
RUTGERS, THE STATE UNIVERSITY OF NEW JERSEY

INTRODUCTION TO DEEP LEARNING PROJECT REPORT

---

# Performance Evaluation of Dropout and Batch Normalization Layers in CNN

---

Pratik Mistry (pdm79)
May 14, 2020

# 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 basic fundamentals of topics related to this experiment being performed.

## 2.1. What is a Convolutional Neural Network?

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.

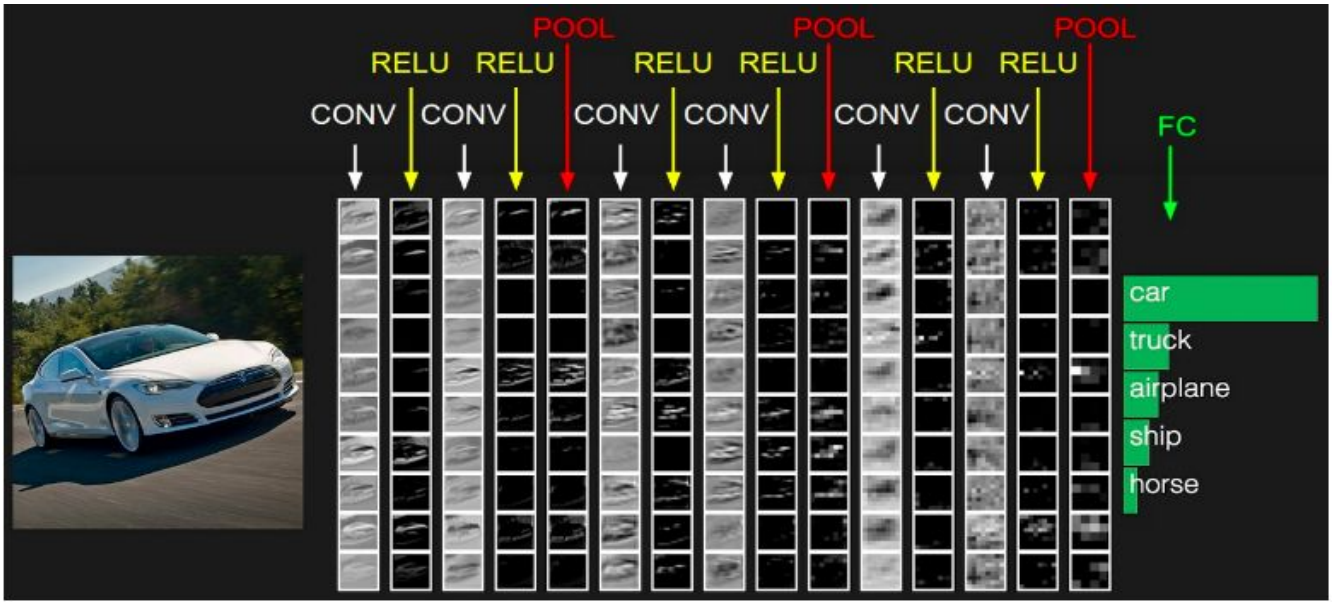



Figure 1: Simple Convolutional Network for Image Classification [2]

Technically convolutional layers means 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. [3]

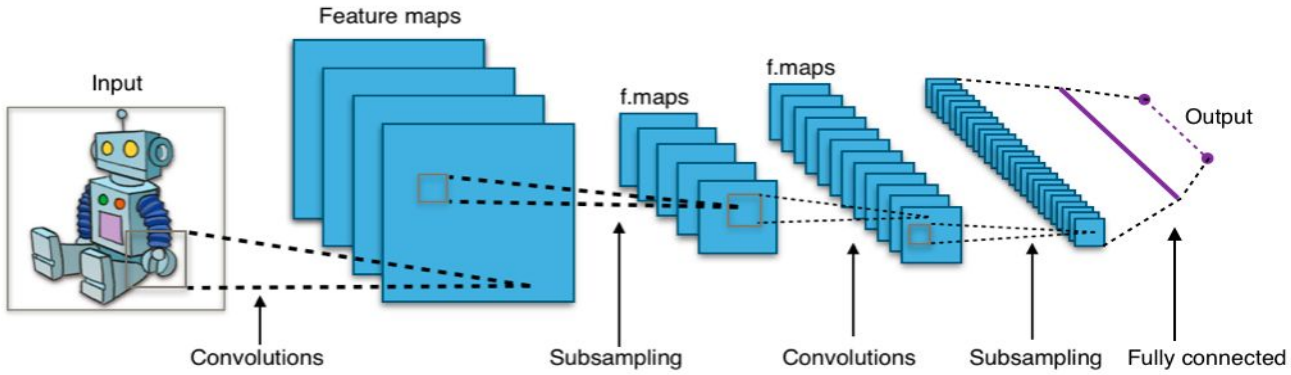



Figure 2: Typical CNN Architecture [3]

