Contents:

Chapter-1

1.1 Abstract:

High performing neural networks are under development so that they can be used with augmented reality based head sets to identify the best fitting fashion accessories and clothing for remotely located customers over the internet, efficient image classification algorithms are in high demand for classifying the type of image input.

In this project I have created various multi layer neural network models and then selected the most accurate and efficient model for classifying an input image as one of the following among T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle Boot.

1.2 Goal of the project:

Goal of this project is to implement and compare the performance of different fully-connected feed forward neural networks on the given dataset.

1.3 Introduction:

This project report is split into 2 dominant parts, where first part elaborates about the type of dataset being used, majorly detailing about the features for better understanding of dataset. Second part of this report details about creating, implementing and comparing various result parameters of different models created for classifying the given dataset.

During this procedure I have described in depth usage of python based algorithms, methods and tools used to achieve the goal of this project. The given dataset is broadly split into training dataset with 54,000 images, validation dataset with 6,000 and test dataset containing 10,000 images respectively, various models are classified later which "vary" in test size, epochs, iterations, validations, test size, activation functions and number neuron layers.

Chapter-2: Understanding the MNIST Fashion data set.

2.1 Loading the dataset:

Before diving into the dataset it is important to understand correct method of loading the dataset. First method is to use a web link to download 122MB of MNIST fashion dataset in csv format.

Second method to load dataset is by using keras.datasets function, which typically loads entire dataset in a variable.

2.2 Description of the MNIST fashion dataset:

Given MNSIT fashion dataset contains 70,000 images each with dimension of 28x28 pixels that classifies 10 different categories of images as T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle Boot. I have used various python based tools and libraries to explain dataset, see figure-1 and figure-2.

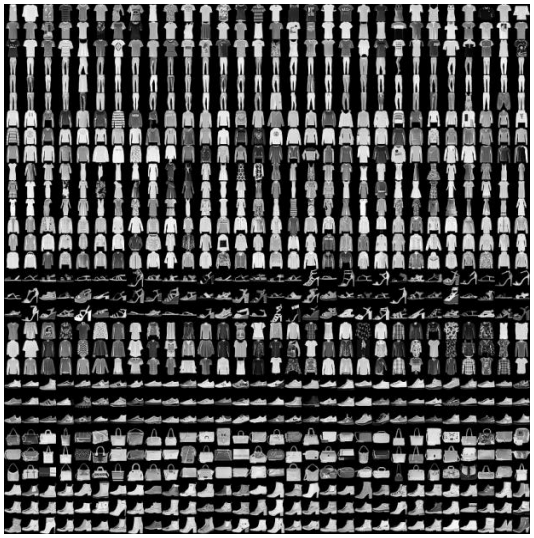

Figure-1: MNIST Fashion dataset in form of images

Figure-2: Regenerated images from MNIST fashion binary data.

Figure-1 shows various gray scale images contained in the dataset, figure-2 shows few images re-generated from binary gray scale data given in the dataset with specified classes.

```
-head()-
   pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  ...  pixel778  pixel779  pixel780  pixel781  pixel782  pixel783  pixel784  cl
ass
0       0       0       0       0       0       0       0       0  ...         0         0         0         0         0         0         0
  9
1       0       0       0       0       0       1       0       0  ...        76         0         0         0         0         0         0
  0
2       0       0       0       0       0       0       0       0  ...         0         0         0         0         0         0         0
  0
3       0       0       0       0       0       0       0       0  ...         0         0         0         0         0         0         0
  3
4       0       0       0       0       0       0       0       0  ...         0         0         0         0         0         0         0
  0

[5 rows x 785 columns]
----
-Describe-
              pixel1        pixel2        pixel3        pixel4  ...      pixel782      pixel783      pixel784         class
count  70000.000000  70000.000000  70000.000000  70000.000000  ...  70000.00000  70000.000000  70000.000000  70000.000000
mean       0.008771      0.006414      0.034486      0.098886  ...      2.75140      0.836529      0.072914      4.500000
std        0.087339      0.296605      1.200882      2.458872  ...     17.38577      9.258426      2.129924      2.872302
min        0.000000      0.000000      0.000000      0.000000  ...      0.00000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000  ...      0.00000      0.000000      0.000000      2.000000
50%        0.000000      0.000000      0.000000      0.000000  ...      0.00000      0.000000      0.000000      4.500000
75%        0.000000      0.000000      0.000000      0.000000  ...      0.00000      0.000000      0.000000      7.000000
max       16.000000     45.000000    218.000000    185.000000  ...    255.00000    255.000000    170.000000      9.000000
```

Figure-3: Various properties of the dataset.

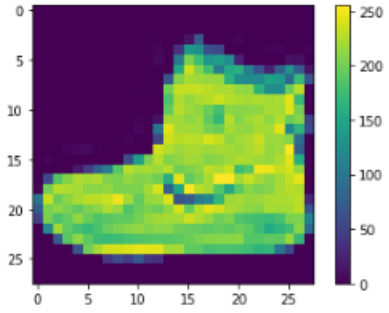

Figure-3.1. Graphical representation of dataset.

I have used various functions to explain my data set, which has 70,000 rows & 784 columns, consider figure-3, where first column is of labels having 70,000 entries. Each label here corresponds to 784 different pixels, therefore each image has 784 different parameters in gray scale ranging from 0 to 255 defining colors from white to black. I regenerated label[3] in figure-3.1 that shows pixels values of an image are between 0 to 255 therefore they need to be divided by 255 for normalization. Consider figure-3-1.1 for finding data type in each column.

```
-Data type of each column-
pixel1      int64
pixel2      int64
pixel3      int64
pixel4      int64
pixel5      int64
            ...
pixel781    int64
pixel782    int64
pixel783    int64
pixel784    int64
class       int64
Length: 785, dtype: object
----
-Total number of Rows and Columns-
(70000, 785)
```

Figure-3-1.1: Describing data type, total number of rows and columns in MNIST fashion dataset.

Figure 3.1.1 shown above describes the data type of each column as int64, which ensures that there are no bad, missing, corrupt entries, with a total of 70,000 rows and 785 columns where 1st column is of labels.

```
-Min and Max Pixel Value-
Minimum Pixel value 0
Maximum Pixel Value 255
----
-Correlation Matrix-
```

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	...	pixel780	pixel781	pixel782	pixel783	pixel784	class
pixel1	1.000000	0.255691	0.064036	0.043483	0.023103	0.013017	...	-0.001516	0.000851	0.008321	0.024112	0.037404	-0.000854
pixel2	0.255691	1.000000	0.670304	0.214933	0.148989	0.103213	...	0.005995	0.016011	0.037044	0.028276	0.016876	-0.000109
pixel3	0.064036	0.670304	1.000000	0.430599	0.215634	0.157399	...	0.008523	0.021486	0.040972	0.019205	0.013684	-0.013411
pixel4	0.043483	0.214933	0.430599	1.000000	0.597551	0.351249	...	0.011914	0.026400	0.017989	0.003675	0.006959	-0.005943
pixel5	0.023103	0.148989	0.215634	0.597551	1.000000	0.718614	...	0.018629	0.024493	0.028515	0.019802	0.005796	-0.002532
...	...	...	...	...	...	...	...	...	...	...	...	...	...
pixel781	0.000851	0.016011	0.021486	0.026400	0.024493	0.036706	...	0.748771	1.000000	0.641700	0.327200	0.137703	-0.017627
pixel782	0.008321	0.037044	0.040972	0.017989	0.028515	0.040029	...	0.341613	0.641700	1.000000	0.647886	0.237353	0.045749
pixel783	0.024112	0.028276	0.019205	0.003675	0.019802	0.025806	...	0.172653	0.327200	0.647886	1.000000	0.490682	0.058216
pixel784	0.037404	0.016876	0.013684	0.006959	0.005796	0.003397	...	0.074947	0.137703	0.237353	0.490682	1.000000	0.021383
class	-0.000854	-0.000109	-0.013411	-0.005943	-0.002532	-0.008768	...	-0.067772	-0.017627	0.045749	0.058216	0.021383	1.000000

Figure-3.2. Min-Max pixel value and correlation matrix of the dataset.

Figure-3.2 shows correlation matrix which is of limited use since correlation of pixels provides no meaningful insight. Additionally it also shows the minimum and maximum values contained under label[3] which gives 0 as minimum and maximum as 255 further justifying that images are in gray scale.

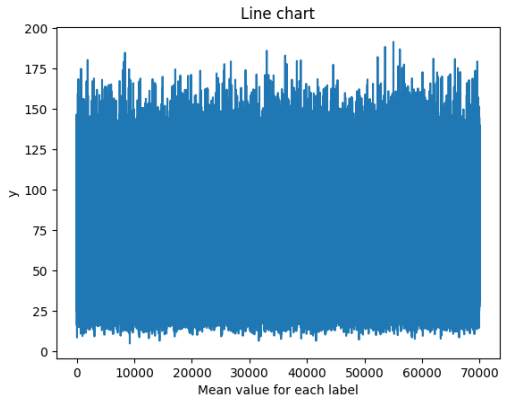

Figure-4. Average value plotted as line graph for each label consisting of 784 pixel values.

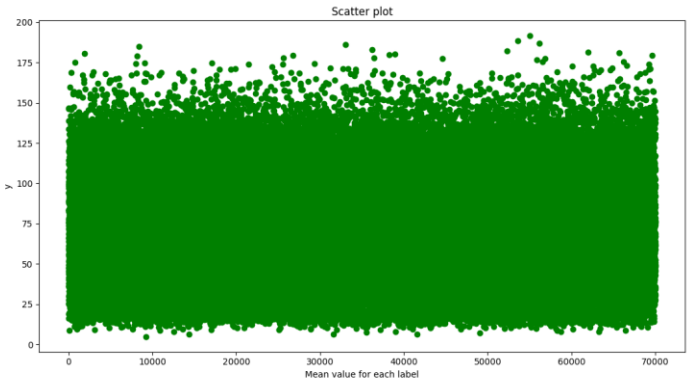

Figure-5. Average value plotted as scatter graph for each label consisting of 784 pixel values.

