Classification using Neural Network

Aditya Bansal
UB Person No.-50291692
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
adityaba@buffalo.edu

Abstract

The aim of this project is to do classification of our data using Single Layer neural network, Multi-Layer Neural Network and Convolution Neural Network. It is for a ten-class problem and we are doing it using the dataset of Fashion-MNIST clothing images. We are designing single layer neural network from scratch for this project and for multi-layer neural network and convolution neural network we are using library Keras.

1 Introduction

1.1 Neural Network model

In Neural network a layer is made up of multiple neurons in a way that output of one neuron is given as input to neurons of another layer. Below is an example of a neural network:

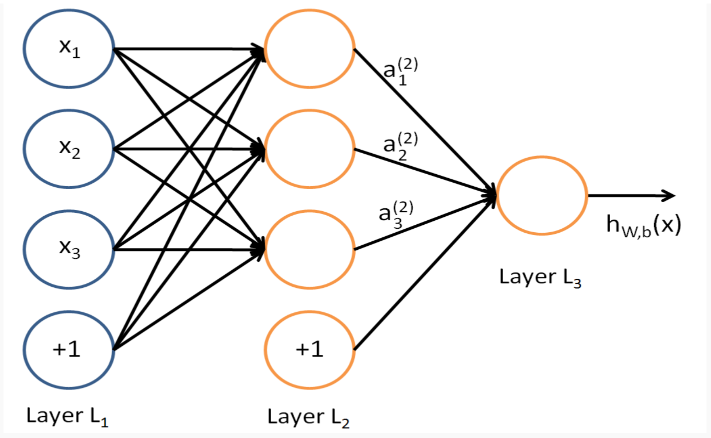

Figure 1. Neural Network Model

In this figure, we have used circles to also denote the inputs to the network. The circles labeled "+1" are called bias units and correspond to the intercept term. The leftmost layer of the network is called the input layer, and the rightmost layer the output layer (which, in this example, has only one node). The middle layer of nodes is called the hidden layer, because its values are not observed in the training set. We also say that our example neural network has 3 input units (not counting the bias unit), 3 hidden units, and 1 output unit.

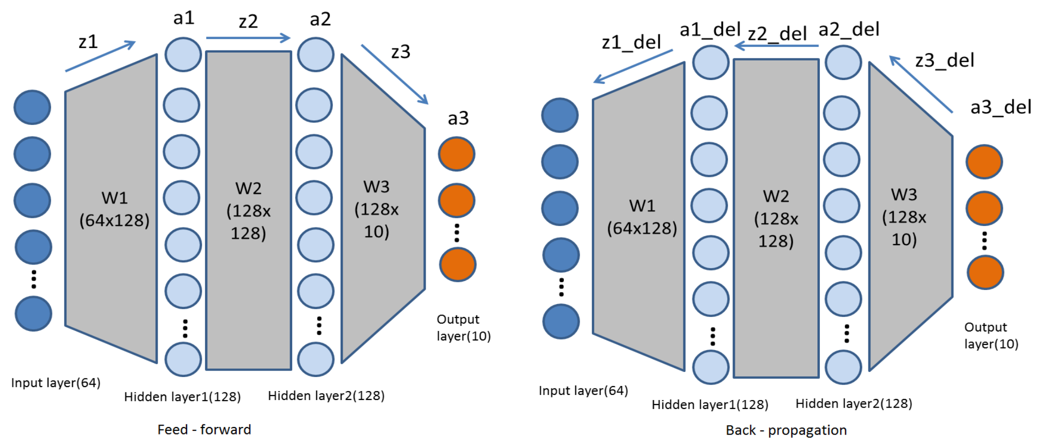

Figure 2. Feed Forward and Back Propagation

Feed-forward process is quite simple. (input x weights) + bias computes z and it is passed into the layers, which contains specific activation functions. These activation functions produces the output a. The output of the current layer will be the input to the next layer and so on. As you could see, the first and second hidden layer contains sigmoid function as the activation function and the output layer has softmax as the activation function.
Fundamentally ***Back-propagation*** computes the error from the output of feed-forward and the true value. This error is back propagated to all the weight matrices through computing gradients in each layer and these weights are updated.