## 2.2. Lenet-5 Architecture

The first successful applications of Convolutional Networks were developed by Yann LeCun in the 1990's is the LeNet architecture [4] 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. Before Lenet was invented, character recognition had been done mostly by using feature engineering by hand, followed by a machine learning model to learn to classify hand engineered features. LeNet made hand engineering features redundant, because the network learns the best internal representation from raw images automatically. [1]

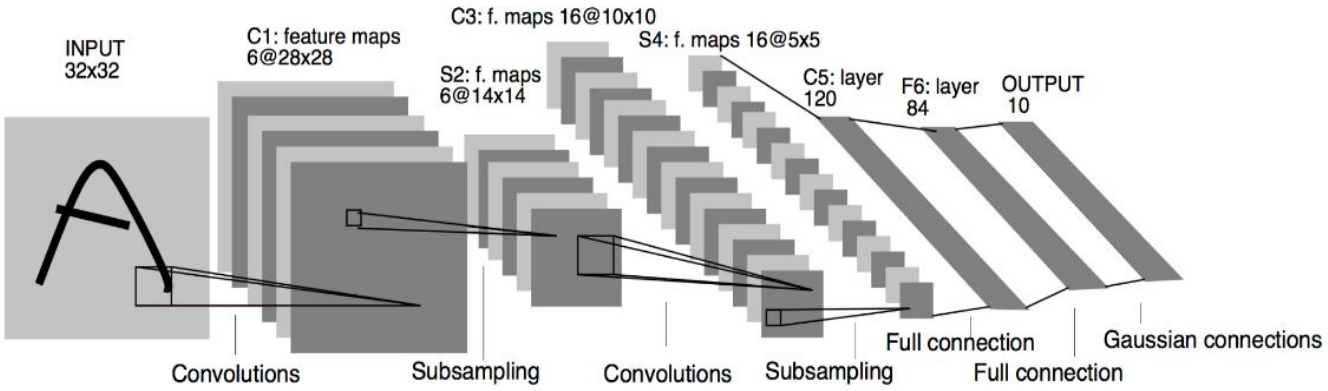



Figure 3: Simple Lenet-5 Architecture for Digits Recognition [4]

A *simple* Lenet-5 model shown in Figure 3 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. There are several interesting architectural choices that were made in LeNet-5 that are not very common in the modern era of deep learning.

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. [1]

LeNet-5 was able to achieve an error rate below 1% on the MNIST data set whose results we will discuss in further sections along with effects of adding combinations of Batch Normalization and Dropout Layers in the model.

## 2.3. 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. [5]

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 regularizer 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. Figure 4 shows the algorithm for Batch Normalization Transform, applied to activation x over a mini-batch.

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma$, $\beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 4: Algorithm for Batch Normalization Transform, applied to activation x over a mini-batch

## 2.4. 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. [6]

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.

Dropout works mainly because of below reasons:

1. If a hidden unit is always trained with another hidden unit, they will co-adapt to the training data
2. If a hidden unit has to work well with a different combination of other hidden neurons, it will optimize itself to be useful.
3. Each time we present a training batch, we randomly omit each hidden unit with probability p

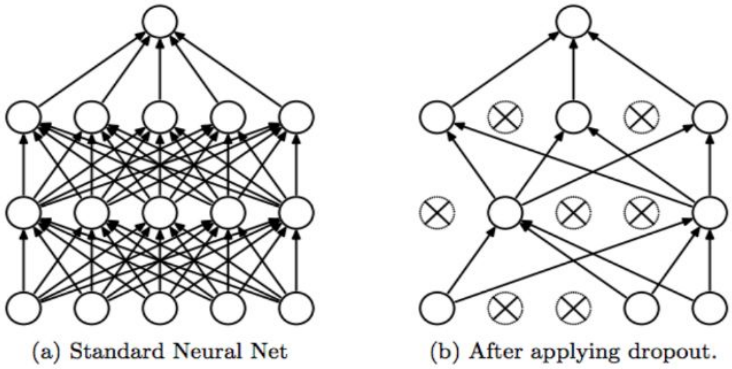



Figure 5: Illustration of Dropout Layer Effect on NN [6]

## 2.5. 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. [7]

Figure 6: Sample Images from MNIST Test Dataset [7]

# 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

## 3.1. General Configuration

- I 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 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 upto 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

## 3.2. 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**

## 3.3. 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**

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

## 3.4. 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.

## 3.5. No Dropout in FC and No Batch Normalization in CONV

As per last/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.

