Utrecht University

Graduate School of Natural Sciences

Assignment 4: Convolutional neural networks

Assignment for Computer Vision (INFOMCV)

Nikola Grigorov (5081629)
Ying-Kai Dang (9389903)

Contents

29th March 2023

1 Description and motivation of baseline model and variants

1.1 Baseline model

We are using the sparse categorical cross-entropy loss function since we have multiple classes, represented as integers.

As activation functions, we're using ReLU for all convolutional and dense layers, except for the last dense layer, where we are using a SoftMax activation function. We chose ReLU because we are sure to not have the vanishing gradient issue.

Our architecture consists of 3 convolutional layers with max-pooling between them, which are followed by two dense layers. The model summary is seen in Table 1.

1. conv2d_1 `Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1))`: The first layer is a 2D convolutional layer with 32 filters with a kernel size `(3,3)`. This also serves as input layer and takes an input shape `(28, 28, 1)`, which represents a grayscale image with the resolution 28x28 pixels.
2. max_pooling_2d_1 `MaxPooling2D((2, 2))`: This layer performs max pooling over the convoluted layer with a pool size of `(2,2)`. This reduces the spatial dimensionality while keeping the most salient features.
3. conv2d_2 `Conv2D(64, (3, 3), activation='relu')`: This is identical to the first layer but with 64 filters instead, to attempt to be able to learn more complex and diverse patterns.
4. max_pooling_2d_2 `MaxPooling2D((2, 2))`: Max pooling again to reduce the spatial dimensions.
5. conv2d_3 `Conv2D(64, (3, 3), activation='relu')`: Same as conv2d_2 - more features!
6. flatten `Flatten()`: This time, no max-pooling is used because our spatial dimensionality is already very small, 3x3 pixels. Reducing the height and width further would lead to significant loss of information. Thus, we flatten the output of the conv2d_3 layer instead to preserve our spatial information and allow the following fully connected dense layers to have access to a larger feature map.
7. dense_1 `Dense(64, activation='relu')`: We chose to use two dense layers because we need to have one dense layer for the output (10 classes) and another one to introduce some more non-linearity. This dense layer has 64 units.
8. dense_2 `Dense(10, activation='softmax')`: This is the output layer with 10 units, representing the 10 classes. The softmax activation function (as opposed to ReLU in the other layers) is used to output the probability of each class.

Layer (type)	Output Shape	Parameter #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36,928
flatten (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36,928
dense_2 (Dense)	(None, 10)	650
Total params: 93,322		
Trainable params: 93,322		
Non-trainable params: 0		

Table 1: Output of `model.summmary()` for the baseline model.

1.2 Variants

We wanted to explore different options in reducing overfitting in our base model with our variants. All four variants try to do that using a different strategy.

1.2.1 V1: Dropout

```
tf.keras.layers.Dropout(.25)
```

This variant implements a dropout layer with 25% chance to drop the neuron weight. It is placed after the second convolution layer and before the max-pooling. This is because this convolution layer has 64 filters (the first one has 32) and it is more likely to overfit the data. Dropout will reduce the correlation between neurons there.

1.2.2 V2: L1 Regularizer

```
tf.keras.layers.Dense(64, activation='relu',kernel_regularizer=tf.keras.regularizers.l1
    (0.001)),
tf.keras.layers.Dense(10, activation='softmax' , kernel_regularizer=tf.keras.
    regularizers.l1(0.001))
```

This variant implements a different regularizer. L1 is placed on both the dense layers since they are responsible for transforming the input features and the output labels. L1 applies a penalty to the loss function which could potentially allow the model to generalize better. L1 regularization tends to produce models where many weights are set to zero, basically selecting the features that are most important to the model, which is why we expect this variant to improve on the base model.

1.2.3 V3: L2 Regularizer

```
tf.keras.layers.Dense(64, activation='relu',kernel_regularizer=tf.keras.regularizers.l2
    (0.001)),
tf.keras.layers.Dense(10, activation='softmax' , kernel_regularizer=tf.keras.
    regularizers.l2(0.001))
```

Similar to the L1 variant, adding L2 to both dense layers makes most sense. L2 prevents overfitting by chipping away at the weights at each iteration, essentially smoothing out the model. This can be more effective when the features are correlated, which we expect to be in this dataset.

