|  |
| --- |
| **CSE574 Intro to Machine Learning Project 2** |

**Utkarsh Behre**

Department of Computer Science

University at Buffalo

Buffalo, NY 14214

[ubehre@buffalo.edu](mailto:ubehre@buffalo.edu)

**Abstract**

In this project, we were asked to perform multi-class classification using Neural Networks and convolutional Neural Network on the given Fashion-MNIST clothing images. We were expected to train the model so that it can recognize the image as one of the 10 classes. We were supposed to train a single-layer Neural Network from scratch along with a multi-layer Neural Network and a Convolutional Neural Network using high level library, Keras on the given dataset. After performing these tasks on the given dataset, herein lies my final report.

**1 Introduction**

In the previous project we implemented logistic regression from scratch. In This project we will be implementing a rather more complex variant of logistic regression in a way. This variant is known as a Neural Network. Neural Network is a powerful tool to perform deep learning. One can say that logistic regression is the simplest from of a Neural Network that has no hidden layers and must classify a sample between only two classes. What is a Neural Network? Let’s discuss that next.

**1.1 Neural Network**

A Neural Network, in the simplest definition, would be a network of nodes that could be present in a single, or multiple layers that are connected to input and output layers. Refer below figure as an example Neural Network. We find out the values of these nodes just like we did in logistic regression using the input X features and corresponding weights. The only difference here is that we may have multiple layers of nodes, where for the first layer the input would still be the given input X features, but for the layers after that the input features become the nodes of the previous layers and values are calculated using new set of weights associated with these nodes.

In a Neural Network, we can find the values in these nodes using different activation functions like, sigmoid, Relu, softmax, and so on. Again, we only used sigmoid function in the case of logistic regression. But here, we will be using sigmoid, Relu, and softmax functions to find out different values of nodes at different layers.

**1.2 Convolution Neural Network**

A Convolution Neural network is Neural Network where some preprocessing is done on the given dataset of images in order to find some features that would not be possible if we were to flatten the image into a 1-dimensional array while also reducing the total number of free parameters, allowing the network to be deeper with fewer parameters. The usual input for a convolutional layer is of shape (number of images) x (image width) x ( image height) x (image depth). The preprocessing is done with convolutional kernels, whose width and height are hyperparameters. Based on number of filters we get a set of images that highlight different possible features of an image. Pooling can also be applied to the images, which basically reduces the dimensions of image while keeping the essence of the image. Two of the major pooling techniques are max-pooling, mean-pooling. Although in our project, we will be just using the max-pooling technique.

**2 Dataset**

For training and testing our classifiers in this project, we are provided with Fashion-MNIST dataset. The dataset consists of the images from Zalando’s article. All the images are 28x28 pixel sized grayscale images. We have 60,000 images as part of our training set, and 10,000 images as part of our testing set. A pixel value is given between 0 to 255 where higher value means a darker pixel. Each image has 784 pixels available, and a class label value that tells us which class of clothing that image belongs to. A pixel value can be between 0 and 255. There are 10 possible classes to which all the images in the training and testing sets can belong to. These classes are t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot.

**3 Preprocessing**

The images are given to us in the form of their pixel values. For the 1st , and 2nd tasks as per the requirement, we need to flatten the 2-dimensional image 28x28 pixels to a 1-dimensional array of 784 pixels. While for the third task, we need the images in their 28x28 size pixel format initially where we will eventually be flattening the pixels into a 1-dimensional array to run into our Neural Network. For all the tasks, we will have the images divides into sets of training, and testing data. We will however use 10% of the training set as our validation set for the purpose of moderating our training process. In addition to above, for the 3rd task, we will be applying convolution layers with pooling to the images before putting them through the hidden layers. All the weights and biases are randomly initialized. The testing set will be used at the end when we have finalized our model to test our model on the untouched testing set.

**4 Architecture**

The code for all the 3 tasks required different approaches. In the 1st task we had to design everything from scratch, while for the 2nd and 3rd tasks we were to use Keras, high level Neural Network library.

**4.1 Single-layer Neural Network**

A single-layer neural network as the name suggests consists of a single layer of the hidden nodes that take the pixel values as the input, uses sigmoid function as their activation function and outputs 10 different nodes for the 10 different classes using a softmax function. In the code, I’ve done the above, where I tried different number of hidden nodes as hyper parameters along with other hyperparameters. The different equations used for finding z1, z2, a1, a2, weights and loss are as given below in figure 1.

