RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

INTRODUCTION TO DEEP LEARNING PROJECT REPORT

Performance Evaluation of Dropout and Batch Normalization Layers in CNN

Swapnil Shashikant Kamate (sk2181)

May 14, 2021

Table of Contents

Abstract **-** This report will discuss the effects of batch normalization and dropout layers on performance of the Convolutional Neural Networks. The architecture/model used in the convolutional neural network for this experiment is Lenet-5 model while the dataset used is MNIST dataset. We will also discuss fundamental concepts of CNN and Lenet-5 architecture, experimental results and analysis along with conclusion.

Keywords **-** Convolutional Neural Network; Lenet-5; MNIST; Batch Normalization, Dropout

1.Problem Statement

Evaluate the performance of dropout on fully connected layers and batch normalization on convolutional layers. The model is LeNet-5 on MNIST dataset. Among four options:

1. FC with dropout, CONV with BN
2. FC with dropout, CONV without BN
3. FC without dropout, CONV with BN
4. FC without dropout, CONV without dropout

Identify which one has the best performance?

2. Introduction

We will discuss some fundamentals of topics related to this experiment being performed.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification, facial recognition, self-driving cars, or object detection. CNNs are the foundation of modern state-of-the art deep learning-based computer vision. These networks are built upon 3 main ideas: local receptive fields, shared weights, and spatial subsampling. Local receptive fields with shared weights are the essence of the convolutional layer which is followed by some form of pooling which results in translation invariant features. CNNs are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. [1]

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product. The activation function is commonly a RELU layer and is subsequently followed by additional convolutions such as pooling layers, fully connected layers, normalization and dropout layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

Technically convolutional layers mean sliding dot product or cross-correlation that has significance for the indices in the matrix, in that it affects how weight is determined at a specific index point.

Architecture of Lenet-5

The first successful applications of Convolutional Networks were developed by Yann LeCun in the 1990's is the LeNet architecture that was used to read zip codes, digits, etc. LeNet-5 was also used on a large scale to automatically classify hand-written digits on bank cheques in the United States. LeNet made hand engineering features redundant, because the network learns the best internal representation from raw images automatically.

A *simple* Lenet-5 model only has 7 layers viz. 3 convolutional layers (C1, C3 and C5), 2 sub-sampling (pooling) layers (S2 and S4), and 1 fully connected layer (F6), that are followed by the output layer. Convolutional layers use 5x5 convolutions with stride 1. Sub-sampling layers are 2x2 average pooling layers. Tanh sigmoid activations i.e. RELU layers are used throughout the network.

First, individual convolutional kernels in the layer C3 do not use all of the features produced by the layer S2 so as to make the network less computationally demanding. Also it makes convolutional kernels learn different patterns so that if different kernels receive different inputs, they will learn different patterns.

Second, the output layer uses 10 Euclidean Radial Basis Function neurons that compute L2 distance between the input vector of dimension 84 and manually predefined weights vectors of the same dimension. The number 84 comes from the fact that essentially the weights represent a 7x12 binary mask, one for each digit. This forces the network to transform input image into an internal representation that will make outputs of layer F6 as close as possible to hand-coded weights of the 10 neurons of the output layer.

Batch Normalization

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization and makes it notoriously hard to train models with saturating nonlinearities referred to as internal covariate shift and address the problem by normalizing layer inputs.

Batch Normalization - a data pre-processing technique allows us to use much higher learning rates. It reduces strong dependency on initialization. It also acts as a form of regularize and in some cases eliminates the need for Dropout.

In a neural network, batch normalization is achieved through a normalization step that fixes the means and variances of each layer's inputs. Ideally, the normalization would be

conducted over the entire training set, but to use this step jointly with stochastic optimization methods, it is impractical to use the global information. Thus, normalization is restrained to each mini batch in the training process.

Dropout

A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. Regularization reduces overfitting by adding a penalty to the loss function. By adding this penalty, the model is trained such that it does not learn an interdependent set of features weights. Those of you who know Logistic Regression might be familiar with L1 (Laplacian) and L2 (Gaussian) penalties. Dropout is an approach to regularization in neural networks which helps reduce interdependent learning amongst the neurons.

Dropout refers to ignoring units (i.e. neurons) during the training phase of a certain set of neurons which is chosen at random. By "ignoring", I mean these units are not considered during a particular forward or backward pass. Dropout is applied at training phases and not in testing phases. In each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. In the testing stage, we use all activations, but reduce them by a factor p (to account for the missing activations during training). Dropout rate typically is 0.5.

MNIST Dataset

The MNIST database (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems. The database is also widely used for training and testing in the field of machine learning. The MNIST database contains 60,000 training images and 10,000 testing images all in Black and White colors.

