Report on Assignment P2 in CS7GV1 (Computer Vision)

Aditya Misra misraa@tcd.ie 18302706

ABSTRACT

This report presents a detailed description regarding the deep learning implementation on Fashion-MNIST data-set. It throws light on the architectural choices for the neural network implemented, describes the training and testing part of the same and also contains the results in the form of graphs showing train/test error and loss.

KEYWORDS

Fashion-MNIST, CNN, PyTorch, Neural Network.

I. INTRODUCTION

For this assignment, I have used one of the popular and growing Deep Learning frameworks, PyTorch, by implementing a Convolutional Neural Network on Fashion MNIST data-set.

PyTorch offers several advantages as a Deep Learning framework. Firstly, it is much easier to use in Python because of it executes code at run-time. In PyTorch, we define the graph as a class of type nn.module, and feed the input data through it. The code runs as the class is called. Secondly, the code is easier to read and intuitive, and because of its run-time execution model, it is easy to debug the code as the data passes through the model. [1]

For this project, I am using the Fashion MNIST data-set. The Fashion MNIST data-set consists of Zalando's article images, with grayscale images of size 28x28, developed as a drop-in replacement for the MNIST handwritten digits data-set. *{https://github.com/zalandoresearch/fashion-mnist}. (Fig.1)*

II. METHODOLOGY

To train images, I have used a convolutional neural network (CNN). CNNs are a type of deep layer neural networks, used to learn Filters that when convolved with the image, can be used to extract features. The CNN architecture model has 2 convolution layers, with 5x5 kernels, followed by a fully connected layer, and a final activation for the last output layer. (Fig. 2)

Convolutional Neural Networks (CNN) is variants of Multi-Layer Perceptron (MLPs) which are inspired from biology. These filters are local in input space and are thus better suited to exploit the strong spatially local correlation present in natural images. Convolutional neural networks are designed to process two-dimensional (2-D) image [2, 3].

A variety of operation were performed on the CNN Model and the accuracy result changed accordingly. At first, I only added layers in the model and performed the batch normalization and ReLU on the output of each CNN layer.
The test accuracy being obtained through this was around 89%. Initially in this model, I had used the SGD booster but tried to play with different boosters and in the end two boosters in particular, namely Adam and Adagrad only gave a substantial increase in the accuracy.
Then I also tried different learning rates, with the same model but the accuracy had reached a stagnantation at 90%.

After this, I remodeled my CNN architecture (Fig. 3) and used 3 layers instead of two. The model uses a new activation function called Swish activation. It has been seen to perform better than ELU and ReLU in some cases and thus, I used it in all the 3 Convolutional Layers. I set the optimizer finally to Adagrad, as it was giving the highest accuracy previously. I introduced a decay in the learning rate after every 3 epochs and the decay value being 0.003 (The learning rate was set to 0.015 after many trials, since it was the most successful), the accuracy increased to nearly 91 %.

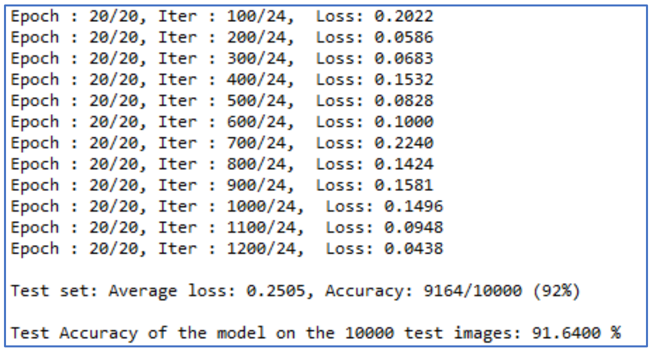
I also tried to tweak the model a bit to give similar or higher accuracy with lesser (2) layers and used deeper initial layers, say 32 instead of 16 and so on, and was successful to push the accuracy around 92% this time.

