Recognition Using Deep Networks

Rahul Kumar
Northeastern University

1. SUMMARY

The purpose of the project is to do a Recognition task using Deep Networks. The task selected is to recognize MNIST digits and MNIST Fashion data. It is done by building an appropriate network architecture using CNN layers which will help to extract features from the input image and the extracted feature vector is then fed into classification layers and finally predicting the label of the input image. Then architecture is trained using MNIST data and is verified by calculating test accuracy. Upon training the network for MNIST data, transfer learning is carried out to classify the Greek letters. Finally, the hyperparameters tuning is done using Random search and optimizing the network performance for MNIST Fashion data.

2. MNIST DIGIT RECOGNITION

2.1. Loading MNIST Digit Data Set

The MNIST dataset is imported from the PyTorch torchvision dataset. The dataset consists of 70,000 handwritten digits with labels. There are 60,000 training images and 10,000 test images, all of which are 28*28 pixels. Below is the plot of the first six example digits:

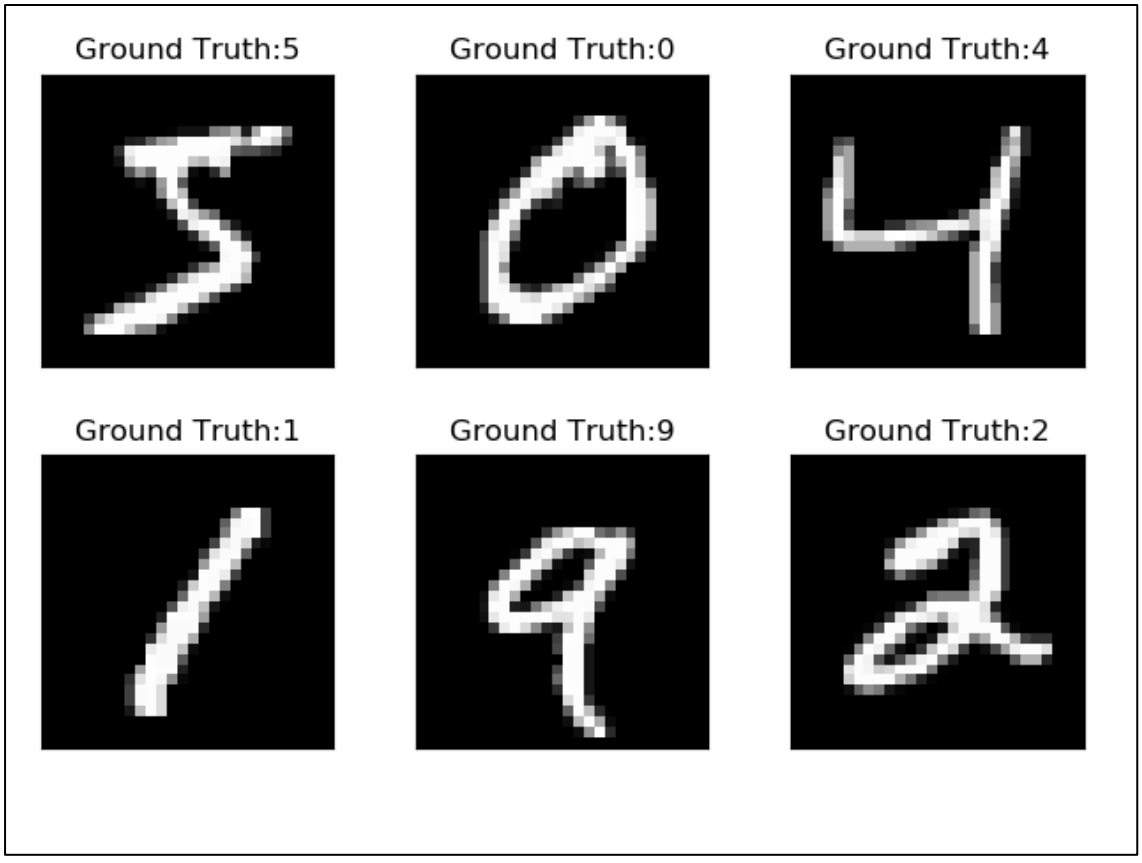

Fig. 1. Plot of first six example digits from the training set

2.2. Building network model

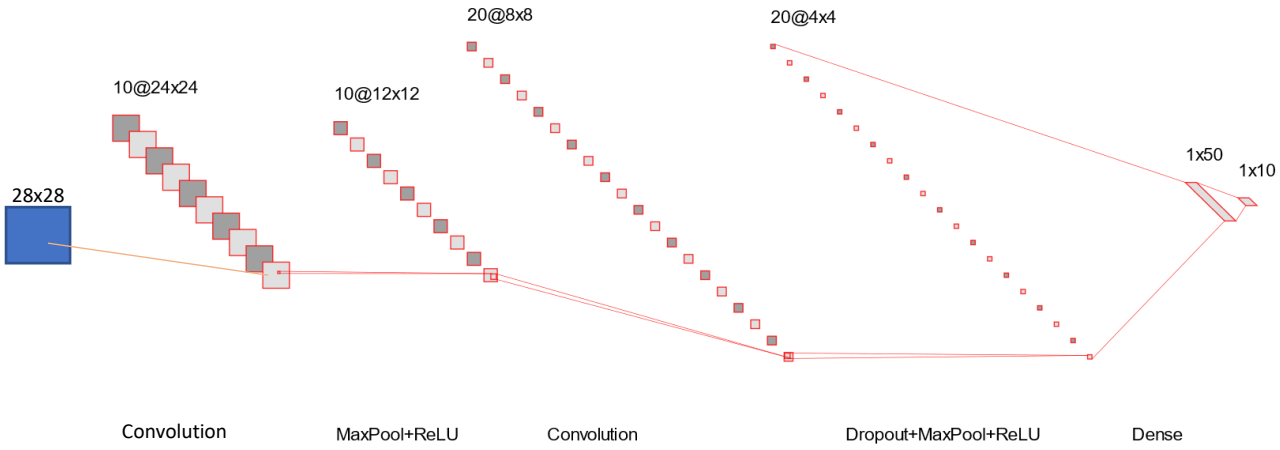

Fig. 2. Network architecture

The network consists of the following layers:

- Convolution layer with 10 5*5 filters
- Max pooling layer with 2*2 window and stride 2 followed by ReLU function
- Convolution layer with 20 5*5 filters
- Dropout layer with 0.5 dropout rate
- Max pooling layer with 2*2 window and stride 2 followed by ReLU function
- Fully connected dense layer with 50 nodes followed by ReLU function
- Fully connected Linear layer with 10 nodes followed by log_softmax function

2.3. Train Model

Once the model is designed, now the model is trained using the training dataset of MNIST. The following hyperparameters are used to train the model.

Batch Size = 64

Learning rate = 0.01

Momentum = 0.5

Epoch = 5

The test accuracy is found 98.6% after 5 epochs of training.

It is observed that training and testing loss decreased with each epoch. Plots of accuracy and loss are provided below:

