Neural Networks and Deep Learning

Amir Sepehr Aminian
210453289

Problem Statement

In this assignment, We have to create a classification pipeline for the fashion-MNIST dataset. This dataset has 10 classes and we aim to build a classifier that has more than 90% accuracy on the test set and is general enough to be used.

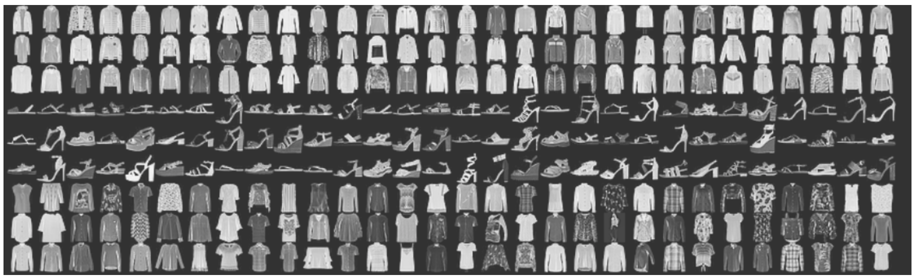
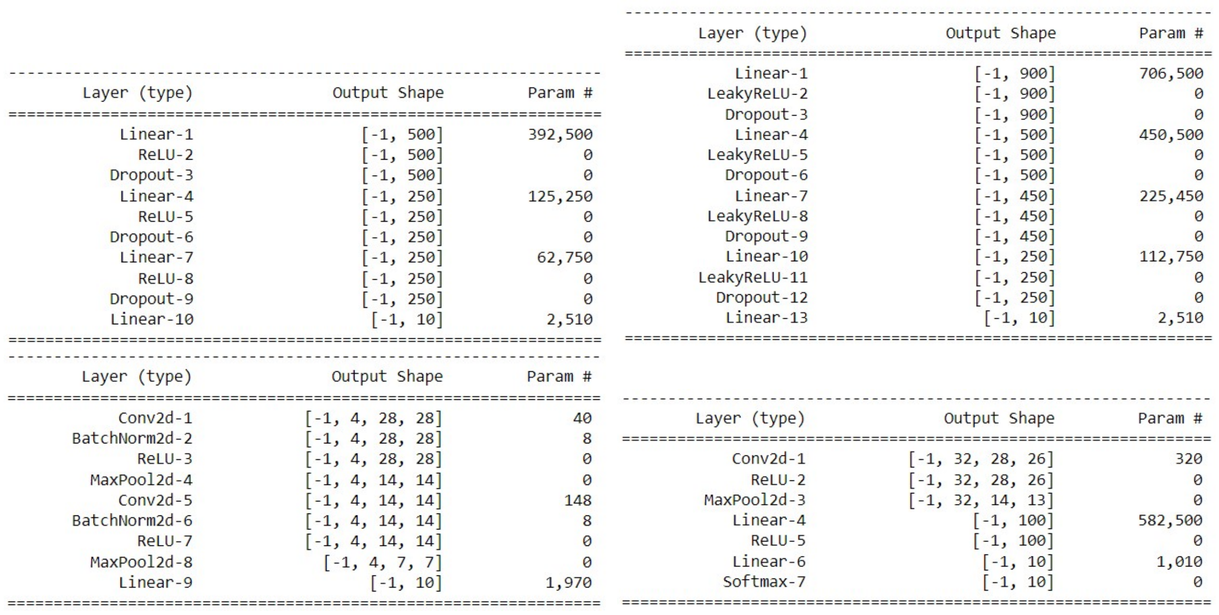


Figure 1: An overview of fashion MNIST dataset

We need a model that can robustly determine what is the label of a previously unseen image. We will be using deep neural networks in the Pytorch environment to develop such a system.

Proposed Solutions

I have developed 5 working models for this dataset, 2 of which is a multilayer perceptron and 3 are convolutional networks combined with Linear layers. Figure 2 shows the models architectures implemented.