1.2 Convolution Neural Network

Convolutional Neural Networks have a different architecture than regular Neural Networks. Convolutional Neural Networks are a bit different. First of all, the layers are organized in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.
CNN is composed of two major parts:
Feature-Extraction:

In this part, the network will perform a series of convolutions and pooling operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognize its stripes, two ears, and four legs.

Classification:
Here, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is.

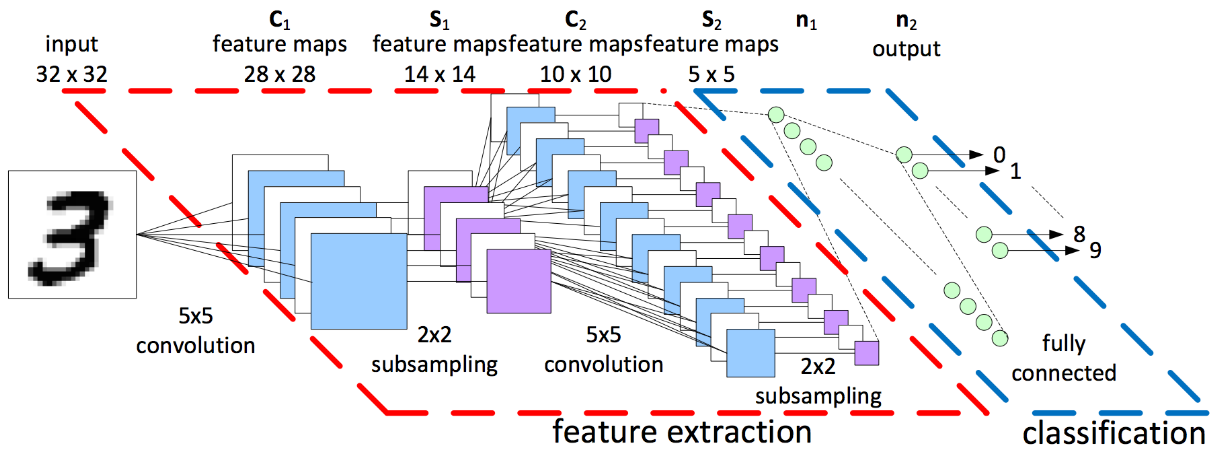

Figure 3. Convolution Architecture

In our project we have taken MNSIT dataset. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Goals of our project are: -

- Train using Neural Network with One Hidden Layer Use Gradient Descent for neural network to train the model using a group of hyperparameters. (Code from scratch in python)
- Train using Multi-Layer Neural Network with high level Neural Network library, Keras using a group of hyperparameters.
- Train using Convolution Neural Network with high level Neural Network library, Keras using a group of hyperparameters.
- Tune hyper-parameters: Validate the classfication performance of your model on the validation set. Change your hyper-parameters and repeat the step. Try to find what values those hyperparameters should take so as to give better performance on the testing set.
- Test your machine learning scheme on the testing set: For steps 3, 4 and 5: After tuning the hyper-parameters, fix your hyper-parameters and model parameter and test your models performance on the testing set. This shows the ultimate effectiveness of your models generalization power gained by learning.

2 Dataset Definition

Context:
Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Zalando intends Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.
The original MNIST dataset contains a lot of handwritten digits. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. "If it doesn't work on MNIST, it won't work at all", they said. "Well, if it does work on MNIST, it may still fail on others."
Zalando seeks to replace the original MNIST dataset

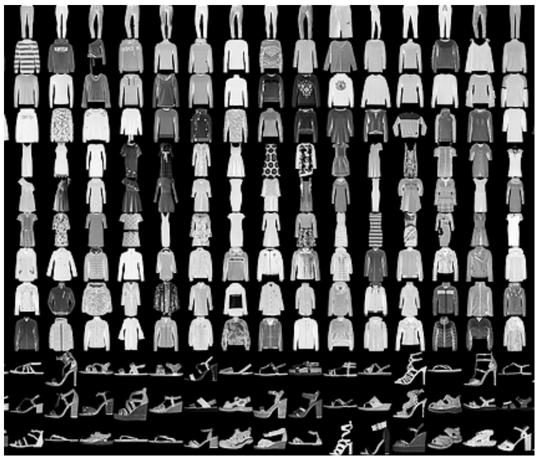

Figure 4: Example of how the data looks like.