Average value of each row i.e. each MNIST fashion image data is plotted in figure-4 and figure-5 in form of a line chart and scatter plot respectively. Data in the graph is distributed between 20 to 180 gray scale values that provide an important insight explaining dataset is composed of uniformly distributed images neither too bright nor too dull. Further data is normalized by scaling it down between 0 to 1 from 0 to 255 by dividing every pixel value by 255, this ensures uniformity by downscaling values and reducing complexity during calculations.

```
   pixel1  pixel2  pixel3  pixel4  pixel5    pixel6  pixel7  ...  pixel779  pixel780  pixel781  pixel782  pixel783  pixel784     class
0     0.0     0.0     0.0     0.0     0.0  0.000000     0.0  ...       0.0       0.0       0.0       0.0       0.0       0.0  0.035294
1     0.0     0.0     0.0     0.0     0.0  0.003922     0.0  ...       0.0       0.0       0.0       0.0       0.0       0.0  0.000000
2     0.0     0.0     0.0     0.0     0.0  0.000000     0.0  ...       0.0       0.0       0.0       0.0       0.0       0.0  0.000000
3     0.0     0.0     0.0     0.0     0.0  0.000000     0.0  ...       0.0       0.0       0.0       0.0       0.0       0.0  0.011765
4     0.0     0.0     0.0     0.0     0.0  0.000000     0.0  ...       0.0       0.0       0.0       0.0       0.0       0.0  0.000000

[5 rows x 785 columns]
----
-Describe-
             pixel1        pixel2        pixel3        pixel4  ...      pixel782      pixel783      pixel784         class
count  70000.000000  70000.000000  70000.000000  70000.000000  ...  70000.000000  70000.000000  70000.000000  70000.000000
mean       0.000003      0.000025      0.000135      0.000388  ...      0.010790      0.003281      0.000286      0.017647
std        0.000343      0.001163      0.004709      0.009643  ...      0.068179      0.036308      0.008353      0.011264
min        0.000000      0.000000      0.000000      0.000000  ...      0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000  ...      0.000000      0.000000      0.000000      0.007843
50%        0.000000      0.000000      0.000000      0.000000  ...      0.000000      0.000000      0.000000      0.017647
75%        0.000000      0.000000      0.000000      0.000000  ...      0.000000      0.000000      0.000000      0.027451
max        0.062745      0.176471      0.854902      0.725490  ...      1.000000      1.000000      0.666667      0.035294

[8 rows x 785 columns]
----
-Info-
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 70000 entries, 0 to 69999
Columns: 785 entries, pixel1 to class
dtypes: float64(785)
memory usage: 419.2 MB
None
----
-Minimum and Maximum Pixel-
0.0
1.0
```

Figure-6. Data information in normalized condition.

Figure-6 represents scaled down mean, maximum and minimum values for each label, whereas figure-7 and figure-8 represent a line chart and scatter plot of mean values for each row, signifying that data is successfully scaled down between 0 to 1 having its mean around 0.5.

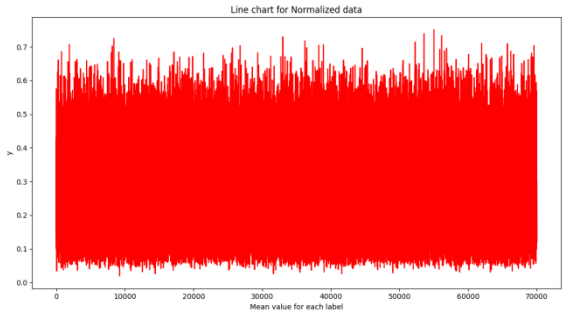

Figure-7. Normalized average value plotted as line graph for each label consisting of 784 pixel values.

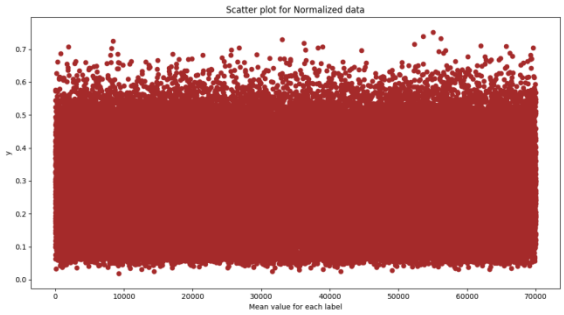

Figure-8. Normalized average value plotted as scatter graph for each label consisting of 784 pixel values.

Furthermore figure-6.1 confirms that all pixels have gray scale resolution from 0 to 255 simultaneously, it is evident from figure-6 that no data is lost during normalization since total count of 784 pixels is still maintained at 70,000, please refer table-1.

Table-1: Basic data description.

Type of data	Number of rows	Number of columns	Min Val	Max Val
Not Normalized	70,000	784	0	255
Normalized	70,000	784	0	1

Chapter-3: Designing a Neural Network Model.

Step 0 – Input data explanation, method used for loading and splitting dataset.

The input to models discussed below are set of 70,000 images each with a dimension of 28x28, in total having 10 different classes as discussed above in detail. Dataset is loaded into a single variable for further processing by using keras.datasets function call.

It should be noted that keras.datasets splits the given dataset into 80% as training set and remaining 20% as test set, therefore training set has 60,000 images whereas, test set has 10,000 images.

Step 1 - Building Basic architecture.

Basic architecture of various models that I have used in this project is multi-layered, fully connected and feed forward neural network with a minimum of 1 hidden layer and 1 output layer as shown in table-2. All building blocks that are used for creating different models for this project are discussed below in detail.

Table-2 Modules used in creating basic neural network.

Index	Code used for creating the basic model	Comments
1.	model_0 = keras.Sequential ([	Creating layered network.
2.	keras.layers.Flatten(input_shape=(28, 28)),	Flattening the inputs into 784 pixels.
3.	keras.layers.Dense(158, activation='relu'),	First hidden layer with 100 neurons & relu.
4.	keras.layers.Dense(10, activation='softmax')])	Output layer with 10 neurons & softmax.

1. Use of Sequential API:

I have used Sequential API in this project since it helps in creating model layer by layer, does not allow sharing of layers and it is easy to extend or reduce the depth of my model over and over again for multiple simulations.

2. Use of Flatten API:

Flatten API is used to convert stacked or layered input into a single flattened out 1 dimensional (1-D) layer, which is easy to process. For example in this project 28x28 pixel image is flattened to 1D layer of 784 pixels.

3. Understanding Dense layer:

A dense layer is a fully connected layer of neurons which receive inputs from previous layers, have their own weights and bias.

output = activation(dot(input, kernel) + bias)……..eq1

For further explanation about dense layer refer eq1 where output of such layers depends upon the output of activation function used in that layer therefore activation refers to activation function, kernel refers to weights associated with that neuron.

Step 2 - Compiling the model

In this project I have used tensor flow and keras which have a specific parameter based function call for compiling above made model as given in table-3. The compile function typically converts my neural network model into a hardware friendly code. All the parameters required to compile above designed model are discussed below.

Table-3 Tenserflow specific function call for compiling NN model.

SI	Function call
1	model_2.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])

2.1 Adam as Optimizer:

I have used Adam instead of gradient descent algorithm for optimizing cost function because, unlike gradient descent algorithm which maintains constant learning rate, Adam typically computes adaptive learning rate by updating from first & second moments of the gradients.

2.2 Sparse_categorical_cross entropy as the Loss function:

I have studied various loss functions in class and I understand that it calculates the error between given output and predicted output. My goal is to minimize this loss function for achieving higher accuracy for my model. There are many loss functions for example Regression Loss Functions, Binary Classification Loss Functions, Multi-Class Classification Loss Functions.

As our problem statement revolves around classifying multi class input dataset into 10 specific classes therefore we are more interested in multi-class classification loss functions which can be further classified into Multi-Class Cross-Entropy Loss, Sparse Multiclass Cross-Entropy Loss and Kullback Leibler Divergence Loss. Since our dataset has multiple classes which are **not one-hot encoded** i.e. all pixel values for individual label and they are in form of an integer, hence we use Sparse Multiclass Cross-Entropy Loss function.

2.3 Use of Metrics accuracy:

Metrics are used to measure the accuracy of the model, keras library allows me to use any loss function as metric function. Generally accuracy metrics is used for classifying image correctly or incorrectly.

2.4 Learning Rate:

Default learning rate is **set to 0.02**, this hyper parameter is changed in model_2.3, please refer that section for specific results and discussions.

Step 3 - Training the model

Tenserflow offers fit function for training the above model, it uses back propagation algorithm for finding appropriate weights, refer to table-4 which shows the function call with correct parameters.

Table-4. Detailed parameters used in evaluate function.

SI	Function call to Train my model
1	model_2.fit(train_images, train_labels, epochs=2)

Fit function can be treated as a command to train a model, as mentioned above 80% of data from MNIST fashion dataset is used for training. The first two parameters train_images and train_labels are the variables that I created which contain 60,000 images and 10 distinct corresponding labels respectively. Last parameter of this function is epochs where one epoch means one forward pass and one backward pass while calculating or training the weights.

Step 4 – Model Evaluation

In this step above established model is evaluated on remaining 10,000 images of test dataset, this function helps in understanding the performance of our model on test data. It returns accuracy and error percentage of test dataset, refer to table-5. All parameters of this function are discussed below in detail.

Table-5. Representation of parameters used in fit function.

SI	Function call to Train my model
1	test_loss, test_acc = model_2.evaluate(test_images, test_labels, verbose=2)

First two parameters of this function are test images which are 20% of total dataset and corresponding 10 distinct image labels respectively. Verbose is set to 2 in this function so that only selected data is returned, otherwise null characters are returned.

Chapter-4: Results & discussions for various NN models.

In this section I have discussed about several NN models and their final results in terms of accuracy percentage and error percentage for both training and test data. Before deep diving into individual models, I have described below certain keywords and their usage which I believe are not obvious to understand in general terms.

Supervised or Unsupervised:

This problem is a supervised learning problem since both output and inputs are given in terms of 10 distinct classes.

Activation function:

These are decision makers mathematical functions which give an output to a set of inputs, I have used various activation functions like tanh, sigmoid, softmax etc., which are described below when required. It is worth mentioning that softmax activation function is used in most of the output layers since it is a multi class classification problem, whereas activation functions and other parameters are varied in hidden layers. Yet another advantage of softmax activation function in output layer is that it gives probabilities as output for classifying given inputs, therefore sum of this function for a given input is always one and output can be easily determined by selecting the one having maximum probability.

Dense layer:

As described above, dense layer is a fully connected, feed forward layer of neurons which receives input from previous layers and computes the output using their activation function.

Epochs:

As described above, in order to complete one epoch, back propagation algorithm used while training the weights requires one forward pass and one backward pass, epochs should not be confused with iterations.

Cross validation:

K-fold cross validation is not required for this example since training dataset has more than 50,000 samples, though automatic cross validation technique is used by splitting 10% of training data for automatic validation while fitting or training all models. Keras offers a parameter to be set in fit function that helps in visualizing validation error and accuracy.

