CS574
Computer vision and machine learning

(Assignment #1)

Supervised By - Arijit Sur

Created by:-
(Group 07)
Aditya(160101010)
Avinash(160101018)
Phool Chandra(160101051)
Savinay(160101062)
Kevin(160101063)

Problem Statement

Consider 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

Further,

1. 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 the over-fitting problem with possible solutions.
2. Write a report stating the detailed summary about this assignment with proper snapshots of training processes, testing processes and change in learning mechanism as you change the parameters. The report should contain appropriate justifications for each dynamic you find during the conduction of this assignment.

Language / libraries used:

- Python 3 used as the Programming language .
- Libraries
  - Sklearn was used for logistic regression.
  - Numpy used for data arrays
  - We used Keras API for building all models.
  - Matplotlib was used for visualisation of results using graphs.

Studying the dataset

The MNIST data comes in two parts. The first part contains 60,000 images of numerical digits to be used as training data. The images are greyscale and 28 by 28 pixels in size. The second part of the MNIST data set is 10,000 images to be used as test data. Again, these are 28 by 28 greyscale images. We are going to use test data set to check the accuracy of our trained model . We regard each training input as a 28×28=784-dimensional vector. A label is attached with every data set with its correct class (0-9). We are going to use data set to train our model for each method given in problem set and analyse each model by changing parameter while training them.

Logistic Regression

We first extracted data from .idx files downloaded from given link in the problem set. The data was converted into numpy array for further use.

Test and Train Image array (Containing Image Intensity Values) was reshaped into numpy arrays of dimension (60000,784) and (10000, 784) respectively.

Test and Train Label array was stored in scalar numpy array.

Then we provided training data set containing image and labels to logistic regression class in sklearn library for training.

Once the model is trained we provide test image data to predict function of logistic regression class to test the model. For the same we match the predicted label with the actual label of the test data set and count the accurate ones to get the accuracy score.

In our case accuracy came was 0.91 when c=1(default) was used.

Observations

Accuracy in Logistic Regression is less as compared to other training methods because it regards all the points and then finds a hyperplane which separates data, not the best hyperplane. Logistic Regression is good for low number of features.

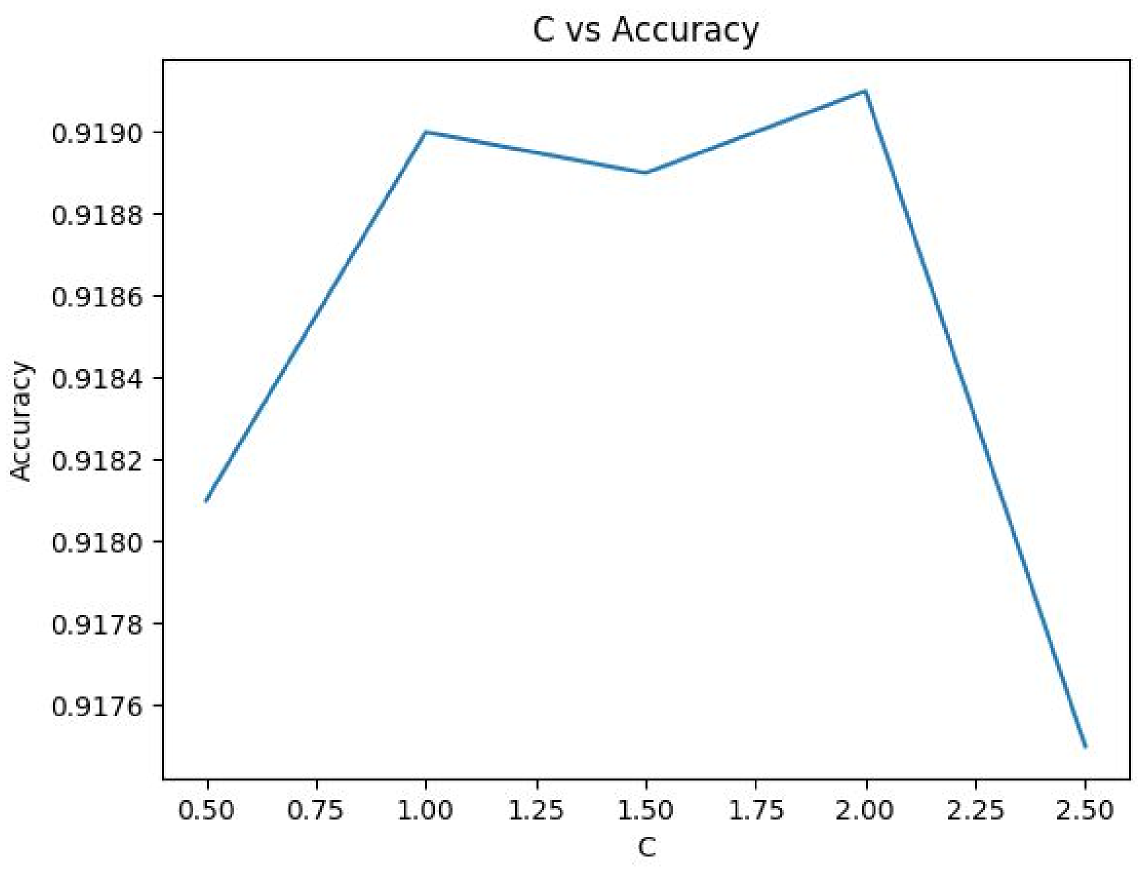
Direct Multi-class Logistic Regression

Observation:

Multi-Class Logistic Regression performs better than one-vs-all Logistic Regression because, as compared to one class vs all classes in simple Logistic Regression in multi class a single data field is simultaneously calculated for all the classes. This results in reduction of run time for multi class Logistic Regression.

We tried to observe variation in accuracy score while changing value of ‘C’ for training.

Observations



● Linear-Kernel is more accurate for informative data which was not the case with the dataset provided which had digital images, as pixels are not very informative, so non-linear kernel performs better.

● We can see from the plot for gamma default and varying values of C that accuracy is higher for higher values of C. As C determines the penalty for error term for every training example. Thus, the weight of each error term is low when C is low, so even larger error values are accepted in training phase.

● But when C has a higher value , the weight of each error term is increased, so lower error values are accepted. So, a smaller margin hyperplane is built, but number of points misclassified are less, so the accuracy increases.

● We observe that for higher values of C there is risk of overfitting as train accuracy keeps increasing with increasing value of C while the test and validation accuracy stop increasing significantly after C=40.

