CSA 250 : Deep Learning Project II
Report

Abhishek Kumar (15648)

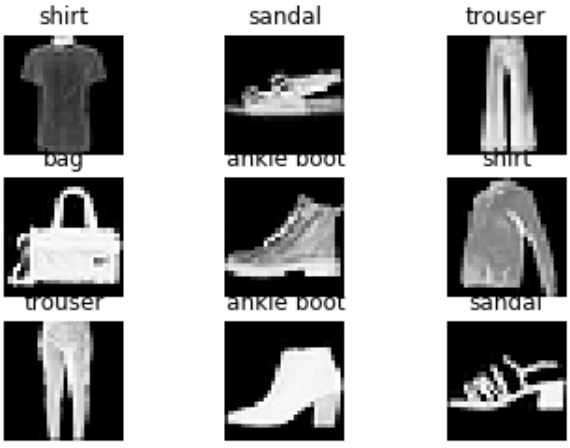
February 21, 2020

Project II : Problem Statement

This project is to implement neural network and convolutional neural network for the task of classification. The classification task will be that of recognizing an image and identify it as one of ten classes. You are required to train the classifiers using Fashion-MNIST clothing images. Following are the twotasks to be performed:
1). Build multi-layer Neural Network with open-source neural-network library (pytorch/tensorflow)on Fashion-MNIST dataset.
2). Build Convolutional Neural Network with open-source neural-network library (pytorch/tensorflow)on Fashion-MNIST dataset.

Dataset

The dataset for this project is Fashion-MNIST. Fashion-MNIST is a dataset of Zalando's article images-consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.



Dataset Preprocessing
To preprocess the dataset the followings steps were followed:
a) Downloaded the training and test datset using the standard torchvision.dataset library.

b) Convert the raw training and test images into tensors tobe used by the neural network
c) Split the dataset into split ratio 9:1 for train and validation set respectively.

Model : Multi-Layer Neural Network

• **Initial Model Architecure** :
No. of hidden layers : 4
Layer 1 : 1024 neurons
Layer 2 : 512 neurons
Layer 3 : 64 neurons
layer 4 : 20 neurons and it is connected to output layer consisting of 10 neurons.

Initial Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.01 and momentum 0.9.
Epochs : 100 epochs (Until validation error started increasing)
Batch Size : 1024
Initial Results:
Accuracy :88.33% and Test loss : 0.001641760636420467

The main idea to start with this architecture is that the input feature is 784 dimensional and we wanted to project it to higher dimensional such that the classes can be classified linearly. Subsequently I reduced it to the number of classes i.e.,10. • **Final Model Architecure** :
No. of hidden layers : 4
Layer 1 : 1024 neurons
Layer 2 : Dropout layer with p = 0.5 Layer 3 : 512 neurons
Layer 4 : 128 neurons and it is connected to output layer consisting of 10 neurons.

Final Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : Adam with learning rate 0.01, $\beta_1 = 0.9$ and $\beta_2 = 0.98$
Epochs : 300 epochs (Until validation error started increasing)
Batch Size : 1024
Final Results:
Accuracy :89.3% and Test loss : 0.0002153349013776826