Model_0:

In model_0 I have used 1 hidden layer having 158 neurons with ReLu activation function and 1 output layer having 5 neurons with softmax activation function, refer to table-6 for further details.

Table-6. Description of Model_0

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	158	ReLu	Error
2	Dense layer	5	Softmax	Error

This model fails to compile refer figure-x since number of neurons used in output layer with softmax as activation functions are less than total number of distinct classes, in this case number of neurons used in are less than 10, refer table-7.

Table-7. Results from Model_0

Epochs	Train -Test Size	Train data set 60,000 images		Test data set 10,000 images	
2	80% & 20%	Accuracy -	Error -	Accuracy -	Error -
Findings from the result		Errors out and fails to execute since parameters are wrong.			

Results from model_0:

Model fails to compile and never executes because parameters for softmax activation function are incorrect.

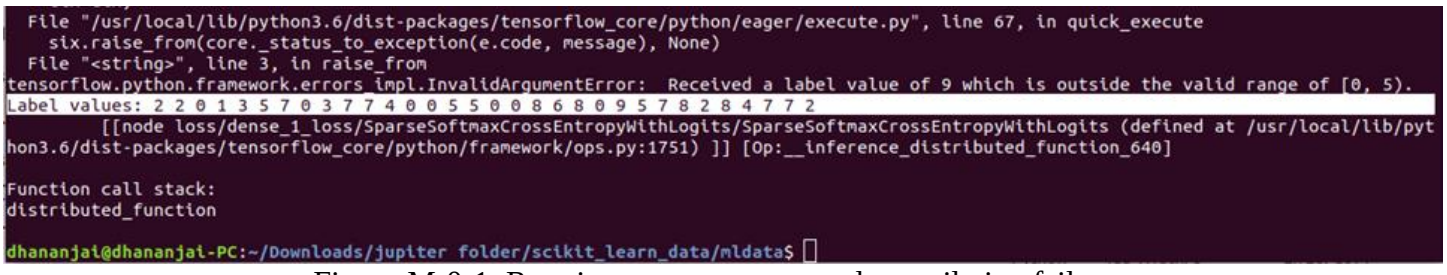

Figure-M-0-1. Run time error message and compilation failure.

Detailed discussions about model_0:

1) It should be noted that while using softmax activation function for classification of a multi-class problem, number of neurons in that layer should be exactly equal to number of distinct labels or classes since sum of

given probabilities in output should be 1, in this case there are 10 inputs for MNIST fashion dataset, therefore output layer must have 10 neurons.

2) If number of neurons are not equal to total number of distinct classes then model will never compile and program will fail to execute as shown in figure-M-0-1.

Model_1:

In model_1 I have used 1 hidden layer having 2 neurons with relu activation function and 1 output layer having 10 neurons with softmax activation function, refer to table-8 for further details.

Table-8. Description of Model_1.

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	2	relu	2
2	Dense layer	10	Softmax	10

Model_1 compiles successfully with no errors, only 2 optimal weights trained for hidden layer and 10 optimal weights attained for output layer respectively, table-9 represents results of this model with accuracy and error percentage.

Table-9. Results from Model_1.

Epochs	Train -Test Size	Train data set 54,000 images		Validation set 6,000 images		Test data set 10,000 images	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	80% & 20%	70	85	71	84	69	86
Findings from the result		Poor training accuracy, high error percentage, less number of weights and less number of neurons therefore under fitting.					

Results from model_1:

Table-9 represents low training accuracy with high error percentage that confirms model_1 is under fitting, this result is consolidated by figure-M-1-1 representing training accuracy and validation accuracy.

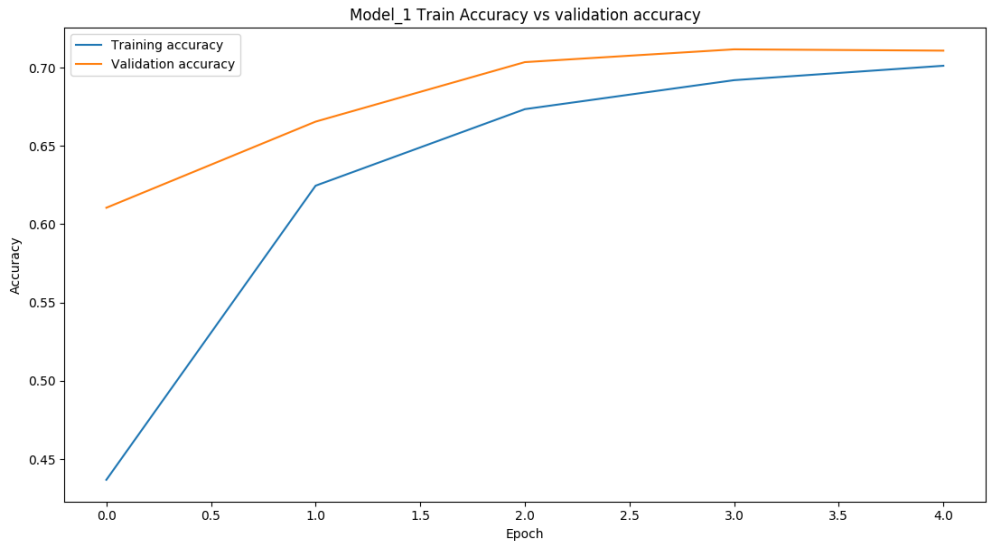

Figure-M-1-1. representing training accuracy against validation accuracy.

Figure-M-1-1 shows the orange line for validation accuracy iterating on 6,000 sample images and blue line for training data accuracy iterating on 54,000 sample images. Initially during epoch 1 difference between accuracy is nearly 20% because default weights used initially are different from optimal weights though as time progresses for higher epochs error percentage reduces and optimal weights are identified by back propagation algorithm which increases the accuracy for both training and validation dataset. It should be denoted that validation accuracy after 3rd epoch becomes constant therefore neither it decreases or increases which means optimal weights are found at 3rd epoch and increasing number of epochs will not make any difference, whereas figure-M-2-2 representing training loss with validation loss, which is fairly low.

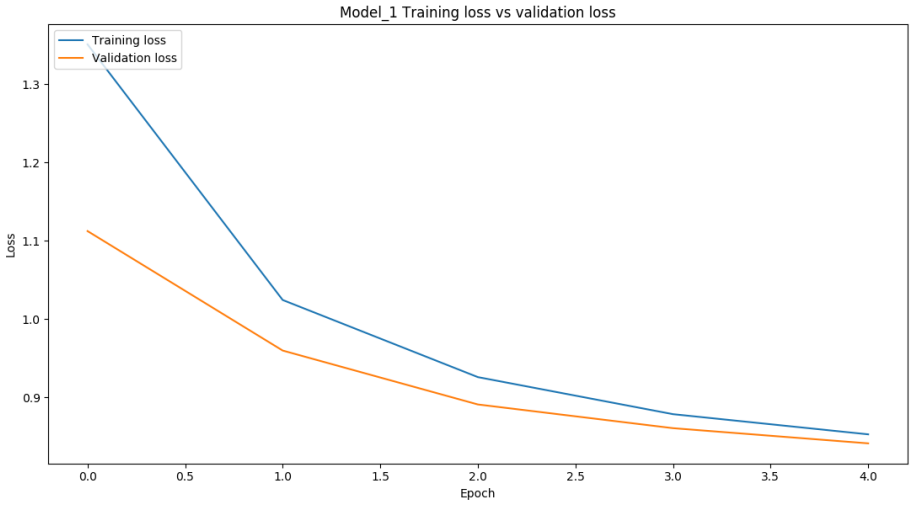

Figure-M-1-2. representing training error against validation error.

Figure-M-1-2 shows orange line for validation loss iterating on 6,000 sample images and blue line for training data loss iterating on 54,000 sample images. Initially loss percentage for both training and validation is very high since optimal weights are not yet calculated, thereafter 3rd epoch error percentage converges and almost becomes constant, that confirms optimal weights are identified. Yet error percentage is very high around 85%. Detailed discussions and reasoning from this model are explained in section below.

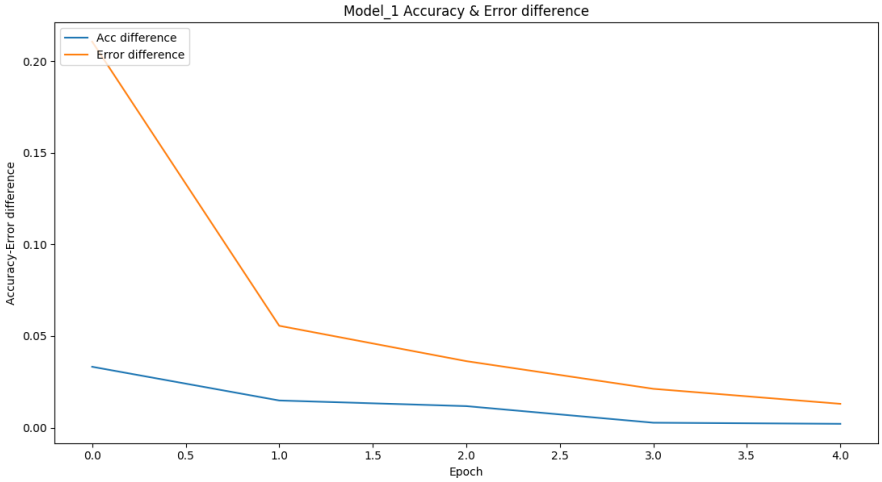

Figure-M-1-3. Representing difference in accuracy and error for training vs validation data.

I have made a separate code for plotting this graphical representation, figure-M-1-3 represents orange line that shows difference between training accuracy percentage and validation accuracy percentage slowly converging towards 0 with increasing number of epochs, similarly blue line is difference between training error percentage and validation error percentage again converging towards 0 because back propagation algorithm finds optimal weights which are not over fitting in this duration.

Detailed discussions about model_1:

1) Figure-M-1-1 shows low accuracy percentage, whereas figure-M-1-2 shows high error percentage during training because only 2 neurons are used in hidden layers, therefore less number of weights are trained during training procedure and unstable over simplified model is determined.

2) Training accuracy is 70% which is very low and indicates that model is under fitting, it should be made more complicated, therefore we need to increase number of neurons in hidden layer and total number of hidden layers should also be increased.

3) Considering above explanation it is no surprise that table-9 represents test accuracy of 69% which is fairly low and test error of 89% which is very high, therefore this model is not acceptable.

