# Programming Assignment 3

Amar Vashishth(CS17m052) & Subrajit Makur(CS17m046)

March 30, 2018

# Contents

## 0.1 Checkbox

- Implementation of:
  - ☑ Batch Normalization
  - ☑ Initializations (Xavier, He)
  - ☑ Early Stopping
- Kaggle:
  - ☑ Submitted Accuracy = 94.066%


- Also Contains:
  - ☑ Plot of Learning Curves of Learning Loss Vs Validation Loss
  - ☑ Answers to Questions in Assignment
  - ☑ Plot of all 64 layers 1 Filters
  - ☑ Guided Back propagation on 10 neurons
- Experiments with:
  - ☑ Initializers
  - ☑ Batch Sizes
  - ☑ Number of Epochs
  - ☑ Strides
  - ☑ Learning Rates
  - ☑ Optimizers
- Bonus:
  - ☑ Data Augmentation
  - ☑ Network Fooling
  - ☑ Experiments with Hyper-parameters and trying out uncommon ones.

## 0.2 Base(default) Network Configuration

As per the given instructions in the assignment we have successfully implemented the given configuration. We call this CNN configuration as the the base configuration to proceed with.

The Base Network Configuration follows as:

1. Input Layer
2. 2D Convolution Layer with a kernel size of $3 \times 3$ running with $relu$ activation units; Outputs a volume of 32 Filter units.
3. 2D Max Pooling Layer, kernel size $= 3 \times 3$
4. 2D Convolution Layer, kernel size $= 3 \times 3$, outputs a volume of 128 Filter units.
5. 2D Max Pooling Layer, kernel size $= 2 \times 2$
6. 2D Convolution Layer, kernel size $= 3 \times 3$, outputs a volume of 256 Filter units.
7. 2D Convolution Layer, kernel size $= 3 \times 3$, outputs a volume of 256 Filter units.
8. 2D Max Pooling Layer, kernel size $= 2 \times 2$
9. Dense Layer(aka fully connected layer), output a volume of 1024 Filter units.
10. Dense Layer, output a volume of 1024 Filter units.
11. Logistic Layer, it's a fully connected layer running with "Softmax" activation function.

When trained on 55000 Fashion-MNIST data samples, with a batch size of 500 sample units over 10 epochs, i.e. 1100 steps. It successfully predicts classes with an accuracy of 91.32%. Final loss value being **0.24862008**.

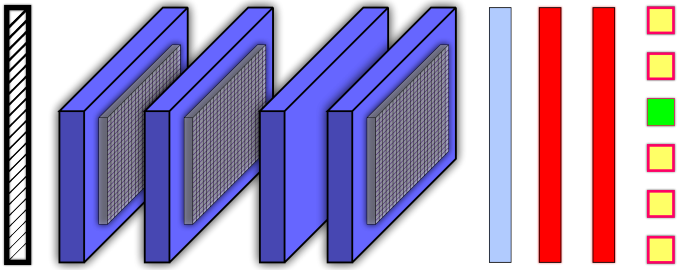

Figure 1: Base CNN Configuration: Input Layer, Convolution Layer, Max Pool layer, Convolution, Convolution, Max Pool, Flatten, Dense, Dense, Logistic

## 0.3 Learning Curves of Training Loss Vs Validation Loss

Following 4 plots are wrt. to both base(default) cnn configuration as well as our custom designed CNN configuration. These plots of Loss and accuracy are wrt. number of epochs.

## 0.4 Plots of Layers

Following shows Plot of layers:

## 0.5 Guided Back Propagation on Neurons

The following results were obtained when performed back propagation from some neurons.

