Master M1 MoSIG – UGA & Grenoble INP

Intelligent Systems

Recognizing Digits using Neural Networks

Authors :
Archit YADAV
Irman FAQRIZAL

Teacher :
Prof. James L CROWLEY

Last Version
April 30, 2020

Contents

1 Preliminaries

In this project we will study the performance of several neural networks on MNIST dataset. This section explains how we carry out the project including what are the tools we are going to use and how we will organize the dataset.

1.1 Methodology

The code in this project is written in python with a library called keras. Using these tools, we will implement from the most basic form, up to the more complicated convolutional neural network (CNN). In each network, some experiments will be conducted by changing parameters such as learning rate, number of epochs, and kernel size (for CNN). Also in every network a performance evaluation will be presented. Note that to compute some of the performance metrics we will use a tool called scikit-learn, and to plot the ROC we used a library called skicitplot. In the end, we will make a conclusion by comparing the pros and cons of each network.

1.2 Dataset

By default in keras, MNIST dataset can be loaded into training set and testing set, each contains respectively 60000 and 10000 of images. However, to validate the training of the network, we will split again the training set by 90%. This means our distribution of the data is the following :

- 54000 of training data.
- 6000 of validation data.
- 10000 for testing.

2 Basic Network

Basic in here is implies to the most 'simple' computation and architecture of the network. The purpose of doing this is simply to later compare the results of this network, with the more complicated ones. Then see if its worth adding complexity, while probably the 'primitive' one is already performs well.

2.1 Architecture

On figure 1, we present the architecture of this network. We are using only 1 hidden layer, with 84 neurons and a linear activation function. Note that for this network and the next network we will use a loss function called 'sparse categorical cross entropy', and we won't be focusing (discuss) on this.

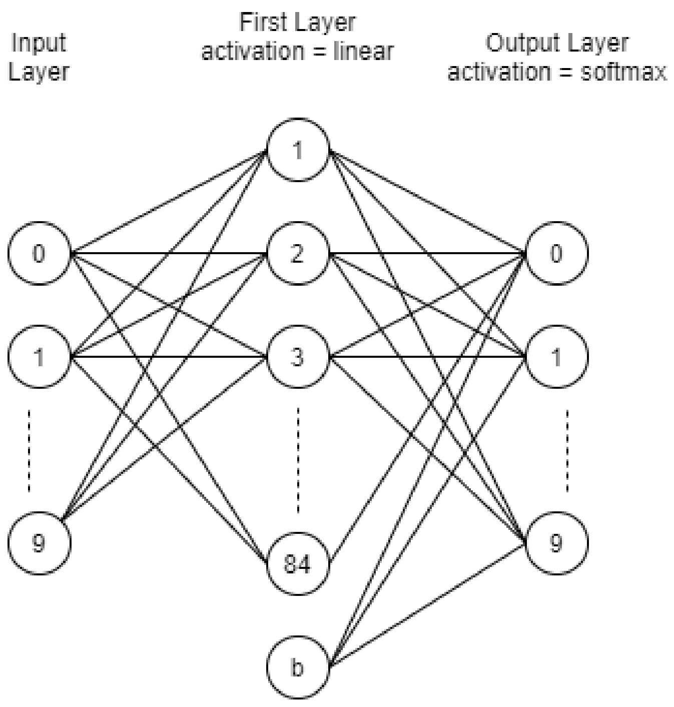
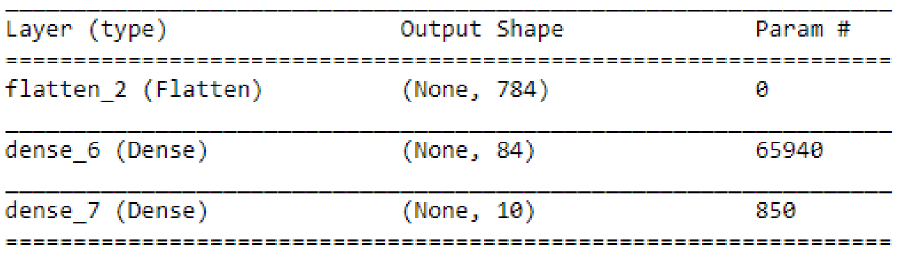


Figure 1: Basic Network Architecture

We can also see a summary of the network from keras itself in figure 2. Noticed that there is flatten layer, this is because we need to 'flatten' the dimension of the data, since our dense layer doesn't accept the default form of MNIST dataset. We can also see there is another layer with 10 neurons, which is the output layer (each neuron represents a single class).



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_6 (Dense)              (None, 84)                65940
_________________________________________________________________
dense_7 (Dense)              (None, 10)                850
=================================================================
```

Figure 2: Basic Neural Network

2.2 Experiments

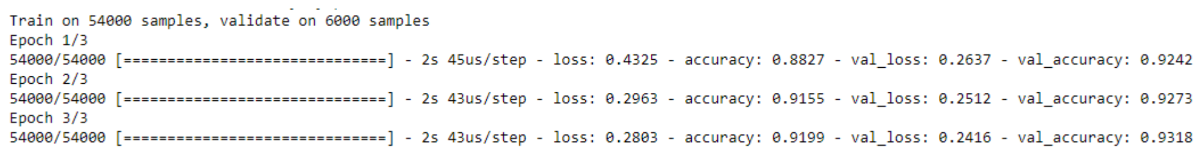
The experiments for this network, we are going to use the following parameters:

Experiment	Learning Rate	Epochs	Batch Size
1	0.001	3	60
2	0.1	3	60
3	0.001	10	60
4	0.001	3	600

As we can see from table above in each experiment we will simply change the value of each parameter. The purpose is to observe how these parameters will affect the performance of the network.

2.3 Performance

Below on figure 3 we present the result of training using the first set of parameters (Experiment 1). We can see that the result is relatively not bad, the validation accuracy successfully reach above 90%. Also, in each epoch the loss is decreasing while the accuracy is increasing, which means that neural network is 'learning'. Noticed that the whole process of training only takes a few seconds (6 seconds in total).



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/3
54000/54000 [==============================] - 2s 45us/step - loss: 0.4325 - accuracy: 0.8827 - val_loss: 0.2637 - val_accuracy: 0.9242
Epoch 2/3
54000/54000 [==============================] - 2s 43us/step - loss: 0.2963 - accuracy: 0.9155 - val_loss: 0.2512 - val_accuracy: 0.9273
Epoch 3/3
54000/54000 [==============================] - 2s 43us/step - loss: 0.2803 - accuracy: 0.9199 - val_loss: 0.2416 - val_accuracy: 0.9318
```

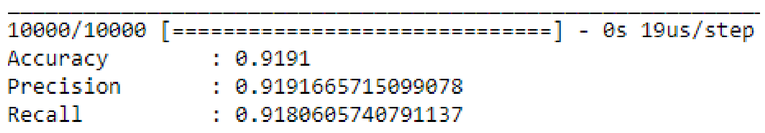
Figure 3: Basic Neural Network Training Result (1st Experiment)

We also evaluate the performance of this network by providing the accuracy, precision, and recall (shown on figure 5). Note that in these results we are using testing dataset.

$$Pr_{macro} = \frac{Pr_1 + Pr_2 + \ldots + Pr_k}{k} = Pr_1\frac{1}{k} + Pr_2\frac{1}{k} + \ldots + Pr_k\frac{1}{k}$$

Figure 4: Precision-Recall 'macro' average

To compute precision and recall (for now) we are using 'macro' average mode (shown on 4), which will reduces the multi-class predictions down to multiple sets of binary predictions. It is true that using this mode, the computation will not consider the class imbalance. However as written on the skicit documentation, using 'micro' average for multi-class and single label for each class will result in the same exact values as accuracy (for both precision and recall). Note that we will present the plot for precision and recall (for each class) in the next subsection.



```
10000/10000 [==============================] - 0s 19us/step
Accuracy        : 0.9191
Precision       : 0.9191665715099078
Recall          : 0.9180605740791137
```

Figure 5: Basic Neural Network Performance Metrics (1st Experiment)

On table below, we present the performance metrics for each experiment mentioned in section 2.2.

Experiment	Accuracy	Precision	Recall
1	0.919	0.919	0.918
2	0.437	0.295	0.428
3	0.922	0.921	0.921
4	0.916	0.915	0.915

From table above we may conclude several points:

- Changing the parameters (learning rate, epochs, batch size), does influence the performance of the network.
- On the second experiment, the performance drastically declined. This is because, increasing the learning rate caused unstable learning (too quickly to adapt on each iteration). Note that there is a warning of zero-division in the computation of precision and recall for this because the predicted classes didn't cover all the available classes.
- On the third and the last experiments, we can see that in this network number of epochs and batch size don't really have much impact on the performance. However, we can see that the third experiment does increase the performance slightly since we simply increase the number of epochs.

Note that we provide all the results (from keras's output) of these experiments on Appendix.

2.4 *Additional Metrics*

To explore further our network, we will present in this section more detailed result of the performance analysis. On figure 6 we can see the plot of precision and recall for each class, and the ROC curve for experiment number 3 (best performing experiment).

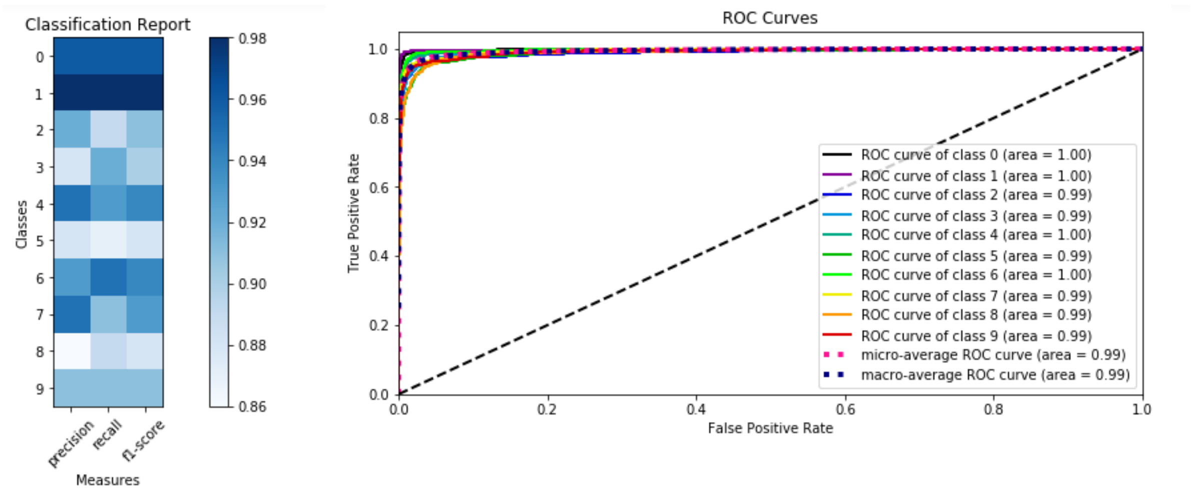


Figure 6: Precision-Recall and ROC Plots (experiment 3)

As we can see from figure above the values of precision and recall vary among different classes. The model successfully classified class '1' almost perfectly (dark blue), however its not performing very well to identify class '5' (lighter blue).

While from the ROC plot, we can see that the model performs really well by looking at the Area Under the Curve (AUC). We can also see the relation between two plots, the classes which reach AUC equals to 1 ('0', '1', '4', '6') are the ones with higher values for precision and recall.

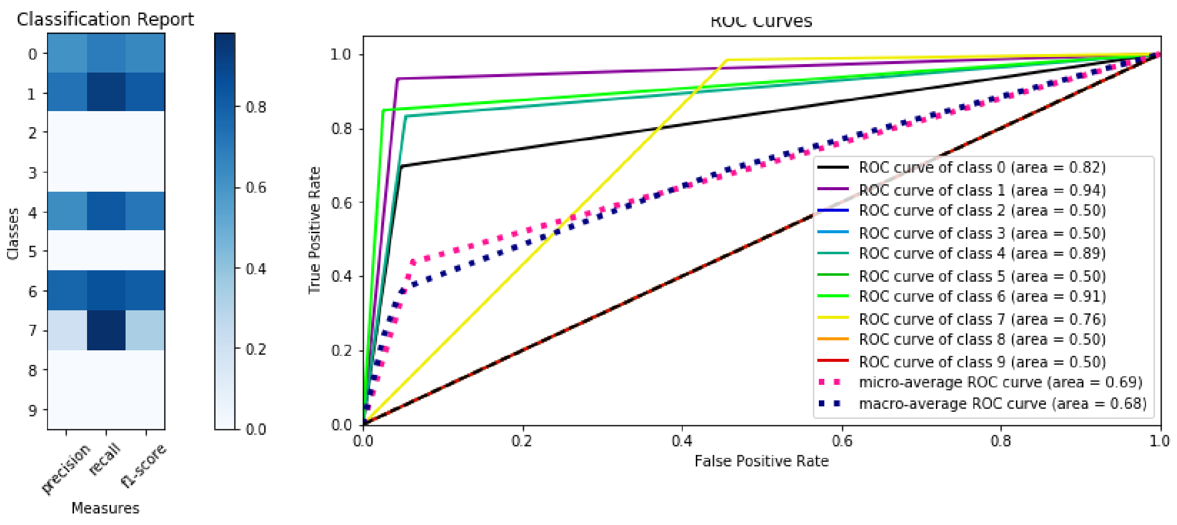


Figure 7: Precision-Recall and ROC Plots (experiment 2)

On figure 7 above we can see, the plots for the second experiment. The network for this experiment actually failed to identify some classes ('2', '3', '5', '8', '9'). As mentioned before, the network adapt too quickly for each iteration. This also means that if we modified slightly the number of epochs and batch size, we could have totally different results (yet still probably poor results).

3 Convolutional Neural Network

In this section we will improve the complexity of our network by adding a convolutional layer.

3.1 *Architecture*

On figure 8, we show the architecture of the network. Note that this architecture is relevant with the first experiment we are going to do, since we will try to tweak some of the parameters such as kernel size and stride.