# 4. Experimental Results

## 4.1. Result Table

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

| **Lenet Architecture** | **Total Time** | |
|---|---|---|
| | **Seconds** | **Minutes** |
| With DO and BN | 4434.412794 | 73.9068799 |
| With DO | 4573.811659 | 76.23019432 |
| With BN | 4461.465428 | 74.35775714 |
| No DO and BN (Simple Lenet) | 4365.753066 | 72.7625511 |

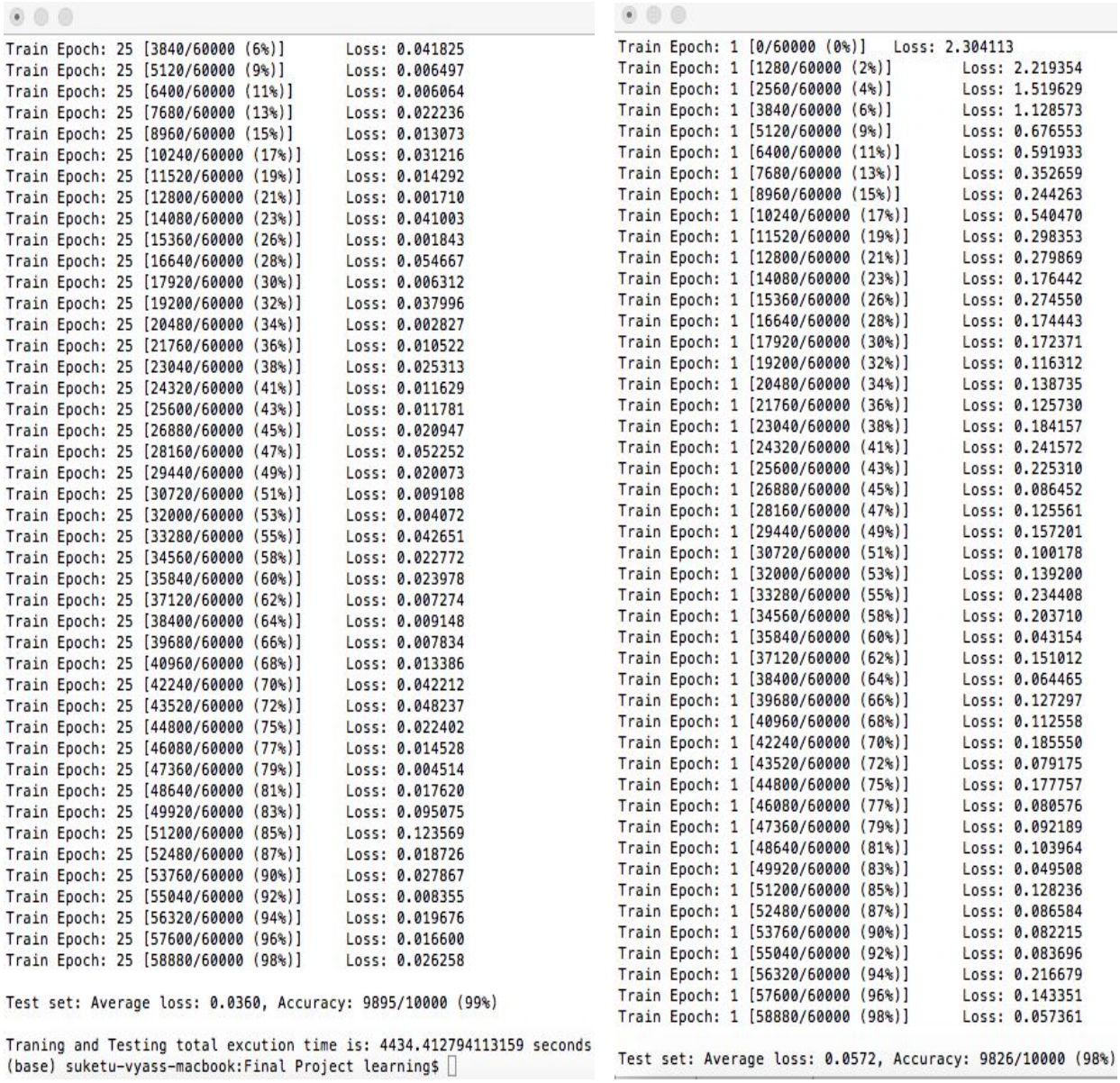
Table 1: Total time taken for different Lenet-5 Models

## 4.2. Results Discussion

Firstly, the performance in terms of accuracy for all the variations of Lenet-5 models is the same i.e. **99%**. So I won't discuss much on the accuracy aspect of the models implemented.