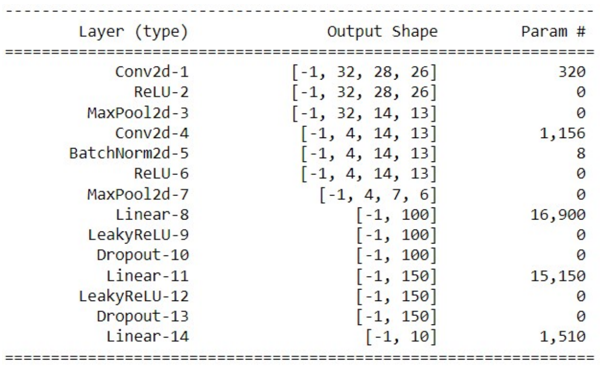


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 500]         392,500
              ReLU-2                  [-1, 500]               0
           Dropout-3                  [-1, 500]               0
            Linear-4                  [-1, 250]         125,250
              ReLU-5                  [-1, 250]               0
           Dropout-6                  [-1, 250]               0
            Linear-7                  [-1, 250]          62,750
              ReLU-8                  [-1, 250]               0
           Dropout-9                  [-1, 250]               0
           Linear-10                   [-1, 10]           2,510
================================================================
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 900]         706,500
         LeakyReLU-2                  [-1, 900]               0
           Dropout-3                  [-1, 900]               0
            Linear-4                  [-1, 500]         450,500
         LeakyReLU-5                  [-1, 500]               0
           Dropout-6                  [-1, 500]               0
            Linear-7                  [-1, 450]         225,450
         LeakyReLU-8                  [-1, 450]               0
           Dropout-9                  [-1, 450]               0
           Linear-10                  [-1, 250]         112,750
        LeakyReLU-11                  [-1, 250]               0
          Dropout-12                  [-1, 250]               0
           Linear-13                   [-1, 10]           2,510
================================================================
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 4, 28, 28]              40
       BatchNorm2d-2            [-1, 4, 28, 28]               8
              ReLU-3            [-1, 4, 28, 28]               0
         MaxPool2d-4            [-1, 4, 14, 14]               0
            Conv2d-5            [-1, 4, 14, 14]             148
       BatchNorm2d-6            [-1, 4, 14, 14]               8
              ReLU-7            [-1, 4, 14, 14]               0
         MaxPool2d-8              [-1, 4, 7, 7]               0
            Linear-9                   [-1, 10]           1,970
================================================================
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 26]             320
              ReLU-2           [-1, 32, 28, 26]               0
         MaxPool2d-3           [-1, 32, 14, 13]               0
            Linear-4                  [-1, 100]         582,500
              ReLU-5                  [-1, 100]               0
            Linear-6                   [-1, 10]           1,010
           Softmax-7                   [-1, 10]               0
================================================================
```



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 26]             320
              ReLU-2           [-1, 32, 28, 26]               0
         MaxPool2d-3           [-1, 32, 14, 13]               0
            Conv2d-4            [-1, 4, 14, 13]           1,156
       BatchNorm2d-5            [-1, 4, 14, 13]               8
              ReLU-6            [-1, 4, 14, 13]               0
         MaxPool2d-7              [-1, 4, 7, 6]               0
            Linear-8                  [-1, 100]          16,900
         LeakyReLU-9                  [-1, 100]               0
          Dropout-10                  [-1, 100]               0
           Linear-11                  [-1, 150]          15,150
        LeakyReLU-12                  [-1, 150]               0
          Dropout-13                  [-1, 150]               0
           Linear-14                   [-1, 10]           1,510
================================================================
```

Figure 2: From left to right and top to bottom models 1 to 5

The first two models are simply linear models with dropouts. Patching the images is implemented but as we discuss in the results section is not beneficial for our purpose. The last three are convolutional models with different combinations of pooling and normalizations. Different optimization algorithms were tried including SGD, ADAM, and RMSProp. In our case, the most sensible loss function is cross Entropy loss. Different configurations for weight initialization methods were tried including the Xavier, He and normal distribution weight initialization. Many hyperparameters such as learning rate, batch size, weight decay and momentum were tested.

There were some unsuccessful attempts as well such as trying VGG11 on this dataset which could be good but there are some issues with my implementation.

Overall, I have reached a point where I concluded my last model which is convolutional is training better with the SGD algorithm and specific custom weight initialization. Patching the input images is not helpful in our case and it leads to worse results compared to not using it.

