CS547

Namit Kumar 160101046
Abhinav Mishra 160101005
Saurabh Bazari 160101061
Ritik Agrawal 160101055
G Sharath Kumar 150101023

Assignment 1

MNIST Handwritten digits

Classification

Problem Statement

Consider the The MNIST database of handwritten digits available at "*http://yann.lecun.com/exdb/mnist/*" and train various classifiers using the following methods to classify the given image sample 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

Exercise on various parameters (if applicable) such as number of hidden layers, type of activation functions, number of convolution kernels, size of convolution kernels.

Language / libraries used

- Programming language - Python 3
- Libraries
    - Scikit-Learn library is used for implementing the Logistics Regression model
    - Keras API is used for building all neural network models. Keras uses Tensorflow and Theano libraries in its backend.
    - Matplotlib was used for visualisation of results using graphs.

Dataset

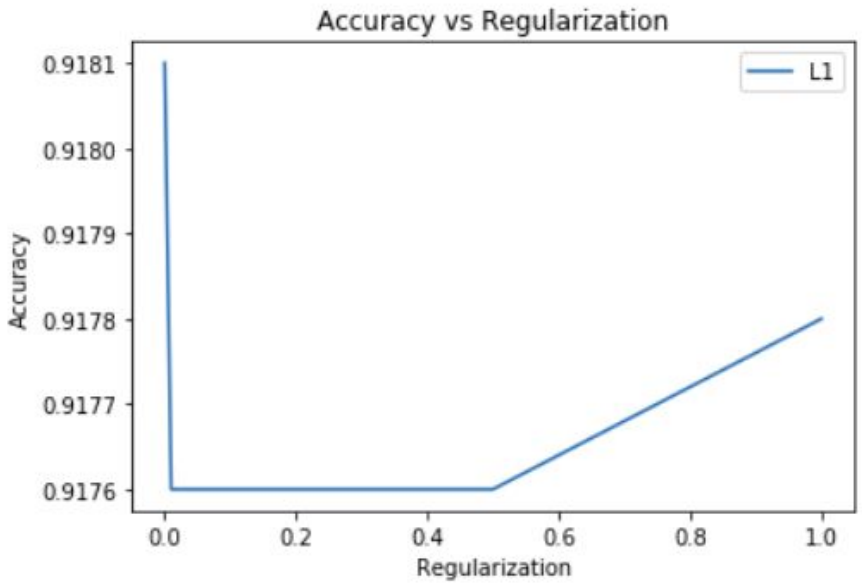
The MNIST data contains 60,000 images of numerical digits to be used as training data. The size of images are 28 by 28 pixels in size. The test data of the MNIST data set contains 10,000 images also 28 by 28 pixels in size and greyscale . Evaluation of our model performance to recognize digits is done using the test data. Each training input is regarded as a 28×28=784-dimensional vector and every image in the dataset is labelled with its correct classification among the 10 classes (digits 0-9). The assignment involved building 4 models and carrying out experiments to figure out value of hyperparameters and learn about dependencies on other parameters.

Logistic Regression:

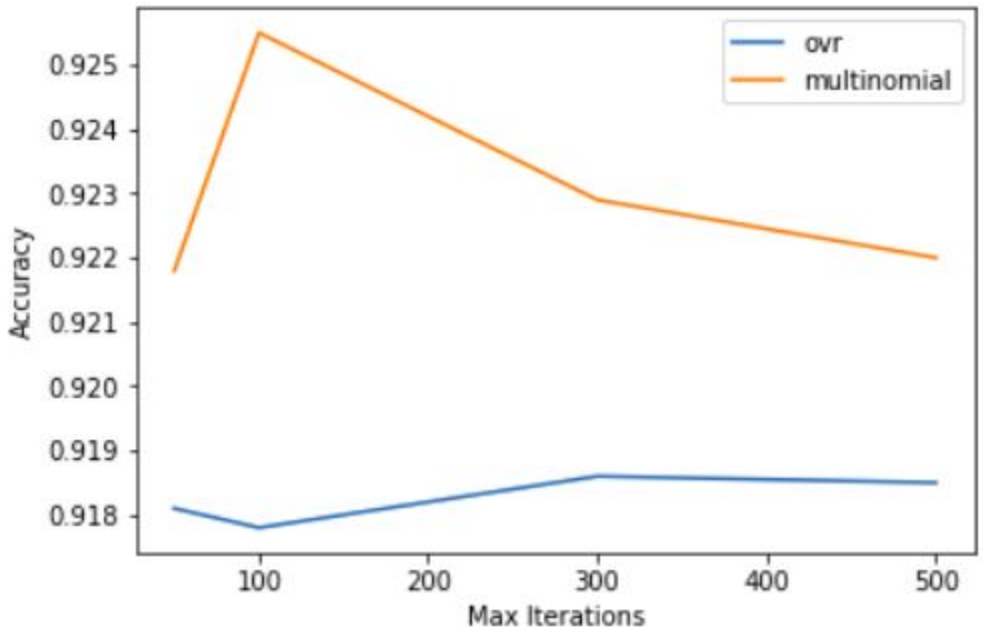
Using the logistic model we could classify the input into 2 or more output classes based on the probability of it belonging to the respective classes. In our case we need to classify the input into 1 of the 10 possible classes of digits.

We built the model in multiple scenarios with different L1 regularization and inverse regularization constants.Currently the multi_class is set to ‘ovr’ . Here are the results -

Best Accuracy was observed on C = 0.0001 inverse regularization constant.



We experimented with different multi_class using multinomial and one vs rest(ovr) for number of iterations 50,100,300 and 500.



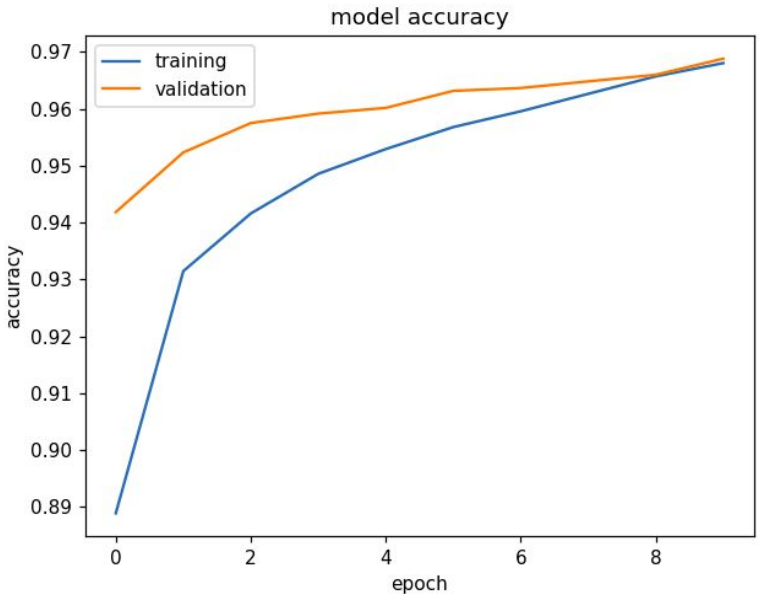
The performance of multinomial was on an average better than ovr.
The iteration value at which maximum accuracy for multinomial was achieved was found to be 100 whereas for ovr it was 300.