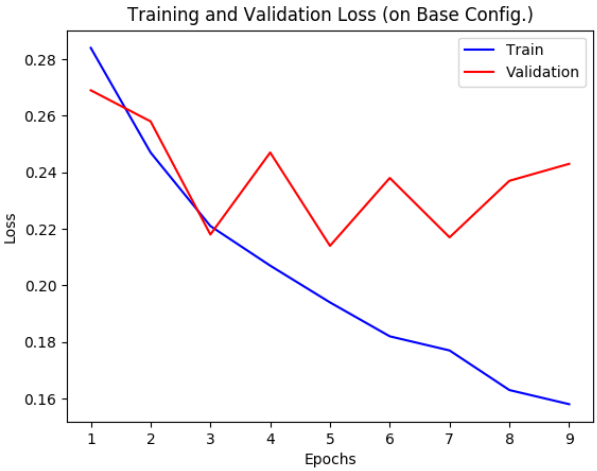


Figure 2: Plot with respect to Base CNN Configuration

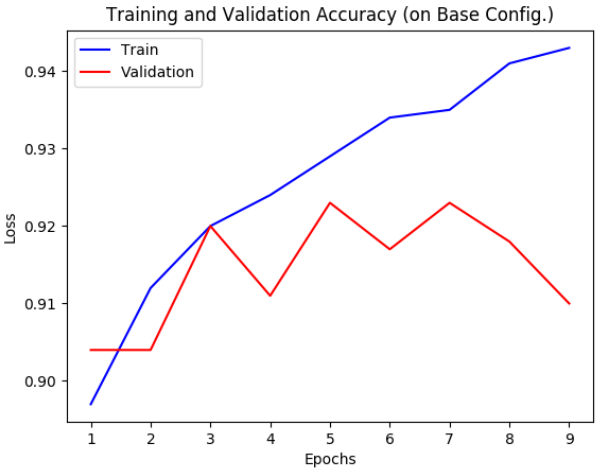


Figure 3: Plot with respect to Base CNN Configuration


Figure 4: Plot with respect to Custom Best CNN Configuration

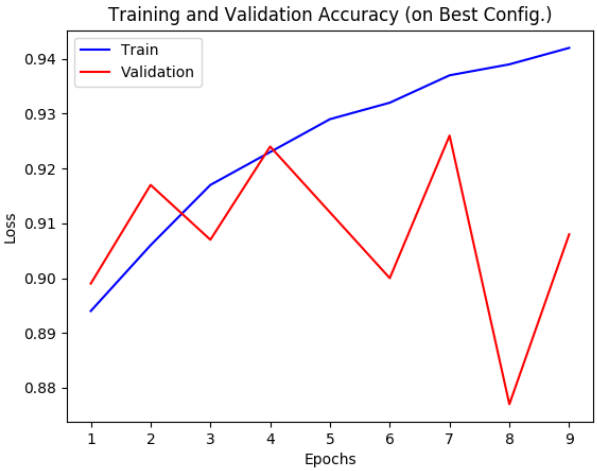


Figure 5: Plot with respect to Custom Best CNN Configuration

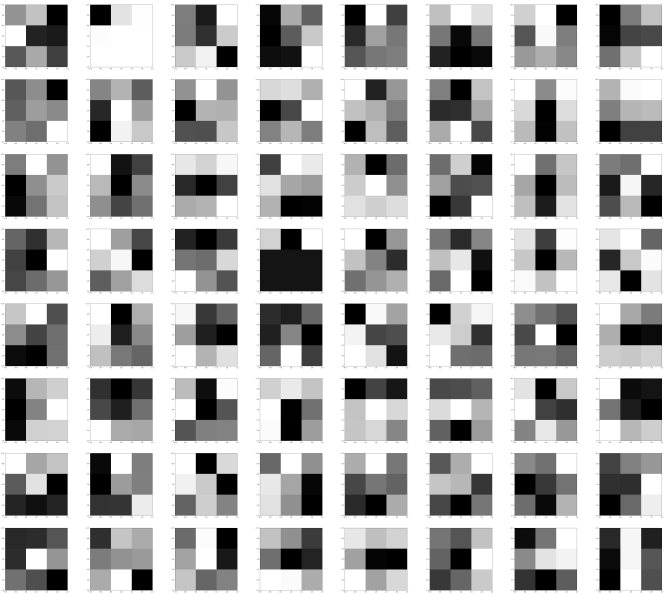

Figure 6: Plot of layers

## 0.6 Hyper Parameter Tuning

### Using Different Initializers

We augmented our base CNN configuration once with Xavier Initializer and then with He Initializer, the following results were obtained:

### Using Different Batch Sizes

We tried with a variety of Batch sizes keeping number of epochs and training data items same. Since, total number of steps depends on number of data items, batch size and epochs; they also varied in accordance with the formula:

$$\#\text{Steps} = \frac{\text{Total Number of Data Items}}{\text{Batch Size}} \times \text{Total Number of Epochs}$$

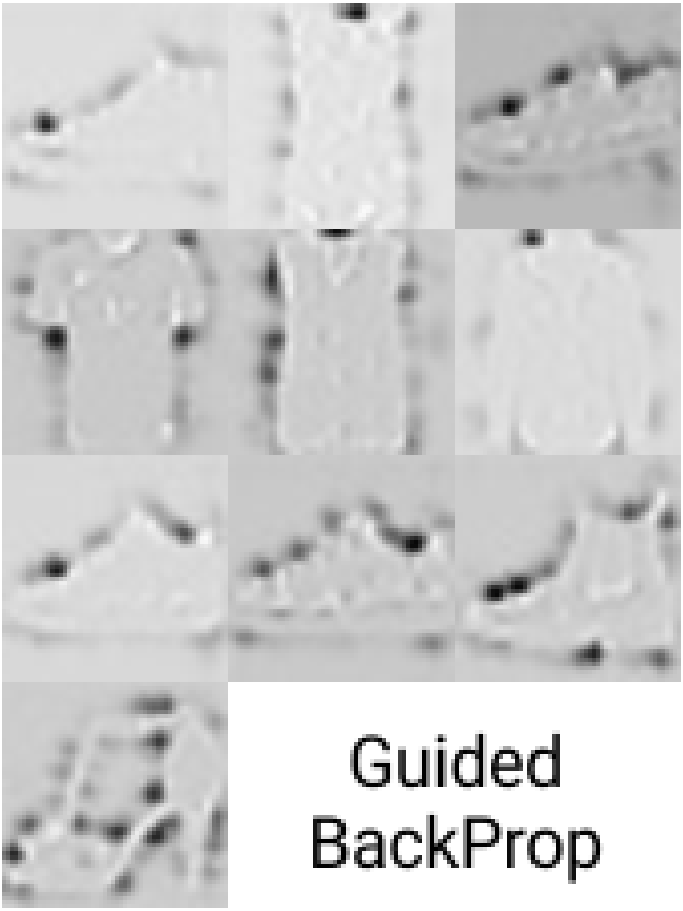


Figure 7: Results of Guided Back Propagation

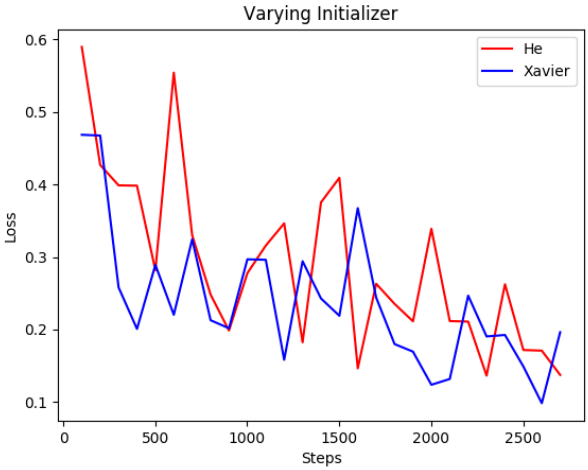


Figure 8: Using different initializers with Base configuration

Following observations were made:

| Batch Size | Accuracy | Loss |
|---|---|---|
| 50 | 91.80% | 0.23633663 |
| 100 | 92.08% | 0.22596498 |
| 500 | 91.68% | 0.23784976 |

