COMP 551 Mini Project 3: Classification of Image Data

Dian Basit
260771254
dian.basit@mail.mcgill.ca

Hadi Zia
260775855
syed.h.rizvi@mail.mcgill.ca

Sagar Nandeshwar
260920948
sagar.nandeshwar@mail.mcgill.ca

Abstract—**This project explores the performance of a machine learning algorithm: Multilayer Perceptron(MLP). This model was created from scratch and was used in an image classification task. The dataset that has been used in training the model is; Fashion-MNIST, which is a dataset of Zalando's article images. We looked at how the Multilayer Perceptron behaved when some of the parameters were changed, such as the number of layers, units, and activation type. We present a thorough analysis of our findings and draw some conclusions.**

I. Introduction

In this project we implemented MLP from scratch and evaluated the performance of this model. We created an MLP with no hidden layers, an MLP with a single hidden layer having 128 units and ReLU activations, and an MLP with 2 hidden layers each having 128 units with ReLU activations, each with a softmax layer at the end. We compared the test accuracy of these three models on the Fashion-MNIST dataset. We also created a CNN using preexisting libraries and trained it on the Fashion-MNIST dataset, after which we compared the accuracy of both models.

A. Multilayer Perceptron

The Multilayer Perceptron is a more complex variant of the Perceptron algorithm in which layers of neurons are stacked each performing the Perceptron algorithm. The input layer receives data and does weighted sum for each input neurons followed by an activation function for every neurons in the hidden layer. After passing through all of the hidden layers, the data reaches the output layer, which has exactly one neuron for each class and outputs a probability score indicating how likely the data is to belong to that class. Our classification selection is then made based on the class with the highest score.

B. Convolutional Neural Network

The Convolutional Neural Network (CNN) is one example of a deep learning algorithm. They are very useful in image classification. CNN is widely used, from photo tagging to self-driving cars. CNN's architecture is similar to how neurons in our brain function. Convolutional layers apply the convolution operation to the input image before passing it on to the next layers. The vector is then sent through a pooling layer before being fed into a fully connected neural network.

II. Datasets

A. Fashion MNIST

The Fashion MNIST Dataset is an MNIST-like dataset of 70,000 28x28 labeled fashion images. It is divided into two subsets: one for training(60,000) and the other one for testing(10,000). Each row in the dataset is a separate image. The first column is the class label whereas the remaining columns are pixel numbers (784 total). Each value is the darkness of the pixel (1 to 255)[1].

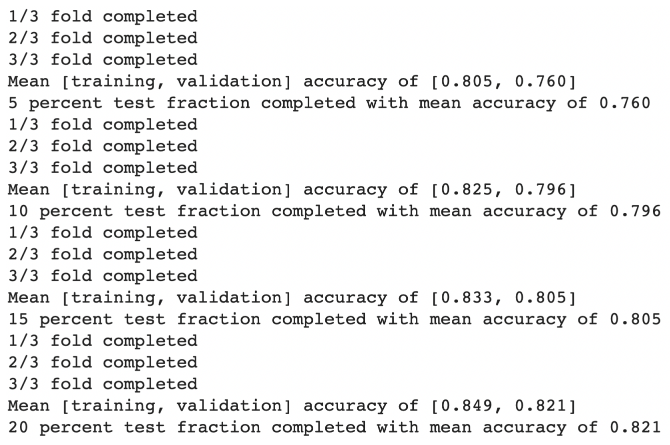
III. Implementation

Multilayer Perceptron was implemented from scratch using the Numpy library and code samples from lecture slides. CNN was implemented using TensorFlow. The implementation can be found in the attached jupyter notebook.

IV. Results

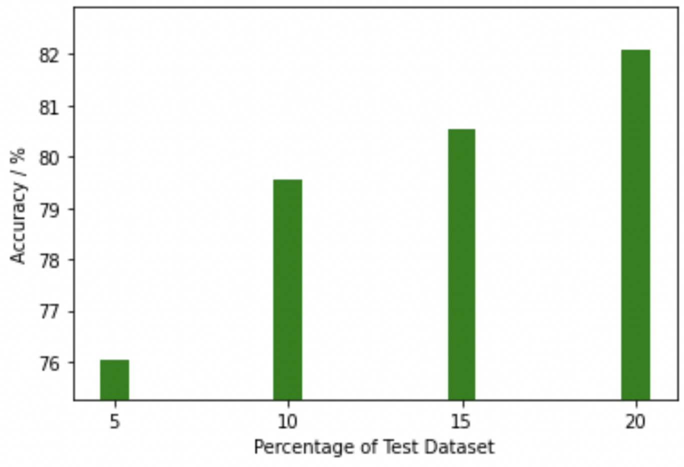
To begin with, we normalized pixel values to a domain of 0 and 1, i.e., [0,255] to [0,1]. It is important to reduce the overflow in the exponent operation.