Multi Layer Perceptron:

For the purpose of using simple neural network (not CNN) we had to reshape our input from 2D to 1D , thus our model won't be able to learn the spatial characteristics of the image.

We first build the model with the following properties and analyze its performance by varying different parameters.

- Neural network with a single hidden layer consisting of 512 neurons and 784 neurons in the input layer and then finally 10 neurons in the output layer.
- Activation function of our hidden layer was Sigmoid and Softmax was used as activation function for the output layer.
- Value of epochs for training the dataset was 10.
- Batch Size is set to be 128.
- The optimizer used is adagrad.

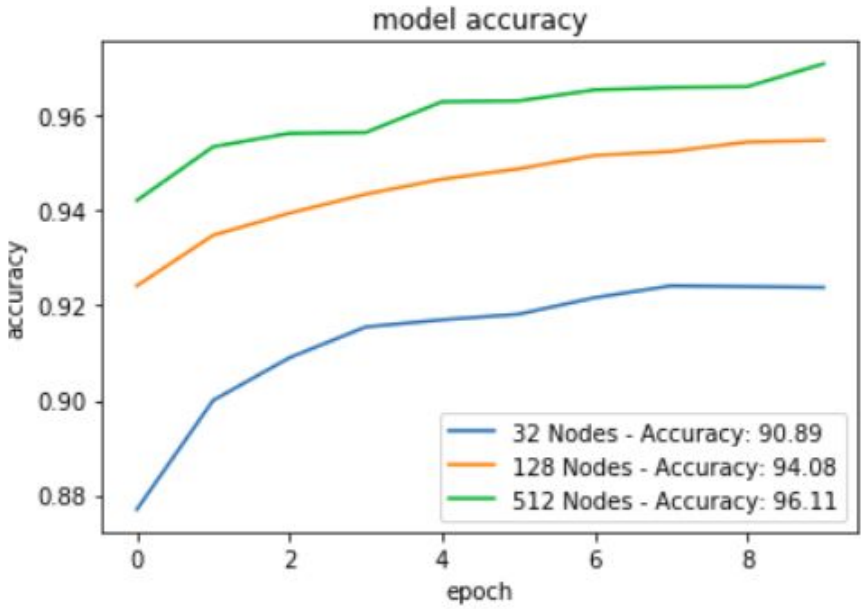


The Test Accuracy and Test Loss observed for this setting was *0.962* and *0.13*.

Variations:

Number of nodes in the layer.

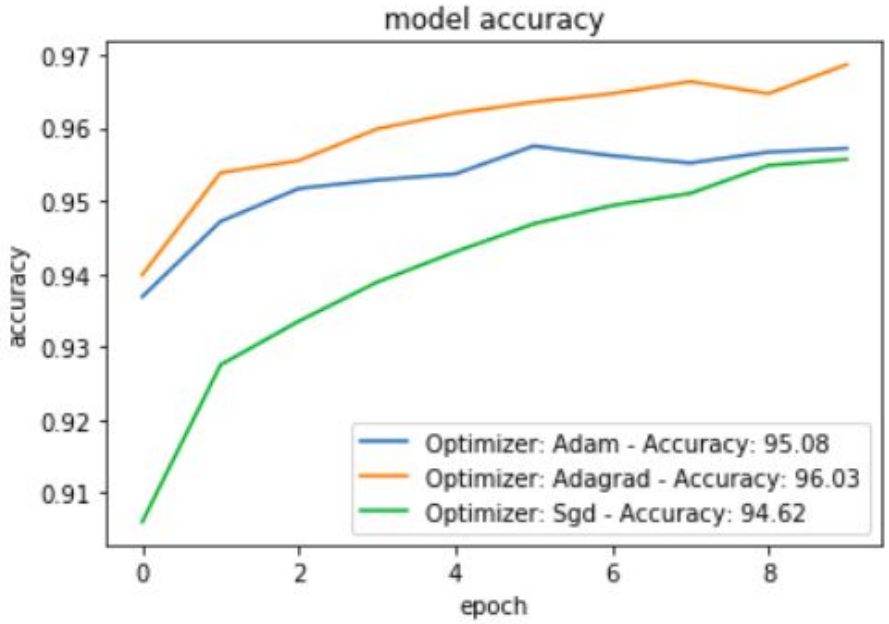
We trained and evaluated our model by using 32, 128 and 512 nodes in a single layer



We observe improvement in the accuracy of our model on increasing the number of nodes . This behaviour can be explained as increasing the number of nodes means our model can take more complex decisions. As the number of nodes increases the number of parameters increases resulting in more sophisticated decision-making process.

Type of Optimizer

We trained and evaluated our model by using 3 different optimizers namely sgd, adagrad and adam.