### 4.2.1. Dropout in FC and Batch Normalization in CONV

- Figure 7 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 7 (left image), it took around **4434.412794 seconds** i.e. **73.9068799 minutes** 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%** as seen in the Figure 7 (right image) below.
- The working code is uploaded in the project submitted on Sakai with name of the file as: **lenet_mnist_DO_BN.py**



```
Train Epoch: 25 [3840/60000 (6%)]       Loss: 0.041825
Train Epoch: 25 [5120/60000 (9%)]       Loss: 0.006497
Train Epoch: 25 [6400/60000 (11%)]      Loss: 0.006064
Train Epoch: 25 [7680/60000 (13%)]      Loss: 0.022236
Train Epoch: 25 [8960/60000 (15%)]      Loss: 0.013073
Train Epoch: 25 [10240/60000 (17%)]     Loss: 0.031216
Train Epoch: 25 [11520/60000 (19%)]     Loss: 0.014292
Train Epoch: 25 [12800/60000 (21%)]     Loss: 0.001710
Train Epoch: 25 [14080/60000 (23%)]     Loss: 0.041003
Train Epoch: 25 [15360/60000 (26%)]     Loss: 0.001843
Train Epoch: 25 [16640/60000 (28%)]     Loss: 0.054667
Train Epoch: 25 [17920/60000 (30%)]     Loss: 0.006312
Train Epoch: 25 [19200/60000 (32%)]     Loss: 0.037996
Train Epoch: 25 [20480/60000 (34%)]     Loss: 0.002827
Train Epoch: 25 [21760/60000 (36%)]     Loss: 0.010522
Train Epoch: 25 [23040/60000 (38%)]     Loss: 0.025313
Train Epoch: 25 [24320/60000 (41%)]     Loss: 0.011629
Train Epoch: 25 [25600/60000 (43%)]     Loss: 0.011781
Train Epoch: 25 [26880/60000 (45%)]     Loss: 0.020947
Train Epoch: 25 [28160/60000 (47%)]     Loss: 0.052252
Train Epoch: 25 [29440/60000 (49%)]     Loss: 0.020073
Train Epoch: 25 [30720/60000 (51%)]     Loss: 0.009108
Train Epoch: 25 [32000/60000 (53%)]     Loss: 0.004072
Train Epoch: 25 [33280/60000 (55%)]     Loss: 0.042651
Train Epoch: 25 [34560/60000 (58%)]     Loss: 0.022772
Train Epoch: 25 [35840/60000 (60%)]     Loss: 0.023978
Train Epoch: 25 [37120/60000 (62%)]     Loss: 0.007274
Train Epoch: 25 [38400/60000 (64%)]     Loss: 0.009148
Train Epoch: 25 [39680/60000 (66%)]     Loss: 0.007834
Train Epoch: 25 [40960/60000 (68%)]     Loss: 0.013386
Train Epoch: 25 [42240/60000 (70%)]     Loss: 0.042212
Train Epoch: 25 [43520/60000 (72%)]     Loss: 0.048237
Train Epoch: 25 [44800/60000 (75%)]     Loss: 0.022402
Train Epoch: 25 [46080/60000 (77%)]     Loss: 0.014528
Train Epoch: 25 [47360/60000 (79%)]     Loss: 0.004514
Train Epoch: 25 [48640/60000 (81%)]     Loss: 0.017620
Train Epoch: 25 [49920/60000 (83%)]     Loss: 0.095075
Train Epoch: 25 [51200/60000 (85%)]     Loss: 0.123569
Train Epoch: 25 [52480/60000 (87%)]     Loss: 0.018726
Train Epoch: 25 [53760/60000 (90%)]     Loss: 0.027867
Train Epoch: 25 [55040/60000 (92%)]     Loss: 0.008355
Train Epoch: 25 [56320/60000 (94%)]     Loss: 0.019676
Train Epoch: 25 [57600/60000 (96%)]     Loss: 0.016600
Train Epoch: 25 [58880/60000 (98%)]     Loss: 0.026258

Test set: Average loss: 0.0360, Accuracy: 9895/10000 (99%)

Traning and Testing total excution time is: 4434.412794113159 seconds
(base) suketu-vyass-macbook:Final Project learning$
```

```
Train Epoch: 1 [0/60000 (0%)]   Loss: 2.304113
Train Epoch: 1 [1280/60000 (2%)]        Loss: 2.219354
Train Epoch: 1 [2560/60000 (4%)]        Loss: 1.519629
Train Epoch: 1 [3840/60000 (6%)]        Loss: 1.128573
Train Epoch: 1 [5120/60000 (9%)]        Loss: 0.676553
Train Epoch: 1 [6400/60000 (11%)]       Loss: 0.591933
Train Epoch: 1 [7680/60000 (13%)]       Loss: 0.352659
Train Epoch: 1 [8960/60000 (15%)]       Loss: 0.244263
Train Epoch: 1 [10240/60000 (17%)]      Loss: 0.540470
Train Epoch: 1 [11520/60000 (19%)]      Loss: 0.298353
Train Epoch: 1 [12800/60000 (21%)]      Loss: 0.279869
Train Epoch: 1 [14080/60000 (23%)]      Loss: 0.176442
Train Epoch: 1 [15360/60000 (26%)]      Loss: 0.274550
Train Epoch: 1 [16640/60000 (28%)]      Loss: 0.174443
Train Epoch: 1 [17920/60000 (30%)]      Loss: 0.172371
Train Epoch: 1 [19200/60000 (32%)]      Loss: 0.116312
Train Epoch: 1 [20480/60000 (34%)]      Loss: 0.138735
Train Epoch: 1 [21760/60000 (36%)]      Loss: 0.125730
Train Epoch: 1 [23040/60000 (38%)]      Loss: 0.184157
Train Epoch: 1 [24320/60000 (41%)]      Loss: 0.241572
Train Epoch: 1 [25600/60000 (43%)]      Loss: 0.225310
Train Epoch: 1 [26880/60000 (45%)]      Loss: 0.086452
Train Epoch: 1 [28160/60000 (47%)]      Loss: 0.125561
Train Epoch: 1 [29440/60000 (49%)]      Loss: 0.157201
Train Epoch: 1 [30720/60000 (51%)]      Loss: 0.100178
Train Epoch: 1 [32000/60000 (53%)]      Loss: 0.139200
Train Epoch: 1 [33280/60000 (55%)]      Loss: 0.234408
Train Epoch: 1 [34560/60000 (58%)]      Loss: 0.203710
Train Epoch: 1 [35840/60000 (60%)]      Loss: 0.043154
Train Epoch: 1 [37120/60000 (62%)]      Loss: 0.151012
Train Epoch: 1 [38400/60000 (64%)]      Loss: 0.064465
Train Epoch: 1 [39680/60000 (66%)]      Loss: 0.127297
Train Epoch: 1 [40960/60000 (68%)]      Loss: 0.112558
Train Epoch: 1 [42240/60000 (70%)]      Loss: 0.185550
Train Epoch: 1 [43520/60000 (72%)]      Loss: 0.079175
Train Epoch: 1 [44800/60000 (75%)]      Loss: 0.177757
Train Epoch: 1 [46080/60000 (77%)]      Loss: 0.080576
Train Epoch: 1 [47360/60000 (79%)]      Loss: 0.092189
Train Epoch: 1 [48640/60000 (81%)]      Loss: 0.103964
Train Epoch: 1 [49920/60000 (83%)]      Loss: 0.049508
Train Epoch: 1 [51200/60000 (85%)]      Loss: 0.128236
Train Epoch: 1 [52480/60000 (87%)]      Loss: 0.086584
Train Epoch: 1 [53760/60000 (90%)]      Loss: 0.082215
Train Epoch: 1 [55040/60000 (92%)]      Loss: 0.083696
Train Epoch: 1 [56320/60000 (94%)]      Loss: 0.216679
Train Epoch: 1 [57600/60000 (96%)]      Loss: 0.143351
Train Epoch: 1 [58880/60000 (98%)]      Loss: 0.057361

Test set: Average loss: 0.0572, Accuracy: 9826/10000 (98%)
```

