**PROJECT PART 2 REPORT**

**Task 1 results**

* **Properties of the implemented MLP (2-nH-1 MLP):**

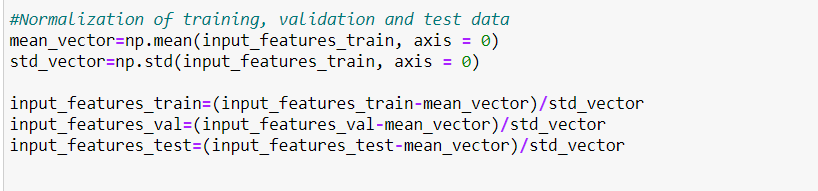
Activation function: Sigmoid

Loss Used : MSE

Learning Method : Batch

Learning Rate : 0.0001

* **Feature Normalization** is performed on training, validation and test data



* Output of the main.py file is as below:

The network is run for the following number of hidden nodes:2,4,6,8,10. For each value of the number of hidden nodes, the network is run multiple times (10 iterations as displayed below) and the accuracy is calculated. The final classification accuracy is thus the average accuracy of all the 10 runs/iterations of the network for the given value of number of hidden nodes.

Accuracy at number of hidden nodes = 2 for 10 iterations are as below:

Accuracy at Iteration 0 = 0.615

Accuracy at Iteration 1 = 0.615

Accuracy at Iteration 2 = 0.615

Accuracy at Iteration 3 = 0.615

Accuracy at Iteration 4 = 0.615

Accuracy at Iteration 5 = 0.615

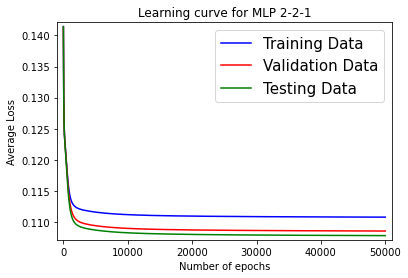
Accuracy at Iteration 6 = 0.615

Accuracy at Iteration 7 = 0.615

Accuracy at Iteration 8 = 0.615

Accuracy at Iteration 9 = 0.615

Average Accuracy at number of hidden nodes = 2 is 0.6150000000000001



Accuracy at number of hidden nodes = 4 for 10 iterations are as below:

Accuracy at Iteration 0 = 0.698

Accuracy at Iteration 1 = 0.698

Accuracy at Iteration 2 = 0.6985

Accuracy at Iteration 3 = 0.698

Accuracy at Iteration 4 = 0.615

Accuracy at Iteration 5 = 0.615

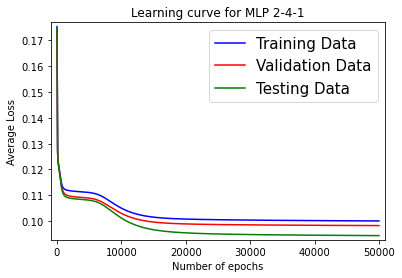
Accuracy at Iteration 6 = 0.697

Accuracy at Iteration 7 = 0.615

Accuracy at Iteration 8 = 0.7785

Accuracy at Iteration 9 = 0.698

Average Accuracy at number of hidden nodes = 4 is 0.6811



Accuracy at number of hidden nodes = 6 for 10 iterations are as below:

Accuracy at Iteration 0 = 0.7775

Accuracy at Iteration 1 = 0.7

Accuracy at Iteration 2 = 0.7785

Accuracy at Iteration 3 = 0.7005

Accuracy at Iteration 4 = 0.701

Accuracy at Iteration 5 = 0.7795

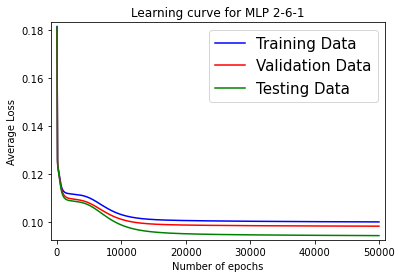
Accuracy at Iteration 6 = 0.615

Accuracy at Iteration 7 = 0.7785

Accuracy at Iteration 8 = 0.7805

Accuracy at Iteration 9 = 0.6995

Average Accuracy at number of hidden nodes = 6 is 0.7310500000000001



Accuracy at number of hidden nodes = 8 for 10 iterations are as below:

Accuracy at Iteration 0 = 0.78

Accuracy at Iteration 1 = 0.7805

Accuracy at Iteration 2 = 0.78

Accuracy at Iteration 3 = 0.701

Accuracy at Iteration 4 = 0.701

Accuracy at Iteration 5 = 0.6995

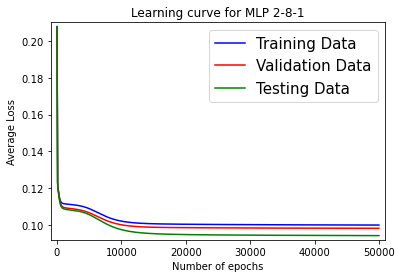
Accuracy at Iteration 6 = 0.6995

Accuracy at Iteration 7 = 0.701

Accuracy at Iteration 8 = 0.7015

Accuracy at Iteration 9 = 0.6995

Average Accuracy at number of hidden nodes = 8 is 0.72435



Accuracy at number of hidden nodes = 10 for 10 iterations are as below:

Accuracy at Iteration 0 = 0.7805

Accuracy at Iteration 1 = 0.7015

Accuracy at Iteration 2 = 0.7795

Accuracy at Iteration 3 = 0.702

Accuracy at Iteration 4 = 0.699

Accuracy at Iteration 5 = 0.768

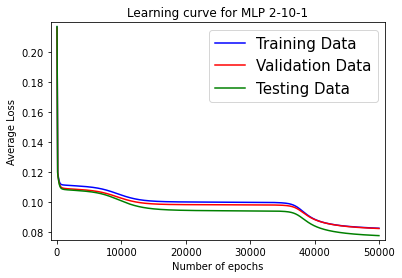
Accuracy at Iteration 6 = 0.6995

Accuracy at Iteration 7 = 0.78

Accuracy at Iteration 8 = 0.7805

Accuracy at Iteration 9 = 0.786

Average Accuracy at number of hidden nodes = 10 is 0.7476499999999999



From the above output of the MLP code (main.py file), we observe that:

Average Accuracy at number of hidden nodes = 2 is 0.6150000000000001

Average Accuracy at number of hidden nodes = 4 is 0.6811

Average Accuracy at number of hidden nodes = 6 is 0.7310500000000001

Average Accuracy at number of hidden nodes = 8 is 0.72435

Average Accuracy at number of hidden nodes = 10 is **0.7476499999999999 (Best value)**

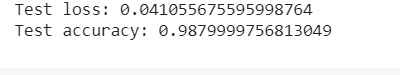
**As we see from the output, the best classification accuracy for the testing set is achieved when number of hidden nodes are 10, which is 0.7476499999999999.**

**Task 2 results**

We tried changing various hyper-parameters to verify our hypothesis about them.

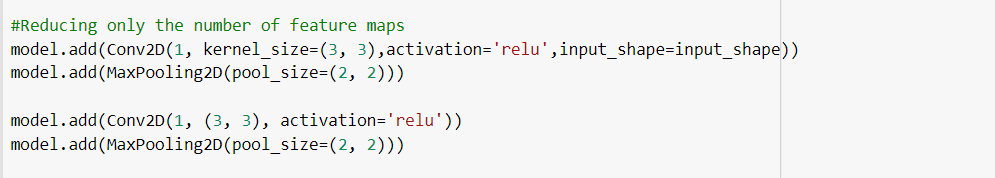
The below code snippets can be run from the file **Project2\_Task2 .ipynb** to see the output.

* **BASELINE EXPERIMENT AS SUCH WITHOUT ANY CHANGE**



* **Hypothesis 1**: **Decreasing number of Feature maps**

Decreasing feature map to 1. The hypothesis is to check that it takes more than 1 convolution feature map to learn generalization on the given data.