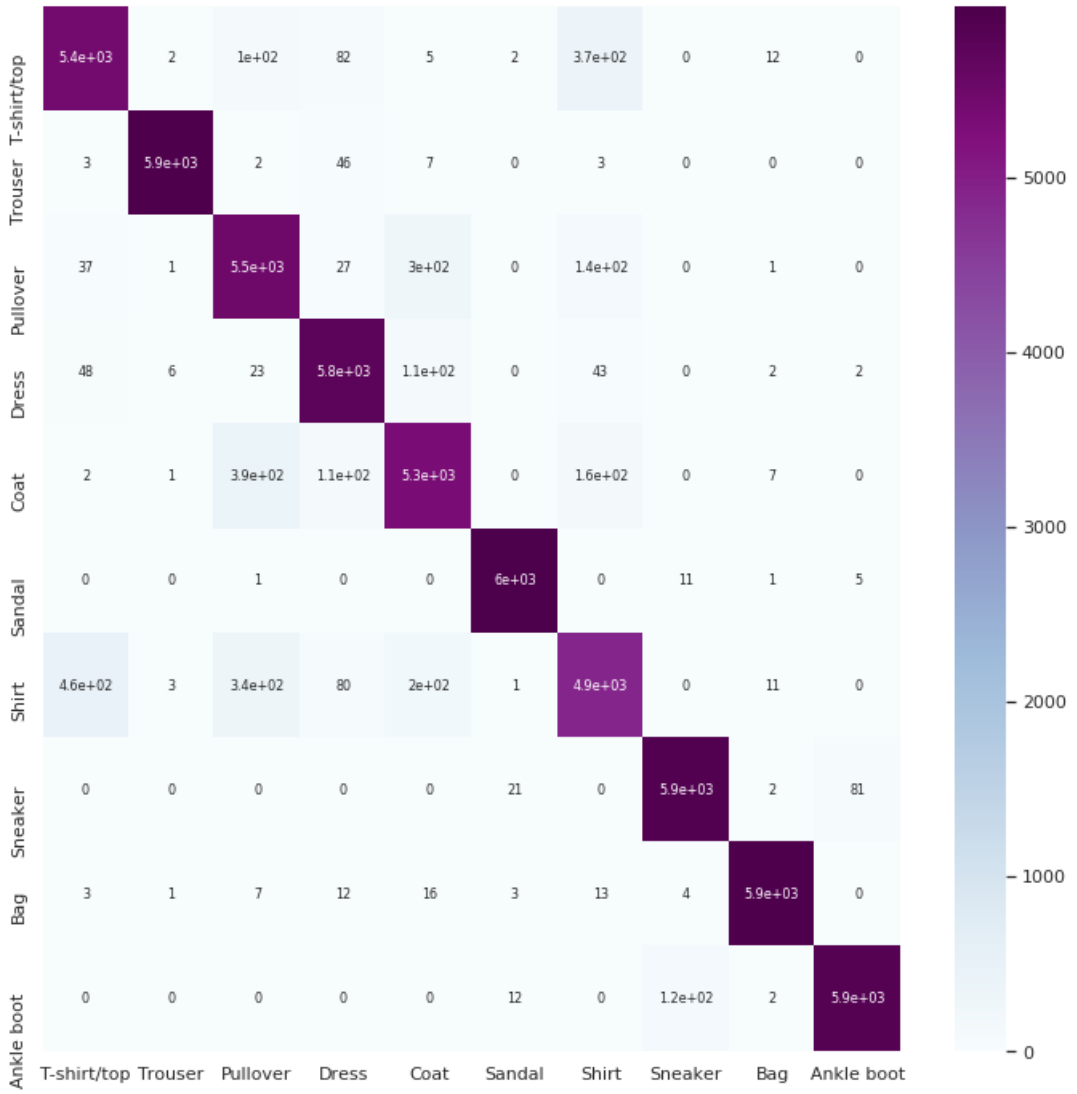
Figure 1: MLP Training Loss

I added a dropout layer after the first layer to add regularization and also to make the model more robust. It will ask the model to put more focus on the important features. I dropped the layer with 64 neurons s.t. the no. of parameters of the model should not be bigger. I experimented with different optimizer and I found that Adam in this case was working more efficient compared to SGD.

Model : Convolutional Neural Network

• **Initial Model Architecure** :
Standard Le-Net architecture :
First Conv layer : Conv2d(1, 32, kernel size=(5, 5), stride=(1, 1), padding=(2, 2))
Second Conv layer : Conv2d(32, 64, kernel size=(5, 5), stride=(1, 1), padding=(2, 2))
Fully connected layer: Linear(input features=3136, output features=20, bias=True)
Output Layer: Linear(input features=20, output features=10, bias=True)

Initial Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.0001 and momentum 0.9.
Epochs : 100 epochs (Until validation error started increasing)
Batch Size : 64
Initial Results:
Accuracy :88.82% and Test loss : 0.0009086105575193237



Since the image size is very small as well as it does not contain much complex structure so its better to start with a simple architecture rather than going for a biffar and complex architecture with Lot

parameter.