3. Lenet-5 Models

In this section we will discuss the general configurations and architectures of different variations of Lenet-5 models mentioned in problem statement for experiments as listed below:

1. Lenet-5 Model with Dropout in Fully Connected Layer and Batch Normalization in Convolutional Layer
2. Lenet-5 Model with only Dropout in Fully Connected Layer
3. Lenet-5 Model with only Batch Normalization in Convolutional Layer
4. Simple/Standard Lenet-5 Model with NO Batch Normalization or Dropout layers

General Configuration

I have used the NN Sequential Module of Pytorch for training and testing the model.

Also, I used Cross Entropy Loss Layer in model training as Pytorch Cross Entropy Layer to calculates the Softmax as well as the loss simultaneously. Also, since this problem was multiclass classification problem, I had used this layer while training for calculating loss. The number of epochs is 25 as after 8-10 epochs accuracy goes to 99%. So, there was no point in going up to 50 epochs for training the model. The batch size considered here for training images is 128 while for testing images is 100. The training dataset is randomly shuffled while the testing dataset is not shuffled. Learning rate is 0.05, momentum is 0.9 and weight decay is 5e-4. Also, I have used SGD optimizer for optimizing weights after backward propagation

Dropout in FC and Batch Normalization in CONV

As per first part of the problem statement, the architecture of Lenet-5 is: Starting layer is Convolution2d layer with input of 1 channel and output as 6 channels with Kernel filter size as 5,5 followed by Relu function **–** c1 and relu1. Now, I normalize the convolved data using batch normalization layer for 6 channels **-** bn1. Then I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 **–** s2. Further, then comes Convolution2d layer with input as 6 channels and outputs as 16 channels with Kernel filter size as 5,5 followed by Relu function **–** c3 and relu3. Again, I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 **–** s4. In the last convolution network, I have Convolution2d layer with input as 16 channels and outputs as 120 channels with Kernel filter size as 4,4 followed by Relu function **–** c5, relu5. Now very important task – I have to flatten the images i.e. output from convolution network so that they can be fed into fully connected layers. Thus, I have flattened the output of convolution network using torch.view() function in forward pass. Then, once data is flattened, I fed it to fully connected Linear layer with input as 120 and output as 84 followed by Relu function **–** f6 and relu6. Now, I have passed data through dropout layer with rate as 0.5 (widely used) - drop6. Further, I again pass the data to Linear Layer with input as 84 and final output as 10 **–** f7. This output is passed to final LogSoftmax layer **–** sig7

Dropout in FC

As per second part of the problem statement, the architecture of Lenet-5 is: Starting layer is Convolution2d layer with input of 1 channel and output as 6 channels with Kernel filter size as 5,5 followed by Relu function – c1 and relu1. Then I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s2. Further, then comes Convolution2d layer with input as 6 channels and outputs as 16 channels with Kernel filter size as 5,5 followed by Relu function – c3 and relu3. Again, I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s4. In the last convolution network, I have Convolution2d layer with input as 16 channels and outputs as 120 channels with Kernel filter size as 4,4 followed by Relu function **–** c5, relu5. Now

a very important task – I have to flatten the images i.e. output from convolution network so that they can be fed into fully connected layers. Thus, I have flattened the output of convolution network using torch.view() function in forward pass. Then, once data is flattened, I fed it to fully connected Linear layer with input as 120 and output as 84 followed by Relu function – f6 and relu6. Now, I have passed data through dropout layer with rate as 0.5 (widely used) - drop6. Further, I again pass the data to Linear Layer with input as 84 and final output as 10 – f7. This output is passed to final LogSoftmax layer – sig7

Batch Normalization in CONV

As per third part of the problem statement, the architecture of Lenet-5 is: Starting layer is Convolution2d layer with input of 1 channel and output as 6 channels with Kernel filter size as 5,5 followed by Relu function – c1 and relu1. Now, I normalize the convolved data using batch normalization layer for 6 channels - bn1. Then I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s2. Further, then comes Convolution2d layer with input as 6 channels and outputs as 16 channels with Kernel filter size as 5,5 followed by Relu function – c3 and relu3. Again, I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s4. In the last convolution network, I have Convolution2d layer with input as 16 channels and outputs as 120 channels with Kernel filter size as 4,4 followed by Relu function – c5, relu5. Now a very important task – I have to flatten the images i.e. output from convolution network so that they can be fed into fully connected layers. Thus, I have flattened the output of convolution network using torch.view() function in forward pass. Then, once data is flattened, I fed it to fully connected Linear layer with input as 120 and output as 84 followed by Relu function – f6 and relu6. Further, I again pass the data to Linear Layer with input as 84 and final output as 10 – f7. This output is passed to final LogSoftmax layer – sig7

The only difference compared to architecture discussed in section 2.1 is that only batch normalization is present in the CONV layer while dropout is removed from the FC layer.