For selecting the optimal training dataset for optimal performance on test set, we kept three things constat; we used k=3 in KFoldCV for cross-validation testing, we used a batch_size of 50 for Mini-Batch Gradient Descent and we used 1 hidden layer in our MLP. The performance metric was based on the accuracy on validation set. We found that a 20% of training dataset gave the best performance. The figures below summarize our findings.



```
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.805, 0.760]
5 percent test fraction completed with mean accuracy of 0.760
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.825, 0.796]
10 percent test fraction completed with mean accuracy of 0.796
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.833, 0.805]
15 percent test fraction completed with mean accuracy of 0.805
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.849, 0.821]
20 percent test fraction completed with mean accuracy of 0.821
```

Fig. 1. Test Fraction and mean accuracy

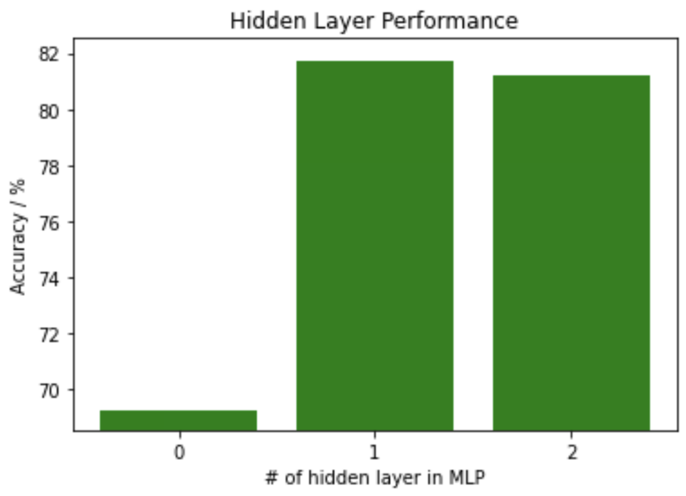


```
0.20 fraction has the best performance
```

Fig. 2. Percentage of Test Data Set vs Accuracy

We then compared the test accuracy of our MLP using 0,1 and 2 hidden layers. The constants kept for these tests are as follows: Epsilon of 0.01, learning rate of 0.005, epoch of 200, mini-batch gradient descent of 50, the activation function was set to ReLU and the number of hidden Layer nodes were 128. MLP with 0 hidden layers completed with an accuracy of 0.693, the MLP with 1 hidden layer completed with an accuracy of 0.817 and the MLP with 2 hidden layers completed with an accuracy of 0.812. The figure on the right shows how the accuracy of the model compares with different hidden layers.

Our decision function can make more complex decisions the more 'non-linear' it is. Having more neurons in the layers with ReLU, a non-linear activation function, means that the output of the network should have a non-linear relationship with the input. Having non-linearity is

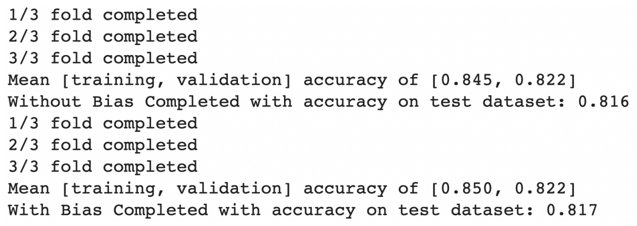


```
Best hidden layer of MLP is: 1
```

Fig. 3. No. of hidden layers vs Accuracy

important because it allows the subsequent layers to build off each other. A non-linear activation function allows the stacking of multiple layers of neurons to create a deep neural network, which is required to learn complex data sets with high accuracy.

We then compare the accuracy of MLP with bias and with non-bias weights. The model with bias weight gives the accuracy of 0.817 which is slightly better than that with bias weight 0.816. During our testing, we have, often, found that accuracy of one model (with or without bias) is better than than other.

