Digit Recognition using MNIST Dataset

M Lohith, Utkarsh Singh, Mahima Malik, Priyanshu Singh, Anasuya Jana

Introduction

The MNIST database is a large database of handwritten digits. The input is a gray level image, the intensity level of which varies from 0 to 255, where each image in MNIST is already normalized to 28x28. MNIST consists of two datasets, one for training (60,000 images) and one for testing (10,000 images).

The digit images in the MNIST dataset have 28 x 28 pixels. These pixels together with the bias term is the number of features. That means that each sample feature vector will have 784 + 1 = 785 features that we will need to find 785 parameters for. The optimization loop will be over the 10 classes (0-9, in this the input image needs to be classified) and will produce a matrix A of optimized parameters by minimizing a cost function for each for the 10 classes. Each column of the 10 columns A will be a model parameter vector corresponding to each of the 10 classes. The resultant vector with the highest probability is the prediction from the model.

The models we use are:

- Logistic Regression
- Multi Layer Perceptron
- Deep Learning Neural Network
- Deep Learning Convolutional Neural Network

Models

Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. It transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes. The function maps any real value into another value between 0 and 1.

In the linear regression, we are performing *classification*, where we want to assign each input *X* to one of 10 classes. The basic modeling idea is that we're going to linearly map our input X on to 10 different real valued outputs. Then, before outputting these values, we'll want to normalize them so that they are non-negative and sum to 1. This normalization allows us to interpret the output y(hat) as a valid probability distribution.

Batch Training:

Let us suppose we have *d* features and *k* output labels. Now we can see the shapes of the variables as follows:

$$\underset{1\times k}{\boldsymbol{z}} = \underset{1\times d}{\boldsymbol{x}}\ \underset{d\times k}{W} + \underset{1\times k}{\boldsymbol{b}}$$

To label the output we are using one-hot encoding, for example y(hat) = 5 would be $\boldsymbol{y}(\text{hat})_{one-hot}$ = [0,0,0,0,1,0,0,0,0,0] when one-hot encoded for a 10-class classification problem. So $y(hat)$ = softmax($\boldsymbol{z}$) becomes

$$\underset{1\times k}{\hat{\boldsymbol{y}}_{one-hot}} = \text{softmax}_{one-hot}\left(\underset{1\times k}{\boldsymbol{z}}\right)$$

Now when we provide a batch of m training samples as input, we would have matrix $X_{m\times d,}$ where each row corresponds to one training sample.

$$X = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_m \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & x_{m3} & \dots & x_{md} \end{bmatrix}$$

Under this batch training situation, $\boldsymbol{y(hat)}_{one-hot}$ =softmax($\boldsymbol{z}$) turns into

$$Y = \text{softmax}(Z) = \text{softmax}(XW + B)$$

where matrix $B_{m\times k}$ is formed by having m copies of $\boldsymbol{b}$ as follows:

$$B = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b} \\ \vdots \\ \boldsymbol{b} \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 & \dots & b_k \\ b_1 & b_2 & b_3 & \dots & b_k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_1 & b_2 & b_3 & \dots & b_k \end{bmatrix}$$

Each row of matrix $Z_{m\times k}$ corresponds to output of one training example. The softmax function operates on each row of matrix Z and returns a matrix $Y_{m\times k}$, each row of which corresponds to the one-hot encoded prediction of one training example, considering the label with highest probability as output.

Loss function:

We can control two parameters in our cost function:
m(weight) and b(bias).

We calculate the partial derivatives of the cost function with respect to each parameter to know the impact each one has on predicted output and store the results in a gradient.

Loss function can be defined as:

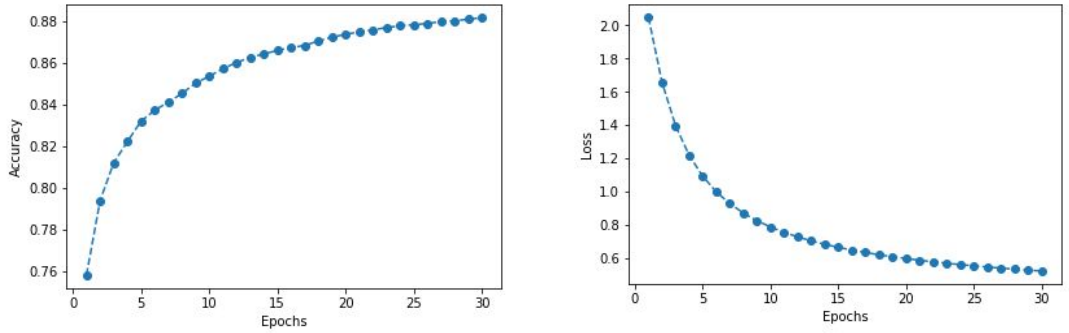
$$f(m,b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

