Image classification Using Neural Network Models with Fashion MNIST Dataset

Uneeth Akula
50317480
Dept. of Computer Science
University at Buffalo
Buffalo, NY 14221
uneethak@buffalo.edu

Abstract

There are many different technique and models to solve the problem of image classification. It is important for us to fully understand the principles behind each model and its performance based on the dataset. The purpose of this project is to gain a deeper understanding of different classification models, and how they perform on the Fashion-MNIST dataset. I implemented a single layer neural network using the classic machine learning model from scratch and the Convolutional neural network (CNN) model using Tensorflow and Keras with various activation functions to boost the performance of the models. .

1 Introduction

Machine learning is one of the most exciting fields in the modern age of technology. Thanks to machine learning, we enjoy robust email spam filters, convenient text and voice recognition, reliable web search engines, challenging chess players, and, hopefully soon, safe and efficient self-driving cars.

Machine learning for image classification is an important problem that has received a lot of attention recently and has been a major challenge for modern science and engineering. It is the core foundation for bigger problems such as Computer Vision, Face Recognition System, or Self-driving car. There are many classification models that can be used for this task; however, it is important for us to fully understand the concepts of each model, and how they perform on different dataset. Machine learning models seek to replicate the remarkable ability of humans and many other animals to make sense of and classify natural images. Some insight into how was achieved by Hubel and Wiesel in the 1950s-1960s when they showed that the visual cortex of monkeys and cats contains neurons that individually respond to small regions of the visual field. Mirroring this, convolutional neural networks (CNNs), with tremendous success, assign weights to small regions (filters) of the pixels of an image instead of assigning individual weights to each pixel input as in a traditional fully-connected neural network. Over the course of training, CNN image classifiers update their weights using back-propagation to improve classification accuracy on the training dataset. However, multi-layer neural networks are less interpretable than simpler linear models, so how the high-level features learned by neural networks contribute to their success is not fully understood.

The reason MNIST is so popular has to do with its size, allowing deep learning researchers to quickly check and prototype their algorithms. This is also complemented by the fact that all machine learning libraries (e.g. scikit-learn) and deep learning frameworks (e.g.

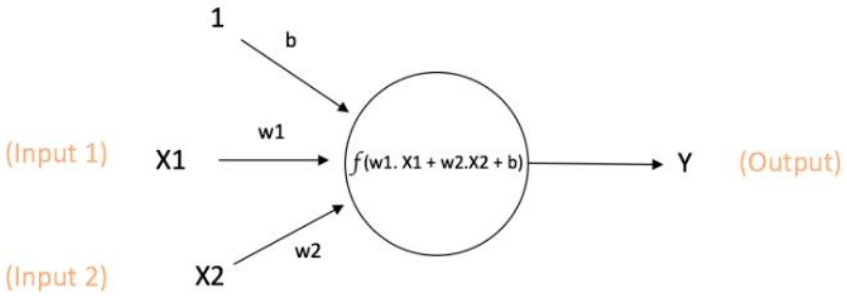
Tensorflow, Pytorch) provide helper functions and convenient examples that use MNIST out of the box.

2 Artificial Neural Network

Artificial neural networks use different layers of mathematical processing to make sense of the information it's fed. Typically, an artificial neural network has anywhere from dozens to millions of artificial neurons called units arranged in a series of layers. The input layer receives various forms of information from the outside world. This is the data that the network aims to process or learn about. From the input unit, the data goes through one or more hidden units. The hidden unit's job is to transform the input into something the output unit can use.

2.1 Single layer Neural Network

A single-layer neural network represents the most simple form of neural network, in which there is only one layer of input nodes that send weighted inputs to a subsequent layer of receiving nodes, or in some cases, one receiving node.



Output of hidden layer

$$Y = f(w1.\,X1 + w2.X2 + b)$$

Where w1, w2 are the weights and b is the bias

2.2 Activation Functions

The models discussed for image classification, use various activation functions which are used by the hidden layer.

2.2.1 Sigmoid Function

The Sigmoid function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}}$$

For large positive values of z, the sigmoid function should be close to 1, while for large negative values of z, the sigmoid function will be close to 0.

