8/28/2018

Computer Vision & Machine Learning

Assignment 1

Assignment 1

Katta Yatheendra Mouli 150101029

Patoliya Meetkumar Krushnadas 150101045

Piyush Jain 150101046

Shivam Gupta 150101068

Shubham Singhal 150101072

Turnitin

PROBLEM STATEMENT

- Use the following machine learning algorithms to classify the given image sample from MNIST database into one of the 10 possible classes (0 to 9):
  1. Softmax regression without convolutional layers
  2. Softmax regression with convolutional layers
- Further, Exercise on various parameters 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.

LANGUAGE / LIBRARIES USED

- Python3
- Keras API for building all neural network models. Keras uses Tensorflow libraries in its backend.
- Matplotlib was used for visualisation of results using graphs.

STUDYING THE DATASET

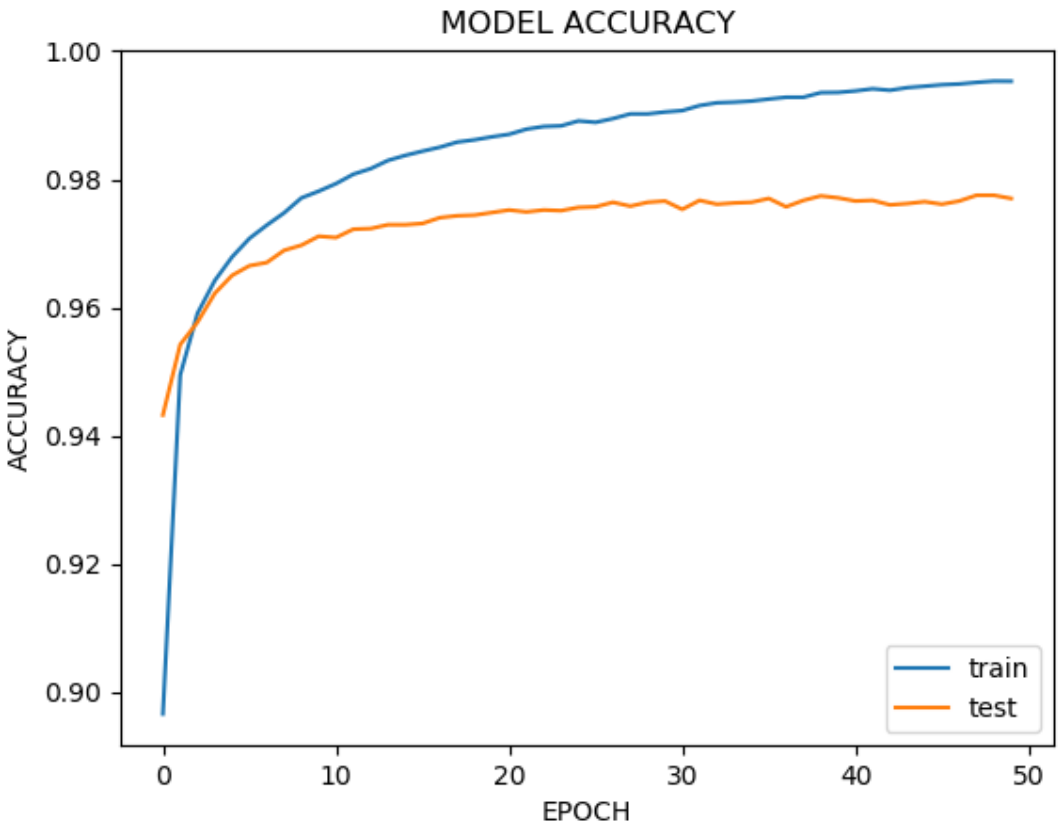
- The MNIST data comes in two parts. The first part contains 60,000 images of numerical digits to be used as training data. The images are greyscale and 28 by 28 pixels in size. The second part of the MNIST data set consists of 10,000 images to be used as test data. We'll use the test data to evaluate accuracy of our neural network to recognize digits. We regard each training input as a 28×28=784 1-dimensional vector. Image in the dataset is classified to belong to any one of the 10 classes for digits 0-9.

SOFTMAX REGRESSION WITHOUT CONVOLUTIONAL LAYERS

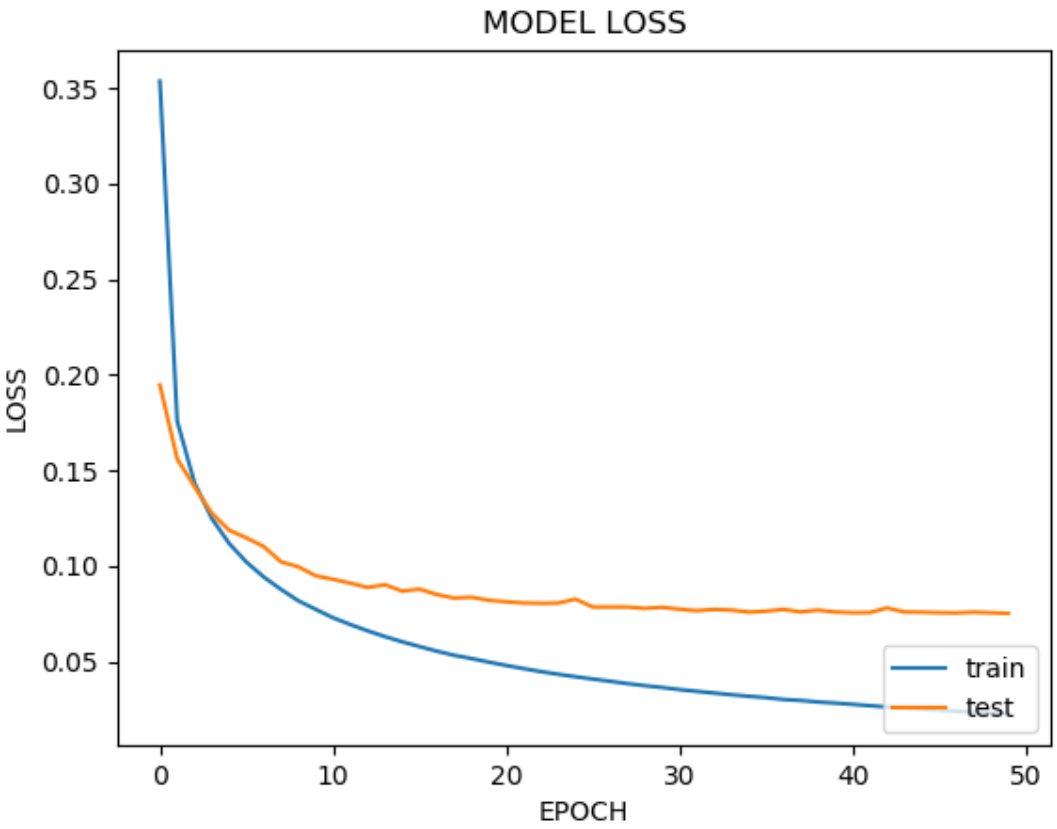
The Network Model was prepared with following properties:

- 1st layer i.e. Input Layer takes Image as Input.
- It has two hidden layers consisting of 100 neurons and 50 neurons respectively,
- Finally there is an output layer with 10 neurons.
- **Softmax** function is used as the activation function in the output layer. Softmax layer outputs a probability distribution over the 10 categories for the input image.
- Batch size of 256 is used and model is run for 25 epochs.