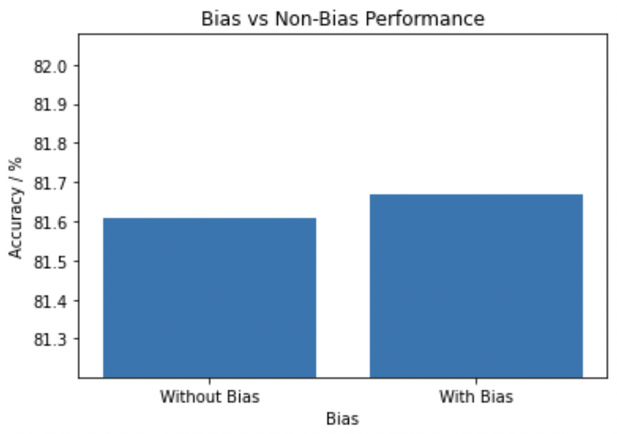


```
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.845, 0.822]
Without Bias Completed with accuracy on test dataset: 0.816
1/3 fold completed
2/3 fold completed
3/3 fold completed
Mean [training, validation] accuracy of [0.850, 0.822]
With Bias Completed with accuracy on test dataset: 0.817
```

Fig. 4. Bias and Non-bias vs Accuracy

It was surprising to see that bias was slightly better (0.1%) than non-bias. It is important to give bias because it gives better accuracy and provide a better fit to the training and test dataset by shifting the function vertically to reduce mis-classification.

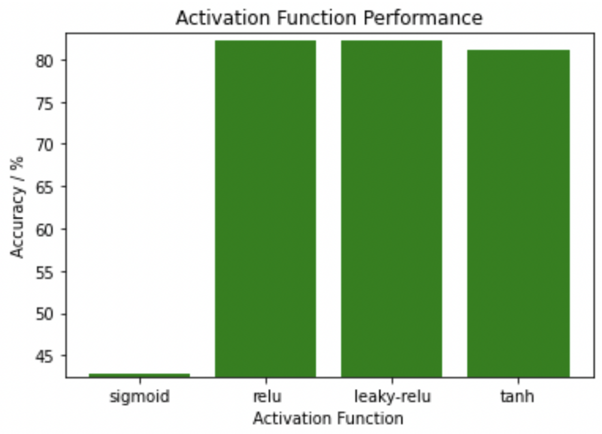
We then tested our model with different activation functions, namely; Sigmoid, Relu, Leaky Relu and tanh, and compared their test accuracy. The constants kept in this experiment were: 128 Hidden layer neurons, 2 hidden layers , epochs of 200, epsilon and learning rate of 0.005 and a batch size of 50 for mini-batch GD and



```
Bias should be selected?  True
```

Fig. 5. Bias and Non-bias vs Accuracy

set bias as True (based on past result). Through the tests conducted, we found that leaky-relu is the best activation function which gives us an accuracy of around 0.823. However, we have found out that with various test, the best activation function seems to weigh between tanh and leaky-relu. However for this test, we select leaky-relu as the best activation. The figure below shows how the accuracy varies using different activation functions.



```
Best activation function is leaky-relu
```

Fig. 6. Activation Function vs Accuracy

```
sigmoid activation completed with accuracy of 0.429
relu activation completed with accuracy of 0.822
leaky-relu activation completed with accuracy of 0.823
tanh activation completed with accuracy of 0.810
```

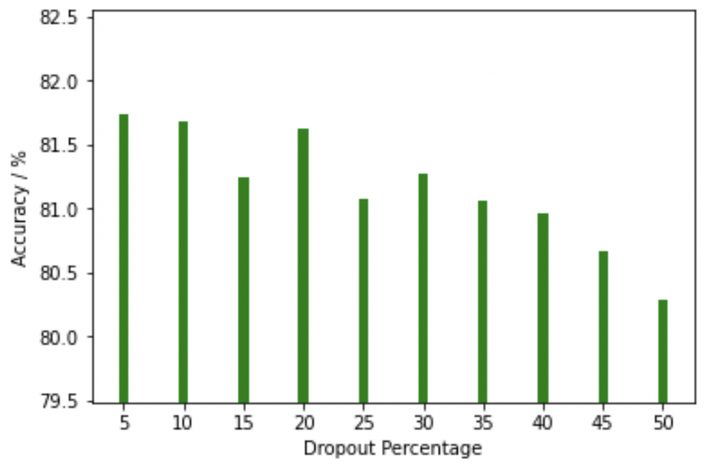
Fig. 7. Activation Function vs Accuracy

We expected tanh to give the best result because it is an odd function and give negative weightage of -1 to misclassified input and give positive weightage of 1 to classified data. However, the results are extremely close i.e. tanh (81%) and leaky-relu (82.3%) with a difference of 1.3%. One possible reason would be because if data inputs are positive then it is scaled linearly giving more weightage to correctly classified data as it goes over 1.0, compared to tanh where the data is limited to 1.0. It is also important to note that leaky-relu is better than relu function because, leaky-relu gives negative weightage to misclassified data while relu brings it down to 0.