2.2.2 tanh

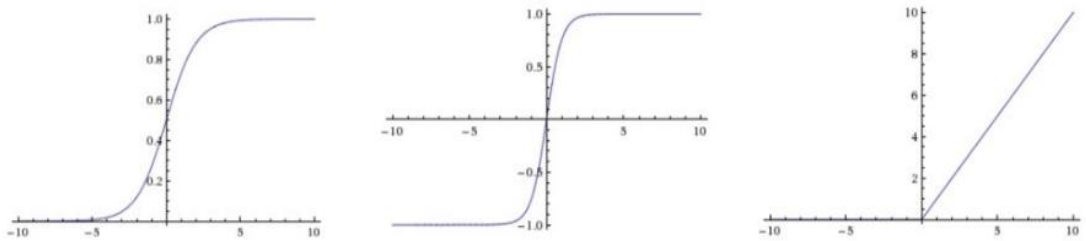
Tanh function takes a real-valued input and squashes it to the range [-1, 1]

$$\tanh(Z) = 2\sigma(2Z) - 1$$

2.2.3 ReLU:

ReLU stands for Rectified Linear Unit. It takes a real-valued input and thresholds it at zero (replaces negative values with zero)

$$f(x) = \max(0, x)$$



Sigmoid tanh ReLU

2.2 Decision Boundary

The softmax function used in the output layer gives the probability of the 10 classes of output for each input. In our model we are using the class with the maximum probability.

2.3 Multi Class Cross Entropy Loss Function

The cost function used in this project is Cross Entropy Loss. It is a softmax activation and a cross entropy loss. The cost function helps us to find the right value of weights and bias in the best possible time so that our decision boundary fits our case. We can predict the value of dependent variable from independent variables. Starting with random values for weights and zero for bias value, we find that difference between actual and predicted value. So, the Cost function is used as a measurement parameter of our model. Loss function is defined as below.

$$H_{y'}(y) := -\sum_i y'_i \log(y_i)$$

y' is the predicted output of softmax function, y is the actual data output

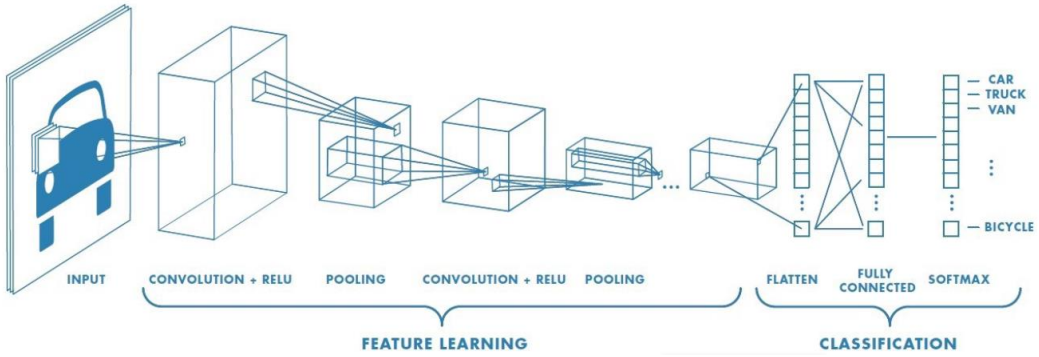
2.4 Hyper Parameters

Number of hidden nodes and Learning Rate are the hyper parameters which are tuned in order to minimize the cost function and increase the accuracy. I used 512 hidden layers and 10^{-6} learning rate for the 1st part. After training the model with the train data set, the accuracy for test data set is calculated.

