Name: Md Sabbir Rahman **ID**: 1705076

Image Data:

For dataset, we are using NumtaDB dataset. From this dataset, we have used *training-a*, *training-b*, *training-c* as training+validation data. *training-d* is used as test data. From the training+validation data, we have used a random 90% + 10% split to benchmark our model.

Image Preprocessing:

We have some preprocessing to each image. In particular we have loaded them using OpenCV as grayscale images. Then we have inverted them as white pixels should be lower value and black pixels should be higher value. Next, we have used *cv2.moments* to compute the centroid of the image and cropped the image so that the digits are somewhat centred. We have then applied an dilation of kernel of 1s of shape 2x2 (also using opencv). Finally we have scaled the image so that the max value becomes 1. The image is then resized to 28x28 size to be used by the CNN.

Structure of Convolutional Neural Network:

We have used a simple LeNet structure. The structure is described below (the actual layers are indented and data shapes are non-indented)

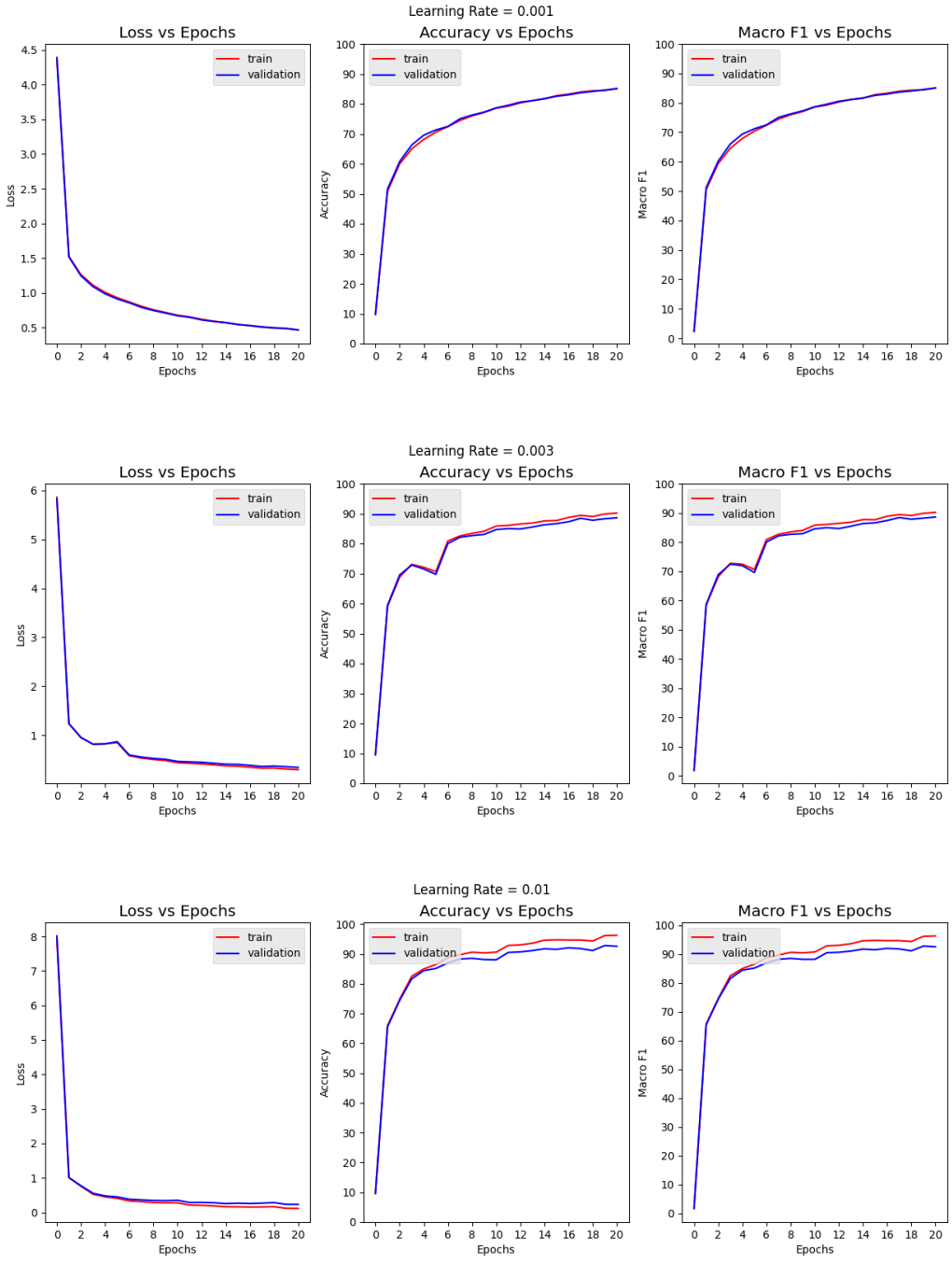
- 28 x 28 data with 1 channel
    - **Convolution layer of 6 kernels of size 5 x 5, with (1, 1) stride and 2 padding**
- 28 x 28 data with 6 channels
    - **ReLU activation layer**
- 28 x 28 data with 6 channels
    - **Max Pooling layer of 2 x 2 filter size and stride (2, 2)**
- 14 x 14 data with 6 channels
    - **Convolution layer of 16 kernels of size 5 x 5 with (1, 1) stride and 0 padding**
- 10 x 10 data with 16 channels
    - **ReLU activation layer**
- 10 x 10 data with 16 channels
    - **Max Pooling layer of 2 x 2 filter size and stride (2, 2)**
- 5 x 5 data with 16 channels
    - **Flattening layer**
- 400 nodes
    - **Dense Layer with 120 output nodes**
- 120 nodes
    - **ReLU activation layer**
- 120 nodes
    - **Dense Layer with 84 output nodes**
- 84 nodes
    - **ReLU activation layer**
- 84 nodes
    - **Dense Layer with 10 nodes**
- 10 nodes
    - **SofMaxLayer**

