DL-LAB Exercise 4 Solution

Name: Shashank Agnihotri Matriculation Number: 4775315
email: agnihotr@tf.uni-freiburg.de
Github repository: www.github.com/shashankskagnihotri/DL-LAB

Exercise 2.1:

In the Vanilla Run the code was running well, could be optimized. I tried running with different epochs and found the default value =12 to be the best fit. Generating the CNN was very tough for me but I fought my way through it. The graph of the learning curve for vanilla run look as follows:

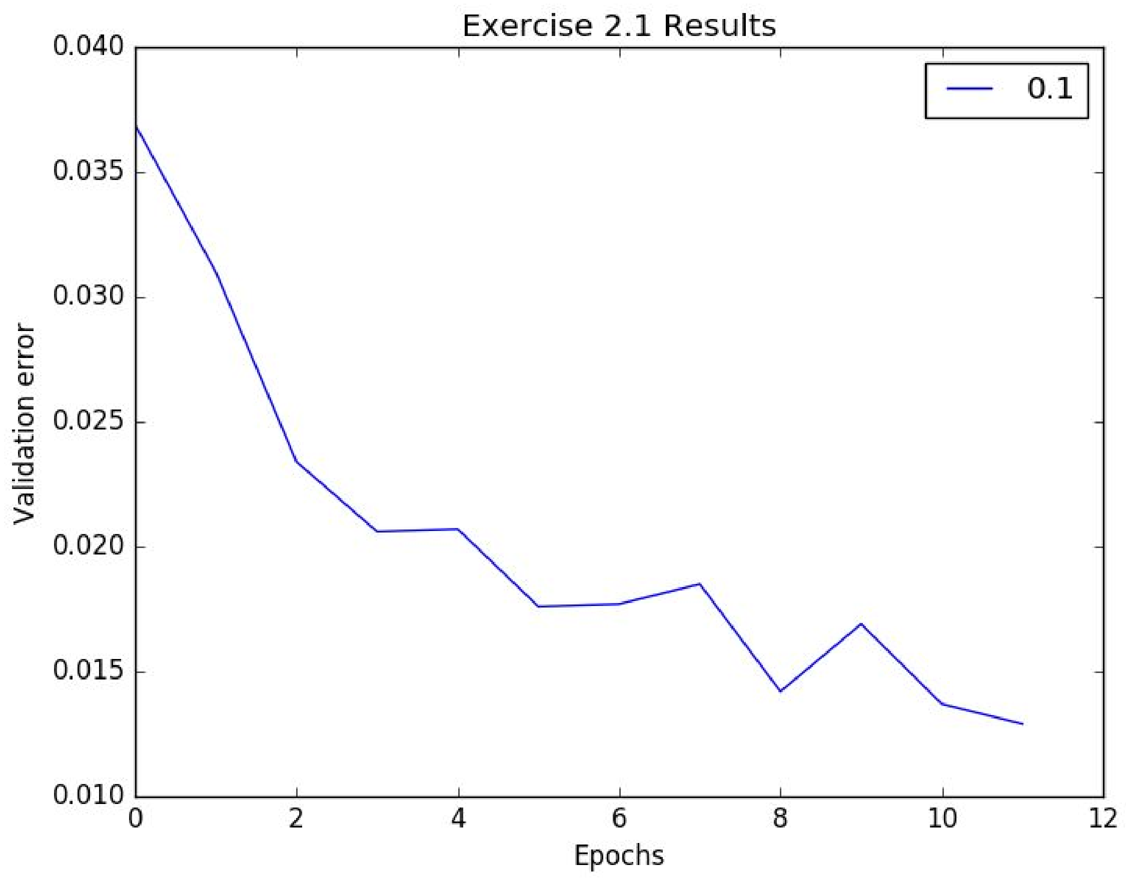


Figure Number: 1

I tried different approaches, used softmax function and then coupled it with sigmoid function with relu as the activation function which kind of gave the best results.

