Project 2: Neural network and Convolutional neural network

UBIT Name: Prashi Khurana
UBIT Number: 50316796
prashikh@buffalo.edu

Abstract

In this project, we will learn how to train and build a nueral network and a convolution nueral network. With the help of this model, we will classify the datatset(which is a set of images under the category of fashion), into the respective category to which they belong.

1 Introduction

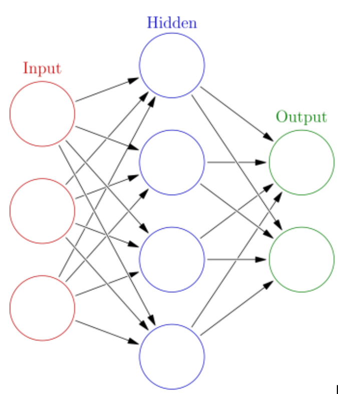
In this project we are focusing on how to train and design nueral networks. This includes training a nueral network from scratch and then learning the same from open tool library, like keras. The output we are expecting is a classification of the input image into one of the 10 classes for output.

1.1 What is Nueral Network and Why is it used

Let us first understand the name Nueral Network. The term comes from the human brain. Let us take a daily life example. For example if we have to hear or see anything, how does the brain interpret the input as a sound or image?. This is where the neurons come in place. Our sensory organs sense the environment variables and pass on the information as an 'input'. When this input is passed through the brain(a network of multilayer neurons), the brain classifies this input into the required output. Nueral Networks have the ability to process a lot of inputs, in a much faster time and classify them into multi-class. The applications for nueral networks range from Image Processing and Character recognition , Forecasting, etc.

1.2 Architecture of a Nueral Netwok:

A single layer nueral network can be shown as below.



[1]

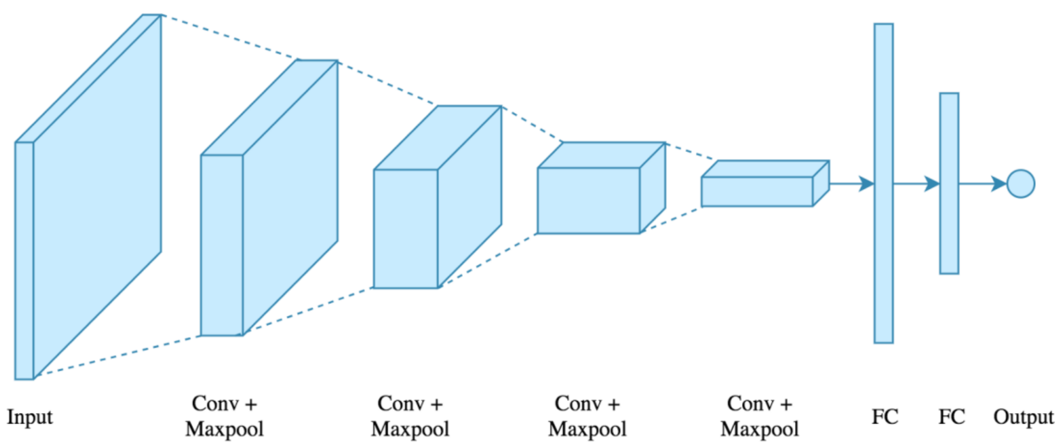
1.3 Multilayer Nueral Network :

The mathematical intuition is that each layer in a feed-forward multi-layer perceptron adds its own level of non-linearity that cannot be contained in a single layer. Each layer's inputs are only linearly combined, and hence cannot produce the non-linearity that can be seen through multiple layers. A handwritten-digit classifier might end up encoding edges in the first layer, curves and corners in the second layer, digit fragments in the third, and so on, finally up to the very abstract concepts of the "number 2" or "number 4" in the output layer. The multiple layers add levels of abstraction that cannot be as simply contained within a single layer of the same number of parameters, if they can be contained at all. [8]

Disadvantages of MLP include too many parameters because it is fully connected. Parameter number = width x depth x height. Each node is connected to another in a very dense web — resulting in redundancy and inefficiency.[9]