- 10 nodes

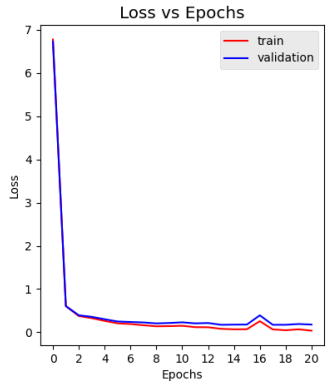
We have used minibatch gradient descent with categorical cross entropy as loss function. The batch size was chosen to be 64.

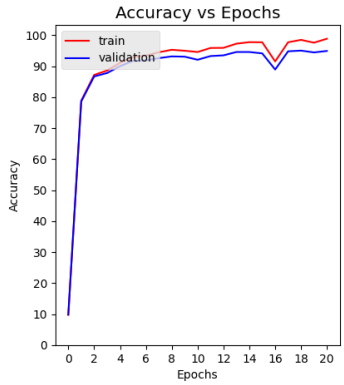
Performance for various learning rates:

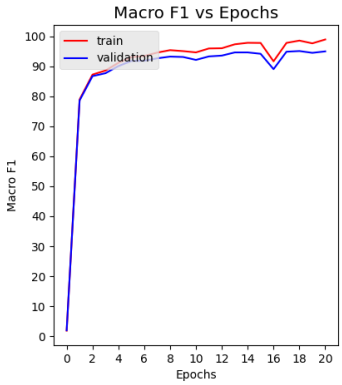
We trained the model on the same training + validation data for learning rates 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0; each for 20 epochs. Loss value, accuracy and macro f1 score were recorded for each epoch. The results are plotted below.