Figure 7: Left image shows accuracy of model (after last epoch) and total time taken.
Right image is accuracy (minimum accuracy) after first epoch

### 4.2.2. Dropout in FC

- Figure 8 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 8 (left image), it took around **4573.811659 seconds** i.e. **76.23019432 minutes** 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%** as seen in the Figure 8 (right image) below.
- The working code is uploaded in the project submitted on Sakai with name of the file as: **lenet_mnist_DO.py**

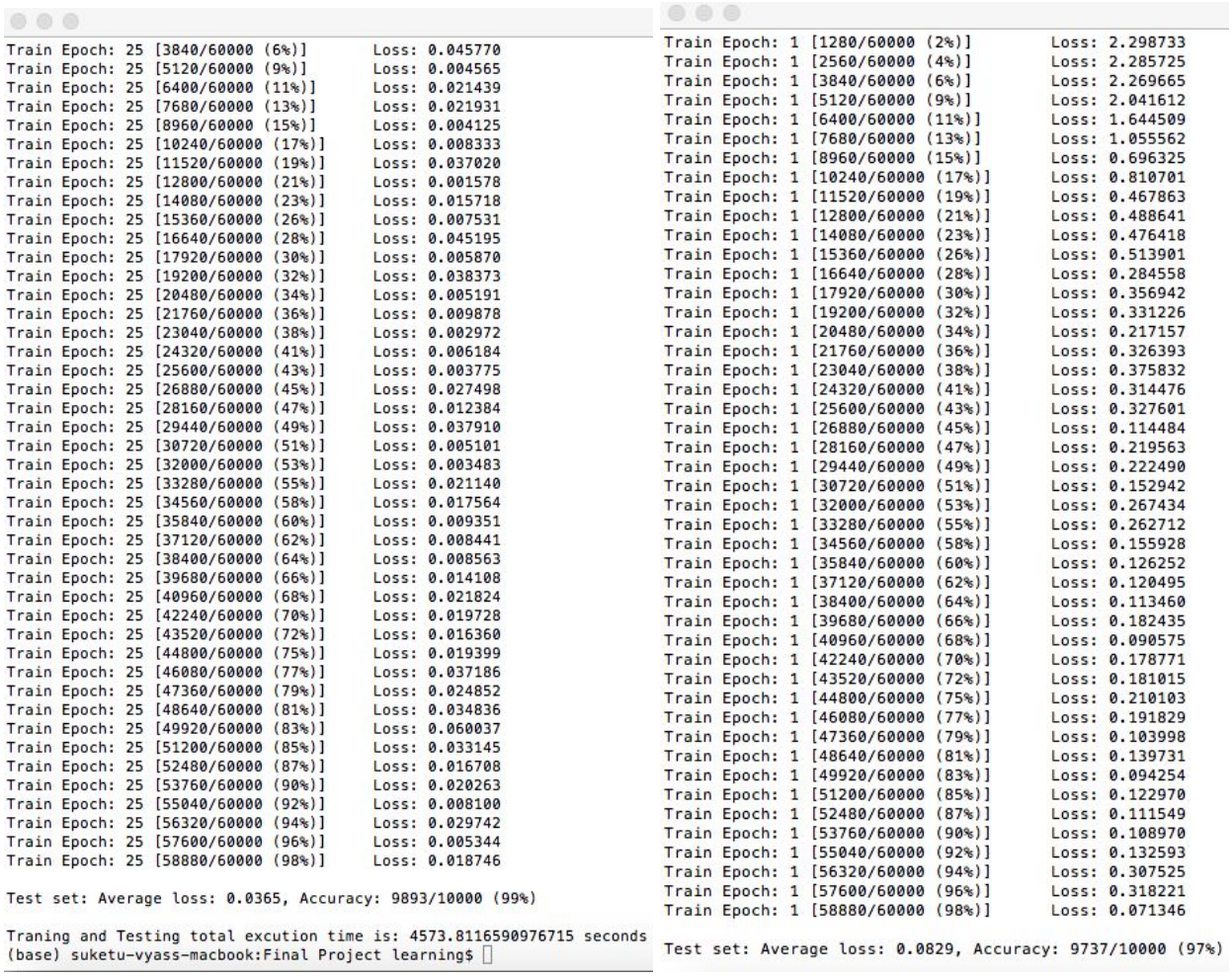



Figure 8: Left image shows accuracy of model (after last epoch) and total time taken. Right image is accuracy (minimum accuracy) after first epoch

### 4.2.3. Batch Normalization in CONV

- Figure 9 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 9 (left image), it took around **4461.465428 seconds** i.e. **74.35775714 minutes** 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%** as seen in the Figure 9 (right image) below.
- The working code is uploaded in the project submitted on Sakai with name of the file as: **lenet_mnist_BN.py**

