K-Class classification using Neural Networks and CNN

Priya Rao
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
prao4@buffalo.edu

Abstract

This paper describes a k-class classification performed by means of both neural networks – one with a single hidden layer and another with multiple hidden layers – and convolutional neural networks. The activation functions of sigmoid (used in logistic regression), softmax and ReLU have been used in achieving the accuracies of each of the three methods. The corpus being used is that of the MNIST fashion dataset which consists of 10 target categories.

1 Introduction

Neural Networks are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. For example, in image recognition, they might learn to identify images that consist of trees by analyzing example images that have been manually labeled as "tree" or "no tree" and using the results to identify trees in other images. They do this without any prior knowledge of trees, for example, that they are tall, have leaves and have a trunk. Instead, they automatically generate identifying characteristics from the examples that they process. The problem of k-class classification described here demands for the data to be classified into k different classes by the neural network. There are three parts discussed here to achieve the above result. The first part covers classification using a single hidden layer (a special case of neural networks which has only one layer) between the input and the output layer. The second part elaborates on using a multi-layer perceptron (MLP) model of neural networks which would consist of several hidden layers. The last part explains convolutional neural networks and how this achieves the best result for the aforementioned dataset.

1.1 Part 1 – Single Hidden Layer

Although very similar to logistic regression, the single hidden layer perceptron consists of applying the activation function of sigmoid and softmax as well as considers a 2D matrix of weights. The equations for the forward propagation of this model is as seen below:

Forward propogation for single hidden layer NN:

$$z_1 = \theta_1^T X + b_1 \tag{1}$$

$$a_1 = \frac{1}{1+e^{-z_1}} \tag{2}$$

$$z_2 = \theta_2^T X + b_2 \tag{3}$$

$$a_2 = \frac{\exp(x_k)}{\sum_j^n \exp(x_j)} \tag{4}$$

$$Loss(\theta) = \frac{-1}{m}\sum_{i=1}^{m} y^{(i)} \log(a_2^{(i)}) \tag{5}$$

The sigmoid function takes a hypothesis function as the input parameter which is calculated using Equation (1). Here Θ_1 is a 2D vector of weights of dimensionality m x n where m is the total number of feature vectors (in this case, M = 784 since we have 784 pixels) and n is the number of nodes in the hidden layer, X is the dataset in consideration (Applicable for both training and validation set) and b_1 is the bias which is a scalar value.

Similarly, the softmax function takes a hypothesis function as the input parameter which is calculated using Equation (3). Here Θ_2 is a 2D vector of weights of dimensionality n x p where n is the number of nodes in the hidden layer and p is the number of output classes (in this case it is 10), X is the dataset in consideration (Applicable for both training and validation set) and b_2 is the bias which is a scalar value.

The loss function (Equation (5)) in turn is a summation of cross entropy loss using y and the previously calculated a_2. Taking log for both cost functions assists during the calculation of the gradient. The above equations combined can demonstrate the loss at each epoch and hence comprise of the forward propagation which is performed for both the training and the validation dataset. The number of nodes that can be present in the hidden layer here can be adjusted as needed and hence along with the number of epochs it is another hyperparameter.

Backward propogation for single hidden layer NN:

$$\delta a_2 = \frac{\delta}{\delta a_2} Loss \tag{6}$$

$$\delta z_2 = \frac{\delta}{\delta z_2} Loss \tag{7}$$

$$\delta z_2 = a_2 - y \tag{8}$$

$$\delta w_2 = (\frac{\delta}{\delta z_2} Loss)(\frac{\delta}{\delta w_2} z_2) \tag{9}$$

$$\delta w_2 = (a_2 - y)a_1 \tag{10}$$

$$\delta a_1 = (\frac{\delta}{\delta z_2} Loss)(\frac{\delta}{\delta z_2} a_1) \tag{11}$$