2.5 Convolutional Neural Networks (CNN)

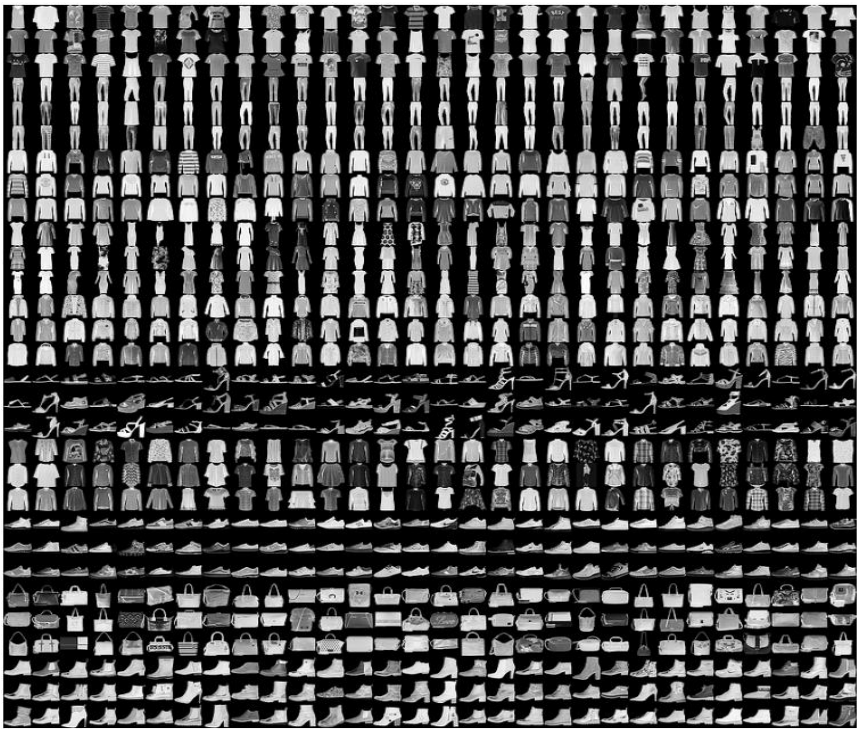
To enable classification, CNNs employ a series of filters to the raw pixel data of an image for extraction of feature selection after learning. The first component of a CNN is its convolutional layers. Convolutional layers apply a given number of convolutional filters to the given image. The convolutional layer carries out a set of mathematical operations for each subregion of the image. The resulting outcome is a singular value produced in the output feature map. Additionally, a rectified linear units (ReLu) function is applied by the

convolutional layers to introduce nonlinearity to the model. The second component is max pooling layers which are inserted in between convolutional layers in a CNN architecture. Pooling layers function by downsampling the given image data extracted by the convolutional layers and reduce the dimensionality of the image. Put differently, for a given pixel tile extracted from the convolutional layer, max pooling distills subregions of the pixel tile containing the maximum values of the subregions. By doing so, multiple benefits for the model are realized. The spatial size of the images is reduced to minimize computation time. Consequently, the number of parameters for feature selection is minimized as well. In turn, the model avoids overfitting by narrowing the scope of parameters being selected.



3 Data Set

I used the Fashion MNIST dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels), as seen here:



Fashion MNIST is intended as a drop-in replacement for the classic MNIST dataset—often used as the "Hello, World" of machine learning programs for computer vision. The MNIST dataset contains images of handwritten digits (0, 1, 2, etc.) in a format identical to that of the articles of clothing you'll use here.

This guide uses Fashion MNIST for variety, and because it's a slightly more challenging problem than regular MNIST. Both datasets are relatively small and are used to verify that an algorithm works as expected. They're good starting points to test and debug code.

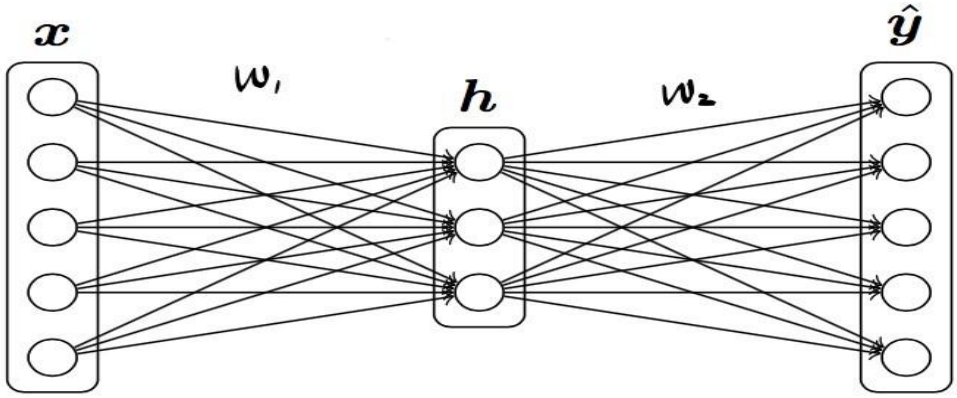
Here, 60,000 images are used to train the network and 10,000 images to evaluate how accurately the network learned to classify images. You can access the Fashion MNIST directly from TensorFlow

4 Preprocessing

The Data given was split into two sets: Training data, Test data with a ratio of 6:1. The data must be preprocessed before training the network. If you inspect the first image in the training set, you will see that the pixel values fall in the range of 0 to 255. Scale these values to a range of 0 to 1 before feeding them to the neural network model. To do so, divide the values by 255. It's important that the training set and the testing set be preprocessed in the same way.