No Dropout in FC and No Batch Normalization in CONV

As per fourth part of the problem statement, the architecture of Lenet-5 is: Starting layer is Convolution2d layer with input of 1 channel and output as 6 channels with Kernel filter size as 5,5 followed by Relu function – c1 and relu1. Then I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s2. Further, then comes Convolution2d layer with input as 6 channels and outputs as 16 channels with Kernel filter size as 5,5 followed by Relu function – c3 and relu3. Again, I perform sub-sampling using MaxPool2d with kernel size as 2,2 and stride of 2 – s4. In the last convolution network, I have Convolution2d layer with input as 16 channels and outputs as 120 channels with Kernel filter size as 4,4 followed by Relu function – c5, relu5. Now a very important task – I have to flatten the images i.e. output from convolution network so that they can be fed into fully connected layers. Thus, I have flattened the output of convolution network using torch.view() function in forward pass. Then, once data is flattened, I fed it to fully connected Linear layer with input as 120 and output as 84 followed by Relu function – f6 and

relu6. Further, I again pass the data to Linear Layer with input as 84 and final output as 10 – f7. This output is passed to final LogSoftmax layer – sig7

This is the simple Lenet-5 model where NO dropout layer and NO batch normalization layers are present in FC and CONV layers.

This is the simple Lenet-5 model where **NO** dropout layer and **NO** batch normalization layers are present in FC and CONV layers.

4. Experimental Result

Below is the table which represents the total time taken for different Lenet-5 Models:

Lenet Architecture	**Total Time**
	Seconds
With DO and BN	558.416530
With DO	404.555684
With BN	444.486130
No DO and BN (Simple Lenet)	407.227121

Results Discussion

Firstly, the performance in terms of accuracy for all the variations of Lenet-5 models is the same i.e., **99%**.

Dropout in FC and Batch Normalization in CONV

- Below is the output of the Lenet-5 model with dropout in fully connected layer and batch normalization in convolutional layer. The figure shows the average test loss for first and last epoch, training time taken, total images tested and model accuracy.
- As seen in the Figure, it took around **558.416530 seconds** for training the model with 60000 training and 10000 testing MNIST dataset images in 25 epochs. The test accuracy of the model as shown in screenshots below is **99%**. The accuracy after the first epoch is the minimum accuracy while model training which is **98%**