1.2.4 V4: Batch Normalization

```
tf.keras.layers.BatchNormalization()
```

A batch normalization is added after first convolution layer conv2d_1. It stabilizes the learning process, normalizes values to 0 and 1, depending on whichever they are closer to using mean and the standard deviation, and solves internal covariate shift. Since we only add one batch norm layer, putting it after the first conv layer the best place since it normalizes the first input of the network.

2 Training and validation loss

Figure 1: Loss and accuracy for the baseline model

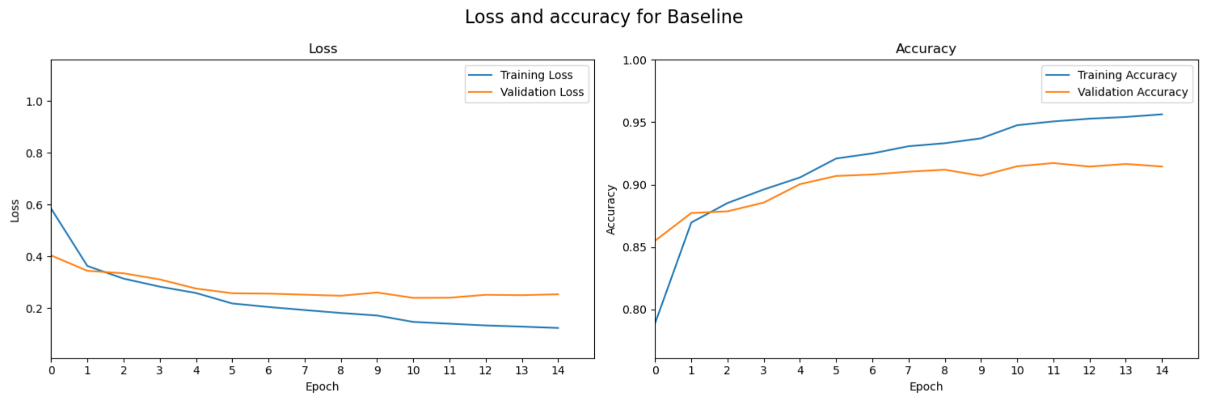


Figure 2: Loss and accuracy for variant 1, dropout

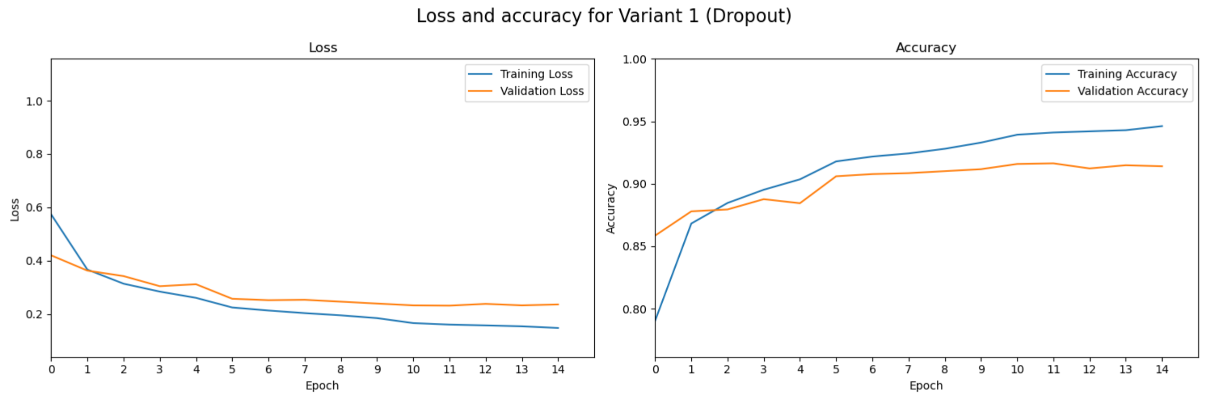


Figure 3: Loss and accuracy for variant 2, L1 regularization

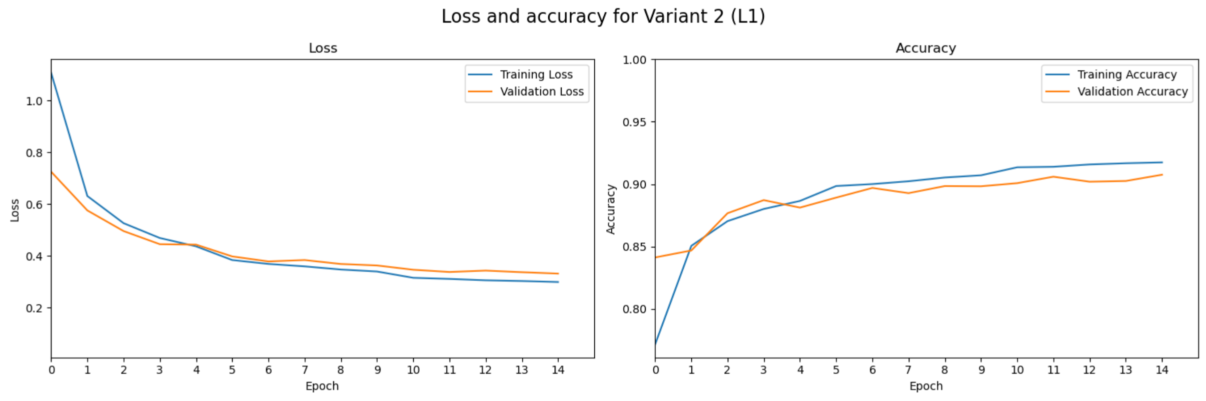


Figure 4: Loss and accuracy for variant 3, L2 regularization

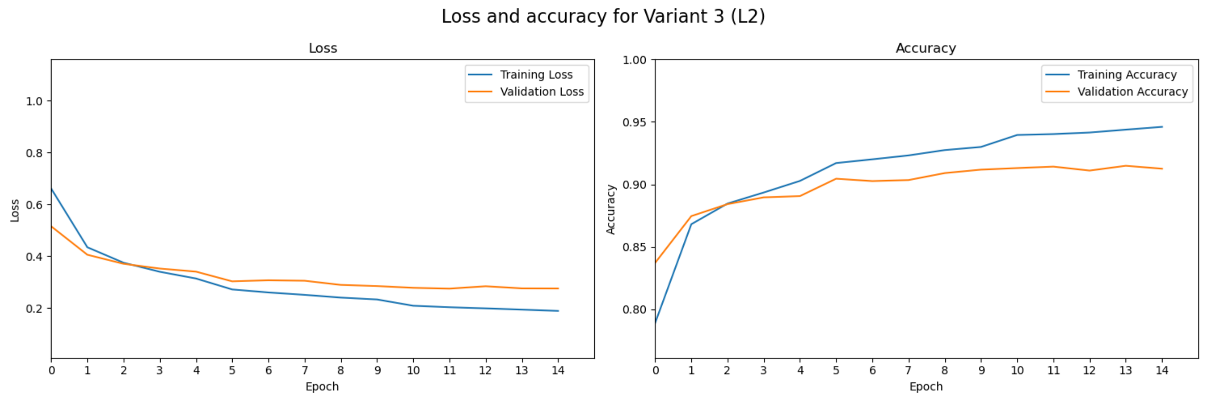
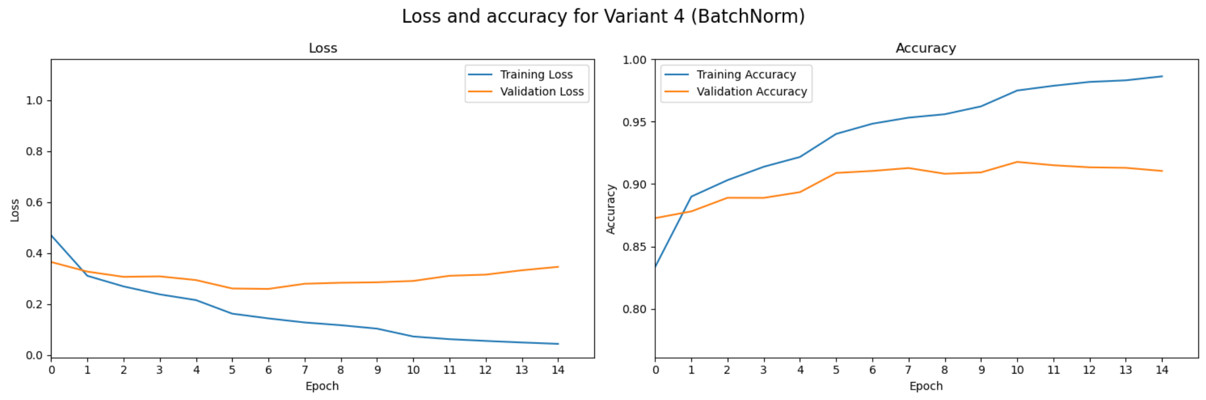


Figure 5: Loss and accuracy for variant 4, batch normalization



3 Link to model weights

`https://github.com/FunnyPocketBook/infocv/raw/ass4/ass4/model_weights/model_weights.zip`