1.3 What is Convolutional Nueral Network and Why is it used

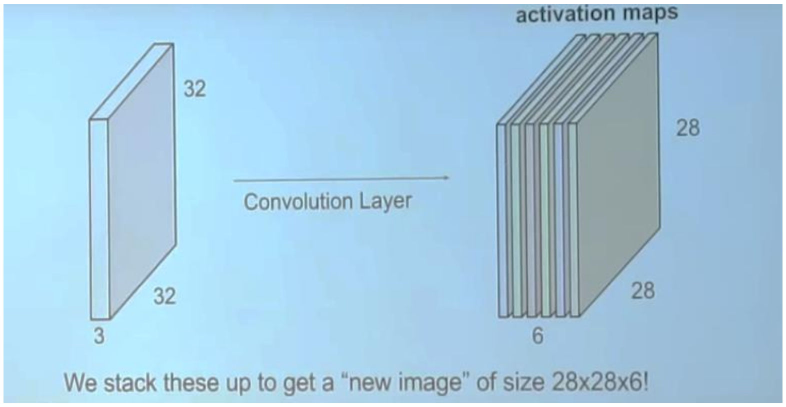
A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex.[3]
A CNN can account for local connectivity (each filter is panned around the entire image according to a certain size and stride, allows the filter to find and match patterns no matter where the pattern in located in the given image). The weights are smaller and shared – less wasteful, easier to train. Can also go deeper. Layers are sparsely connected than fully connected. Every node does not connect to every other node. [9]



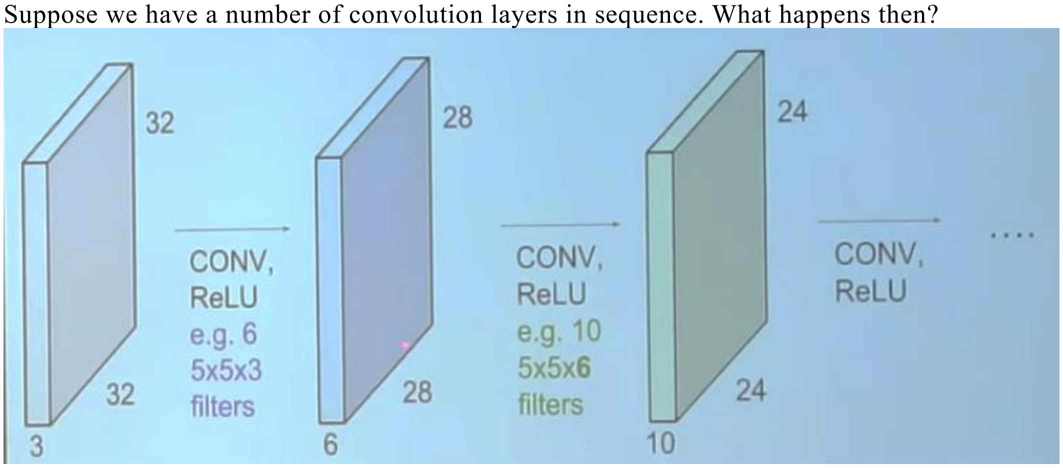
[11]

Let us understand step by step :