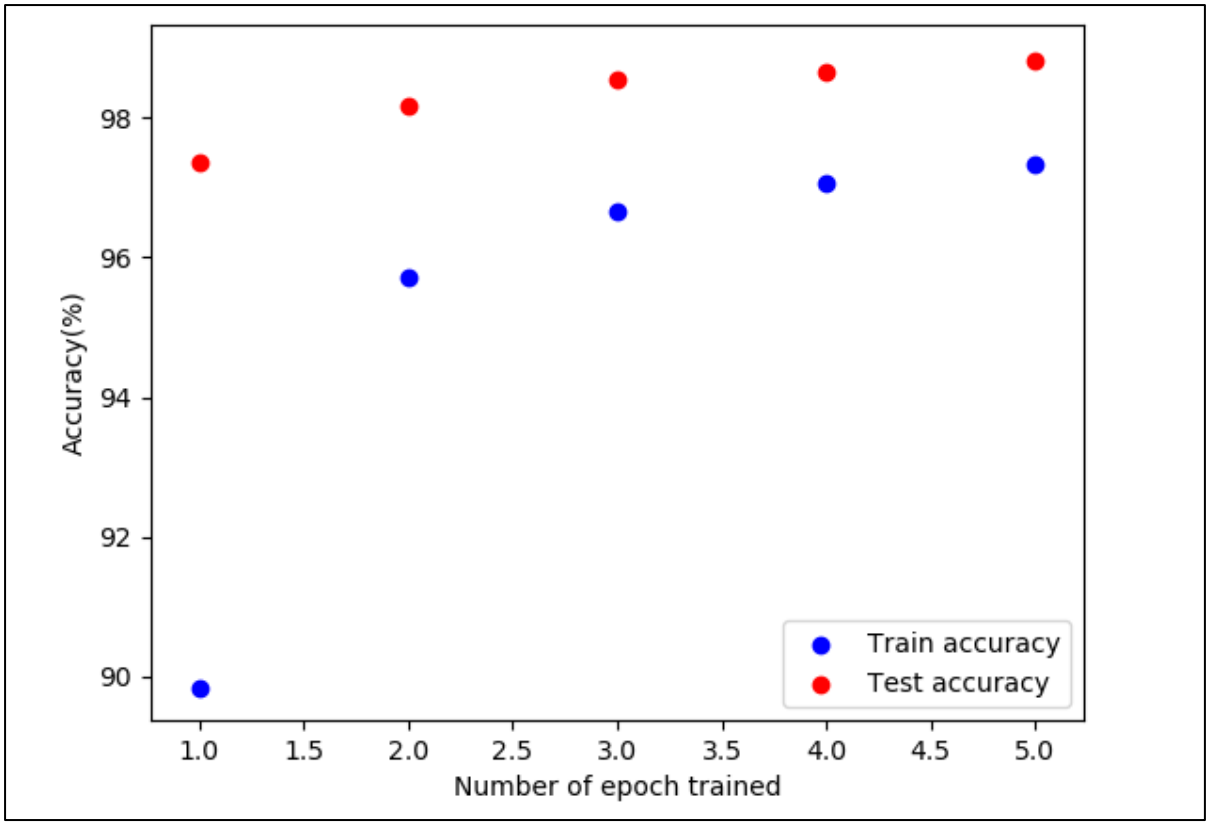

Fig. 3. Test and training accuracy of MNIST data

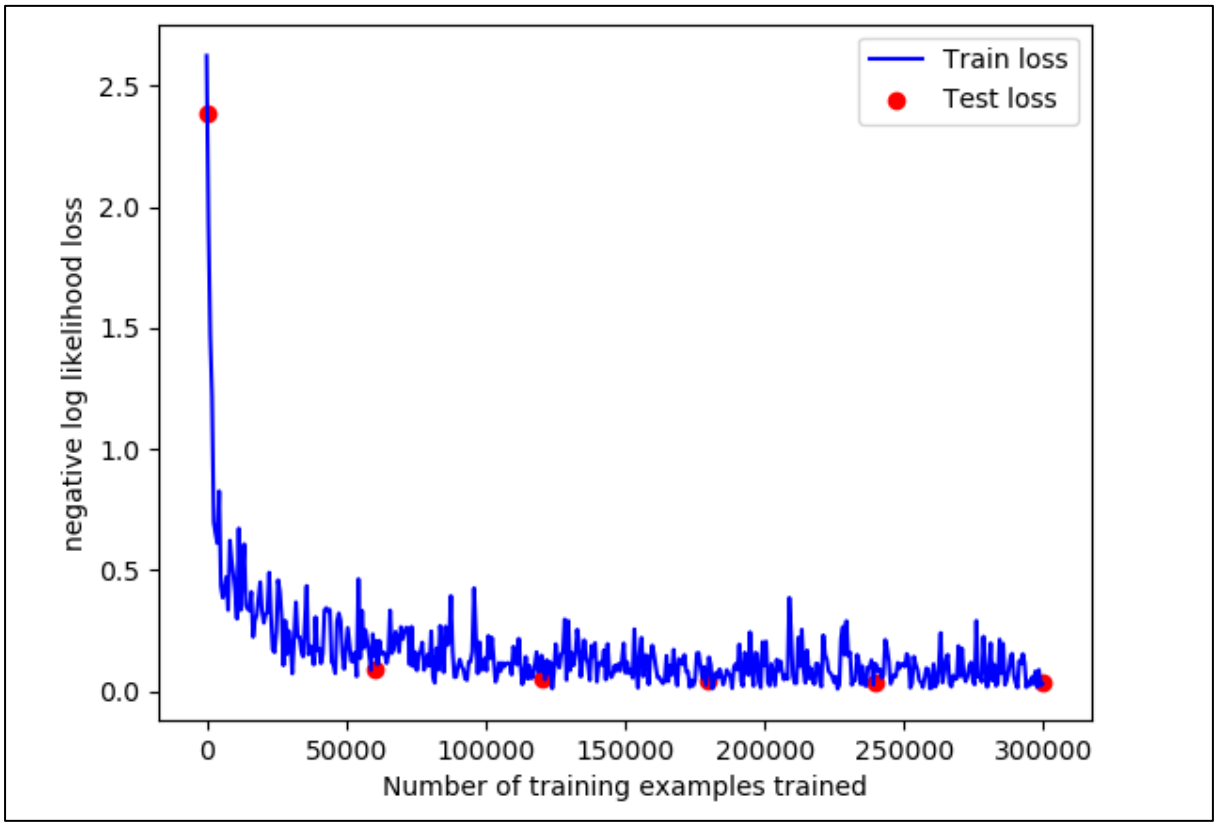

Fig. 4. Test and training loss of MNIST data

2.4. Test Model on MNIST Data

After training the model, it is saved so that It can be used for testing without training again. For testing, predictions for the first 10 test examples are printed and the model is classified all correctly. Predictions for the first 9 examples are plotted below:

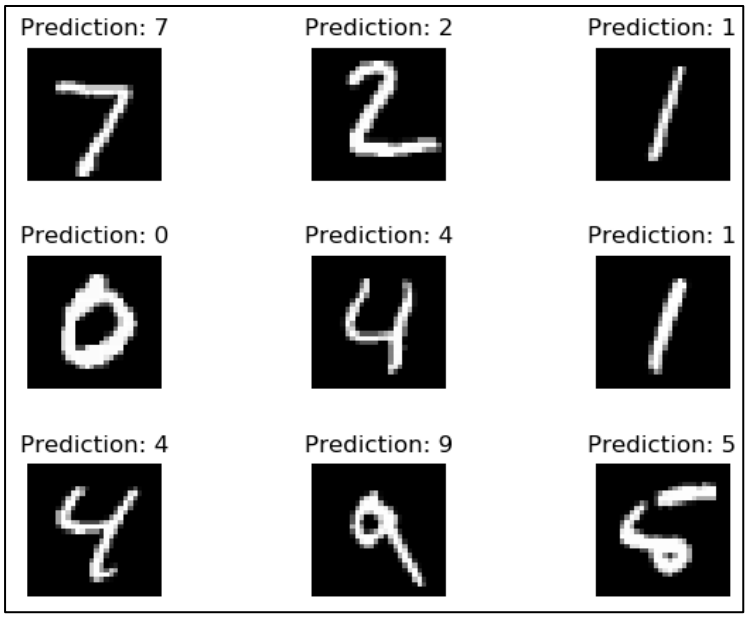

Fig. 5. Prediction of MNIST test data

2.5. Test Model on new digits