**Figure 1: Equations used in single-layer Neural Network**

**4.2 Multi-layer Neural Network**

A multi-layer neural network consists of more than 1 hidden layers with hidden nodes that take the pixel values as the input and outputs to 10 different nodes for the 10 different classes using softmax function. For the hidden layers, we have the option to use different or same activation functions. In the code, I’ve tried adding 2 and 3 dense layers from Keras using a sequential model with different hidden nodes for the layers to see which one the best design for the model would be.

**4.3 Convolutional Neural Network**

A convolutional neural network takes the 2-dimensional images as inputs, then it can have multiple convolutions and pooling layers that can use kernels based on hyperparameters, then once we flatten the pixel values we feed it to a multi-layer neural network which works as mentioned above in the section 4.2. In the code, I’ve tried single, and double convolutional layers with max-pooling applied on the 2-dimensional images at each layer. Then I flattened the images and added dense layers just like for the 2nd task.

**4.4 Hyperparameters**

For the tasks, I had to tune several hyperparameters. These hyperparameters were number of hidden layers, number of hidden nodes in a layer, learning rate, epochs, choice of activation functions to be applied to the hidden layers, loss function, optimizer, kernel size, convolutional layers. I did try to run gradient descent with different batch sizes initially but decided to divide the entire training set in 10 minibatches as it felt like a good enough number of batches to run per epoch. How I tuned these parameters in order to get good models and the results that I had got from them is what I have described in the rest of the report below.

**5 Experimentation**

Since all 3 tasks in this project were sort of independent, I am going to describe what I did for them in their specific sections below. I have however kept a similar pattern of modularizing the code in all the tasks so that the functions can be reused in other tasks and it’s easier to understand.

**5.1 Train using Neural Network with One Hidden Layer**

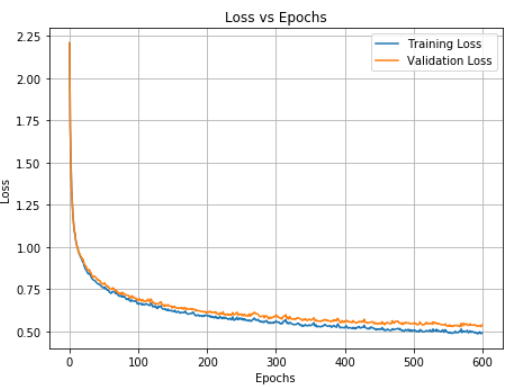
First, I have defined the functions that are needed to train and test the model like sigmoid, softmax, cost, transform\_labels, predict\_output, calc\_accuracy, deep\_learning and other graph plotting functions. The function transform\_labels transforms the int array that represents the class labels to a 2-dimensional binary array where for each sample there will be 0s in 9 classes and 1 in the class that sample belongs to. The function deep\_learning can be called to train the model with several hyperparameters.

I tried different settings for the parameters to see the performance model while train in the training and validation dataset. First I tried with by running gradient descent on the 54000 samples of the training set while validating using the 6000 samples from training set. But that took too much time, so I used mini batches of 5400 to run gradient descent on, and ran predict\_output, cost and calc accuracy to store the changes every epoch. Initially I tried with 0.001 learning rate which turned out to be too high as accuracy was not improving. So, I used 0.0001 learning rate and I could see the improvement in accuracy and loss going down. Now all I had to do was tweak other parameters, mainly number of hidden units and epochs so that I get a good model.

I tried many combinations and was not able to get more than 80% accuracy, so I increased epochs to ridiculous 10,000 epochs and saw that it was slightly overfitting the training set because training accuracy did go up to 89.09% but validation accuracy stopped at around 81.95%. After tweaking the parameters, a little more and making sure validation loss doesn’t rise again during training I arrived at the final set of parameters that can be seen in the last row of below table and in the results as well. The final loss and accuracy graphs can be seen in the Figure 2 after below table.