Exercise 2.2:

The plot for the learning curves is as follows:

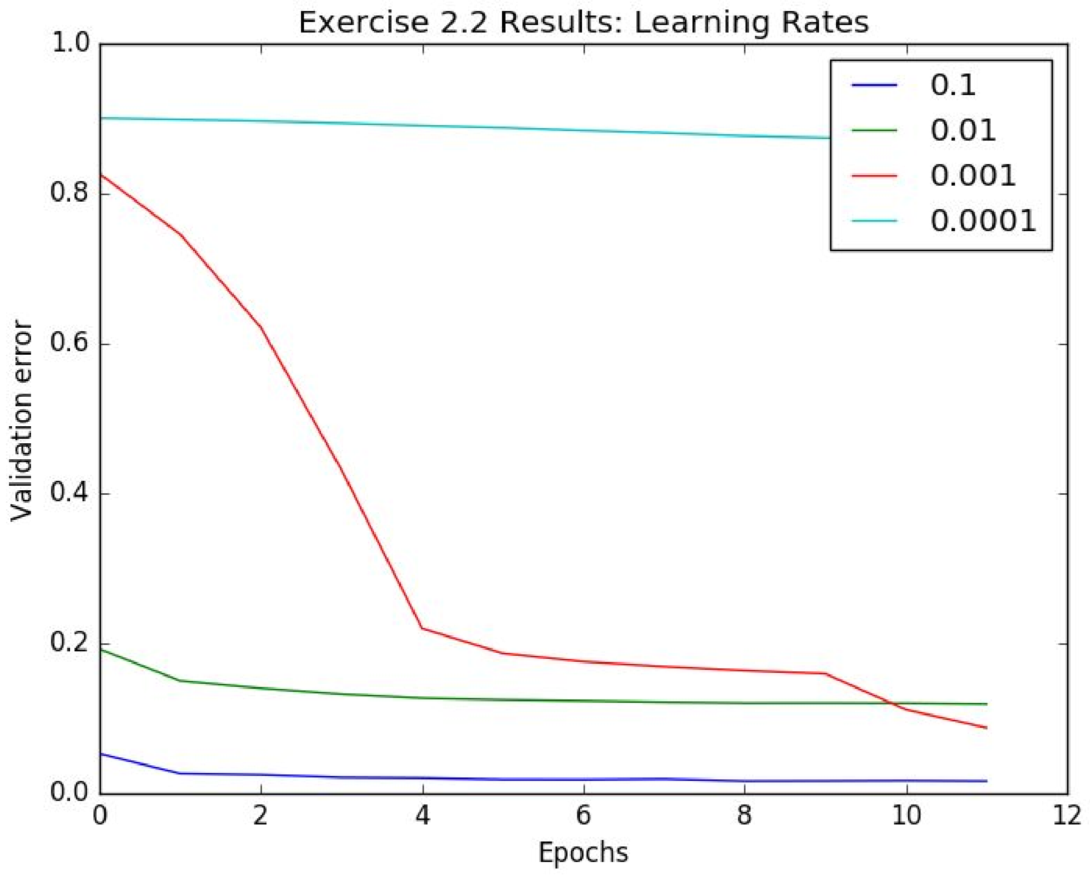


Figure Number: 2

It can be clearly seen from the graph that the learning rate 0.0001 is too small for learning and the model is not able to learn anything or only negligible something over 12 epochs. Maybe this could give good results over 100 or more epochs but that will come with significant computation costs.

It seems that the leaning rate 0.1 and 0.01 train the best until the 11th epoch when the learning rate 0.001 overtakes the 0.01 learning curve. The conclusion that can be drawn from this is 0.1 is nearest to the best results. As discussed above when the learning rate is too small the model is underfitting extensively.

And in this case when the learning rate is too high the model is performing well but, by previous experiences, that might not always be the case.

Exercise 2.3:

The learning curves for the given filter sizes is as follows:

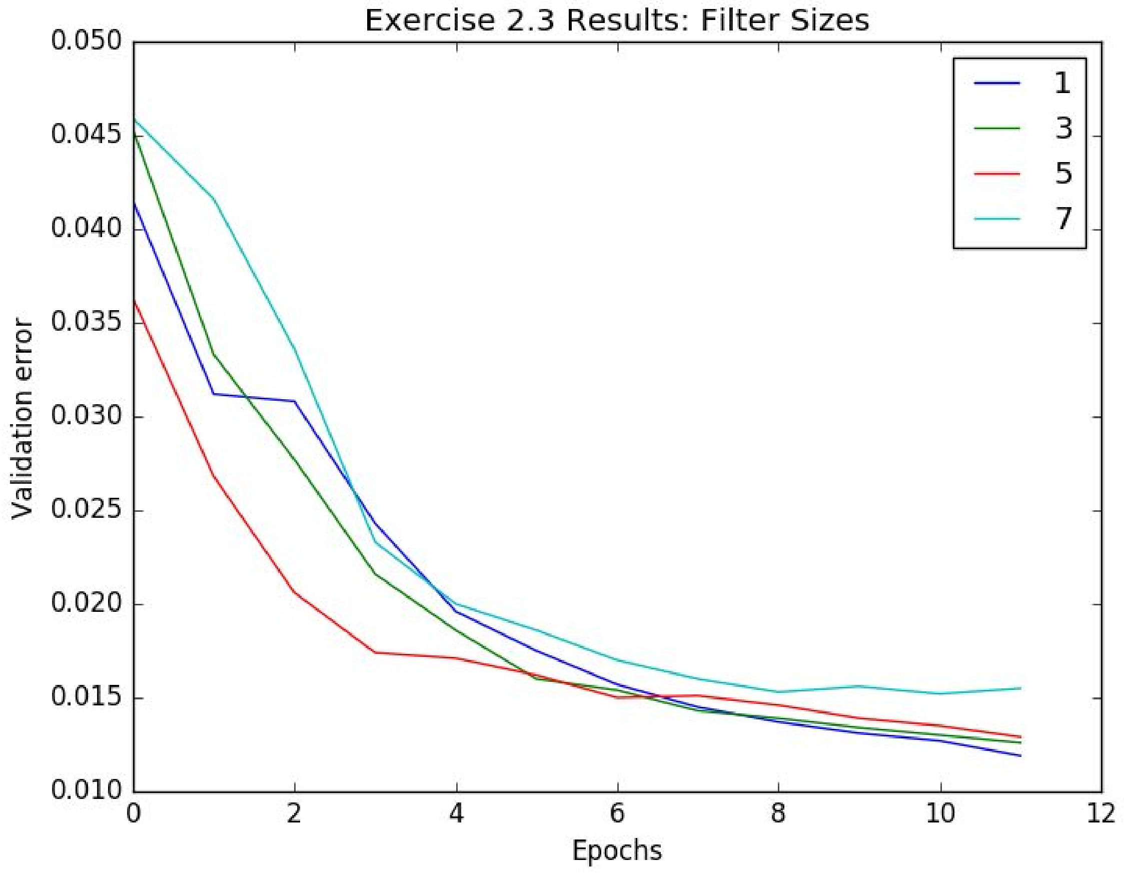


Figure Number: 3

From this graph the conclusion that can be drawn is that smaller filter sizes are better when we have many runs and can spare computational power for that but in cases of very large datasets where number of runs or epochs is a limiting factor higher number of filters seems helpful as in 3 epochs the learning curve for filter_sizes = 5 seems to have minimized the validational loss to a great extent as compared to others. However, going for very high filter sizes can also be harmful as it can be seen from the graph.

Exercise 2.4:

Before displaying the learning curves, I would like to highlight the fact I found interesting, which is, Random Search is really very random and it does not always return the most optimum results or hyperparameters. I would be very interested in learning approaches for the same.

The graph of the data points considered by the random search and their loss versus time is as follows:

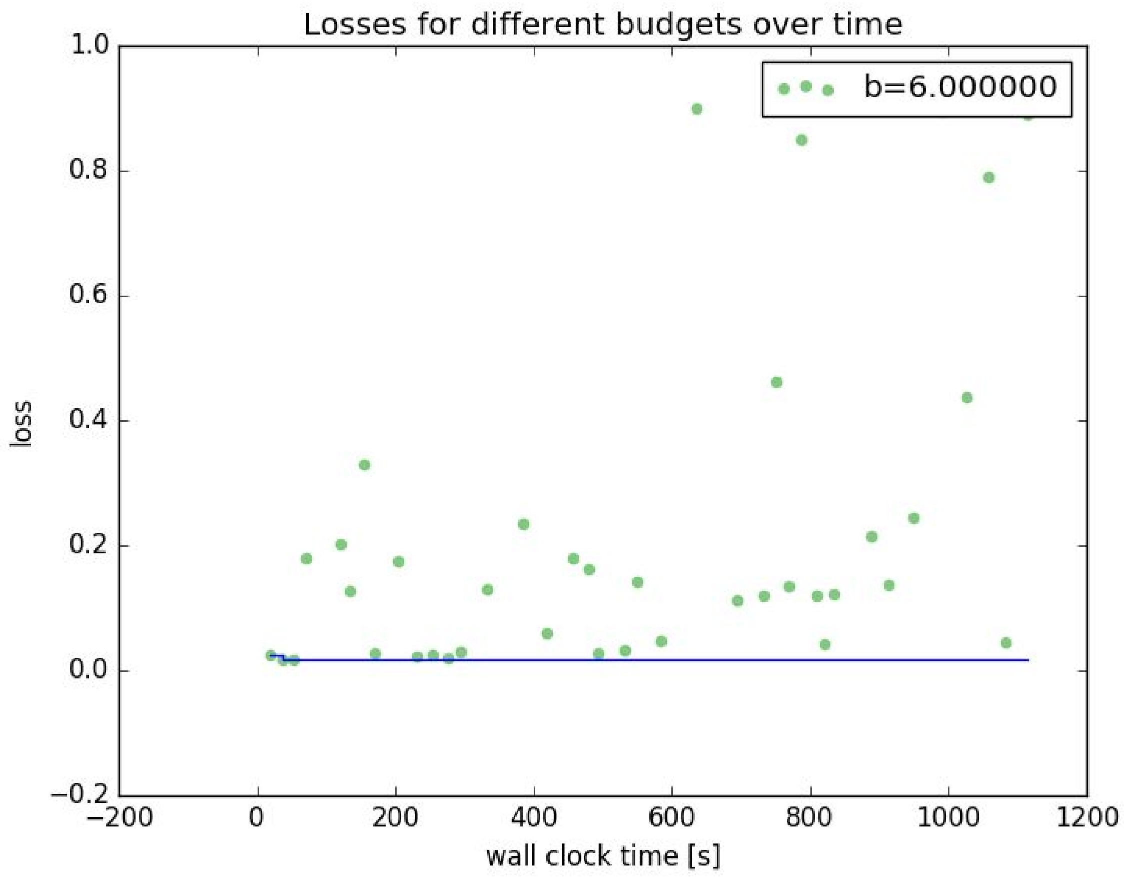


Figure Number: 4

Random search while running was able to find solutions is almost 0 and sometimes actually 0 loss that is 100% accuracy and I believe that is not desirable, it is maybe just the model overfitting the training dataset or finding similar data in the validation as in training.

I ran random search many times to get the most optimal results on running cn_mnist.py again after optimizing the parameters but the results or random search where different every time and none of them can be called optimal.

One of these runs returned the following graph for the learning curve:

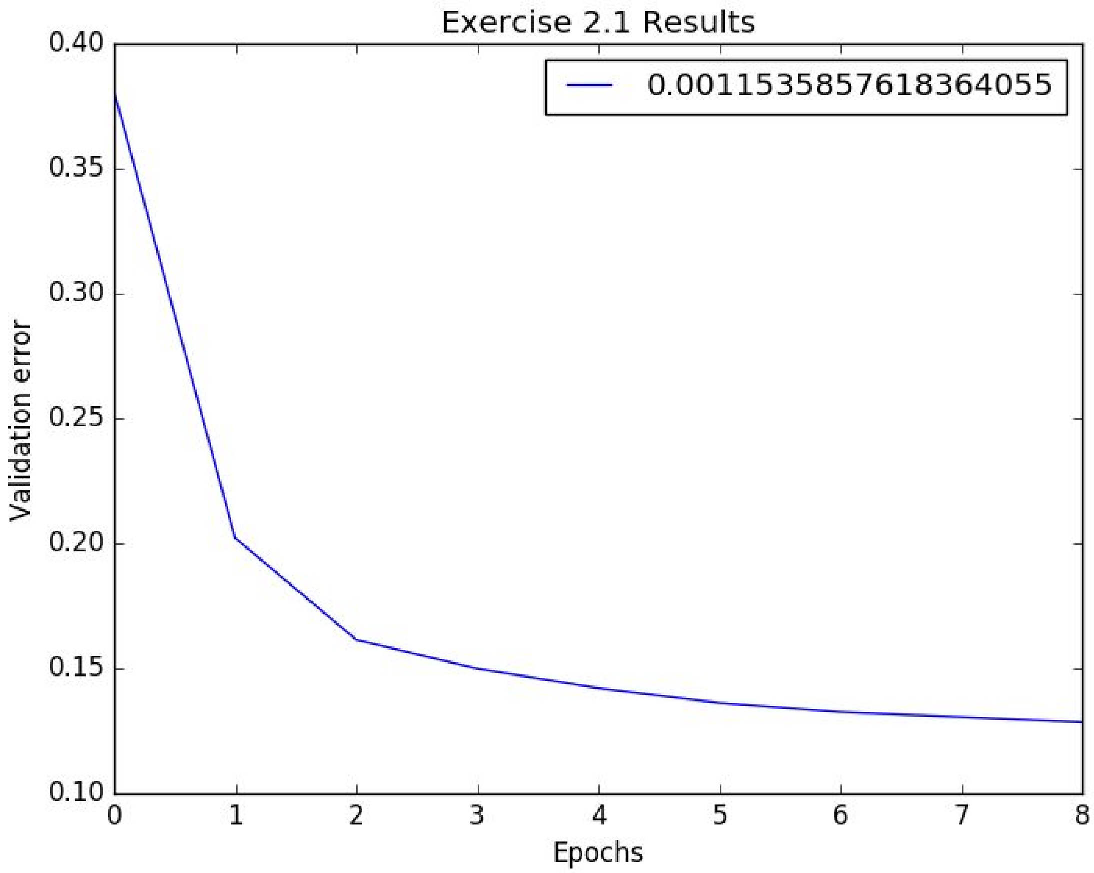


Figure Number: 5

This seemed like one of the most satisfiable solutions provided by the random forest but could not be reproduced. The parameters for this one were:
Best found configuration: {'batch_size': 49.08297250066807, 'learning_rate': 0.00011535857618364056, 'filter_size': 5, 'num_filters': 15}

However, the further results provided by the random search had somewhat similar learning curves and the final one after which I stopped running the model was :