4 Training and validation top-1 accuracy

Model name	Training top-1 accuracy (%)	Validation top-1 accuracy (%)
Baseline	0.9564	0.9146
Variant 1 (Dropout)	0.9461	0.9140
Variant 2 (L1 Regu)	0.9175	0.9076
Variant 3 (L2 Regu)	0.9460	0.9126
Variant 4 (BatchNorm)	0.9864	0.9216

Table 2: Top-1 accuracy of training and validation

5 Results

5.1 Base - Variant 1 (Dropout)

Variant 1 has a slightly more complex architecture than the base model, with an additional dropout layer after the second convolutional layer. The variant is better than the baseline on the validation dataset but worse on the training dataset. This suggests that the base model is overfitting slightly.

5.2 Base - Variant 2 (L1 Regularizer)

Variant 2 adds complexity by introducing an additional hyperparameter as L1 regularization on the dense layers. This variant performs slightly worse on the validation data but a lot worse on the training data. Our assumption is that it cannot converge fast enough in 15 epochs, or the model underfits and reducing the strength could help. Further fine-tuning the hyperparameter could potentially improve the model within 15 epochs.

5.3 Base - Variant 3 (L2 Regularizer)

Variant 3 adds complexity by introducing an additional hyperparameter as L2 regularization on the dense layers, similar to variant 2. However, this variant performs slightly better than the base model in terms of validation accuracy, with a lower validation loss. There is also a lower risk of overfitting, as the training loss is lower than the base model. Similar to variant 2, fine-tuning the hyperparameter could improve the result.

5.4 Base - Variant 4 (Batch Normalization)

Variant 4 adds a batch normalization layer after the first convolutional layer. It performs a lot better than the base model in terms of training and validation accuracy. However, the loss is the highest of all models, which in combination with the high accuracy strongly suggests overfitting. Adding a dropout layer and regularizers could benefit this model.

6 Difference between two models on test set

Model name	Test accuracy (%)	Test loss (%)
Baseline	0.9178	0.2650
Variant 4 (BatchNorm)	0.9079	0.4229

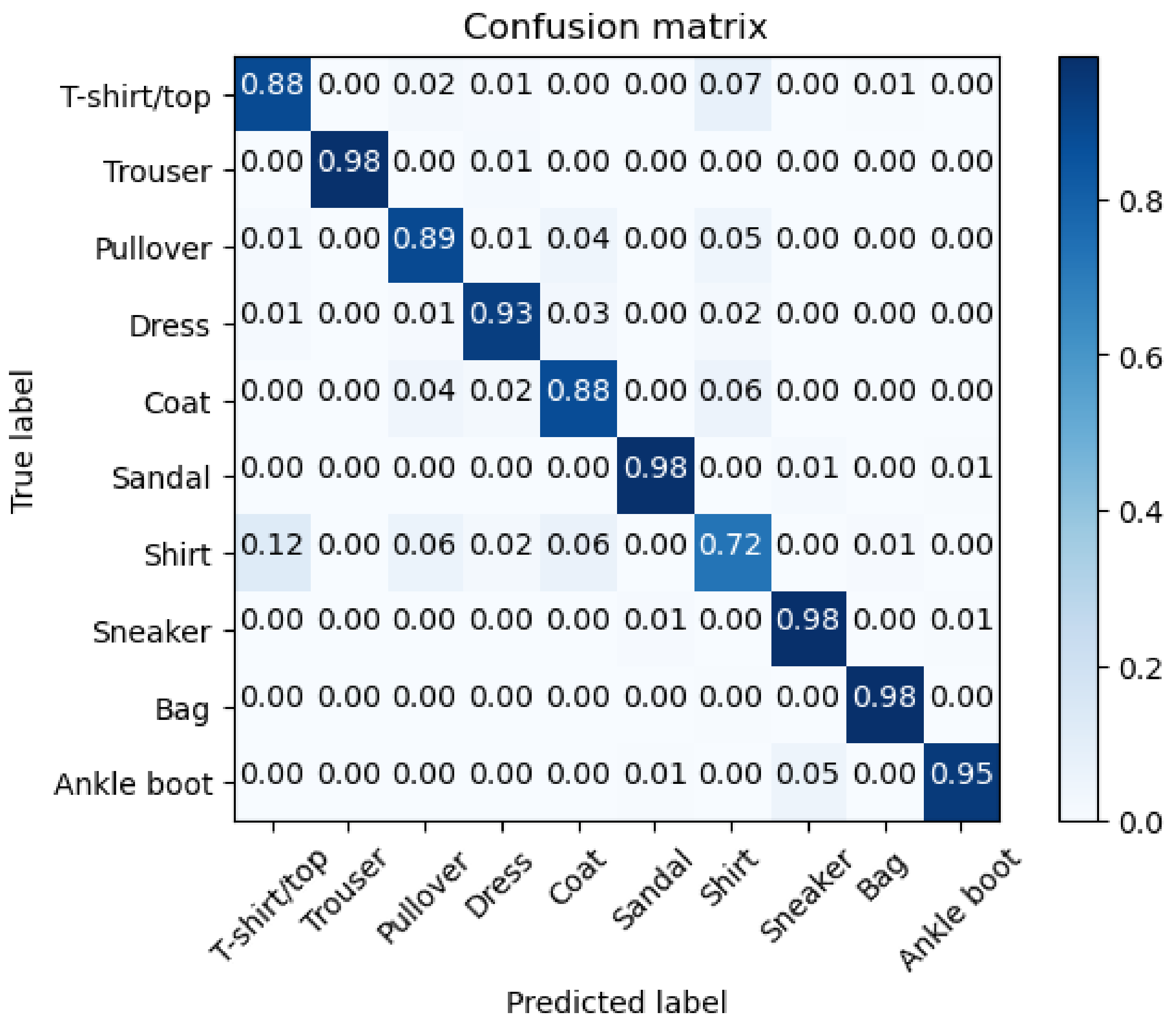
Table 3: Performance of two best models on test set