### Using Different Number of Epochs

We tried running our base model on different number of epochs. Following is the result obtained:

### Using Different Number of Strides

We tried running our base model with different strides values. Following observations were made:

### Using Varying Learning Rates

We have used Adam Optimizer, it's learning rate was varied and the following results were obtaineds:

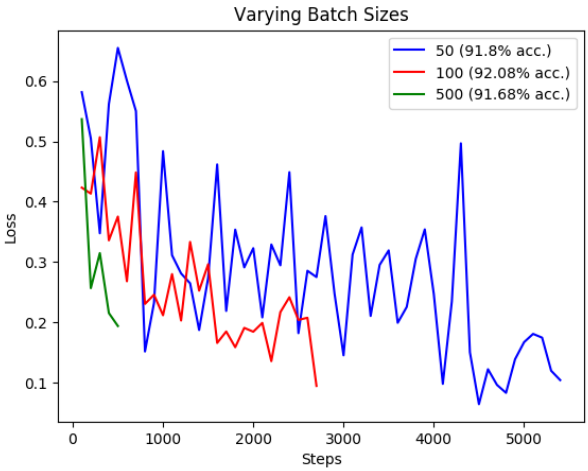


Figure 9: Experiments with Different Batch Sizes

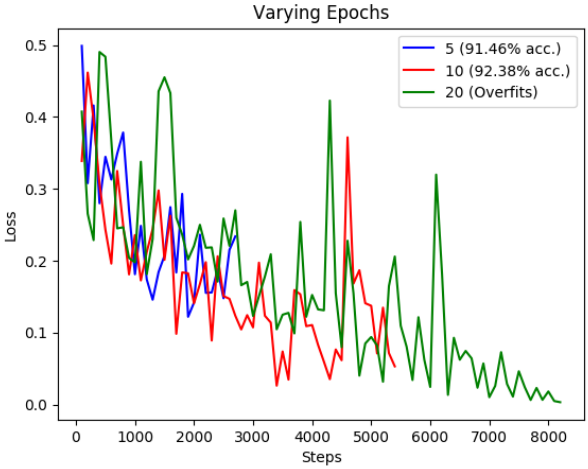


Figure 10: Experiments with varying Epochs

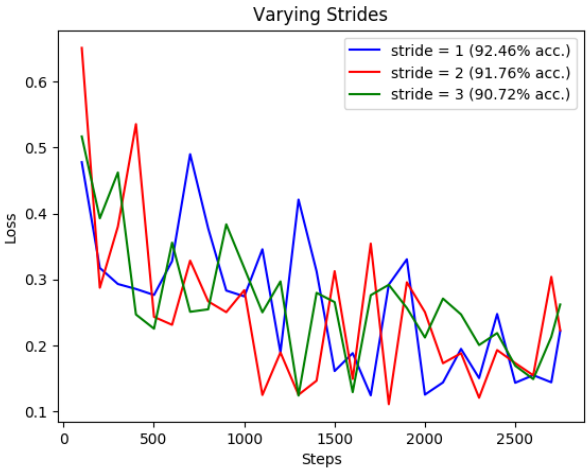


Figure 11: Experiments with varying Strides

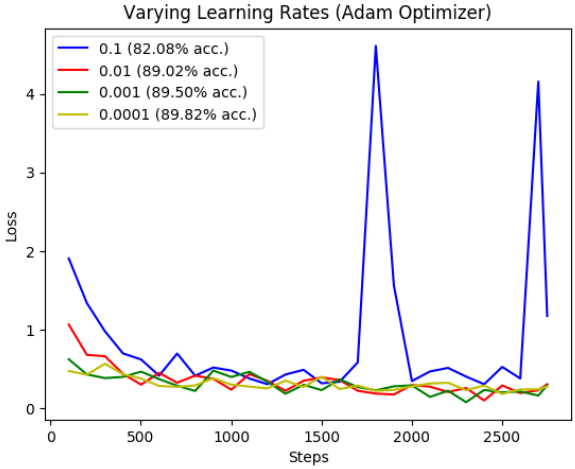


Figure 12: Experiments with varying Learning Rates of Adam Optimizer

### Using Different Optimizers

We also experimented using different optimizers available, such as Adam, Adagrad and Gradient Descent. Here is a comparison between all of them:

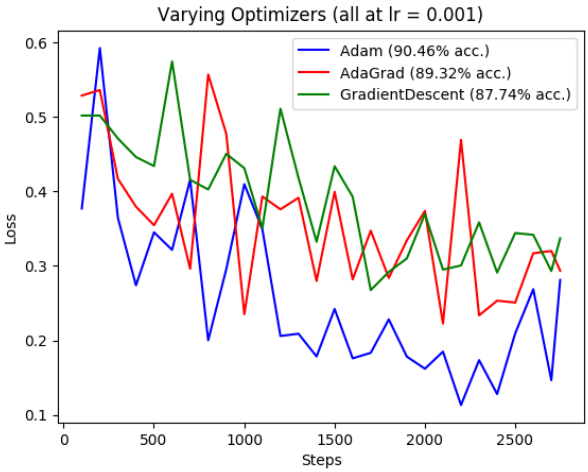


Figure 13: Experiments with different Optimizers

## 0.7 Network Configuration

The Convolutional Neural Network crafted so far has been successfully able to predict classes belonging to Fashion-MNIST dataset with an accuracy of 94.066%.

This is achievable using the following configuration of the Convolutional neural network:

1. Input Layer
2. 2D Convolution Layer with a kernel size of $3 \times 3$ running with *relu* activation, that gives output a volume of 32 Filter Units.

3. 2D Convolution Layer, kernel size = $3 \times 3$, activation = $relu$, output = 64 filter units.
4. Batch Normalization Layer
5. 2D Max pooling layer with filter size = $2 \times 2$ with strides of 1