On testing 10 examples of MNIST data, it is found that all the examples are classified correctly. But to check the performance of the network for new digits, it is tested with my hand-written digits, and below is the plot of prediction on new digits. The network can classify new digits with 100% accuracy.

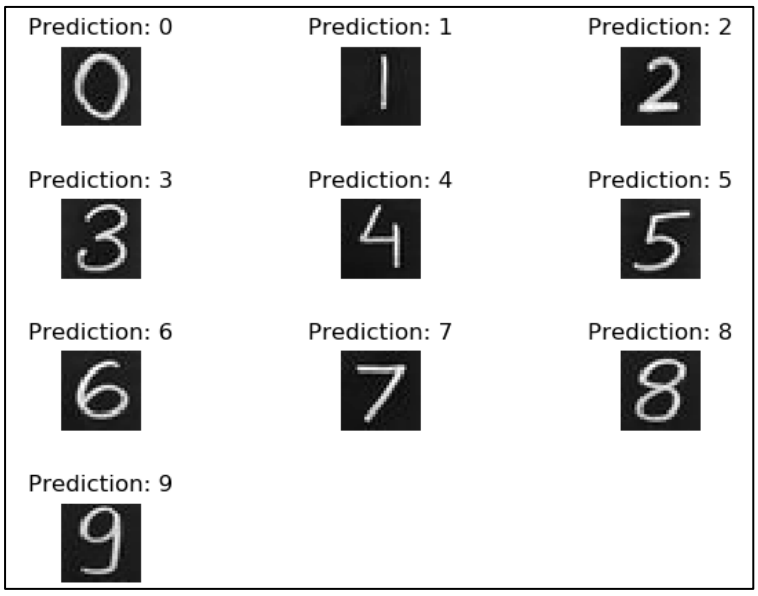

Fig. 6. Prediction of new digits

3. ANALYZING NETWORK

3.1. Analyze first-layer filters

It is very important to analyze the filters in the network that how it is processing the input data which helps to understand the network better. To do so, filters in the first layer of convolution are plotted. The first saved network is loaded and the weights corresponding to the first layer are extracted and visualized using the pyplot.