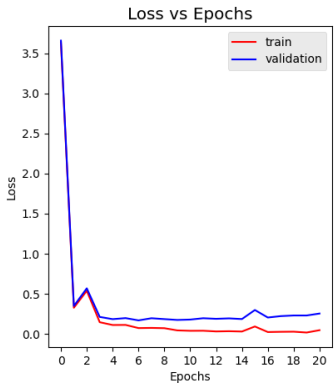
Learning Rate = 0.03

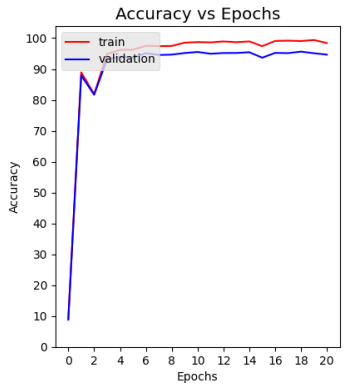


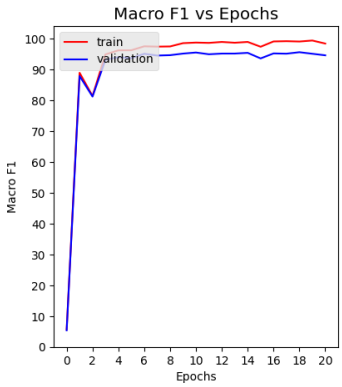




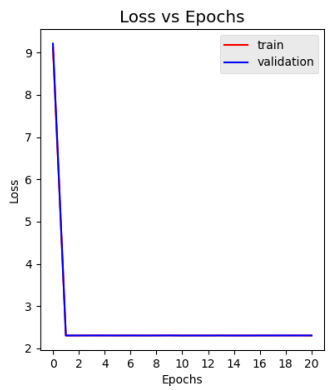
Learning Rate = 0.1

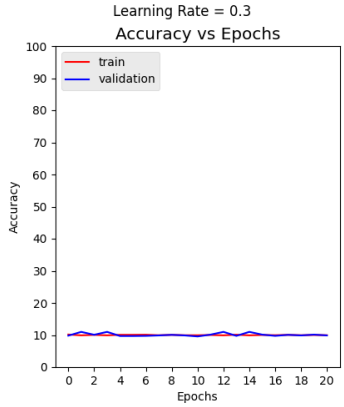


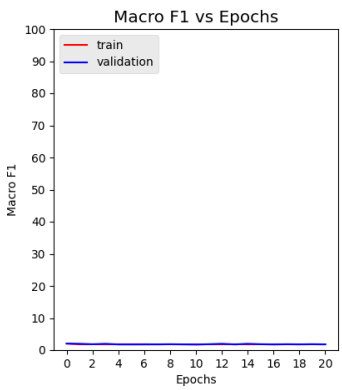




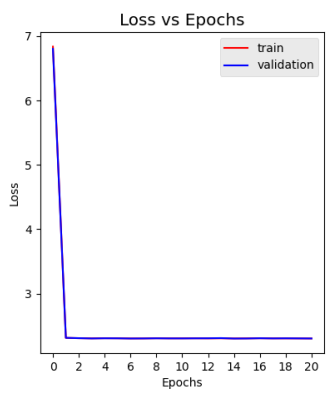
Learning Rate = 0.3

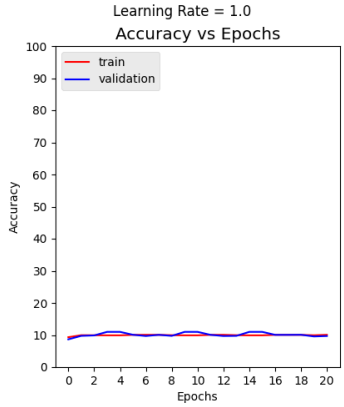


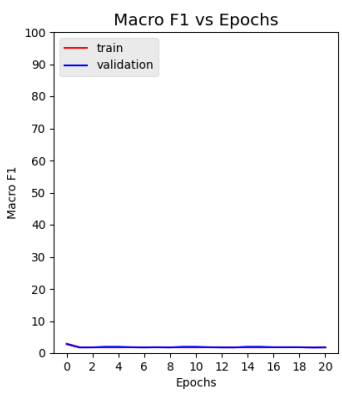




Learning Rate = 1.0



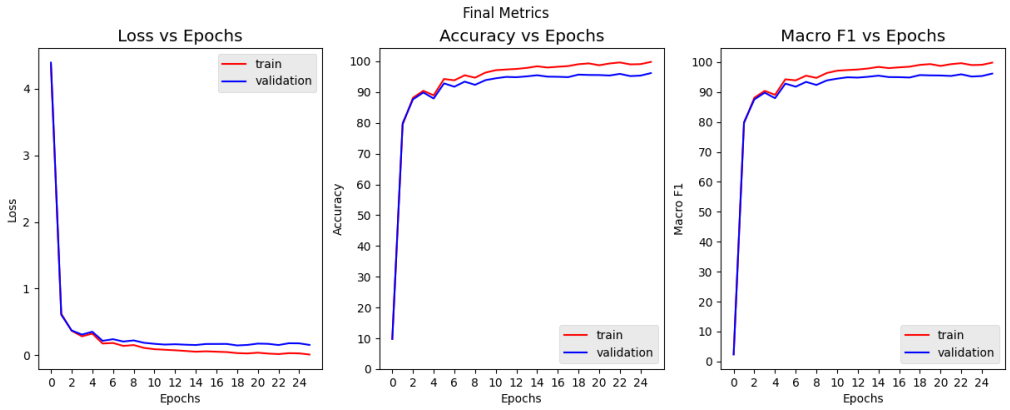




We can see that for rates 0.01, 0.03, and 0.1, the learning is quite fast and general. But in 0.1 validation metrics were lagging, So we chose 0.03 as the optimal learning rate.