6. 2D Max Pooling layer, kernel size = $2 \times 2$, strides = 1
7. 2D Convolution Layer, kernel size = $3 \times 3$, activation = $relu$, output = 256 Filter units.
8. 2D Convolution Layer, kernel size = $3 \times 3$, activation = $relu$, output = 256 Filter units.
9. 2D Max Pooling layer, kernel size = $3 \times 3$, strides = 1
10. Flatten Layer
11. Dropout layer, rate = 0.4
12. Dense Layer, output = 1024 units, activation = $elu$(**not** using $relu$ here)
13. Batch normalization layer
14. Dropout Layer, rate = 0.4
15. Dense Layer, output = 1024 units, activation = $elu$
16. Batch Normalization Layer
17. Logistic Layer, output = 10 units, activation = $softmax$

# Fooling the Network

Successfully we were able to fool the network with the following images: In case of targetted mode, where we try to fool to a specific class, this confidence level is less. During untargetted mode, the Fooling was happening with high confidence in outputs.

| **Layer Name** | **Input Dimensions** | **Output Dimensions** |
|---|---|---|
| Conv1 | [BatchSize, 1, 28, 28] | [BatchSize, 64, 28, 28] |
| Pool1 | [BatchSize, 64, 28, 28] | [BatchSize, 64, 27, 27] |
| Conv2 | [BatchSize, 64, 27, 27] | [BatchSize, 128, 27, 27] |
| Pool2 | [BatchSize, 128, 27, 27] | [BatchSize, 128, 26, 26] |
| Conv3 | [BatchSize, 128, 26, 26] | [BatchSize, 256, 26, 26] |
| Conv4 | [BatchSize, 256, 26, 26] | [BatchSize, 256, 26, 26] |
| Pool3 | [BatchSize, 256, 26, 26] | [BatchSize, 256, 25, 25] |
| FC1 | [BatchSize, 256, 25, 25] | [BatchSize, 1024] |
| FC2 | [BatchSize, 1024] | [BatchSize, 1024] |
| Softmax | [BatchSize, 1024] | [BatchSize, 10] |