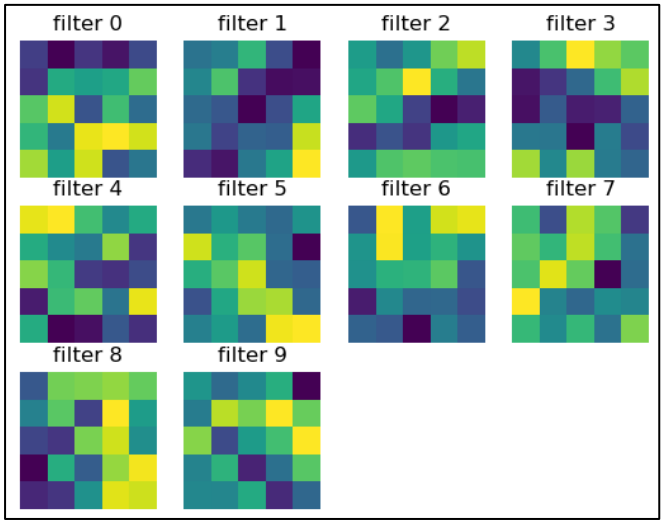

Fig. 7. Plot of filters in the first convolutional layer

3.2. Visualize the effect of filters

Once weights corresponding to each filter are extracted, they are applied to the input image, and the filtered image is shown in fig. 8. It can be seen that different filter extracts different information from the image, the filters in the first column mostly detected the horizontal and inclined edges, and filters in 2nd column detected vertical and inclined edges. It makes sense as detecting edges is critical in detecting any shape.

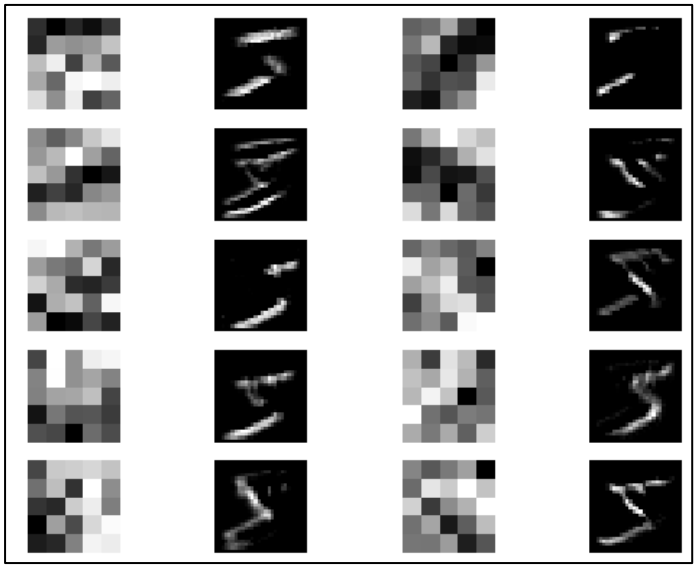

Fig. 8. Plots of the filtered image after 1st convolution layer

4. Transfer Learning on Greek Letters

Re-use of a Pre-Trained network is a common practice known as Transfer Learning and in this task, transfer learning is implemented by re-using the trained MNIST digit recognition network to recognize Greek letters. Instead of recognizing only 3 Greek letters, I have extended it to recognize 6 Greek letters namely Alpha, Beta, Delta, Gamma, Lambda, and Mu. Below is the modified network architecture in which the last dense layer is replaced with 6 nodes.

