Deep Learning Project 2

Rishabh Gupta

MTech(CSA) - 15960

1 The Task

The task was to use deep learning methods on Fashion MNIST dataset. This dataset contains 28x28 gray-scale images of different apparels. There are total 10 classes as follows:

Class	Name
0	T-Shirt
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

Table 1: Apparels in the Fashion MNIST dataset

The task was to train a fully connected neural network and a convolution neural network. And tune the hyper-parameters to get good results.

2 Fully Connected Neural Network

Library used: PyTorch. I trained fully connected neural networks with varying hyper-parameters to see which one gives good accuracy. I changed the number of layers and number of hidden units and other parameters like the learning rate. Following table summarises the different approaches.

Also, I used **data augmentation** by using *RandomHorizontalFlips*. Since the data was clothes like shirts, trousers and coats, flipping them horizontally will still look similar, but will generalize better. Other transforms were not used because the test set contains images without any rotation or any transform. Note that transformations were applied only on the train data and not only the test data.

I started with a simple 2 hidden layer model with 500 hidden units each. The batch size was set to 128. Learning rate was 0.001. All layers used ReLU,

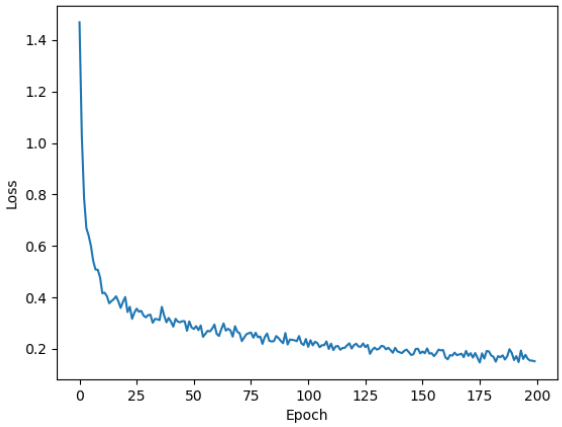
No.	Model	Num Epochs	Train Accuracy	Test Accuracy
1	784-500-500-10	20	89.12%	87.25%
2	784-1024-512-256-128-64-10	20	94.19%	89.28%
3	784-2048-1024-512-256-128-64-10	20	94.01%	89.30%
4	784-2048-1024-512-256-128-64-10	50	98.15%	89.19%
5	784-2048-d-1024-d-512-d-256-d-128-d-64-10	50	93.14%	89.48%
6	784-2048-d-1024-d-512-d-256-d-128-d-64-10	200	97.03%	90.29%

Table 2: Fully Connected NN (Model specifies the number of hidden units, all layers use ReLU except last one which uses softmax (unless stated), d is dropout)

except last one which used softmax. Batch normalisation was also used. Random horizontal flips were on. I got train accuracy of 89% and test accuracy of 87%. I further increased the size of the network, this time 5 hidden layers, starting with 1024 hidden units and decreasing number of units by half in each further layer (Model no. 2). The train accuracy increased to 94%, test accuracy to 89%. To see how adding further layers to the network affects its performance, I added another layer with 2048 units (model no. 3). The accuracies were almost same. But from the plot of loss over time, it looked that the decrease in loss is not saturated. From this observation, I increased the number of epochs to 50 (model no. 4). The train accuracy bumped up to 98%, but test accuracy remain same. Now it looks that the model is overfitting to the train data. To overcome this, I used dropouts in between the layers (model no. 5). The training process was slow so I increased the batch-size upto the point my GPU supports. I found that batch size of 4096 works best for my system. Increasing beyond that gives memory error while training. This allowed me to increase the number of epochs and train the network in a reasonable time. I trained it for 500 epochs. It was able to reach 99% accuracy on train data while 89% on test data. Seems to be overfitting. So I used early stopping and found that around 200 epochs are sufficient for training without overfitting. The final model is model number 6, trained with batch size 4096 for 200 epochs.

2.1 Results

Train Accuracy: 97.03%
Test Accuracy : 90.29%
Training loss vs Epochs plot:



Confusion Matrix:

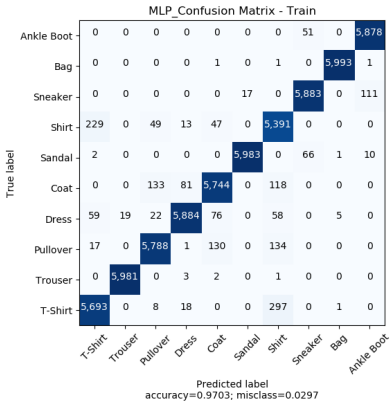


Fig. 1: Train Data

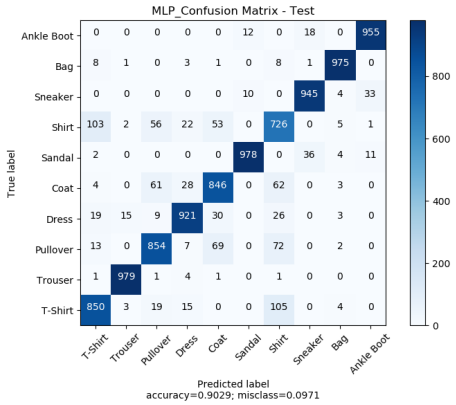


Fig. 2: Test Data

Observations: From the confusion matrix we can observe that the model is mostly confused between shirt - t-shirt, coat - pullover and sneaker - ankle-boot - sandal. There is negligible error between apparels of upper body with apparels of lower body. Only between upper body apparels with other upper body apparels and similarly with lower body and foot apparels. This is intuitive and expected, because these category have similar looking shapes.

3 Convolution Neural Network

Library Used: PyTorch. Convolution neural networks have shown great improvements in computer vision tasks like this. The different models I tried are summarised in table below. I started with a network with 2 convolution layers (kernel

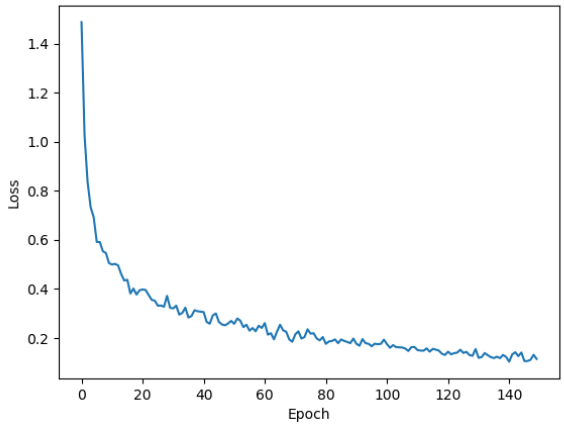
No.	Model	Num Epochs	Train Accuracy	Test Accuracy
1	conv(k=5)-conv(k=5)-64-10	100	96.12%	89.21%
2	conv(k=3)-conv(k=5)-conv(k=7)-256-64-10	200	99.98%	90.64%
3	conv(k=5)-conv(k=5)-conv(k=5)-256-d-64-d-10	150	97.12%	91.62%

Table 3: Convolution NN (Model specifies the number of hidden units, maxpooling is used in convolution layers, all layers use ReLU except last one which uses softmax (unless stated), k is the kernel size, d is dropout layer)

size 5) along with max pooling and then a fully connected layer (model 1). It achieved about 96% accuracy on train data and around 89% on test data in 100 epochs. I further increased the complexity of the model to 3 conv layers with kernel sizes 3, 5 and 7 with max pooling and batch normalization and 2 fully connected layers (model 2). This overfits the train data with 99.98% train accuracy, while test accuracy at 90.64%. To overcome the overfitting problem, I added *RandomCrops* data augmentation. But this doesn't affect much. I added dropouts in the dense layers and changed the kernel sizes to 5 in all 3 convolution layers. The final model is model number 3, trained for 150 epochs.

3.1 Results

Train Accuracy: 97.12%
Test Accuracy : 91.62%
Training loss vs Epochs plot:



Confusion Matrix:

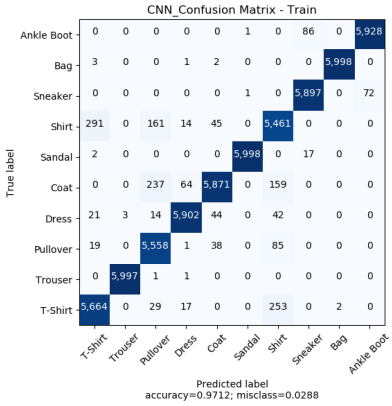


Fig. 3: Train Data

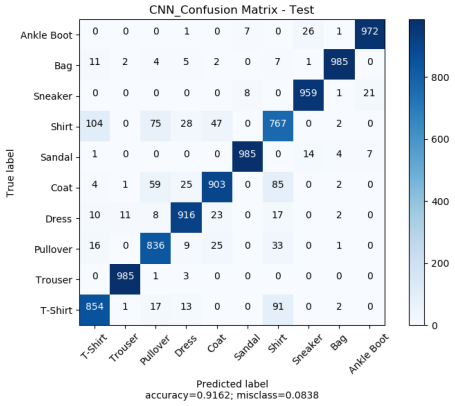


Fig. 4: Test Data

4 Files

There is one *main.py* file which contains the code for testing both the models and generate the expected output files. The files *train_mlp.py* and *train_cnn.py* contains code for training fully connected networks and convolution networks respectively. The *utils.py* file contains some utility functions required by training and testing files. The models are saved in *model* directory.