Table 1: Dimensions of our Neural Network

| | |
|---|---|
| Conv1 | $64 \times 3 \times 3 = 576$ |
| Pool1 | 0 |
| Conv2 | $64 \times 128 \times 3 \times 3 = 73728$ |
| Pool2 | 0 |
| Conv3 | $128 \times 256 \times 3 \times 3 = 294912$ |
| Conv4 | $256 \times 256 \times 3 \times 3 = 589824$ |
| Pool3 | 0 |
| FC1 | $1024 \times 256 \times 25 \times 25 + 1024 = 641024$ |
| FC2 | $1024 \times 1024 + 1024 = 1049600$ |
| Softmax | $1024 \times 10 + 10 = 10250$ |
| **Total** | Conv(969290) + FC(1690624) = **2659914** |

Table 2: Number of Parameters in Final Network

| | |
|---|---|
| Conv1 | $64 \times 3 \times 3 = 576$ |
| Pool1 | 0 |
| Conv2 | $128 \times 3 \times 3 = 1152$ |
| Pool2 | 0 |
| Conv3 | $256 \times 3 \times 3 = 2304$ |
| Conv4 | $256 \times 3 \times 3 = 2304$ |
| Pool3 | 0 |
| FC1 | 1024 |
| FC2 | 1024 |
| Softmax | 10 |
| **Total** | Conv(6336) + FC(2058) = **8394** |

Table 3: Number of Neurons in Final Network

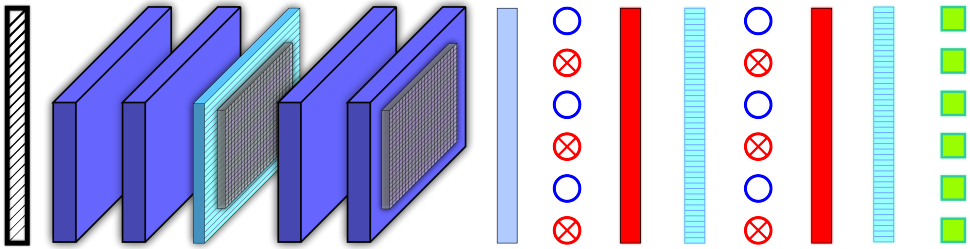

Figure 14: Configuration of the CNN(Forward Propagation from left to right): Input layer, Convo2D, Convo2D, Batch Norm, Max Pool 2D, Convo2D, Convo2D, Max Pool 2D, Flatten, Dropout, Dense, Batch Norm, Dropout, Dense, Batch Norm, Logistic.

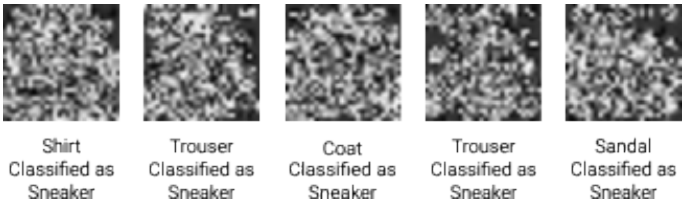


Figure 15: Fooling the Network

# Affect of Batch Normalization

It helps prevent both explosion and vanishing of gradients. It also acts as a regularization. It diminishes the need of L2 Regularization and Dropouts.

# Data Augmentation

We performed it using two ways. First during runtime, Random Erasing was utilized.

Also, as a separate experiment we fed the network with pre-augmented training data, which was generated using, augdata.py. We tried Horizontal flipping, Vertical flipping, Random rotation and Random translate of image.