• **Final Model Architecure** :
First Conv layer : Conv2d(1, 32, kernel size=(3, 3), stride=(1, 1), padding=(1, 1)) Second Conv layer): Conv2d(32, 64, kernel size=(5, 5), stride=(1, 1), padding=(2, 2)) Third : Conv2d(64, 128, kernel size=(7, 7), stride=(1, 1), padding=(3, 3)) Fully connected layer : Linear(input features=6272, output features=100, bias=True) Dropout Layer : Dropout(p=0.25, inplace=False) Output layer : Linear(input features=100, output features=10, bias=True)

Final Hyperparameters :
Loss Function : Cross-Entropy Loss Optimizer : SGD with learning rate 0.01 and momentum 0.9.
Epochs : 300 epochs (Until validation error started increasing)
Batch Size : 2048
Final Results:
Accuracy :92.11% and Test loss : 0.0025724479497558402

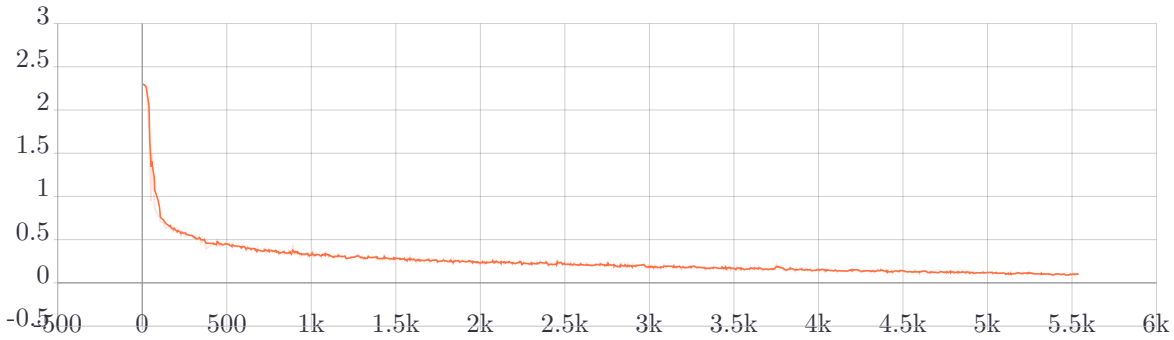
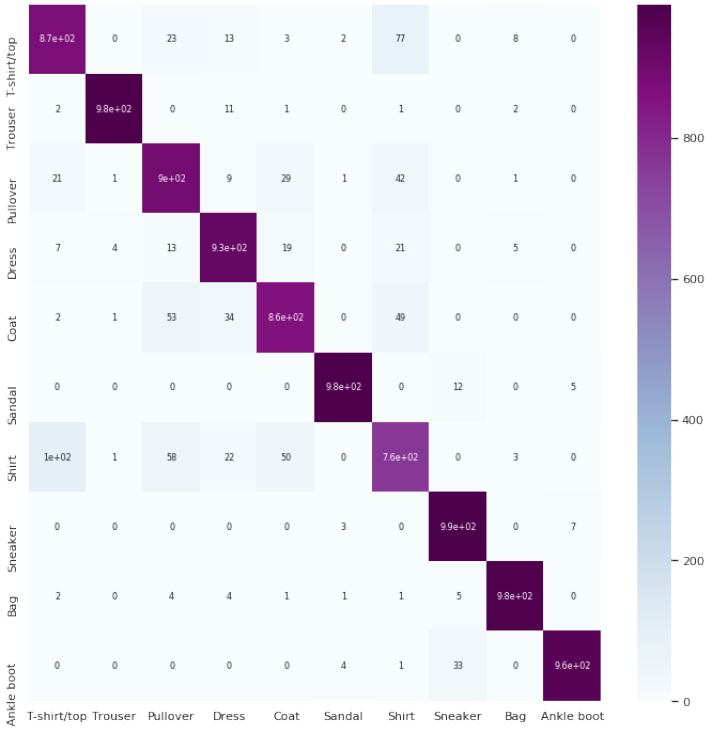


Figure 2: Conv Training Loss



I added a dropout layer after the first layer to add regularization and also to make the model more robust. It will ask the model to put more focus on the important features. I dropped the layer with

64 neurons s.t. the no. of parameters of the model should not be bigger. I experimented with different optimizer and I found that Adam in this case was working more efficient compared to SGD.