Code change:

Results:



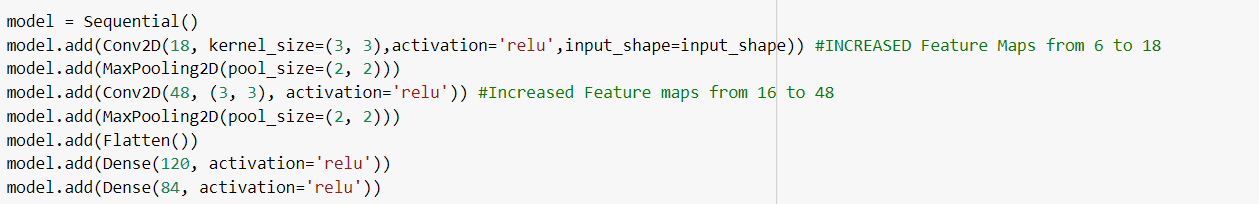
Remarks:

Accuracy decreases, which means that our hypothesis was correct that only one convolution features is not enough to represent the data.

* **Hypothesis 2** **: Increasing number of Feature Maps**

Increasing feature maps by 3 times. The hypothesis is to check that given that model can learn more number of features in convolution, can it learn features that are more relevant.

Code change:



Results:



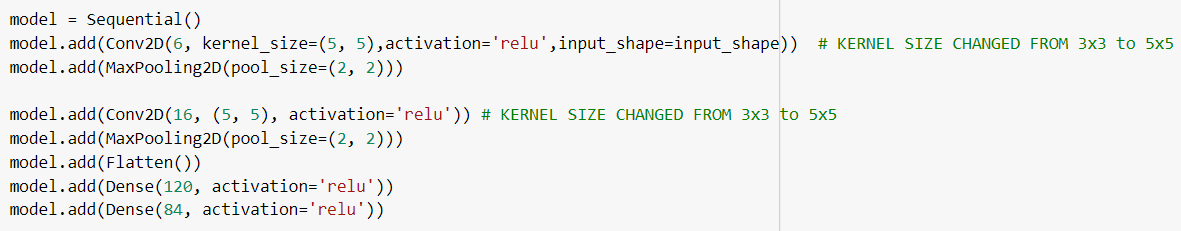
Remarks:

Hypothesis comes out to be true, that is, increasing number of feature maps may result in learning more relevant features, as accuracy increased from baseline accuracy.

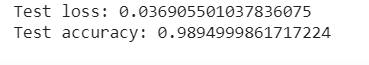
* **Hypothesis 3: Kernel Size**

We will only increase kernel size in this hypothesis, the idea behind this hypothesis is that our images in MNIST data are less complex, so it is possible that we can capture low level features better in a kernel of bigger size

Code change:



Results:



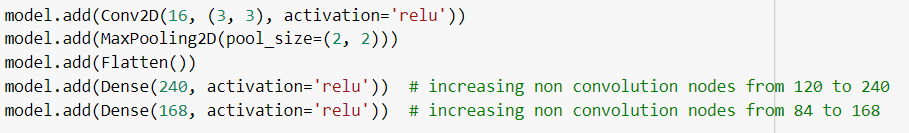
Remarks:

We observe slight improvement over our baseline accuracy. This shows the 5x5 is a better choice in our dataset rather than widely used 3x3 , as our low level features are more extractable when we compare a 25 pixel at a time rather than 9.

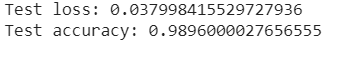
* **Hypothesis 4: Increasing nodes in non convolutional layers**

In this hypothesis we want to increase nodes in last two layers which are non-convolutional layers. The reason is that more nodes can learn more complex representations, so there might be a better representation possible with more nodes, which can help in learning a better model.

Code change:



Results:



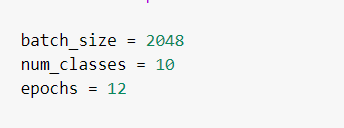
Remarks:

We observe slight improvement with respect to our baseline accuracy, so increasing complexity might be helping in learning a better representation.

