Assignment 2

The Network The neural I have made consists of the following layers: first convolutional layers which has kernel size of 3*3, the first layer has padding = 1. The reason as to why I added padding to the first layers is because padding helps preserving information about the input volume, so the network can extract the low-level features of the images.

All the conv layers have a batch normalization layer All of the conv layers have a ReLU activation function which returns 0 for all negative inputs or returns the input if it is greater than 0. All the conv layers also have a maxpool2d layer which extracts the highest value features from a 2x2 filter.

The convolutional layers look like this:

First conv layer, input: 28 * 28 * 3, output: 28 * 28 * 50.

First maxpool2d layer, input:28 * 28 * 50, output:14 * 14 * 50.

Second conv layer input 14 * 14 * 50, output: 12 * 12 * 100.

Second maxpool2d layer, input: 12 * 12 *100, output: 6 * 6 *100.

Third conv layer, input: 6 * 6 * 100 output 4 * 4 * 200.

Third maxpool2d, input: 4 * 4 * 200, output:2 * 2 * 200

After the convolutional layers comes the linear layers. The linear layers output 10 different features because there are 10 different clothing classes in the dataset.

Training

I tried tweaking all the different parameter until I got a good result from the training.

When it comes to the zero padding the first convolutional layer, the model seemed to produce the best results with padding = 1, the test score decreased with a higher padding.

In the picture to the right I have batch size of 100, (1 -> 16, 16 -> 32, 32 -> 64) epoch of 6 and a learning rate of 0,001.

```
Epoch: 0 | Loss: 0.5174 | Train Accuracy: 81.49
Epoch: 1 | Loss: 0.3095 | Train Accuracy: 88.84
Epoch: 2 | Loss: 0.2701 | Train Accuracy: 90.25
Epoch: 3 | Loss: 0.2490 | Train Accuracy: 90.97
Epoch: 4 | Loss: 0.2271 | Train Accuracy: 91.70
Epoch: 5 | Loss: 0.2111 | Train Accuracy: 92.31
```

I also tried using the sigmoid activation function instead of the ReLU activation function which resulted in an accuracy decrease of almost 3%. Increasing the input/output channels with the sigmoid function applied made the test accuracy decrease a lot more. There was a difference in the training process with the two activation functions. The sigmoid started low on the training accuracy but only ended up with 2% less training accuracy than the ReLU function. The Sigmoid function did overfit, getting about 88% accuracy on the test set while training with the ReLU function resulted in 90% test accuracy.

```
Epoch: 0 | Loss: 0.7336 | Train Accuracy: 73.95
Epoch: 1 | Loss: 0.4009 | Train Accuracy: 85.59
Epoch: 2 | Loss: 0.3433 | Train Accuracy: 87.69
Epoch: 3 | Loss: 0.3102 | Train Accuracy: 88.91
Epoch: 4 | Loss: 0.2878 | Train Accuracy: 89.71
Epoch: 5 | Loss: 0.2694 | Train Accuracy: 90.29
```

Figure 2:Training process with the sigmoid function

```
Epoch: 0 | Loss: 0.5174 | Train Accuracy: 81.49
Epoch: 1 | Loss: 0.3095 | Train Accuracy: 88.84
Epoch: 2 | Loss: 0.2701 | Train Accuracy: 90.25
Epoch: 3 | Loss: 0.2490 | Train Accuracy: 90.97
Epoch: 4 | Loss: 0.2271 | Train Accuracy: 91.70
Epoch: 5 | Loss: 0.2111 | Train Accuracy: 92.31
```

Figure 1:Training process with the ReLU function

Increasing the nn.dropout2d probability seemed to make the model overfit less to the training set, However, increasing the probability to anything higher than 0.5 would negatively impact the test accuracy. The training process shown in the screenshot resulted in a test score of 90%.

```
Epoch: 0 | Loss: 0.4364 | Train Accuracy: 85.01
Epoch: 1 | Loss: 0.2653 | Train Accuracy: 91.13
Epoch: 2 | Loss: 0.2152 | Train Accuracy: 93.04
Epoch: 3 | Loss: 0.1805 | Train Accuracy: 94.28
Epoch: 4 | Loss: 0.1596 | Train Accuracy: 94.97
Epoch: 5 | Loss: 0.1525 | Train Accuracy: 95.26
```

Figure 3:Training Process with nn.dropout2d2(0.5)

Removing the batch normalization layers required me to increase the epoch to get about the same test result as training the model with batch normalization layers. Trying to increase the epochs without removing the batch normalization layers just made the model overfit.