We then added dropout regularization to the network and trained the MLP. The model has tunable hyper-parameter i.e. dropout_percent in the domain of 5-50% with 5% interval. The Constants kept for this experiment were: 128 Hidden layer neurons, 2 hidden layers , 1000 epochs, epsilon and learning rate of 0.01, the ReLU activation function and a batch size of 50 for mini-batch GD and we enabled bias. We concluded that the best dropout percentage was 5% with an accuracy of 0.8117. The figure below shows how the accuracy compares over the hyper-parameter domain.

```
5.0 dropout percentage completed with accuracy of 0.817
10.0 dropout percentage completed with accuracy of 0.817
15.0 dropout percentage completed with accuracy of 0.812
20.0 dropout percentage completed with accuracy of 0.816
25.0 dropout percentage completed with accuracy of 0.811
30.0 dropout percentage completed with accuracy of 0.813
35.0 dropout percentage completed with accuracy of 0.811
40.0 dropout percentage completed with accuracy of 0.810
45.0 dropout percentage completed with accuracy of 0.807
50.0 dropout percentage completed with accuracy of 0.803
```

Fig. 8. Dropout percentage vs Accuracy

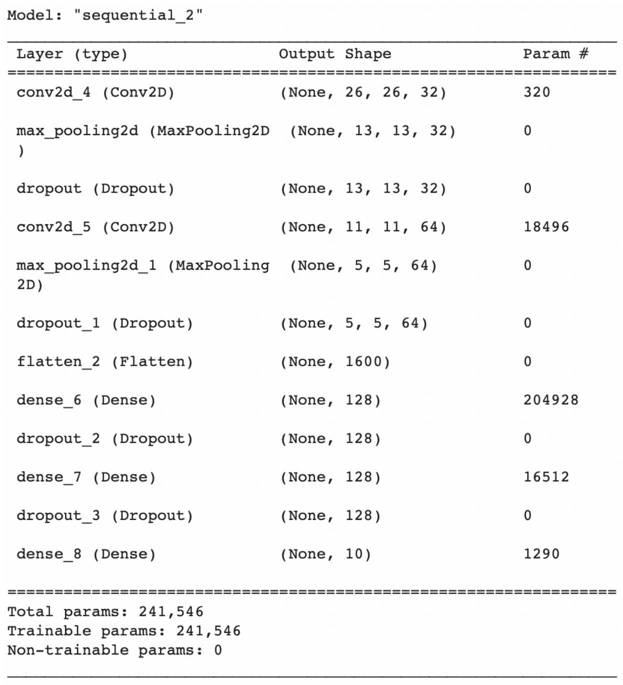


```
Best dropout percentage is 5
```

Fig. 9. Dropout percentage vs Accuracy

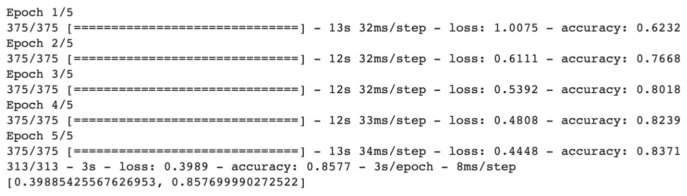
Dropout regularization helps reduce the overfitting of training data and gives more generalization to the model for accurate prediction on unseen data. We were satisfied with low 5-10% of dropout. Since we selected 5% of dropout means, model shuts down 6 out of 128 hidden nodes.

We have tested our model's accuracy against Convolutional neural network that is trained on the same data-set. CNN model on 2D Image data-sets is easily prone to over-fitting, therefore when we just trained the data with just 2 Convolutional layer, with filter numbers size of 32 and 64, and two fully connect layers, we get accuracies of of 0.87, and 0.91 and 0.93 with the kernel size (1,1), (2,2) and (3,3) respectively. When we implement the CNN model with, slightly adjusted, pooling and dropout,it gives an the accuracy 0.85, which is slightly better than that of MLP, as expected.