$$\delta a_1 = (a_2 - y)w_2 \tag{12}$$

$$\delta z_1 = \delta a_1 \frac{\delta}{\delta z_1} a_1 \tag{13}$$

$$\delta z_1 = (a_2 - y)w_2 a_1(1 - a_1) \tag{14}$$

$$\delta w_1 = \delta z_1 \frac{\delta}{\delta w_1} z_1 \tag{15}$$

$$\delta w_1 = (a_2 - y)w_2 a_1(1 - a_1)X \tag{16}$$

$$w_2 = w_2 - \eta \delta w_2 \tag{17}$$

$$w_1 = w_1 - \eta \delta w_1 \tag{18}$$

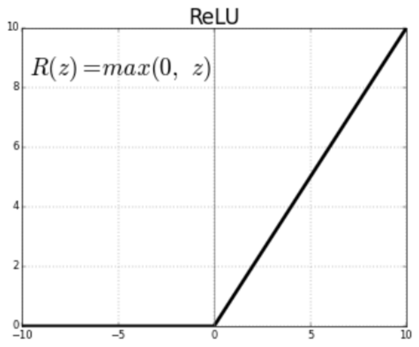
Back propagation involves updating the weights which is done using gradient descent. Gradient descent allows for the losses to be minimized by converging to the local minimum based on the learning rate. This is done by finding the partial derivative of the loss function. The equations from (6) to (18) all are the steps for the back propagation required from the output layer to the hidden layer as well as from the hidden layer to the input layer. After the weight deltas are calculated, the weights are updated accordingly with eta being the learning rate.

The activation functions of sigmoid and softmax are non-linear functions. The key difference between the two of them are that while the sigmoid activation function is used for two class classification, the softmax activation function is used for multi-class classification.

1.2 Part 2 – Multi-Layer Perceptron

The multi-layer perceptron is basically an extension of the single hidden layer neural network. The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating functions. Since MLPs are fully connected, each node in one layer connects with a certain weight to every node in the following layer.

Here every layer can perform a different activation function and with the help of regularization methods, the predictions of the NN can be improved by a significant factor. In this experiment, ReLU and softmax have been used. The ReLU activation is the most widely used activation function in the field of neural networks and deep learning and its graph can be traced as below:



By using relu as the activation function for the hidden layers in the MLP, maximum accuracy was achieved.

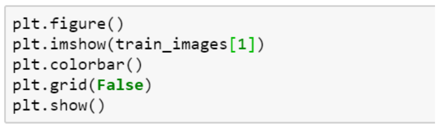
1.3 Part 3 – Convolutional Neural Network

CNNs are the regularized versions of multi-layer perceptrons. Which means that the CNN is expected to perform better than the MLP for the same dataset since CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. CNNs employ several layers but the layers employed in this experiment are –

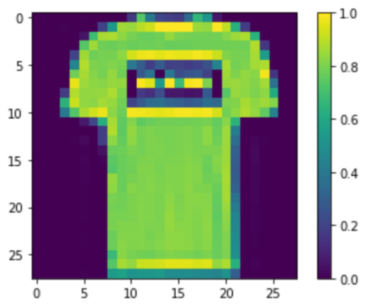
- CONV layer which will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- RELU layer will apply an elementwise activation function, such as the max(0,x)max(0,x) thresholding at zero.
- POOL layer will perform a down sampling operation along the spatial dimensions (width, height)

2 Dataset Definition

The dataset used for this experiment was 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. 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 below figure shows one row of the training dataset:



```
plt.figure()
plt.imshow(train_images[1])
plt.colorbar()
plt.grid(False)
plt.show()
```



The dataset was split as training, validation and testing as Training_Data [6000x28x28], Validation_Data [5000x28x28] and Test_Data[5000x28x28]. All three methods employ the same data split in order to achieve an even training of all three models and draw definitive conclusions for the predictions and accuracy.

3 Preprocessing

As part of data preprocessing, there were three crucial phases involved in all three parts of this experiment:

Phase 1: Data extraction

The data is extracted from the given Fashion-MNIST training and testing database as 60000 for test data and 5000 for validation data and 5000 for testing data:

```
import util_mnist_reader as mnist_reader
from sklearn.model_selection import train_test_split

X_train, y_train = mnist_reader.load_mnist('data\\fashion', kind='train')
X_data, y_data = mnist_reader.load_mnist('data\\fashion', kind='t10k')
X_validation, X_test, y_validation, y_test = train_test_split(X_data, y_data, test_size=0.50, shuffle=False)
```

The shuffle flag is set to False to have a uniform set of data for all three methods involved in this experiment. The input data was then reshaped if needed.

Phase 2: Data normalization

Each of the input vectors were normalized by dividing by 255 since the value of a pixel can range from 0 to 255.