Increasing the batch sizes, input/output channels and learning rate seemed to increase the test accuracy. Increasing batch size to 500 seemed to be the perfect size for the model because everything higher than 500 seemed to not work as well. As for the learning rate, the perfect number seems to be 0,002 for my model. I tried the following input/output channels

for my network (1 -> 32, 32 -> 64, 64 -> 120), (1 -> 50, 50 -> 100, 100 -> 200), (1 -> 100, 100 -> 200, 200 -> 400), with (1 -> 50, 50 -> 100, 100 -> 200) having the best results.

Lastly, I Tried I compared the SGD optimizer against the Adam optimizer. The SGD had significantly lower training accuracy than the Adam optimizer. I was able to get over 90% test accuracy with the SGD optimizer, however that came at a cost. I had to increase the epoch to 30 to get 87% test accuracy with the SGD optimizer. Clearly Adam is the best optimizer for this model. The screenshots below show the difference between the two optimizers.

```
Epoch: 0 | Loss: 1.8716 | Train Accuracy: 53.50
Epoch: 1 | Loss: 1.3042 | Train Accuracy: 71.11
Epoch: 2 | Loss: 1.0160 | Train Accuracy: 73.84
Epoch: 3 | Loss: 0.8630 | Train Accuracy: 75.09
Epoch: 4 | Loss: 0.7720 | Train Accuracy: 76.03
Epoch: 5 | Loss: 0.7114 | Train Accuracy: 76.86
Epoch: 6 | Loss: 0.6672 | Train Accuracy: 77.65
Epoch: 7 | Loss: 0.6327 | Train Accuracy: 78.44
Epoch: 8 | Loss: 0.6044 | Train Accuracy: 79.20
Epoch: 9 | Loss: 0.5803 | Train Accuracy: 80.05
Epoch: 10 | Loss: 0.5592 | Train Accuracy: 80.80
Epoch: 11 | Loss: 0.5403 | Train Accuracy: 81.54
Epoch: 12 | Loss: 0.5232 | Train Accuracy: 82.26
Epoch: 13 | Loss: 0.5076 | Train Accuracy: 82.82
Epoch: 14 | Loss: 0.4931 | Train Accuracy: 83.38
Epoch: 15 | Loss: 0.4796 | Train Accuracy: 83.90
Epoch: 16 | Loss: 0.4670 | Train Accuracy: 84.39
Epoch: 17 | Loss: 0.4552 | Train Accuracy: 84.82
Epoch: 18 | Loss: 0.4442 | Train Accuracy: 85.24
Epoch: 19 | Loss: 0.4338 | Train Accuracy: 85.66
Epoch: 20 | Loss: 0.4241 | Train Accuracy: 86.06
Epoch: 21 | Loss: 0.4150 | Train Accuracy: 86.38
Epoch: 22 | Loss: 0.4065 | Train Accuracy: 86.65
Epoch: 23 | Loss: 0.3984 | Train Accuracy: 86.96
Epoch: 24 | Loss: 0.3909 | Train Accuracy: 87.22
Epoch: 25 | Loss: 0.3837 | Train Accuracy: 87.44
Epoch: 26 | Loss: 0.3770 | Train Accuracy: 87.68
Epoch: 27 | Loss: 0.3706 | Train Accuracy: 87.90
Epoch: 28 | Loss: 0.3646 | Train Accuracy: 88.11
Epoch: 29 | Loss: 0.3588 | Train Accuracy: 88.31
```

Figure 4:Training the model with SGD optimizer, resulting in a 87% test score

```
Epoch: 0 | Loss: 0.5206 | Train Accuracy: 81.73
Epoch: 1 | Loss: 0.2860 | Train Accuracy: 90.44
Epoch: 2 | Loss: 0.2416 | Train Accuracy: 92.08
Epoch: 3 | Loss: 0.2101 | Train Accuracy: 93.10
Epoch: 4 | Loss: 0.1886 | Train Accuracy: 93.94
Epoch: 5 | Loss: 0.1690 | Train Accuracy: 94.62
```

Figure 5: training the model with the Adam optimizer resulting in a 90% test score

Final learning parameter

Here is the list the final parameters for my model and the learning parameter.

- Input/output channels: (1 -> 50, 50 -> 100, 100 -> 200)
- Zero padding: 1
- Batch normalization layers
- Nn.maxpool2d: 2
- Batch size: 500.
- Learning rate: 0.002
- epoch: 6
- nn.dropout2d: 0.5
- Optimizer: Adam optimizer

With these parameters the model got a test accuracy of 90%. One observation I made is that it was har to keep the model from overfitting to the training data, and it was challenging to get more than 90% test accuracy.

The model for the network can be found here: https://github.com/olekaamodt/INFO284-Machine-Learning/tree/master/assigment2