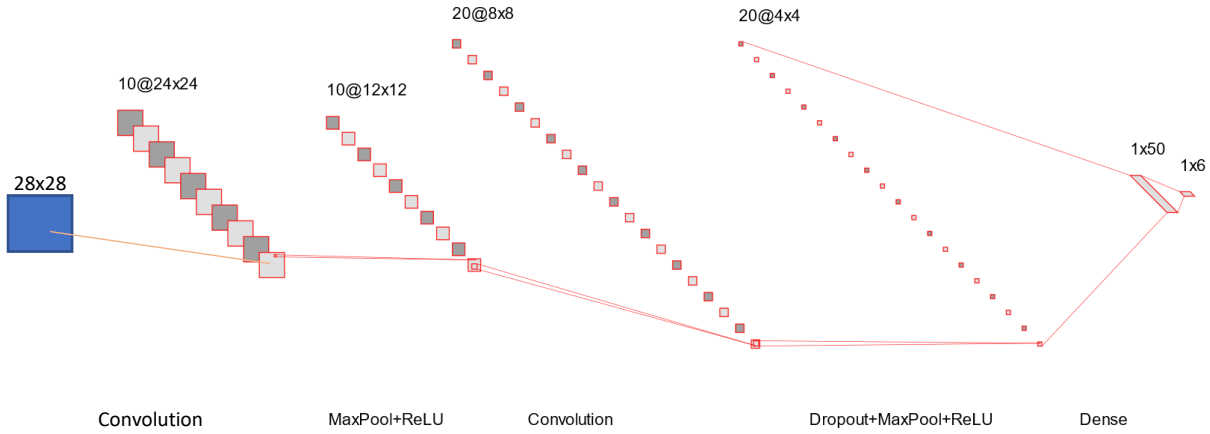

Fig. 9. Modified architecture for Transfer learning

Upon modifying the architecture, the network is trained using 54 examples (9 examples of each letter) with the following hyperparameters.

Batch Size = 3, Learning rate = 0.01, Momentum = 0.7, Epoch = 200

As can be seen from the training loss plot, the loss decreases rapidly to 50 epochs but after that, it decreases very slowly.

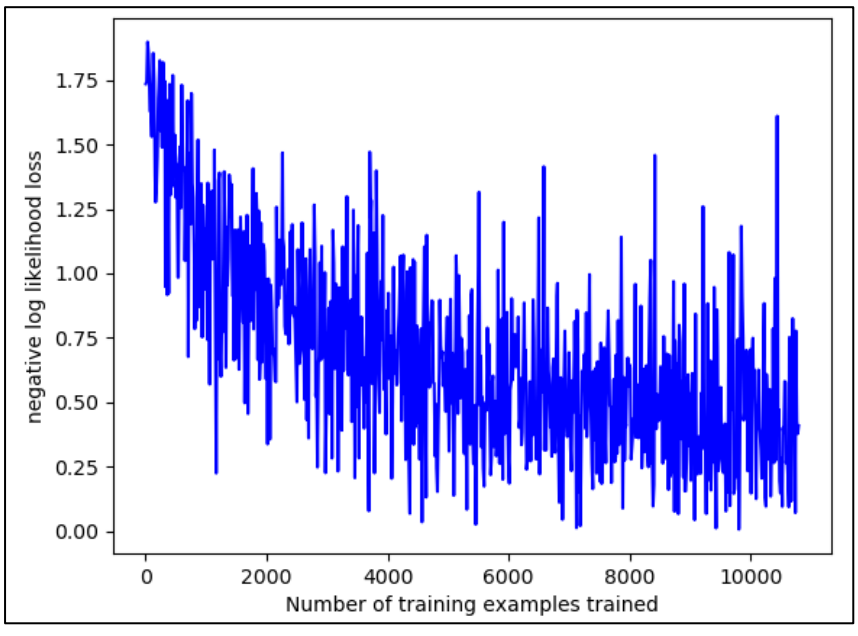

Fig. 10. Training loss plot for the Greek letter

Below is the plot of training error for Greek letters.