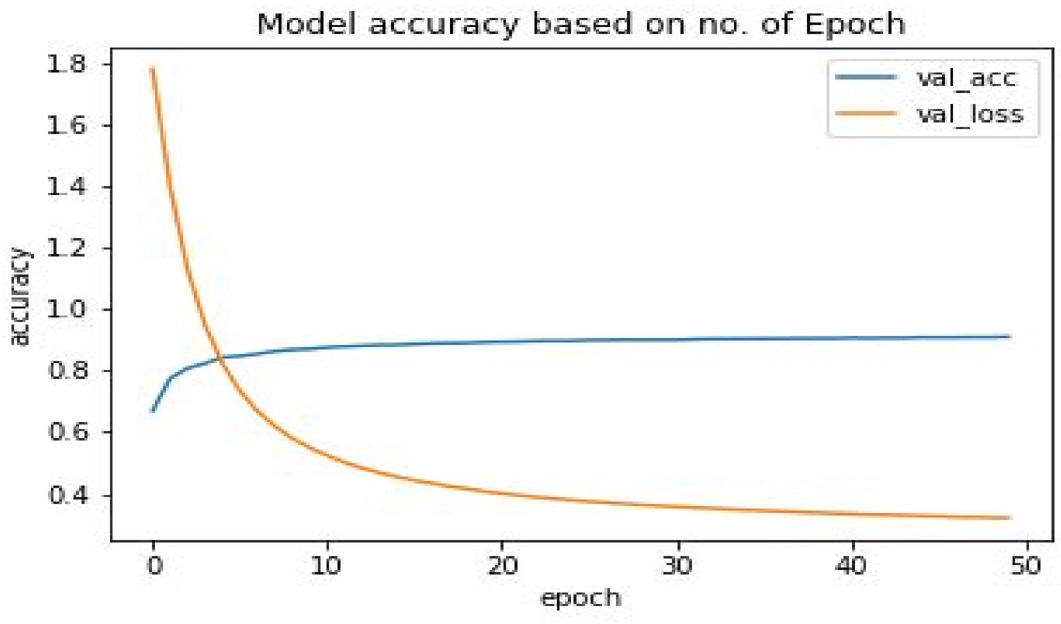
Multi Layer Perceptron

A **multilayer perceptron** (MLP) is a deep, artificial neural network. ... They are composed of an input **layer** to receive the signal, an output **layer** that makes a decision or prediction about the input, and in between those two, an arbitrary number of hidden **layers** that are the true computational engine of the MLP.

In our case number of hidden layers We train and evaluate our model by using 1 hidden layers and plot out the results with 784 number of neurons respectively.

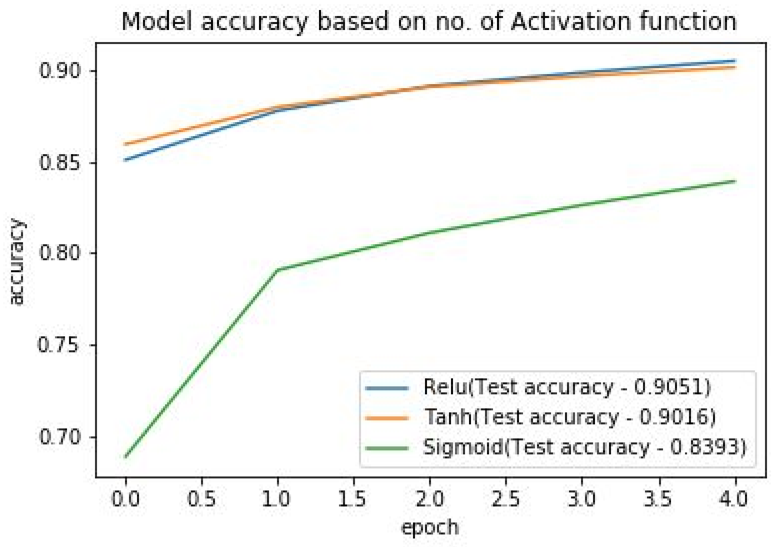
Number of epochs

An epoch basically consists of one forward pass and one backward pass of all the training examples. No. of epochs has a significant impact on the performance of our network.With a small number of epochs, you may not have trained sufficiently. With a larger number of epochs, it looks like validation accuracy typically starts to go down after some number of epochs. Hence, it is very important to figure out the correct amount of epochs for a given model to have good accuracy.



Type of activation function

Since we prefer the output to be probabilistically distributed in 10 classes, we let the last layer, i.e, the output layer to have softmax as its activation function. However, we study 3 different activation functions on our hidden layers (2 hidden layers are used in this example) - Sigmoid(Logistic Activation Function), Tanh(hyperbolic tangent Activation Function) and ReLU(Rectified Linear Unit). As expected, the model using the sigmoid activation function in its hidden layers learns very slowly as compared to the other models. This is because the partial derivatives of the sigmoid function tends to be zero at extremes, resulting in learning slowdown.

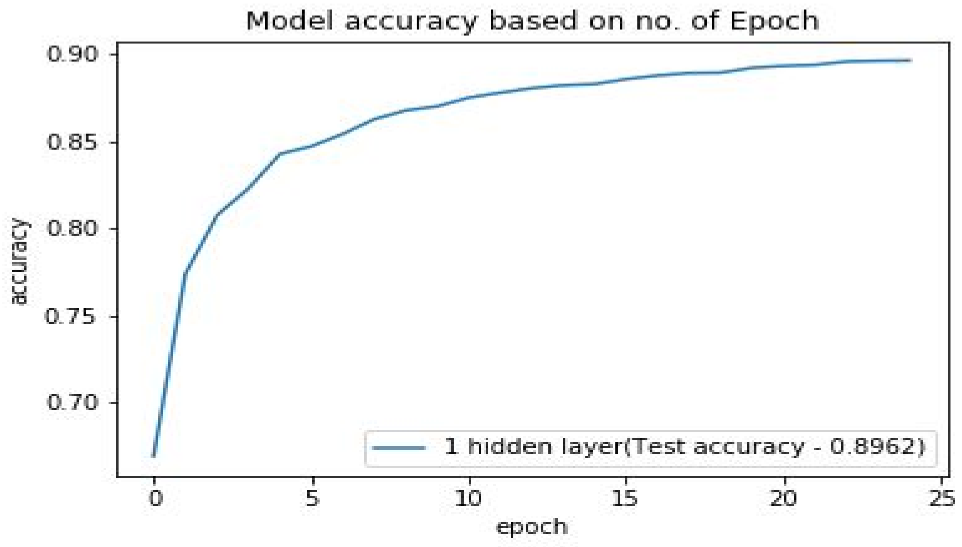


Deep Neural Network

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

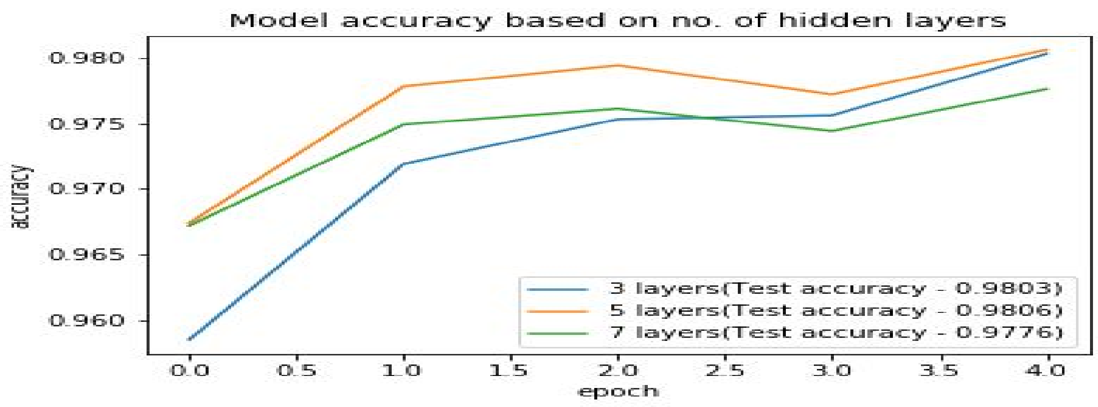
- A neural network with a single hidden layer consisting of 100 neurons and 784 neurons in the input layer and then finally 10 neurons in the output layer.

- We use softmax as the activation function. The fact that a softmax layer outputs a probability distribution is used here to get the probabilities of an image to be in one of the 10 categories.
- Batch size of 200 is used and the model is run for 25 epochs.



Number of hidden layers

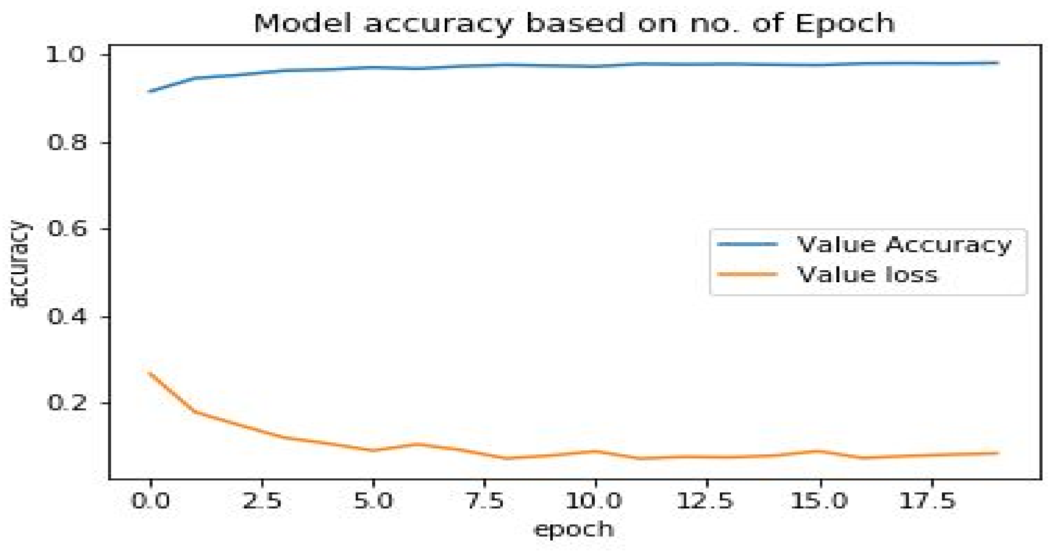
We train and evaluate our model by using 3, 5 and 7 hidden layers and plot out the results. The optimal size of the hidden layer is usually between the size of the input and size of the output layers. Every subsequent layer takes decisions by weighing up decisions given by its previous layer. We see that on increasing the number of hidden layers, the accuracy of the model improves. It is as on increasing the number of layers, our model can take more complex decisions.



Number of epochs

The number of epochs is a hyperparameter that defines the number of 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. Less number of epoch results in underfitting while large epochs result in overfitting of data. So for a given model to have a good accuracy, it need appropriate amount of epochs .So set the number of epochs as high as possible training as it help to decrease error rates.



Type of activation function

Sigmoid / Logistic

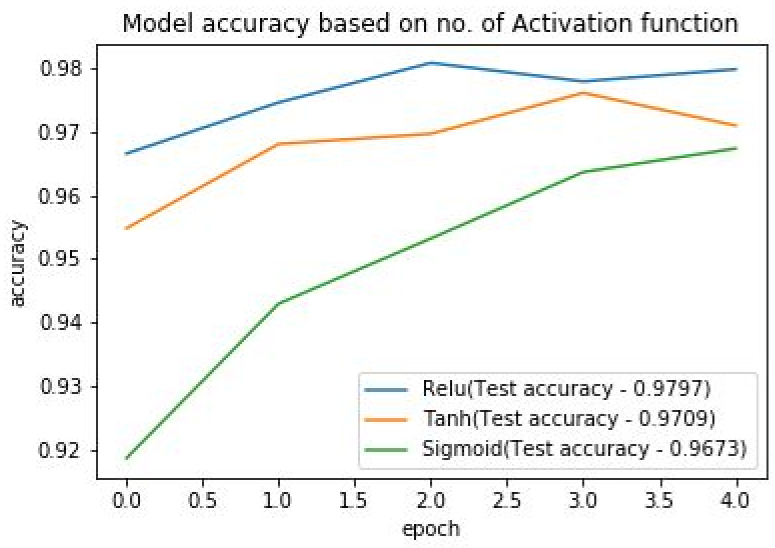
- Smooth gradient, preventing “jumps” in output values.
- Output values bound between 0 and 1, normalizing the output of each neuron.
- Clear predictions—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.

TanH / Hyperbolic Tangent

- Zero centered—making it easier to model inputs that have strongly negative, neutral, and strongly positive values.

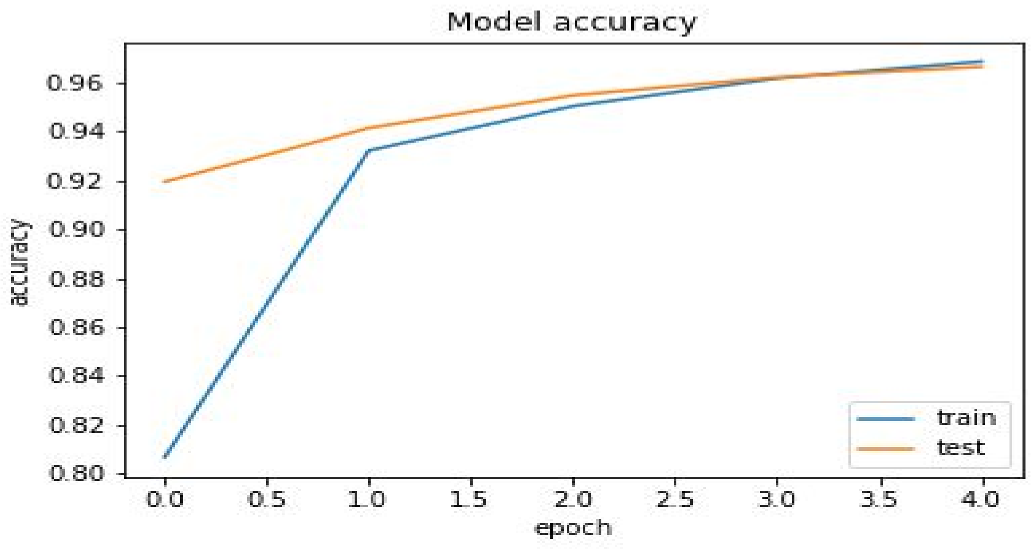
ReLU (Rectified Linear Unit)

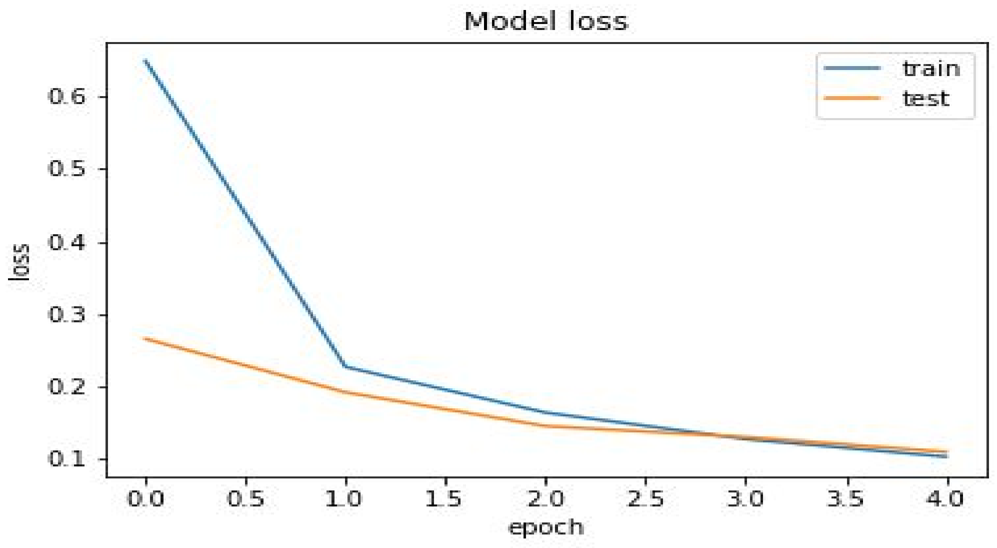
- Computationally efficient—allows the network to converge very quickly
- Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation



Overfitting problem

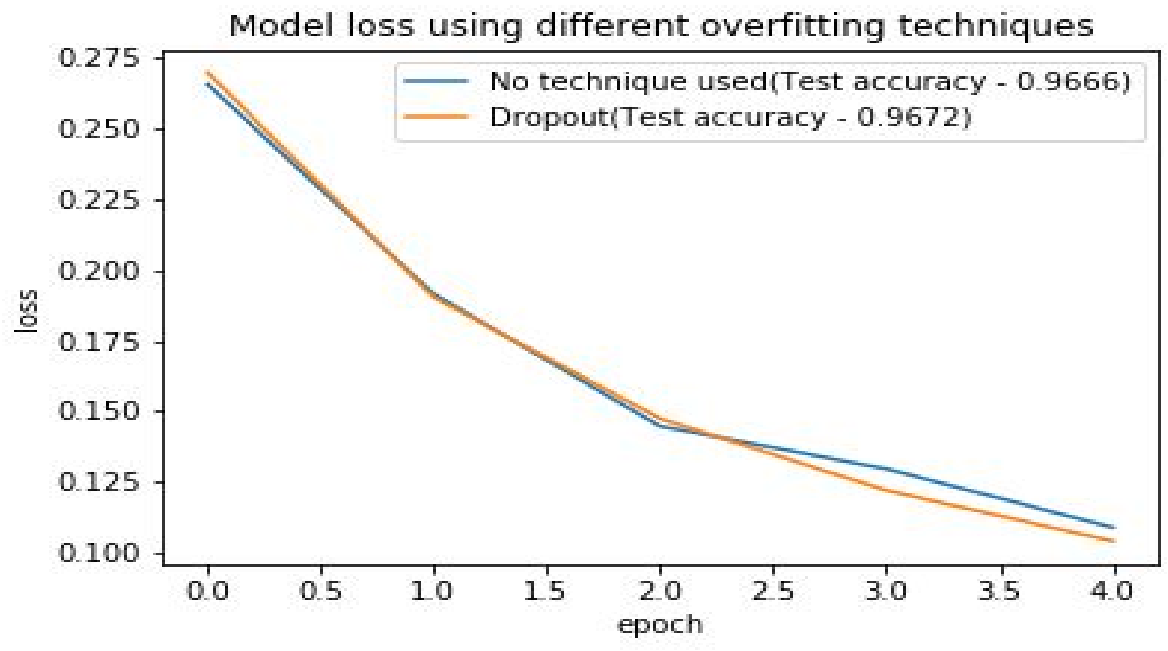
Overfitting refers to a model that models the training data too well. It happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.it occurs when our model does not generalize well to real-world cases although it fits the training data well. It happens when increased or it is trained for high epochs than required. After a point of time, test accuracy ceases to increase.





We study a method to remove the problem of overfitting of data -

● **Dropout** - Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped-out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.



Deep Convolutional Neural Network

Our previous model has the following properties -

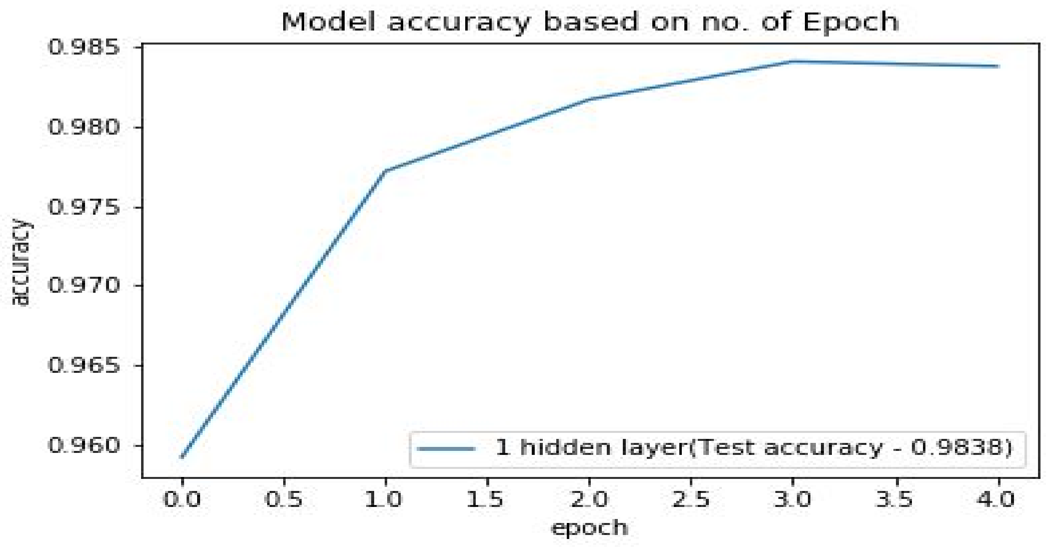
● Logistic Regression - The model was trained using one vs rest scheme with varying value of C

● Neural Network - It was modelled with a single hidden convolutional layer having kernel size of 5*5 and number of the kernel were 32.

● ReLu activation function was used for the hidden layer and We use softmax as the activation function of the output layer for the same reason as in the previous model.

● We also add a 2D max-pooling layer of 2x2 over our convolutional layer.

● Batch size of 200 is used and the model is run for 5 epochs.



Accuracy in our case was 0.98 which is good. Now we observe accuracy while changing different parameters to get more accuracy.

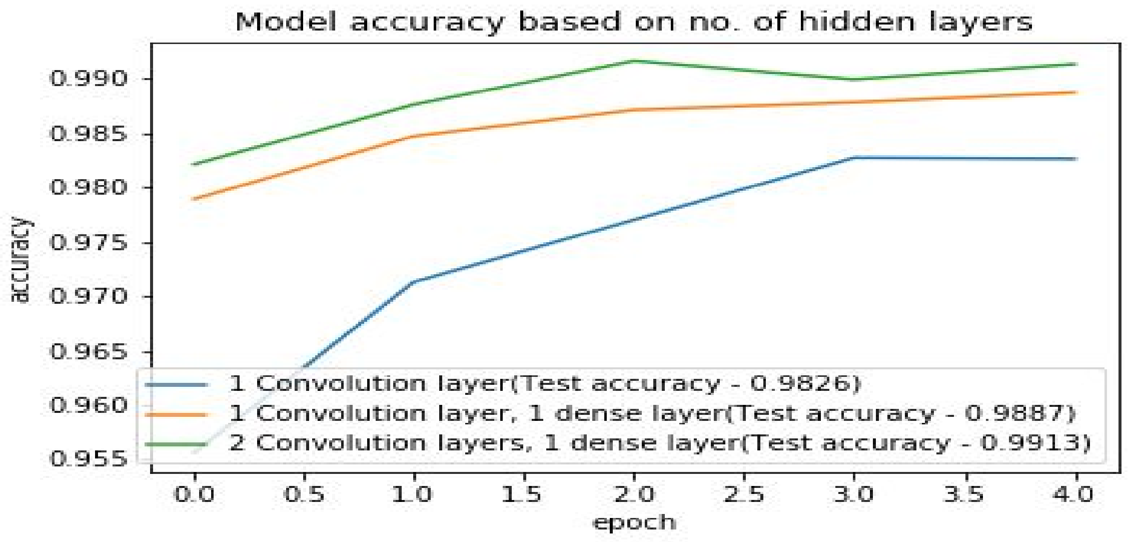
Number of layers

We try with three different combination of convolutional and dense layers -

● First we tried Single convolutional layer

● Secondly we Single convolutional layer (32 kernels of size 3x3) with a dense layer having 128 neurons.

