

CVML CS

Assignment 1

Report on

Handwritten Digits Recognition Models *on MNIST dataset*

Group - 24

Prayas Barahate(194101013)
Shubham Damkondwar(194101016)
Mayank Verma(194101031)
Sweety Dhale(194101052)
Priyanka Khare(194101023)

Objective

To train various classifiers/model using the following methods to classify the image sample given by MNIST dataset into one of the 10 possible classes (0 to 9).

1. Logistic Regression
2. Multi Layer Perceptron
3. Deep Neural Network
4. Deep Convolutional Neural Network

Further,

Exercise on various parameters (if applicable) such as number of hidden layers, type of activation functions, number of convolution kernels, size of convolution kernels. Also exercise on over-fitting problem with possible solutions.

Abstract

We implemented various machine/deep learning approach to detect the hand written digits for which we classify the MNIST dataset into 10 classes of digits [0-9]. We use 4 approaches which are Logistic regression, Multi-layer perceptron, Deep neural network and Deep convolutional neural network. The prediction is done by using softmax regression. The tensorflow library provides backend for complex mathematical computation to perform deep neural networks and logistic regression.

Introduction

The MNIST dataset that we have used in this model consist of 70000 images of handwritten digits. These 70000 images are classified into 3 types - training data, testing data and validation data. 60000 examples are used as training data out of which 5000 are used as validation data, and 10000 examples are used as testing data. To avoid the condition of overfitting we use validation data. In this MNIST dataset the dimension of each handwritten digit image is 28x28 pixels so total 784 pixels are used to represent every image, and the pixel value for each pixel lies between the range 0 - 255. The pixel value 0 is used for complete black and value 255 is used for complete white. In order to do the computations it is always good to store the image in single dimensional vector of size 1x784. Our goal is to predict the numerical label that is represented by the image, and these labels are categorized into 10 classes [0-9].

Methods

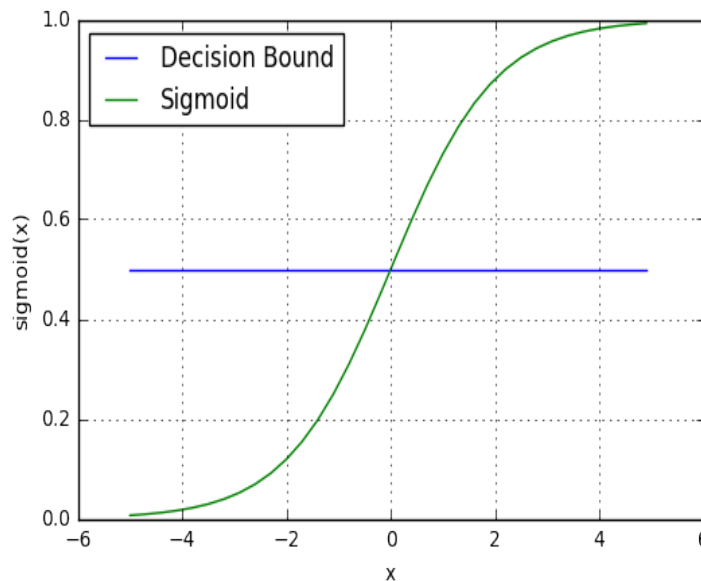
1 : LOGISTIC REGRESSION

Logistic Regression is a technique in machine learning which is used for classification. Logistic regression is used when the output we want is binary i.e logistic regression gives output either true or false. One classic example is whether a given object is red or not. In this case our Logistic Regression will classify object as 'red' if the object is of red color and classify as 'not red' if the object is not of red color. No third class is present.

The activation function used is Sigmoid function. This function is used to map the predicted values to probabilities.

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

The plot of sigmoid function will look as figure given below.



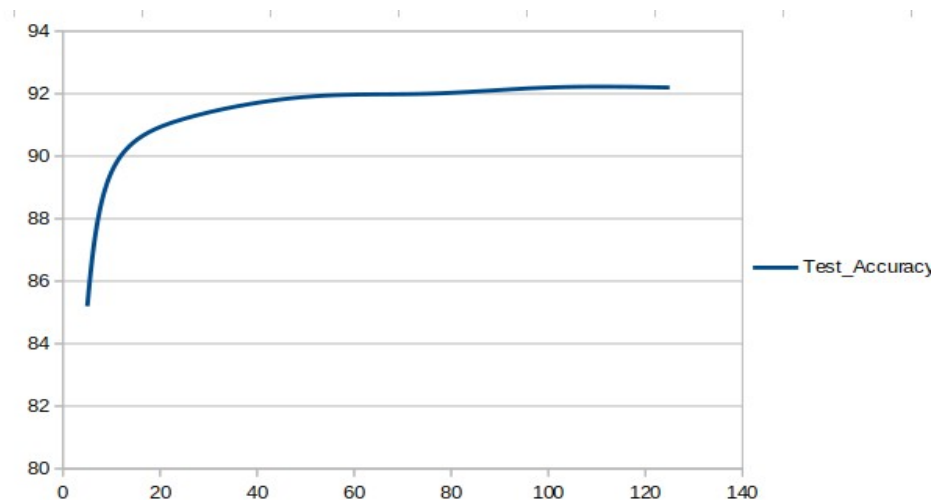
A decision bound is set at probability equals to 0.5 . For a given point if its probability is above decision bound we will classify it as some class and if its probability is below decision bound we will classify it as another remaining class.