The gradient is calculated as below:

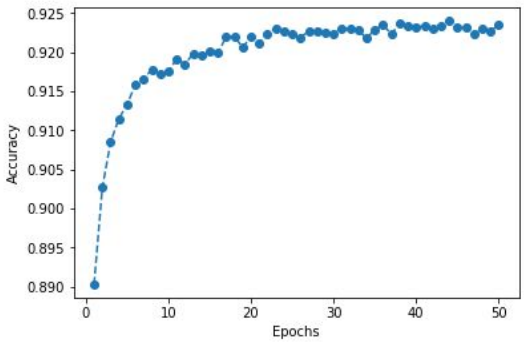
$$f'(m,b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N}\sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

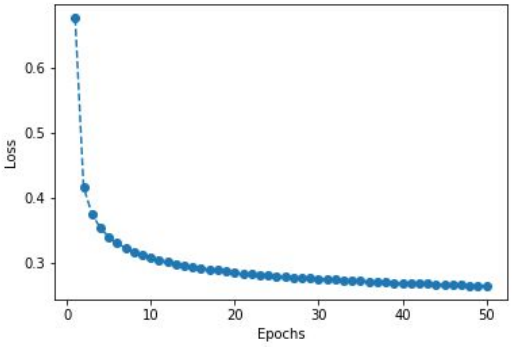
Result:

For Learning rate = 0.001, Accuracy = 88.18

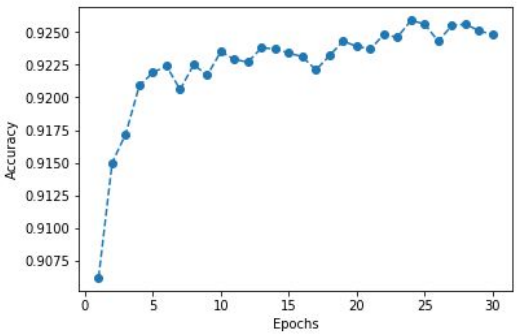


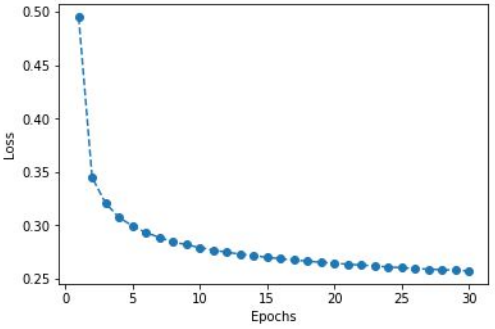
For Learning Rate = 0.05, Accuracy = 92.36

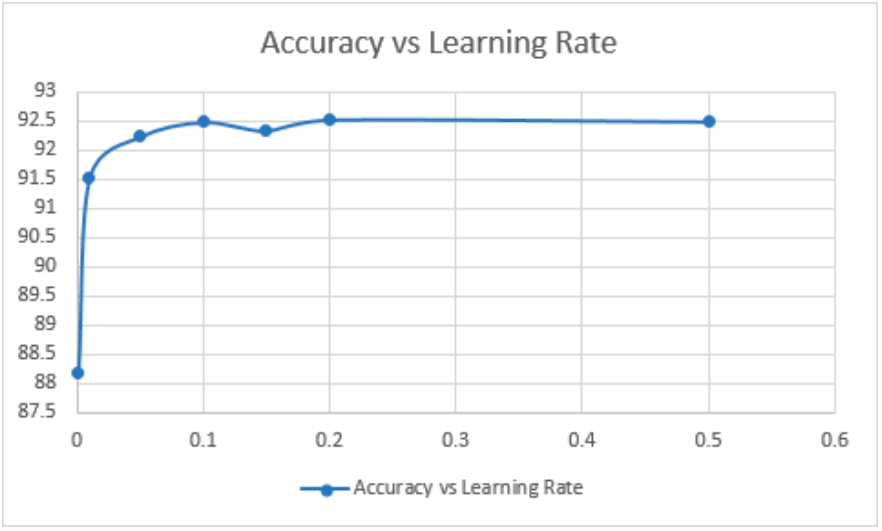




For Learning Rate = 0.15, Accuracy = 92.34







For logistic regression, with respect to learning rates, accuracy is increasing, and the model is performing best at learning rate 0.2, after which it is sort of constant.

Training and Testing the Model for below models:

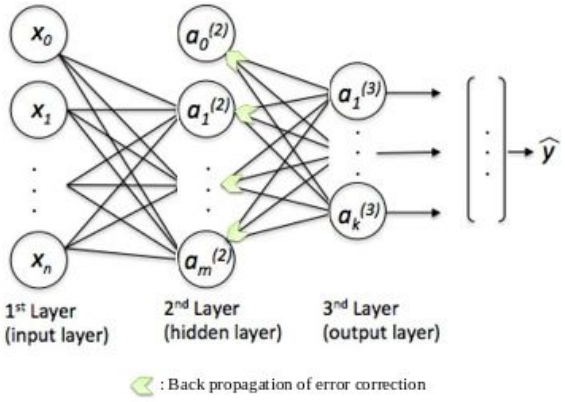
Below models are trained using tensorflow executing the following steps:

- Firstly, we launch the graph in a session initializing all the variables and other parameters.
- Now we run the session, which will execute the variables that were initialized in the previous step and evaluates the tensor.
- Next, run a for loop for all the training samples as epochs, and another loop for batches (that is dividing the total number of images by the batch size) in which we divided the training samples.
- Input the images based on the batch size to input training batch(x_batch) and their respective labels in output training batch (y_batch).
- Feed the placeholders x and y the actual data in a dictionary and run the session by passing the cost and the accuracy that you had defined earlier. It returns the loss (cost) and accuracy.
- You can print the loss and training accuracy after each epoch (training iteration) is completed.
- After the completion of all epochs, we can plot the graph of accuracy and loss with respect to the number of epochs.

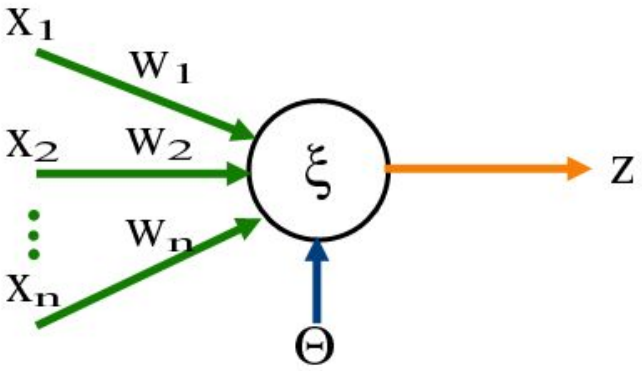
After each training iteration is completed, you run only the accuracy by passing all the 10000 test images and labels. This will give you an idea of how accurately your model is performing while it is training.

Multi-Layer Perceptron

The Multilayer Perceptron(MLP) is a feed-forward layered network, where the data circulates in one way, from the input layer to the output layer through multiple hidden layers. The input layer contains the inputs features of the network. The first hidden layer receives the weighted inputs from the input layer and sends data from the previous layer to the next one. The use of additional layers makes the perceptron able to solve nonlinear classification problems. The output layer contains the classification result. It consists of four stages: initializing weights, feed forward, back propagation of errors and weight update.

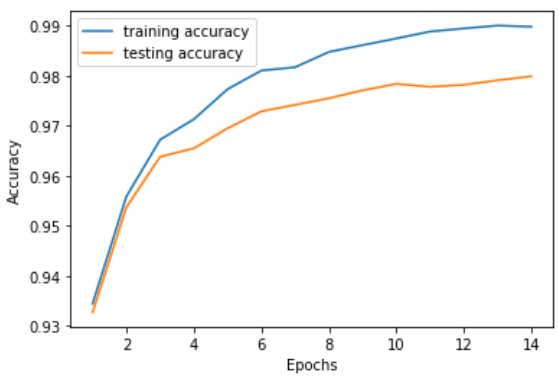


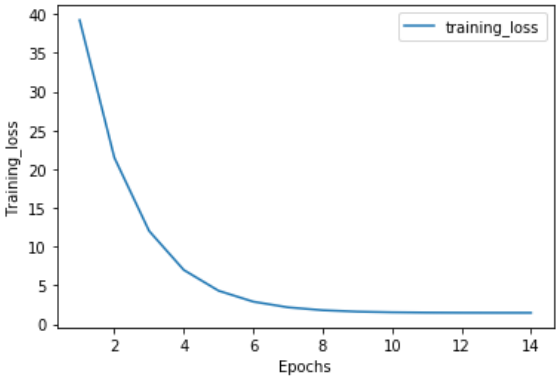
Perceptron/neuron is a linear model which takes multiple inputs and produce an output. Perceptron takes a bunch of inputs multiply them with weights and add a bias term to generate an output.



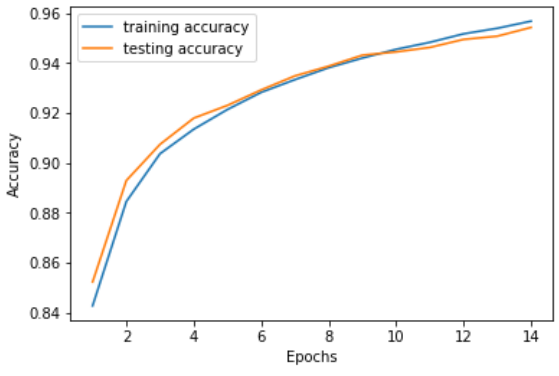
Results:

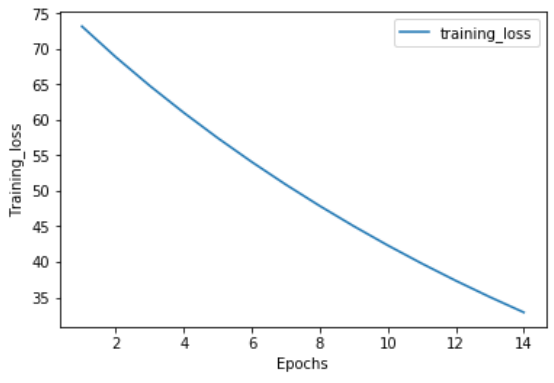
Learning Rate: 0.001, Accuracy = 98.0



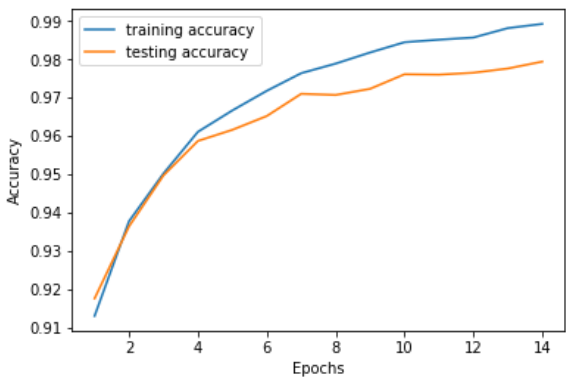


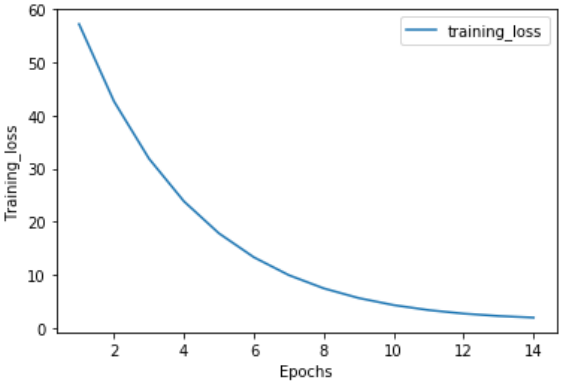
Learning Rate: 0.0001, Accuracy = 95.4

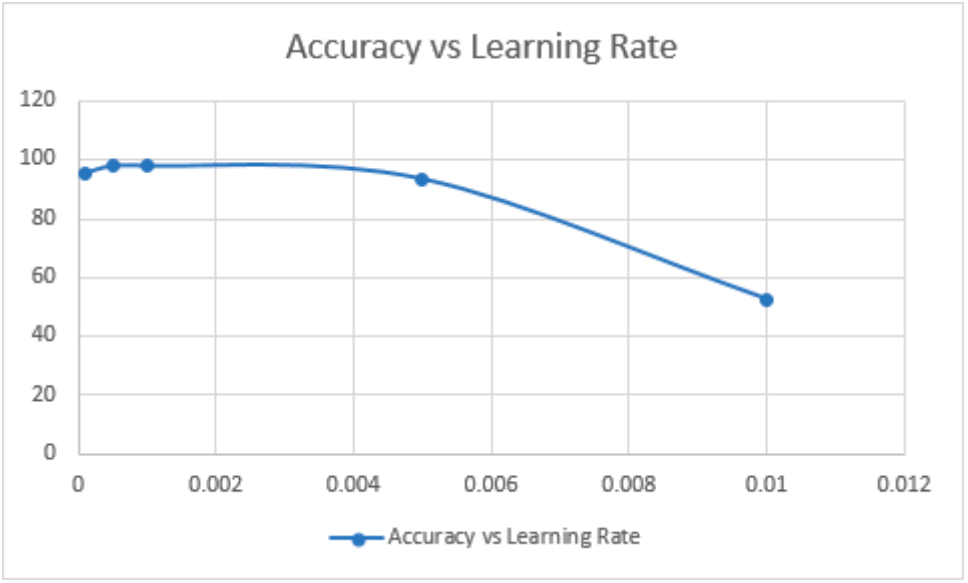




Learning Rate: 0.0005, Accuracy = 97.9







For multi-layer perceptron, with respect to learning rates, accuracy is increasing initially, and decreasing drastically henceforth, the model is performing best at learning rate 0.001 with accuracy 98.

Deep Neural Network

Deep neural networks is network that have an input layer, an output layer and at least one hidden layer in between. Each layer performs specific types of operation 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. Deep neural networks use sophisticated mathematical modeling to process data in complex ways. In our model we use ReLU for Activation Function.

Activation function

It is a function that is used to get the output of a node. It is also Known as Transfer Function. Here we use Dense matrix.

Forward propagate

The natural step to do after initialising the model at random, is to check its performance. We start from the input we have, we pass them through the network layer and calculate the actual output of the model straightforwardly. This step is called forward-propagation, because the calculation flow is going in the natural forward direction from the input -> through the neural network -> to the output.

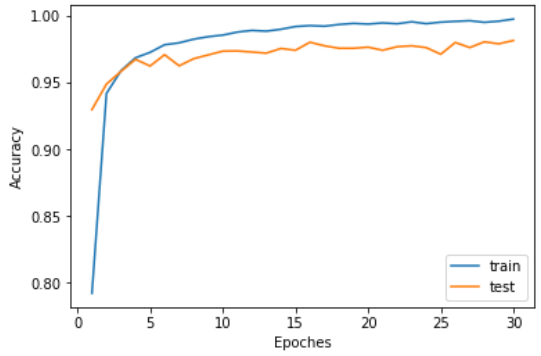
Backpropagation

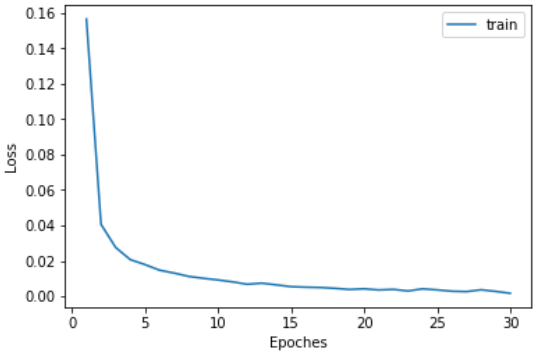
A neural network propagates the signal of the input data forward through its parameters towards the moment of decision, and then backpropagates information about the error, in reverse through the network, so that it can alter the parameters. This happens step by step:

- The network makes a guess about data, using its parameters
- The network’s cost is measured with a loss function
- The error is back propagated to adjust the wrong-headed parameters

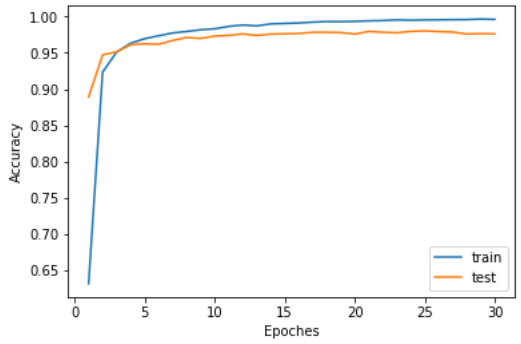
Result:

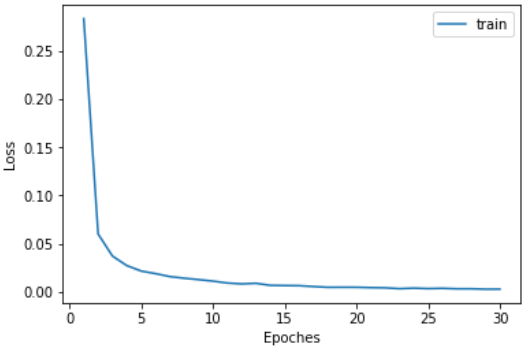
Learning Rate:0.04, Accuracy=98.2



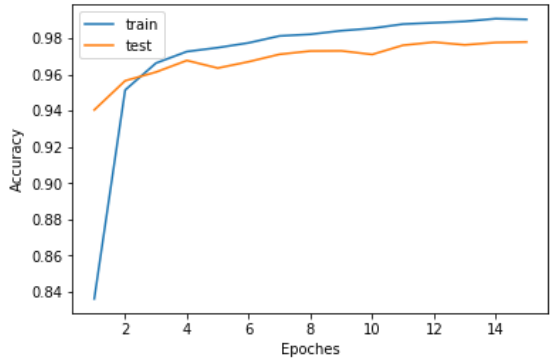


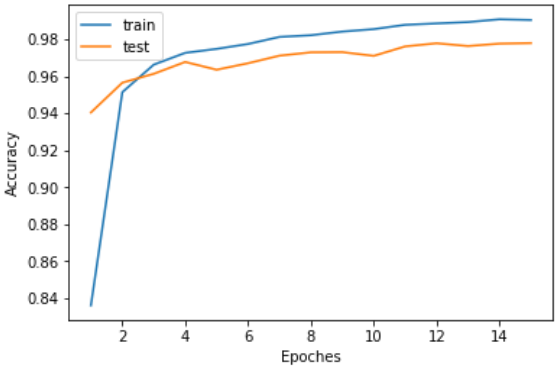
Learning Rate:0.004, Accuracy=97.6

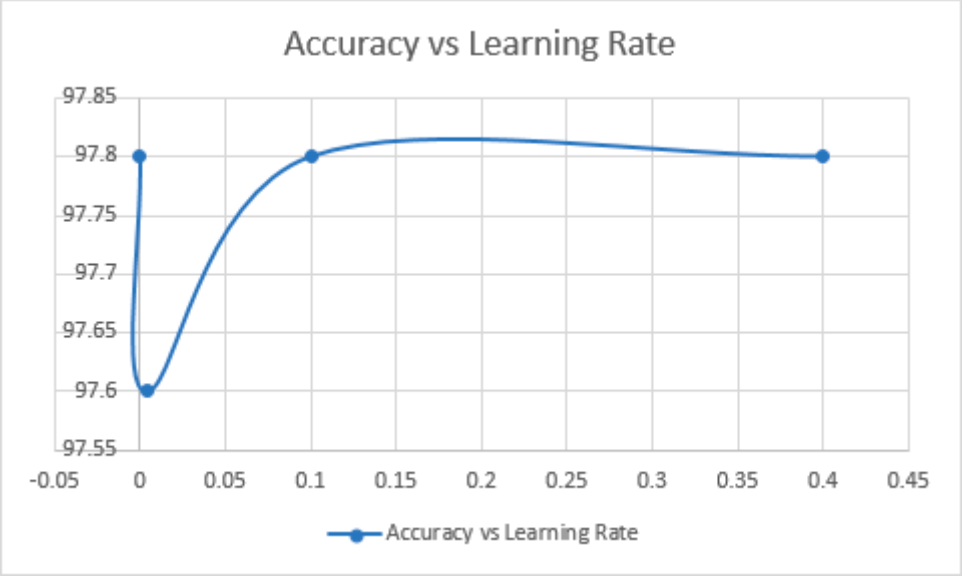




Learning Rate:0.1, Accuracy=97.8







For DNN, the accuracy decreases initially, and increases and thereafter remains almost constant from the learning rate 0.1, with accuracy 97.8.