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_5 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0

 flatten_2 (Flatten)         (None, 1600)              0

 dense_6 (Dense)             (None, 128)               204928

 dropout_2 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 128)               16512

 dropout_3 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 10)                1290

=================================================================
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
_________________________________________________________________
```

Fig. 10. CNN layers



```
Epoch 1/5
375/375 [==============================] - 13s 32ms/step - loss: 1.0075 - accuracy: 0.6232
Epoch 2/5
375/375 [==============================] - 12s 32ms/step - loss: 0.6111 - accuracy: 0.7668
Epoch 3/5
375/375 [==============================] - 12s 32ms/step - loss: 0.5392 - accuracy: 0.8018
Epoch 4/5
375/375 [==============================] - 12s 33ms/step - loss: 0.4808 - accuracy: 0.8239
Epoch 5/5
375/375 [==============================] - 13s 34ms/step - loss: 0.4448 - accuracy: 0.8371
313/313 - 3s - loss: 0.3989 - accuracy: 0.8577 - 3s/epoch - 8ms/step
[0.39885425567626953, 0.857699990272522]
```

Fig. 11. CNN result

We tested our model on un-normalized image using different activation function. The accuracies between normalized and unnormalized dataset varies across the activation functions. For relu and leaky-relu, normalized data-set performed significantly better than unnormalized images (very consistent on various test-runs). For tanh, both datasets gives comparable (almost same) accuracies where else, for sigmoid, unnormalized dataset gives better accuracy than normalized data set. We have also found out that unnormalized images took significantly more time to compute as compares to normalized images.

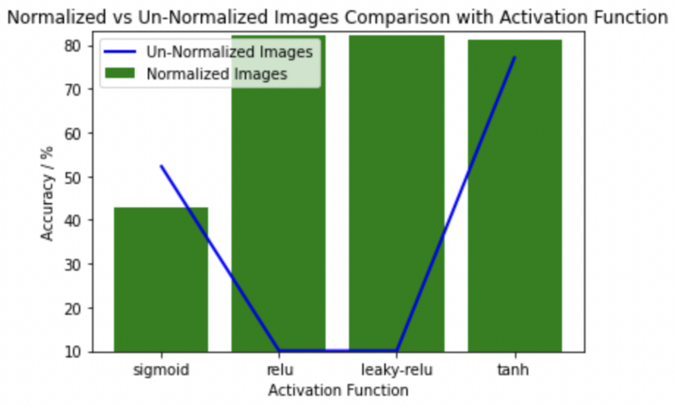
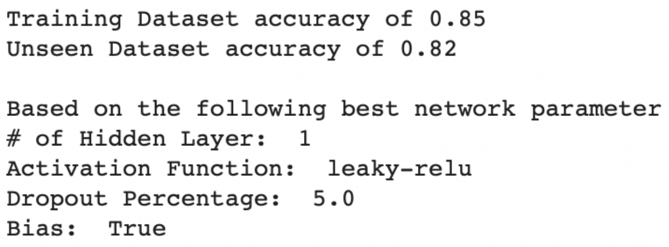


Fig. 12. Unnormalized and normalized image vs accuracy

Based off of our experiments, the best resulting MLP was trained using 20% Training Dataset, leaky-relu Activation function, a Dropout percentage of 5% and Hidden layer of 1. The constants kept were: 128 Hidden layer neurons, 200 epochs, epsilon and learning rate of 0.005 and a batch size of 50 for mini-batch GD. With these optimal parameters, we got a training Dataset accuracy of 0.85 and an Unseen Dataset accuracy of 0.82.



```
Training Dataset accuracy of 0.85
Unseen Dataset accuracy of 0.82

Based on the following best network parameter
# of Hidden Layer:  1
Activation Function:  leaky-relu
Dropout Percentage:  5.0
Bias:  True
```

Fig. 13. Best result on test data

V. Discussion and Conclusion

From the experiments we conducted, we concluded that tanh activation function is the best activation function for our MLP, using a 15% training dataset gives us the best accuracy, adding dropout regularization to the network, specifically a dropout percentage of 20%

increases the accuracy of the model and having 1 hidden layer gives us the best results. Using our optimal parameters, we were able to get an accuracy of 0.82 on unseen data.

CNN (with pooling and dropouts) gives the accuracy of 0.85 which is slightly better than that of MLP 0.82 on unseen Image data-set. This is because an MLP model takes vector (784*1) as it input where else a CNN takes tensor (28*28*1) as its input, therefore CNN model has the ability to understand spatial relation between pixels of images better than MLP for images classification.

VI. Statement of Contribution

Dian - Created MLP class with bias parameter and gradient descent optimizer. Also responsible for tuning hyper-parameters and selecting the best parameters for optimal result on test dataset.

Hadi - Primary duty for analyzing test results, summarize findings and writing the report.

Sagar Nandeshwar - Implemented and tested CNN with different parameters and layers and wrote its results and conclusion in report.

References

[1] [1]"Fashion MNIST", Kaggle.com, 2022. [Online]. Available: https://www.kaggle.com/datasets/zalando-research/fashionmnist. [Accessed: 01- Apr- 2022].