Snapshot of training and testing of our model given below:

```
training_epochs = 50
batch_size      = 100
display_step    = 5
# SESSION
sess = tf.Session()
sess.run(init)
# MINI-BATCH LEARNING
for epoch in range(training_epochs):
    avg_cost = 0.
    num_batch = int(mnist.train.num_examples/batch_size)
    for i in range(num_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        sess.run(optm, feed_dict={x: batch_xs, y: batch_ys})
        feeds = {x: batch_xs, y: batch_ys}
        avg_cost += sess.run(cost, feed_dict=feeds)/num_batch
    # DISPLAY
    if epoch % display_step == 0:
        feeds_train = {x: batch_xs, y: batch_ys}
        feeds_test = {x: mnist.test.images, y: mnist.test.labels}
        train_acc = sess.run(accr, feed_dict=feeds_train)
        test_acc = sess.run(accr, feed_dict=feeds_test)
        print ("Epoch: %03d/%03d cost: %.9f train_acc: %.3f test_acc: %.3f"
              % (epoch, training_epochs, avg_cost, train_acc, test_acc))
print ("DONE")
```

Epoch: 000/050 cost: 2.273109296 train_acc: 0.650 test_acc: 0.676
Epoch: 005/050 cost: 2.013840492 train_acc: 0.730 test_acc: 0.740
Epoch: 010/050 cost: 1.804733397 train_acc: 0.780 test_acc: 0.762
Epoch: 015/050 cost: 1.634545778 train_acc: 0.820 test_acc: 0.785
Epoch: 020/050 cost: 1.495173216 train_acc: 0.780 test_acc: 0.797
Epoch: 025/050 cost: 1.381612569 train_acc: 0.780 test_acc: 0.806
Epoch: 030/050 cost: 1.288967186 train_acc: 0.820 test_acc: 0.814
Epoch: 035/050 cost: 1.210072822 train_acc: 0.760 test_acc: 0.819
Epoch: 040/050 cost: 1.143169935 train_acc: 0.790 test_acc: 0.824
Epoch: 045/050 cost: 1.084837631 train_acc: 0.840 test_acc: 0.828
DONE

Training epochs versus Accuracy :



X axis represents training epochs.

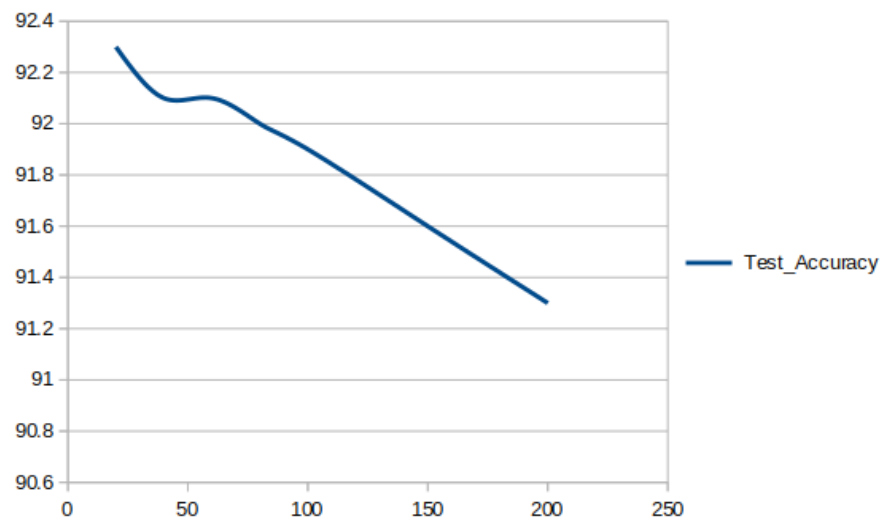
Y axis represents testing accuracy.

Learning rate is kept 0.01

Batch size is kept 100

As we can observe from the graph, as we increase epochs, testing accuracy increases.

Batch size versus Accuracy :



X axis represents batch size.

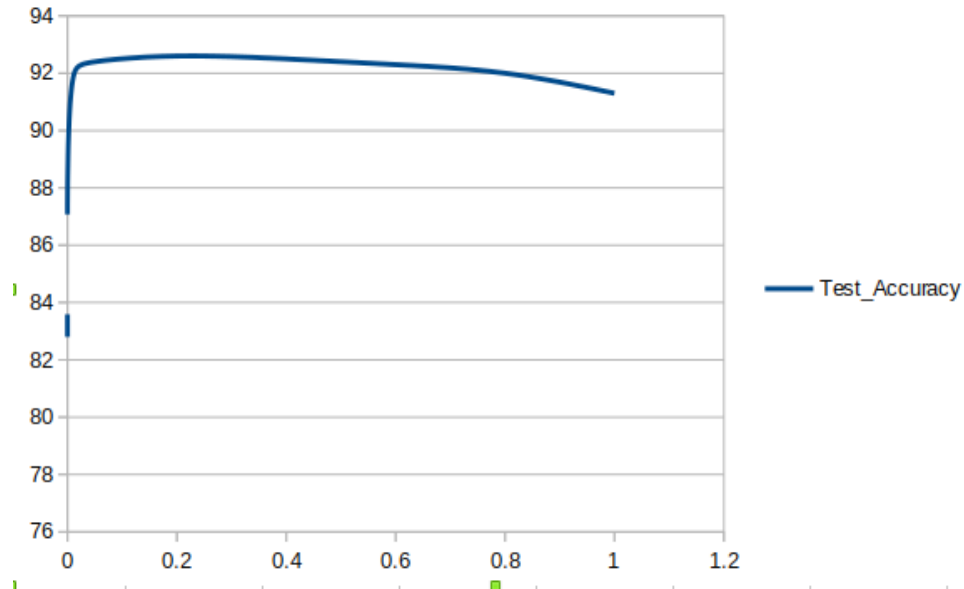
Y axis represents testing accuracy.

Learning rate is kept 0.01

Training epochs is kept 50

As we can observe from the graph, as we increase batch size, testing accuracy decreases.

Learning Rate versus Accuracy :



X axis represents learning rate.

Y axis represents testing accuracy.

Batch size is kept 100

Training epochs is kept 50

Learning rate is kept in between 0 and 1.