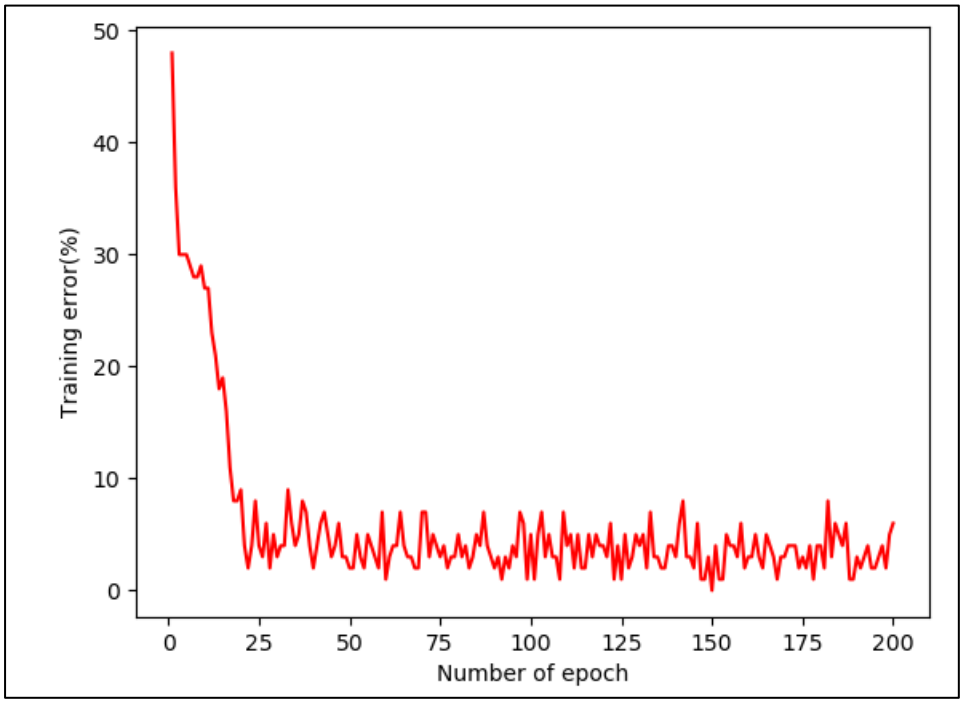

Fig. 11. Training error plot for the Greek letter

On testing with my handwritten letter, the maximum accuracy achieved is 68%. Since the training examples are very less, the network is not able to classify all the letters correctly. Even after training with 200 epochs, the accuracy does not improve much. Below is the prediction of the network. The network can classify 7 correctly out of 9 examples.

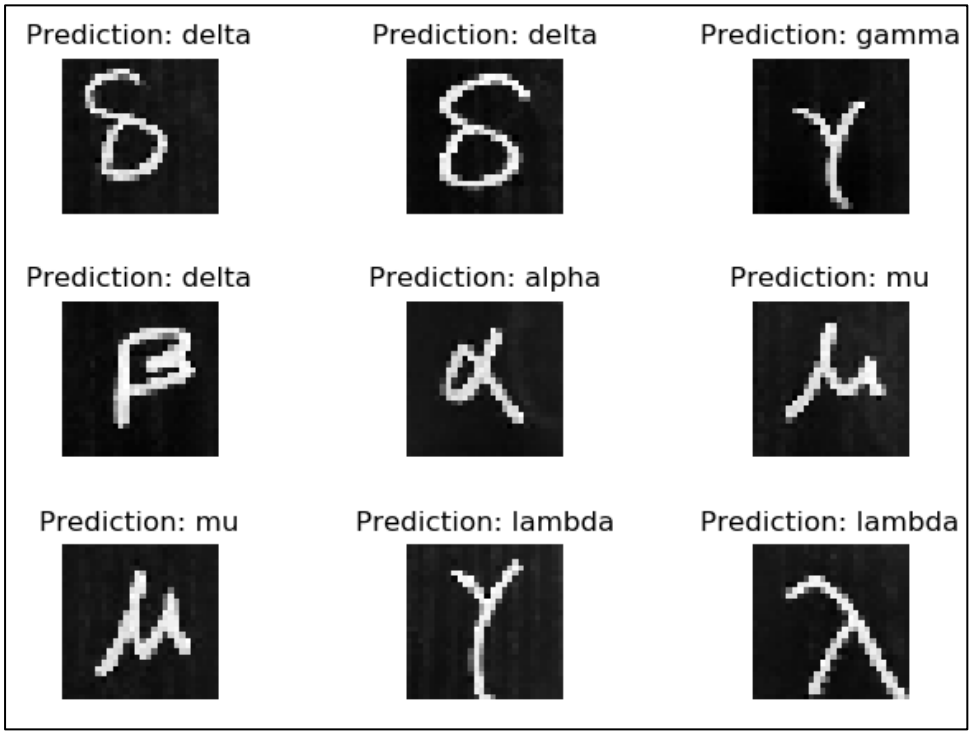

Fig. 12. Prediction of Greek Letter dataset

5. Hyperparameters tuning

5.1. Developing a plan for choosing hyperparameters

The network consists of many hyperparameters which can be tuned to achieve higher accuracy. But tuning every hyperparameter is not feasible practically. So, a set of hyperparameters is chosen which can contribute to the improvement of accuracy. As an extension instead of tuning 3 dimensions, I have decided to tune 7 dimensions. The following hyperparameters are chosen to be tuned with respective values:

Learning rate = [0.005,0.01,0.1]

Batch size = [32,64,128]

Momentum = [0.5,0.7,0.9]

Filter channels in 2nd convolutional layer = [32,64,96]

Filter size = [3,5]

Dropout rate = [0.3,0.4,0.5]

Epoch = [3,5,8,10]

Though I have selected only 7 hyperparameters with less range of values, even then the total combinations are 1944 which is also not possible to evaluate each.