**Table 1. Tuning Hyperparameters vs Accuracy for the task 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Batchsize** | **Learning Rate** | **Epochs** | **Units** | **Val\_Accuracy** |
| 5400 | 0.001 | 1000 | 5 | 54.98% |
| 5400 | 0.001 | 500 | 20 | 10.15% |
| 5400 | 0.0001 | 500 | 20 | 79.46% |
| 5400 | 0.0001 | 500 | 30 | 79.55% |
| 5400 | 0.0003 | 500 | 50 | 77.65% |
| 5400 | 0.0001 | 500 | 100 | 81.52% |
| 5400 | 0.0001 | 10000 | 35 | 81.15% |
| 5400 | 0.0001 | 500 | 40 | 79.2 |
| 5400 | 0.0001 | 600 | 40 | 81.33 |



**Figure 2: Training vs Validation loss graph for task 1**

**5.2 Train using Multi-Layer Neural Network**

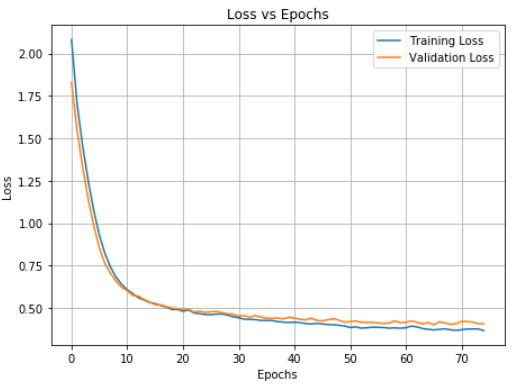
First, I imported the high-level library Keras that is going to be used for this task, specifically Sequential model and Dense layer. Now using the Sequential model, I made a model with 4 dense layers (1 for output, and 3 hidden layers). The first one takes input from pixels while others take input from layer before them. The hidden layers use combination of sigmoid and Relu as activation functions. However, the last layer always uses softmax function to generate probabilities of a sample to belong in all 10 classes. The class with highest probability is taken as the model prediction for that sample.

Model is compiled using “categorical\_crossentropy” loss function and optimizer that I used initially was “adam”. Later I tried SGD as well as the optimizer, but it seemed like adam gave better performance. Then we use model.fit() function to train the model with hyperparameters being number of epochs, combination of activation functions, number of layers, number of hidden units in a layer, loss function, and the optimizer.

The modeling was straight forward and once this was done, I immediately started getting good gradient descents. The model was learning faster using the “adam” as the optimizer, compared to “sgd” (stochastic gradient descent). After this I tweaked the hyperparameters mentioned above and got different results as shown in the table below with the last row being my final choice of parameters. Few of the findings have not been shown in the table as there were many hit and trials with different layer count and activation functions. But the ones that lead me to the final settings are given in the table and the result is also included in the results section.

**Table 2. Tuning Hyperparameters vs Accuracy for the task 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No. of Hidden Layers** | **No. of Hidden Units** | **Activation Functions (for hidden layers only)** | **Optimizer** | **Epochs** | **Validation Accuracy** |
| 3 | 100, 100, 100 | Sigmoid, Relu, Sigmoid | Adam | 100 | 85.77% |
| 3 | 50, 50, 50 | Sigmoid, Relu, Sigmoid | Adam | 100 | 85.28% |
| 3 | 35, 35, 35 | Relu, Relu, Relu | Adam | 100 | 83.35% |
| 3 | 40, 40, 40 | Sigmoid, Relu, Sigmoid | SGD | 100 | 64.63% |
| 2 | 40, 40 | Sigmoid, Sigmoid | Adam | 100 | 84.88% |
| 2 | 50, 50 | Sigmoid, Sigmoid | Adam | 100 | 85.17% |
| 2 | 50, 50 | Relu, Relu | Adam | 100 | 80.6% |
| 2 | 50, 50 | sigmoid, Relu | Adam | 100 | 85.53% |
| 2 | 50, 50 | Sigmoid, Relu | Adam | 1000 | 86.12% |
| 2 | 50, 50 | Sigmoid, Relu | Adam | 200 | 85.72% |
| 2 | 40,40 | Sigmoid, Relu | Adam | 120 | 85.6% |
| 2 | 40,40 | Sigmoid, Relu | Adam | 75 | 85.68 |



**Figure 3: Training vs Validation loss graph for task 2**

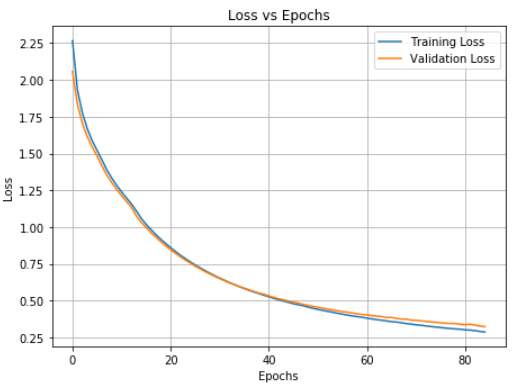
**5.3 Train using Convolution Neural Network**

The third task was to implement convolution neural network which was to add convolution layers on the images before flattening the pixels and running it through the layer of nodes like in task 2. So first, I wrote another function “load\_mnist\_for\_CNN” similar to “load\_mnist” which would return the images in 28x28 format instead of 784 pixels array format. I then used this function to get the training and testing sets from the dataset. While making the model, the only thing that was extra from the 2nd task was that we had to add convolution layers into the model first along with pooling. So, I added 2 convolution layers with max-pooling to the model and then flattened the output pixels before putting it through next series of layers.