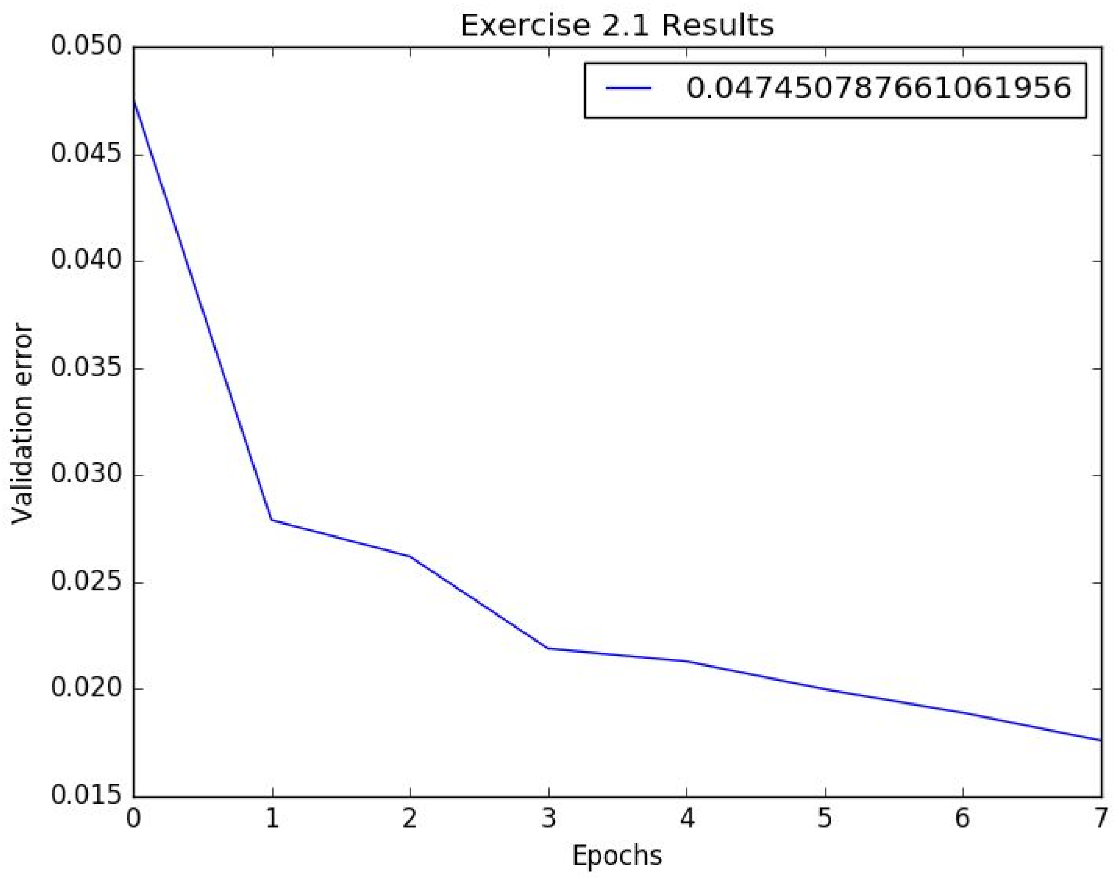
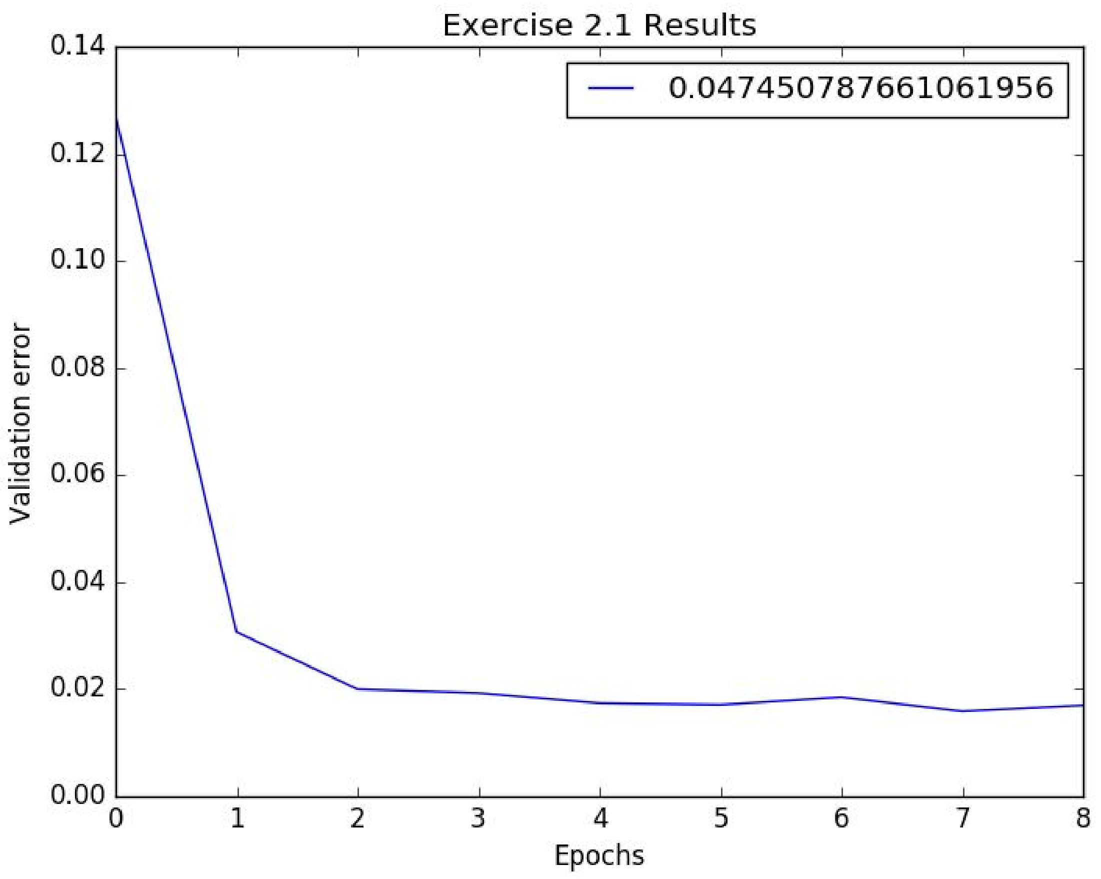


Figure Number: 6

The parameters for this one were:
Best found configuration: {'batch_size': 76.4252544475399, 'filter_size': 5, 'learning_rate': 0.047450787661061956, 'num_filters': 8}

Moreover, an even more interesting thing to look out for was the change in the learning curve with changes in the epochs. And choosing the best epoch was very crucial. As the same configurations with a different epoch gave different learning curves, for example the following learning curve with 8 epochs:



Just adding one step or epoch made a huge difference in the learning curve.

So the best configuration that I found was:
Best found configuration: {'batch_size': 76.4252544475399, 'filter_size': 5, 'learning_rate': 0.047450787661061956, 'num_filters': 8}

And the graph for it is **Figure Number : 5**.

More graphs of the extra run can be found in the Github repository if need for reference or validation.

References:

[1]https://chromium.googlesource.com/external/github.com/tensorflow/tensorflow/+/r0.7/tensorflow/g3doc/tutorials/mnist/pros/index.md

[2] https://github.com/tensorflow/tensorflow