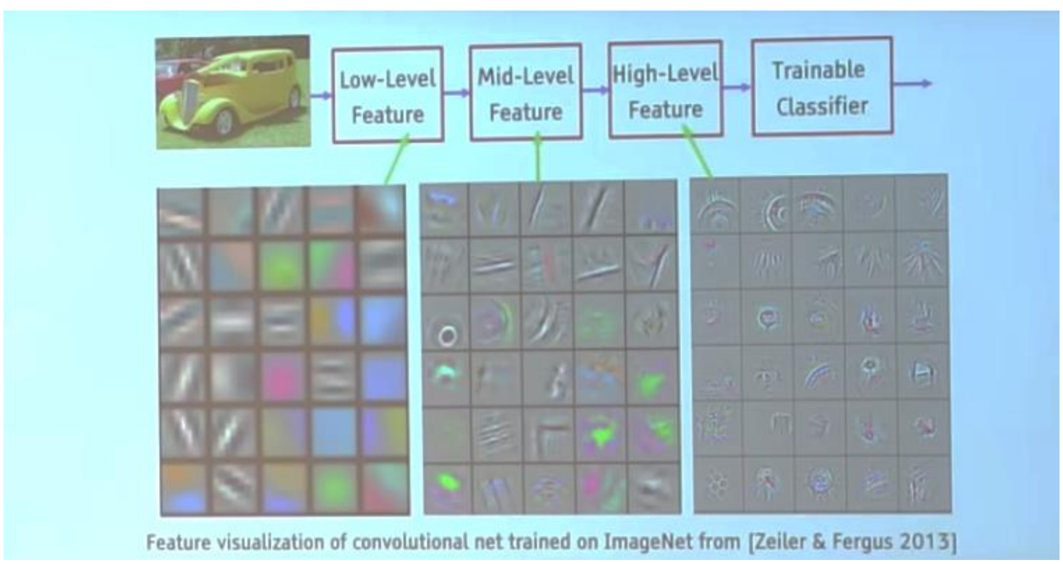
The convolution layer comprises of a set of independent filters (6 in the example shown). Each filter is independently convolved with the image and we end up with 6 feature maps of shape 28*28*1.



Suppose we have a number of convolution layers in sequence. What happens then?



All the filters are initially randomized and become parameters that will be learnt.



For a particular feature map (the output received on convolving the image with a particular filter is called a feature map), each neuron is connected only to a small chunk of the input image and all the neurons have the same connection weights. So again coming back to the differences between CNN and a neural network.[10]

2 Dataset Definition

The dataset is *Fashion-MNIST clothing images* . A few instances of data set:

Training Input size -> (60000, 784)
Training Output -> (60000,1)
Validation Input size -> (10000, 784)
Validation Output size->(10000,1)

3. Pre-processing

3.1 Preprocessing the data

The data which is generally used for training models might be inconsistent, incomplete and needs pre-processing. In our project, the following pre-processing functions were required:

3.1.1 Normalize the data

Normalization is an important part of the pre-processing of the data with machine learning. Normalization is the process of getting all the features of the of the instance in a common range. Gradient descent converges much faster when the features are normalized. The equation of normalization is X_train = X_train/255

4 Nueral Network Model and Convolutional Nueral Network

We will see the different functions and models used in Nueral Networks and Convolutional Nueral Networks.

4.1 Hyper-parameters

We need to initialize a few hyper parameters which include *weights(w), bias(b),learning rate(alpha)* and *hidden nodes*. We need to initialize these values to get started and later we will see how our choice affects the accuracy of the test set.

4.2 Sigmoid Function

The sigmoid function is defined as: **g(z) = 1/(1+e^{-z}).** The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

4.3 Softmax Function

The softmax function, also known as softargmax or normalized exponential function, is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.[4]

4.3 Cost Function

Instead of Mean Squared Error, we use a cost function called Cross-Entropy, also known as Log Loss.

Let us assume that[5]

$P(y = 1 \mid x; \theta) = h\theta(x)$ $\qquad$ $P(y = 0 \mid x; \theta) = 1 - h\theta(x)$

Note that this can be written more compactly as: $\quad p(y \mid x; \theta) = (h\theta(x))^{y} (1 - h\theta(x))^{1-y}$

Assuming that the m training examples were generated independently, we can then write down the likelihood of the parameters as: [5]

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

As before, it will be easier to maximize the log likelihood:

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

With regularization the cost function equation is :

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2.$$

4.4 Gradient Descent

To minimize our cost, we use Gradient Descent. Gradient Descent of a function is the algorithm to find the minimum of a function.

4.5 Understanding mini-batch gradient descent

Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.[5]

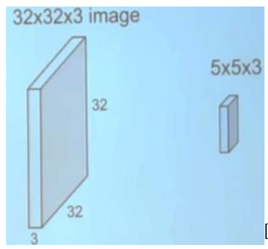
4.6 Understanding Accuracy.