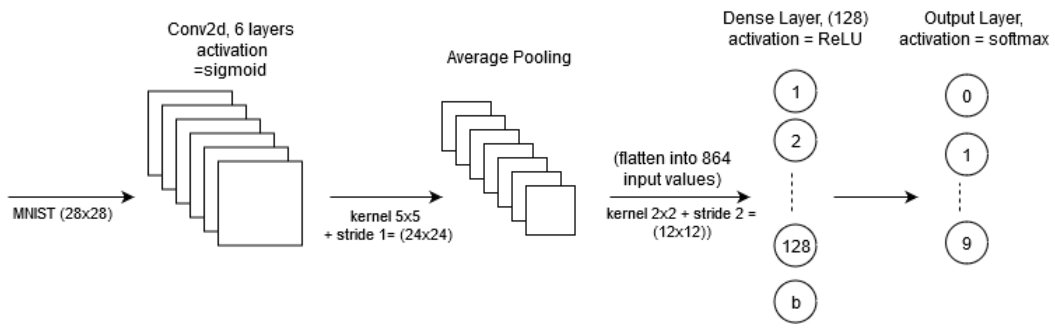
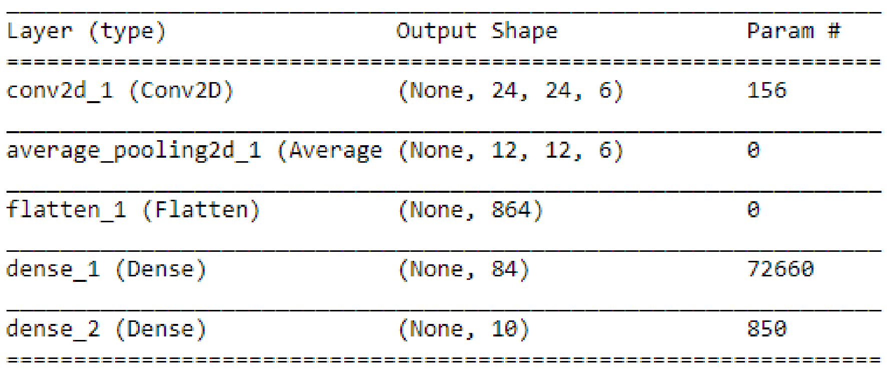


Figure 8: Convolutional Neural Network Architecture (1st experiment)

As we can see now we have 2 more layers, these are a covolutional layer and a pooling layer. For convolutional layer we are using 5 by 5 kernel size, stride 1, and depth 6 (feature maps), with sigmoid as an activation function. While for pooling layer we are using 2 by 2 kernel size, stride 2, and the type of the pooling is average pooling. On the third layer we are again using dense layer, however this time we will try to increase the number of neuron to 128 and using ReLU activation function, which is non linear.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 12, 12, 6)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 864)               0
_________________________________________________________________
dense_1 (Dense)              (None, 84)                72660
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
```

Figure 9: Convolutional Neural Network

We present the model summary from keras on figure 9. As we can see we still have the flatten layer and the output layer. We can also see that for the first layer (convolutional layer) the output of the layer is size 24 by 24. This is because we are using kernel of size of 5 by 5 and stride 1, and we lost 4 pixels of each axis (MNIST image is 28x28). The output of pooling layer is 12 by 12, because we are using 2 by 2 kernel size, and stride 2, which will simply halves the input dimension.

3.2 Experiments

Since in the previous network we already observed the impact of parameters such as learning rate and number of epochs. In this set of experiments of the network we will fixed these parameters, and observe another kind of parameters such as kernel size, stride, and activation function.

Thus we present below the fixed parameters we will use:

- 0.001 learning rate.
- 10 epochs
- 60 batch size.

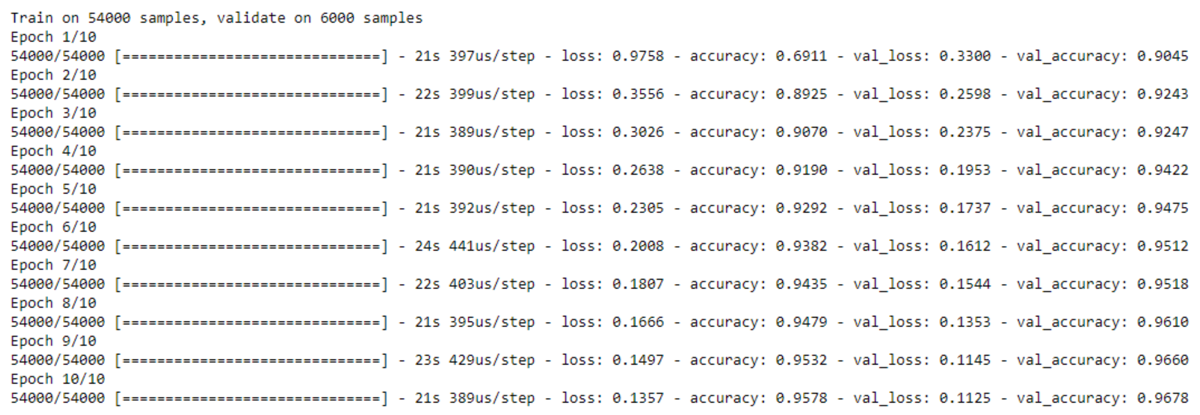
The other parameters we will observe by changing their values are shown on table below:

Experiment	Layer	K. Size	Stride	F. Maps	Act. Func./P. Type
1	Conv.	5x5	1	6	sigmoid
	Pooling	2x2	2	-	average
2	Conv	10x10	2	6	sigmoid
	Pooling	2x2	2	-	average
3	Conv.	5x5	1	6	sigmoid
	Pooling	4x4	4	-	average
4	Conv.	5x5	1	6	tanh
	Pooling	2x2	2	-	average
5	Conv.	5x5	1	6	sigmoid
	Pooling	2x2	2	-	max
6	Conv.	5x5	1	24	sigmoid
	Pooling	2x2	2	-	average

As we can see from table above, for each experiment we simply want to know what will happen if we changed the value of each parameter. For the second and third experiments we changed the value of kernel size and stride, both for convolutional layer and pooling layer respectively. The purpose is to observe what will be the impact of the coverage of the image pixels during training. The fourth and the fifth we will try to examine different kind of activation functions and type of pooling. Lastly on the sixth experiment, we try to see what is the effect of changing the number of feature maps (depth).

3.3 Performance

We present on figure 10 the result of training using for the first experiment.



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 21s 397us/step - loss: 0.9758 - accuracy: 0.6911 - val_loss: 0.3300 - val_accuracy: 0.9045
Epoch 2/10
54000/54000 [==============================] - 22s 399us/step - loss: 0.3556 - accuracy: 0.8925 - val_loss: 0.2598 - val_accuracy: 0.9243
Epoch 3/10
54000/54000 [==============================] - 21s 389us/step - loss: 0.3026 - accuracy: 0.9070 - val_loss: 0.2375 - val_accuracy: 0.9247
Epoch 4/10
54000/54000 [==============================] - 21s 390us/step - loss: 0.2638 - accuracy: 0.9190 - val_loss: 0.1953 - val_accuracy: 0.9422
Epoch 5/10
54000/54000 [==============================] - 21s 392us/step - loss: 0.2305 - accuracy: 0.9292 - val_loss: 0.1737 - val_accuracy: 0.9475
Epoch 6/10
54000/54000 [==============================] - 24s 441us/step - loss: 0.2008 - accuracy: 0.9382 - val_loss: 0.1612 - val_accuracy: 0.9512
Epoch 7/10
54000/54000 [==============================] - 22s 403us/step - loss: 0.1807 - accuracy: 0.9435 - val_loss: 0.1544 - val_accuracy: 0.9518
Epoch 8/10
54000/54000 [==============================] - 21s 395us/step - loss: 0.1666 - accuracy: 0.9479 - val_loss: 0.1353 - val_accuracy: 0.9610
Epoch 9/10
54000/54000 [==============================] - 23s 429us/step - loss: 0.1497 - accuracy: 0.9532 - val_loss: 0.1145 - val_accuracy: 0.9660
Epoch 10/10
54000/54000 [==============================] - 21s 389us/step - loss: 0.1357 - accuracy: 0.9578 - val_loss: 0.1125 - val_accuracy: 0.9678
```

Figure 10: Convolutional Neural Network Training Result (1st Experiment)

The first thing we noticed when we execute the training is this network takes a lot more time (22 seconds each epoch). However, the result is quite good. Each epoch, the network learn progressively well, we can observe this by looking at the increasing accuracy and the decreasing loss. The validation results at every epochs also showing positive results (no indication of overfitting). Note that we also present the results for another experiments on appendix (Section 6.1.2).

The same as the previous network, we also compute the performance metrics. On table below we present the performance results of the experiments:

Experiment	Accuracy	Precision	Recall
1	0.963	0.963	0.963
2	0.973	0.973	0.973
3	0.952	0.953	0.952
4	0.978	0.978	0.978
5	0.967	0.967	0.967
6	0.114	0.011	0.100

As we can see from table above, the performance of the networks are barely noticeable (except number 6). We may conclude several points from these experiments:

- Increasing the kernel size and stride for convolutional layer (first layer) may improve the performance (experiment 2), on the other hand for pooling layer may worsen the result (experiment 3).
- The 4th experiment has slightly better performance than the others. This experiment (4th) is using activation ’tanh’, probably this function is indeed more suitable for this problem (used in LeNet).
- We can see that in this case, changing the type of pooling in the pooling layer (max pooling in experiment 5) doesn’t really affect the result.
- For the last experiment we are using 24 feature maps, and the result is horrible. We will try to investigate this on the next subsection.

3.4 *Additional Metrics*

This time we will present two results (plots) from our experiments. On figure 11, the same as previous network we show our best performing experiment (number 4).

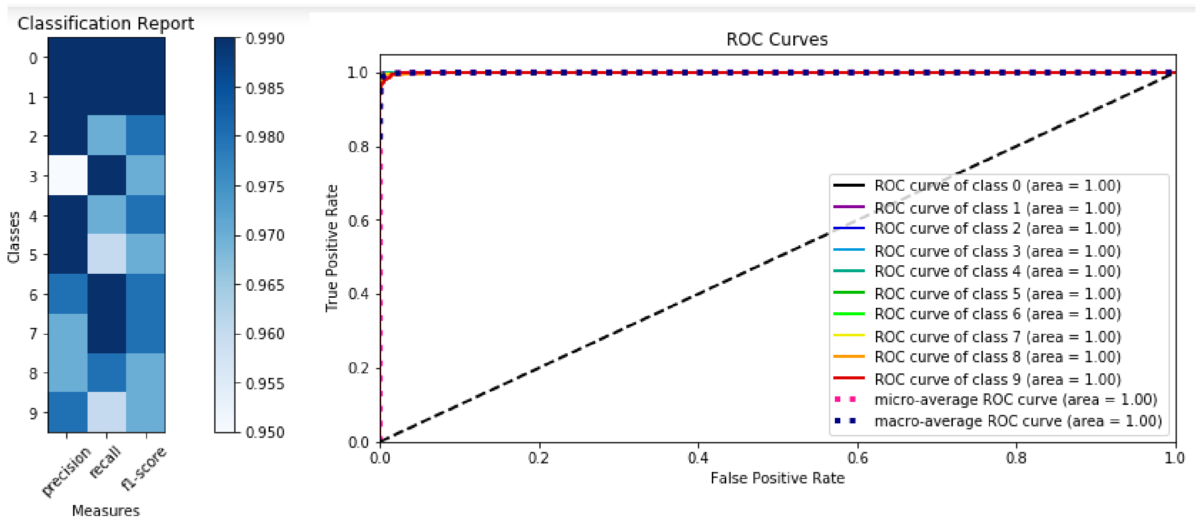


Figure 11: Precision-Recall and ROC Plots (experiment 4)

We can see that from the ROC above, all classes identified to have 'perfect testing' (AUC = 1), however this doesn't mean its overall accuracy is 1. While on the precision and recall plot, it is interesting that unlike the first network, this network managed to classify class '5' better however lower precision value for class '4'.

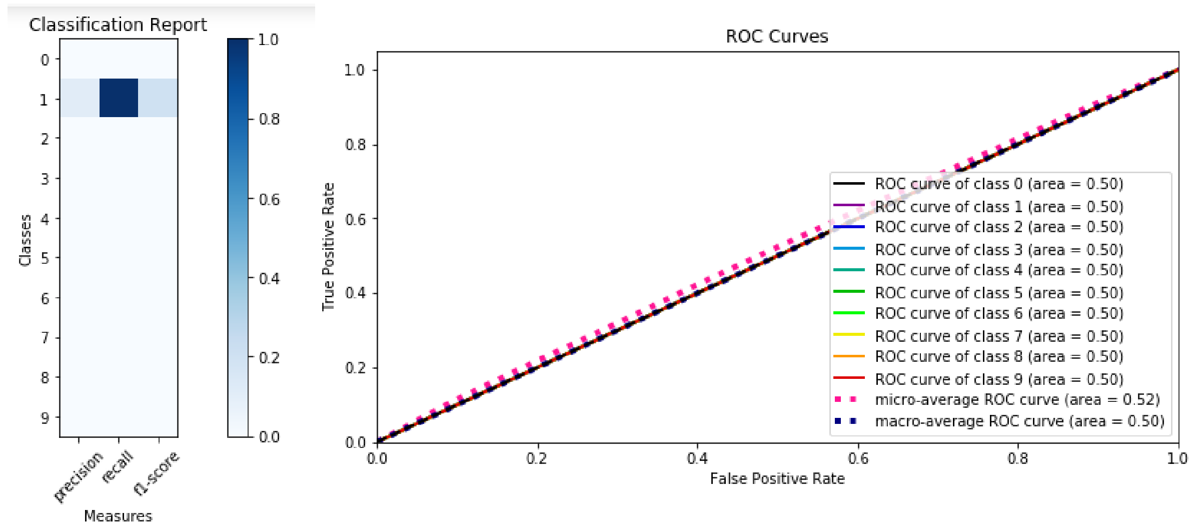


Figure 12: Precision-Recall and ROC Plots (experiment 6)

On figure 12 above we will try to investigate why the accuracy of the model on the last experiment is terrible (we can see table on the performance table section 3.3). We can see from the precision and recall plots, it seems that the model classify all the data into class ′1′. The ROC plot showing that all classes have AUC of 0.5, which means it is not better than random guesses.