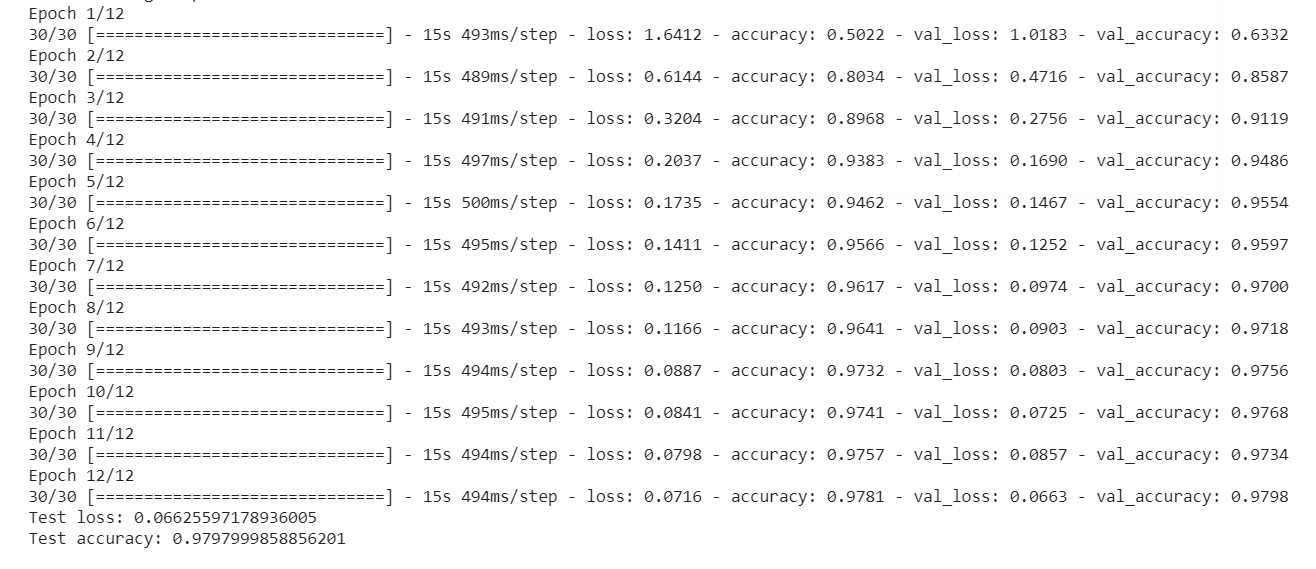
* **Hypothesis 5: Batch Size**

Larger batch size allows computational speed using parallelization of GPUs. In general, too large of a batch size leads to poor generalization. For convex functions, using a batch equal to the entire dataset guarantees convergence to the global optima of the objective function but this costs a slower convergence to that optima. Using smaller batch have faster convergence to good solutions but not necessarily will reach to global optima. In this hypothesis we will test, if we increase batch size, keeping the epochs same, does the accuracy decrease, as it needs more epoch to generalize.

Code change:



Results:



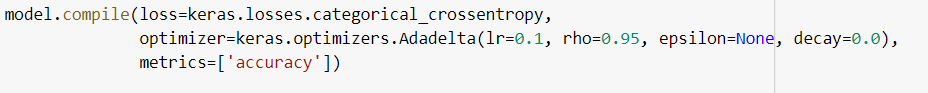
Remarks:

Since Validation acc > training acc, we can run it for more epochs, this shows that larger batch size takes mores epochs to generalize.

