Image analysis - Lab 2

Adam Breitholtz

February 13, 2019

1 Answers

2.3

For the split of the data the Pareto principle was used. Thus 80% was used for training and 20% for validation. The elements were shuffled as they were ordered in the cell array.

2.6

As we see in 2, the increase in the s-parameter increases the spread of the weights which also seems to be almost gaussian in its distribution. This spread decreases as s decreases; although when s=0.01 where we see a slight bias toward negative weights and the previous distribution is no longer visible. This is probably due to the fact that we want to find a weights close to zero so that the gradients do not become too large and in turn the weights do not fluctuate too much. This so the convergence to the correct set of weights is not inhibited.

2.8

The data augmentation For the training data, I was able to achieve 95% accuracy. For the validation data, I was able to achieve 90% accuracy surprisingly.

2.9

These results are with the augmented dataset. For the training data, I was able to achieve 95.94% accuracy. For the validation data, I was able to achieve 93.75% accuracy.

2.14

How many parameters does the basic CNN network contain? We can roughly count the number of parameters of fully connected layers by multiplying the input size with the output size. For the convolutional layer we calculate it by multiplying the amount of filters by the size of each filter window. Thus the

convolutional layer has 5*5*20=500 parameters while the fully connected layer has 14*14*20*10=39200 parameters.

2.16

For the network that we consider we have an input of some size, say 28*28. Then the input to the blue layer is 28*28*10 and for the red layer the input has been max-pooled twice and as such is 7*7*10, this since we assume the stride to be 2. We have here used the formula for the output width and height after pooling which is

$$W_{out} = (W_{in} - N)/S + 1$$

where N is the filter size, W_{in} is the input width and S is the stride. We may also assume that the convolution is non-separable and as such the complexity of one filter is $\mathcal{O}(M * M * N * N)$, where M is the input size and N the filter size. Note that we have here used that the sizes are squares. So the complexity of the blue layer is

$$\mathcal{O}(28 * 28 * 10 * 5 * 5) = \mathcal{O}(196000)$$

and for the red layer

$$\mathcal{O}(7 * 7 * 20 * 5 * 5) = \mathcal{O}(24500).$$

Thus we can easily calculate that the red layer is about 8 times faster. If the stride is taken to be 1 instead, the input sizes are then 28*28 and 20*20 and the relationship in complexity then changes to the red layer being about 2% slower than the blue layer.

2.17

What then happens when we change a layer with 10 5*5 filters with two layers with 10 3*3 filters?

The amount of parameters are changed as follows:
Before: 10*5*5*10=2500 **After**: 10*3*3*10+10*3*3*10=1800
So the number of parameters are less than before.

We knew from the previous exercise that the complexity before was

$$\mathcal{O}(28 * 28 * 10 * 5 * 5) = \mathcal{O}(196000).$$

And after it is simply

$$\mathcal{O}(2 * 28 * 28 * 10 * 3 * 3) = \mathcal{O}(141120).$$

Thus the complexity should also be less, but it is of the same order of magnitude still.

The accuracy may suffer from this approach as the filter may not be able to resolve larger features that the larger filter may have been able to "see" due to its larger size.

2.18

The following options were used for the network.

```
options=trainingOptions('sgdm', ...
    'MaxEpochs',20,...
    'LearnRateSchedule','piecewise',...
    'InitialLearnRate',0.1,...
    'LearnRateDropFactor',0.2,...
     'LearnRateDropPeriod',5);
```

The precision and recall of the different classes are contained in the tables below.

	Precision	Recall
0	0.9950	1.0000
1	0.9951	0.9951
2	0.9855	0.9903
3	0.9898	0.9848
4	0.9949	1.0000
5	0.9874	0.9949
6	0.9949	0.9850
7	0.9976	0.9976
8	0.9820	0.9744
9	0.9899	0.9899

Table 1: Precision and recall for the training-dataset.

	Precision	Recall
0	0.9897	0.9796
1	0.9684	0.9892
2	0.9659	0.9551
3	1.0000	0.9524
4	1.0000	0.9823
5	0.9369	0.9811
6	0.9505	0.9505
7	0.9756	0.9877
8	0.9450	0.9450
9	0.9623	0.9714

Table 2: Precision and recall for the validation-dataset.