What we suggest happened in experiment number 6 is that we may have an imbalance of the kind of features extracted from the image. We know that the purpose of using multiple feature maps is to extract different features in the image. However when we have too many of these, since the values of each kernel are randomized, the extracted features may become too concentrated on only one or few kind of features.

4 LeNet

4.1 *Architecture*

In order to understand the different parameters playing a part in the LeNet architecture, let's have a look at the following flowchart representation:

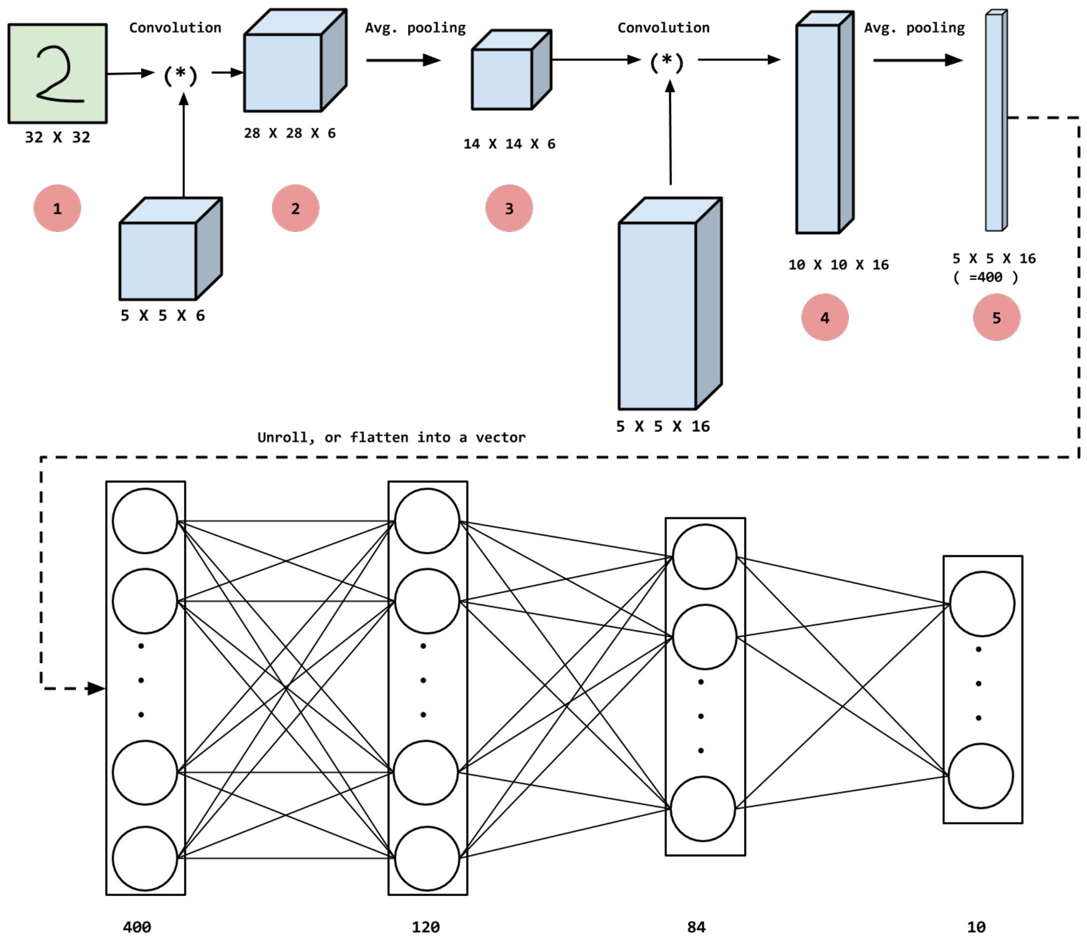


Figure 13: Architecture flowchart of LeNet

The process involved from step 1 to 2, as well as from step 3 to 4 involves convolution. The dimensions of the resultant can be represented by the following formulae, taking into account depth (f), stride (s), padding (p):

$$\lfloor \frac{n+2p-f}{s} \rfloor + 1$$

Similarly, for pooling, there is a dependence on the pooling window dimensions and the stride length s. The buildup of the architecture can be conveniently represented in the following tabular fashion:

Layer	Layer Type	Feature maps	Size	Kernal Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5X5	1	tanh
2	Average pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5X5	1	tanh
4	Average pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	Fully Connected	-	84	-	-	tanh

4.2 Experiments

- For this experiment, we decided to observe metrics after the end of every epoch.
- We decided to tweak around some parameters and observe their effect on the change after each epoch.
- For the experiments on this LeNet network, Following is a table that lists all the experiments done, along with the different parameters:

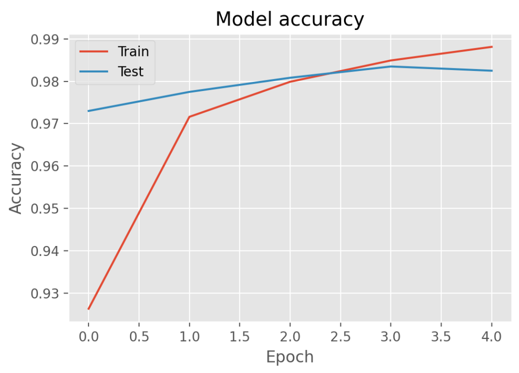
No.	Rate	BatchSize	Epoch	Pooling	KernalSize	StrideLength	Activation
1	0.001	32	5	Average	5x5	1	tanh
2	0.001	**64**	5	Average	5x5	1	tanh
3	**0.01**	32	5, **10**	Average	5x5	1	tanh
4	0.001	32	5	Average	**3x3**	1	tanh
5	0.001	32	5	Average	5x5	1	**sigmoid**

4.3 Performance

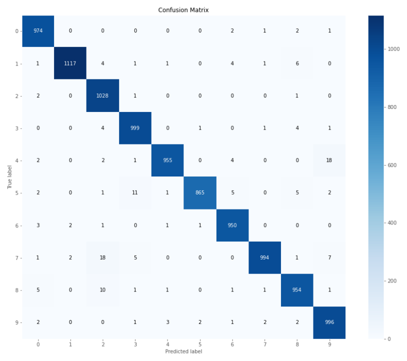
4.3.1 Experiment 1

The first case is the default one, in which the LeNet architecture was built with all the default parameters. We observe the following:

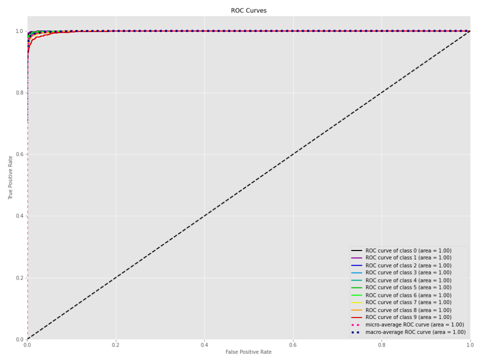
Epoch	Accuracy	Precision	Recall	Val. Acc.
1	0.9263	0.9917	0.9512	0.9730
2	0.9716	1.000	0.9919	0.9775
3	0.9799	1.000	0.9956	0.9808
4	0.9849	1.000	0.9968	0.9835
5	0.9881	1.000	0.9977	0.9825



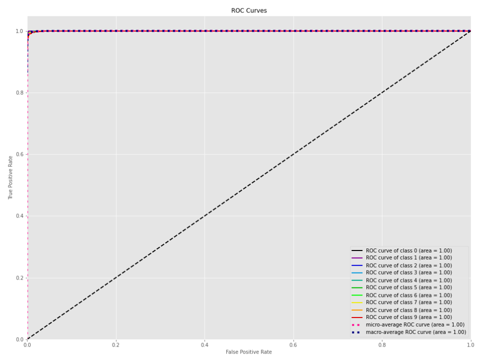
(a) Accuracy with epoch



(b) Confusion matrix



(c) ROC curve after 1st epoch



(d) ROC curve after 5th epoch

Figure 14: Some metrics for experiment 1

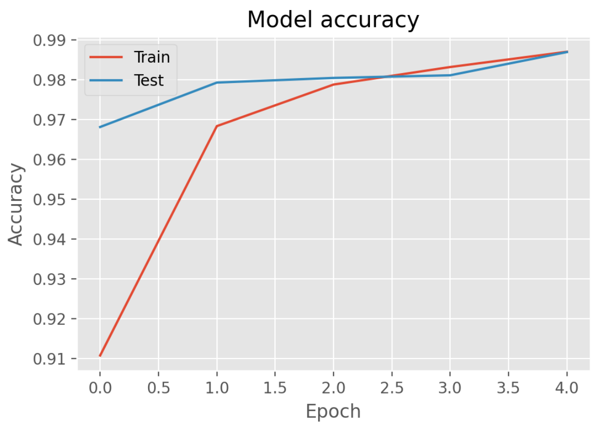
Initial observations:

- The LeNet architecture with all its default parameters gave a quite good performance compared to the previous CNN architecture (Section 3)
- As can be seen from Fig. 13 (c) and (d) the area under ROC is almost 1 after the 1st epoch only.

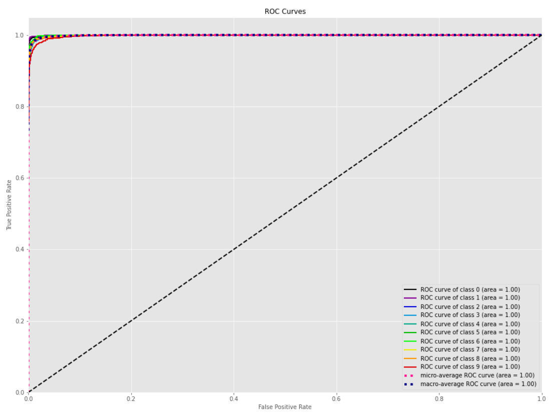
4.3.2 Experiment 2

For 2nd experiment, the batch size was doubled from default 32 to 64. Following are the results:

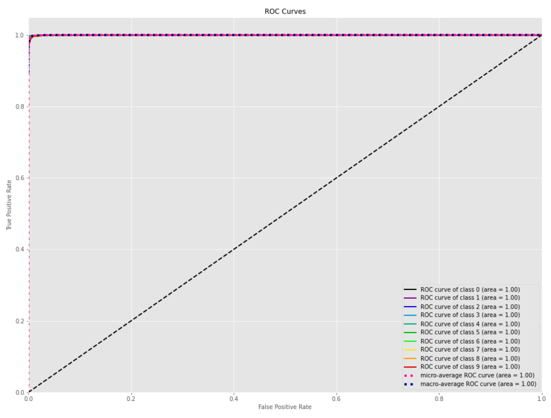
Epoch	Accuracy	Precision	Recall	Val. Acc.
1	0.9107	0.9858	0.9272	0.9682
2	0.9684	1.000	0.9893	0.9793
3	0.9788	1.000	0.9938	0.9805
4	0.9832	1.000	0.9958	0.9812
5	0.9871	1.000	0.9966	0.9870



(a) Accuracy with epoch



(b) ROC curve after 1st epoch



(c) ROC curve after 5th epoch

Figure 15: Metrics for experiment 2

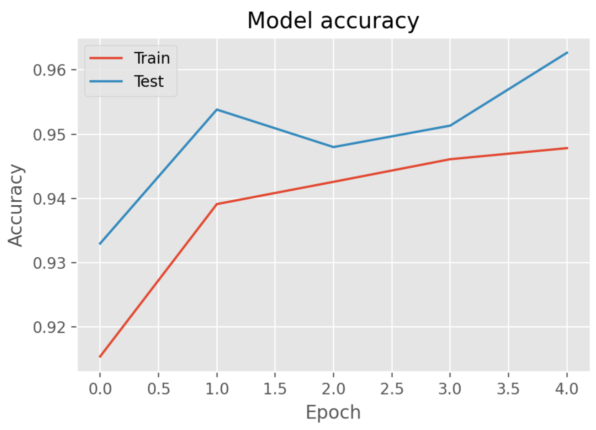
Observations:

- The time taken to train the model was quite less (almost half the time).
- There was almost no difference in any of the metrics.

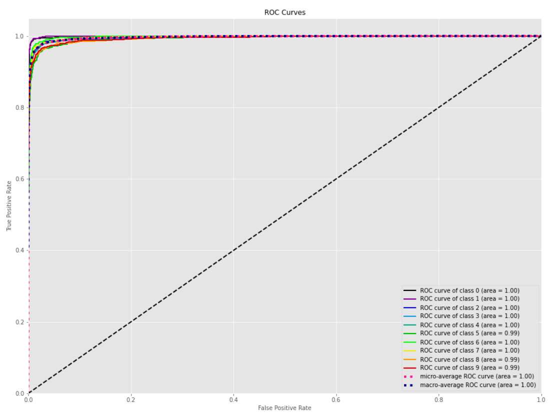
4.3.3 Experiment 3

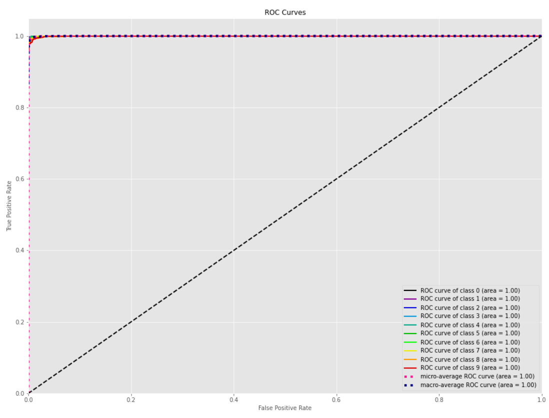
For this experiment, the learning rate was modified from 0.001 to 0.01, which is a 10 times increase.