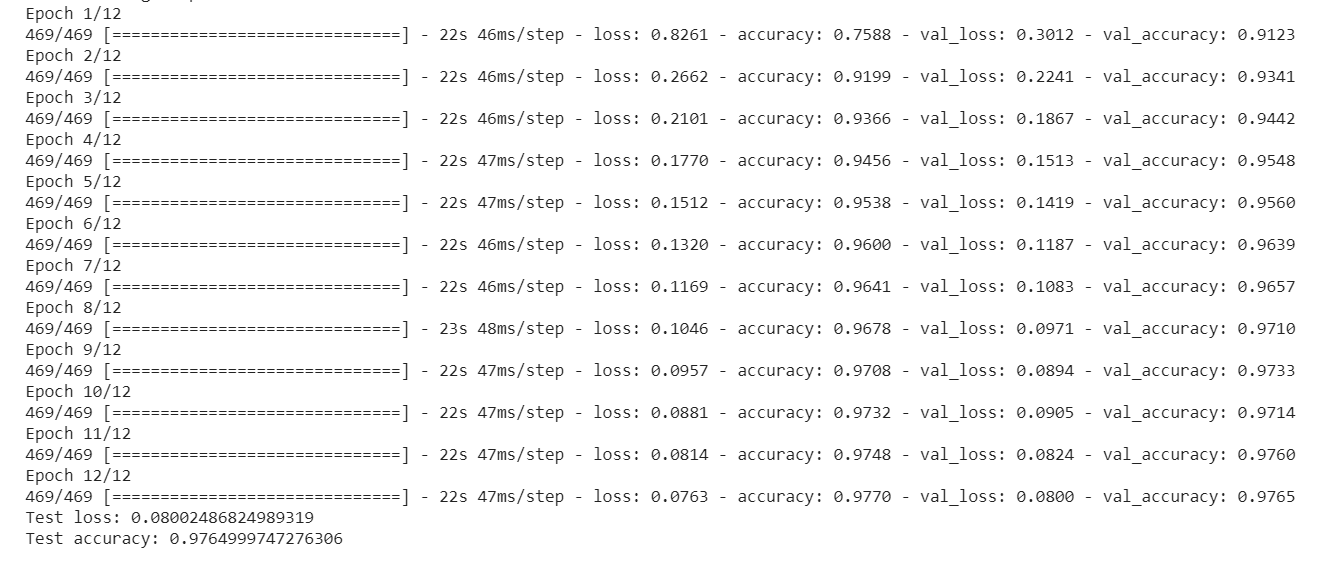
* **Hypothesis 6: Decreasing Learning rate**

Ideal learning rate is hard to predict, a slower learninig rate can increase compuational complexity whereas a large one can take us away from global optima. In this hypothesis, we will test with slower learning rate and show that in same number of epochs it will not converge, in other words it needs more epochs to converge.

Code change:



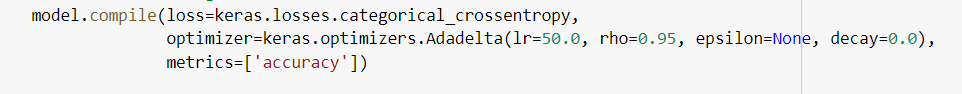
Results:



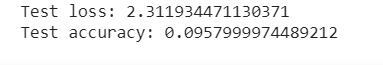
* **Hypothesis 7: Increasing Learning rate**

Ideal learning rate is hard to predict, a slower learning rate can increase computational complexity whereas a large one can take us away from the global optima. In this hypothesis, we will test with higher learning rate and show that it is taking us away from the solution.

Code change:



Results:



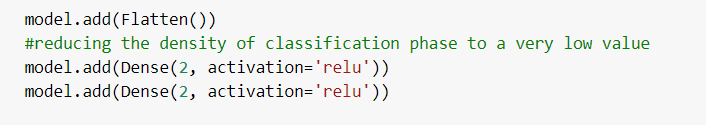
Remarks:

So far this is the least accuracy we have seen, higher learning rate is shifting us away from target vector.

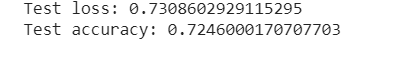
* **Hypothesis 8: Decreasing nodes in non convolutional layers**

In this hypothesis we want to decrease nodes in last two layers which are non-convolutional layers. The reason is that we want to check if convolutional layers itself can act as final features, without representing them again in higher dimension before the output layer.

Code change:



Results:



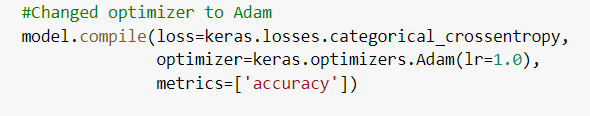
Remarks:

This shows that representiting the output from convolutional layers in higher dimiension is important to learn and use features from it, to train the model.

* **Hypothesis 9: Changing optimizer to Adam**

There are different types of gradient descent optimizers. In baseline code, we have used adadelta, which adapts learning rates based on a moving window of gradient updates. In this hypothesis we want to test Adam, which is based on the adaptive estimation of first-order and second-order moments.

Code change:



Results:



Remarks:

Hence, this optimizer is not working in this scenario.