2.19

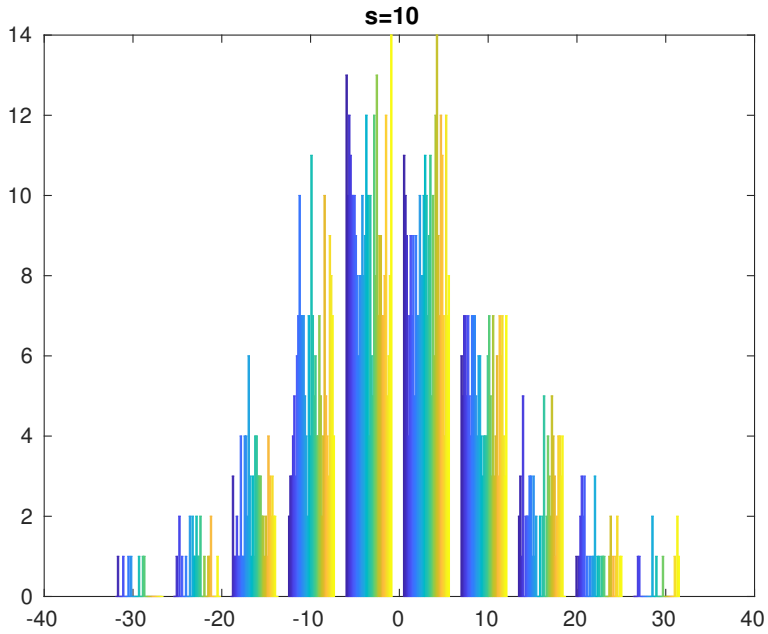
The precision and recall of the different classes are contained in the table below.

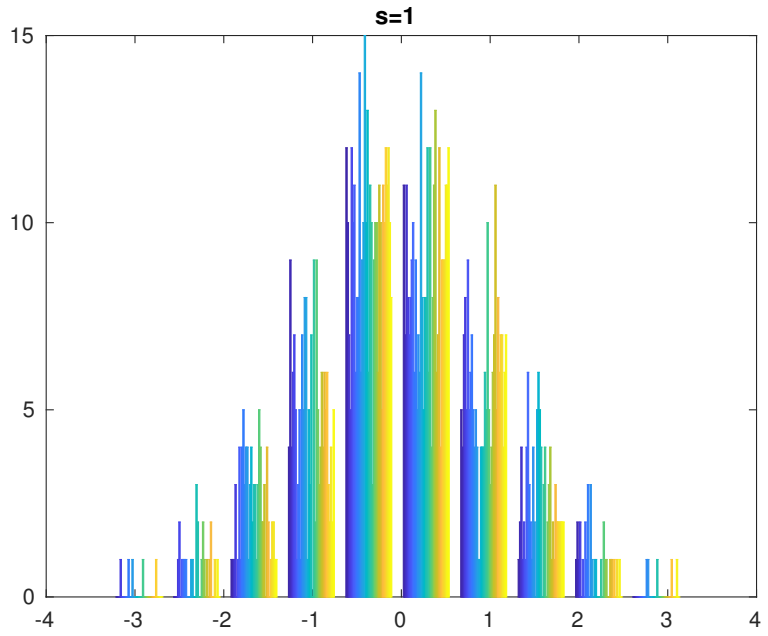
	Precision	Recall
0	0.9880	0.9900
1	0.9858	0.9720
2	0.9605	0.9720
3	0.9737	0.9640
4	0.9980	0.9940
5	0.9612	0.9920
6	0.9733	0.9480
7	0.9880	0.9860
8	0.9560	0.9560
9	0.9723	0.9820

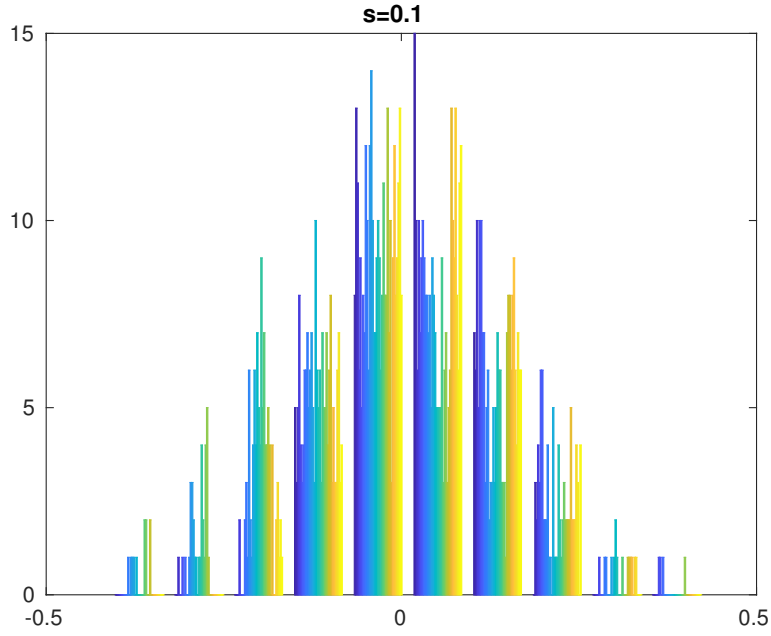
Table 3: Precision and recall for the test-dataset.