Epoch	Accuracy	Precision	Recall	Val. Acc.
1	0.9107	0.9994	0.9724	0.9330
2	0.9391	1.000	0.9874	0.9538
3	0.9426	1.000	0.9893	0.9480
4	0.9461	1.000	0.9898	0.9513
5	0.9478	1.000	0.9905	0.9627



(a) Accuracy with epoch





(b) ROC curve after 1st epoch

(c) ROC curve after 5th epoch

Figure 16: Metrics for experiment 3

In this case, there were some interesting observations:

- The training accuracy was a bit less than the validation accuracy. That means performance after training was better and the model was working better at new dataset, after every epoch.
- The ROC curve are still the same. Only difference is that for some classes, the area under curve was 0.99 instead of perfect 1 previously everytime.
- Now if we had continued this to 10 epochs (5 more epochs), even then the performance wouldn't be so great, as can be seen from the graph:

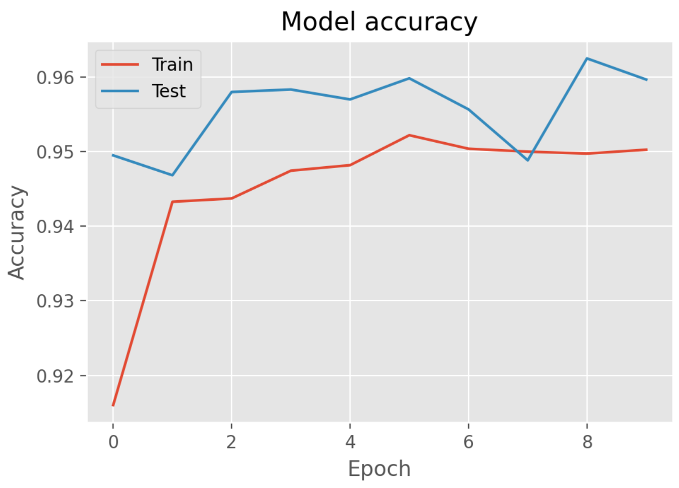


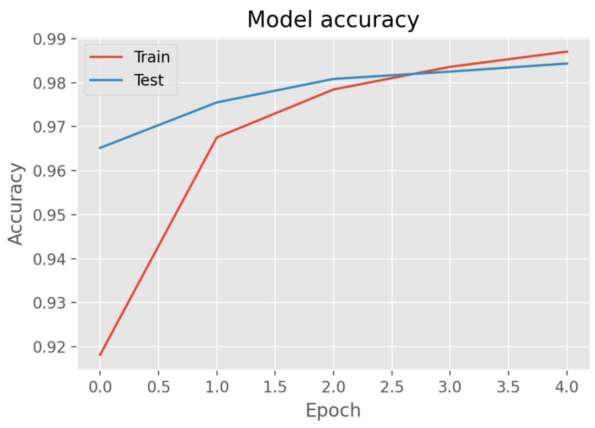
Figure 17: Accuracy with epoch, for 10 epochs

- Clearly, from the graph, increasing the **learning rate didn't had a very good effect**. The testing (Validation here) accuracy actually went down after 5th epoch.

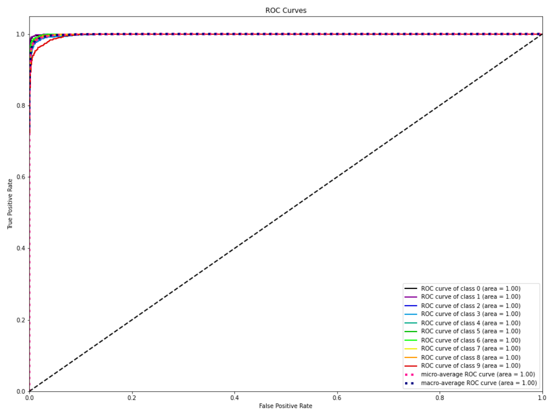
4.3.4 Experiment 4

Now we modified the kernal size from 5x5 to 3x3 for both the 2D convolution layers.

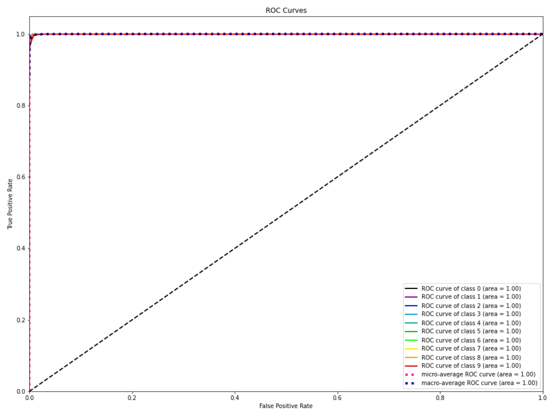
Epoch	Accuracy	Precision	Recall	Val. Acc.
1	0.9182	0.9923	0.9512	0.9449
2	0.9676	1.000	0.9919	0.9902
3	0.9784	1.000	0.9956	0.9944
4	0.9836	1.000	0.9968	0.9964
5	0.9870	1.000	0.9977	0.9976



(a) Accuracy with epoch



(b) ROC curve after 1st epoch



(c) ROC curve after 5th epoch

Figure 18: Metrics for experiment 4

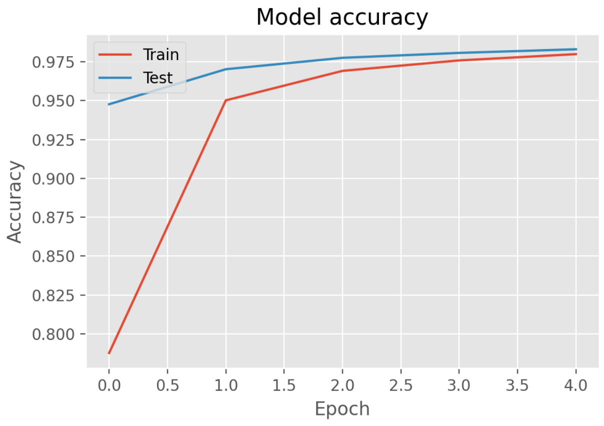
Some remarks we can draw:

- Compared to the original 1st experiment (With default parameters), the only thing that seemed to improve was the validation accuracy, but that too very marginally.
-

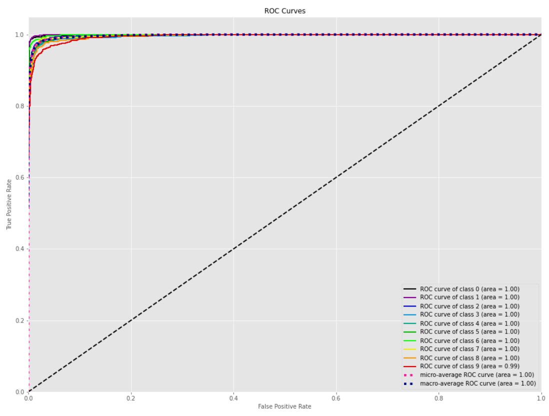
4.3.5 Experiment 5

This time the activation function for both the 2D convolution layers was changed from tanh to sigmoid.

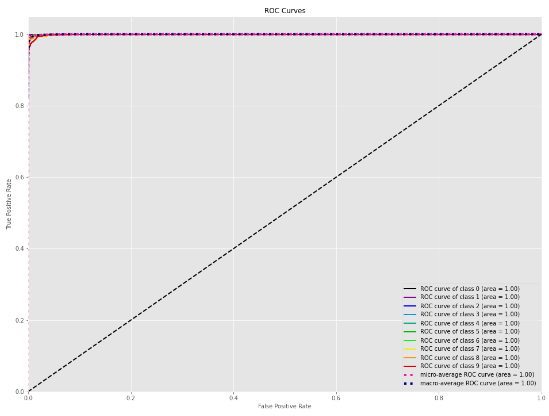
Epoch	Accuracy	Precision	Recall	Val. Acc.
1	0.7877	0.8726	0.7957	0.9477
2	0.9501	1.000	0.9836	0.9702
3	0.9691	1.000	0.9914	0.9775
4	0.9758	1.000	0.9947	0.9807
5	0.9799	1.000	0.9960	0.9830



(a) Accuracy with epoch



(b) ROC curve after 1st epoch



(c) ROC curve after 5th epoch

Figure 19: Metrics for experiment 5

Some remarks we can draw:

- The performance seriously degraded atleast for the first epoch. The accuracy went to 77%
- For rest of epochs, it quickly picked up and accuracy went high, but not as high as other models.

- Even if we increase the epochs to 10, the model reaches upto 98%, which is about the same as other models with lesser epochs.

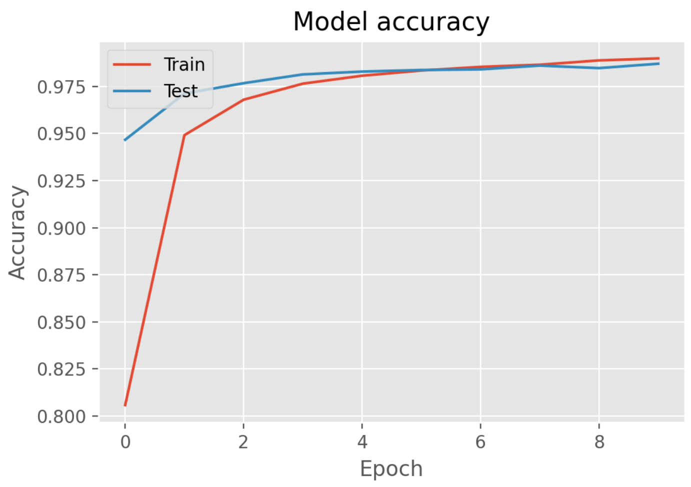


Figure 20: Accuracy with epoch, for 10 epochs

4.4 *LeNet Architecture Remarks*

Based on the above experiments for LeNet architecure, we can draw somex conclusions/remarks.

- The LeNet architecture seems to work better than the CNN architecture or the basic one, even with the default parameters.
- Going after too many epochs seems too pointless. For most experiments, the rate of increase of accuracy seems to decline after 4-5 epochs. Going all the way to 10 epochs (as seen from 2 experiments) either reduced the accuracy, or increased it way too slowly.
- The confusion matrix was shown for only 1st experiment. Though it was generated for all experiments and for all epochs, there wasn't much point looking at it. The major diagonal section was obviously where most of the numbers were. Only there were few numbers which were not predicted correctly.

5 Conclusion

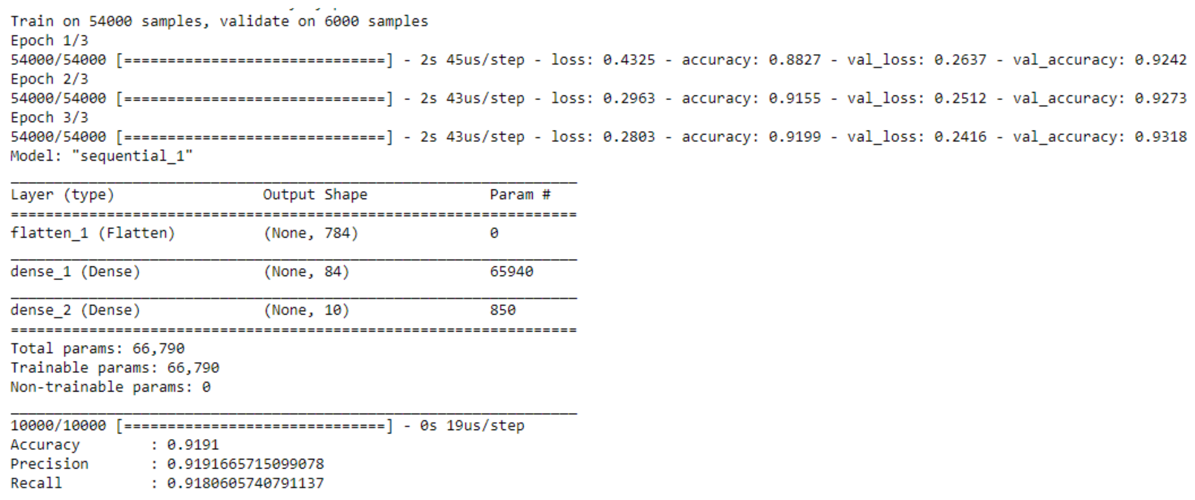
From all the networks we have built and all the experiments we did, we can conclude several points below:

- Modifying parameters such as batch size, number of epochs, and learning rate does affect the resulting performance of the network, especially learning rate (we can see this from the 2nd experiment in Basic Network).
- Adding convolutional layer does improve the performance (with the right parameters). From basic network to convolutional network we got about 5% increase in overall accuracy.
- The LeNet architecture was a significant improvement over the CNN Network as well as basic network. However, there wasn't any scope of improvement for this architecture, as per experiments performed in Section 4 (LeNet Architecture). As in, the architecture's performance would increase only very marginally when tweaking with the parameters.
- A small remark: Initially we were skeptical about the function $mnist.load\_data()$ which was being used to load the data. We thought that everytime running this function was loading the data dynamically, i.e. the training and testing data might've been different on every call. But upon looking up on an online forum and looking at the source code of this built-in function, it was seen that the function already had separated it into training and testing. Hence every call of $mnist.load\_data()$ was resulting in the same training and testing data only. Ideally, we should manually separate them into training and testing so as to ensure that we're always training/testing at the same dataset whenever we tweak the parameters and re-run the whole network.