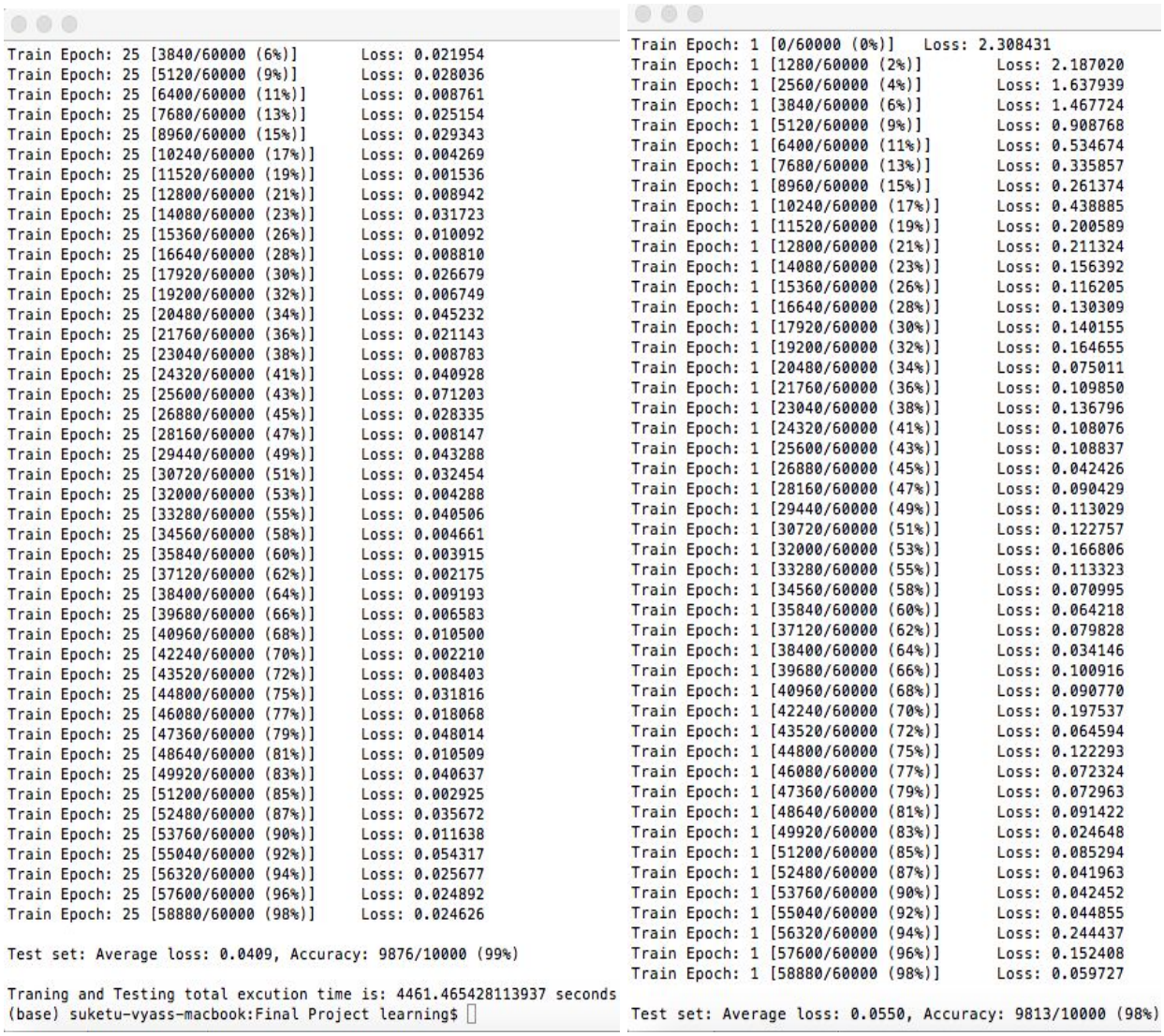



Figure 9: Left image shows accuracy of model (after last epoch) and total time taken. Right image is accuracy (minimum accuracy) after first epoch

### 4.2.4. No Dropout in FC and No Batch Normalization in CONV

- Figure 10 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 10 (left image), it took around **4365.753066 seconds** i.e. **72.7625511 minutes** 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%** as seen in the Figure 10 (right image) below.
- The working code is uploaded in the project submitted on Sakai with name of the file as: **lenet_mnist_No_DO_BN.py**