Accuracy is defined as the quality or state of being correct or precise. This means how many instances of the test data set did the logistic regression was able to classify correctly.

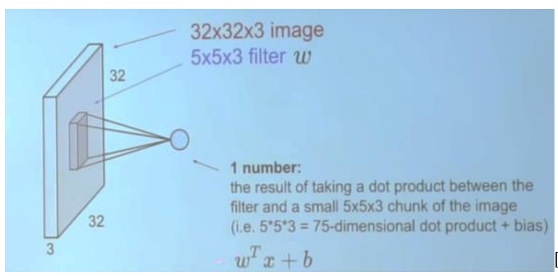
4.7 Cross Entropy

Cross entropy is a loss function, used to measure the dissimilarity between the distribution of observed class labels and the predicted probabilities of class membership. Categorical refers to the possibility of having more than two classes (instead of binary, which refers to two classes). Sparse refers to using a single integer from zero to the number of classes minus one (e.g. { 0; 1; or 2 } for a class label for a three-class problem), instead of a dense one-hot encoding of the class label (e.g. { 1,0,0; 0,1,0; or 0,0,1 } for a class label for the same three-class problem).[6]

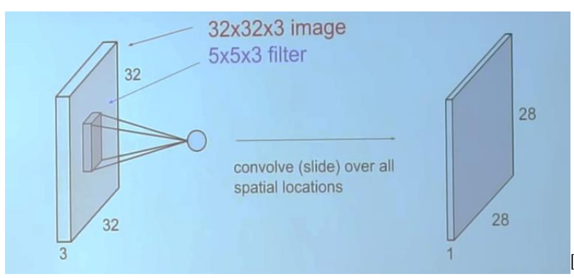
4.8 Convolution and Filters:



[10]



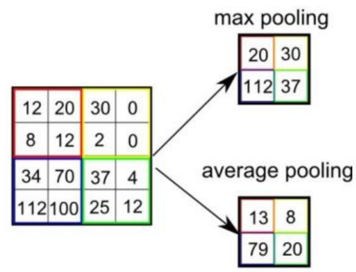
[10]



[10]

4.8 Max Pooling

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling.[10] It combines the output of neuron cluster at one layer into a single neuron in the next layer.



[11]

5 Actual Implementation

5.1 Pseudocode from scratch for Nueral Network

1. Pre-process the data (3)

2. Initialize the hyper-parameters (4.1)

3. For each *learning rate* in the learning rate set

 3.1 for each *hidden node* in the hidden node set

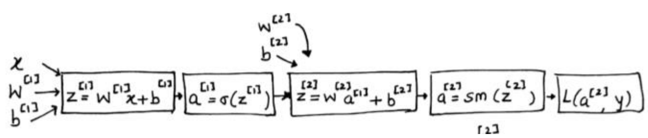
  3.1.1 Initialise *weights* and *bias*

  3.1.2 For each iteration-1,5,7 in iteration set

   3.1.2.1 for each iteration above,

    3.1.2.1.1 Run mini batch as the training dataset and use gradient descent to find out optimal weights

4. Steps to Train the data set:



4.1 Feed Forward

 4.1.1 Pass the input – xtrain,ytrain

 4.1.2 For the hidden layer use the activation function as sigmoid on the inputs. (a1)

 4.1.3 For the output layer use the activation function as softmax on the hidden layer outputs. (a2)

4.2 Back propagation to update the weights:

 4.2.1 We will use the following equations to update the weights.

$$\Delta W^{[2]} = (a^{[2]} - y)\, a^{[1]}$$

$$\Delta W^{[1]} = (a^{[2]} - y)\, W^{[2]} a^{[1]} (1 - a^{[1]})\, x$$

Let us understand what do the symbols mean:

W2 – Weights between the hidden nodes and the output nodes

W1 – weights between the input nodes and the hidden nodes

A2 – Activation function from hidden nodes (softmax)

A1 – Activation function from input nodes (sigmoid)