6 Appendix

6.1 Training Results

6.1.1 Basic Network



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/3
54000/54000 [==============================] - 2s 45us/step - loss: 0.4325 - accuracy: 0.8827 - val_loss: 0.2637 - val_accuracy: 0.9242
Epoch 2/3
54000/54000 [==============================] - 2s 43us/step - loss: 0.2963 - accuracy: 0.9155 - val_loss: 0.2512 - val_accuracy: 0.9273
Epoch 3/3
54000/54000 [==============================] - 2s 43us/step - loss: 0.2803 - accuracy: 0.9199 - val_loss: 0.2416 - val_accuracy: 0.9318
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_1 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 84)                65940     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850       
=================================================================
Total params: 66,790
Trainable params: 66,790
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 0s 19us/step
Accuracy        : 0.9191
Precision       : 0.9191665715099078
Recall          : 0.9180605740791137
```

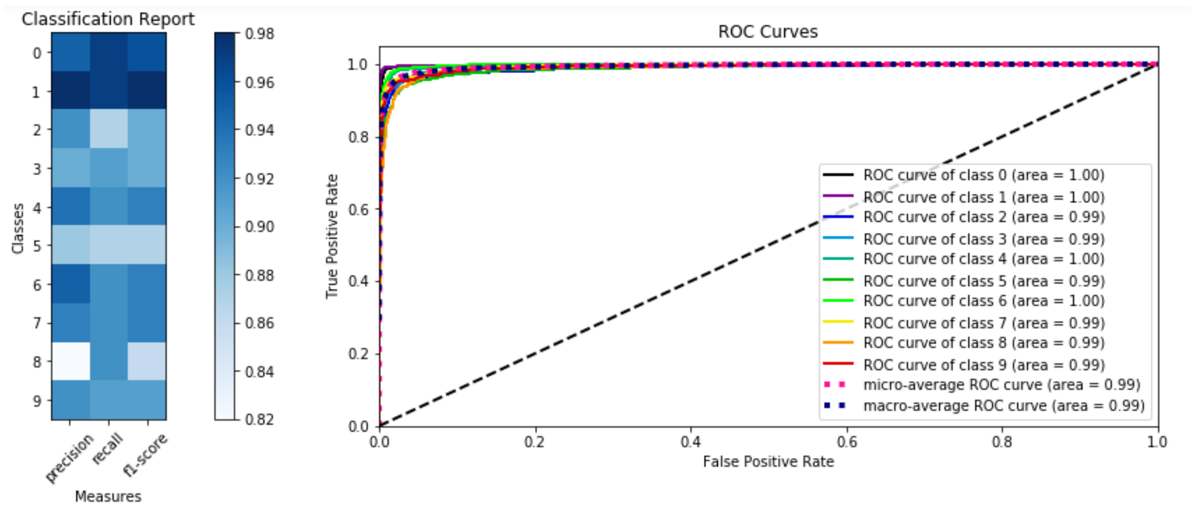


Figure 21: 1st Experiment

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/3
54000/54000 [==============================] - 3s 48us/step - loss: 9.6313 - accuracy: 0.3996 - val_loss: 9.7232 - val_accuracy: 0.3965
Epoch 2/3
54000/54000 [==============================] - 2s 45us/step - loss: 8.7633 - accuracy: 0.4561 - val_loss: 8.8250 - val_accuracy: 0.4523
Epoch 3/3
54000/54000 [==============================] - 2s 45us/step - loss: 9.0674 - accuracy: 0.4373 - val_loss: 9.1014 - val_accuracy: 0.4353
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 84)                65940
_________________________________________________________________
dense_4 (Dense)              (None, 10)                850
=================================================================
Total params: 66,790
Trainable params: 66,790
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 0s 21us/step
Accuracy         : 0.4369
Precision        : 0.2948389421748408
Recall           : 0.428036143249982
```

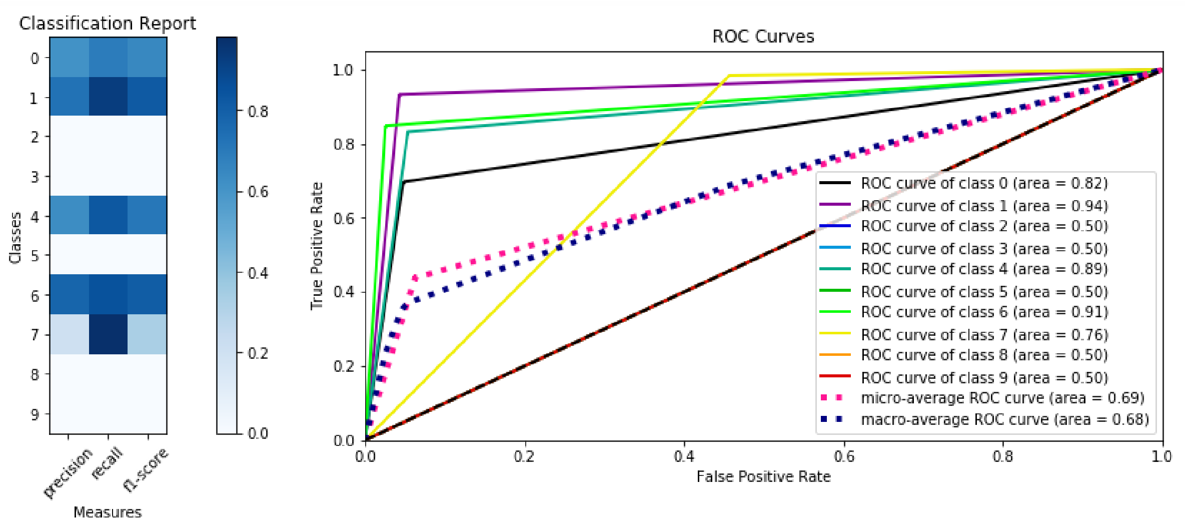
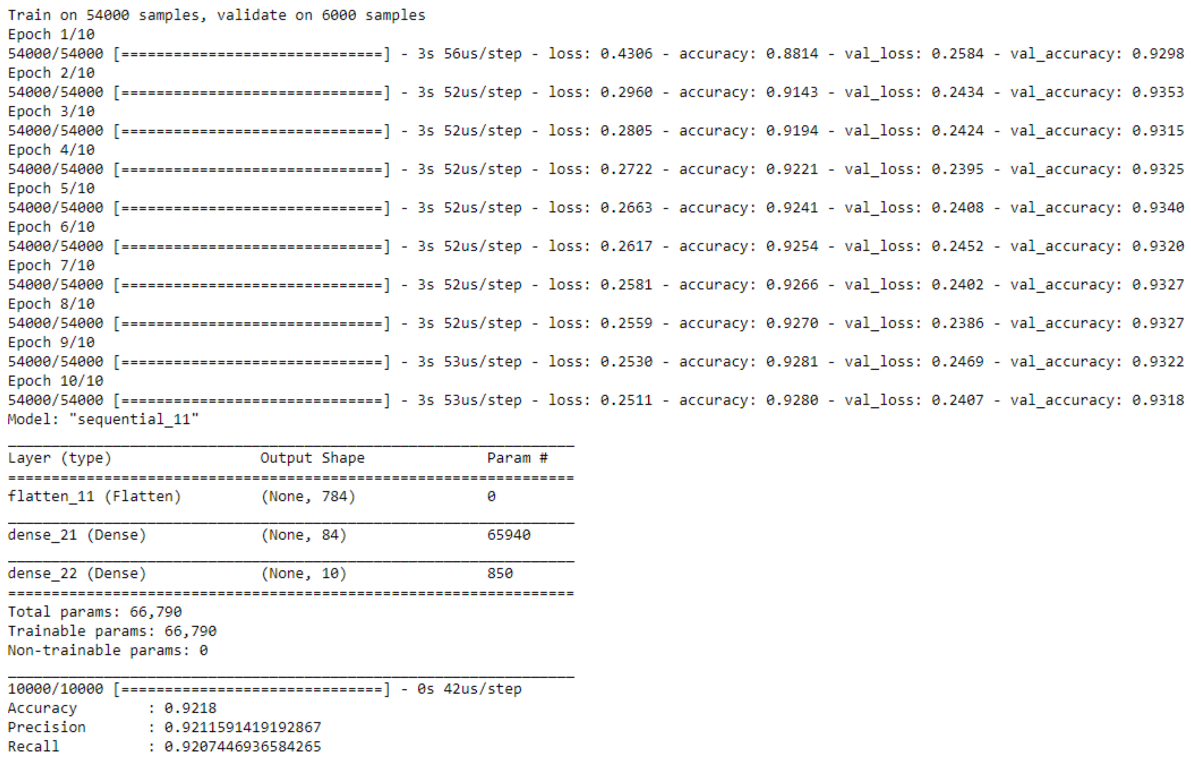


Figure 22: 2nd Experiment



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 3s 56us/step - loss: 0.4306 - accuracy: 0.8814 - val_loss: 0.2584 - val_accuracy: 0.9298
Epoch 2/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2960 - accuracy: 0.9143 - val_loss: 0.2434 - val_accuracy: 0.9353
Epoch 3/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2805 - accuracy: 0.9194 - val_loss: 0.2424 - val_accuracy: 0.9315
Epoch 4/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2722 - accuracy: 0.9221 - val_loss: 0.2395 - val_accuracy: 0.9325
Epoch 5/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2663 - accuracy: 0.9241 - val_loss: 0.2408 - val_accuracy: 0.9340
Epoch 6/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2617 - accuracy: 0.9254 - val_loss: 0.2452 - val_accuracy: 0.9320
Epoch 7/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2581 - accuracy: 0.9266 - val_loss: 0.2402 - val_accuracy: 0.9327
Epoch 8/10
54000/54000 [==============================] - 3s 52us/step - loss: 0.2559 - accuracy: 0.9270 - val_loss: 0.2386 - val_accuracy: 0.9327
Epoch 9/10
54000/54000 [==============================] - 3s 53us/step - loss: 0.2530 - accuracy: 0.9281 - val_loss: 0.2469 - val_accuracy: 0.9322
Epoch 10/10
54000/54000 [==============================] - 3s 53us/step - loss: 0.2511 - accuracy: 0.9280 - val_loss: 0.2407 - val_accuracy: 0.9318
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_11 (Flatten)         (None, 784)               0         
_________________________________________________________________
dense_21 (Dense)             (None, 84)                65940     
_________________________________________________________________
dense_22 (Dense)             (None, 10)                850       
=================================================================
Total params: 66,790
Trainable params: 66,790
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 0s 42us/step
Accuracy        : 0.9218
Precision       : 0.9211591419192867
Recall          : 0.9207446936584265
```

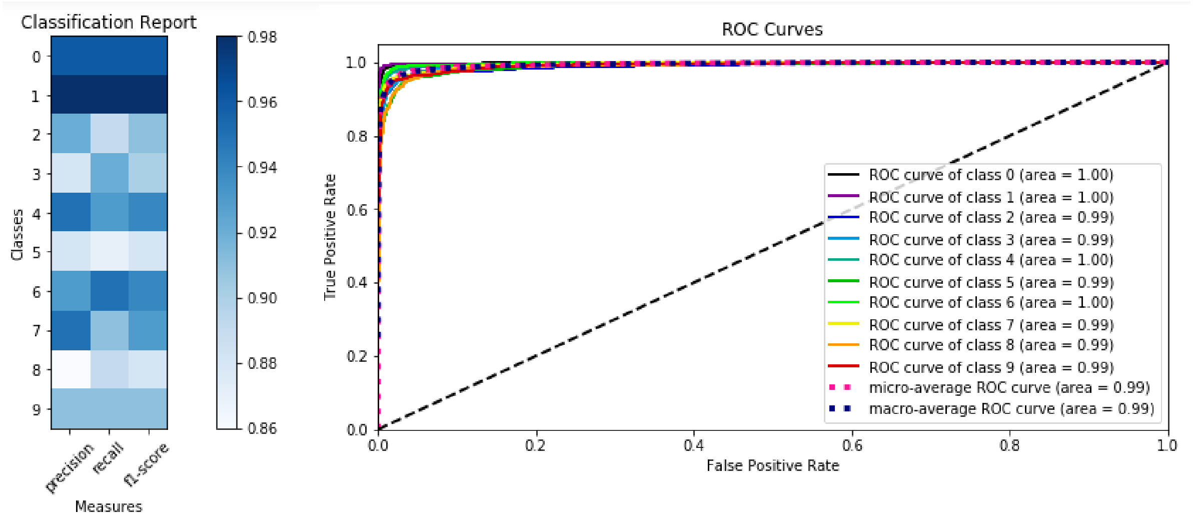


Figure 23: 3rd Experiment

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/3
54000/54000 [==============================] - 1s 24us/step - loss: 0.8850 - accuracy: 0.7900 - val_loss: 0.3808 - val_accuracy: 0.9052
Epoch 2/3
54000/54000 [==============================] - 1s 20us/step - loss: 0.3890 - accuracy: 0.8914 - val_loss: 0.2911 - val_accuracy: 0.9208
Epoch 3/3
54000/54000 [==============================] - 1s 21us/step - loss: 0.3309 - accuracy: 0.9072 - val_loss: 0.2646 - val_accuracy: 0.9277
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_7 (Dense)              (None, 84)                65940
_________________________________________________________________
dense_8 (Dense)              (None, 10)                850
=================================================================
Total params: 66,790
Trainable params: 66,790
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 0s 24us/step
Accuracy        : 0.9157
Precision       : 0.9151870092297928
Recall          : 0.9145439827420823
```