Model_2:

Learning from the outcomes of model_1, in model_2 I have used 1 hidden layer having 158 neurons with relu activation function and 1 output layer having 10 neurons with softmax activation function, refer to table-10 for further details.

Table-10. Description of Model_2.

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	158	relu	158
2	Dense layer	10	Softmax	10

Model_2 compiles successfully with no errors, 158 optimal weights are attained in first hidden layer and 10 optimal weights are trained for output layer respectively, table-11 represents results of this model with accuracy and error percentage.

Table-11. Results from Model_2

Epochs	**Train -Test Size**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	80% & 20%	89	29	87	34	86	36
Findings from the result		High accuracy percentage for training, validation and testing dataset, but none of the plots stabilize at 5 epochs, therefore more epochs should be increased.					

Results from model_2:

Table-11 represents high training accuracy with low error percentage, because this model is more complex with 158 optimal weights for 158 neurons in hidden layer. Figure-M-2-1 representing training accuracy with validation accuracy and figure-M-2-2 representing training loss with validation loss confirms that this model can be deployed for MNIST fashion dataset classification problem. Detailed discussions and reasoning from this model are explained in section below.

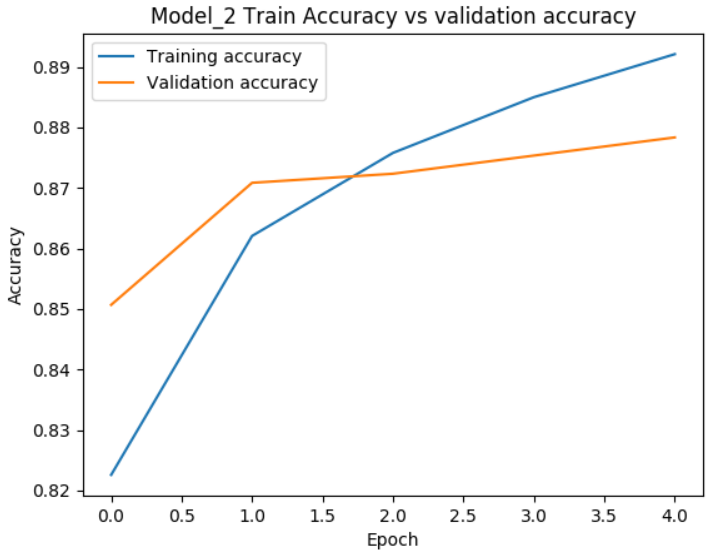

Figure-M-2-1 representing training accuracy with validation accuracy.

Figure-M-2-1 shows orange line for validation accuracy iterating on 6,000 sample images and blue line for training data accuracy iterating on 54,000 sample images. Initially during epoch 1 difference between accuracy is very low only 3% because default weights used initially are very close to optimal weights. Though as time progresses for higher epochs accuracy percentage increases, around 2nd epoch we can clearly see that both datasets have exact same accuracy of 87%, but after further training accuracy percentage further increases and becomes constant at 89% for test, 87% for validation.

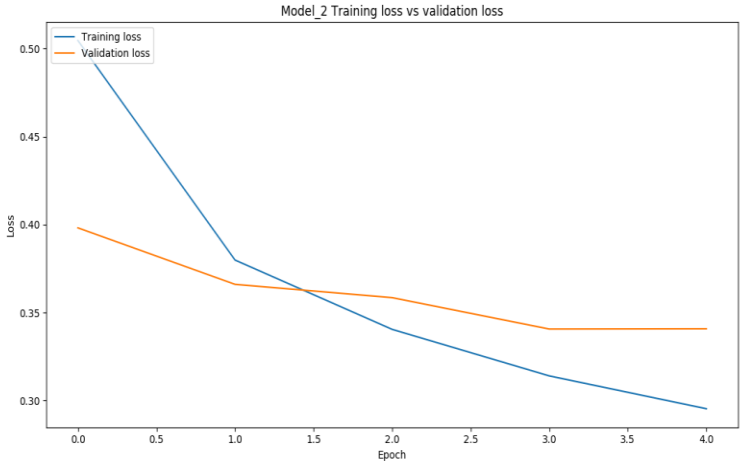

Figure-M-2-2 representing training loss with validation loss.

In figure-M-2-2 initially loss percentage for both training and validation is very low around 10% because weights are untrained, though as epochs increase back propagation algorithm slowly minimizes the cost function which is clearly shown in this figure since error percentage converges down to 30%.

Usage of Relu in hidden layer:

Relu activation function returns element-wise max(x, 0) for all x>0, else it is 0, this provides very simple and fast computation abilities in comparison to sigmoid or tanh functions. Output is always linear but non-negative, therefore it generalizes input data by setting negative inputs to 0 during final output unlike sigmoid function.

Special Observation:

For relu as activation function with 5 epochs training accuracy in figure-M-2-1 never crosses 90% mark whereas, training error in figure-M-2-2 never falls below 15%. Therefore it is important to increase number of epochs and replace relu with some other activation function in following models. It should be noted that increasing training accuracy and minimizing training error does not guarantee high test accuracy percentage.

Detailed discussions about model_2:

1) Figure-M-2-1 shows high training accuracy percentage with high validation accuracy, whereas figure-M-2-2 shows low training error percentage with low validation error percentage, because more number of neurons are used in hidden layer as compared to model_1.

More number of neurons in this case give higher accuracy and make a neural network fairly complex. Because this model is fairly complicated it trains multiple weights that ensure greater adaptability for entire input range. A complex model with a set of optimal weights can predict classes of a diversified input.

2) Table-11 represents test accuracy which is fairly high and test error is low, but test accuracy never crosses above 90% mark, therefore number of epochs, activation function or number of hidden layers should be changed in further models, thus this model can not be used for classifying MNSIT fashion data.

Model_2.1: Reducing training sample size.

Model_2.1 differs from model_2 in terms of training data and validation data size which is 36,000 training samples and 24,000 validation samples, table-11.1 shares various results for this model.

Table-11.1: Various results of Model_2.1.

Model_2.1 has smaller training data set.							
Epochs	**Model Name**	**Train data set 36,000 images**		**Validation set 24,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	Model_2.1	81	52	81	54	80	58
Findings from the result		Lower training and test accuracy when training data samples are reduced.					

Explanation of Results:

Training and test accuracy drop significantly when training samples are reduced since enough data is not provided to train the weights, therefore test results have poor accuracy.

Model_2.2: Experimenting with loss function (MSE).

Model_2.2 differs from model_2 in terms of loss functions, this model uses mean square error as a loss function, results of this model are shared in table-11.2.

Table-11.2: Various results of Model_2.2.

Model_2.2 has mean square error as loss function							
Epochs	**Model Name**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	Model_2.2	4	2700	0.5	2700	0.04	2700
Findings from the result		In multiclass classification problems, MSE loss function generates poor accuracy and huge error percentage.					

Explanation of Results:

Mean square error basically squares every error number before taking final average, therefore big error numbers have a huge impact on the network, in general MSE is not preferred for multiclass classification problem, it can be used in regression problems. Therefore using a regression problem loss function for a classification problem ends up in a total mismatch thereby giving almost 0 accuracy and huge error percentage. Therefore sparse categorical cross entropy loss function should be used for classifying multi class problems.

Model_2.3: Finding best learning rate.

Model_2.3 differs from model_2 in terms of learning rate, this model has been simulated multiple times to find best learning rate, results of this model are shared in table-11.3.

Table-11.3: Various results of Model_2.3.

Model_2.3 uses different learning rates whereas epochs are set to 5.							
Epochs	**Learning Rates**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	0.0001	84	47	83	46	82	49
5	0.001	87	34	86	37	85	39
5	0.01	88	30	87	34	86	31
5	0.1	83	49	80	63	79	43
5	1.00	14	223	15	225	15	221
Findings from the result		Highest test accuracy and lowest test error are obtained when learning rate is set to 0.01, which is neither very high nor very low.					

Explanation of Results:

Very small learning rate like 0.0001 makes smaller changes in each weight, therefore it will take more number of epochs and longer training time to attain high accuracy percentage. Very large learning rate like 1.0 makes very big changes in the weight, although it consumes smaller training time but fails in finding optimal weights, therefore results in very high error percentage and low accuracy. Consider figure-2-3-1 and 2-3-1 that display highest accuracy and lowest error percentage against optimal learning rate of 0.01 respectively.

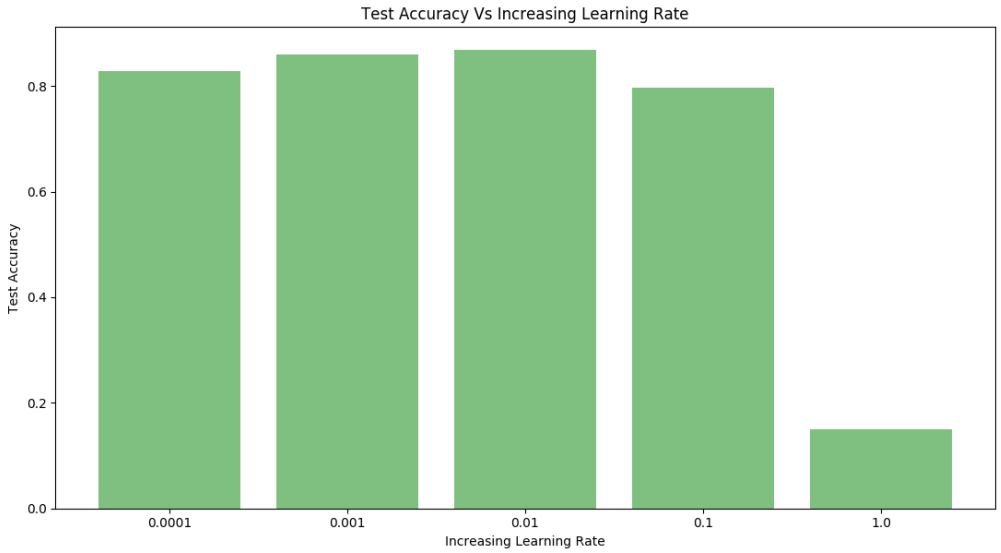

Figure-2-3-1: Test accuracy against increasing learning rates.

It is very evident from figure-2-3-1 and figure-2-3-2 that highest test accuracy and lowest test error is obtained when learning rate is set to 0.01. This is obvious since weights are neither trained very slowly nor trained very fast, therefore giving enough time for back propagation algorithm to find optimal weights.