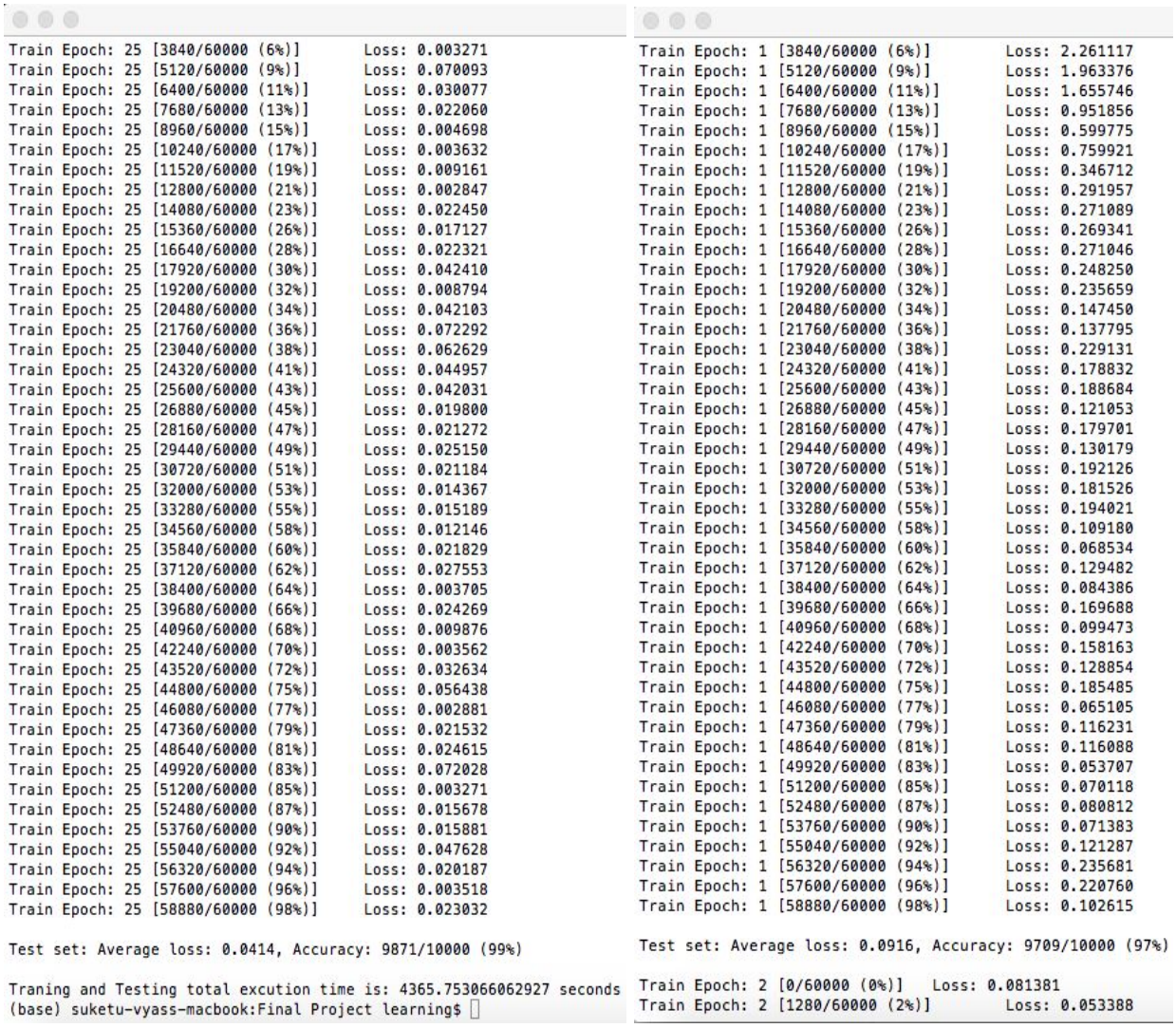



Figure 10: Left image shows accuracy of model (after last epoch) and total time taken. Right image is accuracy (minimum accuracy) after first epoch

# 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.

## 5.1. 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
- MNIST dataset consists of images in black and white format and thus it does not have any RGB channels like CIFAR-10 images *(which was used for Homework 4).*
- 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 was not the case with 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.