As seen in the comparison already, the base model performed decently and did not show big signs of overfitting. Batch normalization can also help prevent overfitting. However, it is possible that the use of batch normalization in variant 4 is not having a significant impact on the model's performance, or is even slightly hurting performance in this particular case. During training, the model achieved a high training accuracy of 0.9896 and a low loss of 0.0324. However, the test loss is 0.4229, which highly suggest strong overfitting of the model on the training data. While batch normalization is usually used for preventing overfitting (if used correctly), in our case it introduced overfit. Apart from overfitting, a couple other reasons as to why this could happen is because of the generally small training size. It is possible that the model can't converge fast enough into the correct labels from features and batch normalization doesn't have enough time to estimate the variance accurately. Similarly, batch normalization acts sort of like a regularizer and we already had regularization established within the model. We believe that adding this extra layer with batch normalization might have been overkill. If we take a look at the confusion matrix 6, we can see that our biggest mislabeling happens at "Shirt" and "T-Shirt" since batch normalization is intended to solve the internal covariate shift, in theory you could say that it probably could slow down the network's ability to learn the difference between these two images and label them appropriately. Finally, the baseline model was already pretty good at estimating the images. Even our best performing variants, the accuracy didn't increase that much on the training set.

7 Choice tasks

7.1 Choice 1: Confusion matrix

In this confusion matrix we see that the baseline model is worse at classifying top-clothing, such as shirts and coats, than non-top clothing, such as trousers and shoes. Particularly, we see that shirts have been mislabeled the most, with an accuracy of only 72%. 12% was misclassified as t-shirt/top, which is not a big surprise for us, because even we weren't able to correctly distinguish between shirts and t-shirts/tops. Since their appearances are very similar, we expect the model to have some errors there.



True label / Predicted label	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	0.88	0.00	0.02	0.01	0.00	0.00	0.07	0.00	0.01	0.00
Trouser	0.00	0.98	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
Pullover	0.01	0.00	0.89	0.01	0.04	0.00	0.05	0.00	0.00	0.00
Dress	0.01	0.00	0.01	0.93	0.03	0.00	0.02	0.00	0.00	0.00
Coat	0.00	0.00	0.04	0.02	0.88	0.00	0.06	0.00	0.00	0.00
Sandal	0.00	0.00	0.00	0.00	0.00	0.98	0.00	0.01	0.00	0.01
Shirt	0.12	0.00	0.06	0.02	0.06	0.00	0.72	0.00	0.01	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.98	0.00	0.01
Bag	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	0.00
Ankle boot	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.05	0.00	0.95