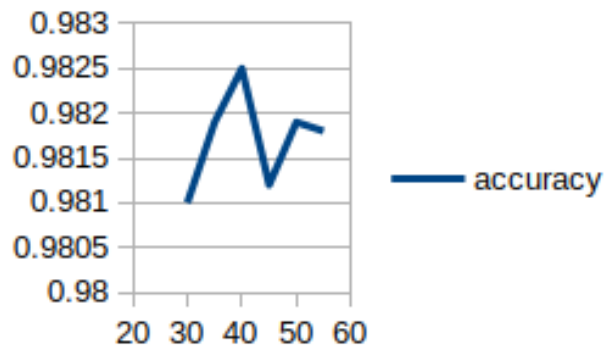
As we can observe from the graph, when the learning rate is very small testing accuracy is very less. As we increase learning rate testing accuracy increases till some specific point. After that point the testing accuracy decreases again.

2: Multi Layer Perceptron:

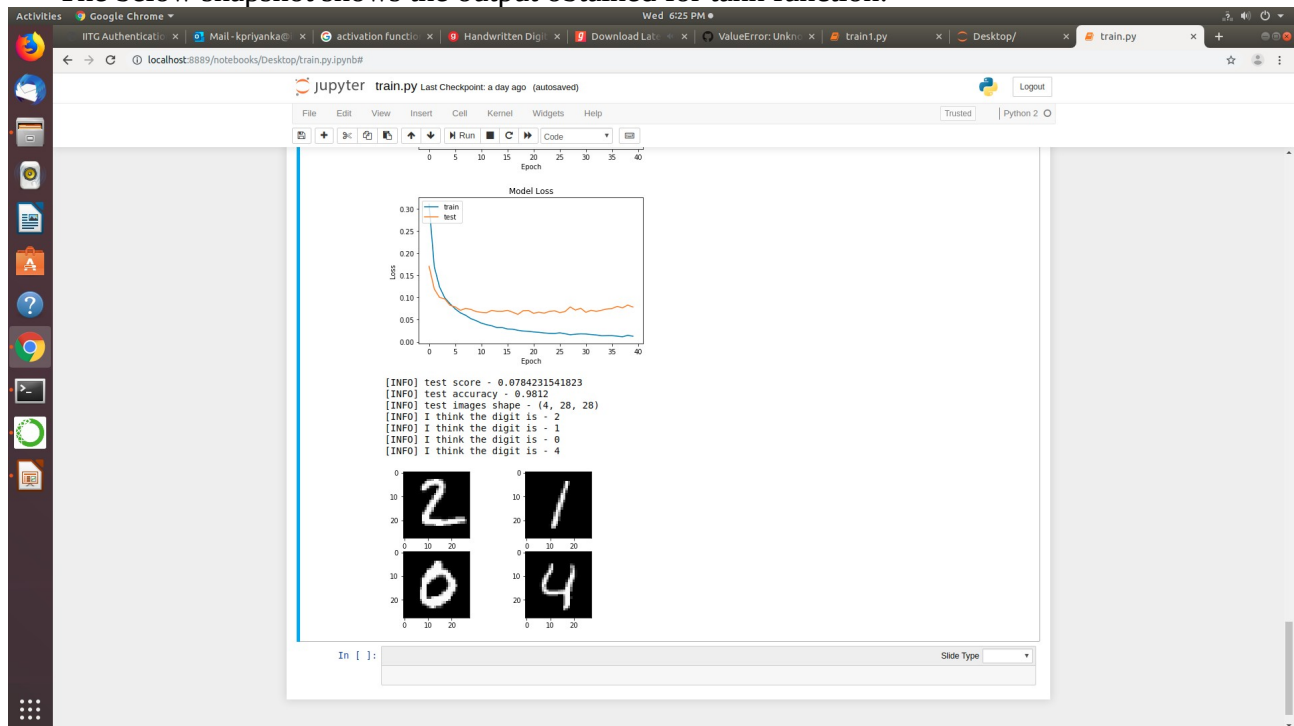
Multi Layer Perceptron Network is also called as Feed Forward Neural Network. The goal is to approximate a function. Each layer in the network is composed of units that perform an affine transformation of a linear sum of inputs and is represented by $y=f(WxT+b)$ where f is the activation function, w is weight in the layer, x is the input vector, that can also be output of previous layer and b is the bias vector.

The libraries used in the program are Matplotlib, Keras and Numpy. We have used our network with 784 input neurons and two input layers, 512 neurons in first hidden layer and 256 neurons in second hidden layer.

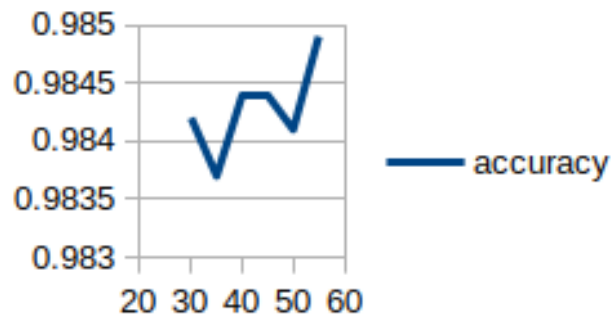
The following graph is plotted for the **tanh activation function**. Epoch is plotted on the X axis and Accuracy on the Y axis. Here, the maximum accuracy achieved is 0.983. tanh is used as the activation function for hidden layers and softmax as the activation function for outer layers.



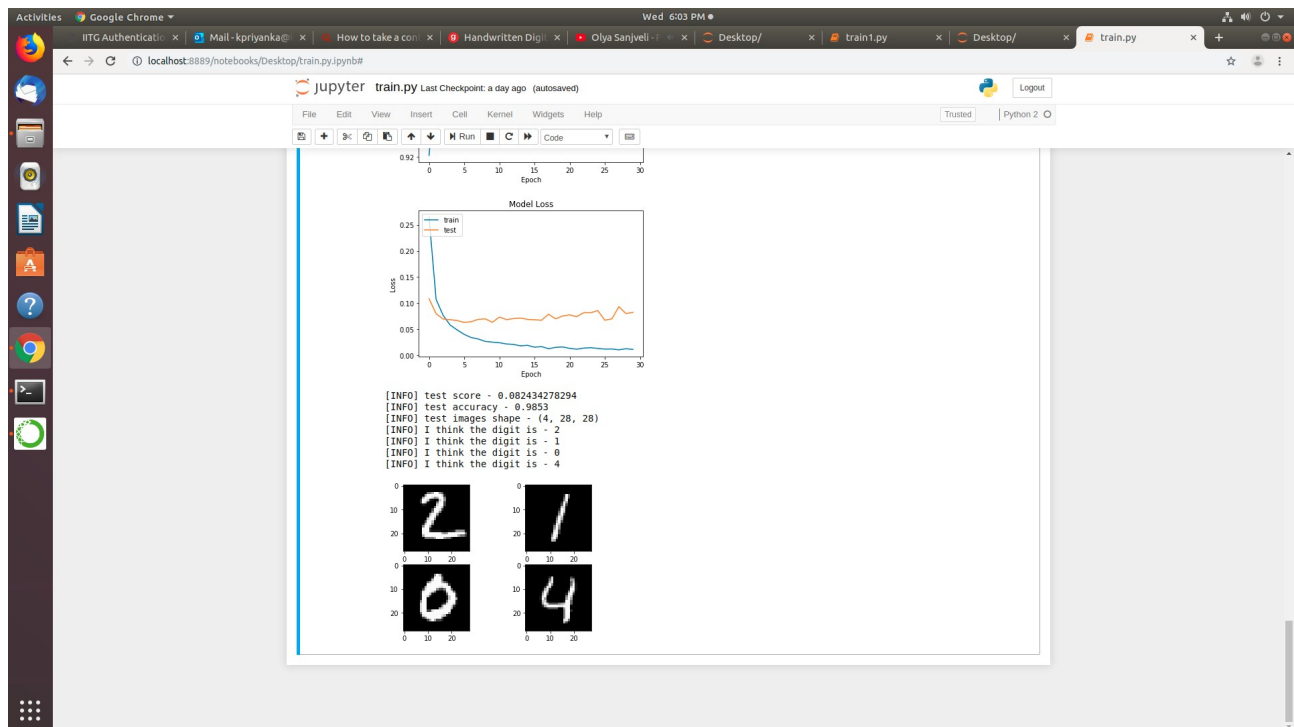
The below snapshot shows the output obtained for tanh function:



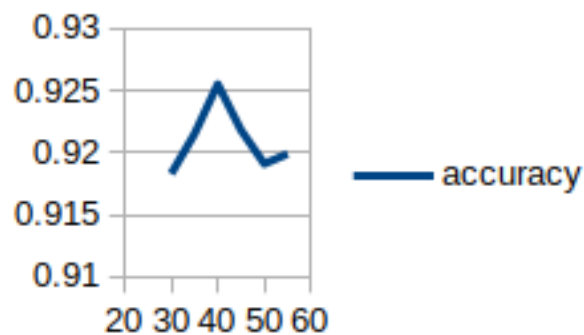
The following graph is plotted for **the ReLU activation function**. Epoch is plotted on the X axis and Accuracy on the Y axis. Here, the maximum accuracy achieved is 0.985. ReLU is used as the activation function for hidden layers and softmax as the activation function for outer layers.



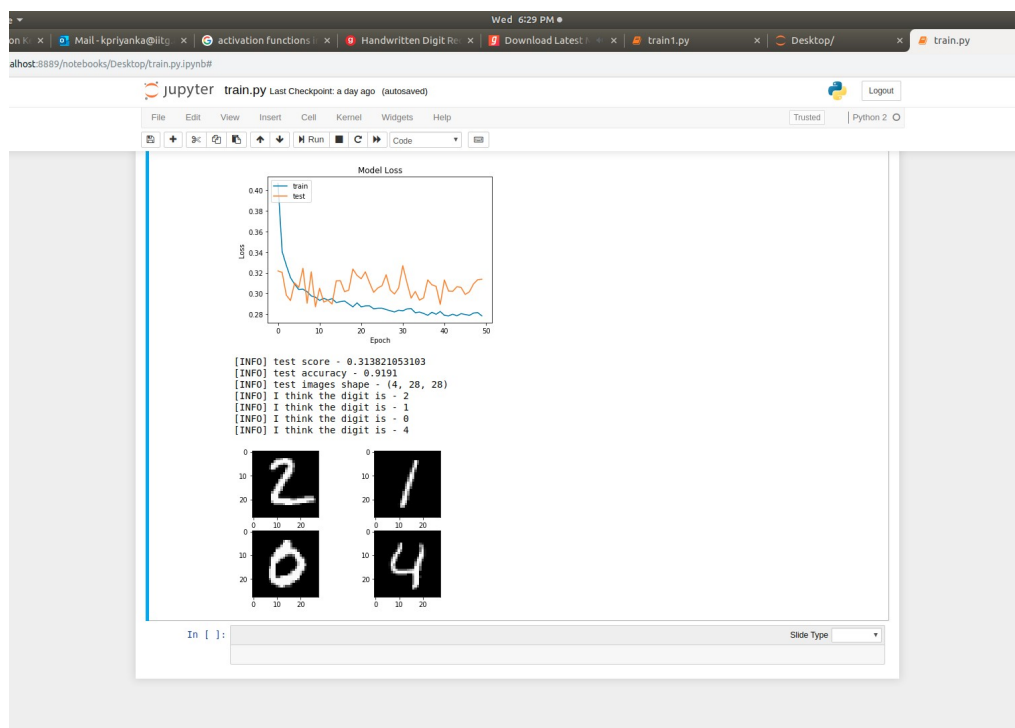
The below snapshot shows the output obtained for ReLU function:



The following graph is plotted for **the linear activation function**. Epoch is plotted on the X axis and Accuracy on the Y axis. Here, the maximum accuracy achieved is 0.93. Linear is used as the activation function for hidden layers and softmax as the activation function for outer layers.



The below snapshot shows the output obtained for linear function:

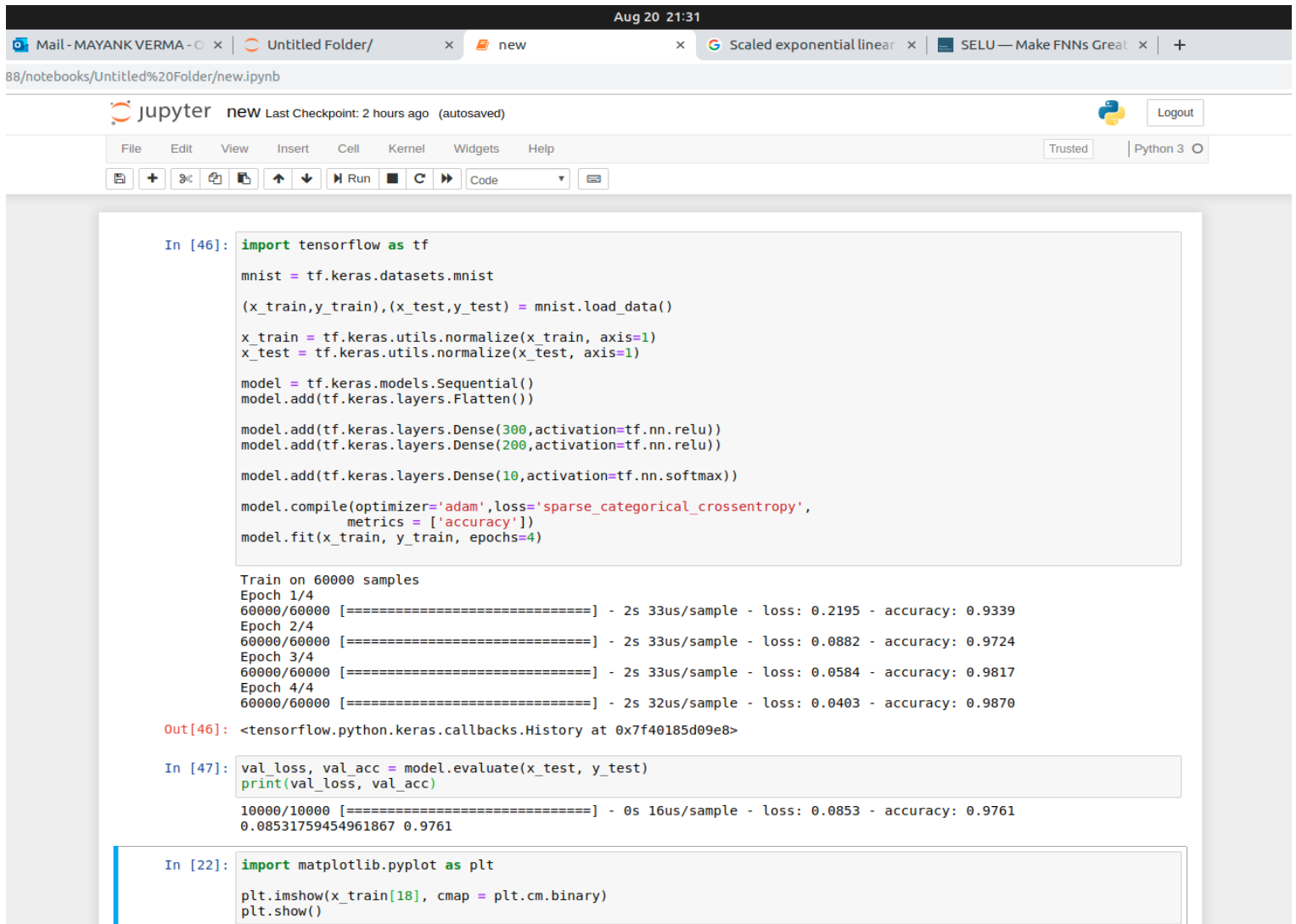


3. Deep Neural Network

A neural network, in general, is a technology built to simulate the activity of the human brain – specifically, pattern recognition and the passage of input through various layers of simulated neural connections.

Many experts define deep neural networks as networks that have an input layer, an output layer and at least one hidden layer in between. Each layer performs specific types of sorting and ordering in a process that some refer to as “feature hierarchy.” One of the key uses of these sophisticated neural networks is dealing with unlabeled or unstructured data. The phrase “deep learning” is also used to describe these deep neural networks, as deep learning represents a specific form of machine learning where technologies using aspects of artificial intelligence seek to classify and order information in ways that go beyond simple input/output protocols.

We create model to load dataset from MNIST using tensorflow and keras library also use same as backend for complex calculation of each hidden layer. Further we train our model with ‘Relu’ activation function and ‘adam’ optimizer.



The screenshot displays a Jupyter Notebook interface with a browser window at the top showing the URL `88/notebooks/Untitled%20Folder/new.ipynb`. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The main area contains three code cells:

```
In [46]: import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train,y_train),(x_test,y_test) = mnist.load_data()
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Flatten())

model.add(tf.keras.layers.Dense(300,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(200,activation=tf.nn.relu))

model.add(tf.keras.layers.Dense(10,activation=tf.nn.softmax))

model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
              metrics = ['accuracy'])
model.fit(x_train, y_train, epochs=4)

Train on 60000 samples
Epoch 1/4
60000/60000 [=====] - 2s 33us/sample - loss: 0.2195 - accuracy: 0.9339
Epoch 2/4
60000/60000 [=====] - 2s 33us/sample - loss: 0.0882 - accuracy: 0.9724
Epoch 3/4
60000/60000 [=====] - 2s 33us/sample - loss: 0.0584 - accuracy: 0.9817
Epoch 4/4
60000/60000 [=====] - 2s 32us/sample - loss: 0.0403 - accuracy: 0.9870

Out[46]: <tensorflow.python.keras.callbacks.History at 0x7f40185d09e8>

In [47]: val_loss, val_acc = model.evaluate(x_test, y_test)
print(val_loss, val_acc)

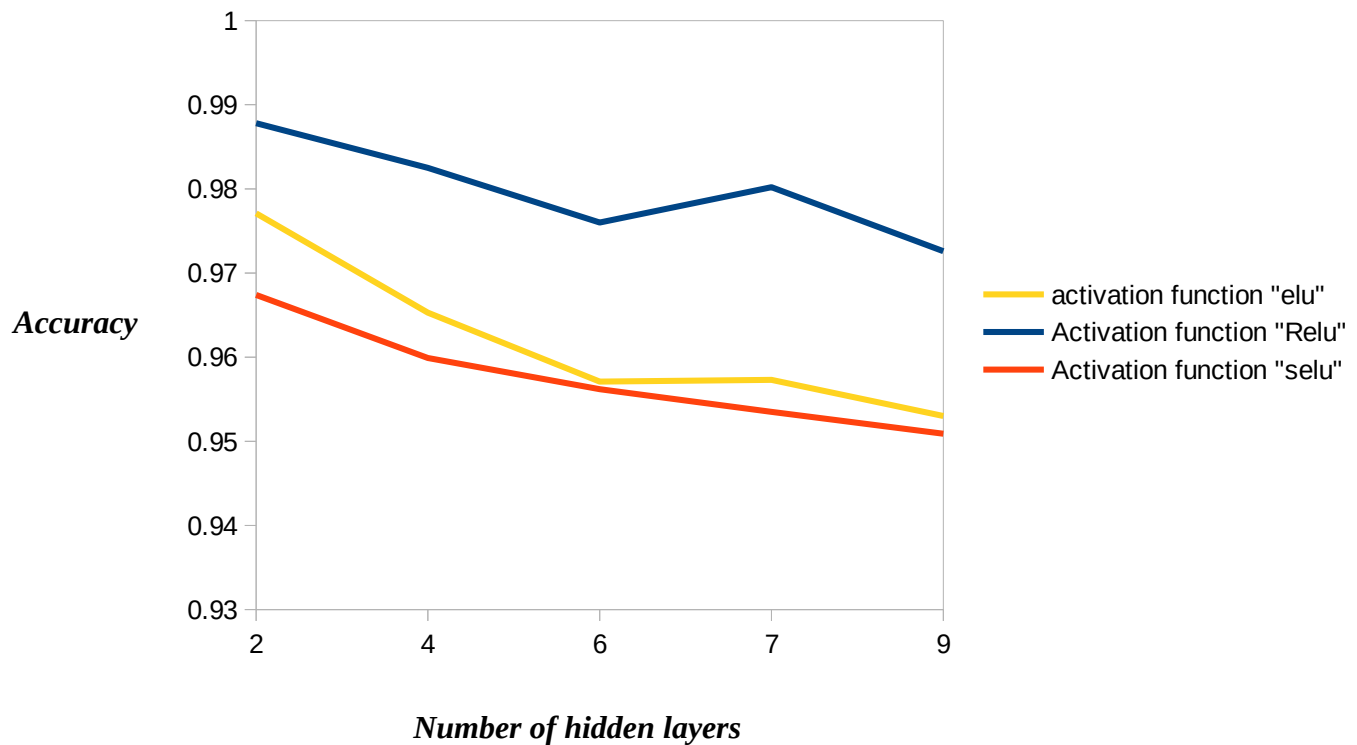
10000/10000 [=====] - 0s 16us/sample - loss: 0.0853 - accuracy: 0.9761
0.08531759454961867 0.9761

In [22]: import matplotlib.pyplot as plt

plt.imshow(x_train[18], cmap = plt.cm.binary)
plt.show()
```

We also analyse our model with multiple and various parameters like different numbers of hidden layers in model, different activation function and different number of neurons in network. As we are changing parameters the learning mechanism of our model changes so as the accuracy and loss of our model.

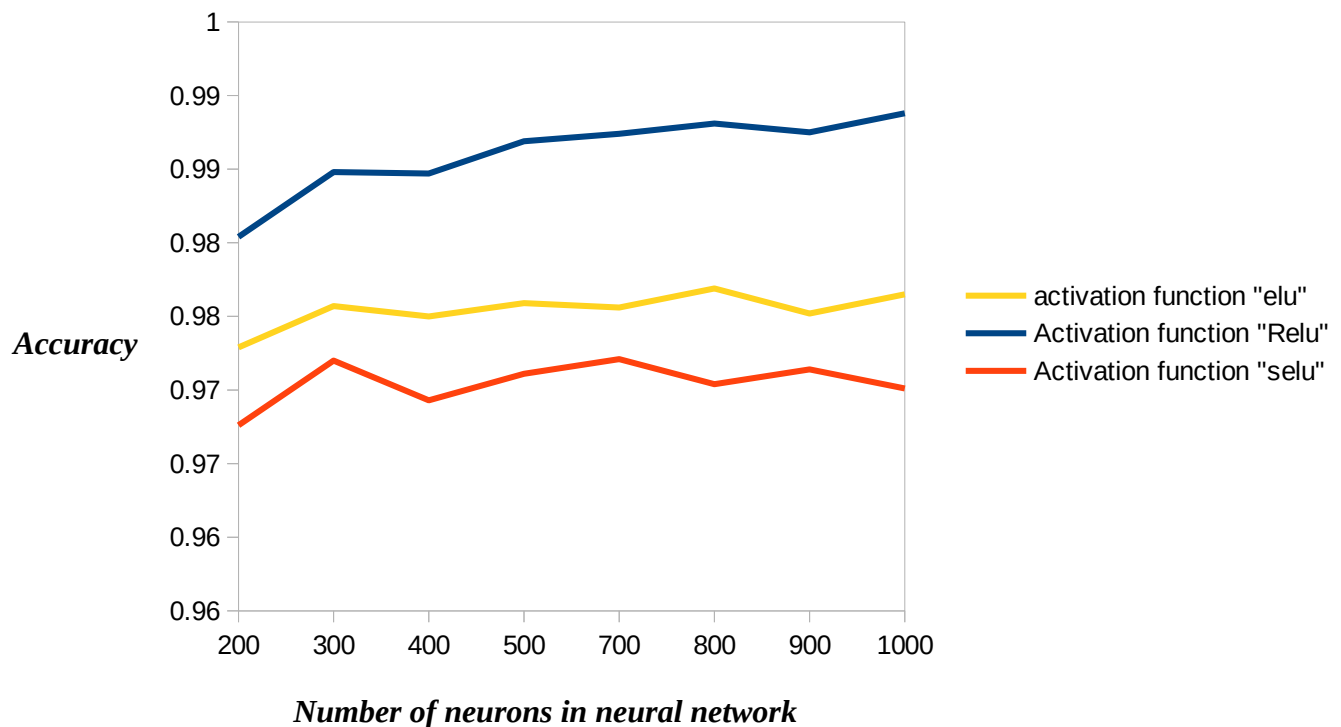
As we *change the number of hidden layer in network* the accuracy of model changed.



After doing so, we analyse that the accuracy of model was decreasing as we were increasing the number of hidden layer. Same is shown in graph representation as number of hidden layer as x-axis and accuracy as the y-axis.

Though accuracy was decreasing using all three activation functions, Relu show us the best accuracy among all the activation functions we use to analyse. 'ReLU' short hand for Rectified Linear Unit.

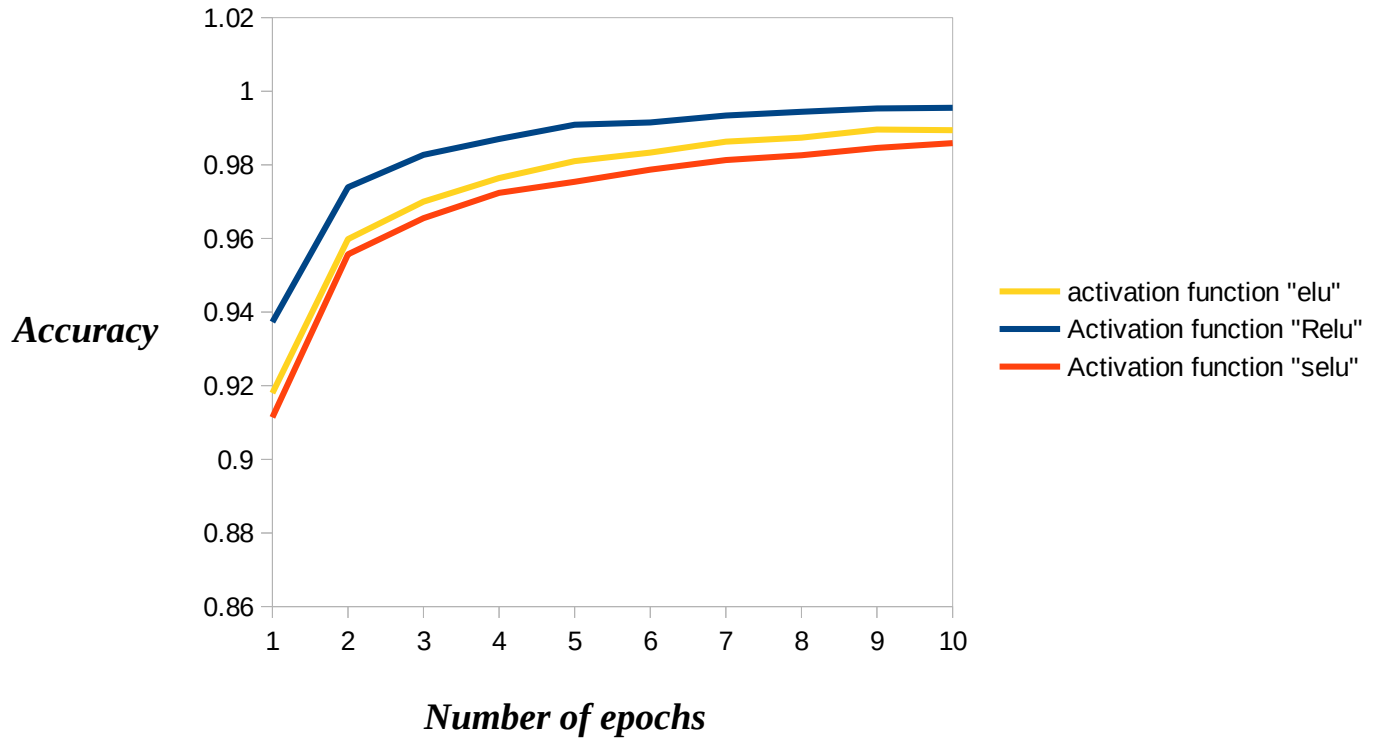
On *increasing the size of neural network that is increasing the neurons in network*, we found this.



On doing so, we analyse that the accuracy of our model was increasing on increasing the neurons in the network. Same is shown in above graph representation x-axis as the number of neurons and y-axis as the accuracy.

Further more we analyse the same on every known valid activation functions the 'relu' function shows the best result here also. From this analysis, we deduce that increasing the numbers of neurons cause the increase in efficiency of network as no of neurons act same as human neuron cells which helps to remember things, thus these neurons of network help in the remembering the pattern of image and help in learning.

And, *on increasing the number of epochs* , we analyse this.



On doing so, we analyse that as we increase the no of epoch our model gets better and better. The accuracy of model/network is strictly increases on increasing the number of epochs.

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. The number of epochs is the number of complete passes through the training dataset. The more epochs we set the more our model gets better and yield better accuracy.