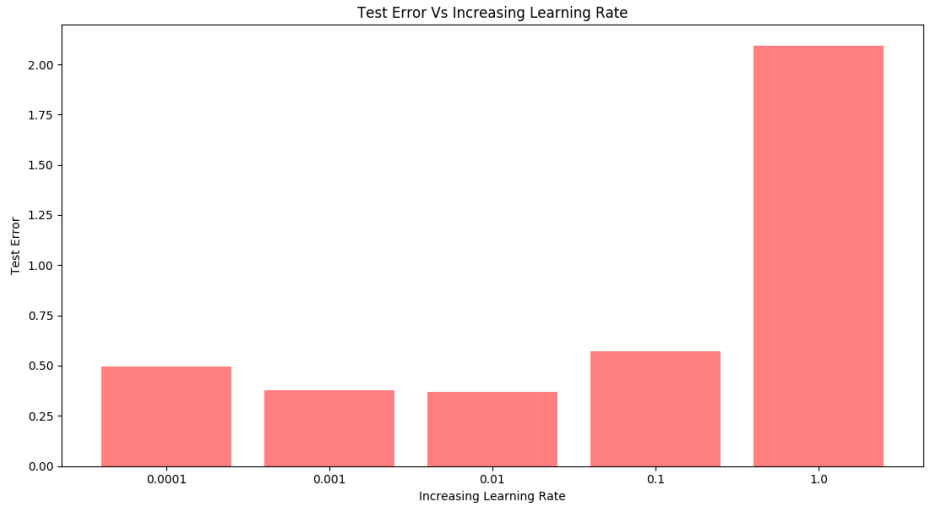

Figure-2-3-2: Test error against increasing learning rates.

Learning rates are a very tricky parameter but this model clearly concludes that very high or very low learning rates will not provide optimal weights, therefore a mid value of 0.01 serves as the best learning rate. Thereafter all models simulated before and after this result will have a **default learning rate of 0.02.**

Model_2.4: ADAM vs SGD as optimizer.

Model_2.4 differs from model_2 in terms of optimizer function as SGD, results of this model are shared in table-11.4.

Table-11.4: Various results of Model_2.4.

Model_2.4 has SGD as optimizer.							
Optimizer	**Model Name**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
ADAM	Model_2	89	29	87	34	86	36
SGD	Model_2.4	85	43	84	43	83	46
Findings from the result	ADAM optimizer performs better than SGD optimizer function.						

Explanation of Results:

SGD or stochastic gradient descent is a variant of traditionally used gradient descent with a distinguish feature of selecting data randomly for optimizing the cost function. Unlike traditional gradient descent algorithm which slowly samples through the data computing partial derivatives, SGD randomly selects data which makes it faster than gradient descent. But in this case Adam optimizer ha better training, validation and test accuracy since ADAM optimizer adapts typically computing adaptive learning rate by updating from first & second moments of the gradients.

Model_2.5: ADAM vs RMSprop as optimizer.

Model_2.5 differs from model_2 in terms of optimizer function as RMSprop, results of this model are shared in table-11.5.

Table-11.5: Various results of Model_2.5.

Model_2.5 has RMSprop as optimizer.							
Optimizer	**Model Name**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
ADAM	Model_2	89	29	87	34	86	36
SGD	Model_2.4	85	43	84	43	83	46
RMSprop	Model_2.5	88	31	87	36	86	39
Findings from the result	ADAM optimizer performs better than RMSprop optimizer function.						

Explanation of Results:

RMSprop performs better than previously used SGD optimizer because, RMSprop uses technique of applying weighted average method while computing the gradient of second moment. Thereby finding more weights than SGD, but ADAM optimizer outperforms RMSprop since it uses techniques of RMSprop along with bias correction for computing adaptive learning rate by updating from first & second moments of the gradients.

Model_3:

Model_3 is completely similar to model_2 with only one difference in number of epochs increased to 40 in hope of getting higher training accuracy and test accuracy above 90%, refer to table-12 for further details.

Table-12: Description of Model_3.

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	158	Relu	158
2	Dense layer	10	Softmax	10

Model_3 compiles successfully with no errors, refer to table-13 that represents results of this model with accuracy and error percentage.

Table-13: Results from Model_3.

Epochs	Model Name	Train data set 54,000 images	Validation set 6,000 images	Test data set 10,000 images

		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	Model_2	89	29	87	34	86	36
40	Model_3	95	10	89	44	89	30
Findings from the result		Improvement of 3% in test accuracy while using 40 epochs therefore finding optimal weights.					

Results from model_3:

Table-13 shows model_2 against model_3 using training, validation and test dataset as parameters for comparing accuracy percentage and error percentage. Please refer figure-M-3-1 and figure-M-3-2 for graphical representation of the same data.

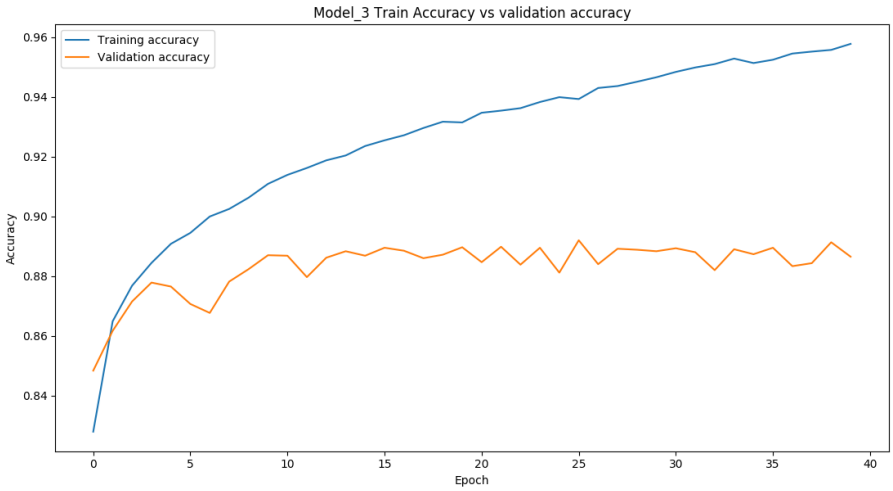

Figure-M-3-1: Representing training accuracy with validation accuracy.

It is very prominent from figure-M-3-1 that both test accuracy and validation accuracy percentage have increased because more number of epochs help in multiple backward and forward passes during back propagation algorithm thus help in finding optimal weights as final solution, ensuring 95% of training accuracy.

Figure-M-3-2: Representing training loss with validation loss.

Figure-M-3-2 shows that training error never saturates, it will keep on decreasing, therefore increasing number of epochs thereafter is of less use whereas, validation error is fluctuating around 35%. Lower percentage of error is a result of higher number of epochs since back propagation algorithm gets more number of forward and backward passes for obtaining optimal weights.

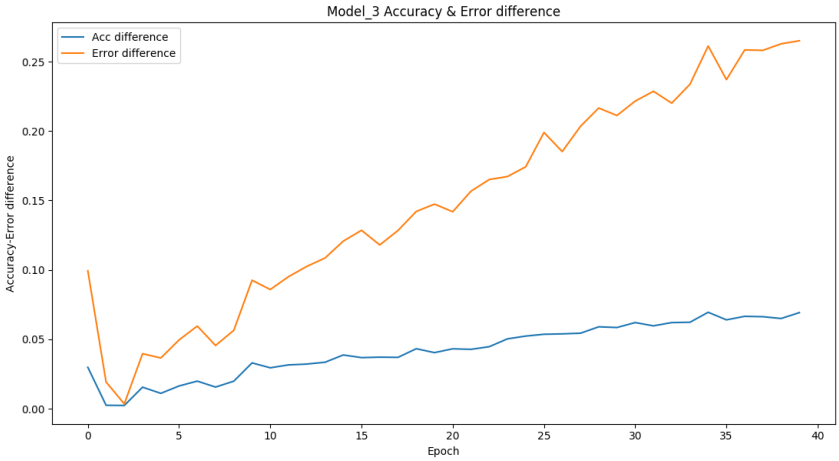

Figure-M-3-3: Representing difference in accuracy and error for training vs validation data.

I have made a separate code for plotting this graphical representation, orange line shows difference between training accuracy percentage & validation accuracy percentage diverging from 0, blue line is difference between training error percentage & validation error percentage diverging from 0 but saturating at 6%, which is a very low number. Difference in accuracy is because validation data set might contain extreme data points.

Detailed discussions about model_3:

1) More epochs help in finding optimal weights, therefore training error is lower and training accuracy has drastically improved up to 95%.

2) More epochs do not guarantee higher test accuracy, from table-13 it is evident that test accuracy has only improved from 86% to 89% whereas epochs have gone up from 5 to 40 which is 8 time more.

3) Table-13 shows 89% test accuracy, thus this network can be used to classify MNIST fashion dataset.

Model_3.1:

Model_3.1 is similar to model_3 with only one difference is that hidden layer's activation function is tanh in this case with respect to relu function assuming tanh will not improve the test accuracy when compared to relu activation function in previous model, refer to table-14 for further details.

Table-14. Description of Model_3.1

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	158	tanh	158
2	Dense layer	10	Softmax	10

Model_3.1 compiles successfully with no errors, refer to table-15 that represents results of this model with accuracy and error percentage.

Table-15. Results from Model_3.1

Epochs	Model Name	Train data set 54,000 images		Validation set 6,000 images		Test data set 10,000 images	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
40	Model_3	95	10	89	44	89	30
40	Model_3.1	95	11	88	38	88	34
Findings from the result		Test accuracy reduces by 1% when relu is replaced with tanh as activation function in hidden layer.					

Results from model_3.1:

Table-15 shows model_3 against model_3.1 using training, validation and test dataset as parameters for comparing accuracy percentage and error percentage. Please refer figure-M-3.1-1 and figure-M-3.1-2 for graphical representation of the same data.