5 Architecture

5.1 Single layer neural network



We must choose the best parameters Weights and bias to minimize the errors. Here W1, W2, b1, b2 is not a single parameter. We will minimize the cost function by finding the best possible values of W1, W2, b1, b2. The minimization will be performed by gradient descent algorithm, whose task is to parse the loss function output until it finds the lowest output.

Firstly, we implement the forward propagation by activating the sigmoid function for the hidden layer and softmax function for the output layer then find the loss for the obtained values.

For Multi Class Cross Entropy Loss function is defined as:

$$H_{y'}(y) := -\sum_i y'_i \log(y_i)$$

Divide the loss function by number of samples m.

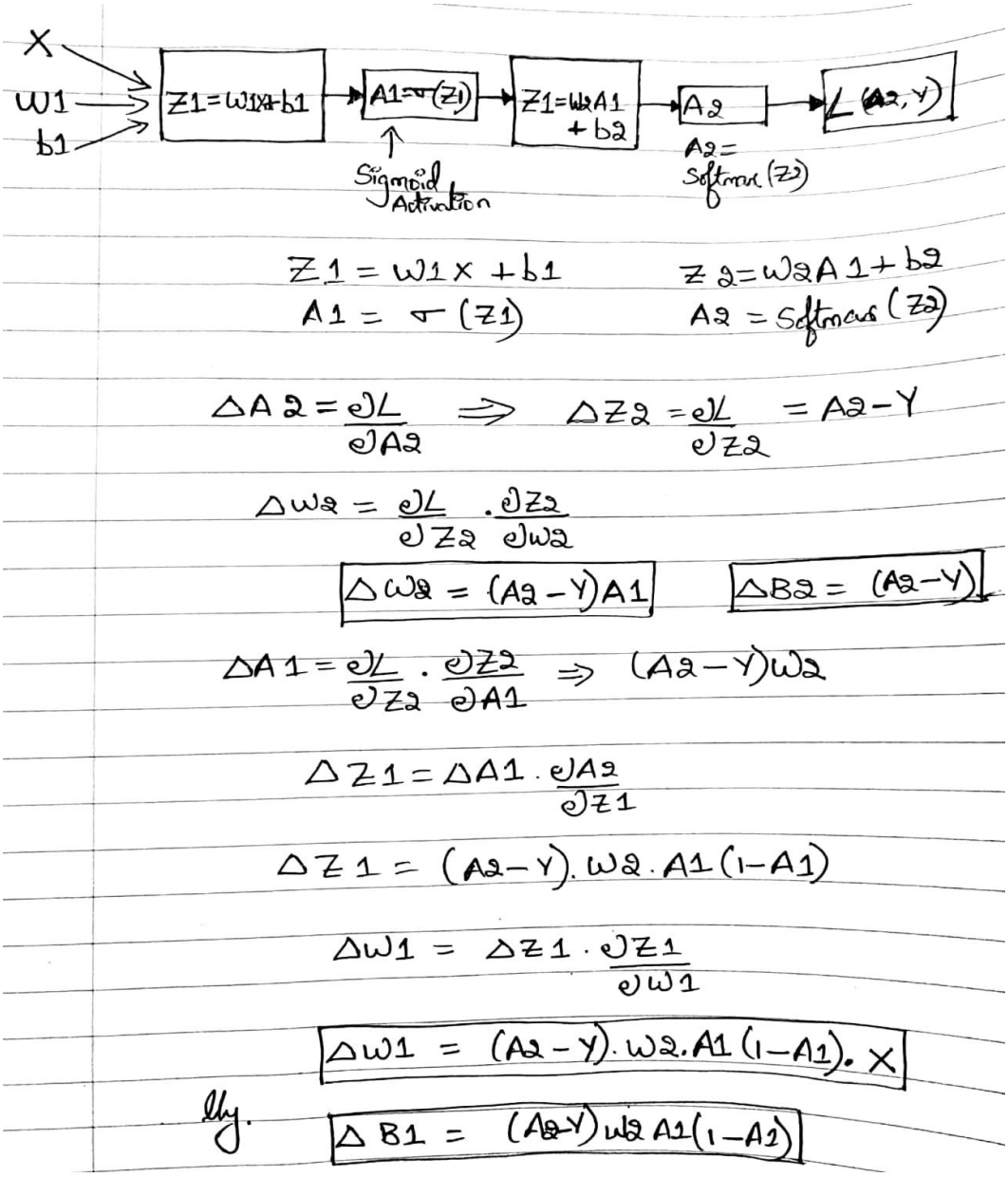
We must minimize the cost function, which will output the best set of parameters W1, W2,

b1, b2. The way we are going to minimize the cost function is using the gradient descent. To minimize the cost function, we must run the gradient descent function on each parameter.

Repeat until convergence for all the parameters in each weight vector W1, W2

Let A1 = sigmoid ($W1^T$.X+B1), then

A2 = softmax ($W2^T$.A1+B2)



The obtained gradients or derivatives are then used to update the parameters to by multiplying with learning rate for k no. of iterations to minimize the cost function.

Finally, we get optimum values for W1, W2 and B1, B2 which are the best combinations of weights and Bias. We use these values in the forward propagation to predict the output of the test data.

5.2 Multilayer Neural Network with Keras

To implement multilayer neural network, I installed Tensorflow and Keras for faster computations on Graphics Processing Unit.

A problem with training neural networks is in the choice of the number of training epochs to use. Too many epochs can lead to overfitting of the training dataset, whereas too few may result in an underfit model. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a dataset.

I used **Early Stopping Regularization** to prevent overfitting with patience = 5. So, when there is an increase in the cost function consecutively for 5 iterations, it stops running.

I used two hidden layers, one with 128 hidden nodes and the next with 64 hidden nodes. The activation function used for first hidden layer is tanh and the sigmoid activation is used for second hidden layer. The output layer uses Softmax activation.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
```

5.3 Classification using Convolutional Neural Network

- In the convolutional neural network, I used one convolutional layer with 64 filters, and kernel size of 2 with activation function of relu.
- The next layer is a max pooling layer with a pool size of 2.
- The next layer is a convolutional layer with 16 filters with a filter size of 2 with activation function of tanh.
- The next layer is a max pooling layer with a pool size of 2.
- Later I flattened the image and then gave a dense fully connected layer with 128 neurons with activation function of relu.
- Finally, the output layer consists of 10 neurons with an activation of SoftMax.
- I used early stopping on validation accuracy, patience of 2, so, when there is an increase in the cost function consecutively for 2 iterations, it stops running. The training stopped after 10 epochs.
- I got an accuracy of 95.70% and a validation accuracy of 91.21%.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 16)        4112
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 106,202
Trainable params: 106,202
Non-trainable params: 0
_________________________________________________________________
```

4 Results

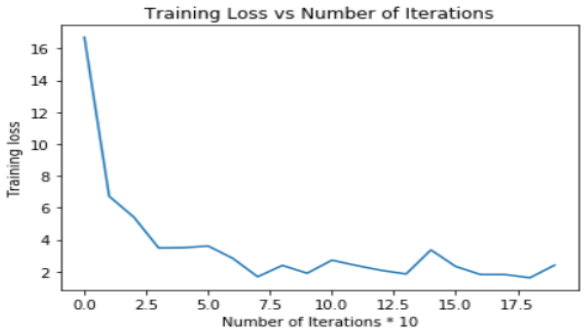
4.1 Artificial Single Layer Neural Network from scratch

The Loss and Accuracy with learning rate = 10^{-6} for 200 iterations

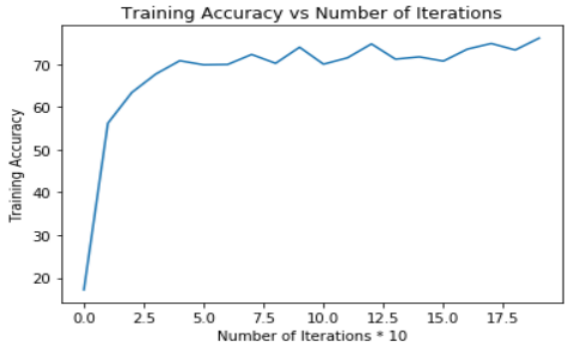
```
Loss function value after 20 iterations:  5.4045579009865685
Training Accuracy after 20 iterations: 63.431666666666665
Loss function value after 30 iterations:  3.475105086885631
Training Accuracy after 30 iterations: 67.74833333333333
Loss function value after 40 iterations:  3.4890337596630383
Training Accuracy after 40 iterations: 70.86833333333334
Loss function value after 50 iterations:  3.597062154999609
Training Accuracy after 50 iterations: 69.90666666666667
Loss function value after 60 iterations:  2.816124053577066
Training Accuracy after 60 iterations: 69.99
Loss function value after 70 iterations:  1.6666039891512248
Training Accuracy after 70 iterations: 72.33500000000001
Loss function value after 80 iterations:  2.3779895964410707
Training Accuracy after 80 iterations: 70.25666666666666
Loss function value after 90 iterations:  1.885798837036697
Training Accuracy after 90 iterations: 74.03166666666667
Loss function value after 100 iterations:  2.705138406904616
Training Accuracy after 100 iterations: 70.05333333333333
Loss function value after 110 iterations:  2.3689983603599702
Training Accuracy after 110 iterations: 71.55166666666666
Loss function value after 120 iterations:  2.0649918604891697
Training Accuracy after 120 iterations: 74.81166666666667
Loss function value after 130 iterations:  1.840206844041104
Training Accuracy after 130 iterations: 71.23666666666666
Loss function value after 140 iterations:  3.3499740793254715
Training Accuracy after 140 iterations: 71.77166666666666
Loss function value after 150 iterations:  2.313261645756238
Training Accuracy after 150 iterations: 70.775
Loss function value after 160 iterations:  1.8080410887658003
Training Accuracy after 160 iterations: 73.57833333333333
Loss function value after 170 iterations:  1.8047191714041997
Training Accuracy after 170 iterations: 74.91666666666667
Loss function value after 180 iterations:  1.6024174117544174
Training Accuracy after 180 iterations: 73.38166666666666
Loss function value after 190 iterations:  2.39438316138482
Training Accuracy after 190 iterations: 76.17
```

Training Accuracy obtained after 200 iterations is 76.17%

The Cross-Entropy Loss vs No. of iterations



The training accuracy vs No. of iterations



Finally, with the learning = 10^{-6} and epochs = 200 the test data set was plugged in to predict the diagnosis with the obtained weights and bias. The results for the test dataset is shown

```
test_samples = X_test.shape[0]
A1_test,A2_test=forward_prop(X_test, W1, W2, b1, b2)
test_predictions = predict(X_test,W1,W2,b1,b2)
current_test_accuracy=sum(Y_test_actual==test_predictions)/test_samples
print("Test Accuracy with the current Model:", current_test_accuracy*100)
```

```
Test Accuracy with the current Model: 73.56
```

Confusion Matrix:

```
array([[510,  15,   1,  67,   9,   4, 375,   0,  19,   0],
       [  3, 924,   2,  32,  11,   0,  26,   0,   2,   0],
       [ 11,  11, 217,  18, 253,   1, 479,   1,   7,   2],
       [ 16,  27,   1, 762,  68,   4, 110,   2,   8,   2],
       [  1,   4,   6,  25, 689,   6, 262,   0,   7,   0],
       [  2,   0,   0,   1,   0, 861,   9,  72,  16,  39],
       [ 45,  11,  21,  27,  95,   7, 770,   0,  24,   0],
       [  0,   1,   0,   0,   0,  66,   1, 865,   3,  64],
       [  1,   5,   2,   9,  12,  13,  73,  15, 868,   2],
       [  0,   0,   0,   1,   0,  38,   6,  63,   2, 890]], dtype=int64)
```

4.2 Multi-Layer Neural Network with Tensorflow and Keras

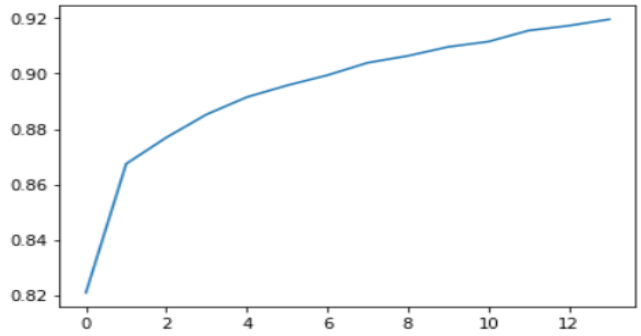
Two hidden layers were used, one with 128 hidden nodes and the second with 64 hidden nodes.

Training and validation's loss and accuracy for this model is shown below.

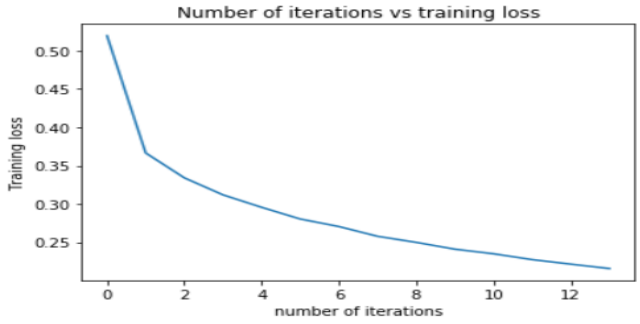
```
Epoch 8/100
60000/60000 [==============================] - 5s 88us/step - loss: 0.2578 - accuracy: 0.9039 - val_loss: 0.3343 - val_accurac
y: 0.8768
Epoch 9/100
60000/60000 [==============================] - 5s 89us/step - loss: 0.2499 - accuracy: 0.9064 - val_loss: 0.3241 - val_accurac
y: 0.8835
Epoch 10/100
60000/60000 [==============================] - 5s 88us/step - loss: 0.2411 - accuracy: 0.9097 - val_loss: 0.3403 - val_accurac
y: 0.8769
Epoch 11/100
60000/60000 [==============================] - 5s 88us/step - loss: 0.2350 - accuracy: 0.9116 - val_loss: 0.3464 - val_accurac
y: 0.8746
Epoch 12/100
60000/60000 [==============================] - 5s 90us/step - loss: 0.2274 - accuracy: 0.9156 - val_loss: 0.3333 - val_accurac
y: 0.8811
Epoch 13/100
60000/60000 [==============================] - 6s 93us/step - loss: 0.2215 - accuracy: 0.9173 - val_loss: 0.3298 - val_accurac
y: 0.8833
Epoch 14/100
60000/60000 [==============================] - 6s 97us/step - loss: 0.2158 - accuracy: 0.9196 - val_loss: 0.3391 - val_accurac
y: 0.8807
Restoring model weights from the end of the best epoch
Epoch 00014: early stopping
```

Since we used early stopping regularization, model terminates after 14 epochs, even though the no. of epochs given is 100.

Accuracy vs No. of iterations



Loss vs No. of iterations



Confusion Matrix

```
array([[870,   0,   6,  13,   3,   0, 105,   0,   3,   0],
       [ 10, 970,   1,  15,   3,   0,   1,   0,   0,   0],
       [ 73,   0, 741,   7, 117,   1,  61,   0,   0,   0],
       [ 61,   7,   4, 866,  42,   1,  17,   0,   2,   0],
       [ 38,   1,  60,  16, 848,   0,  37,   0,   0,   0],
       [  4,   0,   0,   0,   0, 946,   0,  27,   1,  22],
       [181,   0,  52,  20,  89,   0, 655,   0,   3,   0],
       [  5,   0,   0,   0,   0,  13,   0, 951,   0,  31],
       [ 28,   0,   2,   4,   5,   4,   3,   4, 950,   0],
       [  3,   0,   0,   0,   0,   6,   1,  30,   0, 960]], dtype=int64)
```