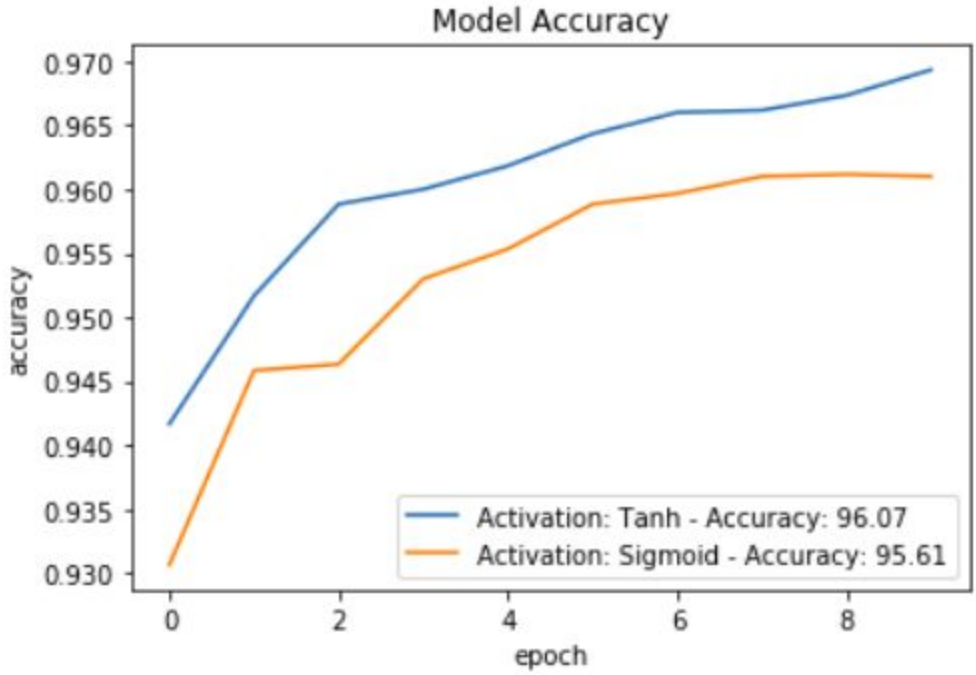


We observe that using different optimization algorithm in our model resulted in different accuracy. The accuracy observed was highest using adagrad followed by adam and finally sgd.

The problem with SGD is that if there is a saddle point that surrounded by plateau, it will be very difficult for SGD to get out since the gradients are close to zero in such places.Adagrad (Adaptive Gradient) adapts learning rate for each parameter separately. If features of parameter occur rarely we make bigger step, i.e. it has bigger learning rate, than parameters, whose features occur often — for such parameters we do smaller step, i.e. it has a smaller learning rate.The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for carefully search.

Type of Activation Function

We trained and evaluated our model by applying different activation functions on the hidden layer however activation function for the output layer was always taken as softmax.



We observe that using different Activation function in our model resulted in different accuracy. The accuracy observed was highest using sigmoid.

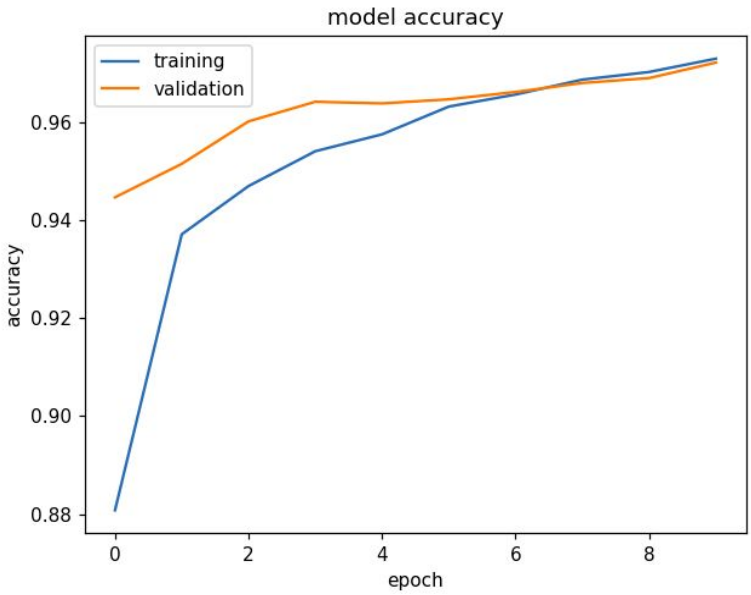
Deep Neural Network:

For the purpose of using simple neural network (not CNN) we had to reshape our input from 2D to 1D , thus our model won't be able to learn the spatial characteristics of the image.

We first build the model with the following properties and analyze its performance by varying different parameters.

- Neural network with two hidden layer consisting of 512 neurons and 784 neurons in the input layer and then finally 10 neurons in the output layer.

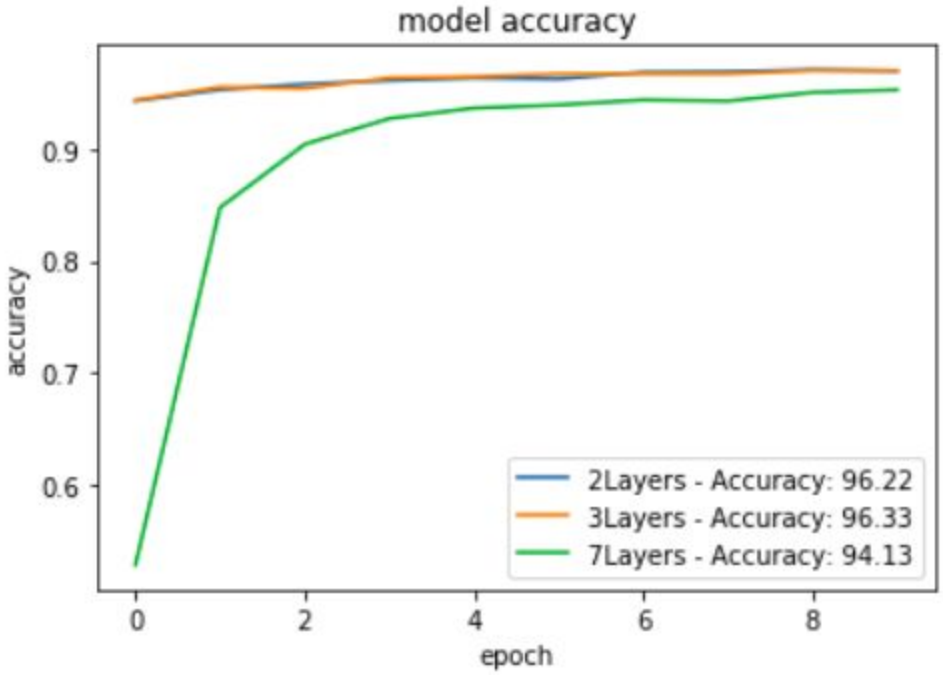
- Activation function of our hidden layer was Sigmoid and Softmax was used as activation function for the output layer.
- Value of epochs for training the dataset was 10.
- Batch Size is set to be 128.
- The optimizer used is adagrad.
- The overfitting problem is also encountered using L2 Regularization



Variations:

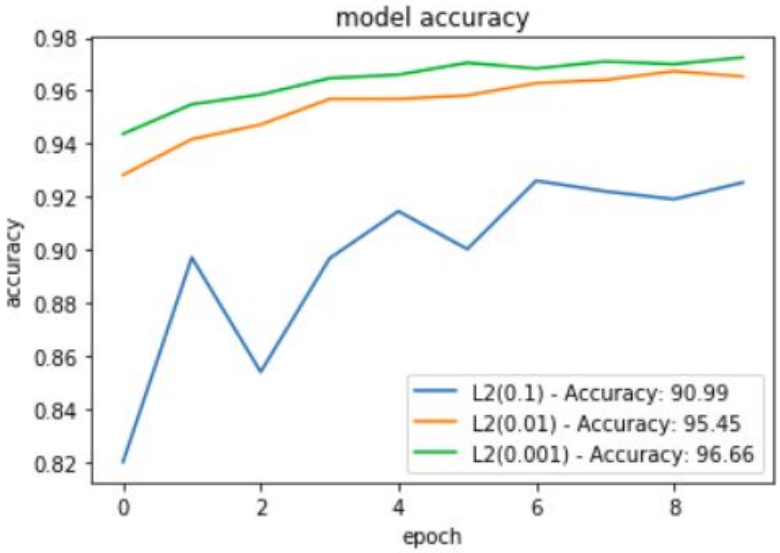
Number of layers.