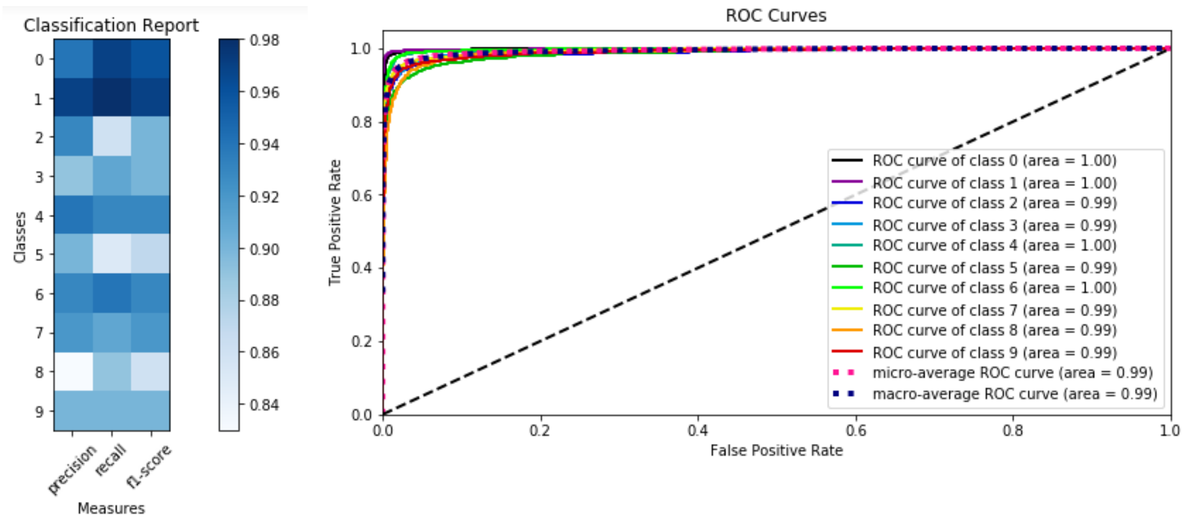
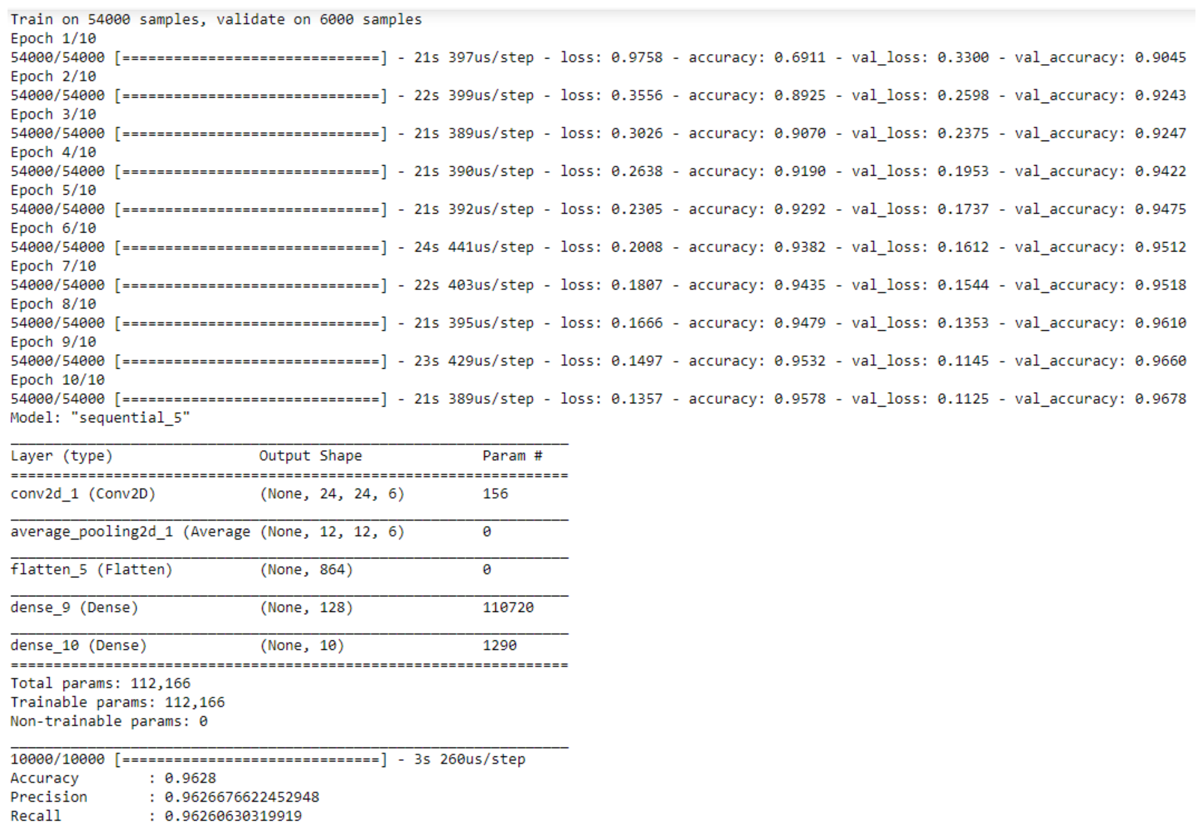


Figure 24: 4th Experiment

6.1.2 Convolutional Network



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 21s 397us/step - loss: 0.9758 - accuracy: 0.6911 - val_loss: 0.3300 - val_accuracy: 0.9045
Epoch 2/10
54000/54000 [==============================] - 22s 399us/step - loss: 0.3556 - accuracy: 0.8925 - val_loss: 0.2598 - val_accuracy: 0.9243
Epoch 3/10
54000/54000 [==============================] - 21s 389us/step - loss: 0.3026 - accuracy: 0.9070 - val_loss: 0.2375 - val_accuracy: 0.9247
Epoch 4/10
54000/54000 [==============================] - 21s 390us/step - loss: 0.2638 - accuracy: 0.9190 - val_loss: 0.1953 - val_accuracy: 0.9422
Epoch 5/10
54000/54000 [==============================] - 21s 392us/step - loss: 0.2305 - accuracy: 0.9292 - val_loss: 0.1737 - val_accuracy: 0.9475
Epoch 6/10
54000/54000 [==============================] - 24s 441us/step - loss: 0.2008 - accuracy: 0.9382 - val_loss: 0.1612 - val_accuracy: 0.9512
Epoch 7/10
54000/54000 [==============================] - 22s 403us/step - loss: 0.1807 - accuracy: 0.9435 - val_loss: 0.1544 - val_accuracy: 0.9518
Epoch 8/10
54000/54000 [==============================] - 21s 395us/step - loss: 0.1666 - accuracy: 0.9479 - val_loss: 0.1353 - val_accuracy: 0.9610
Epoch 9/10
54000/54000 [==============================] - 23s 429us/step - loss: 0.1497 - accuracy: 0.9532 - val_loss: 0.1145 - val_accuracy: 0.9660
Epoch 10/10
54000/54000 [==============================] - 21s 389us/step - loss: 0.1357 - accuracy: 0.9578 - val_loss: 0.1125 - val_accuracy: 0.9678
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 6)         156
_________________________________________________________________
average_pooling2d_1 (Average (None, 12, 12, 6)         0
_________________________________________________________________
flatten_5 (Flatten)          (None, 864)               0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               110720
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290
=================================================================
Total params: 112,166
Trainable params: 112,166
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 3s 260us/step
Accuracy        : 0.9628
Precision       : 0.9626676622452948
Recall          : 0.9626063031991 9
```

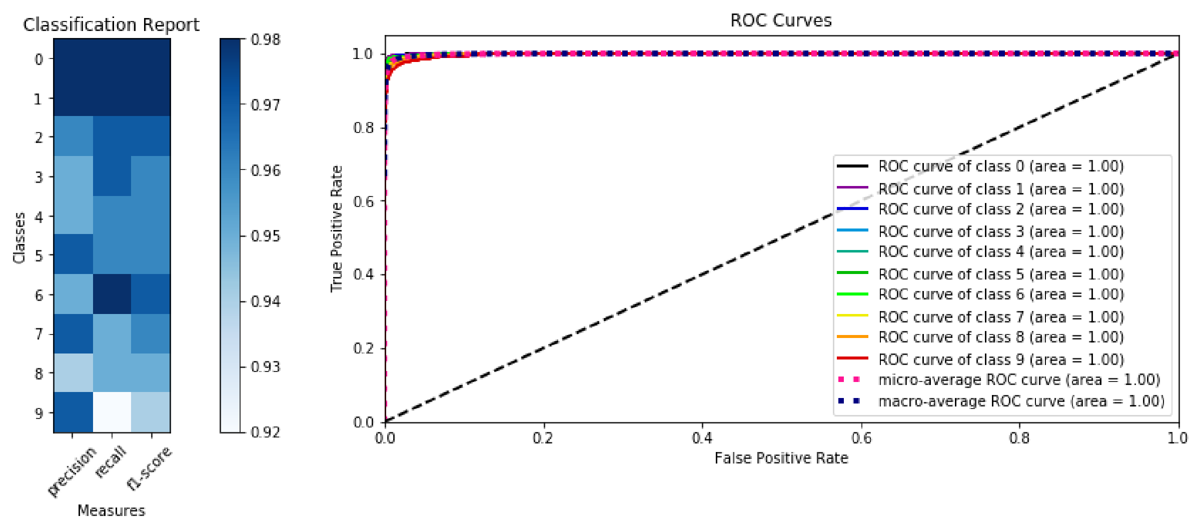
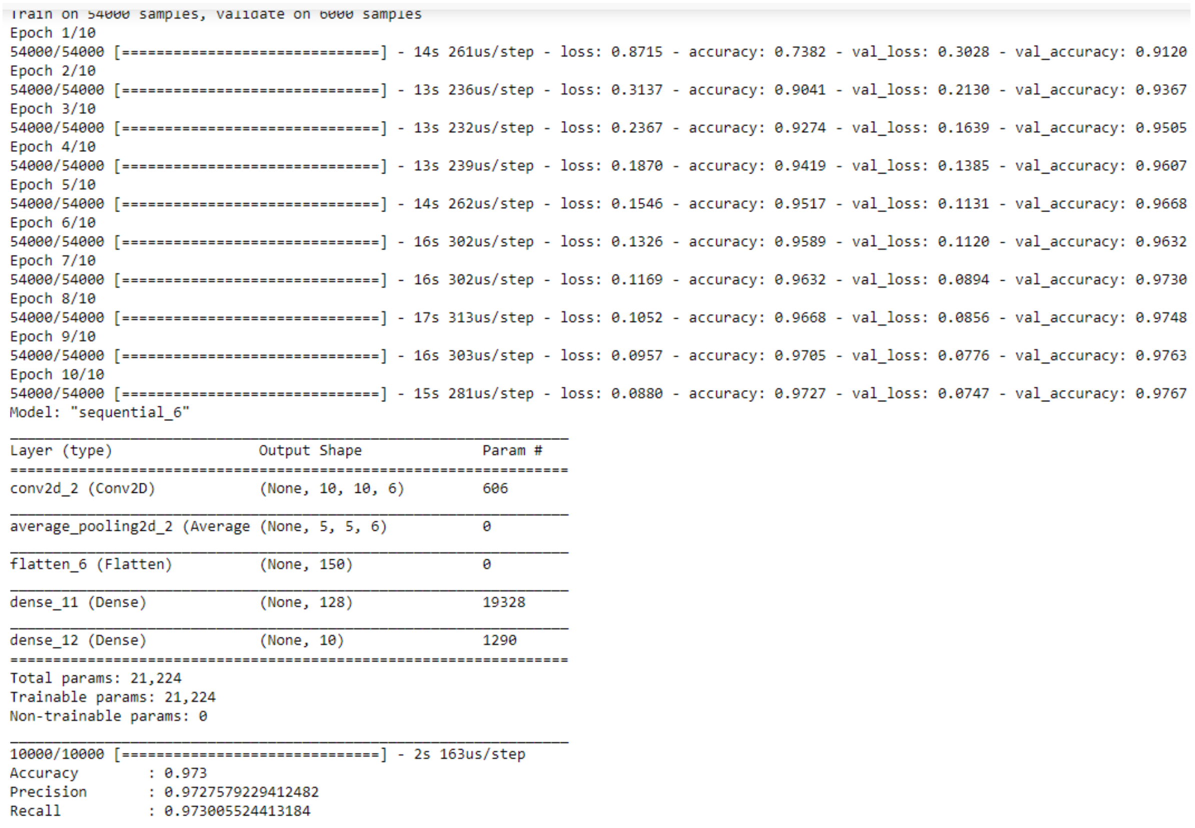


Figure 25: 1st Experiment



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 14s 261us/step - loss: 0.8715 - accuracy: 0.7382 - val_loss: 0.3028 - val_accuracy: 0.9120
Epoch 2/10
54000/54000 [==============================] - 13s 236us/step - loss: 0.3137 - accuracy: 0.9041 - val_loss: 0.2130 - val_accuracy: 0.9367
Epoch 3/10
54000/54000 [==============================] - 13s 232us/step - loss: 0.2367 - accuracy: 0.9274 - val_loss: 0.1639 - val_accuracy: 0.9505
Epoch 4/10
54000/54000 [==============================] - 13s 239us/step - loss: 0.1870 - accuracy: 0.9419 - val_loss: 0.1385 - val_accuracy: 0.9607
Epoch 5/10
54000/54000 [==============================] - 14s 262us/step - loss: 0.1546 - accuracy: 0.9517 - val_loss: 0.1131 - val_accuracy: 0.9668
Epoch 6/10
54000/54000 [==============================] - 16s 302us/step - loss: 0.1326 - accuracy: 0.9589 - val_loss: 0.1120 - val_accuracy: 0.9632
Epoch 7/10
54000/54000 [==============================] - 16s 302us/step - loss: 0.1169 - accuracy: 0.9632 - val_loss: 0.0894 - val_accuracy: 0.9730
Epoch 8/10
54000/54000 [==============================] - 17s 313us/step - loss: 0.1052 - accuracy: 0.9668 - val_loss: 0.0856 - val_accuracy: 0.9748
Epoch 9/10
54000/54000 [==============================] - 16s 303us/step - loss: 0.0957 - accuracy: 0.9705 - val_loss: 0.0776 - val_accuracy: 0.9763
Epoch 10/10
54000/54000 [==============================] - 15s 281us/step - loss: 0.0880 - accuracy: 0.9727 - val_loss: 0.0747 - val_accuracy: 0.9767
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 10, 10, 6)         606       
_________________________________________________________________
average_pooling2d_2 (Average (None, 5, 5, 6)           0         
_________________________________________________________________
flatten_6 (Flatten)          (None, 150)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 128)               19328     
_________________________________________________________________
dense_12 (Dense)             (None, 10)                1290      
=================================================================
Total params: 21,224
Trainable params: 21,224
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 2s 163us/step
Accuracy        : 0.973
Precision       : 0.9727579229412482
Recall          : 0.973005524413184
```