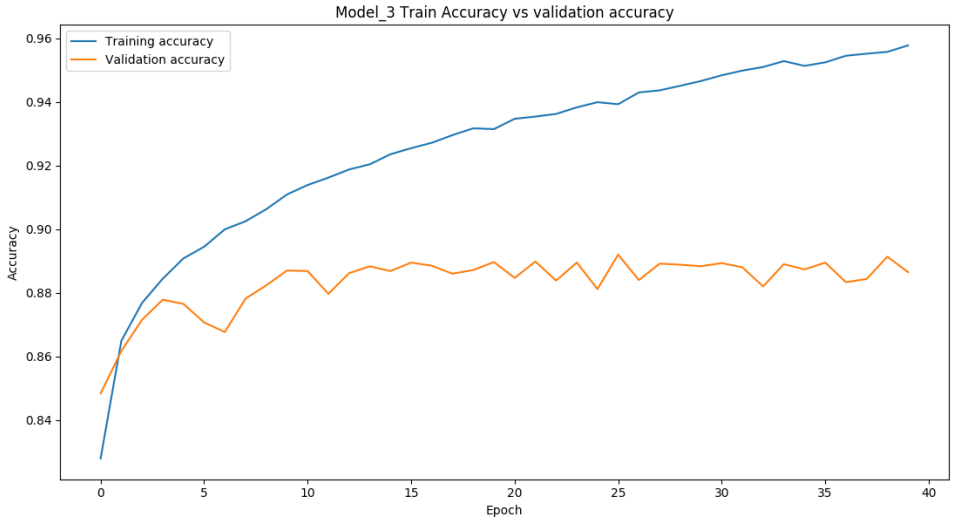

Figure-M-3.1-1 representing training accuracy with validation accuracy.

Figure-M-3.1-1 shows gradual increase in training accuracy and constant validation accuracy with increasing number of epochs since back propagation algorithm gets more number of forward and backward passes to find optimal weights.

Figure-M-3-2 representing training loss with validation loss.

Above explanation is proved in figure-M-3.1-2 as error percentage for both training and validation dataset reduce significantly for same reasons. Table-15 proves that in this case relu is a better activation function than tanh as explained in section below.

Detailed discussions about model_3.1:

1) Tanh activation function gives non linear output where as relu in previous case gives rectified linear output. For given MNIST input tanh turns out to be less productive than relu mostly because softmax function performs better with linear inputs given outputs from hidden layer.

2) It is better to use relu as activation function in hidden layers with respect to tanh as explained above.

Model_4:

This model attempts to increase test accuracy and decrease test error by increasing number of neurons in hidden layer and number of hidden layers. This is in context of model_2 which concluded that implementing above mentioned improvements might help in making this model more accurate, refer to table-16 for further details.

Table-16: Description of Model_4.

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense hidden layer	200	relu	200
2	Dense hidden layer	150	relu	150
3	Dense hidden layer	100	relu	100
4	Dense output layer	10	Softmax	10

Model_4 compiles successfully with no errors, refer to table-17 represents results of this model with accuracy and error percentage.

Table-17: Results from Model_4.

Epochs	Model Name	Train data set 54,000 images		Validation set 6,000 images		Test data set 10,000 images	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
40	Model_3	95	10	89	44	89	30
40	Model_4	95	11	89	51	88	55
Findings from the result		Test accuracy reduces by 1% when model is made more complicated by increasing number of neurons in hidden layer and number of hidden layers.					

Results from model_4:

Table-17 shows model_3 against model_4 using training, validation and test dataset as parameters for comparing accuracy percentage and error percentage. Please refer figure-M-4-1 and figure-M-4-2 for graphical representation of the same data. It is very evident that increasing complexity of this network results in same training accuracy and reduced test accuracy percentage.

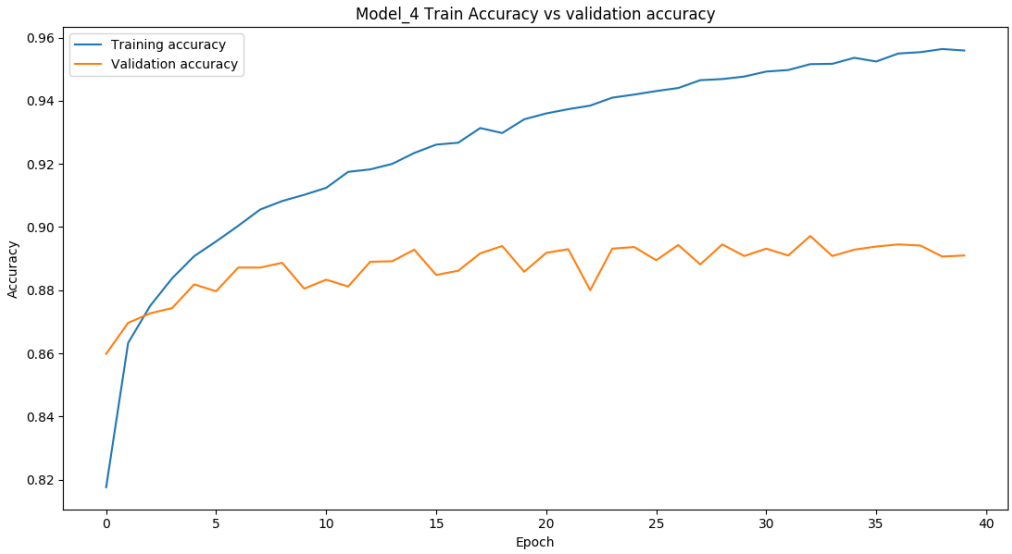

Figure-M-4-1 representing training accuracy with validation accuracy.

Figure-M-4-1 shows that training accuracy percentage slowly increase beyond 95% with increasing number of epochs, because back propagation algorithm gets more number of passes to find optimal weights, whereas validation accuracy percentage saturates around 90% for same reason.

Figure-M-4-2 representing training loss with validation loss

Figure-M-4-2 shows training error continuously going down due to similar explanation given above, as training accuracy increase corresponding error will be minimized. Whereas validation error increases to 50% this suggests that I might be over fitting in this case, therefore I should reduce complexity of my model by reducing number of hidden layers and total number of neurons in each layer.

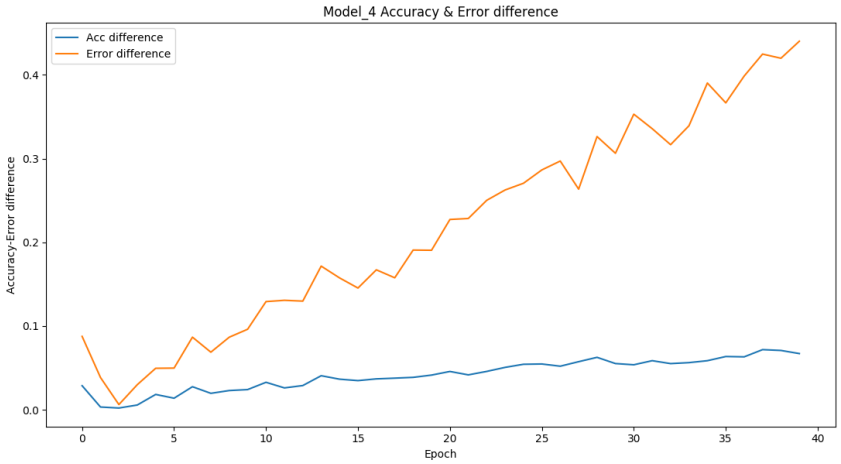

Figure-M-4-3. Representing difference in accuracy and error for training vs validation data.

Figure-M-4-3 represents orange line that shows difference between training accuracypercentage and validation accuracy percentage slowly divergingaway from 0, similarly blue line is difference between training error percentage and validation error percentage diverging from 0 but saturating at 10% with increasing number of

epochs, since complicated network results in high training accuracy but lower validation accuracy. The difference in error percentage is significant around 10% at end of epochs therefore this model is over fitting.

Special Observation:

Since validation error rapidly increases from 40% towards 60% in figure-M-4-2, it is possible that I am over fitting in this model.

Detailed discussions about model_4:

1. Table-17 confirms that complex and deeper models with high number of neurons and hidden layers do not guarantee higher performance than two layered models.

2. Figure-4-2 shows that validation error is high, probably due to over fitting therefore model should be made less complex by reducing number of hidden layers and number of neurons in each layer.

3. Training of a deep neural network in this case takes a longer time and has 1% reduced test accuracy with respect to model_3, therefore model_4 is not recommended for this input dataset.

Model_5:

Model_5 is similar to model_3 with only one difference in hidden layer's activation function is sigmoid with respect to relu function, assuming sigmoid might not be able to improve test accuracy as compared to relu activation function in previous model, refer to table-14 for further details.

Table-18. Description of Model_5

Layer Number	Layer description	Number of neurons per layer	Activation function	Number of weights trained per layer
1	Dense layer	158	sigmoid	158
2	Dense layer	10	Softmax	10

Model_5 compiles successfully with no errors, refer to table-19 that represents results of this model with accuracy and error percentage.

Table-19: Results from Model_5.

Epochs	Model Name	Train data set 54,000 images		Validation set 6,000 images		Test data set 10,000 images	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
40	Model_3	95	10	89	44	89	30
40	Model_5	93	17	88	33	88	36
Findings from the result		Test accuracy reduces by 1% with respect to relu activation function.					

Results from model_5:

Table-19 shows model_3 against model_5 using training, validation and test dataset as parameters for comparing accuracy percentage and error percentage. Please refer figure-M-5-1 and figure-M-5-2 for graphical representation of the same data.

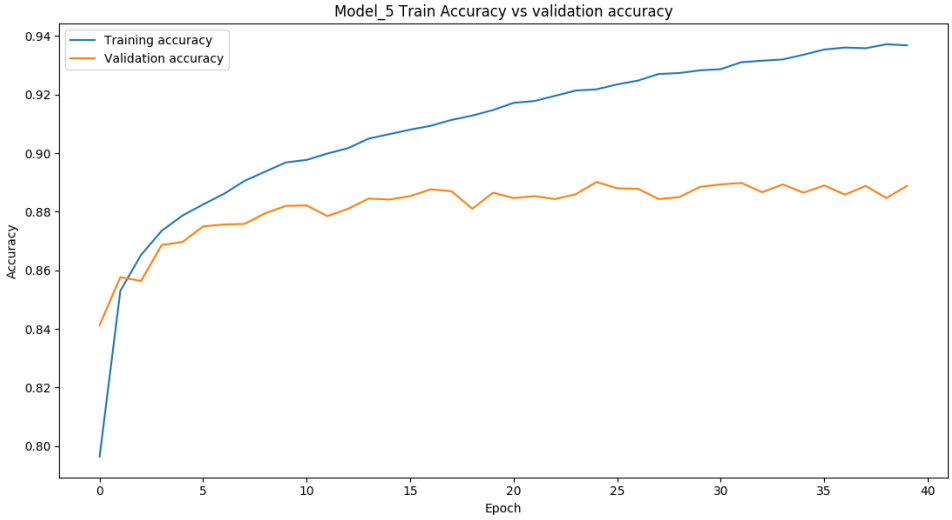

Figure-M-5-1 representing training accuracy with validation accuracy.