4.3 Classification with Convolutional Neural Network

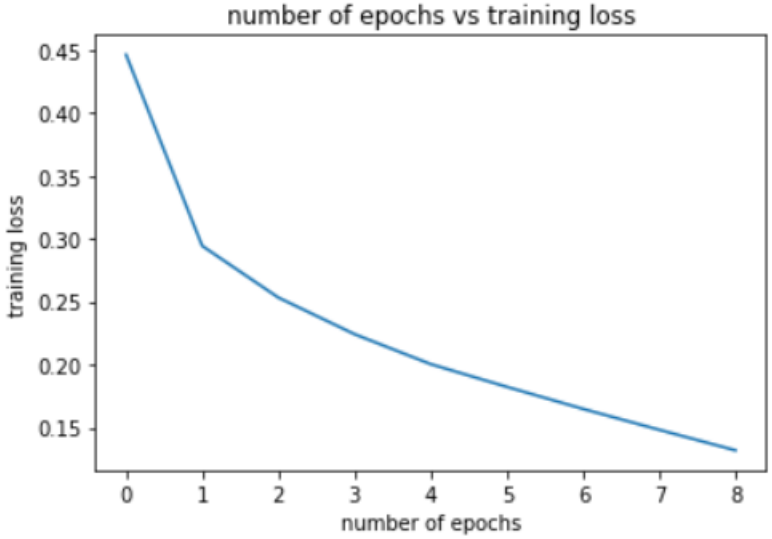
Two hidden layers were used, one with 64 nodes and the second with 16 nodes, and their respective pooling matrices.

Training and validation's loss and accuracy for this model is shown below.

```
Epoch 5/50
60000/60000 [==============================] - 40s 671us/step - loss: 0.2004 - accuracy: 0.9255 - val_loss: 0.2813 - val_accura
cy: 0.8980
Epoch 6/50
60000/60000 [==============================] - 39s 657us/step - loss: 0.1824 - accuracy: 0.9327 - val_loss: 0.2506 - val_accura
cy: 0.9120
Epoch 7/50
60000/60000 [==============================] - 40s 659us/step - loss: 0.1649 - accuracy: 0.9393 - val_loss: 0.2728 - val_accura
cy: 0.9083
Epoch 8/50
60000/60000 [==============================] - 40s 666us/step - loss: 0.1483 - accuracy: 0.9449 - val_loss: 0.2741 - val_accura
cy: 0.9087
Epoch 9/50
60000/60000 [==============================] - 40s 675us/step - loss: 0.1320 - accuracy: 0.9507 - val_loss: 0.2677 - val_accura
cy: 0.9124
Restoring model weights from the end of the best epoch
Epoch 00009: early stopping
```

Since we used early stopping regularization, model terminates after 9 epochs, even though the no. of epochs given is 50.

Cross Entropy Loss vs No. of iterations



Confusion Matrix

```
array([[883,   0,   9,  28,   5,   1,  72,   0,   2,   0],
       [  8, 981,   0,  11,   0,   0,   0,   0,   0,   0],
       [ 46,   1, 862,   9,  43,   0,  39,   0,   0,   0],
       [ 33,   8,   6, 921,  18,   0,  13,   0,   1,   0],
       [ 33,   1,  33,  32, 849,   0,  52,   0,   0,   0],
       [  0,   0,   0,   0,   0, 982,   0,  15,   0,   3],
       [146,   1,  57,  29,  65,   0, 699,   0,   3,   0],
       [  1,   0,   0,   0,   0,   3,   0, 980,   0,  16],
       [ 19,   1,   2,   3,   2,   1,   4,   4, 961,   3],
       [  3,   0,   0,   0,   0,   7,   0,  39,   0, 951]])
```

The Validation Accuracy is 91.24

6 Conclusion

Successfully trained the models using the given Fashion MNIST data and tested the model using the testing data. The accuracy obtained for the 1st part, Single Layer Neural network from scratch is 73.56. The validation accuracy for multilayer NN with Keras is 88.07%. The validation accuracy for two-layer Convolutional Neural network is 91.24%. Hence, we can observe that using a Multilayer layer network is better than a single layer neural network and a convolutional neural network with pooling matrices gives better accuracy and a good model than the other two models. So, one can use Convolutional Neural Network for a multi class classification.

7 References

[1]https://towardsdatascience.com/how-to-build-a-simple-neural-network-from-scratch-with-python-9f011896d2f3 Neural Network from Scratch

[2] https://www.tensorflow.org/tutorials/keras/classification Classification Tutorial-Tensorflow

[3] https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ Quick Introduction to Neural Networks

[4] https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping// Early Stopping to Halt training of NN at right Time.

[5]https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html Pandas.DataFrame Pandas

[6]https://docs.scipy.org/doc/numpy/reference/ Numpy

[7] https://www.tensorflow.org/install/pip Installing Tensorflow

[8] https://inmachineswetrust.com/posts/deep-learning-setup/ Installing Keras