lenet_mnist_DO_BN.py

```
Extracting ./test\MNIST\raw\t10k-labels-idx1-ubyte.gz to ./test\MNIST\raw

Processing...
Done!
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.324278
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.164549
Train Epoch: 1 [2560/60000 (4%)]        Loss: 1.318981
Train Epoch: 1 [3840/60000 (6%)]        Loss: 0.778869
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.478179
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.281966
Train Epoch: 1 [7680/60000 (13%)]       Loss: 0.289395
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.196837
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.164424
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.169962
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.092814
Train Epoch: 1 [14080/60000 (23%)]      Loss: 0.089485
Train Epoch: 1 [15360/60000 (26%)]      Loss: 0.135696
Train Epoch: 1 [16640/60000 (28%)]      Loss: 0.268402
Train Epoch: 1 [17920/60000 (30%)]      Loss: 0.123606
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.202491
Train Epoch: 1 [20480/60000 (34%)]      Loss: 0.085648
Train Epoch: 1 [21760/60000 (36%)]      Loss: 0.279535
Train Epoch: 1 [23040/60000 (38%)]      Loss: 0.140939
Train Epoch: 1 [24320/60000 (41%)]      Loss: 0.140660
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.179136
Train Epoch: 1 [26880/60000 (45%)]      Loss: 0.044494
Train Epoch: 1 [28160/60000 (47%)]      Loss: 0.142534
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.141849
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.099916
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.064560
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.067758
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.105002
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.102125
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.131554
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.050322
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.120336
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.059327
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.041703
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.039434
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.094811
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.128982
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.045584
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.108354
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.129443
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.034728
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.035553
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.035457
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.092779
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.052664
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.036783
Train Epoch: 25 [3840/60000 (6%)]       Loss: 0.019296
Train Epoch: 25 [5120/60000 (9%)]       Loss: 0.019363
Train Epoch: 25 [6400/60000 (11%)]      Loss: 0.008514
Train Epoch: 25 [7680/60000 (13%)]      Loss: 0.010361
Train Epoch: 25 [8960/60000 (15%)]      Loss: 0.038668
Train Epoch: 25 [10240/60000 (17%)]     Loss: 0.017809
Train Epoch: 25 [11520/60000 (19%)]     Loss: 0.010262
Train Epoch: 25 [12800/60000 (21%)]     Loss: 0.002471
Train Epoch: 25 [14080/60000 (23%)]     Loss: 0.016261
Train Epoch: 25 [15360/60000 (26%)]     Loss: 0.003769
Train Epoch: 25 [16640/60000 (28%)]     Loss: 0.007713
Train Epoch: 25 [17920/60000 (30%)]     Loss: 0.026593
Train Epoch: 25 [19200/60000 (32%)]     Loss: 0.010590
Train Epoch: 25 [20480/60000 (34%)]     Loss: 0.035446
Train Epoch: 25 [21760/60000 (36%)]     Loss: 0.010754
Train Epoch: 25 [23040/60000 (38%)]     Loss: 0.006429
Train Epoch: 25 [24320/60000 (41%)]     Loss: 0.007398
Train Epoch: 25 [25600/60000 (43%)]     Loss: 0.019680
Train Epoch: 25 [26880/60000 (45%)]     Loss: 0.014840
Train Epoch: 25 [28160/60000 (47%)]     Loss: 0.014682
Train Epoch: 25 [29440/60000 (49%)]     Loss: 0.061213
Train Epoch: 25 [30720/60000 (51%)]     Loss: 0.031298
Train Epoch: 25 [32000/60000 (53%)]     Loss: 0.005842
Train Epoch: 25 [33280/60000 (55%)]     Loss: 0.016596
Train Epoch: 25 [34560/60000 (58%)]     Loss: 0.016001
Train Epoch: 25 [35840/60000 (60%)]     Loss: 0.003768
Train Epoch: 25 [37120/60000 (62%)]     Loss: 0.064415
Train Epoch: 25 [38400/60000 (64%)]     Loss: 0.013877
Train Epoch: 25 [39680/60000 (66%)]     Loss: 0.030422
Train Epoch: 25 [40960/60000 (68%)]     Loss: 0.003318
Train Epoch: 25 [42240/60000 (70%)]     Loss: 0.027766
Train Epoch: 25 [43520/60000 (72%)]     Loss: 0.002573
Train Epoch: 25 [44800/60000 (75%)]     Loss: 0.005494
Train Epoch: 25 [46080/60000 (77%)]     Loss: 0.028925
Train Epoch: 25 [47360/60000 (79%)]     Loss: 0.007190
Train Epoch: 25 [48640/60000 (81%)]     Loss: 0.007700
Train Epoch: 25 [49920/60000 (83%)]     Loss: 0.026652
Train Epoch: 25 [51200/60000 (85%)]     Loss: 0.012153
Train Epoch: 25 [52480/60000 (87%)]     Loss: 0.004504
Train Epoch: 25 [53760/60000 (90%)]     Loss: 0.048481
Train Epoch: 25 [55040/60000 (92%)]     Loss: 0.029908
Train Epoch: 25 [56320/60000 (94%)]     Loss: 0.008246
Train Epoch: 25 [57600/60000 (96%)]     Loss: 0.012449
Train Epoch: 25 [58880/60000 (98%)]     Loss: 0.003771

Test set: Average loss: 0.0293, Accuracy: 9907/10000 (99%)

Traning and Testing total excution time is: 558.4165308475494 seconds

C:\Users\Swapnil\Desktop\Deep learning project>
```

Dropout in FC

- Below is the output of the Lenet-5 model with only dropout in fully connected layer and NO batch normalization in convolutional layer. The figure shows the average test loss for first and last epoch, training time taken, total images tested and model accuracy. As seen in the Figure, it took around 404.555684 **seconds** for training the model with 60000 training and 10000 testing MNIST dataset images in 25 epochs. The test accuracy of the model as shown in screenshots below is **99%**. The accuracy after the first epoch is the minimum accuracy while model training which is **97%**

