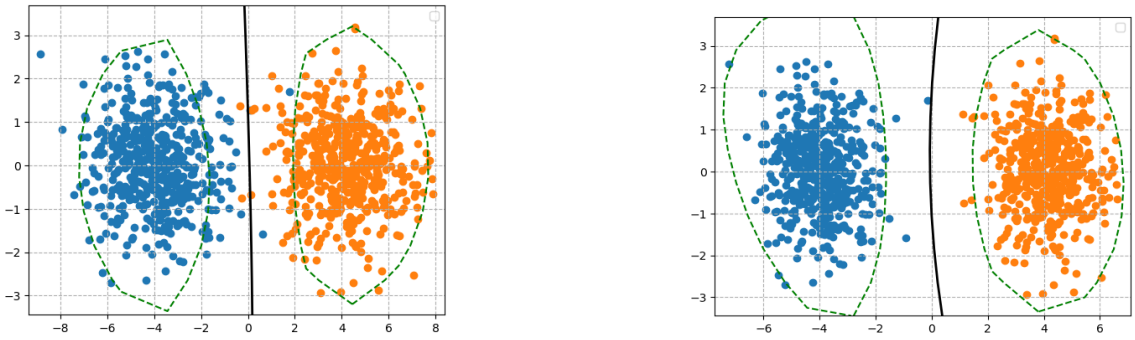


(a) C = 0.01

(b) C = 1

Figure 1: SVM decision function and its support vectors based on different C values.

As easily observable the linear case is quite simple, but non-linearly separable data is not as simple. For SVM to find a decision boundary it has to transform the data. The transformation of the data and search for the decision boundary is handled by the rbf kernel. In Figure 2 a comparison again between different C values is plotted, showcasing the influence of the regularization term. Here we can see how a higher value of C tries to fit the training points better and include more of its own points. Lower value of C has a smaller penalty for points in the margins, which leads to more points being in the margin area as we can observe in Figure 2a.



(a) C = 0.1 (b) C = 1

Figure 2: SVM decision function and its support vectors based on different C values.

2 Part: Image Classification with Fashion-MNIST

In this part different approaches for image classification on the Fashion-MNIST dataset, introduced on page 1, are explored. Most of the examined models are shallow such as Logistic Regression, but CNN is also considered, which is part of the deep learning models.

2.1 Shallow learning

In this chapter Logistic Regression, Decision Tree and the K-Nearest Neighbor classifier are going to be used for classification on the dataset. Hyperparameter optimization was performed. For Logistic Regression values for "max_iter", "penalty" and "C" were optimized, for Decision Tree values for "max_depth", "min_samples_leaf" and "ccp_alpha" were optimized, and for KNN the values for "n_neighbors", "n_jobs" and "weights" were optimized. A comparison between the models on a unseen test set can be observed in the table below:

Model	**Hyperparameter**	**Accuracy**
`Logistic Regression`	max_iter = 200 penalty = L2 C = 1	0.8404
`Decision Tree`	max_depth = 15 min_samples_leaf = 10 ccp_alpha = 0.0001	0.8049
`KNN`	n_neighbors = 7 n_jobs = 5 weights = distance	0.8504

As we can see Decision Tree was outperformed with the given hyperparameter, this is probably due to the reason, that way too many features are present in images and a big decision tree would be

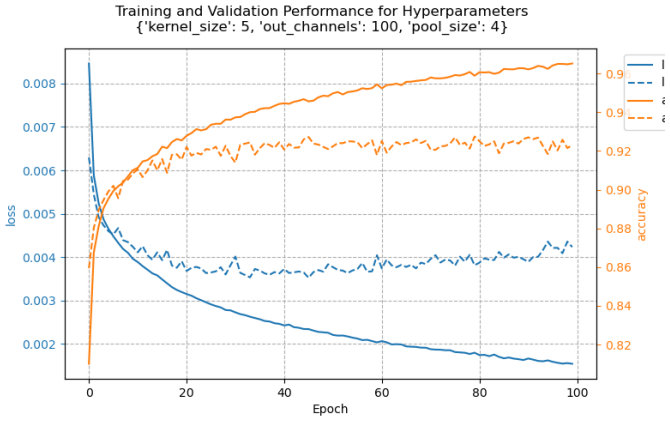


Figure 3: Performance graph on the training of the best CNN model

needed to have a stronger prediction. This would take a lot of training time and due to that reason those limiting hyperparameter were chosen.