## 5.2. Minimum Accuracy (Accuracy after First Epoch)

- The minimum accuracy in the experiment performed was typically observed after the first epoch in all the variants of Lenet-5 model implemented in the project.
- The minimum accuracy(accuracy after the first epoch) of 97% was lowest for Lenet-5 models where only a dropout layer was present (variant 2) and where no dropout and batch normalization layer was present in the model (variant 4).
- This is because in the case where only a dropout layer is present in the model (variant 2), the model initially doesn't 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**).

## 5.3. Time Analysis

- In this section, we will compare performance between all the variants of Lenet-5 model with respect to the total time taken to 60000 training and 10000 testing MNIST images by referring to Table 1 mentioned in earlier Results section
- The largest time taken among all the variants of models implemented was for variant 2 i.e. where only the dropout layer is present in the model.
    - The primary reason is because the dropout is in FC layers i.e. after **f6** as per the architecture and very few features are dropped as input data to dropout is of size 84.
    - Another reason is the data is not normalized in the convolutional layers and thus the entire model trains on raw dataset.
- Further, the second largest time taken by the model is for variant 3 i.e. where only batch normalization layer is present in the model.
    - The reason why it takes less time compared to the variant 2 model discussed above is that data is always normalized after the first convolutional layer i.e. after **c1** and before the RELU layer (**relu1**).
    - Thus the model always trains and tests after being normalized with batch normalization layer (**bn1**) thus increasing the speed.
    - The reason why it takes more time compared to variant 1 is because here data is **NOT** dropped out with the help of dropout layer in fully connected layer.
    - Also, it takes slightly more time compared to variant 4 because everytime the data has to be normalized after the first convolutional layer i.e. **c1.**
- Now, the third largest time taken by the model is for variant 1 i.e. where both dropout and batch normalization is present in fully connected and convolutional layers.
    - The reason why it takes less time compared to variant 2 is because data is always normalized after first convolutional layer i.e. **c1,** thus the model always trains and test on normalized dataset
    - Also, the reason for less time compared to variant 3 is because some data is always dropped out after fully connected and relu layers i.e. **f6 and relu 6**, thus it takes less time to train and test
    - Now, the reason for taking more time compared to variant 4 is because the model always normalizes the dataset after the first convolutional layer i.e. **c1.**
    - Also, there is no effect of presence of dropout layer in the model compared to variant 4 where no dropout is present only because the dropout occurs in deeper layer i.e. after **f6** and **relu6** where the input data size is 84
- Lastly, the least time taken by the model is for variant 4 i.e. where no dropout and no batch normalization is present in fully connected and convolutional layers
    - I have discussed the above reasons for each of the variant but will again go through the reasons of taking least time for model training and testing

- The reason for taking less time compared to variants 1 and 3 is because there is no batch normalization layer present in convolutional layers. Thus, high computations for normalizing data is avoided by the model and training and testing of data happens on raw dataset.
- Also, it should ideally take more time compared to variant 2 i.e. where only a dropout layer is present in the model as no dropout occurs in the model. But the reason for taking less time is dropout in variant 2 occurs in deeper layers i.e. after **f6** and **relu6** where the data size is only 84
- Also, variant 4 avoids computation of randomly selecting neurons to drop compared to variant 2.
- As a result, there is not a high difference in total time taken compared to variant 2 as the difference is only **1.144 minutes.**
- Very Important: If the dropout layer could have inserted in convolutional layers i.e. typically after **c2 (***for variants 1 and 2)***,** where data size is typically higher, more neurons would have dropped and model would have taken less time to train and test compared to variant 4 where no dropout happens.

- Now, the primary reason why model training and testing for all the variants took around 72-77 minutes (which is very high) for only 25 epochs is because of the machine used whose specifications are mentioned in the next section.

## 5.4. Machine Specifications

Below are the machine specifications used for the performance analysis of variations in Lenet-5 models for MNIST dataset as discussed in previous sections:

*Machine used*: MacBook Air (Retina, 13-inch, 2019)
*Processor* - 1.6 Ghz intel Core i5
*Memory* - 8 GB 2133 MHz LPDDR3
*Graphics* - Intel UHD Graphics 617 1563 MB
*Operating System* - MacOS 10

# 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.

Lastly, to conclude the models with dropout layer could have taken less training and testing time i.e. less overall time if dropout could have added in the convolutional layers i.e. typically after **c1 or c2** as the data size in those layers is typically high and dropping neurons in those layers could have been very efficient.

# 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