Content:
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above) and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.
To locate a pixel on the image, suppose that we have decomposed x as x = i * 28 + j, where i and j are integers between 0 and 27. The pixel is located on row i and column j of a 28 x 28 matrix.

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Labels:
Each training and test example are assigned to one of the following labels:
0 T-shirt/top
1 Trouser
2 Pullover
3 Dress
4 Coat
5 Sandal
6 Shirt
7 Sneaker
8 Bag
9 Ankle boots

Each row is a separate image
Column 1 is the class label.
Remaining columns are pixel numbers (784 total).
Each value is the darkness of the pixel (1 to 255)

3 Pre-Processing

3.1 Extract feature values and labels from the data: Fashion MNIST dataset is downloaded and processed into a Numpy array that contains the feature vectors and a Numpy array that contains the labels using fashion mnist reader notebook present inside the scripts folder.

3.2 Data Partitioning: The Fashion MNIST dataset is originally partitioned into a training set and a testing set. You will use this partition and train your model on the training set.We have divided our data in four groups X-train,y-train,X-test,y-test

3.3 Data Normalization: Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values.
All the data in X-train and X-test is in the form of 0-255 RGB values of every bit of image so to normalize this data we have divided it by 255.0.
Also our y-train and y-test are of the shapes (60000*1) and (10000*1) respectively, but we have to process it with our predicted values which will be of the form (60000*10) and (10000*10) because we are doing classification into 10 categories using Softmax function. So, by using function keras.utils.to_categorical() we have transforms data into (60000*10) and (10000*10).

3.4 Reshaping of Data (Only for Convolution Neural Network): Train and test images (28 x 28)
We reshape all data to 28x28x1 3D matrices. Keras needs an extra dimension in the end which correspond to channels. Our images are gray scaled, so it uses only one channel.

4 Architecture

4.1 Train using Neural Network with One Hidden Layer (From Scratch):
In *feed forward network* we first find z1_train by simply passing values from linear equation and then we passed the output from an activation function. The output from current layer becomes input for another layer. In our project for first layer we are using sigmoid function and for last layer we are using SoftMax for classification into ten classes.
First, we will find z1_train by doing dot product of our training data and weights and adding bias:

$$z^{[1]} = w^{[1]}.x + b^{[1]}$$

where $w^{[1]}$= weight 1
x = training set
$b^{[1]}$= bias 1
$z^{[1]}$= predicted values for the input x (layer 1)

Passing our z1_train to first activation function which in first model we are using sigmoid also we could have used other activation functions like tanh or relu.

$$a^{[1]} = \sigma\left(z^{[1]}\right)$$

Now the result which we got that is p1_train is predicted value from first hidden layer now we will do the same for next feed forward layer but this time we are going to use softmax in place of sigmoid because we have to classify our data in 10 classes and sigmoid only classify data into binary form.

$$z^{[2]} = w^{[2]}.a^{[1]} + b^{[2]}$$

where
$w^{[2]}$= weight 2
x = training set
$b^{[2]}$= bias 2
$z^{[2]}$= predicted values for the input as layer 1 (output layer)

$$a^{[2]} = softmax(z^{[2]})$$

Defining Sigmoid and Softmax functions which we used in first and second layers respectively.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$
$$softmax(z)_i = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}}$$

In *Backward Propagation* we compute the error in our forward propagation and then propagates it backwards which means we change our weights and biases according to the error we calculated from previous forward propagation.