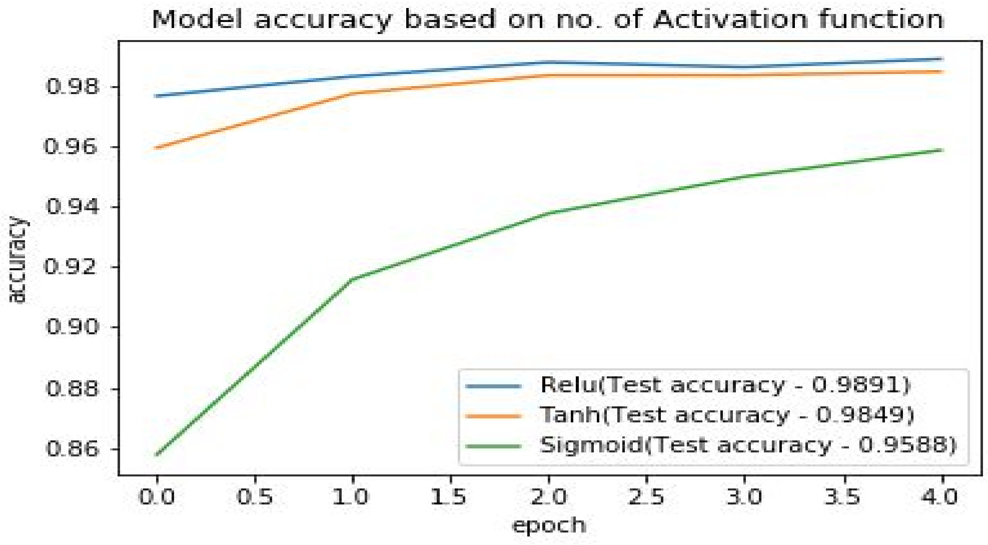
● Third we have 2 convolutional layers (second one with 64 kernels of size 3x3) and a dense layer



As can be observed that the third model is better as number of convolution layers have increased.

Type of activation function

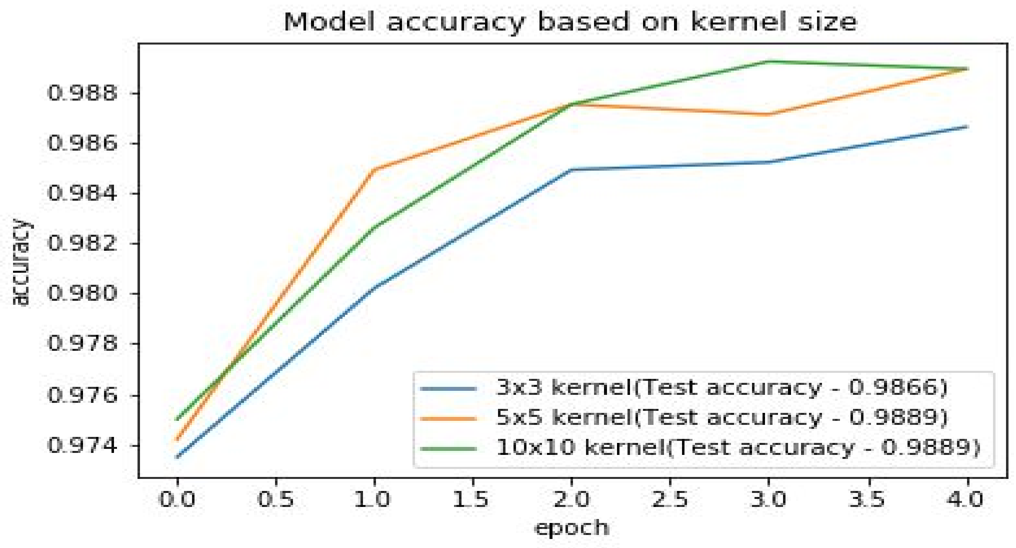
Here we observe different activation functions in hidden layers where softmax is at output layer.



As we can observed Relu is having good accuracy where as Sigmoid is slow learner and its accuracy is also worst.

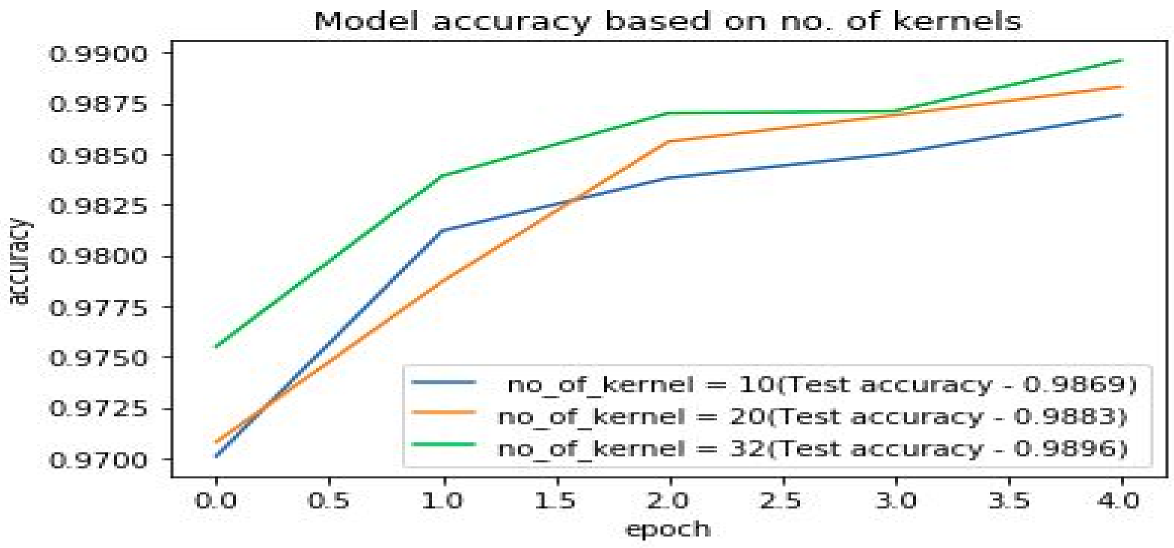
Kernel size

We observe that accuracy for 5x5 kernel is better. Size of the filters plays an important role in finding the key features. A larger size kernel can overlook 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. For the given data set of images (28x28) in size, 5x5 seems to be the most suitable size in effectively determining the details.



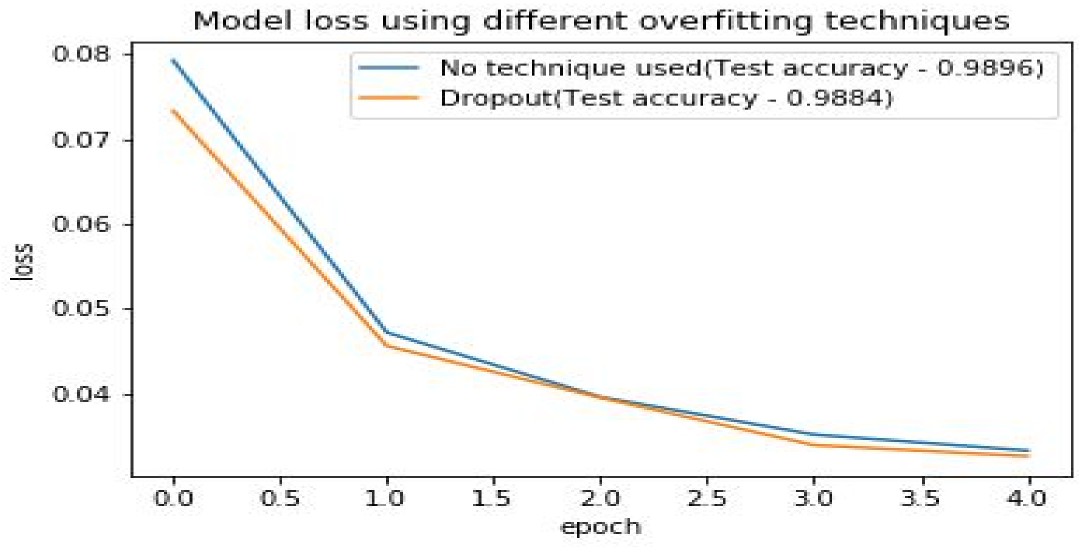
Number of kernels

We experiment with 10, 20 and 32 feature maps / kernels in our model. The more complex the dataset is, it is expected that networks with more kernels perform better. Intuitively, a number of the kernel at layer expected to bigger in the previous layers, as number of possible combination grow. That is why, in general, first layer kernels are less than mid-high-level ones. As can be seen from the plot, 5 kernels are way too less for our data and the accuracy increases on using higher number of kernels as expected.



Overfitting

Overfitting problem might arise in CNNs too, because of more or less the same reasons. We again try the 2 techniques to reduce overfitting - Dropout and L2 regularisation. Note that we apply dropout on only the dense layer of the network and not convolutional layers. This is because the shared weights in convolutions mean that convolutional filters are forced to learn from across the entire image. This makes them less likely to pick up on local idiosyncrasies in the training data.



Using dropout reduces the loss in test data which is clearly visible in the above plot. The higher loss means a higher level of overfitting.

Conclusion

- As the training data is fed multiple times to the neural net, the model becomes more accurate as can be seen from the fact that accuracy improves with each epoch. Thus, there is a trade-off between time and accuracy.

- Convolutional Neural Network performs better as they take into consideration the features in the image and make use of common patterns across it.
- Tuning of hyperparameter should be done on validation data.
- Since a lot of techniques and models are heuristic in nature, models and parameters are very much dependent on the type of problem.

Code

1) Logistic Regression

```
import time
stime = time.time()

import struct as st
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
import array


train_file = {'train_images' : 'train-images.idx3-ubyte'
,'train_labels' : 'train-labels.idx1-ubyte', 'test_images' :
'test-images.idx3-ubyte' ,'test_labels' : 'test-labels.idx1-ubyte'}

train_labels_array = np.array([])

data_types = {
      0x08: ('ubyte', 'B', 1),
      0x09: ('byte', 'b', 1),
      0x0B: ('>i2', 'h', 2),
      0x0C: ('>i4', 'i', 4),
      0x0D: ('>f4', 'f', 4),
     0x0E: ('>f8', 'd', 8)}

for name in train_file.keys():
    if name == 'train_images':
      train_imagesfile = open(train_file[name],'r+')
    if name == 'train_labels':
      train_labelsfile = open(train_file[name],'r+')
    if name == 'test_images':
      test_imagesfile=open(train_file[name],'r+')
    if name == 'test_labels':
      test_labelsfile = open(train_file[name],'r+')


train_imagesfile.seek(0)
magic = st.unpack('>4B',train_imagesfile.read(4))
if(magic[0] and magic[1]) or (magic[2] not in data_types):
    raise ValueError("File Format not correct")

test_imagesfile.seek(0)
magic2 = st.unpack('>4B',test_imagesfile.read(4))
if(magic2[0] and magic2[1]) or (magic2[2] not in data_types):
     raise ValueError("File Format not correct")

nDim = magic[3]
print ("Data is ",nDim,"-D")
nDim2 = magic2[3]
print ("Data is ",nDim2,"-D")

#offset = 0004 for number of images
#offset = 0008 for number of rows
#offset = 0012 for number of columns
#32-bit integer (32 bits = 4 bytes)
train_imagesfile.seek(4)
nImg = st.unpack('>I',train_imagesfile.read(4))[0] #num of
images/labels
nR = st.unpack('>I',train_imagesfile.read(4))[0] #num of rows
nC = st.unpack('>I',train_imagesfile.read(4))[0] #num of columns
nBytes = nImg*nR*nC
```

```
train_labelsfile.seek(8) #Since no. of items = no. of images and is
already read
print ("no. of trainimages :: ",nImg)
print ("no. of trainrows :: ",nR)
print ("no. of traincolumns :: ",nC)

test_imagesfile.seek(4)
nImg2 = st.unpack('>I',test_imagesfile.read(4))[0] #num of
images/labels
nR2 = st.unpack('>I',test_imagesfile.read(4))[0] #num of rows
nC2 = st.unpack('>I',test_imagesfile.read(4))[0] #num of columns
nBytes2 = nImg2*nR2*nC2
test_labelsfile.seek(8) #Since no. of items = no. of images and is
already read
print ("no. of testimages :: ",nImg2)
print ("no. of testrows :: ",nR2)
print ("no. of testcolumns :: ",nC2)

#Read all data bytes at once and then reshape
train_images_array = 255 -
np.asarray(st.unpack('>'+'B'*nBytes,train_imagesfile.read(nBytes))).r
eshape((nImg,nR*nC))
train_labels_array =
np.asarray(st.unpack('>'+'B'*nImg,train_labelsfile.read(nImg)))

test_images_array = 255 -
np.asarray(st.unpack('>'+'B'*nBytes2,test_imagesfile.read(nBytes2))).
reshape((nImg2,nR2*nC2))
test_labels_array =
np.asarray(st.unpack('>'+'B'*nImg2,test_labelsfile.read(nImg2)))



print (train_labels_array)
print (train_labels_array.shape)
print (train_images_array.shape)

print (test_labels_array)
print (test_labels_array.shape)
print (test_images_array.shape)



x = list()
y=list()


v=0
for i in range(1,100,10):
    x.append(i)
    logisticRegr = LogisticRegression(C=i)
    logisticRegr.fit(train_images_array, train_labels_array)
    score = logisticRegr.score(test_images_array,test_labels_array)
    y.append(score)



plt.ylabel('Accuracy')
plt.xlabel('C')
plt.plot(x,y)
plt.title('C vs Accuracy')
plt.savefig('CvsAccuracy_0to100.png')
plt.close()

for i in y:
    print(i)





print ("Time of execution : %s seconds" % str(time.time()-stime))
```

2) Multi-Layer Perceptron

```
# import libraries
import os
from os.path import dirname, realpath

import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.optimizers import RMSprop
from keras.utils import np_utils
import matplotlib.pyplot as plt
```

```
PLOTS_DIR = os.path.join("PROJECT_DIR_Multi_Layer_Percepton",
"plots")
if not os.path.exists(PLOTS_DIR):
    os.makedirs(PLOTS_DIR)
```

```
def Get_Data():
    # fix random seed for reproducibility
    seed = 7
    numpy.random.seed(seed)

    # load data
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    # flatten 28*28 images to a 784 vector for each image
    num_pixels = X_train.shape[1] * X_train.shape[2]

    X_train = X_train.reshape(X_train.shape[0],
num_pixels).astype('float32')
    X_test = X_test.reshape(X_test.shape[0],
num_pixels).astype('float32')
```

```
    # normalize inputs from 0-255 to 0-1
    X_train = X_train / 255
    X_test = X_test / 255

    # one hot encode outputs
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
    return X_train, y_train, X_test, y_test
```

```
def Build_baseline_Model(activation_fn = "sigmoid",optimizer_name =
"sgd",layers = 2,add_Dropout_layer=False, Dropout_value = 0.2):
    num_classes = 10
    num_pixels = 784

    # create model
    model = Sequential()
    model.add(Dense(784, input_dim=num_pixels,
activation=activation_fn))

    model.add(Dense(num_classes, kernel_initializer='normal',
activation='softmax'))

    model.summary()

    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer =
optimizer_name, metrics=['accuracy'])

    return model
```

```
def change_Epoch(X_train, y_train, X_test, y_test, batch_size = 200,
no_of_epochs = 10):
    plot_type = 'epochs'
    model1 = Build_baseline_Model()
    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=50, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history1.history['val_loss'])

    plt.title('Model accuracy based on no. of Epoch')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['val_acc', 'val_loss'], loc='upper right')
    plt.savefig(PLOTS_DIR + '/acc_simple_{}.png'.format(plot_type))
    plt.show()
    plt.close()
```

```
def change_Activation_fn(X_train, y_train, X_test, y_test, batch_size
= 200, no_of_epochs = 10):
    plot_type='activation'
    model1 = Build_baseline_Model(activation_fn= "relu")
    model2 = Build_baseline_Model(activation_fn= "tanh")
    model3 = Build_baseline_Model(activation_fn= "sigmoid")

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on no. of Activation function')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = 'Relu(Test accuracy - {})'.format(score1[1])
    legend2 = 'Tanh(Test accuracy - {})'.format(score2[1])
    legend3 = 'Sigmoid(Test accuracy - {})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/acc_simple_{}.png'.format(plot_type))
    plt.show()
    plt.close()
```

```
if __name__ == '__main__':
    X_train, y_train, X_test, y_test = Get_Data()
    change_Activation_fn(X_train = X_train, y_train = y_train, X_test
= X_test, y_test = y_test, no_of_epochs = 5)
    change_Epoch(X_train = X_train, y_train = y_train, X_test =
X_test, y_test = y_test, no_of_epochs = 5)
    pass
```

3) Deep Neural Network

```
# import libraries
import os
from os.path import dirname, realpath

import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Activation
from keras.optimizers import RMSprop
```

```python
from keras.utils import np_utils
import matplotlib.pyplot as plt
```

```python
PLOTS_DIR = os.path.join("DNN", "plots")
if not os.path.exists(PLOTS_DIR):
    os.makedirs(PLOTS_DIR)

def Get_Data():
    # fix random seed for reproducibility
    seed = 7
    numpy.random.seed(seed)

    # load data
    (X_train, y_train), (X_test, y_test) = mnist.load_data()

    # flatten 28*28 images to a 784 vector for each image
    num_pixels = X_train.shape[1] * X_train.shape[2]

    X_train = X_train.reshape(X_train.shape[0],
num_pixels).astype('float32')
    X_test = X_test.reshape(X_test.shape[0],
num_pixels).astype('float32')

    # normalize inputs from 0-255 to 0-1
    X_train = X_train / 255
    X_test = X_test / 255

    # one hot encode outputs
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]
    return X_train, y_train, X_test, y_test


def Build_baseline_Model(activation_fn = "sigmoid",optimizer_name =
"adam",layers = 3,add_Dropout_layer=False, Dropout_value = 0.2):
    num_classes = 10
    num_pixels = 784
    # create model
    model = Sequential()
    model.add(Dense(784, input_dim=num_pixels,
activation=activation_fn))

    if add_Dropout_layer == True:
        model.add(Dropout(Dropout_value))

    if layers >= 2:
        model.add(Dense(512))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    if layers >= 3:
        model.add(Dense(256))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    if layers >= 4:
        model.add(Dense(200))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    if layers >= 5:
        model.add(Dense(128))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    if layers >= 6:
        model.add(Dense(80))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))
    if layers >= 7:
        model.add(Dense(56))
        model.add(Activation(activation_fn))
        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    model.add(Dense(num_classes, kernel_initializer='normal',
activation='softmax'))

    model.summary()

    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer =
optimizer_name, metrics=['accuracy'])

    return model


def change_layers(X_train, y_train, X_test, y_test, batch_size = 200,
no_of_epochs = 10):
    plot_type='layers'
    model1 = Build_baseline_Model(layers = 3)
    model2 = Build_baseline_Model(layers = 5)
    model3 = Build_baseline_Model(layers = 7)

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on no. of hidden layers')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = '3 layers(Test accuracy - {})'.format(score1[1])
    legend2 = '5 layers(Test accuracy - {})'.format(score2[1])
    legend3 = '7 layers(Test accuracy - {})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/acc_dnn_{}.png'.format(plot_type))
    plt.show()
    plt.close()


def change_Activation_fn(X_train, y_train, X_test, y_test, batch_size
= 200, no_of_epochs = 10):
    plot_type='activation'
    model1 = Build_baseline_Model(activation_fn= "relu")
    model2 = Build_baseline_Model(activation_fn= "tanh")
    model3 = Build_baseline_Model(activation_fn= "sigmoid")

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on no. of Activation function')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = 'Relu(Test accuracy - {})'.format(score1[1])
    legend2 = 'Tanh(Test accuracy - {})'.format(score2[1])
    legend3 = 'Sigmoid(Test accuracy - {})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/acc_dnn_{}.png'.format(plot_type))
    plt.show()
    plt.close()


def change_Epoch(X_train, y_train, X_test, y_test, batch_size = 200,
no_of_epochs = 10):
    plot_type = 'epochs'
    model1 = Build_baseline_Model()

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=20, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history1.history['val_loss'])

    plt.title('Model accuracy based on no. of Epoch')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = 'Value Accuracy'
    legend2 = 'Value loss'
    plt.legend([legend1, legend2], loc='center right')
    plt.savefig(PLOTS_DIR + '/acc_dnn_{}.png'.format(plot_type))
    plt.show()
    plt.close()


def change_overfitting(X_train, y_train, X_test, y_test, batch_size =
200, no_of_epochs = 10):
    plot_type='overfitting'
    model1 = Build_baseline_Model()
    model2 = Build_baseline_Model(add_Dropout_layer = True)
    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['acc'])
    plt.plot(history1.history['val_acc'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='lower right')
    #plt.show()
    plt.savefig(PLOTS_DIR +
'/acc_dnn_test_train.png'.format(plot_type))
    plt.close()

    plt.plot(history1.history['loss'])
    plt.plot(history1.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper right')
    #plt.show()
    plt.savefig(PLOTS_DIR +
'/loss_dnn_test_train.png'.format(plot_type))
    plt.close()

    plt.plot(history1.history['val_loss'])
    plt.plot(history2.history['val_loss'])
    plt.title('Model loss using different overfitting techniques')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    legend1 = 'No technique used(Test accuracy -
{})'.format(score1[1])
```

```
    legend2 = 'Dropout(Test accuracy - {})'.format(score2[1])
    plt.legend([legend1, legend2], loc='upper right')
    #plt.show()
    plt.savefig(PLOTS_DIR + '/loss_dnn_{}.png'.format(plot_type))
    plt.close()


if __name__ == '__main__':
    X_train, y_train, X_test, y_test = Get_Data()
    change_layers(X_train = X_train, y_train = y_train, X_test = 
X_test, y_test = y_test, no_of_epochs = 5)
    change_Activation_fn(X_train = X_train, y_train = y_train, X_test 
= X_test, y_test = y_test, no_of_epochs = 5)
    change_Epoch(X_train = X_train, y_train = y_train, X_test = 
X_test, y_test = y_test, no_of_epochs = 5)
    change_overfitting(X_train = X_train, y_train = y_train, X_test = 
X_test, y_test = y_test, no_of_epochs = 5)
    pass
```

4) Deep Convolutional Neural Network

```
# import libraries
import os
from os.path import dirname, realpath
import numpy
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
import matplotlib.pyplot as plt
from keras import backend as K
K.set_image_dim_ordering('th')

PLOTS_DIR = os.path.join("CNN", "plots")
if not os.path.exists(PLOTS_DIR):
    os.makedirs(PLOTS_DIR)

def Get_Data():
    # fix random seed for reproducibility
    seed = 7
    numpy.random.seed(seed)

    # load data
    (x_train, y_train), (x_test, y_test) = mnist.load_data()

    # reshape to be [samples][pixels][width][height]
    x_train = x_train.reshape(x_train.shape[0], 1, 28,
28).astype('float32')
    x_test = x_test.reshape(x_test.shape[0], 1, 28,
28).astype('float32')

    # normalize inputs from 0-255 to 0-1
    x_train = x_train / 255
    x_test = x_test / 255

    # one hot encode outputs
    y_train = np_utils.to_categorical(y_train)
    y_test = np_utils.to_categorical(y_test)
    num_classes = y_test.shape[1]

    return x_train, y_train, x_test, y_test

def Build_baseline_Model(activation_fn = "relu",optimizer_name =
"adam", layers=1,add_Dropout_layer=True, Dropout_value = 0.2,
                        kernel_size=(5, 5), no_of_kernels=32,
add_dense_layer=True,dense_layer = 1):
    num_classes = 10

    model = Sequential()
    model.add(Conv2D(no_of_kernels, kernel_size = kernel_size,
input_shape=(1, 28, 28), activation=activation_fn))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    if layers == 2:
        model.add(Conv2D(64, kernel_size = kernel_size,
input_shape=(1, 28, 28), activation=activation_fn))
        model.add(MaxPooling2D(pool_size=(2, 2)))

    if add_Dropout_layer == True:
        model.add(Dropout(Dropout_value))

    model.add(Flatten())

    if add_dense_layer == True:
        model.add(Dense(128, activation=activation_fn))

        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))

    if dense_layer == 2 and add_dense_layer == True:
        model.add(Dense(50, activation=activation_fn))

        if add_Dropout_layer == True:
            model.add(Dropout(Dropout_value))


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

    # Compile model
    model.compile(loss='categorical_crossentropy',
optimizer=optimizer_name, metrics=['accuracy'])

    return model
```

```
def change_no_kernel(x_train, y_train, x_test, y_test, batch_size =
200, no_of_epochs = 10):
    # build the model
    model1 = Build_baseline_Model(no_of_kernels=10)
    model2 = Build_baseline_Model(no_of_kernels = 20)
    model3 = Build_baseline_Model(no_of_kernels = 32)

    # Fit the model
    history1 = model1.fit(x_train, y_train, validation_data=(x_test,
y_test), epochs=no_of_epochs, batch_size=batch_size, verbose=2)
    history2 = model2.fit(x_train, y_train, validation_data=(x_test,
y_test), epochs=no_of_epochs, batch_size=batch_size, verbose=2)
    history3 = model3.fit(x_train, y_train, validation_data=(x_test,
y_test), epochs=no_of_epochs, batch_size=batch_size, verbose=2)

    # Final evaluation of the model
    score1 = model1.evaluate(x_test, y_test, verbose=0)
    score2 = model2.evaluate(x_test, y_test, verbose=0)
    score3 = model3.evaluate(x_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])

    plt.title('Model accuracy based on no. of hidden layers')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = ' no_of_kernel = 10(Test accuracy -
{})'.format(score1[1])
    legend2 = 'no_of_kernel = 20(Test accuracy -
{})'.format(score2[1])
    legend3 = 'no_of_kernel = 32(Test accuracy -
{})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    #plt.show()
    plt.savefig(PLOTS_DIR + '/acc_CNN_{}.png'.format(plot_type))
    plt.close()
    print("CNN Error of Model 1: %.2f%%" % (100-score1[1]*100))
    print("CNN Error of Model 2: %.2f%%" % (100-score2[1]*100))
    print("CNN Error of Model 3: %.2f%%" % (100-score3[1]*100))

def change_layers(X_train, y_train, X_test, y_test, batch_size = 200,
no_of_epochs = 10):
    plot_type='layers'
    model1 = Build_baseline_Model(layers = 1, add_dense_layer=False)
    model2 = Build_baseline_Model(layers = 1)
    model3 = Build_baseline_Model(layers = 2)

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plot_type = 'No_kernel'
    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on no. of hidden layers')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = '1 Convolution layer(Test accuracy -
{})'.format(score1[1])
    legend2 = '1 Convolution layer, 1 dense layer(Test accuracy -
{})'.format(score2[1])
    legend3 = '2 Convolution layers, 1 dense layer(Test accuracy -
{})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/acc_CNN_{}.png'.format(plot_type))
    plt.show()
    plt.close()

def change_Activation_fn(X_train, y_train, X_test, y_test, batch_size
= 200, no_of_epochs = 10):
    plot_type='activation'
    model1 = Build_baseline_Model(activation_fn= "relu")
    model2 = Build_baseline_Model(activation_fn= "tanh")
    model3 = Build_baseline_Model(activation_fn= "sigmoid")

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on no. of Activation function')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = 'Relu(Test accuracy - {})'.format(score1[1])
    legend2 = 'Tanh(Test accuracy - {})'.format(score2[1])
    legend3 = 'Sigmoid(Test accuracy - {})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/act_CNN_{}.png'.format(plot_type))
    plt.show()
    plt.close()

def change_kernel_size(X_train, y_train, X_test, y_test, batch_size =
200, no_of_epochs = 10):
    plot_type='kernel_size'
    model1 = Build_baseline_Model(kernel_size=(3, 3))
    model2 = Build_baseline_Model(kernel_size=(5, 5))
    model3 = Build_baseline_Model(kernel_size=(15, 15))

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history3 = model3.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)

    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)
    score3 = model3.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['val_acc'])
    plt.plot(history2.history['val_acc'])
    plt.plot(history3.history['val_acc'])
    plt.title('Model accuracy based on kernel size')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    legend1 = '3x3 kernel(Test accuracy - {})'.format(score1[1])
    legend2 = '5x5 kernel(Test accuracy - {})'.format(score2[1])
    legend3 = '15x15 kernel(Test accuracy - {})'.format(score3[1])
    plt.legend([legend1, legend2, legend3], loc='lower right')
    plt.savefig(PLOTS_DIR + '/acc_CNN_{}.png'.format(plot_type))
    plt.close()

def change_overfitting(X_train, y_train, X_test, y_test, batch_size =
200, no_of_epochs = 10):
    plot_type='overfitting'
    model1 = Build_baseline_Model()
    model2 = Build_baseline_Model(add_Dropout_layer = True)

    history1 = model1.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
    history2 = model2.fit(X_train, y_train, validation_data=(X_test,
y_test), batch_size=batch_size, epochs=no_of_epochs, verbose=2)
```

```
    score1 = model1.evaluate(X_test, y_test, verbose=0)
    score2 = model2.evaluate(X_test, y_test, verbose=0)

    plt.plot(history1.history['acc'])
    plt.plot(history1.history['val_acc'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='lower right')
    #plt.show()
    plt.savefig(PLOTS_DIR +
'/acc_CNN_test_train.png'.format(plot_type))
    plt.close()

    plt.plot(history1.history['loss'])
    plt.plot(history1.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper right')
    #plt.show()
    plt.savefig(PLOTS_DIR +
'/loss_CNN_test_train.png'.format(plot_type))
    plt.close()

    plt.plot(history1.history['val_loss'])
    plt.plot(history2.history['val_loss'])
    plt.title('Model loss using different overfitting techniques')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    legend1 = 'No technique used(Test accuracy -
{})'.format(score1[1])
    legend2 = 'Dropout(Test accuracy - {})'.format(score2[1])
    plt.legend([legend1, legend2], loc='upper right')
    #plt.show()
    plt.savefig(PLOTS_DIR + '/loss_CNN_{}.png'.format(plot_type))
    plt.close()
```

```
if __name__ == '__main__':
    x_train, y_train, x_test, y_test = Get_Data()
    change_no_kernel(x_train,y_train,x_test,y_test,no_of_epochs=5)
    change_layers(x_train,y_train,x_test,y_test,no_of_epochs=5)

change_Activation_fn(x_train,y_train,x_test,y_test,no_of_epochs=5)
    change_kernel_size(x_train,y_train,x_test,y_test,no_of_epochs=5)
    change_overfitting(x_train, y_train, x_test,y_test, no_of_epochs
= 5)
    pass
```