2.2 Deep Learning

As already mentioned a CNN model was also trained and tested on the dataset. The CNN consisted of one hidden layer, it has one convolution, including padding, followed by a ReLU activation, and an average pooling. The result of the procedure is given to a linear layer, that gives the logits for the 10 classes. The model used Cross Entropy as its loss function and the Adaptive Moment Estimation optimizer. Two performance measures were used, the loss function and accuracy of the model. Hyperparameter optimization was performed, the parameters to be optimized were "kernel_size", "out_channels" and "pool_size". The model was evaluated on combinations of the following values:

$$\begin{aligned} \text{kernel\_size} &= [3, 5, 7] \\ \text{out\_channels} &= [10, 50, 100] \\ \text{pool\_size} &= [2, 4, 7] \end{aligned}$$

The chosen values for the pool size were not random, they have to be divisible by the image dimensions, since the pooling reduces the size of the image, and lets say 25 pixels cant be divided by 2. There may be ways to deal with this problem for example with padding, but this way we can ensure divisibility is possible. Side note, not all combinations of parameters were possible since training was stopped on the 10 hours mark. last training was performed on kernel size = 7, out channel = 50 and pool size = 7. With that in mind the training of the model with the best performance can be observed in Figure 3. This model managed to achieve an accuracy of 0.91870 on the test set and a loss of 0.00387.
It is possible that since this CNN architecture includes only 1 hidden layer, that if the training reached a kernel size of 7×7 it would outperform those parameters since it will have more broader

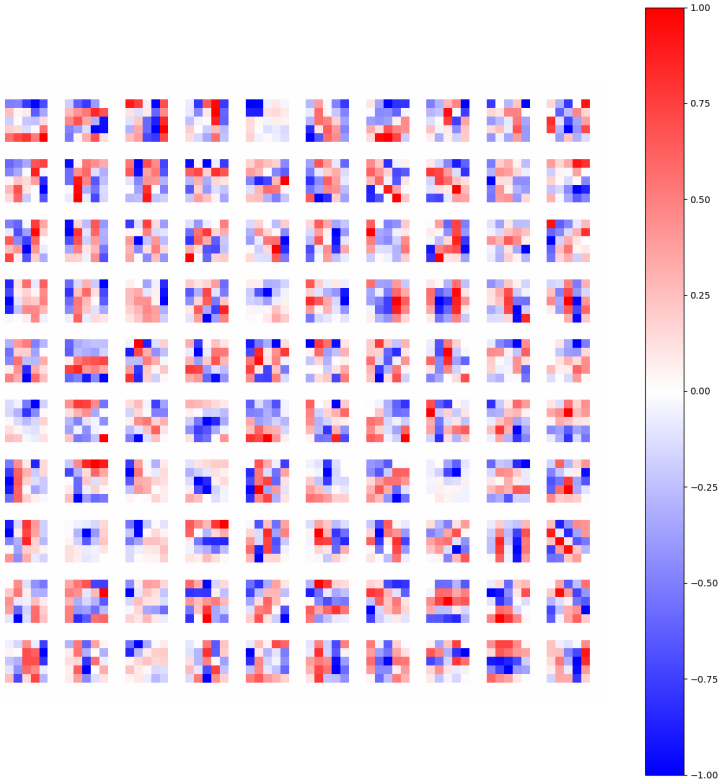


Figure 4: Caption

view of the image and thus more information. Nevertheless in Figure 4 are the kernels of the best model depicted. Here we can see that a wide variety of kernels were learned, we can see also some easily interpretable ones that are typical for diagonal, vertical or horizontal edge detection.