  lenet_mnist_DO.py

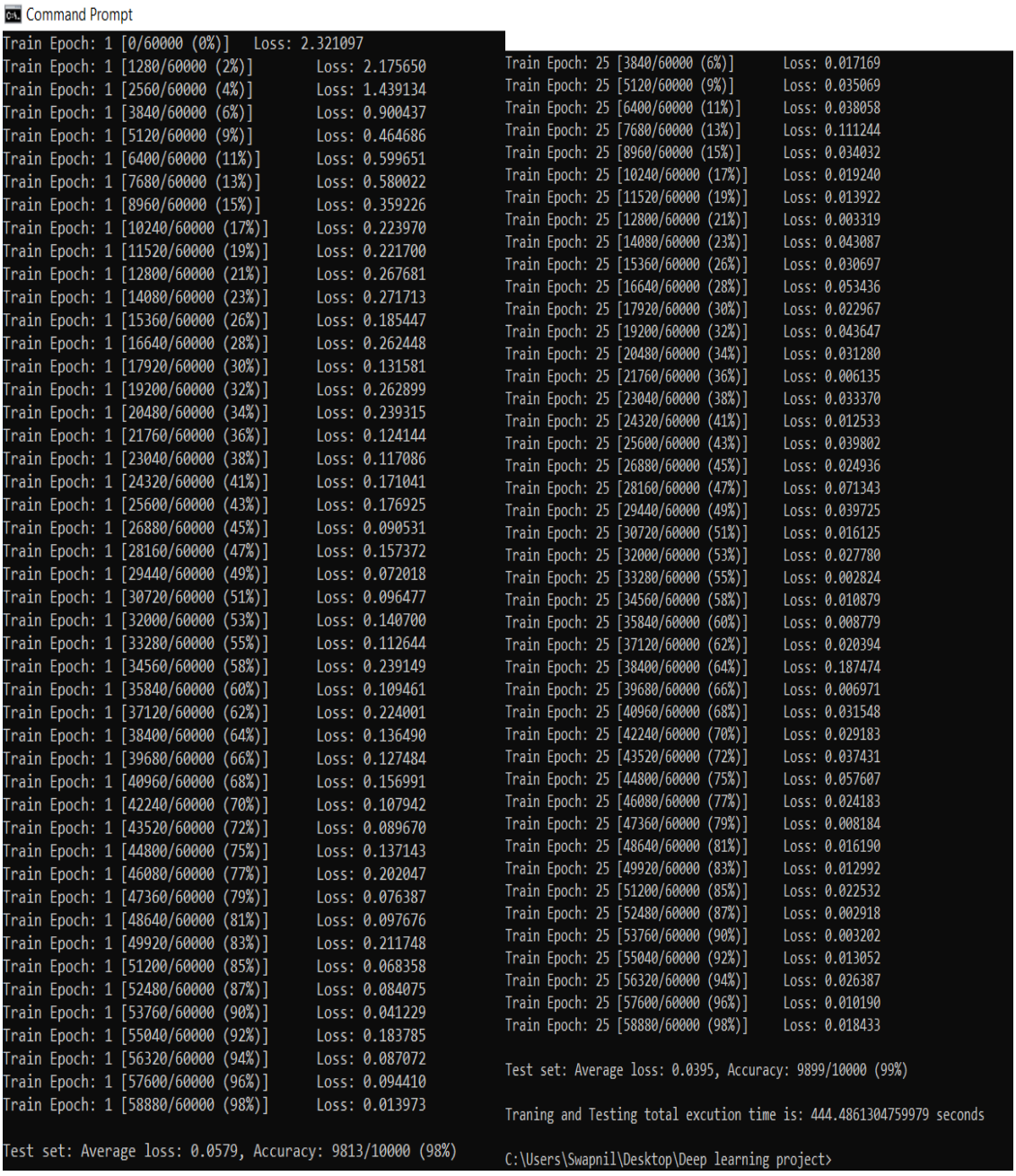
Command Prompt

```
C:\Users\Swapnil\Desktop\Deep learning project>python lenet_mnist_DO.py
==> Preparing data..
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.319211
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.303621
Train Epoch: 1 [2560/60000 (4%)]        Loss: 2.289746
Train Epoch: 1 [3840/60000 (6%)]        Loss: 2.268143
Train Epoch: 1 [5120/60000 (9%)]        Loss: 2.104385
Train Epoch: 1 [6400/60000 (11%)]       Loss: 1.329896
Train Epoch: 1 [7680/60000 (13%)]       Loss: 1.011046
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.663381
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.442650
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.365886
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.494768
Train Epoch: 1 [14080/60000 (23%)]      Loss: 0.304094
Train Epoch: 1 [15360/60000 (26%)]      Loss: 0.269762
Train Epoch: 1 [16640/60000 (28%)]      Loss: 0.599358
Train Epoch: 1 [17920/60000 (30%)]      Loss: 0.221742
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.408206
Train Epoch: 1 [20480/60000 (34%)]      Loss: 0.271966
Train Epoch: 1 [21760/60000 (36%)]      Loss: 0.267249
Train Epoch: 1 [23040/60000 (38%)]      Loss: 0.258748
Train Epoch: 1 [24320/60000 (41%)]      Loss: 0.276937
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.287201
Train Epoch: 1 [26880/60000 (45%)]      Loss: 0.217653
Train Epoch: 1 [28160/60000 (47%)]      Loss: 0.314181
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.185965
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.143744
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.134035
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.163740
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.192608
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.110559
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.284515
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.145412
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.164847
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.158056
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.112738
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.167141
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.191698
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.236986
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.074818
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.149412
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.155139
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.112390
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.080829
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.084081
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.160786