Figure-M-5-1 shows gradual increase in training accuracy and constant validation accuracy with increasing number of epochs since back propagation algorithm gets more number of forward and backward passes to find optimal weights.

Figure-M-5-2 representing training loss with validation loss.

Similarly figure-M-5-2 shows gradual descent in training and validation error since with more number of epochs cost function in minimized to higher extent, though sigmoid activation function gives lower test accuracy than relu activation function. This is explained and discussed in section below.

Detailed discussions about model_5:

1. For given set of input relu performs better than sigmoid mostly because softmax activation function performs better with linear inputs. It is know that sigmoid does not provide linear output unlike relu which gives rectified linear output.

2. Table-19 confirms that test accuracy of this model is reduced by 1% as compared to model_3 having relu, whereas test error is increased by 6%, therefore model_5 is not advised to be used for given use case.

Chapter-5: How to select the best model.

In this report I have created 7 different models and represented their training, validation, test data accuracy and error percentage. A cumulative accuracy of all the models can be seen in figure-C-1, where M_1, M_2, M_3, M_3.1, M_4, M_5 correspond to model_1, model_2, model_3, model_3.1, model_4, model_5 respectively.

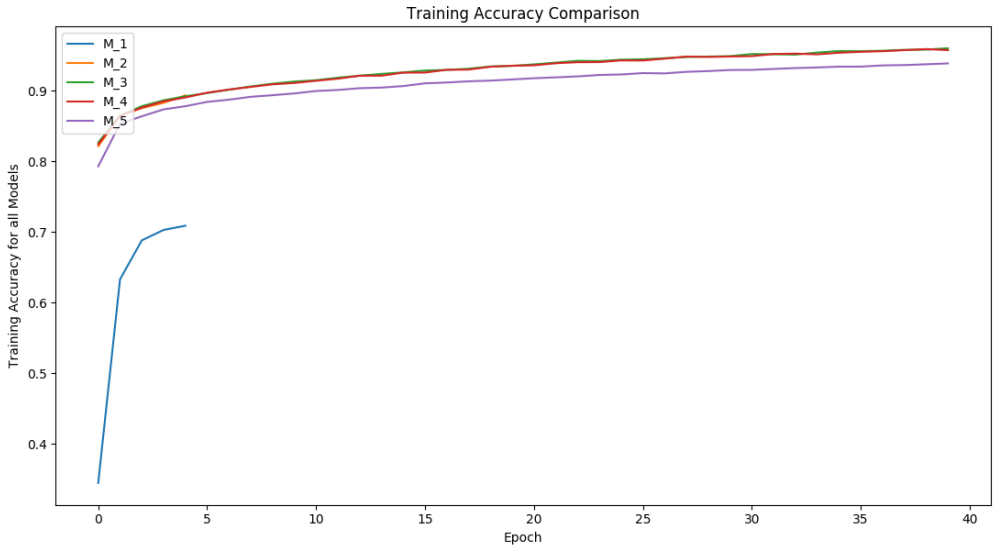

Figure-C-1. Training accuracy of all models.

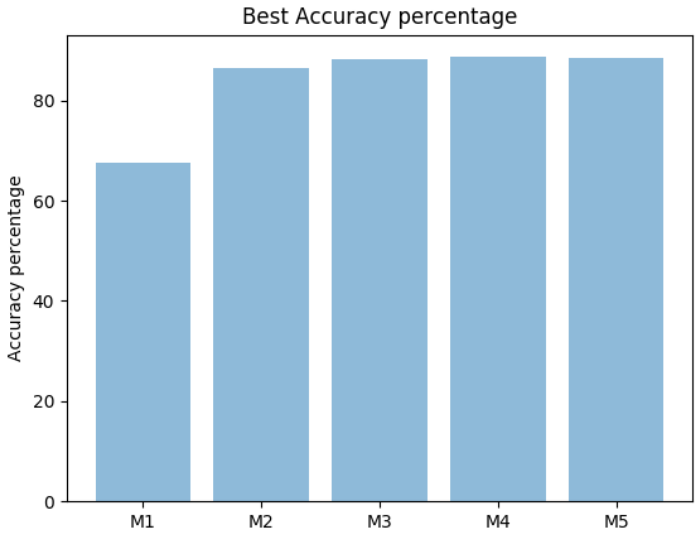

Figure-C-2. Training accuracy percentage as a bar chart.

Figure-C-1 shows that model_3 has maximum training accuracy percentage, whereas model_1 and model_2 do not exist after 5 epochs. Please refer to figure-C-2 that represents the same as a bar chart.

It is evident that model_3 outperforms all other models with 95% of training accuracy which at least 1% higher training accuracy than all other models. Please refer to figure-C-3 for lowest training error percentage.

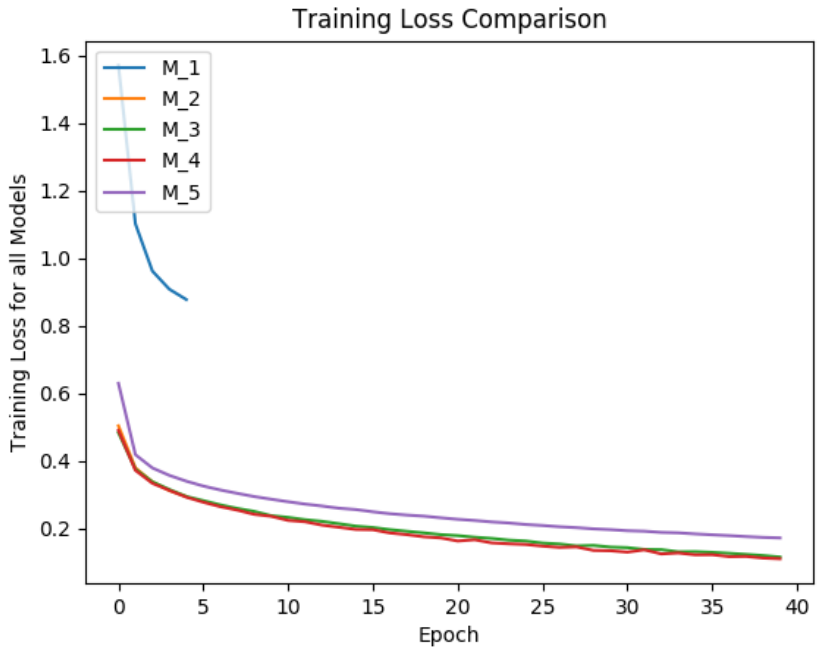

Figure-C-3: Training error percentage.

Figure-C-3 shows that model_3 has lowest training error closely followed by model_4. Please note that all valid reasons for these graphical representations in this section have been discussed and well explained in respective result section of each model. Table-20 below shows comparisons of all 7 models and establishes model_3 as best performing model, this data can be used in determining best performing model.

Table-20. Results from 7 models.

Epochs	**Model Name**	**Train data set 54,000 images**		**Validation set 6,000 images**		**Test data set 10,000 images**	
		Accuracy %	Error %	Accuracy %	Error %	Accuracy %	Error %
5	Model_0	-	-	-	-	-	-
5	Model_1	70	85	71	84	69	86
40	Model_2	89	29	87	34	86	36
4	Model_3	95	10	89	44	89	30
40	Model_3.1	95	11	88	38	88	34
5	Model_4	95	11	89	51	88	55
40	Model_5	93	17	88	33	88	36
Findings from the result		Model_3 has highest training accuracy, highest test accuracy and lowest test error percentage.					

Above displayed graphical data and explanations discussed in previous sections very clearly determine that Model_3 is best performing model with highest training accuracy of 95%, highest test accuracy of 89% and lowest test error percentage of 30%.

The same result can be obtained using figure-C-4, which shows confusion matrix for all relevant models, it should be noted that confusion matrix for model_3.1 is not given here since it is a non-productive extension in model_3.

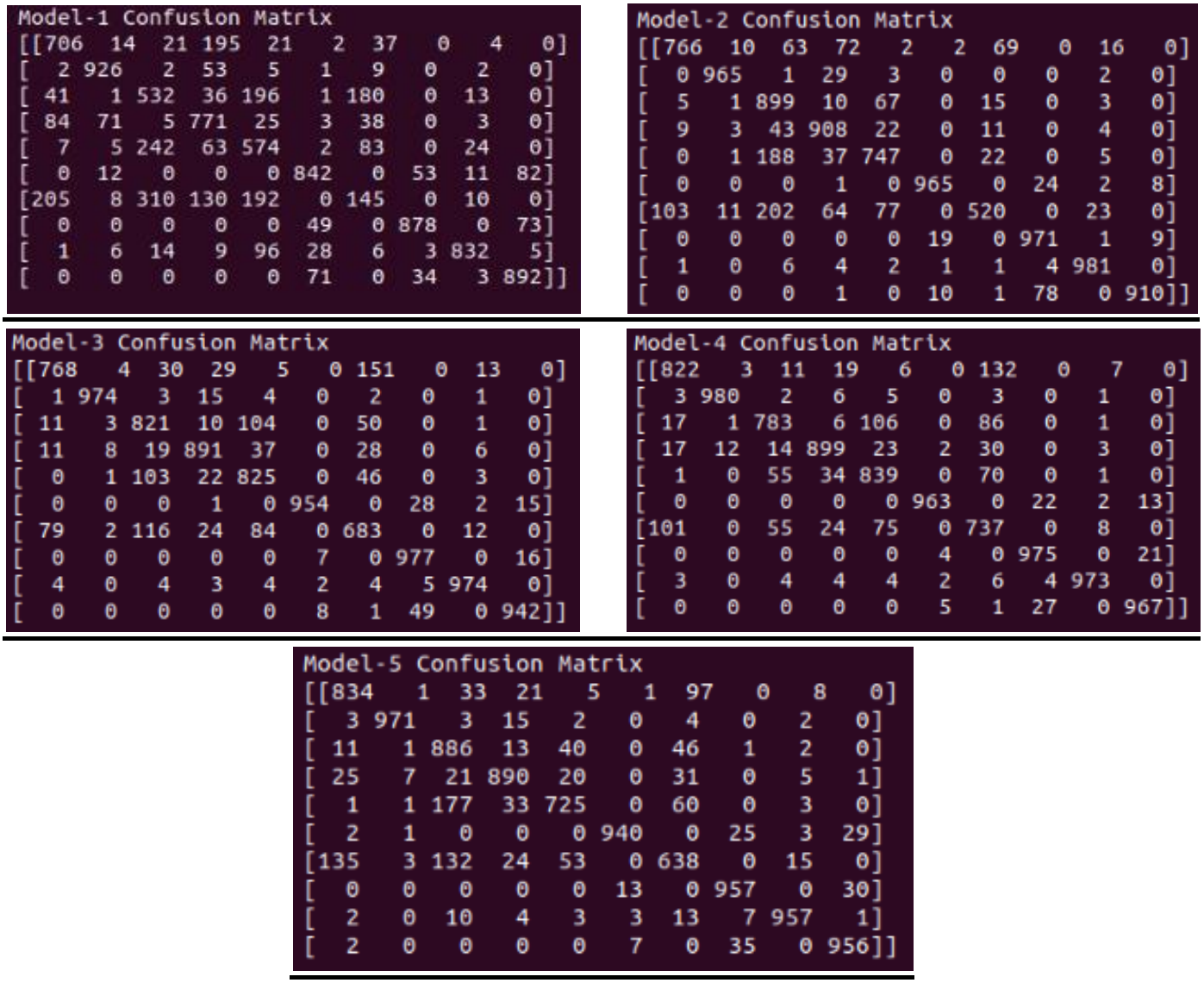

Figure-C-4: Confusion matrix for all relevant models.