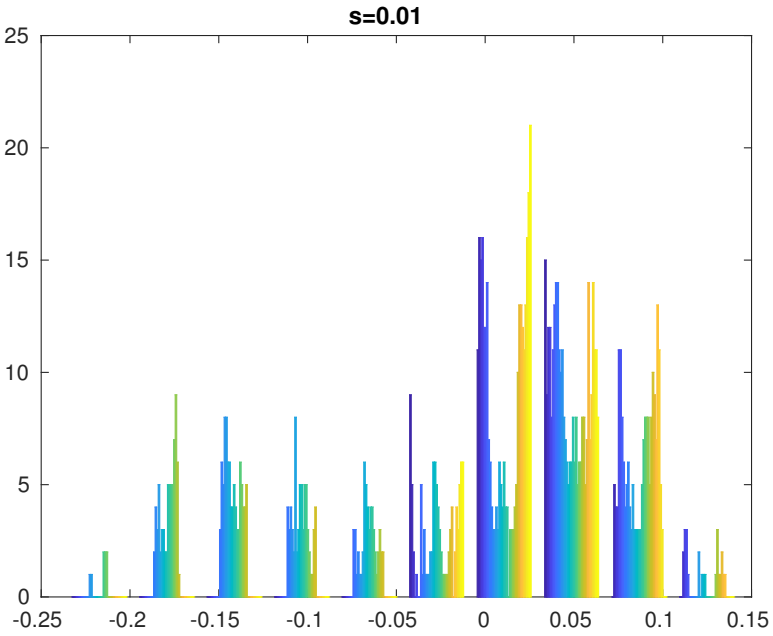
2 Images

2.6




s=1
15
10
5
0
-4
-3
-2
-1
0
1
2
3
4


s=0.1
15
10
5
0
-0.5
0
0.5


s=0.01
25
20
15
10
5
0
-0.25
-0.2
-0.15
-0.1
-0.05
0
0.05
0.1
0.15

2.7

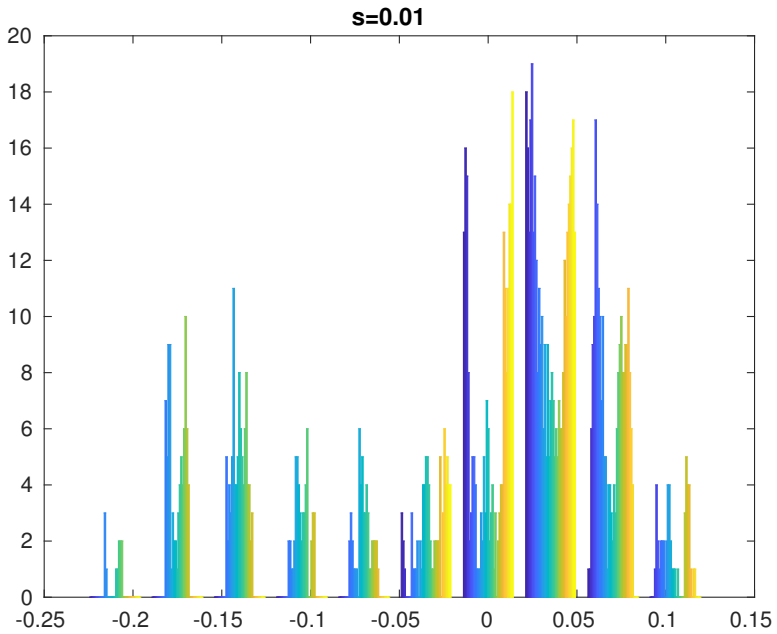


Figure 1: The result of iteratively training the classifier on the data.