X – Input dataset

Y – Labels for the input dataset

5.2 Pseudocode for Nueral Network using Keras

1. Pre-process the data (3) - x_train = x_train.astype('float32')/255
2. Set a model
3. Set the hidden layer with activation function as sigmoid
4. Add one more layer with 128 nodes and activation function as RELU
5. Set the ouput layer with activation function as softmax
6. Compile the model with optimizer as ‘sgd’ and loss as ‘'sparse_categorical_crossentropy'
7. Then run the model on test data set
8. Evaluate the model to get the accuracy

5.3 Pseudocode for Convolutional Nueral Network using Keras

1. conv1 = layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)
2. conv2 = layers.Conv2D(64, (3,3), activation='relu')
3. conv3 = layers.Conv2D(128, (3,3), activation='relu')
4. max_pool_1 = layers.MaxPooling2D((2,2))
5. max_pool_1 = layers.MaxPooling2D((2,2))
6. max_pool_3 = layers.MaxPooling2D((2,2))
7. flat_layer = layers.Flatten()
8. fc = layers.Dense(128, activation='relu')
9. output = layers.Dense(10, 'softmax')
10. add to the model and run

6 Results

6.1 Understanding the results

6.1.1 Nueral Network From Scratch :

```
For Lambda 0.001
        Hidden Nodes:  10
                Iteration 1
                Cost for Iteration  1 is:  1.9405562416798192
                Iteration 5
                Cost for Iteration  5 is:  0.971068601062768
                Iteration 7
                Cost for Iteration  7 is:  0.879766012658019
        Accuracy for:  10 is:  70.39
        Hidden Nodes:  32
                Iteration 1
                Cost for Iteration  1 is:  1.3748969320322548
                Iteration 5
```

Cost for Iteration 5 is: 0.9979137024798113
Iteration 7
Cost for Iteration 7 is: 0.76031644062548
Accuracy for: 32 is: 73.24000000000001
Hidden Nodes: 64
Iteration 1
Cost for Iteration 1 is: 1.528181670935715
Iteration 5
Cost for Iteration 5 is: 0.9502488071689618
Iteration 7
Cost for Iteration 7 is: 0.7758979646140699
Accuracy for: 64 is: 71.97
Final Accuracy for lambda 0.001 is: 75.3