Cost-Entropy Function

$$dz^{[2]} = \left(a^{[2]} - y\right).$$

$$dw^{[2]} = dz^{[2]}a^{[1]}$$

Sigmoid Derivative

$$sigmoid_{derivative} = a^{[1]}\left(1 - a^{[1]}\right)$$

$$dz^{[1]} = \left(a^{[2]} - y\right)w^{[2]} * sigmoid_{derivative}$$

$$dw^{[1]} = dz^{[1]}x$$

Adjusting our weights and biases by back propagation

$$db^{[1]} = \sum dz^{[1]}$$
$$db^{[2]} = \sum dz^{[2]}$$

$$w^{[2]} = w^{[2]} - \propto dw^{[2]}$$

$$w^{[1]} = w^{[1]} - \propto dw^{[1]}$$

$$b^{[2]} = b^{[2]} - \propto db^{[2]}$$

$$b^{[1]} = b^{[1]} - \propto db^{[1]}$$

where
$a^{[1]}, a^{[2]}$ = predicted values for each layer
y = actual target value
x = training set
α = learning rate

Loss Function

$$L(a^{[2]}, y) = -\sum y \log a^{[2]}$$

where
y = training set target
$a^{[2]}$ = predicted output

4.2 Train using Multi-Layer Neural Network (Using Keras):
We are doing same operations as we did in 4.1 but here, we are doing it using library Keras.
model = Sequential()
# The sequential API allows you to create models layer-by-layer for most problems.
model.add(Dense(128, input_dim=X_train.shape[1], activation='sigmoid'))
# Dense implements the feed forward layer in this we are using 128 nodes at every layer.
model.add(Dense(128, activation='relu'))
# Hidden Layer 1-Using relu activation and 128 nodes.
model.add(Dense(128, activation='relu'))
# Hidden Layer 2-Using relu activation and 128 nodes.
model.add(Dense(10, activation='softmax'))
# Final output layer-Using softmax activation and classifying into 10 labels.
model.compile(optimizer=Adadelta(), loss='categorical_crossentropy', metrics=['categorical_accuracy'])
# Compiling our model and using cross entopy as our loss function.
history=model.fit(X_train, Y_train,validation_data=(X_test,Y_test) ,epochs=30, batch_size=64)
# Training our data using epochs =30 and batch size =64.

4.3 Train using Convolution Neural Network (Using Keras):

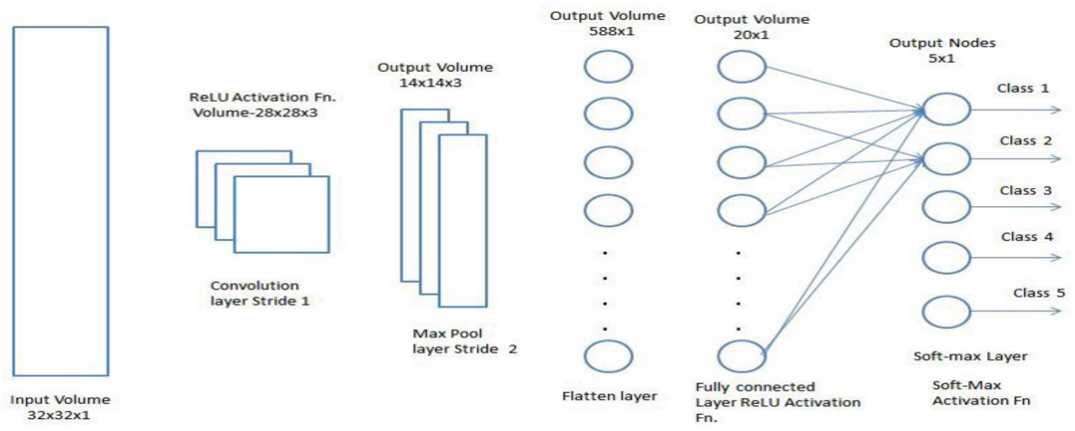

Figure 5. Convolution Neural Network Architecture

Convolution Accuracy steps :-
1. Feature Extraction:Convolution
# Reshaping our data
X_train_conv = X_train_conv.reshape(-1, 28,28, 1)
X_test_conv = X_test_conv.reshape(-1, 28,28, 1)
model_c = Sequential()
# The sequential API allows you to create models layer-by-layer for most problems.
model_c.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model_c.add(Activation('relu'))
model_c.add(MaxPooling2D(pool_size=(2,2)))

```
model_c.add(Conv2D(64, (3,3)))
model_c.add(Activation('relu'))
model_c.add(MaxPooling2D(pool_size=(2,2)))
model_c.add(Flatten())
model_c.add(Dense(64))
model_c.add(Dense(10))
model_c.add(Activation('softmax'))
model_c.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.Adam(),metrics=['accuracy'])
```

4.4. Evaluation of results

4.4.1 Plot graph of training loss vs number of epochs while training on each classifier (Neural Network with single hidden layer, multi-layer neural network and convolutional neural network).