We trained and evaluated our model by using 2, 3 and 7 hidden layers



We first observe a slight improvement in the accuracy of our model on increasing the number of layers from 2 to 3. This is expected as increasing the number of layers means our model can take more complex decisions. However, further increasing the number of layers to 7 leads to a decrease in performance of our model because of overfitting.

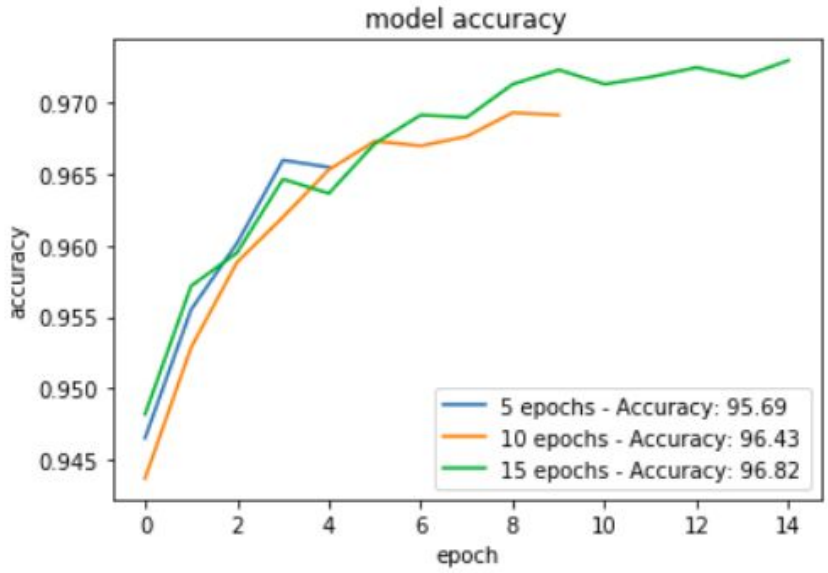
L2 Regularization Value



We made use of L2 regularization to overcome overfitting using values 0.1,0.01 and 0.001
Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent. From our results we observed that the highest accuracy is achieved with L2 value 0.001.

Number of epochs

We trained our model on 5, 10 and 15 epochs and evaluated it.

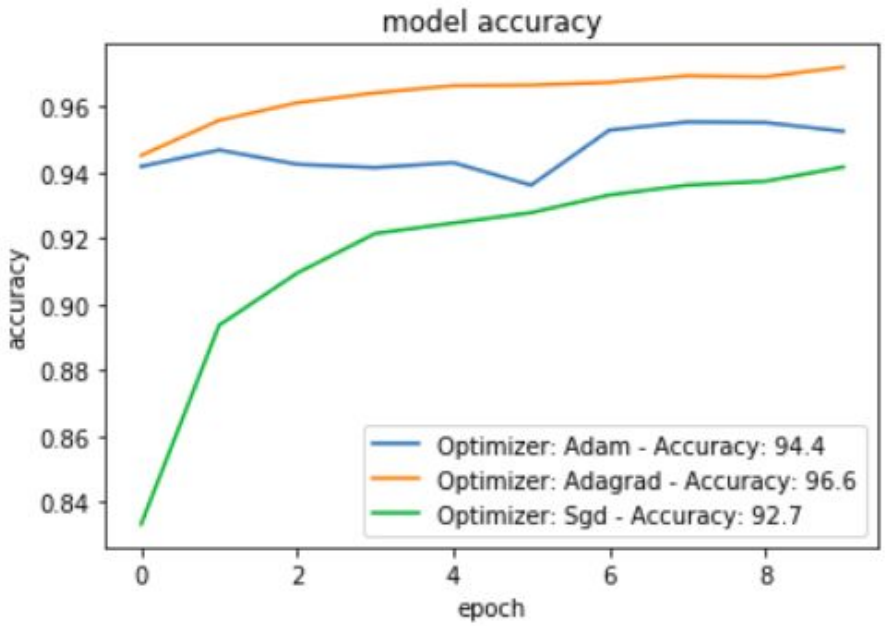


An epoch is one forward pass and one backpropagation of all training examples. Number of epochs can significantly affect the performance of the model while fewer number of epochs leads to underfitting a relatively high number of epochs can lead to overfitting. We observed in our case the accuracy kept increasing with the number of epochs signifying overfitting has not occurred till number of epochs 15.

Type of optimizer

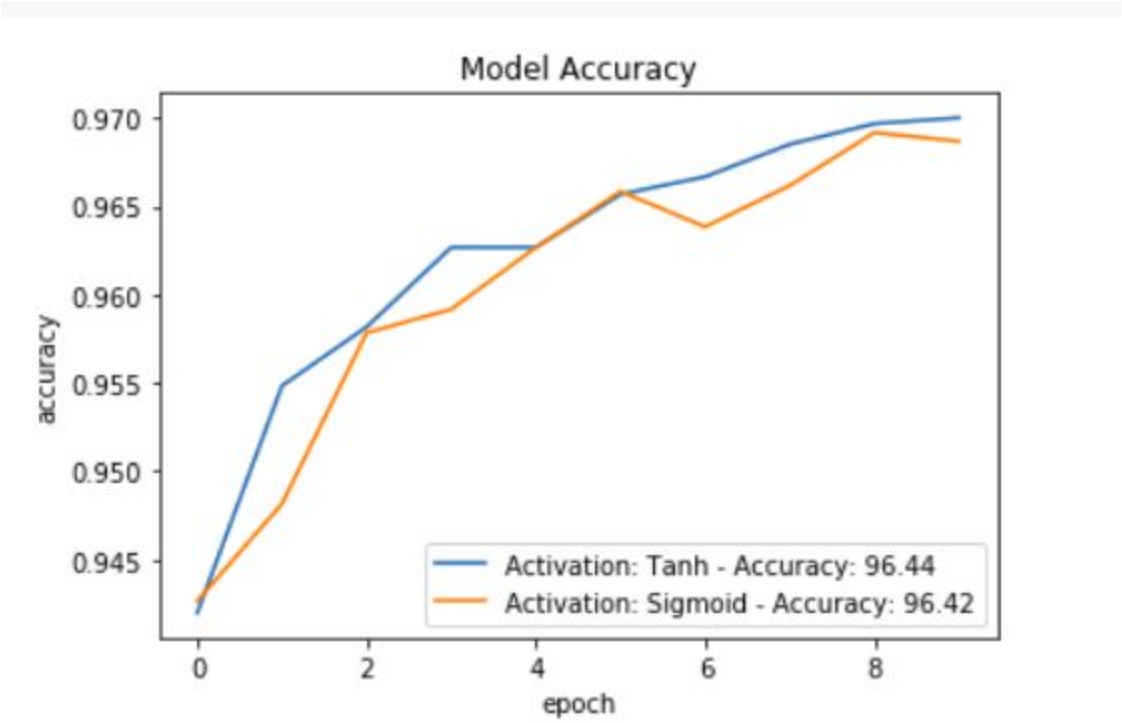
We trained and evaluated our model by using 3 different optimizers namely sgd, adagrad and adam.