6.1.2 Nueral Network with Multiple Layers :

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

6.1.3 Convolutional Nueral Networks :

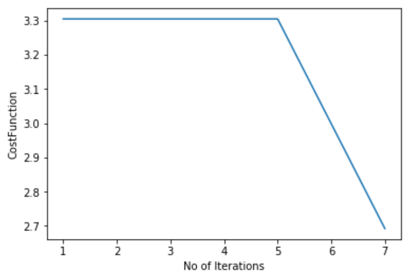
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 22, 22, 128)       73856
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 11, 11, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1982592
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,076,554
Trainable params: 2,076,554
Non-trainable params: 0
```

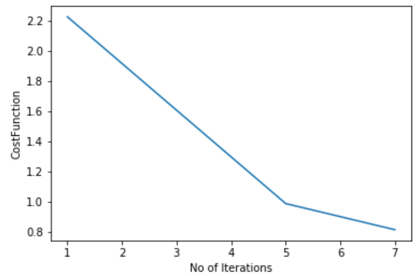
6.2 Graphs for Nueral Network From Scratch

6.2.1 Cost Function(Training) Vs Iterations

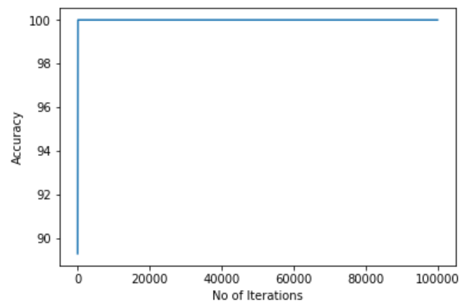
For Fixed Learning Rate & Hidden Nodes:

Learning Rate: 10e-5 Hidden Nodes:10 Learning Rate: 0.001 Hidden Nodes:64

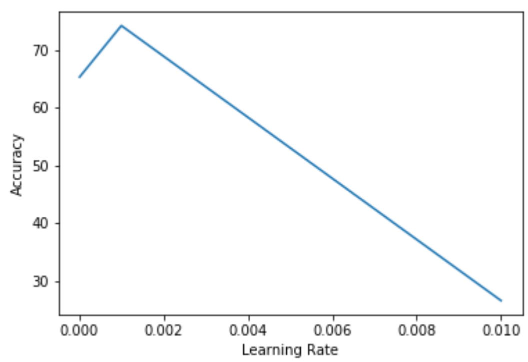




6.2.2 Iterations Vs Accuracy



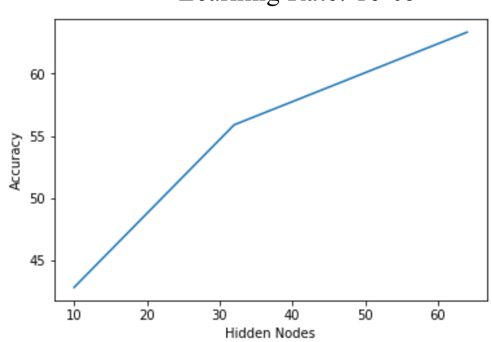
6.2.3 Learning Rate Vs Accuracy



6.2.4 Hidden Nodes Vs Accuracy

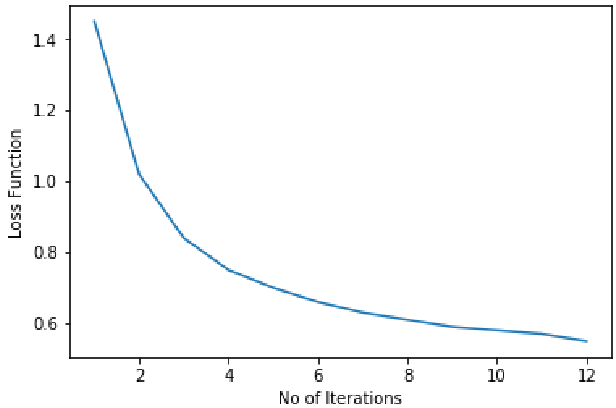
Nodes Vs Accuracy for a fixed Learning Rate:

Learning Rate: 1e-05



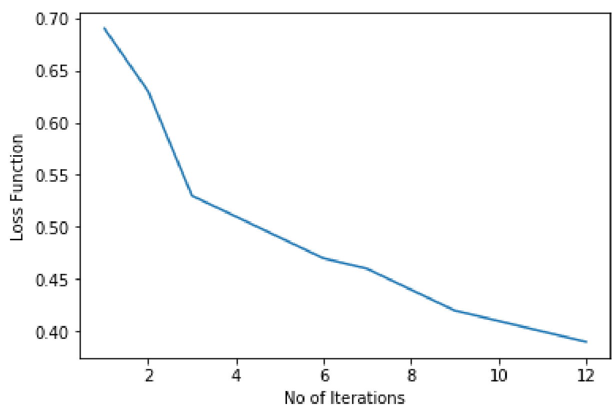
6.3 Graphs from Multi Layer Nueral Network

6.3.1 Cost Function(Training) Vs Iterations



6.4 Graphs from Convolutional Nueral Network

6.4.1 Cost Function(Training) Vs Iterations



6.4 Confusion Matrix:

6.4.1 Nueral Network From Scratch

True Positives: 928
True Negatives :858
False Positive: 1
False Negative:3
Final Accuracy: 72.5

6.4.2 Multi Layer Nueral Network

Confusion matrix is

```
 [[812   2  19  55   5   4  88   0  15   0]
 [  2 946  11  31   6   0   2   0   2   0]
 [ 20   1 700   9 178   1  77   0  14   0]
 [ 30  13  12 851  43   1  45   0   5   0]
 [  0   0  84  30 807   0  74   0   5   0]
```

```
 [  0   0   0   1   0 908   0  56   3  32]
 [157   1 123  44 139   1 503   0  32   0]
 [  0   0   0   0   0  45   0 900   0  55]
 [  1   1   8  10   2   4  19   5 949   1]
 [  0   0   0   0   0  21   0  43   1 935]]
```

Accuracy : 83.11%

6.4.3 Convolutional Nueral Network

Result Set A:

Confusion matrix is