Final Performance:

We trained the model again on the same parted data, but this time for more epochs, training with learning rate = 0.03 and epochs = 25, we get the following graph



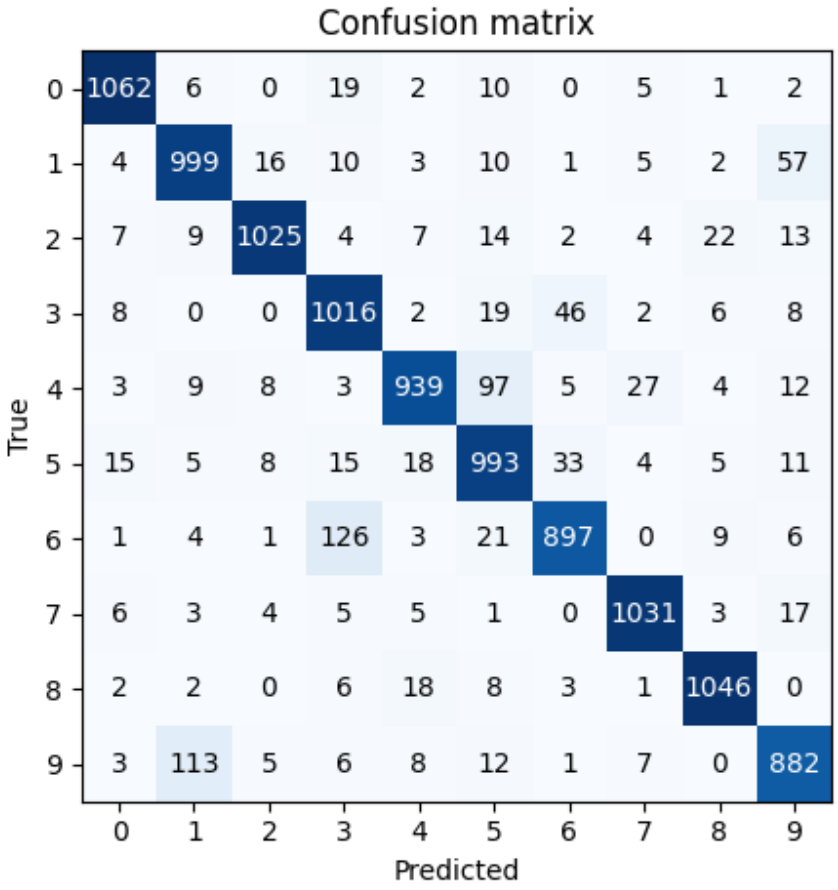
The final metrics were

	Loss	Accuracy	Macro f1
Training data	0.01025	99.807%	99.806%
Validation data	0.15471	96.167%	96.151%

Finally, to test the model, we trained the model again, but this time over all of *training-a, training-b, training-c.* Then we tested it on *training-d* data, which contained about 11000 labelled images. The metrics were

	Loss	Accuracy	Macro f1
Testing data	0.38881	90.667%	90.657%

The corresponding confusion matrix for test data is following


Confusion matrix
True
Predicted
0 1062 6 0 19 2 10 0 5 1 2
1 4 999 16 10 3 10 1 5 2 57
2 7 9 1025 4 7 14 2 4 22 13
3 8 0 0 1016 2 19 46 2 6 8
4 3 9 8 3 939 97 5 27 4 12
5 15 5 8 15 18 993 33 4 5 11
6 1 4 1 126 3 21 897 0 9 6
7 6 3 4 5 5 1 0 1031 3 17
8 2 2 0 6 18 8 3 1 1046 0
9 3 113 5 6 8 12 1 7 0 882
0 1 2 3 4 5 6 7 8 9