We observe that using different optimization algorithm in our model resulted in different accuracy. The accuracy observed was highest using adagrad followed by adam and finally sgd. The problem with SGD is that if there is a saddle point that surrounded by plateau, it will be very difficult for SGD to get out since the gradients are close to zero in such places.Adagrad (Adaptive Gradient) adapts learning rate for each parameter separately. If features of parameter occur rarely we make bigger step, i.e. it has bigger learning rate, than parameters, whose features occur often — for such parameters we do smaller step, i.e. it has a smaller learning rate.The intuition behind the Adam is that we don't want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for carefully search.



Type of Activation Function

We trained and evaluated our model by applying different activation functions on the hidden layer however activation function for the output layer was always taken as softmax.

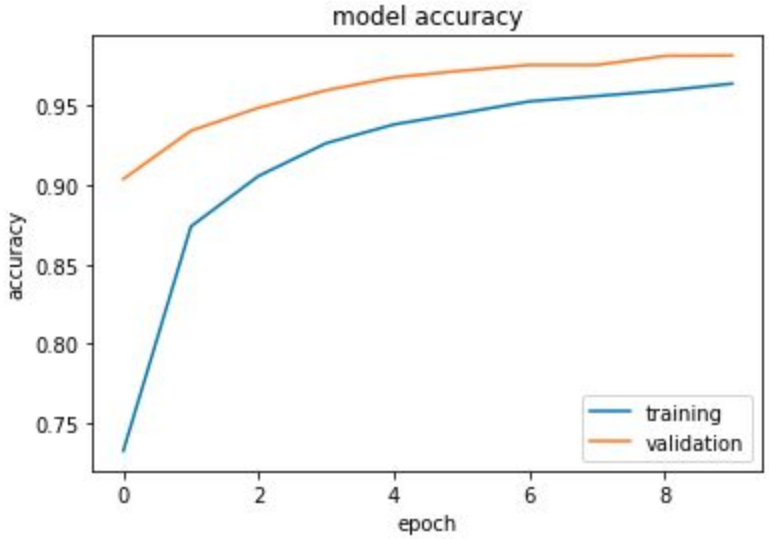


We observed only a slight change in the performance of model using different activation function. The accuracy observed using sigmoid was slightly higher.

Deep Convolutional Neural Network:

We first build the model with the following properties and analyze its performance by varying different parameters.

- Neural network with single hidden convolutional layer with kernel size 3x3 and 32 number of kernels or feature maps
- We use softmax as the activation function of the output layer . ReLU activation function is used for hidden layers.
- Value of epochs for training the dataset was 10.
- Batch Size is set to be 128.
- The optimizer used is adadelta.
- The overfitting problem is also encountered using L2 Regularization and Dropout Techniques

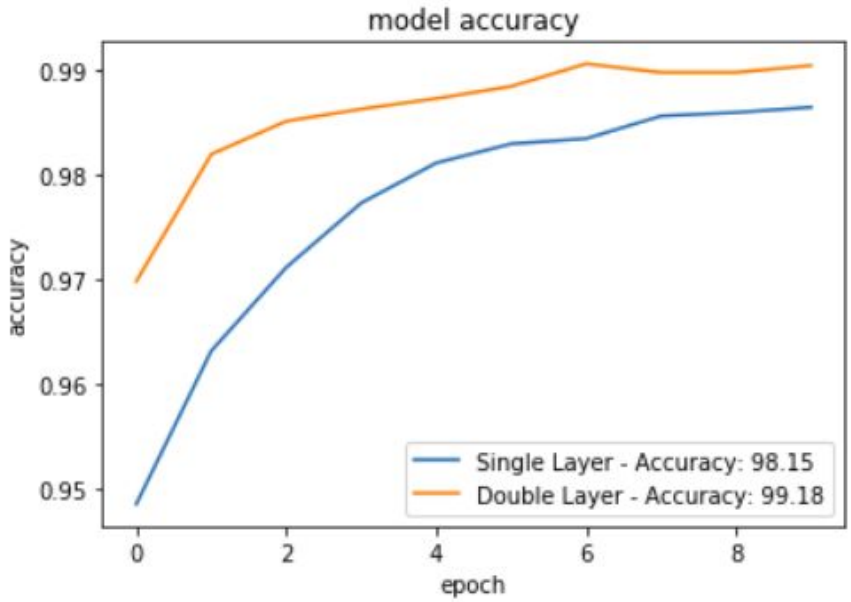



```
loss= 0.07330448315776884
accuracy= 0.9777
```

Variations:

Number of Layers

We trained and evaluated our model on 1 and 2 CNN layers. For the second layer we added a layer with 64 filters and 5x5 kernel size.

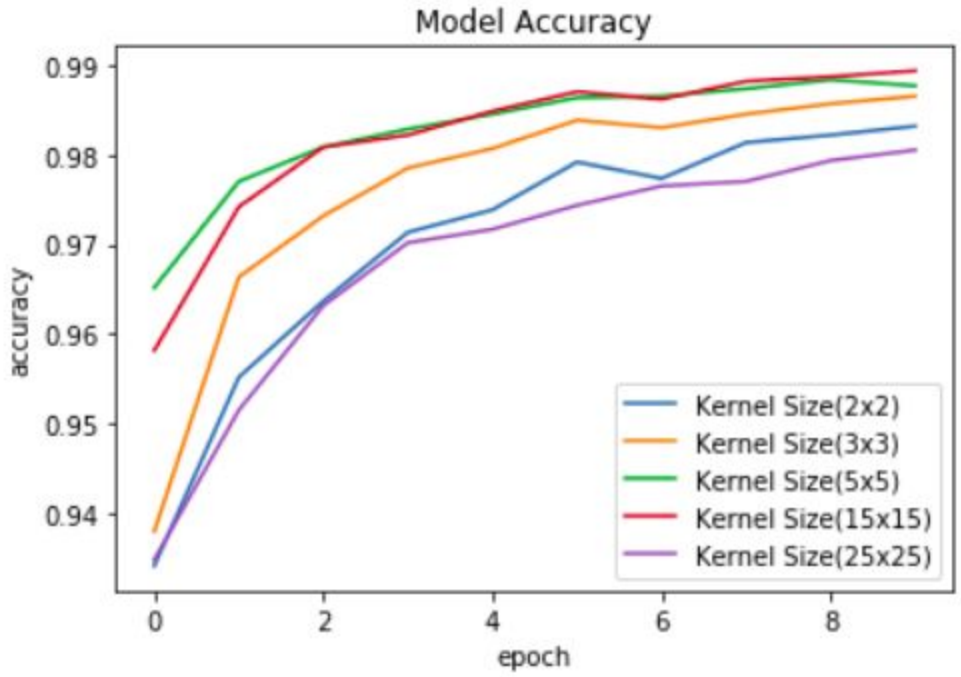


We observed that there was a significant increase in model performance with increase in number of layers . This can be explained as increasing the number of layers means our model can take more complex decisions.

Kernel Size

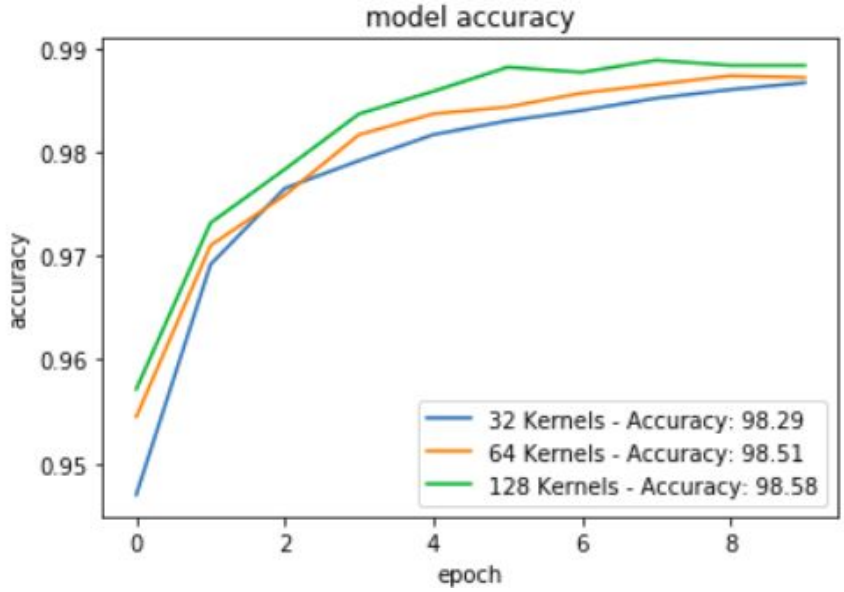
We trained and evaluated our model on kernel size of 2x2, 3x3 , 5x5 and 25X25.

Size of the filters play an important role in finding the key features. A larger size kernel can overlook at the features and could skip the essential details in the images whereas a smaller size kernel could provide more information leading to more confusion. Thus there is a need to determine the most suitable size of the kernel/filter.
From our experiments we found that accuracy to be highest for 5x5 kernel size .



Number of kernels/filters

We trained and evaluated our model on number of kernels equal to 32,64 and 128.

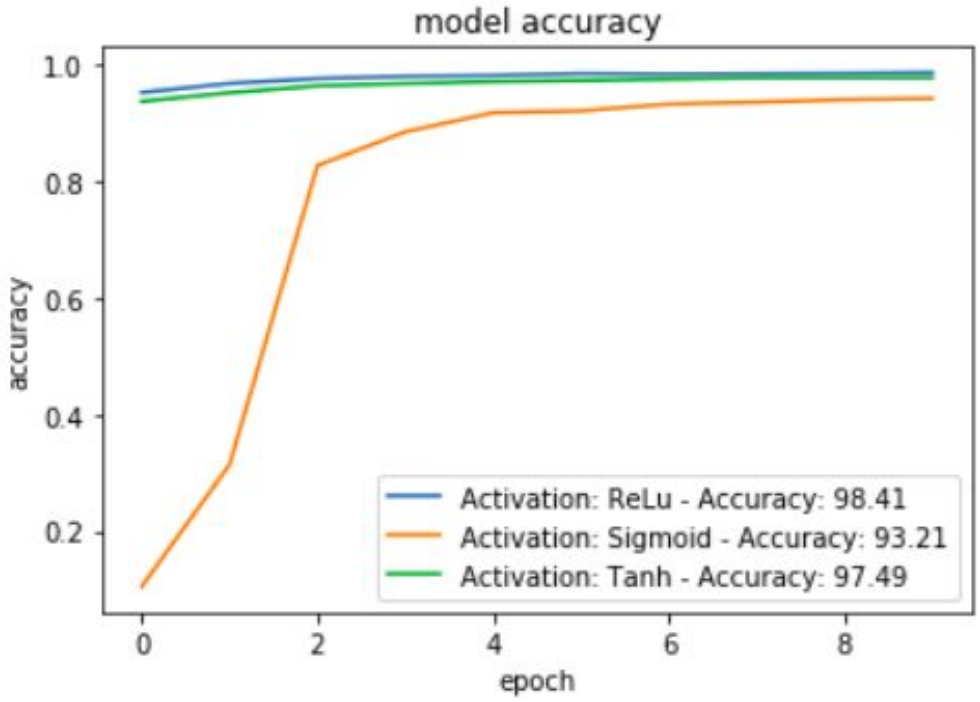


As the complexity of the dataset increases the network requires more number of kernels.
The number of kernels in consecutive layers are expected to be in an increasing sequence as number of possible combinations grow.
From our experiments we observed that the performance of our model increased as the number of kernels increased with the highest accuracy being achieved for number of kernels equal to 128.

Type of Activation Function

We trained and evaluated our model by applying different activation functions on the hidden layer however activation function for the output layer was always taken as softmax.
The activation function taken into consideration were namely ReLu, tanh and sigmoid.