Phase 3: Target label categorization

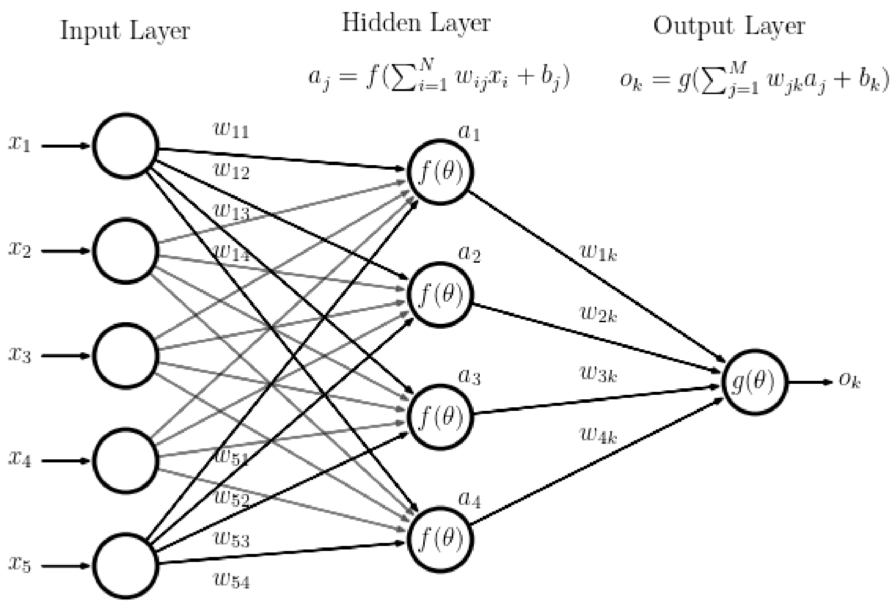
Each of the target vectors were categorized. Each of the target values is a single integer value and it was converted to obtain a classification such as the value of 2 corresponds to

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

4 Architecture

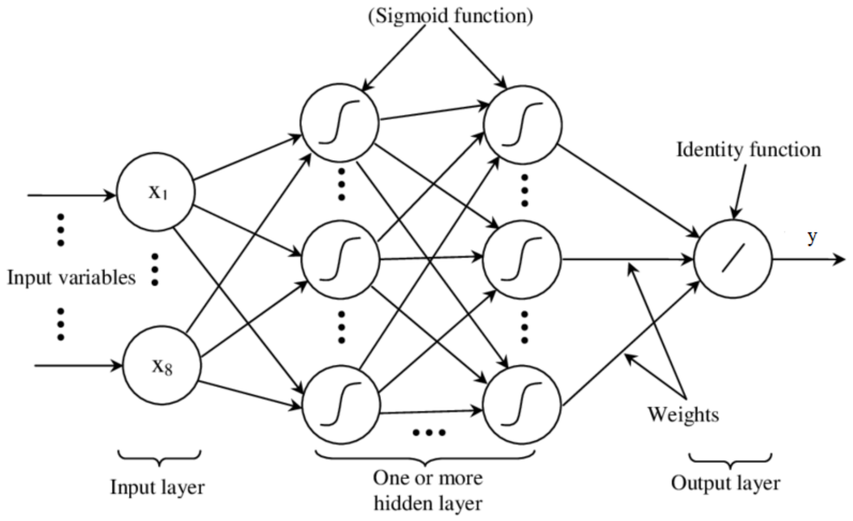
4.1 Part 1 – Single Hidden Layer

The generic computational model for a neural network with a single hidden layer is similar to a logistic regression classifier (as can be seen in the diagram below). The main differences between the two is the usage of vector weights in place of scalar and the presence of 2 activation functions (sigmoid and softmax in this part). This also gives rise to backpropagation being done twice in the same epoch – one corresponding to each layer on which activation was applied. In effect, a logistic classifier is a special case of a single hidden layer neural network.