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.063281
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.072356
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.074188
Train Epoch: 25 [3840/60000 (6%)]       Loss: 0.019236
Train Epoch: 25 [5120/60000 (9%)]       Loss: 0.051453
Train Epoch: 25 [6400/60000 (11%)]      Loss: 0.022628
Train Epoch: 25 [7680/60000 (13%)]      Loss: 0.035540
Train Epoch: 25 [8960/60000 (15%)]      Loss: 0.024681
Train Epoch: 25 [10240/60000 (17%)]     Loss: 0.005798
Train Epoch: 25 [11520/60000 (19%)]     Loss: 0.063105
Train Epoch: 25 [12800/60000 (21%)]     Loss: 0.025938
Train Epoch: 25 [14080/60000 (23%)]     Loss: 0.057344
Train Epoch: 25 [15360/60000 (26%)]     Loss: 0.020427
Train Epoch: 25 [16640/60000 (28%)]     Loss: 0.060198
Train Epoch: 25 [17920/60000 (30%)]     Loss: 0.011882
Train Epoch: 25 [19200/60000 (32%)]     Loss: 0.053859
Train Epoch: 25 [20480/60000 (34%)]     Loss: 0.018983
Train Epoch: 25 [21760/60000 (36%)]     Loss: 0.003597
Train Epoch: 25 [23040/60000 (38%)]     Loss: 0.028050
Train Epoch: 25 [24320/60000 (41%)]     Loss: 0.010619
Train Epoch: 25 [25600/60000 (43%)]     Loss: 0.036520
Train Epoch: 25 [26880/60000 (45%)]     Loss: 0.009366
Train Epoch: 25 [28160/60000 (47%)]     Loss: 0.050910
Train Epoch: 25 [29440/60000 (49%)]     Loss: 0.027335
Train Epoch: 25 [30720/60000 (51%)]     Loss: 0.018068
Train Epoch: 25 [32000/60000 (53%)]     Loss: 0.019919
Train Epoch: 25 [33280/60000 (55%)]     Loss: 0.004190
Train Epoch: 25 [34560/60000 (58%)]     Loss: 0.014143
Train Epoch: 25 [35840/60000 (60%)]     Loss: 0.009681
Train Epoch: 25 [37120/60000 (62%)]     Loss: 0.015294
Train Epoch: 25 [38400/60000 (64%)]     Loss: 0.080325
Train Epoch: 25 [39680/60000 (66%)]     Loss: 0.014368
Train Epoch: 25 [40960/60000 (68%)]     Loss: 0.009645
Train Epoch: 25 [42240/60000 (70%)]     Loss: 0.042381
Train Epoch: 25 [43520/60000 (72%)]     Loss: 0.009515
Train Epoch: 25 [44800/60000 (75%)]     Loss: 0.045635
Train Epoch: 25 [46080/60000 (77%)]     Loss: 0.004092
Train Epoch: 25 [47360/60000 (79%)]     Loss: 0.012198
Train Epoch: 25 [48640/60000 (81%)]     Loss: 0.021738
Train Epoch: 25 [49920/60000 (83%)]     Loss: 0.011526
Train Epoch: 25 [51200/60000 (85%)]     Loss: 0.046517
Train Epoch: 25 [52480/60000 (87%)]     Loss: 0.002334
Train Epoch: 25 [53760/60000 (90%)]     Loss: 0.011498
Train Epoch: 25 [55040/60000 (92%)]     Loss: 0.027203
Train Epoch: 25 [56320/60000 (94%)]     Loss: 0.014415
Train Epoch: 25 [57600/60000 (96%)]     Loss: 0.020355
Train Epoch: 25 [58880/60000 (98%)]     Loss: 0.021183

Test set: Average loss: 0.0355, Accuracy: 9903/10000 (99%)

Traning and Testing total excution time is: 404.55568408966064 seconds

C:\Users\Swapnil\Desktop\Deep learning project>
```