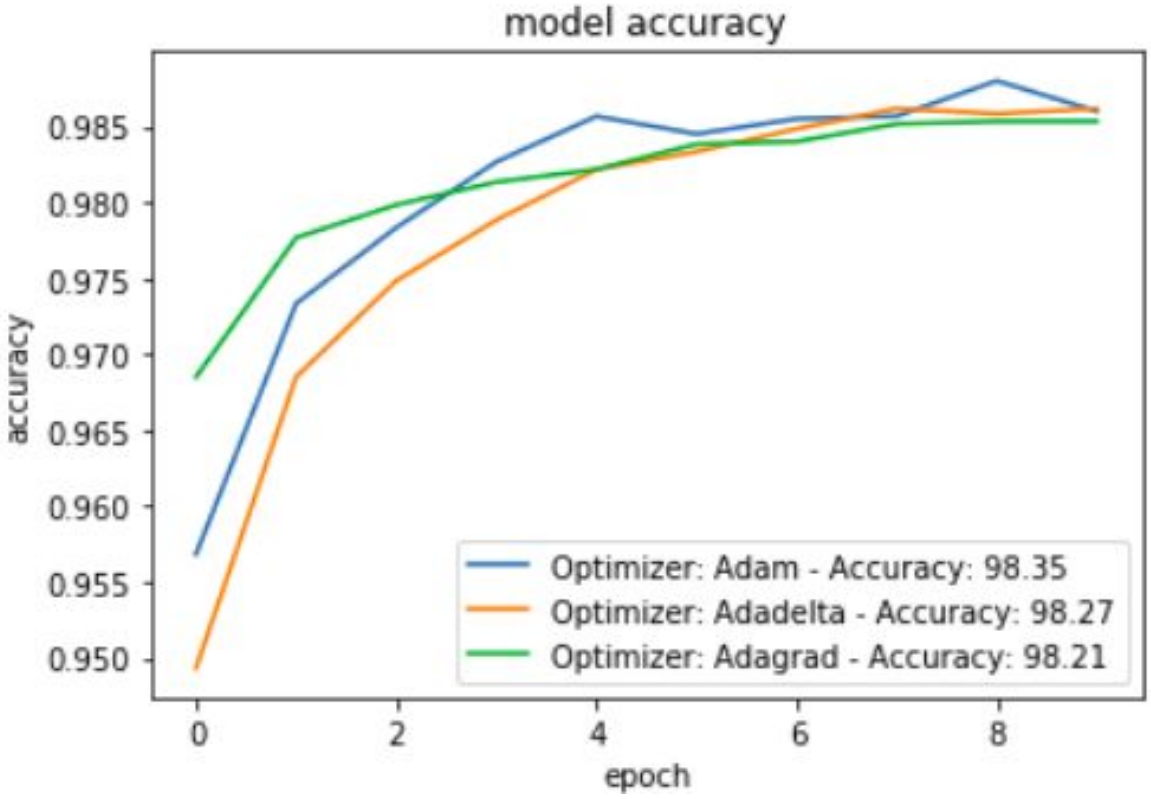
We observed different accuracy results for the different activation function with the highest accuracy being achieved using ReLu as activation function.ReLU is important because it does not saturate; the gradient is always high (equal to 1) if the neuron activates. As long as it is not a dead neuron, successive updates are fairly effective. ReLU is also very quick to evaluate. Compare this to sigmoid or tanh, both of which saturate (if the input is very high or very low, the gradient is very, very small).



Type of optimizer

We trained and evaluated our model by using 3 different optimizers namely adadelta, adagrad and adam.

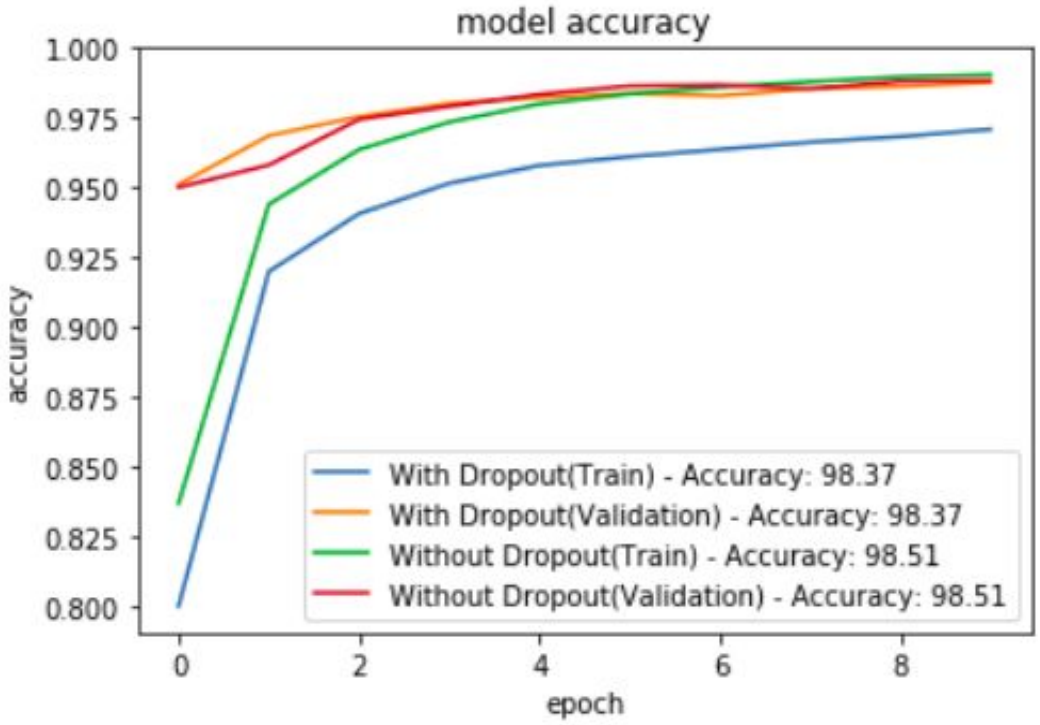
We observe that using different optimization algorithm in our model resulted in different accuracy. The accuracy observed was highest using adam followed by adadelta and finally adagrad.



Dropout

We trained and evaluated our model with and without using dropout technique.

We observed that the accuracy was higher when dropout technique was utilized to overcome overfitting as compared to when it was not used.