4.2 Part 2 – Multi-layer perceptron neural network

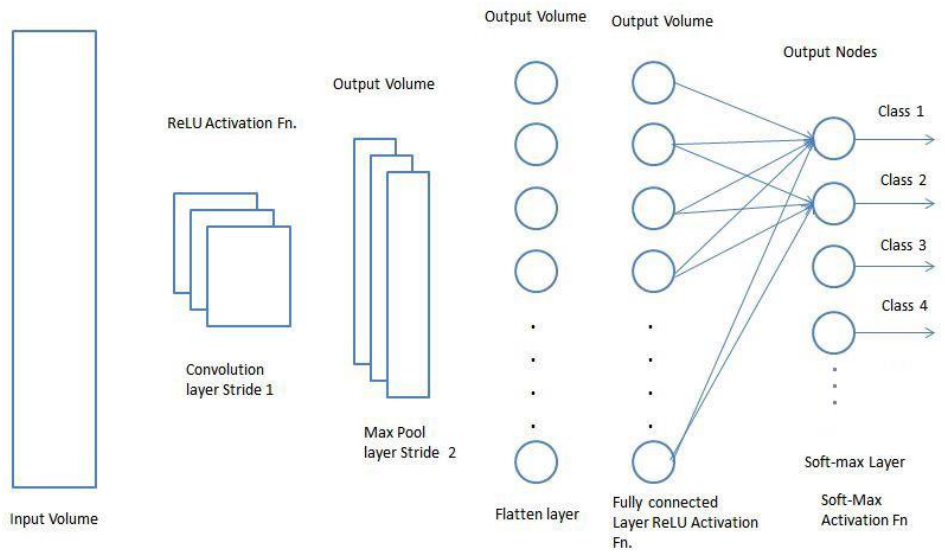
The single hidden layered neural network is a special case of MLP where there can be several layers present between the input and the output layer. In addition to these layers which employ the activation functions (here ReLU and softmax are used), dropout is employed in order to drop a percent of nodes which do not perform as good. Logistic regression with the One-versus-All approach can also be employed but the issue would arise when multiple layers would be needed. When a single layer is used, the logistic regression model would perform well since the logistic cost function is convex, finding the global minimum is a guaranteed result. But when this same model is employed across several layers, this convexity would be lost. Hence MLP would be preferable.



4.3 Part 3 – Convolutional Neural Network

The crux of any CNN lies in the convolutional layer. This layer's parameters consist of a set of learnable filters (also known as kernels), which have a small receptive field, but extend through the full depth of the input volume. During forward propagation, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input, producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map. Other layers of significance are the activation layers (ReLU and softmax here) and the MaxPool and Flatten layers.



5 Results

The results of the experiment were drawn based on tuning of the hyperparameters as well as using different regularization techniques and activation functions. The loss function was plotted against the epoch for both the validation dataset and the training dataset. The resultant confusion matrix was calculated in terms of accuracy, precision, recall and F1 score. Their respective formulas are:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

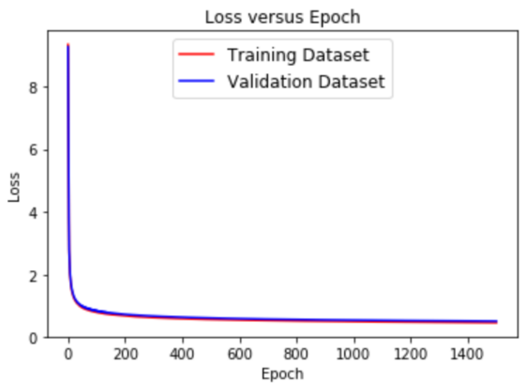
$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

- Accuracy provides the ratio of correctly labelled result out of all the predicted data.
- Precision provides the ratio of the correctly labelled positive result out of all the positive results.
- Recall gives the ratio of the correctly labelled positive result out of all the results.
- F1 Score gives the weighted average of precision and recall.

The classification report generated for each of the models presents all four metrics mentioned above. On the other hand, in the confusion matrix generated each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). The name stems from the fact that it is easy to identify if the system is confusing two classes. The diagonal represents the true positives achieved by the model.

5.1 Part 1 – Single Hidden Layer

Upon plotting the loss versus epoch of the single hidden layer neural network, there is an accuracy of 82 percent achieved in 1501 iterations with a learning rate of 1 and number of nodes specified in the hidden layer as 64. Increasing or decreasing the number of nodes usually alters the accuracy a little but does little to no impact on the recall. The Loss versus Epoch graph for the training and validation set is as shown below followed by the confusion matrix and the classification report.



```
[[420   6  14  24   0   0 126   0   5   0]
 [  3 485   3   8   0   0   1   0   0   0]
 [ 10   6 353  13  75   0  85   0   7   0]
 [ 24  14   4 408  14   1  26   0   2   0]
 [  3   5  76  32 370   0  91   0   5   0]
 [  0   0   0   0   1 448   3  18   4  14]
 [ 22   1  22  11  16   0 160   0   3   0]
 [  0   0   0   0   0  42   0 454   3  16]
 [ 11   2   7   3   3   4  26   1 445   1]
 [  0   0   0   1   0  20   0  27   0 492]]
```

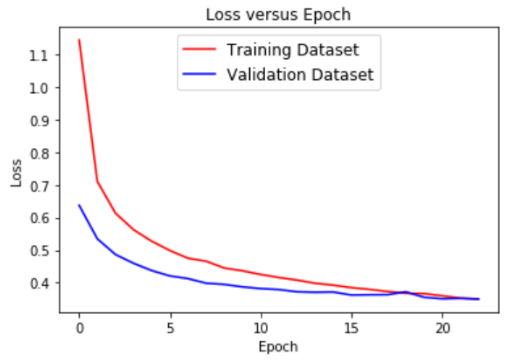
```
              precision    recall  f1-score   support

           0       0.85      0.71      0.77       595
           1       0.93      0.97      0.95       500
           2       0.74      0.64      0.69       549
           3       0.82      0.83      0.82       493
           4       0.77      0.64      0.70       582
           5       0.87      0.92      0.89       488
           6       0.31      0.68      0.42       235
           7       0.91      0.88      0.89       515
           8       0.94      0.88      0.91       503
           9       0.94      0.91      0.93       540

    accuracy                           0.81      5000
   macro avg       0.81      0.81      0.80      5000
weighted avg       0.84      0.81      0.82      5000
```

Confusion Matrix

Classification Report

5.2 Part 2 – Multi-layer perceptron neural network

Upon plotting the loss versus epoch of the multi-layer perceptron, there is an accuracy of 88 percent achieved in 23 iterations with 4 hidden layers – 3 ReLU and 1 softmax layer. Increasing or decreasing the number of nodes usually alters the accuracy a little but does little impact on the precision. The Loss versus Epoch graph for the training and validation set is as shown below followed by the confusion matrix and the classification report.



```
[[430   4   9   9   0   0  88   0   1   1]
 [  0 496   0   4   0   0   0   0   0   0]
 [  5   0 392   3  39   0  52   0   1   0]
 [ 12  12   2 455  17   0  15   0   2   0]
 [  0   4  57  12 394   0  54   0   1   0]
 [  0   0   0   0   0 488   0   9   2   3]
 [ 43   2  18  14  26   0 298   0   2   0]
 [  0   0   0   1   0  16   0 474   1  15]
 [  3   1   1   2   3   1  11   0 464   0]
 [  0   0   0   0   0  10   0  17   0 504]]
```

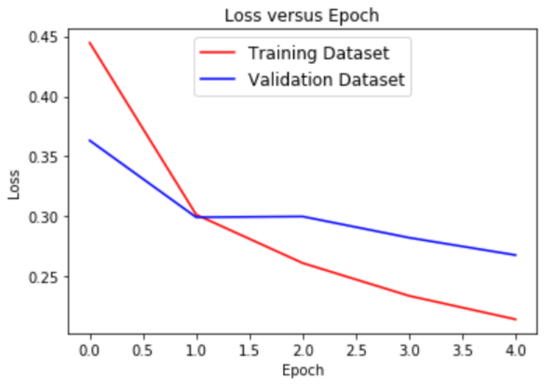
Confusion Matrix

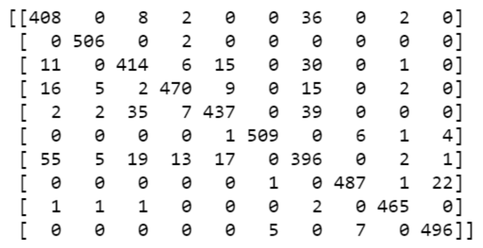
```
              precision    recall  f1-score   support

           0       0.87      0.79      0.83       542
           1       0.96      0.99      0.97       500
           2       0.82      0.80      0.81       492
           3       0.91      0.88      0.90       515
           4       0.82      0.75      0.79       522
           5       0.95      0.97      0.96       502
           6       0.58      0.74      0.65       403
           7       0.95      0.93      0.94       507
           8       0.98      0.95      0.97       486
           9       0.96      0.95      0.96       531

    accuracy                           0.88      5000
   macro avg       0.88      0.88      0.88      5000
weighted avg       0.89      0.88      0.88      5000
```

Classification Report

5.3 Part 3 – Convolutional Neural Network

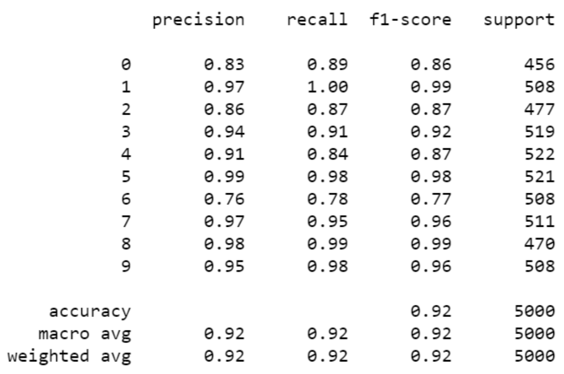
Upon plotting the loss versus epoch of the convolutional neural network, there is an accuracy of 92 percent achieved in 4 iterations. The Loss versus Epoch graph for the training and validation set is as shown below followed by the confusion matrix and the classification report.





```
[[408   0   8   2   0   0  36   0   2   0]
 [  0 506   0   2   0   0   0   0   0   0]
 [ 11   0 414   6  15   0  30   0   1   0]
 [ 16   5   2 470   9   0  15   0   2   0]
 [  2   2  35   7 437   0  39   0   0   0]
 [  0   0   0   0   1 509   0   6   1   4]
 [ 55   5  19  13  17   0 396   0   2   1]
 [  0   0   0   0   0   1   0 487   1  22]
 [  1   1   1   0   0   0   2   0 465   0]
 [  0   0   0   0   0   5   0   7   0 496]]
```

Confusion Matrix



```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       456
           1       0.97      1.00      0.99       508
           2       0.86      0.87      0.87       477
           3       0.94      0.91      0.92       519
           4       0.91      0.84      0.87       522
           5       0.99      0.98      0.98       521
           6       0.76      0.78      0.77       508
           7       0.97      0.95      0.96       511
           8       0.98      0.99      0.99       470
           9       0.95      0.98      0.96       508

    accuracy                           0.92      5000
   macro avg       0.92      0.92      0.92      5000
weighted avg       0.92      0.92      0.92      5000
```

Classification Report

6 Conclusion

Through the three models, altering the number of nodes, hidden layers, activation functions being used, regularization techniques, epochs and the learning rate, alters the results giving rise to different accuracies and predictions. In the case of part 1 of this experiment, an accuracy of 82% was achieved using a single hidden layer and sigmoid and softmax activation functions in roughly 1500 epochs. In the case of part 2 of this experiment, an accuracy of 87% was achieved by using a multi-layer perceptron with ReLU and softmax layers, stochastic gradient descent optimizer and dropouts between the layers in roughly 25 epochs. In the final case of part 3, an accuracy of 91% was achieved using ReLU and softmax activation in the CNN model within roughly 5 epochs.

References

[1] Wikipedia – Artificial Neural Network https://en.wikipedia.org/wiki/Artificial_neural_network

[2] Christopher M. Bishop ©2006 Pattern Recognition and Machine Learning. Springer-Verlag Berlin, Heidelberg

[3] Medium – Convolutional Neural Networks versus Fully Connected Neural Networks https://medium.com/datadriveninvestor/convolution-neural-networks-vs-fully-connected-neural-networks-8171a6e86f15

[4] Quora – https://www.quora.com/What-is-the-relationship-between-neural-networks-and-logistic-regression-given-that-a-single-layer-feed-forward-NN-looks-like-logistic-regression

[5] Wikipedia – Artificial Neural Network https://en.wikipedia.org/wiki/Multilayer_perceptron

[6] Towards Data Science – Activation functions neural networks https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[7] Wikipedia – Convolutional Neural Network https://en.wikipedia.org/wiki/Convolutional_neural_network

[8] Towards Data Science – Comprehensive guide to convolutional neural networks https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[9] Github.io – Convolutional Networks http://cs231n.github.io/convolutional-networks/

[10] Jonathan Weisberg – Neural Network from scratch https://jonathanweisberg.org/post/A%20Neural%20Network%20from%20Scratch%20-%20Part%201/

[11] Keras Classification - https://www.tensorflow.org/tutorials/keras/classification

[12] CNN from scratch https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/