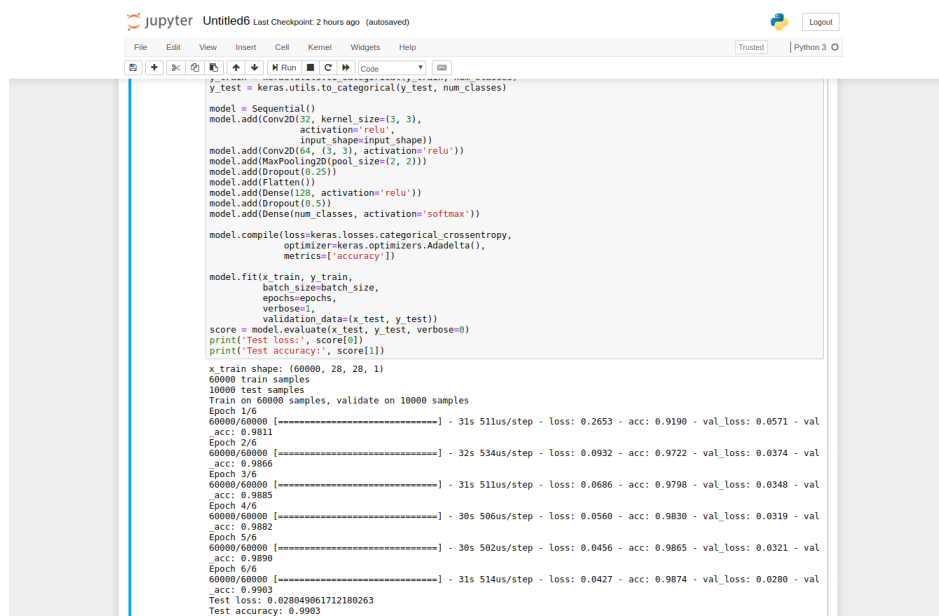
4. Convolution Neural Network :

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign weights and biases to various aspects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics.

A CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

We have created a CNN model which works on data of MNIST dataset using keras library. We've also used keras library for all the calculations of hidden layers in CNN. We first trained training dataset of MNIST and then used test images of MNIST library to test accuracy of our algorithm.

For training purpose first we have used 'relu' activation function and 'adadelata' optimizer. We then changed different parameters to analyze our algorithm. We analyzed our algorithm by changing parameters such as batch size, epoch, Kernel activation functions, number of hidden layers, loss function and optimizer.



```
jupyter Untitled6 Last Checkpoint: 2 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/6
60000/60000 [-----] - 31s 511us/step - loss: 0.2653 - acc: 0.9190 - val_loss: 0.0571 - val
acc: 0.9811
Epoch 2/6
60000/60000 [-----] - 32s 534us/step - loss: 0.0932 - acc: 0.9722 - val_loss: 0.0374 - val
acc: 0.9866
Epoch 3/6
60000/60000 [-----] - 31s 511us/step - loss: 0.0686 - acc: 0.9798 - val_loss: 0.0348 - val
acc: 0.9885
Epoch 4/6
60000/60000 [-----] - 30s 506us/step - loss: 0.0560 - acc: 0.9830 - val_loss: 0.0319 - val
acc: 0.9882
Epoch 5/6
60000/60000 [-----] - 30s 502us/step - loss: 0.0456 - acc: 0.9865 - val_loss: 0.0321 - val
acc: 0.9890
Epoch 6/6
60000/60000 [-----] - 31s 514us/step - loss: 0.0427 - acc: 0.9874 - val_loss: 0.0289 - val
acc: 0.9903
Test loss: 0.028049961712180263
Test accuracy: 0.9903
```

	1	2	3	4	5	6	7
Batch	128	128	128	50	128	128	128
Epoch	6	12	6	6	6	6	6
Activation	relu	relu	tanh	relu	relu	relu	relu
Hidden Layers	3	3	3	3	3	3	2
Loss	categorical_crossentropy	categorical_crossentropy	categorical_crossentropy	categorical_crossentropy	mean_squared_error	mean_squared_error	mean_squared_error
Optimizer	Adadelta	Adadelta	Adadelta	Adadelta	Adadelta	Adam	Adam
Accuracy	99.01	99.23	98.81	99.05	98.27	98.91	98.74

Activation function:

In CNN activation function is responsible for transforming weighted input from the node to output for that input.

Here, we have used two activation functions

1) Relu : A node that use the rectified linear activation function is as a **rectified linear activation unit**, or ReLU.

2)tanh: This function is defined by

$$\begin{aligned}
 g_{\tanh}(z) &= \frac{\sinh(z)}{\cosh(z)} \\
 &= \frac{e^z - e^{-z}}{e^z + e^{-z}}
 \end{aligned}$$

Epoch:

Epoch is number of times we are training our algorithm with same dataset.

If number of epoch are less then we may end up with underfitting problem.

In our example, when we changed epoch from 6 to 12 our accuray of algorithm increased from 99.01% to 99.23%.

Also when training we have observed that as epoch size is decreased, accuracy decreases.

Batch size:

batch size is number of image samples feeded to CNN at a time. When batch size is too low or too high it may lead to underfitting.

Hidden layers:

Hidden layers are the main computational unit of CNN. We have used 3 hidden layers in our algorithm. When we decreased number of hidden layers, accuracy of our algorithm decreased which we can see in column 7.

Loss function:

We have used two loss functions: `categorical_crossentropy` and `mean_squared_error`.

As we can see in above table, `categorical_crossentropy` is best for our algorithm.

Optimizer:

From the two optimizers that we have used for our algorithm, i.e. `adadelta` and `adam`, we found that `adadelta` is superior to `adam`.

Conclusion

After exercising and analysing on logistic regression , multilayer perceptron , deep neural network and deep convolutional neural network by various parameters , we conclude as follows,

1. Logistic regression gives best result in term of accuracy, when we increase training epochs, reduce the batch size and keep learnig rate in range [0.2 – 0.4].
2. Multi-layer perceptron gives best result in term of accuracy, when we use 'ReLU' activation function. The accuracy was 0.985 .
3. Deep neural network gives best result in term of accuracy, when we use 'ReLU' activation function, keep less hidden layer, maximise the size of neural network and epochs.
4. from the above observations we found that as the number of epochs and hidden layers increases the accuracy of algorithm increases.