Results

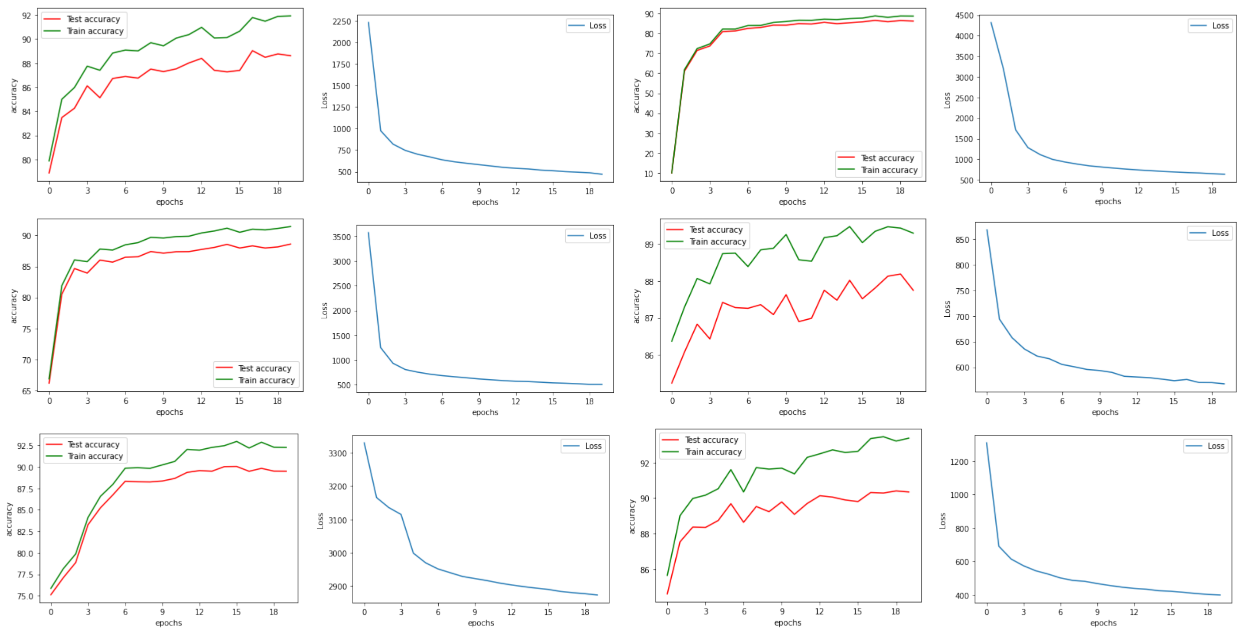


Figure 3: 1 is the train and test accuracy of MLP4 with no patching. 2 is the loss of MLP4 with no patching within 20 epochs. 3 is the train and test accuracy of MLP4 with patching. 4 is the loss of MLP4 with patching within 20 epochs. 5 is train and test accuracy of Custom MLP with no patching. 6 is the loss of custom MLP with no patching within 20 epochs. 7 is the train and test accuracy of CNN A with no patching. 8 is the loss of CNN A with no patching within 20 epochs. 9 is the train and test accuracy of CNN B with no patching. 10 is the loss of CNN B with no patching within 20 epochs. 11 is the train and test accuracy of CNN C with no patching. 12 is the loss of CNN C with no patching within 20 epochs

Model	Train Accuracy	Test Accuracy	Loss	Configuration
MLP4	91.9	88.62	469.58	No Patching
MLP4	88.67	86.18	636.18	Patching
MLP Custom	91.435	86.61	503.31	No Patching
CNN A	89.30	87.75	567.61	No Patching
CNN B	92.26	89.49	2873.43	No Patching
CNN C	93.38	90.34	400.0	No Patching

Table 1: The evaluation of each model and the patching condition

Based on many experiments conducted, I concluded that SGD with momentum is the best possible optimizer for this task. ADAM is also a good option but not useful since we are not dealing with a huge volume of data. The hyperparameters for the SGD depend on the model that is being trained. MLP4 models work well with 0.1 as their learning rate but for other models, we should consider much lower values between 0.01 and 0.17. The momentum is better to be set to 0.9 as it makes the SGD converge faster. The batch size of 32 works well but is a little bit slow while training. I have tried bigger batch sizes but it will decrease the accuracy in general but you can train faster.

By comparing the first two rows of table 1, we can observe that patching the images to smaller parts is not beneficial for this task as the test accuracy of the patched inputs fed into the model is lower than feeding the whole images to the model. These techniques are normally used with images with high resolutions like medical images which have 2000 by 2000 pixels. We can see that the more complex models are getting better accuracy on the test set. This is apparent since CNN C has the best accuracy compared to all other models. The loss is lower than the other models as well.

By checking the graphs at figure and table 1 we can see that the difference between test accuracy and train accuracy is relatively low. We can conclude that the overfitting is prevented in all the models with Dropout layers and batch normalization.

There have been many experiments on different weight initialization methods. These include using normal distribution initialization, Xavier both normal and uniform and He et al. initialization. My conclusion was that for linear layers, it is best to initiate with Normal and for Convolutional layers, it is best to go with He initialization. Therefore, I created a custom initializer to exactly do this based on layer type.

Also, there were attempts to use VGG11 on this dataset but there were some technical issues including size inconsistencies and model complications that can be solved in the future.

Finally, Our main goal is achieved. We received over 90% accuracy on the test set and the model is small enough to give real-time answers. We can conclude that patching with different settings for MLP is not beneficial and having convolutional layers added to the model is useful for this case