Figure 6: Confusion matrix of results of baseline model on test set

Actual / Predicted	Positive	Negative
Positive	TP	FP
Negative	FN	TN

Table 4: Confusion matrix, binary classification

7.2 Choice 2: Decaying learning rate

```python
def lr_scheduler(epoch, lr):
    if epoch % 5 == 0 and epoch > 0:
        lr = lr / 2
    return lr
```

```

lr_decay = tf.keras.callbacks.LearningRateScheduler(lr_scheduler)
model.fit(..., callbacks=[lr_decay])
```

7.3 Choice 3: k-fold cross-validation

Average accuracy: 0.9160
Average loss: 0.2510

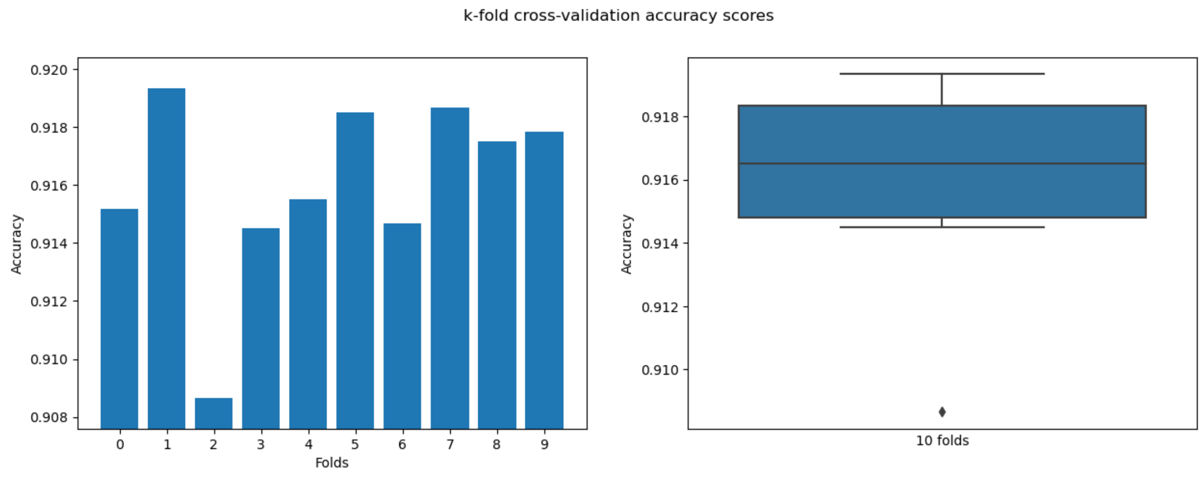


Figure 7: Accuracy over all folds in 10-fold cross-validation on the baseline model

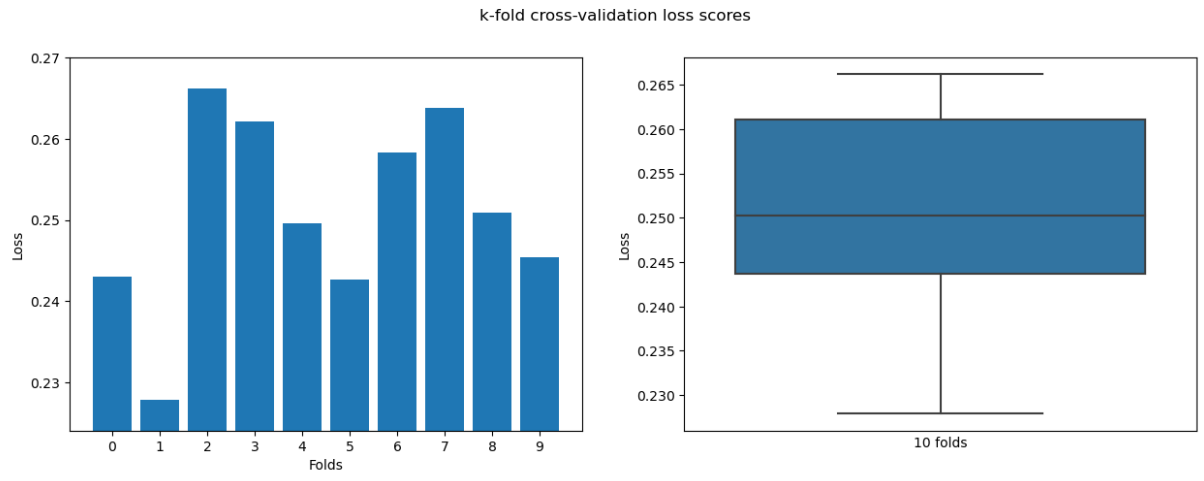


Figure 8: Loss over all folds in 10-fold cross-validation on the baseline model