The final CNN Model (Fig 3), hence, has the following layers in sequence:

Convolutional layer with 32 feature maps of size 3 x 3
BatchNorm layer followed by Swish activation.
Max Pooling layer of size 2 x 2.
Convolutional layer with 64 feature maps of size 3 x 3
BatchNorm layer followed by Swish activation.
Max Pooling layer of size 2 x 2.
Fully connected layer of size 10.

III. RESULTS

This CNN Model coupled with Adagrad Booster, Learning rate = 0.015(with learning rate decay at every 3 epochs by a factor of 0.003 to improve the performance), batch size = 50 and number of epochs = 20 gives an accuracy of around 92%, which is a substantial increase of 3% from my initial state.



```
Epoch : 20/20, Iter : 100/24,  Loss: 0.2022
Epoch : 20/20, Iter : 200/24,  Loss: 0.0586
Epoch : 20/20, Iter : 300/24,  Loss: 0.0683
Epoch : 20/20, Iter : 400/24,  Loss: 0.1532
Epoch : 20/20, Iter : 500/24,  Loss: 0.0828
Epoch : 20/20, Iter : 600/24,  Loss: 0.1000
Epoch : 20/20, Iter : 700/24,  Loss: 0.2240
Epoch : 20/20, Iter : 800/24,  Loss: 0.1424
Epoch : 20/20, Iter : 900/24,  Loss: 0.1581
Epoch : 20/20, Iter : 1000/24,  Loss: 0.1496
Epoch : 20/20, Iter : 1100/24,  Loss: 0.0948
Epoch : 20/20, Iter : 1200/24,  Loss: 0.0438

Test set: Average loss: 0.2505, Accuracy: 9164/10000 (92%)

Test Accuracy of the model on the 10000 test images: 91.6400 %
```

The graphs showing train/test error (Fig. 4) and loss during training (Fig. 5) can be seen in the appendix section.

ACKNOWLEDGEMENTS

I would like to thank my Professor for the module Prof. Rozenn Dahyot for imparting me with the valuable knowledge regarding the classical computer vision techniques during the course. I would also like to extend my gratitude towards Mr. Matej Ulciny for providing us with the Deep Learning Lab Tutorials, demo codes and giving the regular invaluable feedbacks on the performance of the deep learning models.

REFERENCES

[1] Fei-Fei Li, Justin Johnson, Serena Yeung, CS231n Lecture 8 Deep Learning Software, Stanford University, 2017.
[2] Yann LeCun, Leon Bottou, Yodhua Bengio and Patrick Haffner, "Gradient-Based Learning Applied to Document Recognition", IEEE, November 1998.
[3] Jaswal, Deepika & Vishvanathan, Sowmya & Kp, Soman. (2014). Image Classification Using Convolutional Neural Networks. International Journal of Scientific and Engineering Research. 5. 1661-1668. 10.14299/ijser.2014.06.002.