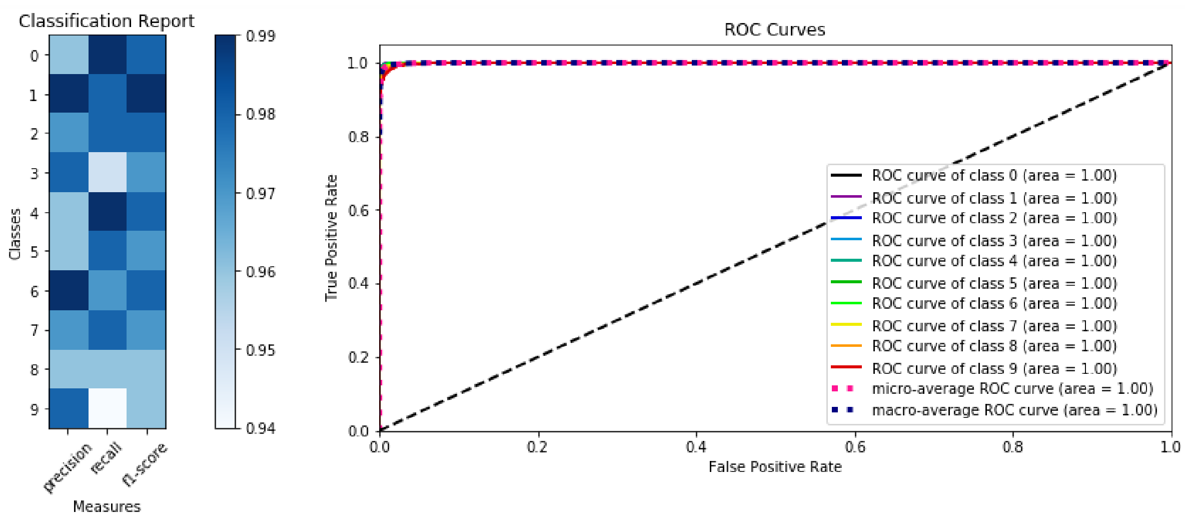


Figure 26: 2nd Experiment

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 26s 485us/step - loss: 1.2927 - accuracy: 0.6032 - val_loss: 0.4277 - val_accuracy: 0.8892
Epoch 2/10
54000/54000 [==============================] - 26s 476us/step - loss: 0.4118 - accuracy: 0.8789 - val_loss: 0.2894 - val_accuracy: 0.9145
Epoch 3/10
54000/54000 [==============================] - 26s 483us/step - loss: 0.3367 - accuracy: 0.8999 - val_loss: 0.2481 - val_accuracy: 0.9250
Epoch 4/10
54000/54000 [==============================] - 26s 477us/step - loss: 0.3055 - accuracy: 0.9066 - val_loss: 0.2212 - val_accuracy: 0.9330
Epoch 5/10
54000/54000 [==============================] - 26s 477us/step - loss: 0.2796 - accuracy: 0.9140 - val_loss: 0.2185 - val_accuracy: 0.9312
Epoch 6/10
54000/54000 [==============================] - 26s 484us/step - loss: 0.2554 - accuracy: 0.9232 - val_loss: 0.1847 - val_accuracy: 0.9418
Epoch 7/10
54000/54000 [==============================] - 25s 472us/step - loss: 0.2316 - accuracy: 0.9297 - val_loss: 0.1661 - val_accuracy: 0.9488
Epoch 8/10
54000/54000 [==============================] - 27s 498us/step - loss: 0.2088 - accuracy: 0.9364 - val_loss: 0.1561 - val_accuracy: 0.9528
Epoch 9/10
54000/54000 [==============================] - 26s 475us/step - loss: 0.1889 - accuracy: 0.9420 - val_loss: 0.1310 - val_accuracy: 0.9615
Epoch 10/10
54000/54000 [==============================] - 26s 474us/step - loss: 0.1690 - accuracy: 0.9475 - val_loss: 0.1295 - val_accuracy: 0.9633
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 24, 24, 6)         156       
_________________________________________________________________
average_pooling2d_3 (Average (None, 6, 6, 6)           0         
_________________________________________________________________
flatten_7 (Flatten)          (None, 216)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 128)               27776     
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290      
=================================================================
Total params: 29,222
Trainable params: 29,222
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 3s 290us/step
Accuracy        : 0.9524
Precision       : 0.9531985750145472
Recall          : 0.9518970397538131
```

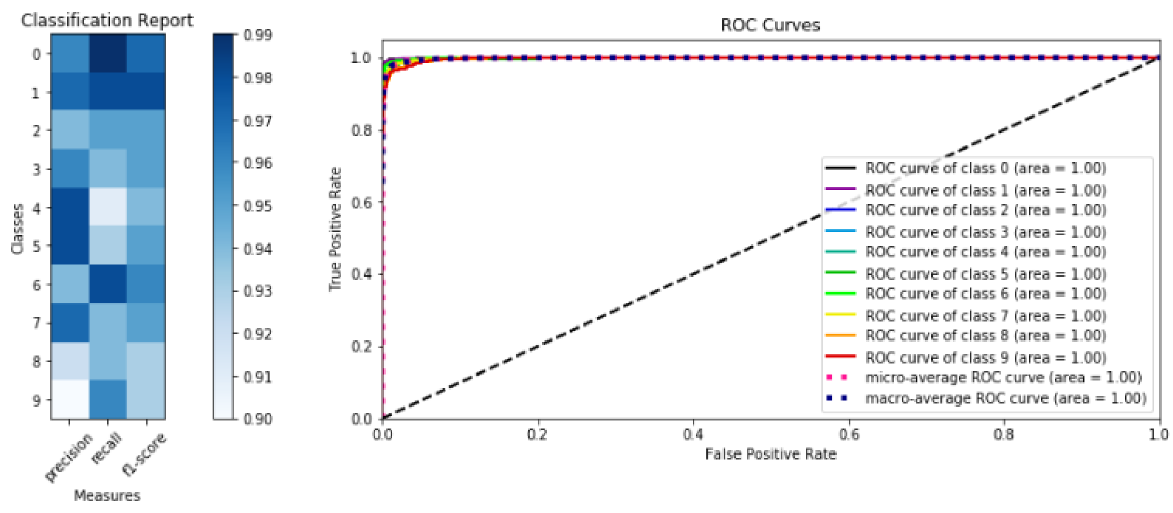
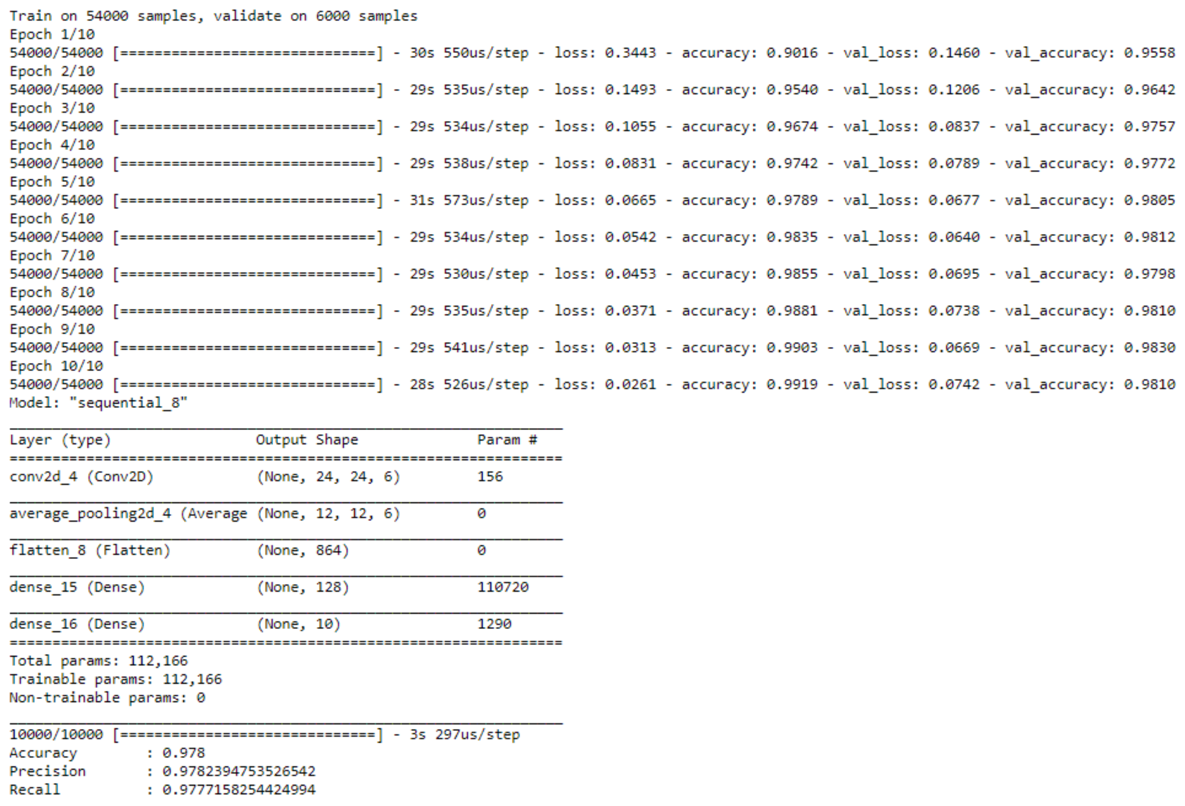


Figure 27: 3rd Experiment



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 30s 550us/step - loss: 0.3443 - accuracy: 0.9016 - val_loss: 0.1460 - val_accuracy: 0.9558
Epoch 2/10
54000/54000 [==============================] - 29s 535us/step - loss: 0.1493 - accuracy: 0.9540 - val_loss: 0.1206 - val_accuracy: 0.9642
Epoch 3/10
54000/54000 [==============================] - 29s 534us/step - loss: 0.1055 - accuracy: 0.9674 - val_loss: 0.0837 - val_accuracy: 0.9757
Epoch 4/10
54000/54000 [==============================] - 29s 538us/step - loss: 0.0831 - accuracy: 0.9742 - val_loss: 0.0789 - val_accuracy: 0.9772
Epoch 5/10
54000/54000 [==============================] - 31s 573us/step - loss: 0.0665 - accuracy: 0.9789 - val_loss: 0.0677 - val_accuracy: 0.9805
Epoch 6/10
54000/54000 [==============================] - 29s 534us/step - loss: 0.0542 - accuracy: 0.9835 - val_loss: 0.0640 - val_accuracy: 0.9812
Epoch 7/10
54000/54000 [==============================] - 29s 530us/step - loss: 0.0453 - accuracy: 0.9855 - val_loss: 0.0695 - val_accuracy: 0.9798
Epoch 8/10
54000/54000 [==============================] - 29s 535us/step - loss: 0.0371 - accuracy: 0.9881 - val_loss: 0.0738 - val_accuracy: 0.9810
Epoch 9/10
54000/54000 [==============================] - 29s 541us/step - loss: 0.0313 - accuracy: 0.9903 - val_loss: 0.0669 - val_accuracy: 0.9830
Epoch 10/10
54000/54000 [==============================] - 28s 526us/step - loss: 0.0261 - accuracy: 0.9919 - val_loss: 0.0742 - val_accuracy: 0.9810
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 24, 24, 6)         156       
_________________________________________________________________
average_pooling2d_4 (Average (None, 12, 12, 6)         0         
_________________________________________________________________
flatten_8 (Flatten)          (None, 864)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 128)               110720    
_________________________________________________________________
dense_16 (Dense)             (None, 10)                1290      
=================================================================
Total params: 112,166
Trainable params: 112,166
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 3s 297us/step
Accuracy        : 0.978
Precision       : 0.9782394753526542
Recall          : 0.9777158254424994
```

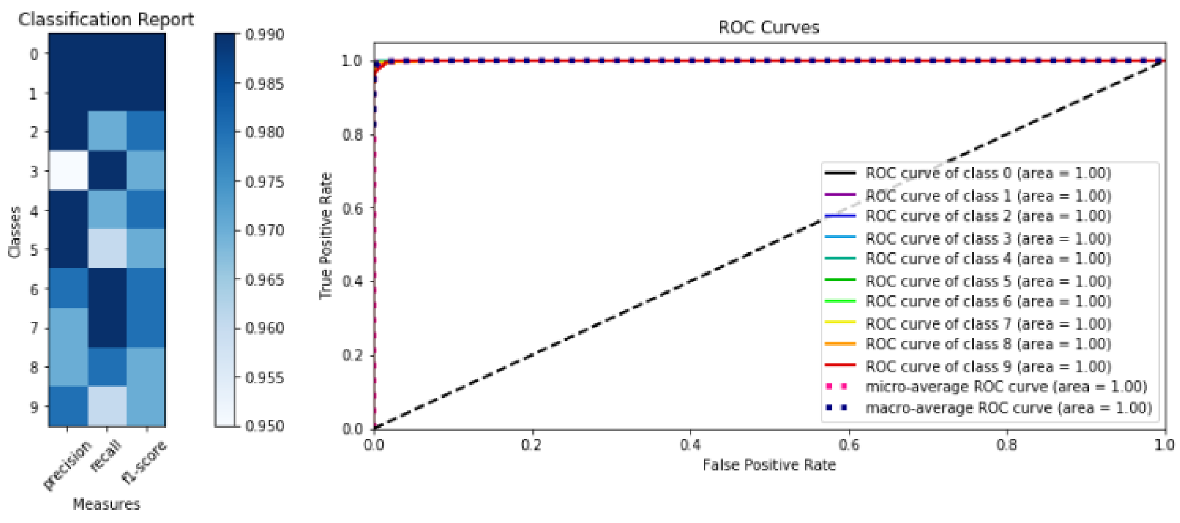
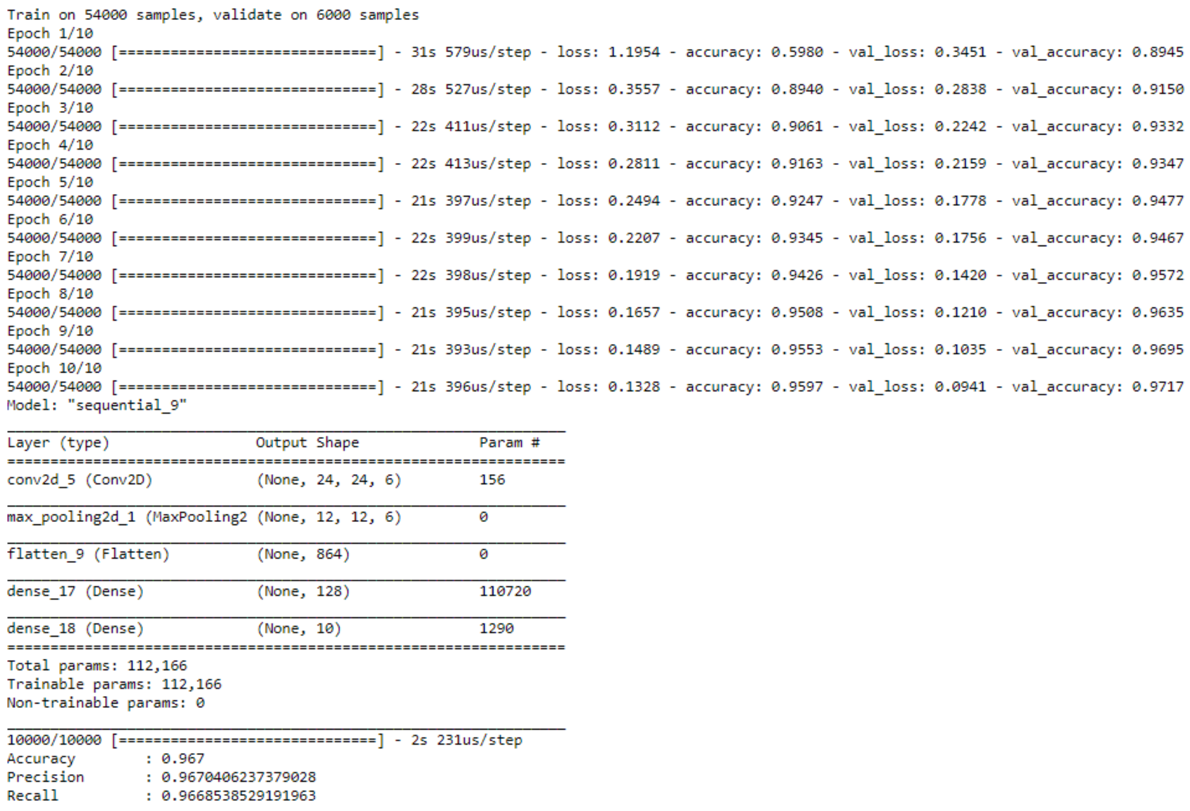


Figure 28: 4th Experiment



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 31s 579us/step - loss: 1.1954 - accuracy: 0.5980 - val_loss: 0.3451 - val_accuracy: 0.8945
Epoch 2/10
54000/54000 [==============================] - 28s 527us/step - loss: 0.3557 - accuracy: 0.8940 - val_loss: 0.2838 - val_accuracy: 0.9150
Epoch 3/10
54000/54000 [==============================] - 22s 411us/step - loss: 0.3112 - accuracy: 0.9061 - val_loss: 0.2242 - val_accuracy: 0.9332
Epoch 4/10
54000/54000 [==============================] - 22s 413us/step - loss: 0.2811 - accuracy: 0.9163 - val_loss: 0.2159 - val_accuracy: 0.9347
Epoch 5/10
54000/54000 [==============================] - 21s 397us/step - loss: 0.2494 - accuracy: 0.9247 - val_loss: 0.1778 - val_accuracy: 0.9477
Epoch 6/10
54000/54000 [==============================] - 22s 399us/step - loss: 0.2207 - accuracy: 0.9345 - val_loss: 0.1756 - val_accuracy: 0.9467
Epoch 7/10
54000/54000 [==============================] - 22s 398us/step - loss: 0.1919 - accuracy: 0.9426 - val_loss: 0.1420 - val_accuracy: 0.9572
Epoch 8/10
54000/54000 [==============================] - 21s 395us/step - loss: 0.1657 - accuracy: 0.9508 - val_loss: 0.1210 - val_accuracy: 0.9635
Epoch 9/10
54000/54000 [==============================] - 21s 393us/step - loss: 0.1489 - accuracy: 0.9553 - val_loss: 0.1035 - val_accuracy: 0.9695
Epoch 10/10
54000/54000 [==============================] - 21s 396us/step - loss: 0.1328 - accuracy: 0.9597 - val_loss: 0.0941 - val_accuracy: 0.9717
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 24, 24, 6)         156
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 6)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 864)               0
_________________________________________________________________
dense_17 (Dense)             (None, 128)               110720
_________________________________________________________________
dense_18 (Dense)             (None, 10)                1290
=================================================================
Total params: 112,166
Trainable params: 112,166
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 2s 231us/step
Accuracy        : 0.967
Precision       : 0.9670406237379028
Recall          : 0.9668538529191963
```

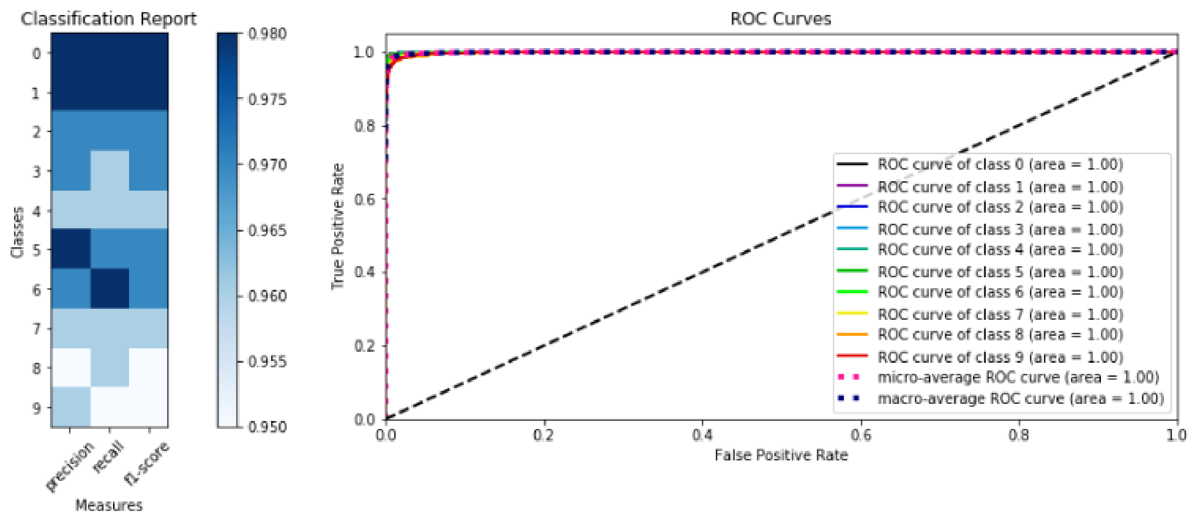
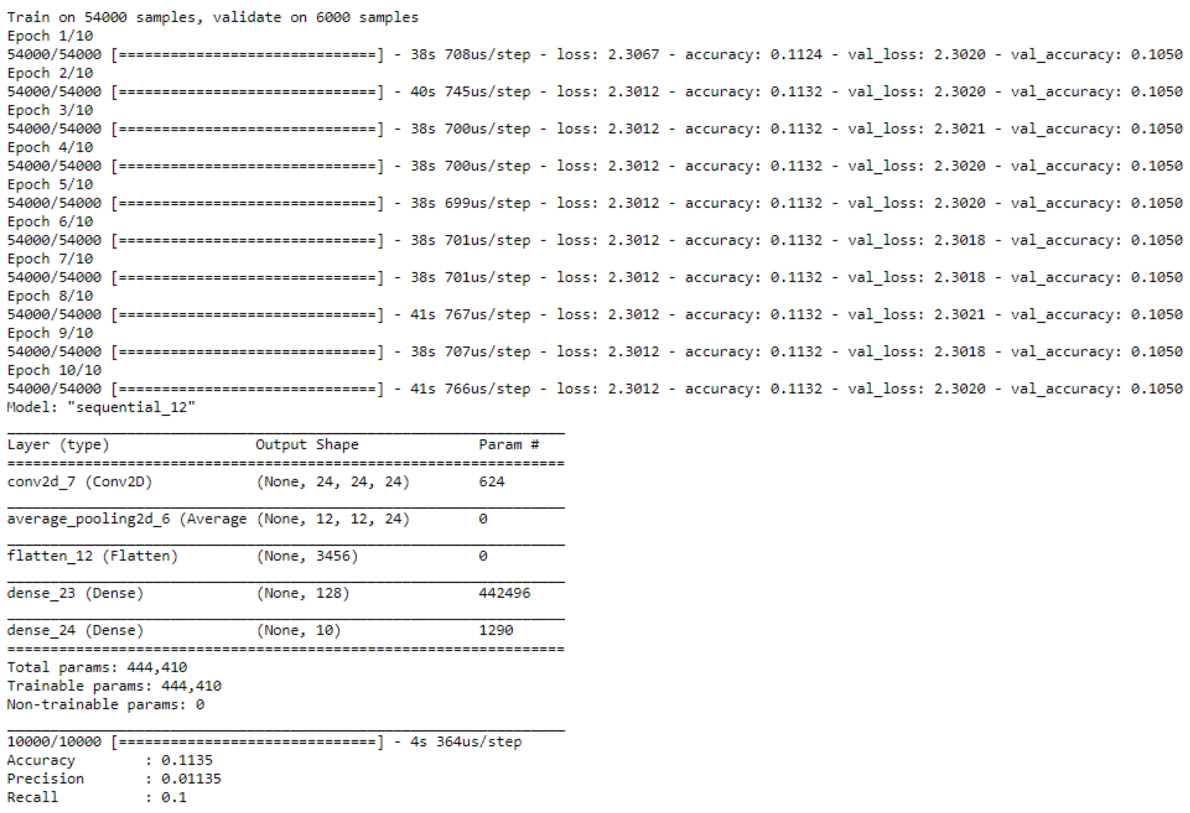


Figure 29: 5th Experiment



```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 38s 708us/step - loss: 2.3067 - accuracy: 0.1124 - val_loss: 2.3020 - val_accuracy: 0.1050
Epoch 2/10
54000/54000 [==============================] - 40s 745us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3020 - val_accuracy: 0.1050
Epoch 3/10
54000/54000 [==============================] - 38s 700us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3021 - val_accuracy: 0.1050
Epoch 4/10
54000/54000 [==============================] - 38s 700us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3020 - val_accuracy: 0.1050
Epoch 5/10
54000/54000 [==============================] - 38s 699us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3020 - val_accuracy: 0.1050
Epoch 6/10
54000/54000 [==============================] - 38s 701us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3018 - val_accuracy: 0.1050
Epoch 7/10
54000/54000 [==============================] - 38s 701us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3018 - val_accuracy: 0.1050
Epoch 8/10
54000/54000 [==============================] - 41s 767us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3021 - val_accuracy: 0.1050
Epoch 9/10
54000/54000 [==============================] - 38s 707us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3018 - val_accuracy: 0.1050
Epoch 10/10
54000/54000 [==============================] - 41s 766us/step - loss: 2.3012 - accuracy: 0.1132 - val_loss: 2.3020 - val_accuracy: 0.1050
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_7 (Conv2D)            (None, 24, 24, 24)        624
_________________________________________________________________
average_pooling2d_6 (Average (None, 12, 12, 24)        0
_________________________________________________________________
flatten_12 (Flatten)         (None, 3456)              0
_________________________________________________________________
dense_23 (Dense)             (None, 128)               442496
_________________________________________________________________
dense_24 (Dense)             (None, 10)                1290
=================================================================
Total params: 444,410
Trainable params: 444,410
Non-trainable params: 0
_________________________________________________________________
10000/10000 [==============================] - 4s 364us/step
Accuracy         : 0.1135
Precision        : 0.01135
Recall           : 0.1
```

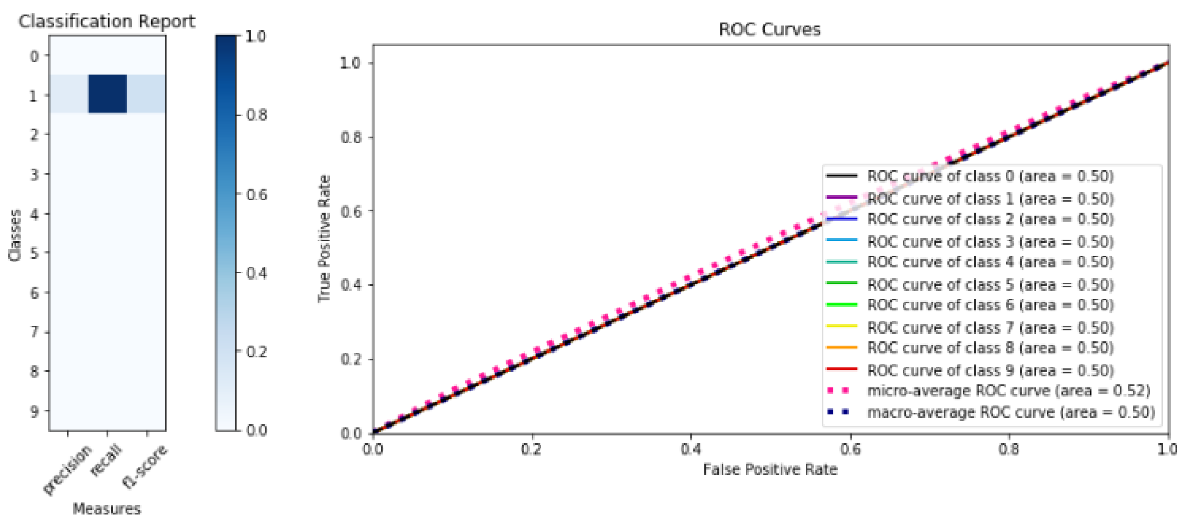


Figure 30: 6th Experiment