4.4.2 For each classifier evaluate solution on the test set using classification accuracy:

$$Accuracy = \frac{N_{correct}}{N}$$

where $N_{correct}$ = number of corrected classified data samples

N = total number of samples of the set.

4.4.3 Constructing a confusion matrix for each classifier and observe the relative strengths and weaknesses.

5 Results

We have taken results for different value of hyperparameters after comparing results of loss function and accuracy score of train data and validation data. Also, we tried to observe the difference in results by using different types of activation factions.

5.1. For One Layer Neural Network:-

A. Hyperparameters- Epochs=5, Learning Rate=0.75, Nodes=256, Batch size = 500

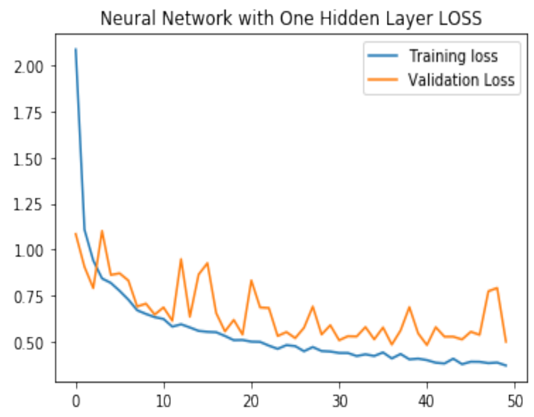

Figure 6

Train Accuracy Percent :-86.97

Test Accuracy Percent :-82.09

B. Hyperparameters- Epochs=70 ,Learning Rate=1 ,Nodes=256 , Batch size = 500

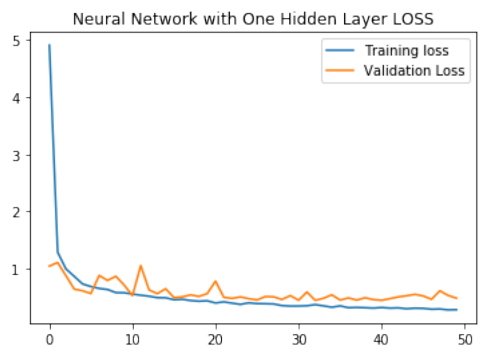

Figure. 7

Train Accuracy Percent:-88.37

Test Accuracy Percent :-84.15

Confusion Matrix;-

```
[[612   2   3   7   0   0  46   0   3   0]
 [  0 963   0  12   0   0   2   0   0   0]
 [ 15   3 730  10 104   1  76   0   2   0]
 [ 36  21  10 860  27   0  27   0   8   1]
 [  3   2  77  25 692   0  39   0   2   0]
 [  3   0   1   1   0 940   0  33   5  12]
 [322   8 170  77 174   0 801   0  28   1]
 [  0   0   1   0   0  35   0 915   4  30]
 [  9   1   8   8   3   1   9   0 946   0]
 [  0   0   0   0   0  23   0  52   2 956]]
```