2.20

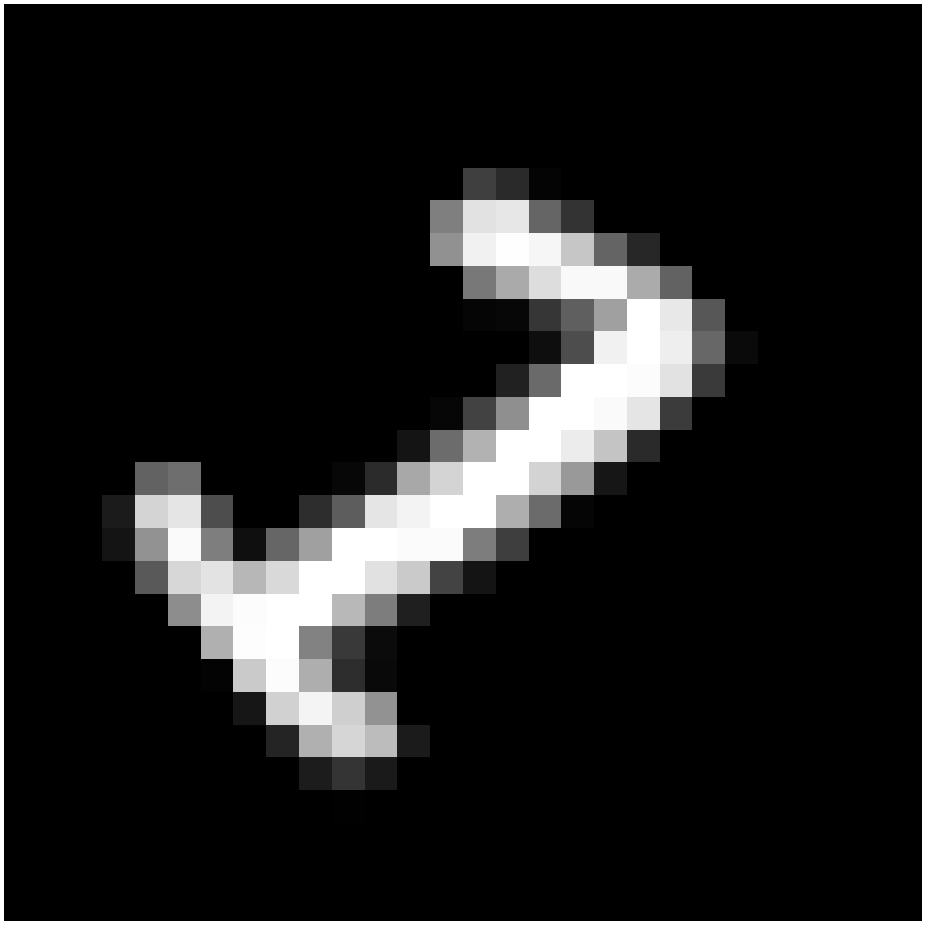


Figure 2: Mistaken for a 2

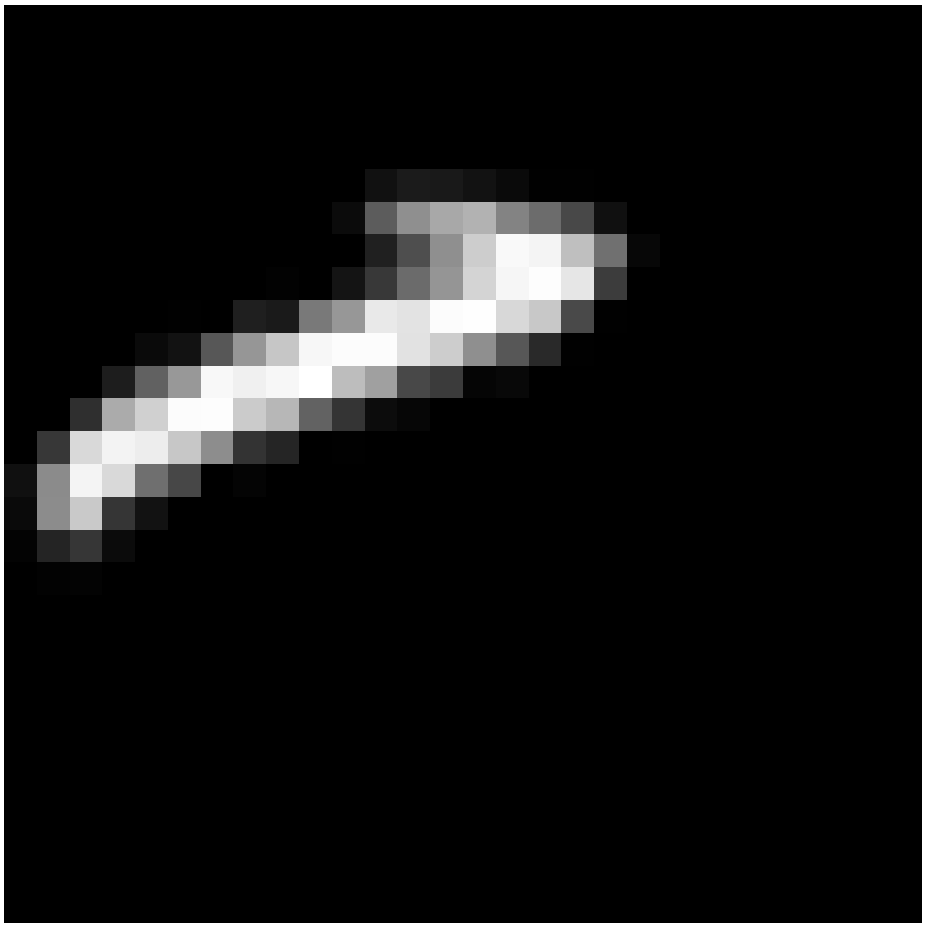


Figure 3: Mistaken for a 7

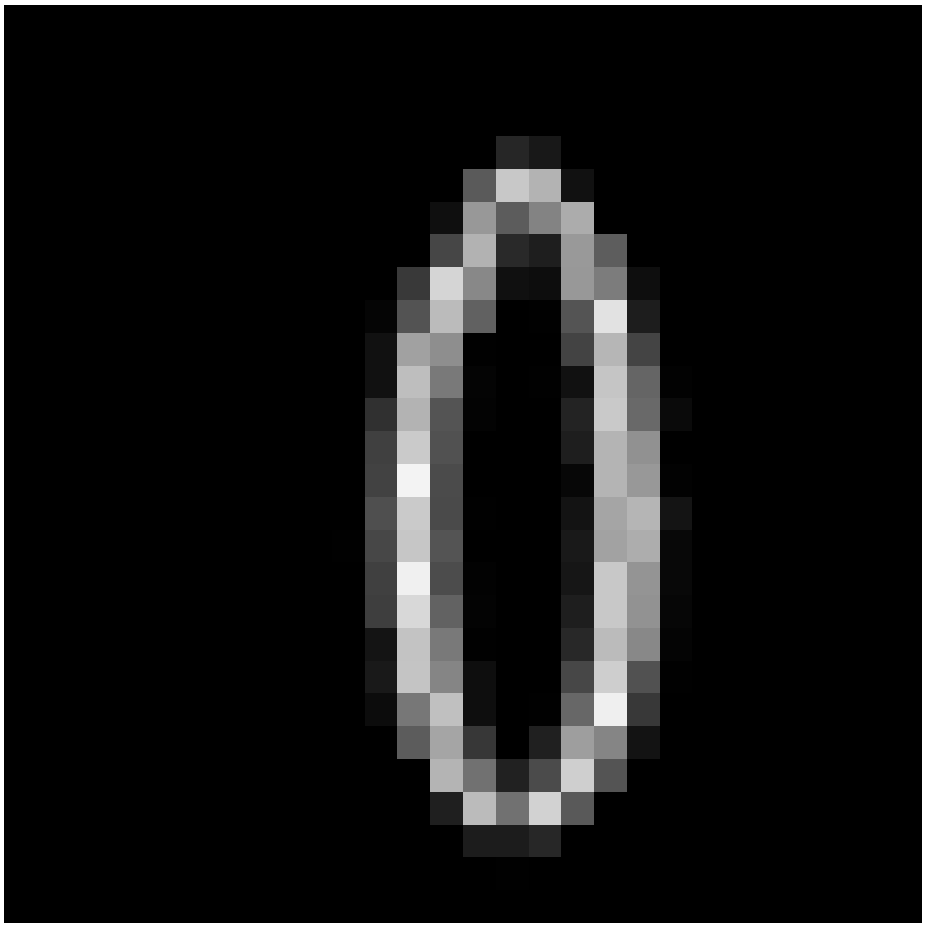


Figure 4: Mistaken for an 8