Convolutional Neural Network

CNN uses multilayer perceptrons to do computational work. CNNs use relatively little pre-processing compared to other image classification algorithms. This means the network learns through filters that in traditional algorithms were hand-engineered. So, for image processing task CNNs are the best-suited option.A convolutional neural network consists of several layers:

Convolution Layer : It extract features from the input feature matrix. The parameters consist of a set of learnable filters which are used for recognizing patterns in the input matrix. The process involves sliding the filter over the input matrix and taking the dot product between the filter and chunks of the input matrix.

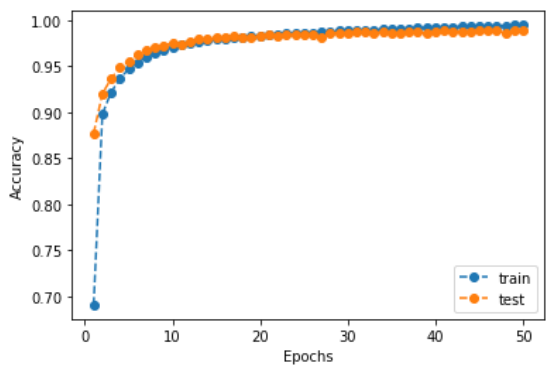
Pooling Layer: Pooling layer reduce the size of feature maps by using some functions to summarize sub-regions, such as taking the average or the maximum value. Pooling works by sliding a window across the input and feeding the content of the window to a pooling function.The purpose of pooling is to reduce the number of parameters in our network (hence called down-sampling)

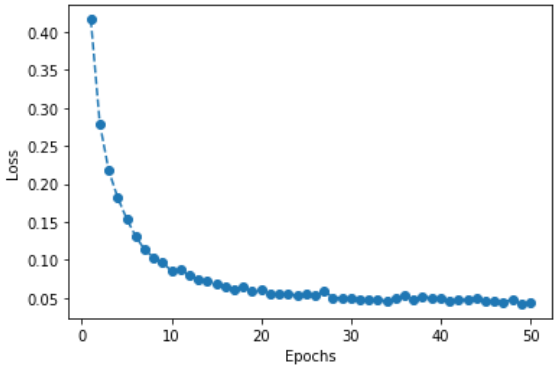
ReLU Layer(Rectified Linear Unit): The non-linear operation in ReLU are applied per pixel and replaces all negative pixel values in the feature map by zero.

Fully Connected Layer: A fully connected layer takes all neurons in the previous layer (fully connected, pooling, or convolutional) and connects it to every single neuron it has. Adding a fully-connected layer is also another way of learning non-linear combinations of these features. Most of the features learned from convolutional and pooling layers may be good, but combinations of those features might be even better.

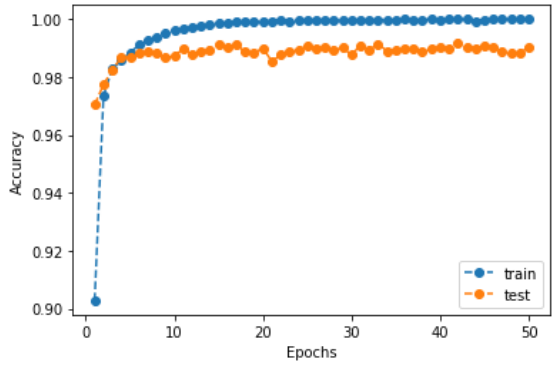
Results:

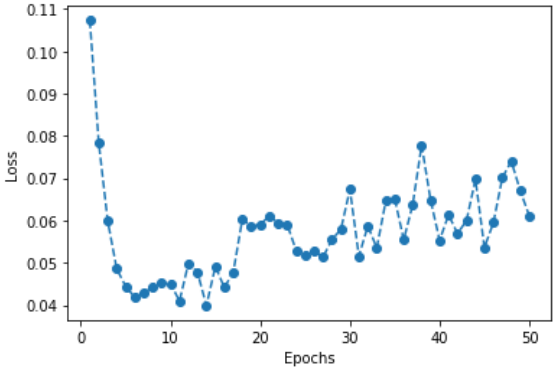
Learning Rate: 0.0001, Accuracy = 99.19



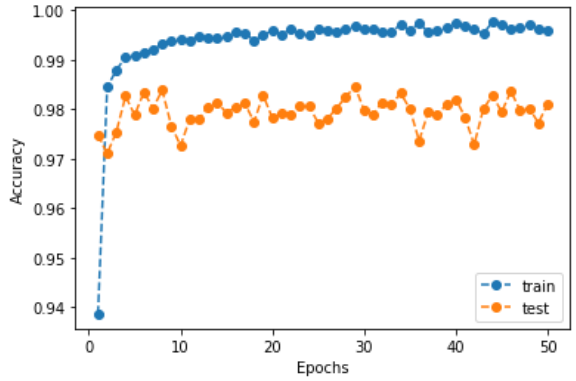


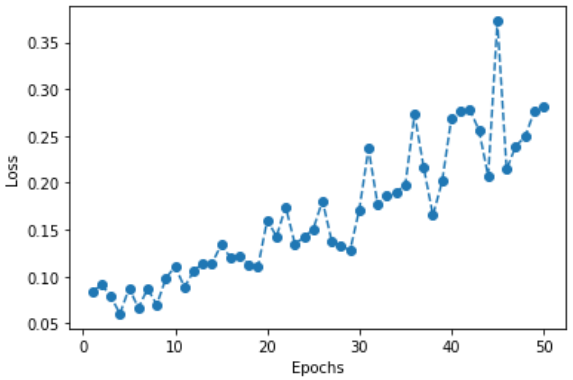
Learning Rate: 0.001, Accuracy = 99.04

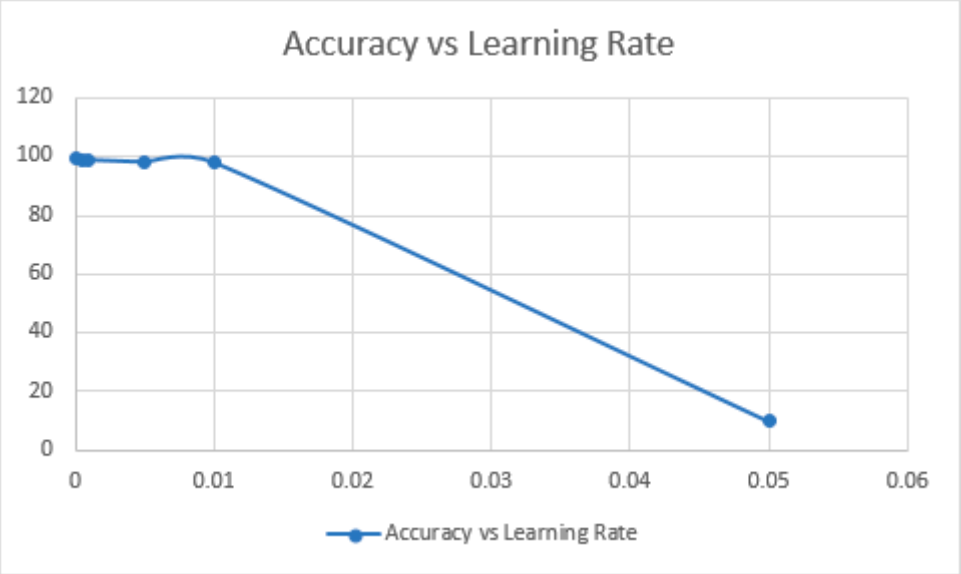




Learning Rate: 0.01, Accuracy = 98.10







For CNN, the accuracy is decreasing as we are increasing the learning rate and the model performs best at learning rate = 0.0001 with accuracy = 99.19

Overfitting:

Overfitting occurs when the model wants to fit most of the points in the data including noise which makes the model incapable to fit additional data or predict observations. An overfitted model will have more parameters than required to predict. We control this by reducing the learning rate of the model as the learning rate comes closer to 100% the model tries to fit all the data and becomes incapable for prediction.

Underfitting:

Underfitting occurs when the model cannot fit the adequate amount of data to capture the underlying structure which makes model incapable to make predictions. As the learning rate comes closer to 0% our model is very unlikely to change the structure and becomes incapable for predicting the observations.

Conclusion

We can observe that as the Deep Convolutional Neural Network gives best accuracy than any other model. CNN has the advantage that it can automatically detects important feature without any human interventions. And Deep Neural Networks are least affected by learning rate because it has a huge number of neurons/perceptrons at each hidden layer and number of layers also more.