It is evident from this figure that maximum number of misclassification occurs for model_1, which is an under fitting model, whereas model_3 has minimum misclassifications. This can be verified by comparing diagonal values of each confusion matrix. Whereas, non-diagonal elements in confusion matrix of Model_3 are mostly 0 or in some cases a relatively smaller number as compared to non-diagonal numbers of other models, thus signifying misclassification rate in model_3 is minimum. Also none of the above established models have 100% prediction rate, since none of the diagonals have all 1,000 elements as true positives. Please refer figure-C-5 and figure-C-6 that represent Precision, Recall, Accuracy & F1 score matrix for all models discussed above.

Following figures helped me in selecting best model, please take a look at the images attached below, which clearly show outputs of various scores for all 10 classes predicted by every model. Precision, accuracy, recall, F1 scores for all 7 models are shown in figure-C-5 and figure-C-6.

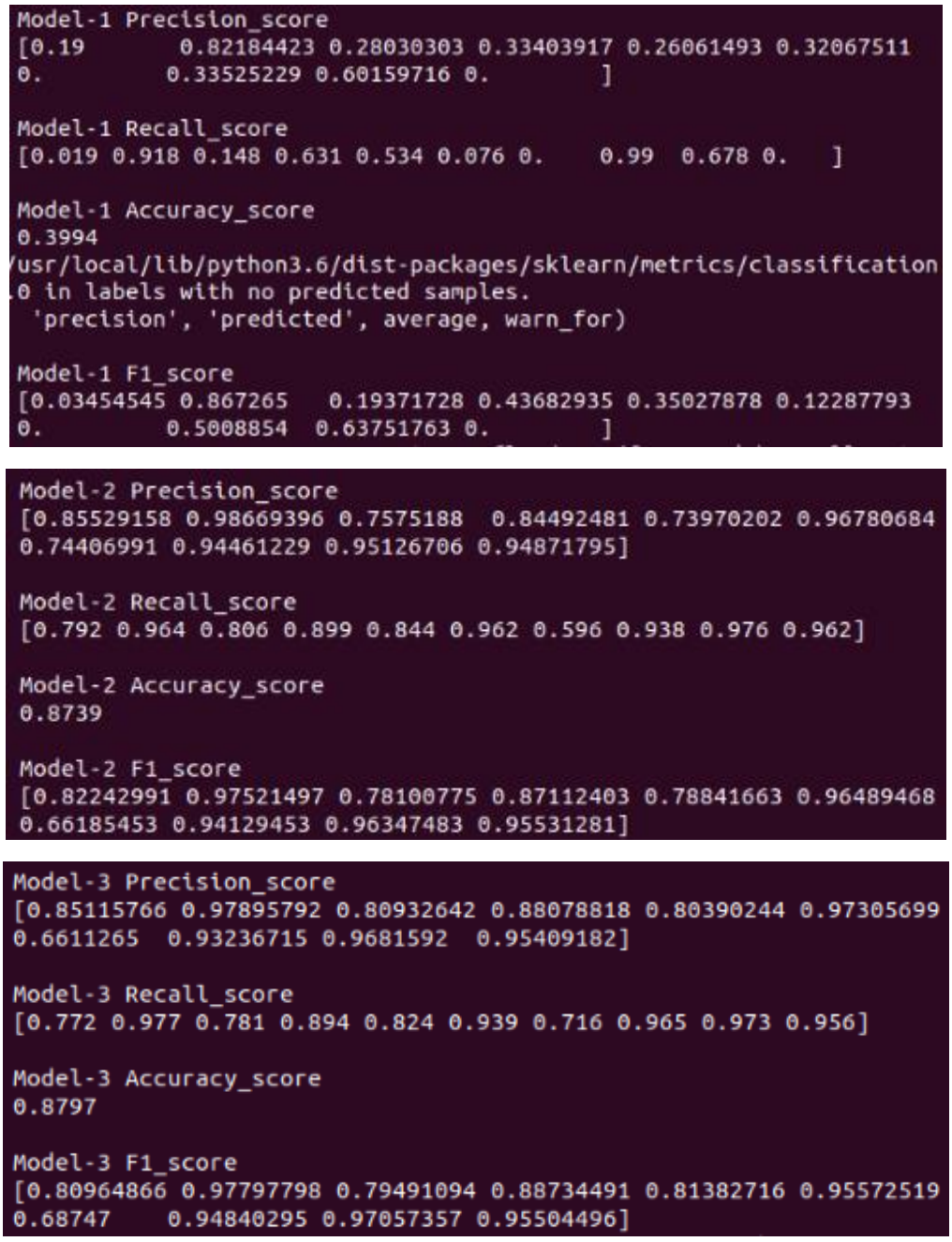

Figure-C-5: Precision, Recall, Accuracy & F1 score matrix for Model-1, Model-2 to Model-3.

Accuracy score: For all the models except model_1 are high, this parameter is not very significant because it can be deceptive in deciding the best model when given input data is unbalanced. Accuracy score should not be considered as a single factor while selecting a particular model, refer figure-C-6 that shows similar parameters.

Precision score: A model having precison score for each class closest to 1 is considered to be a better model, since this score shows ability of the classifier not to label as positive a sample that is negative. For all 7 images shown in figure-C-5 above and figure-C-6 below Model-3 has best precision score.

```
Model-3.1 Precision_score
[0.82815534 0.98585859 0.74616695 0.85849057 0.8556701  0.9742268
0.76781609 0.96369295 0.97297297 0.91682243]

Model-3.1 Recall_score
[0.853 0.976 0.876 0.91  0.747 0.945 0.668 0.929 0.972 0.981]

Model-3.1 Accuracy_score
0.8857

Model-3.1 F1_score
[0.84039409 0.98090452 0.80588776 0.88349515 0.79765083 0.95939086
0.7144385  0.94602851 0.97248624 0.94782609]
```

```
Model-4 Precision_score
[0.84576613 0.98483316 0.81338742 0.89850746 0.84502618 0.96893788
0.69750231 0.94856578 0.97965412 0.956     ]

Model-4 Recall_score
[0.839 0.974 0.802 0.903 0.807 0.967 0.754 0.959 0.963 0.956]

Model-4 Accuracy_score
0.8924

Model-4 F1_score
[0.84236948 0.97938663 0.80765358 0.90074813 0.82557545 0.96796797
0.72465161 0.95375435 0.97125567 0.956     ]
```

```
Model-5 Precision_score
[0.80608365 0.98472505 0.8172973  0.88212181 0.76181818 0.97250509
0.71489818 0.9368932  0.97766497 0.96072508]

Model-5 Recall_score
[0.848 0.967 0.756 0.898 0.838 0.955 0.667 0.965 0.963 0.954]

Model-5 Accuracy_score
0.8811

Model-5 F1_score
[0.82651072 0.97578204 0.78545455 0.88999009 0.79809524 0.96367306
0.69011899 0.95073892 0.97027708 0.95735073]
```

Figure-C-6: Precision, Recall, Accuracy & F1 score matrix for Model-3.1, Model-4 to Model-5.

Recall score: In this particular example recall score should be considered before selecting the final model, since it is more impactful. It shows the ability of a model for presenting how many predictions were correctly classified with respect to total number of actual positives present in dataset, in other words it is a ratio of true positives over number of postives in data set. Recall score of 1 is supposed to be the best, therefore model-3 has comparatively higher recall score.

F1 socre: It is a harmonc mean of recall score and precison score, this parameter is mostly used when data is uneven, therefore F1 scores closer to 1 are preffered. Mostly F1 socre is more impactful that accuracy score as discussed above. In this case it is evident that Model-3 and Model-5 have almost smilar socres.

Learning rate: As discussed in model_2.3, learning rate should neither be low nor very high, in this case model having leaning rate of 0.02 should be selected.

Loss function and optimizer: Models having loss fucntion as sparse categorical cross entropy and adam as optimizer, clearly outperform other models with different attributes.

Chapter-6: Conclusion and Future Work.

Conclusion:

In this report various models were created and analyzed on their performance parameters, listed below are my conclusions extracted from detailed analyses discussed in above sections.

1) If model is under fitting then number of hidden layers and number of neurons should be increased.

2) Highly complicated network does not guarantee best performance.

3) Adam optimizer works better than SGD and RMSprop optimizers.

4) MSE performs poorly with respect to sparse categorical cross entropy loss function.

5) Learning rate for given problem statement should neither be very high nor very low, therefore fixed at 0.02.

6) If validation accuracy is very low in comparison to training accuracy, simultaneously training accuracy itself is above 90%, then either validation data set has completely distinct data points or model is over fitting.

7) Higher number of epochs will increase the training accuracy but it does not guarantee high test accuracy.

8) It is important to determine when to stop training your model, so that neither it under fits nor it over fits.

9) Relu activation function in hidden layer performs better than sigmoid and tanh functions when softmax function is used in output layer.

10) While using softmax activation function in output layer, number of neurons in this layer should be exactly equal to total number of distinct labels.

11) Accuracy score should not be the one and only criteria while selecting best model.

12) Recall score and F1 score should always be considered before selecting a model for final deployment.

After experimenting with various models it can be concluded that model-3 has best recall score, highest test data accuracy, highest training accuracy, limited complexity, smaller training time and high F1 score therefore model-3 is the best performing model.

Future Work:

Convolution neural networks will perform better in image analysis tasks, therefore a new model should be created following CNN and results of this model should be compared to model_3.