C. Hyperparameters- Epochs=70, Learning Rate=1, Nodes=256, Batchsize=500

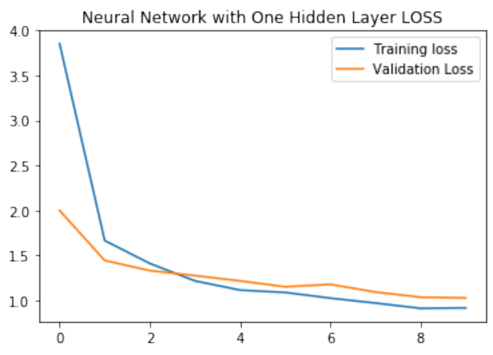

Figure. 8

Train Accuracy Percent :- 76.1

Test Accuracy Percent :- 75.85

5.2. For Multi-Layer Neural Network

A. Hyperparameters- Epochs=50, Batchsize=500, Activation function- Relu

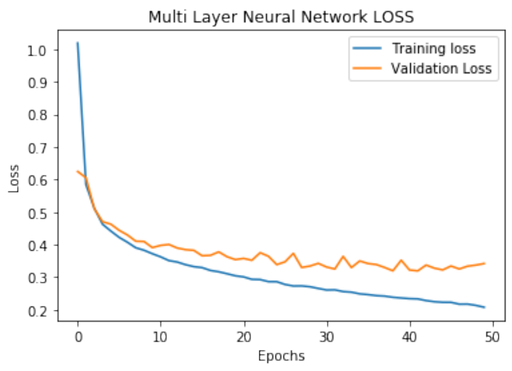

Figure 9

Train Accuracy Percent of Multilayer Neural Network:- 92.19
Test Accuracy Percent of Multilayer Neural Network:- 87.24

B. Hyperparameters- Epochs=10, Batchsize=64, Activation function-Sigmoid

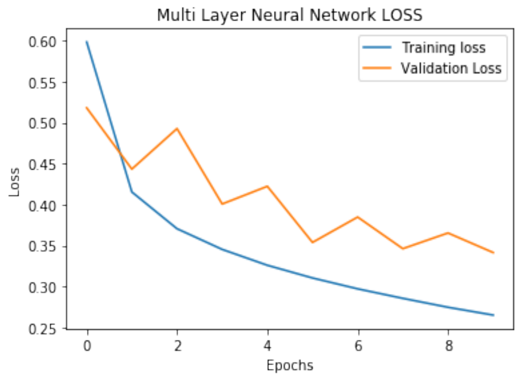

Figure. 10

Train Accuracy Percent of Multilayer Neural Network:- 90.07
Test Accuracy Percent of Multilayer Neural Network:- 87.24

C. Hyperparameters- Epochs=50, Batchsize=500, Activation function-Relu

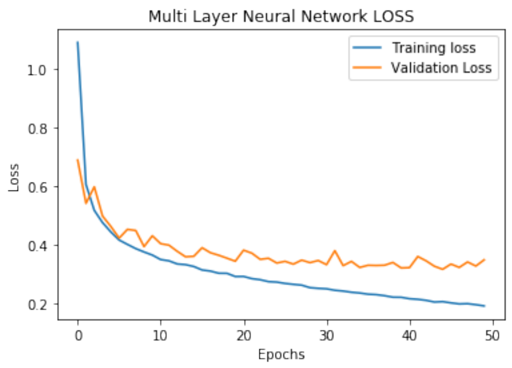

Figure. 11

Train Accuracy Percent of Multilayer Neural Network:- 90.07
Test Accuracy Percent of Multilayer Neural Network:- 88.56

Confusion Matrix:

```
[[797   2  10  16   0   0  78   0   5   1]
 [  2 974   1  15   2   0   2   0   1   0]
 [ 19   1 779  15  61   0  68   0   1   0]
 [ 11  16   8 864  20   0  19   0   4   0]
 [  3   3 121  39 852   0  61   0   3   0]
 [  2   0   1   0   0 964   0  25   6   4]
 [158   3  78  44  61   0 763   0   6   0]
 [  0   0   0   0   0  14   0 939   3  19]
 [  8   1   2   7   4   1   9   1 971   0]
 [  0   0   0   0   0  21   0  35   0 976]]
```