```
 [[491   5 249   0 133   2  80   0  39   1]
 [  1 857   0   0 110   1   8   0   3  20]
 [ 13   2 630   0 281   1  65   0   7   1]
 [ 48  17 137 107 485   5  39   0  69  93]
 [  0   2  34   0 909   0  44   0   6   5]
 [ 41   0   0   2   3 797  89  26  33   9]
 [103   3 107   3 513   1 232   0  34   4]
 [  3   0   0   2   0 189 129 568 109   0]
 [ 84   0  12   0  95   1 137   1 669   1]
 [  7   1   0   1   5   4 266 132 159 425]]
```

Accuracy : 88.74%

Result Set B:

```
[[470   7 200  17  71   0 194   0  41   0]
 [  1 920   1  20  10   0  29   0  18   1]
 [ 18   0 625   6 208   1 136   0   6   0]
 [  4  23  19 627  97   0 120   0 103   7]
 [  2   2  40  20 847   0  78   0   9   2]
 [ 91   3   0   0   0 848  12   0  44   2]
 [ 59   4  51  14 312   1 542   0  17   0]
 [  4   0   0   0   0 279  48 467 137  65]
 [ 32   3  16   3   4   2  46   0 893   1]
 [106   5   0   0   0  16 425   9  93 346]]
```

Accuracy : 90.26%

6.6 Accuracy:

Nueral Network:

Accuracy : 75.3 %

Multi Layer Nueral Network using Keras:

Accuracy : 83.11%

Convolution Nueral Network:

Accuracy : 90.26%

7 Conclusion

Once we have trained the logistic regression model, we need to understand the results of the model.

- Even though we are using the same dataset to train the model, it is not necessary that we get the same accuracy each time. This is because each time we use a different set of the training data.

- Understanding the graphs in **6.2:**
  - *Cost Function Vs Iterations*:
    We calculate the cost function and plot the graph against the cost function and the number of iterations. As the number of iterations increase we see that the cost function decreases. This is because as you train the data, the weights become more accurate and the loss function or cost function decreases. Decreasing of the cost function means your trained logistic regression will have more accuracy.
  - *Iterations Vs Accuracy*:
    As we can see the graph, initially the accuracy increases with the number of iterations, but it stabilizes later and does not increase any further with the number of iterations.
  - *Learning Rate Vs Accuracy*:
    We see that the learning rate vs accuracy graph is similar to the iterations vs accuracy graph.
  - *Hidden Nodes Vs Accuracy*:
    We see as the hidden nodes increase so does the accuracy.

- Understanding Accuracy for different models:
  - Accuracy for Nueral Network from Scratch:
    We have achieved an accuracy of 75.3%(approximately).
  - Accuracy for Multi Layer Nueral Network in Keras :
    We have achieved an accuracy of 83.11%(approximately), because we used more hidden nodes, and more layers.
  - Accuracy for Convolutional Nueral Network in Keras :
    We were able to achieve and accuracy of 90.26%, because we have more dense layers and a lot of hidden nodes per layer.

- So we can conclude with the increase of the hidden nodes, and hidden layers a better accuracy is guaranteed.

References

[1] https://en.wikipedia.org/wiki/Artificial_neural_network

[2] https://www.jeremyjordan.me/convolutional-neural-networks/

[3] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[4] https://en.wikipedia.org/wiki/Softmax_function

[5]https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[6]https://www.reddit.com/r/MLQuestions/comments/93ovkw/what_is_sparse_categorical_crossentropy/

[7]https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/logistic-regression-analysis-r/tutorial/

[8]https://www.quora.com/Why-do-neural-networks-with-more-layers-perform-better-than-a-single-layer-MLP-with-a-number-of-neurons-that-leads-to-the-same-number-of-parameters

[9]https://medium.com/data-science-bootcamp/multilayer-perceptron-mlp-vs-convolutional-neural-network-in-deep-learning-c890f487a8f1

[10]https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

[11] https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks