1 The architectures

1.1 Simple Architecture

In this first simple architecture, we simply provide the network a tensor containing both images in two different channels and pass it through the convolutional part of the network. At the end of this part we reshape the tensor to a 1d tensor and pass it through two successive linear layers to make the decision. The figure 1 shows an implementation of this principle using three convolutional layers.

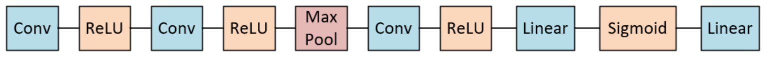

Figure 1: Simple Architecture

In the beforehand scenario, the network doesn't take into account any information regarding the structure of the input, for example the fact that the input consists of two images containing numbers. This can be accounted for by using weight sharing as shown in the next part.

1.2 Weight Sharing

By using the structure of the input data, we can create two parallel paths to process each image separately and compare them afterwards. As both paths should train to recognize handwritten digits, it makes sense to share the weights between the two convolutional parts to achieve better performance.

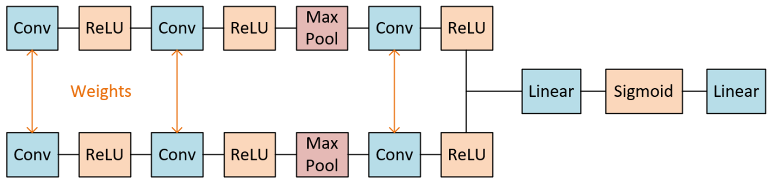

Figure 2: Addition of weight sharing

After the convolutional part, we merge the two outputs together and make a decision. By doing this, we don't train the network to recognise the numbers them self, but only the combination of the images. This can be done using auxiliary losses.

1.3 Auxiliary losses

The goal of auxiliary losses is to optimise some parts of the network to a specific task, in this case we can train the network to recognise numbers after the convolutional layers. Again we use weight sharing in those parts as they do the same job of recognising digits.

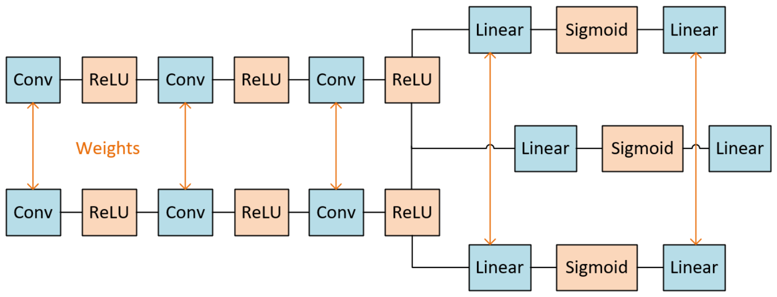

Figure 3: Addition of auxiliary losses

We expect this architecture to work better than the two previous ones as we use the information regarding the structure of the input data.

2 Results

2.1 Simple Architecture

We tried different networks with one, two and three convolutional layers and keeping the number of parameters constant and there is no change in performance. By using the parameters and architecture in our code, with 25 epochs, a minibatch size of 50 and 1000 train samples, we get an error rate of 20.95% with a standard deviation of 0.6241.

2.2 Weight Sharing

Again, the same way as before we tried different depths of convolutional layers, but this time we got better results by using three layers instead of one. Again, we kept the number of parameters constant.

# of layers	% of error	std deviation
1	21.55	0.7382
2	18.62	0.8626
3	17.82	0.9028

If we plot the output of the network in a plane (one hot output), we get the following graphs.

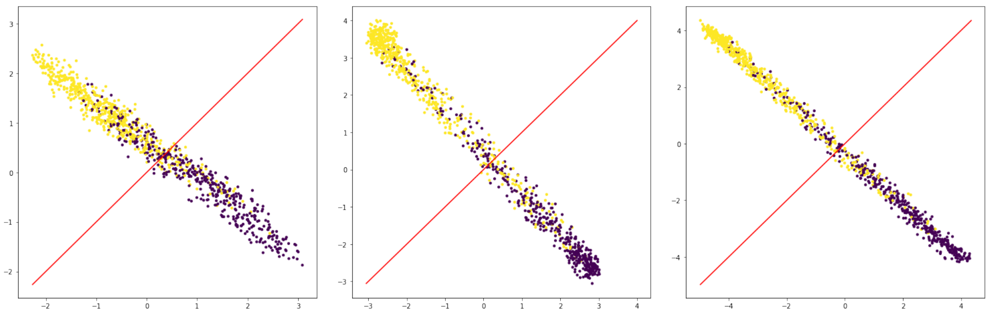

Figure 4: Network output with one, two and three convolutional layer

In these graphs, the color of each point corresponds to the correct label and the red line is the decision boundary. In the ideal case fo a 100% success rate we would have a yellow cluster on top and a purple cluster on the bottom. We can observe that with more layers, the output is spread further apart and thus tends to separate the clusters better.

2.3 Auxiliary Losses

As said before, with the auxiliary losses we expect the best results.

# of layers	% of error	std deviation
1	26.85	3.35
2	14.62	0.9461
3	12.38	0.7208

And indeed we get the best results by using three convolutional layers, weight sharing and auxiliary losses with only 12.4% of error. We can also notice that with only one layer, the result is worse than all of the tests above. This is due to the fact that the network isn't able to recognize the digits properly and thus influences the loss incorrectly.

2.4 Comparison

If we have a look at the plots of the outputs, we can observe that the further the network is able to spread the points, the better the performance.

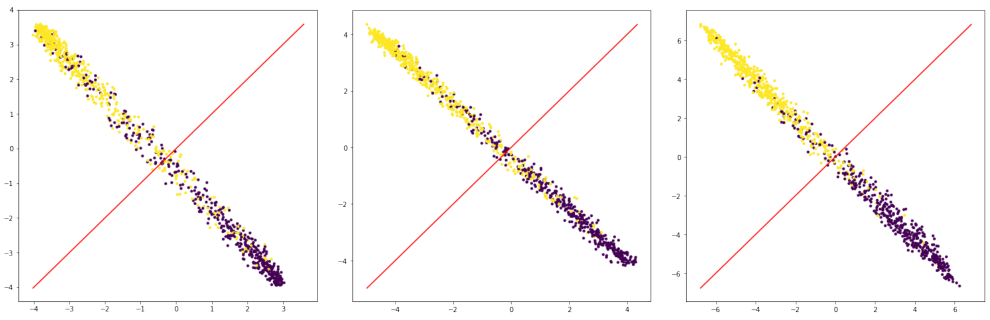

Figure 5: Network output with simple, weight sharing and auxiliary loss architecture with three convolutional layers each