5.3. For Convolution Neural Network

A. Hyperparameters- Epochs=30, Batchsize=256, Activation function- Tanh

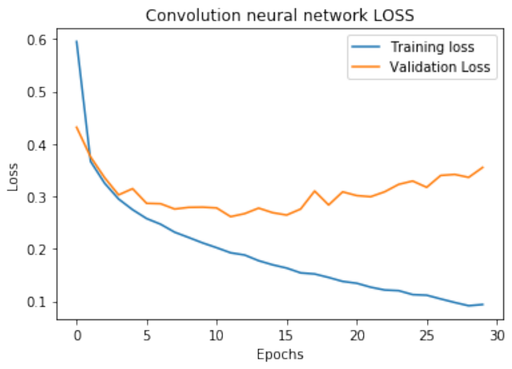

Figure 12

Train Accuracy:- 87.45
Test Accuracy:- 89.39

B. Hyperparameters- Epochs=50, Batchsize=1000, Activation function- Relu

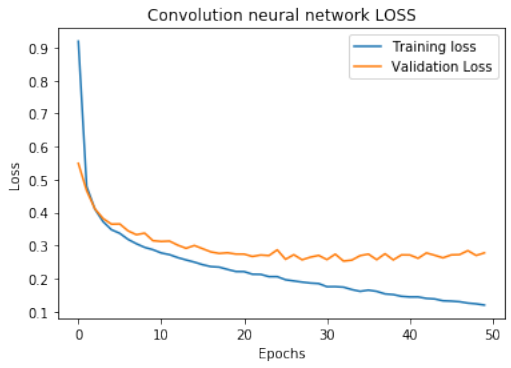

Figure. 13

Train Accuracy:- 95.99
Test Accuracy:- 90.86

Confusion Matrix:

```
[[816   3  11   7   1   0  89   0   2   1]
 [  1 981   1   4   0   0   0   0   0   0]
 [ 30   2 887  13  47   0  73   0   4   0]
 [ 39  11  11 950  33   0  32   0   4   0]
 [  5   0  44  14 866   0  87   0   1   0]
 [  1   0   0   0   0 972   0   9   1   8]
 [102   3  45  12  52   0 715   0   7   0]
 [  0   0   0   0   0  20   0 984   2  53]
 [  6   0   1   0   1   2   4   0 979   1]
 [  0   0   0   0   0   6   0   7   0 937]]
```

C. Hyperparameters- Epochs=10, Batchsize=256, Activation function- Relu

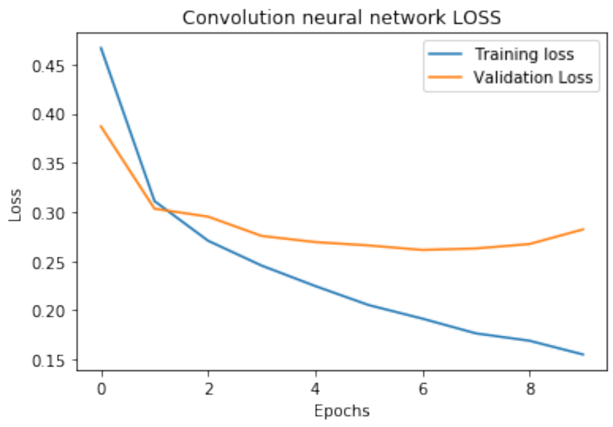

Figure. 14
Train Accuraccy :- 90.06
Validation Accuracy :- 86.76

6. Conclusion

We compared three models of neural network and checked these models for different hyperparameters our aim is to maximize the accuracy score for our test data.For first model we got best accuracy score of 84.15% for second model best accuracy percent is 88.56% and for third model which is convolution neural network best accuracy is 90.86%

7. References

[1] THE IMPORTANCE OF LOGISTIC REGRESSION IMPLEMENTATIONS IN THE TURKISH LIVESTOCK SECTOR AND LOGISTIC REGRESSION IMPLEMENTATIONS/FIELDS Murat KORKMAZ1 *, Selami GÜNEY2, Şule Yüksel YİĞÎTER3. J.Agric. Fac. HR.U., 2012, 16(2): 25-36

[2] https://towardsdatascience.com/neural-networks-from-scratch-easy-vs-hard-b26ddc2e89c7

[2] https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148, Introduction to Logistic Regression Ayush Pant.

[3] https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[5] Machine Learning week 1: Cost Function, Gradient Descent and Univariate Linear Regression, Lachlan Miller

[6] An Introduction to Logistic Regression Analysis and Reporting CHAO-YING JOANNE PENG KUK LIDA LEE GARY M. INGERSOLL Indiana University-Bloomington.

[7] Pattern Recognition and Machine Learning, Christopher M Bishop.

[8] Accuracy, Precision, Recall or F1? Koo Ping Shung, https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9