APPENDIX

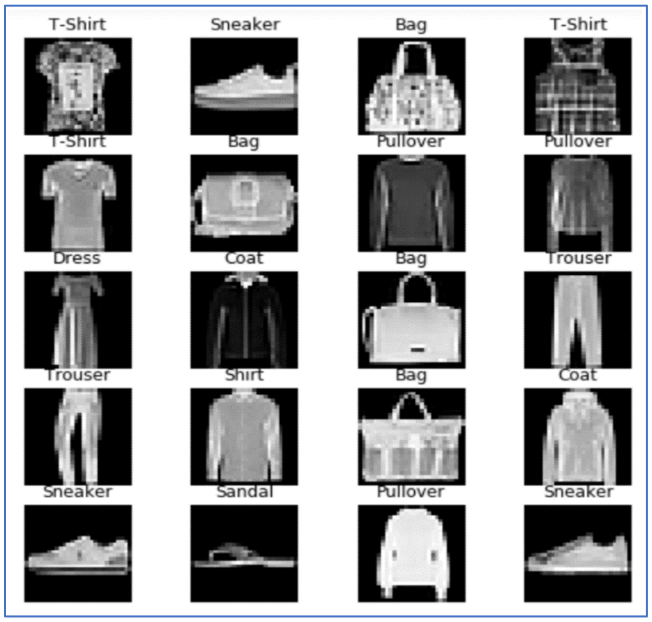


Fig 1: Sample of Images from the Training Dataset

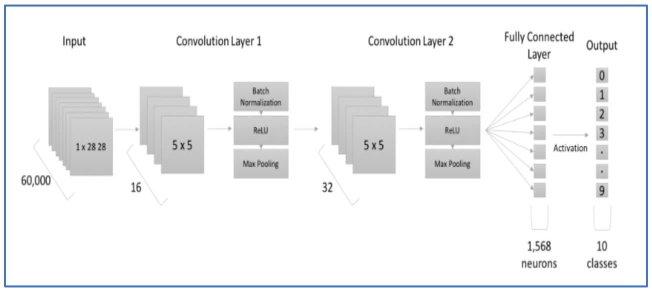
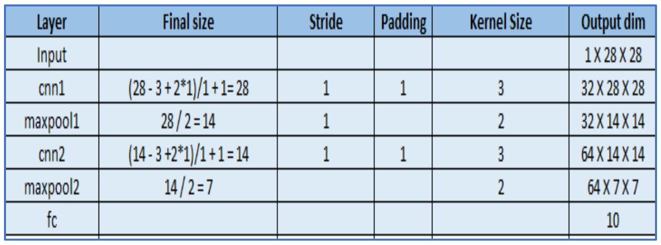


Fig 2: Initial CNN Architecture



Layer	Final size	Stride	Padding	Kernel Size	Output dim
Input					1 X 28 X 28
cnn1	(28 - 3 + 2*1)/1 + 1 = 28	1	1	3	32 X 28 X 28
maxpool1	28 / 2 = 14	1		2	32 X 14 X 14
cnn2	(14 - 3 +2*1)/1 + 1 = 14	1	1	3	64 X 14 X 14
maxpool2	14 / 2 = 7			2	64 X 7 X 7
fc					10

Fig 3: Final CNN Architecture

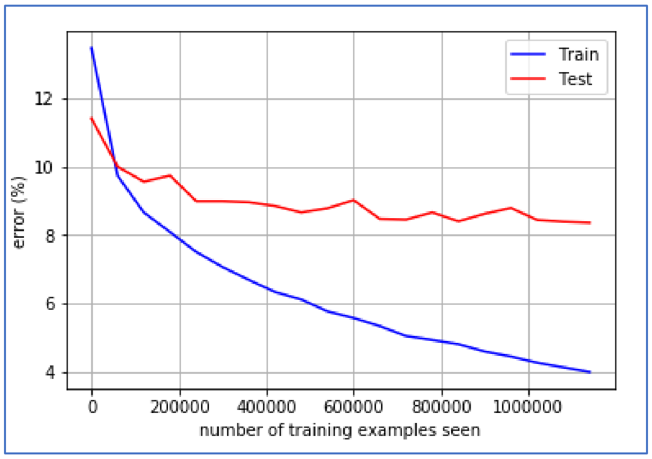


Fig 4: Train & Test Error

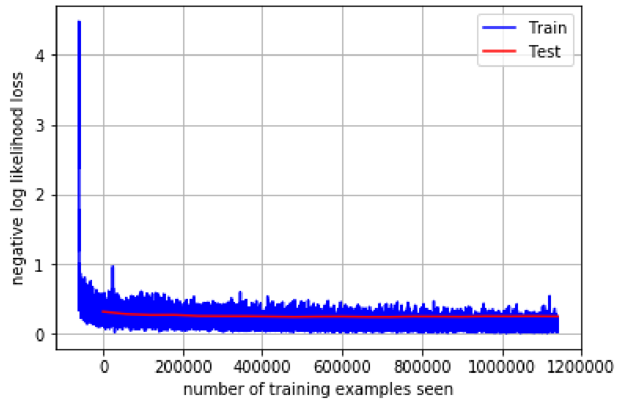


Fig 5: Log of likelihood loss