The remaining process was same as task 2 with extra parameters being kernel dimensions, number of convolution layers and activation functions to use for them. Since I had done 2nd task by this point I just had to tweak convolution layers mainly, with only a slight tweaking in the rest of the dense layers. My findings for the different tweaks are in below table with the final setting mentioned in the last row and in the results section.

**Table 3. Tuning Hyperparameters vs Accuracy for the task 3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Conv Layers** | **No. of Kernels** | **Kernel sizes** | **Kernel Activation functions** | **No. of Hidden Layers** | **No. of Hidden Units** | **Epochs** | **Validation Accuracy** |
| 2 | 8,8 | 2x2,2x2 | Relu, Relu | 2 | 40,40 | 75 | 85.79% |
| 2 | 8,8 | 2x2,2x2 | Relu, Relu | 2 | 40,40 | 20 | 82.26% |
| 2 | 32,32 | 3x3, 2x2 | Sigmoid, Relu | 2 | 40,40 | 150 | 90.72 |
| 2 | 32,32 | 3x3, 2x2 | Sigmoid, Relu | 2 | 40,40 | 75 | 90.92 |
| 1 | 32 | 3x3 | sigmoid | 1 | 30 | 75 | 90.87 |
| 2 | 64,64 | 3x3, 2x2 | Sigmoid, Relu | 2 | 30,30 | 25 | 90.47 |
| 1 | 32 | 3x3 | Sigmoid | 1 | 30 | 85 | 90.95 |



**Figure 4: Training vs Validation loss graph for task 3**

**5.4 Observations between different models**

The model designed in the first task only had a single hidden layer in the network. The only option was to increase the number of nodes for that layer. Initially increasing the nodes did improve the accuracy of the system. But at about 30-35 nodes and above it felt like the final accuracy was just not improving over 78% without overfitting the model. The limitation of having only 1 hidden layer was glaringly obvious. So, when I added 2 more hidden layers, the accuracy improved further to 85%. This was much better compared to the last one but again did not improve much no matter how many extra nodes were added in these hidden layers. Finally, in the third task, after adding convolution layers in the model the accuracy of the model improved even further to about 89%. This showed that handling images is better if convolution layers are used.

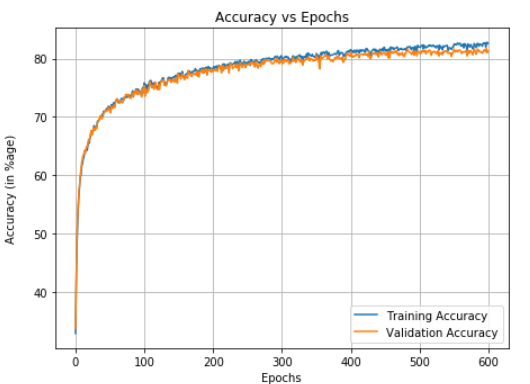
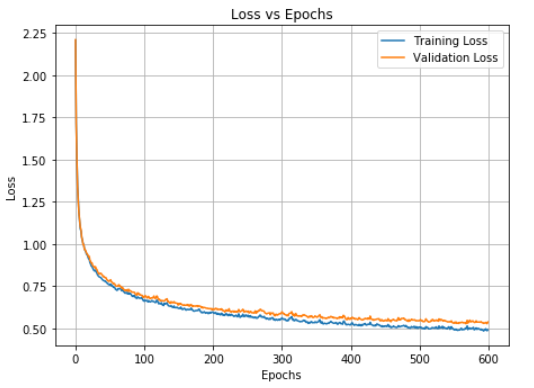
It is going to be based on complexity of the dataset that we must help us decide what kind of model should we be using and will differ case by case. For example, if we have a set of images that are clearly identifiable like say horizontal lines and vertical lines, we may be just fine with using a single layer Neural Network. But if further complexity gets involved where cases start overlapping like an image of a t-shirt and a shirt then it would be better to use convolution layers to help identify more features of the images. However, we could still use a simple multi-layer Neural Network for other simpler use-cases where there may not be that much overlapping in between the dataset features. Ultimately, It’s going to take some form of hit and trial before we understand what model will be best to choose for some given dataset.