Batch Normalization in CONV

- Below is the output of the Lenet-5 model with **NO** dropout in fully connected layer and only batch normalization in convolutional layer. The figure shows the average test loss for first and last epoch, training time taken, total images tested and model accuracy. As seen in the figure it took around 444.486130 **seconds** for training the model with 60000 training and 10000 testing MNIST dataset images in 25 epochs. The test accuracy of the model as shown in screenshots below is **99%**. The accuracy after the first epoch is the minimum accuracy while model training which is **98%**

lenet_mnist_BN.py



No Dropout in FC and No Batch Normalization in CONV

- Below is the output of the simple Lenet-5 model with **NO** dropout in fully connected layer and **NO** batch normalization in convolutional layer. The figure shows the average test loss for first and last epoch, training time taken, total images tested and model accuracy. As seen in the Figure it took around 407.227121 **seconds** for training the model with 60000 training and 10000 testing MNIST dataset images in 25 epochs. The test accuracy of the model as shown in screenshots below is **99%**. The accuracy after the first epoch is the minimum accuracy while model training which is **97%**

  lenet_mnist_No_DO_BN.py

Command Prompt

```
C:\Users\Swapnil\Desktop\Deep learning project>python lenet_mnist_No_DO_BN.py
==> Preparing data..
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.318271
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.304561
Train Epoch: 1 [2560/60000 (4%)]        Loss: 2.286617
Train Epoch: 1 [3840/60000 (6%)]        Loss: 2.264895
Train Epoch: 1 [5120/60000 (9%)]        Loss: 2.025578
Train Epoch: 1 [6400/60000 (11%)]       Loss: 1.598038
Train Epoch: 1 [7680/60000 (13%)]       Loss: 0.983933
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.836616
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.359403
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.363787
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.327247
Train Epoch: 1 [14080/60000 (23%)]      Loss: 0.235913
Train Epoch: 1 [15360/60000 (26%)]      Loss: 0.225580
Train Epoch: 1 [16640/60000 (28%)]      Loss: 0.402391
Train Epoch: 1 [17920/60000 (30%)]      Loss: 0.152297
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.359527
Train Epoch: 1 [20480/60000 (34%)]      Loss: 0.194780
Train Epoch: 1 [21760/60000 (36%)]      Loss: 0.189668
Train Epoch: 1 [23040/60000 (38%)]      Loss: 0.246455
Train Epoch: 1 [24320/60000 (41%)]      Loss: 0.193409
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.245804
Train Epoch: 1 [26880/60000 (45%)]      Loss: 0.108875
Train Epoch: 1 [28160/60000 (47%)]      Loss: 0.238848
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.116739
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.108330
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.123298
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.108128
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.098889
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.120987
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.211614
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.140945
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.097594
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.140885
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.141541
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.106014
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.073318
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.123109
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.057151
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.104294
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.081617
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.045292
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.093753
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.048462
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.126347
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.057656
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.040542
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.063238
```

Command Prompt

```
Train Epoch: 25 [3840/60000 (6%)]       Loss: 0.005294
Train Epoch: 25 [5120/60000 (9%)]       Loss: 0.009308
Train Epoch: 25 [6400/60000 (11%)]      Loss: 0.008288
Train Epoch: 25 [7680/60000 (13%)]      Loss: 0.017238
Train Epoch: 25 [8960/60000 (15%)]      Loss: 0.009672
Train Epoch: 25 [10240/60000 (17%)]     Loss: 0.004013
Train Epoch: 25 [11520/60000 (19%)]     Loss: 0.006451
Train Epoch: 25 [12800/60000 (21%)]     Loss: 0.010100
Train Epoch: 25 [14080/60000 (23%)]     Loss: 0.029227
Train Epoch: 25 [15360/60000 (26%)]     Loss: 0.003467
Train Epoch: 25 [16640/60000 (28%)]     Loss: 0.034978
Train Epoch: 25 [17920/60000 (30%)]     Loss: 0.028433
Train Epoch: 25 [19200/60000 (32%)]     Loss: 0.033883
Train Epoch: 25 [20480/60000 (34%)]     Loss: 0.026913
Train Epoch: 25 [21760/60000 (36%)]     Loss: 0.009614
Train Epoch: 25 [23040/60000 (38%)]     Loss: 0.008082
Train Epoch: 25 [24320/60000 (41%)]     Loss: 0.017787
Train Epoch: 25 [25600/60000 (43%)]     Loss: 0.007120
Train Epoch: 25 [26880/60000 (45%)]     Loss: 0.003011
Train Epoch: 25 [28160/60000 (47%)]     Loss: 0.026792
Train Epoch: 25 [29440/60000 (49%)]     Loss: 0.039008
Train Epoch: 25 [30720/60000 (51%)]     Loss: 0.026403
Train Epoch: 25 [32000/60000 (53%)]     Loss: 0.042561
Train Epoch: 25 [33280/60000 (55%)]     Loss: 0.007437
Train Epoch: 25 [34560/60000 (58%)]     Loss: 0.014657
Train Epoch: 25 [35840/60000 (60%)]     Loss: 0.008013
Train Epoch: 25 [37120/60000 (62%)]     Loss: 0.066748
Train Epoch: 25 [38400/60000 (64%)]     Loss: 0.038199
Train Epoch: 25 [39680/60000 (66%)]     Loss: 0.033618
Train Epoch: 25 [40960/60000 (68%)]     Loss: 0.022350
Train Epoch: 25 [42240/60000 (70%)]     Loss: 0.015867
Train Epoch: 25 [43520/60000 (72%)]     Loss: 0.007732
Train Epoch: 25 [44800/60000 (75%)]     Loss: 0.009843
Train Epoch: 25 [46080/60000 (77%)]     Loss: 0.036903
Train Epoch: 25 [47360/60000 (79%)]     Loss: 0.010623
Train Epoch: 25 [48640/60000 (81%)]     Loss: 0.001925
Train Epoch: 25 [49920/60000 (83%)]     Loss: 0.016201
Train Epoch: 25 [51200/60000 (85%)]     Loss: 0.027004
Train Epoch: 25 [52480/60000 (87%)]     Loss: 0.008546
Train Epoch: 25 [53760/60000 (90%)]     Loss: 0.047914
Train Epoch: 25 [55040/60000 (92%)]     Loss: 0.019798
Train Epoch: 25 [56320/60000 (94%)]     Loss: 0.000613
Train Epoch: 25 [57600/60000 (96%)]     Loss: 0.004655
Train Epoch: 25 [58880/60000 (98%)]     Loss: 0.064847

Test set: Average loss: 0.0331, Accuracy: 9897/10000 (99%)

Traning and Testing total excution time is: 407.227121591568 seconds

C:\Users\Swapnil\Desktop\Deep learning project>
```