2.3 Shallow models on processed data

Since process the features and give them to other hidden layers, what would happen if those processed features are given to the shallow models introduced in subsection 2.1, how will it change their performance? The models are learned on this new dataset of features that the hidden layer in the CNN from subsection 2.2 constructed, then their performance is tested on the test set once again. The results can be observed in the following table:

Model	**Accuracy**
`Logistic Regression`	0.9174
`Decision Tree`	0.8271
`KNN`	0.88

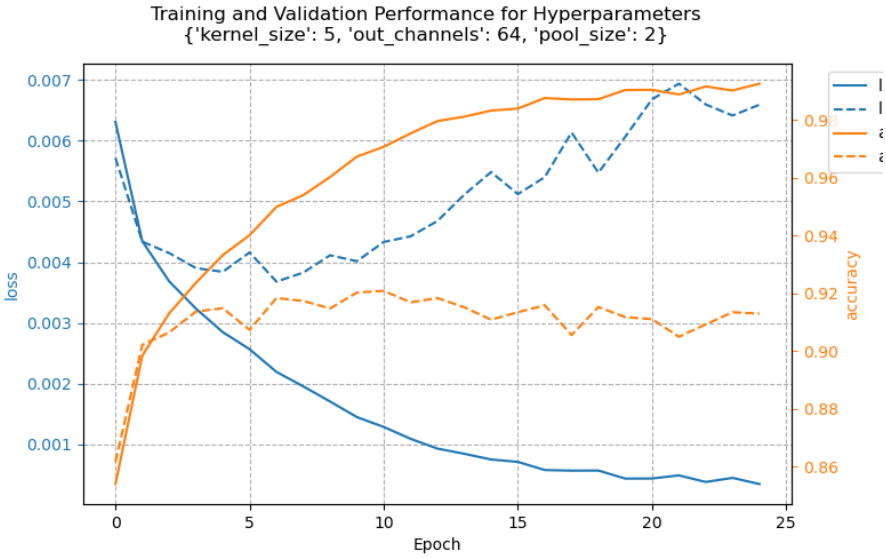


Figure 5: Performance graph on the training of the Deep CNN model with channels $16 \rightarrow 32 \rightarrow 64$

All of the models improved after being trained on the processed features. This is due to the reason that spatial information is also included in this way, and not only independent values of the features. This time Logistic Regression outperformed the rest of the models by a lot. This could be due to the reason that the linear layer of the CNN works in a similar fashion with the logits. Decision Tree had a small improvement.

2.4 Deeper Learning

In this section a new CNN was constructed with deeper layers. This CNN can have n hidden layers, as long the input is big enough. Each hidden layer consist of a convolution operation with a padding, followed by a batch normalization, a ReLU activation and a Max pooling for size reduction. After all hidden layers the features are given to a linear function that returns the logits for the prediction of the 10 classes. The best performance was achieved with 3 hidden layers with a kernel size of 5 and channel sizes of $16 \rightarrow 32 \rightarrow 64$ corresponding to each hidden layer. The model managed to achieve an accuracy of 0.91460, so the performance didn't increase in comparison to the normal CNN but a comparison can't be really done, since the architecture of both models differ. But even if a small one an improvement was observed if 3 instead of 2 hidden layers were used. This is also only up till a point, since even if each hidden layers manages to widen the receptive field we lose a lot of information that way and this can also decrease the performance of a model. This is the reason why a combination of techniques have to be used to overcome this limitation, as using residual connections or maybe upsampling in between. The model also overfit pretty fast to the training data as it can be observed in Figure 5. The conclusion of those experiments is that deep learning models can improve models but hyperparameter have to be chosen with caution.

Declaration of Used AI Tools

Tool	Purpose	Where?	Useful?
ChatGPT	Rephrasing	Part 2	++

Signatures
Mannheim, October 29th, 2025