So, Random search is used to evaluate different combinations of these hyperparameters, and a total of 100 network variations are evaluated.

5.2. Predicting result

Since each hyperparameter affects the network differently, the below hypothesis explains the effect of individual hyperparameters on the network.

- Learning Rate: A low learning rate such as 0.005 provides stable convergence but the model can stick in local minima. A high learning rate such as 0.1 can provide faster convergence but can also overshoot and may diverge.
- Batch Size: Larger batch size (128) can lead to faster training time but may overfit the data. Smaller batch sizes (32) can generalize the data more but take a large training time. Generally, for larger batch sizes, the learning rate should be kept high to achieve convergence.
- Momentum: High momentum leads to faster convergence and less generalization with robustness to the high value of learning rate and batch size. Low momentum will provide slower convergence but a better generalization.
- Filter channels: Increasing the number of filter channels help the network in learning more complex features and thus improved performance but very high channels can lead to overfitting.

- Filter Size: Small filter size (3*3) helps in preserving the spatial resolution and provides the ability to detect fine patterns while a larger filter size (5*5) provides a large receptive field and allows the network to process complex features.
- Dropout Rate: High dropout rate (0.5) can provide better regularization of data and prevent overfitting but can remove critical features for detecting the data correctly. A low dropout rate (0.3) can be a conservative value that provides less generalization but preserve most of the information.
- Epoch: Large epoch (10) helps in improving the accuracy of the model and thus allows the model to detect features with high probability.

5.3. Executing the plan and Evaluating the result

On executing the plan and carrying out the 100 iterations, below is the top 3 and bottom 3 results:

Top 3 accuracies with Corresponding Hyperparameters:

Accuracy	Learning Rate	Batch Size	Momentum	Filter Channel	Filter Size	Dropout Rate	Epoch
91.44%	0.1	128	0.7	96	3	0.3	10
91.25%	0.1	64	0.5	96	3	0.3	10
91.08%	0.005	32	0.9	64	5	0.3	10

Bottom 3 accuracies with Corresponding Hyperparameters:

Accuracy	Learning Rate	Batch Size	Momentum	Filter Channel	Filter Size	Dropout Rate	Epoch
10%	0.1	32	0.9	32	5	0.5	3
10%	0.1	32	0.9	64	3	0.5	5
43.83%	0.1	32	0.9	96	3	0.5	3

On analyzing the results and comparing them with my hypothesis, the following are the observations:

- A high learning rate with a high batch size gives higher accuracy and a high learning rate with a smaller batch size reduces the accuracy which matches the hypothesis.
- Large Epoch improves the accuracy which is the same as predicted.
- A high filter channel provided higher accuracy.
- The prediction of Filter size and momentum does not exactly match with the result as all values of filter size and momentum have given a good accuracy.
- A low dropout rate provided higher accuracy which also matches with a prediction as it can generalize the data more.

6. EXTENSION

For extension, I have extended the Greek letter dataset and classified 6 letters instead of 3.

Also, in experimenting with the design, I experimented with 7 dimensions instead of 3.

7. LEARNING

This project has provided me with knowledge of Deep Neural Networks and the formulation of architecture. I have learned about the different components of a model and the effects of several hyperparameters on the accuracy of the model. I have learned to visualize the filters and their effect on the image which helped me to understand the inner working of CNN. In Transfer learning, I learned to deploy the pre-trained network to classify the similar type of data.

ACKNOWLEDGEMENT

1. https://pytorch.org/tutorials/beginner/basics/intro.html
2. https://nextjournal.com/gkoehler/pytorch-mnist
3. https://towardsdatascience.com/handwritten-digit-mnist-pytorch-977b5338e627
4. http://alexlenail.me/NN-SVG/LeNet.html
5. https://www.analyticsvidhya.com/blog/2021/05/transfer-learning-using-mnist/