CODE FOR LOGISTICS REGRESSION

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
from sklearn.linear_model import LogisticRegression

from mnist import MNIST

mndata = MNIST('sample')

x_train, y_train = mndata.load_training()
x_test, y_test = mndata.load_testing()

def trainModel(model, x_train, y_train, x_test, y_test):
   model.fit(x_train, y_train)
   score = model.score(x_test, y_test.tolist())
   print(score)
   return score

logisticRegr1 = LogisticRegression(solver='lbfgs', max_iter=50)
logisticRegr2 = LogisticRegression(solver='lbfgs')
logisticRegr3 = LogisticRegression(solver='lbfgs', max_iter=300)
logisticRegr7 = LogisticRegression(solver='lbfgs', max_iter=500)

logisticRegr4 = LogisticRegression(solver='lbfgs',
multi_class='multinomial', max_iter=50)
logisticRegr5 = LogisticRegression(solver='lbfgs',
multi_class='multinomial')
logisticRegr6 = LogisticRegression(solver='lbfgs',
multi_class='multinomial', max_iter=300)
logisticRegr8 = LogisticRegression(solver='lbfgs',
multi_class='multinomial', max_iter=500)

score1 = trainModel(logisticRegr1, x_train, y_train, x_test, y_test)
score2 = trainModel(logisticRegr2, x_train, y_train, x_test, y_test)
score3 = trainModel(logisticRegr3, x_train, y_train, x_test, y_test)
score4 = trainModel(logisticRegr4, x_train, y_train, x_test, y_test)
```

```
score5 = trainModel(logisticRegr5, x_train, y_train, x_test, y_test)
score6 = trainModel(logisticRegr6, x_train, y_train, x_test, y_test)
score7 = trainModel(logisticRegr7, x_train, y_train, x_test, y_test)
score8 = trainModel(logisticRegr8, x_train, y_train, x_test, y_test)

# Commented out IPython magic to ensure Python compatibility.
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
# %matplotlib inline
x = [50, 100, 300, 500]
y = [score1, score2, score3, score7]
y1 = [score4, score5, score6, score8]
x = np.array(x)
y = np.array(y)
y1 = np.array(y1)

plt.plot(x, y)
plt.plot(x, y1)
plt.title('')
plt.ylabel('Accuracy')
plt.xlabel('Max Iterations')
plt.legend(['ovr', 'multinomial'], loc='best')
plt.show()
```

CODE FOR MULTI - LAYER PERCEPTRON / DEEP NEURAL NETWORK

```
def createModel(layer_sizes, activation='sigmoid'):
   image_size = 784
   num_classes = 10
   model = Sequential()
   model.add(Dense(layer_sizes[0], activation=activation,
input_shape=(image_size,)))

   for s in layer_sizes[1:]:
       model.add(Dense(units = s, activation = activation))

   model.add(Dense(units=num_classes, activation='softmax'))
   return model

def trainModel(model, batch_size=128, epochs=20, optimizer='adagrad'):
   model.summary()
   model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])
   history = model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_split=.1)
   loss, accuracy  = model.evaluate(x_test, y_test, verbose=False)
   return history, accuracy

def preprocess(y_train, y_test):
   y_train = keras.utils.to_categorical(y_train, num_classes)
   y_test = keras.utils.to_categorical(y_test, num_classes)
   return y_train, y_test

def plotGraph(accuracy, history):
   plt.plot(history.history['acc'])
   plt.plot(history.history['val_acc'])
   plt.title('Model Accuracy' + str(accuracy))
   plt.ylabel('Accuracy')
   plt.xlabel('Epoch')
   plt.legend(['Training', 'Validation'], loc='best')
   plt.show()
```

```
from mnist import MNIST
import numpy as np
mndata = MNIST('sample')

x_train, y_train = mndata.load_training()
x_test, y_test = mndata.load_testing()


x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)



# Flatten the images
image_vector_size = 28*28
print("Training data shape: ", x_train.shape)
print("Test data shape", x_test.shape)

import keras
num_classes = 10
y_train, y_test = preprocess(y_train, y_test)


from keras.layers import Dense
from keras.models import Sequential
from keras.regularizers import l2

image_size = 784
num_classes = 10
model = createModel([512])
history, accuracy = trainModel(model)

import numpy as np
import matplotlib.pyplot as plt

plotGraph(accuracy, history)
```

CODE FOR DEEP CONVOLUTIONAL NEURAL NETWORK

```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Model
from keras.models import Sequential
from keras.layers import Dense, Input
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten
from keras import backend as k

def preprocessing(x_train, x_test, y_train, y_test):
 img_rows, img_cols=28, 28
 if k.image_data_format() == 'channels_first':
     x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
     x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
     input_shape = (1, img_rows, img_cols)

 else:
     x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
     x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
     input_shape = (img_rows, img_cols, 1)

 x_train = x_train.astype('float32')
 x_test = x_test.astype('float32')
 x_train /= 255
 x_test /= 255
 y_train = keras.utils.to_categorical(y_train)
 y_test = keras.utils.to_categorical(y_test)
 return x_train, x_test, y_train, y_test, input_shape

def createModel(input_shape):
 model = Sequential()
 model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
 # model.add(Conv2D(64, (3, 3), activation='relu'))
 model.add(MaxPooling2D(pool_size=(2, 2)))
 model.add(Dropout(0.5))
```

```
 model.add(Flatten())
 model.add(Dense(128, activation='relu'))
 model.add(Dropout(0.5))
 model.add(Dense(10, activation='softmax'))
 model.compile(optimizer=keras.optimizers.Adadelta(),
               loss=keras.losses.categorical_crossentropy,
               metrics=['accuracy'])
 return model

def trainModel(x_train, y_train, x_test, y_test, model):
 history = model.fit(x_train, y_train, epochs=10, batch_size=500,
validation_split=.1)
 score = model.evaluate(x_test, y_test, verbose=0)
 return history, score

def plotGraph(score, history):
 plt.plot(history.history['acc'])
 plt.plot(history.history['val_acc'])
 plt.title('Model Accuracy - ' + str(score[1]))
 plt.ylabel('accuracy')
 plt.xlabel('epoch')
 plt.legend(['Training', 'Validation'], loc='best')
 plt.show()
 print('loss=', score[0])
 print('accuracy=', score[1])

from mnist import MNIST
mndata = MNIST('sample')

x_train, y_train = mndata.load_training()
x_test, y_test = mndata.load_testing()
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)

x_train, x_test, y_train, y_test, input_shape = preprocessing(x_train,
x_test, y_train, y_test)
```

```
model = createModel(input_shape)
history, score = trainModel(x_train, y_train, x_test, y_test, model)

import matplotlib as mpl
import matplotlib.pyplot as plt
# %matplotlib inline
plotGraph(score, history)
```