**6 Results**

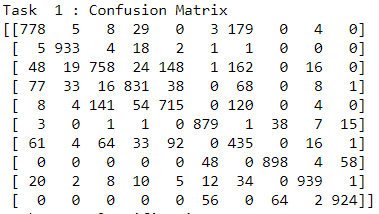
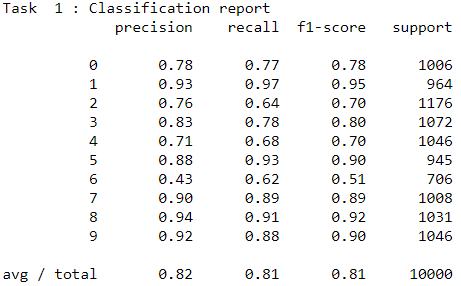
The final setting of parameters along with the accuracy of the system on the unseen test data is given in below tables 4, 5, and 6 for all three tasks. The final training and validation loss, and training and validation accuracy graphs that we got for all three tasks are shown in the below figures 5, 7, and 9. The confusion matrix and the classification report were generated using the sklearns.metrics library. These are shown in the figures 6, 8, 10 for the three tasks.

**Table 4: Final parameter values with accuracy on test set for Task 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Batchsize** | **Learning Rate** | **Epochs** | **Units** | **Accuracy** |
| 5400 | 0.0001 | 600 | 40 | 80.9% |



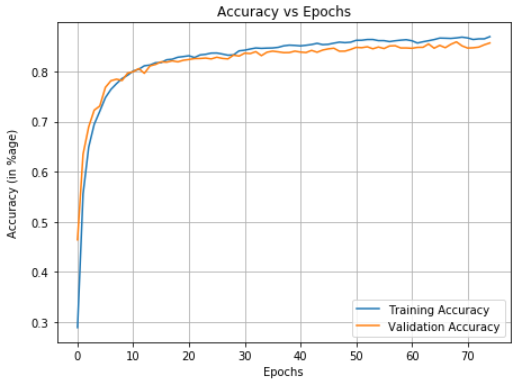
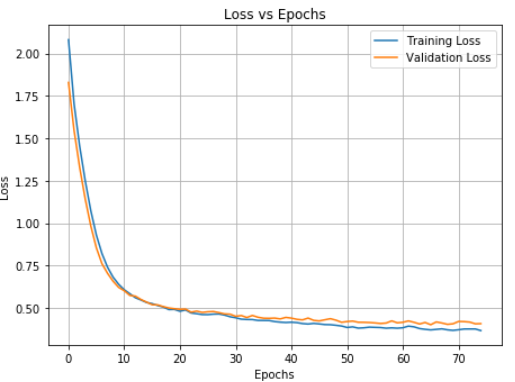
**Figure 5: Training and Validation Loss and Accuracy Vs Epochs Graph for Task 1**

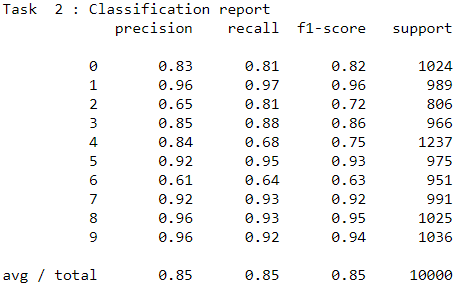
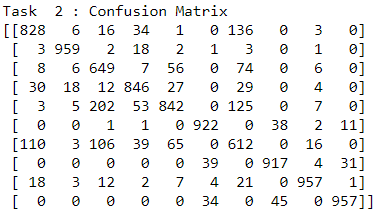
**Figure 6: Confusion Matrix and Classification Report for Task 1**

**Table 5: Final parameter values with accuracy on test set for Task 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No. of Hidden Layers** | **No. of Hidden Units** | **Activation Functions (for hidden layers only)** | **Optimizer** | **Epochs** | **Accuracy** |
| 2 | 40,40 | Sigmoid, Relu | Adam | 75 | 85.68% |



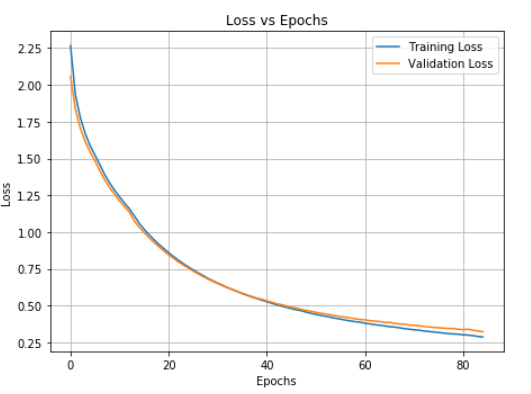
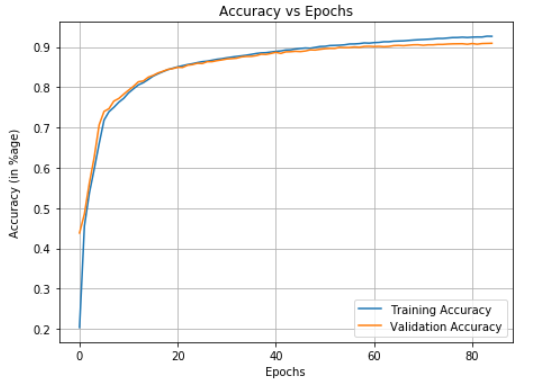
**Figure 7: Training and Validation Loss and Accuracy Vs Epochs Graph for Task 2**



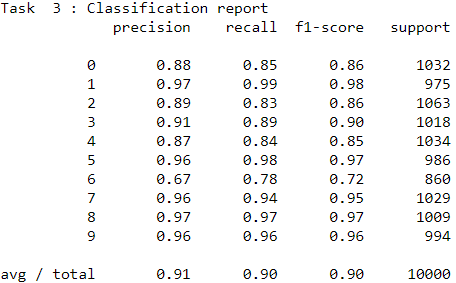
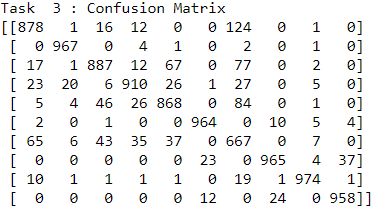
**Figure 8: Confusion Matrix and Classification Report for Task 2**

**Table 6: Final parameter values with accuracy on test set for Task 3**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Conv Layers** | **No. of Kernels** | **Kernel sizes** | **Kernel Activation functions** | **No. of Hidden Layers** | **No. of Hidden Units** | **Epochs** | **Validation Accuracy** |
| 1 | 32 | 3x3 | Sigmoid | 1(Sigmoid) | 30 | 85 | 90.38% |

**Figure 9: Training and Validation Loss and Accuracy Vs Epochs Graph for Task 3**



**Figure 10: Confusion Matrix and Classification Report for Task 3**

**7 Conclusion**

I was able to design the neural network from scratch and by using Keras for predicting the class of clothing an image, from Fashion-MNIST dataset, belongs to with a descent accuracy after tweaking the hyperparameters involved in all the tasks. I got to a final set of values of the hyperparameters that made our classifiers good enough for recognize an image from an unseen dataset and classify it as one of the 10 classes.

**Acknowledgments**

I am grateful to Professor Sargur Srihari for helping me understand the concepts of Neural Networks and its application. I also thank Mihir Chauhan for taking efforts in providing clarifications and better understanding of implementation of logistic regression in python.

**References**

[1] https://stackoverflow.com/questions/29576430/shuffle-dataframe-rows

[2]https://ublearns.buffalo.edu/bbcswebdav/pid-5107775-dt-content-rid-25436584\_1/courses/2199\_23170\_COMB/1.1.2%20Python%2BMLFrameworks.pdf

[3] https://matplotlib.org/3.1.1/api/\_as\_gen/matplotlib.pyplot.subplots.html

[4] https://towardsdatascience.com/accuracy-paradox-897a69e2dd9b

[5] https://pythonspot.com/matplotlib-bar-chart/

[6] https://codeyarns.com/2014/10/27/how-to-change-size-of-matplotlib-plot/

[7] https://www.researchgate.net/publication/303326261\_Machine\_Learning\_Project

[8] My project report for project 1.

[9] https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.astype.html

[10]https://d1b10bmlvqabco.cloudfront.net/attach/jzsomflkppr3r4/isamd3soc56z/k1ghfbf2snxm/Equations2.pdf

[11]https://stackoverflow.com/questions/50920908/get-confusion-matrix-from-a-keras-multiclass-model

[12] https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

[13] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion\_matrix.html

[14] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification\_report.html

[15] https://gist.github.com/RyanAkilos/3808c17f79e77c4117de35aa68447045