5. Performance Analysis

In the following sections we will analyze the behavior of Lenet-5 models and its different variants mentioned in earlier sections for deeper understanding of the project implemented.

High accuracy

- The reason for such high accuracy for any of the variants of the Lenet-5 model implemented is only because of the dataset used. We have used MNIST dataset for training and testing of different Lenet-5 models. Since, MNIST dataset has single channel black and white images, the model trains very well and hence the accuracy is high i.e., **99**%. This is not same as CIFAR-10 dataset images where the accuracy was between 50-65% for Lenet-5 based models which is quite low with respect to MNIST dataset.

Minimum Accuracy (Accuracy after First Epoch)

- The minimum accuracy (accuracy after the first epoch) was lowest for Lenet-5 models where only a dropout layer was present and where no dropout and batch normalization layer were present in the model. This is because in the case where only a dropout layer is present in the model (variant 2), the model initially does not train well because data isn't normalized, and also important features would have dropped in the fully connected layer. Also, in case where no dropout and no batch normalization layer is present in the model, the model initially doesn't train well because data isn't normalized and only the raw data is passed through different layers of the model (variant 4).Thus, due to above reasons, the minimum accuracy (accuracy after first epoch) is higher i.e. 98% for other two variants (1 and 3) i.e. models where only batch normalization is present and also where both batch normalization and dropout is present in the models as data is always normalized after first convolutional layer (**c1**).

6. Future Scope and Applications

The future scope of the project implemented could be very wide and more analysis can be done with respect to the performance for the different variants of the model specified. This can be done by changing different hyper parameters like learning rate, momentum, batch sizes for training and testing, changing placements of dropout layers from fully connected to convolutional layers, etc.

There are wide applications for using Lenet-5 based models in the field of deep learning ranging from reading zip codes, digits classification and recognition, image classification and many more.

7. Conclusion

In this report, we discussed the fundamentals like details of convolutional neural networks, Lenet-5 architecture, information on dropout, batch normalization and MNIST dataset. Later on, we discussed all the variants of the Lenet-5 models that were required to be used in this project and its respective architecture in detail. Further, we discussed the experimental results which involved total time taken and testing accuracy for each of the variants of Lenet-5 models. Lastly, we discussed a very important part i.e., performance analysis between all the variants of Lenet-5 models implemented in the project.

As per the performance analysis, simple Lenet-5 model (variant 4) where NO dropout and NO batch normalization layers are present in model, takes the least time to train and test followed by variant 1 i.e., model with dropout and batch normalization layers in model further followed by variant 3 i.e., model with only batch normalization layer and lastly variant 2 i.e., model with only dropout layer in the model which takes the most time for training and testing. Also, we discussed the reason for high accuracy across all the models which was due to MNIST dataset having 1 channel dataset i.e., only black, and white images and also some analysis on the accuracy of models after the first epoch.

8. References

1. https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4
2. https://cs231n.github.io/convolutional-networks/
3. https://en.wikipedia.org/wiki/Convolutional_neural_network
4. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
5. https://arxiv.org/pdf/1502.03167v3.pdf
6. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
7. https://en.wikipedia.org/wiki/MNIST_database