Accuracy on Test Data : 97.49%



Loss on Test Data : 0.012

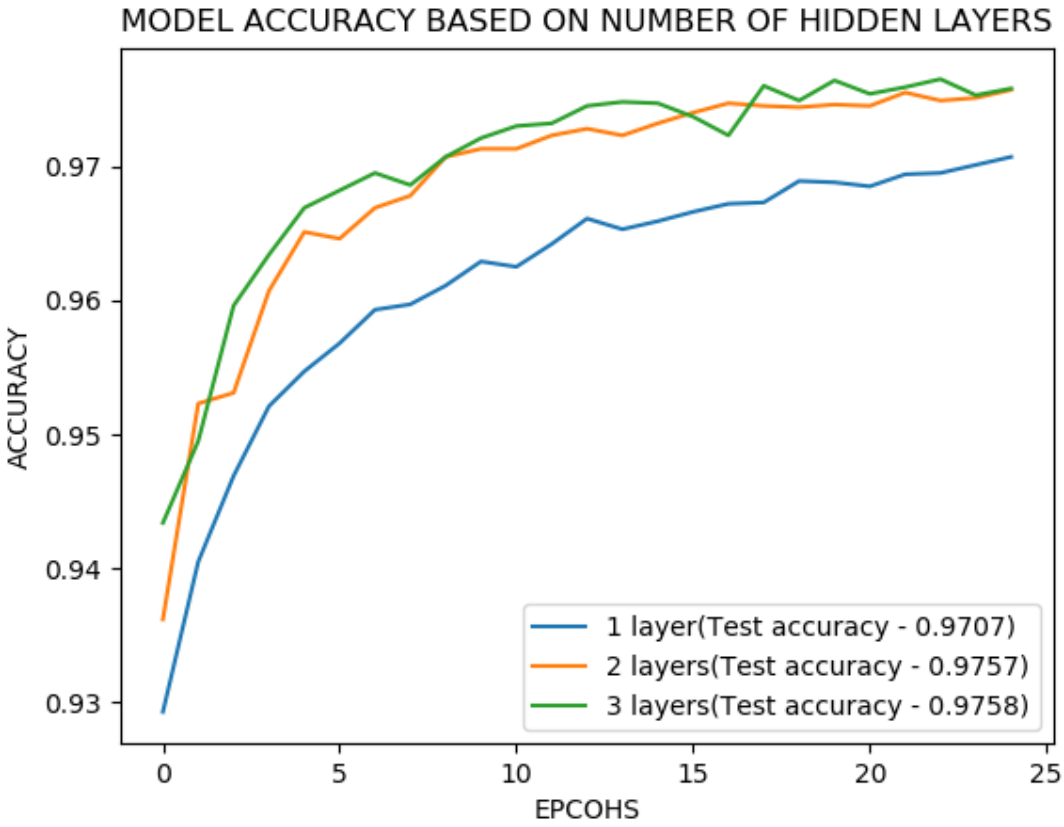


CHANGING HYPER-PARAMETERS

We change the values of hyper-parameters to improve the results.

NUMBER OF HIDDEN LAYERS

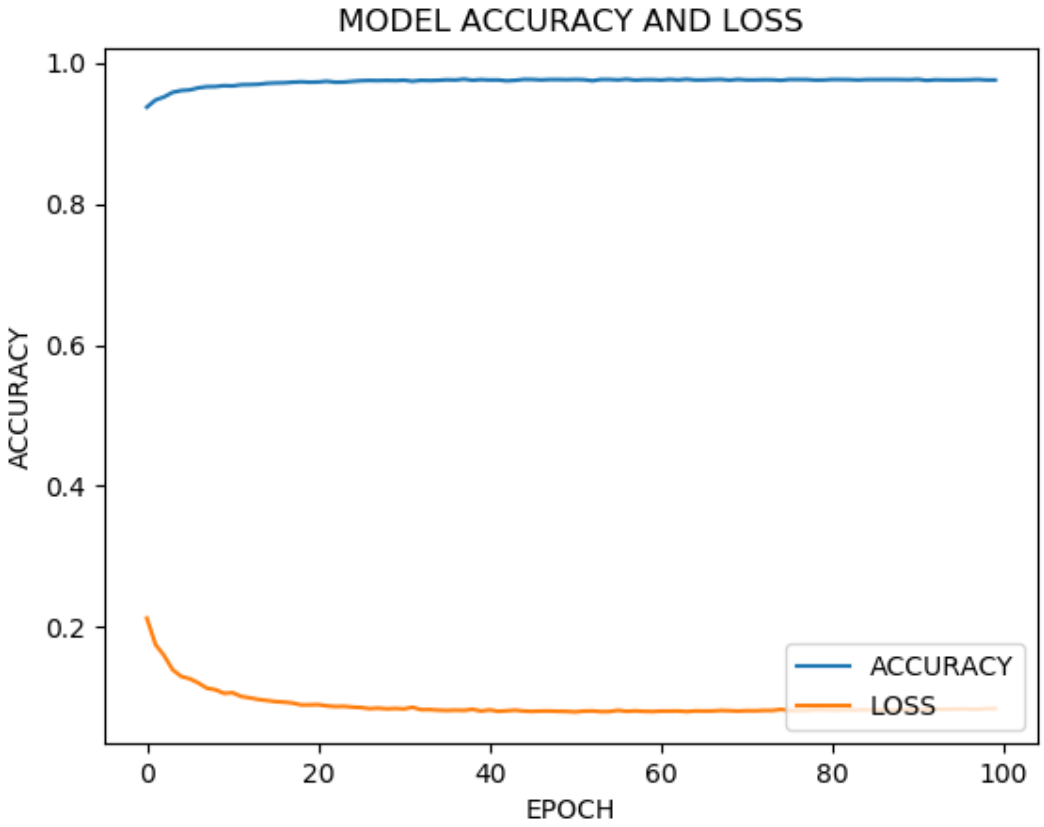
We used diff. no of hidden layers to train our neural network.
Neural Network was trained with 1,2 and 3 hidden layers.
Accuracy for each of them is plotted in graph below.



It can be observed that there is significant improvement in accuracy when hidden layer is changed from 1 to 2.
But when we further increase the no. of hidden layers, there is not much significant improvement.
Neural Network with 2 layers was complex enough and was able to predict them properly. Adding another layer does not add some extra complexity.

NUMBER OF EPOCHS

An epoch is defined as a single pass through your entire training set while training a machine learning model. In a single epoch, all training samples are presented to your model once. So the total number of epochs in training a model gives the number of cycles through the entire training dataset.
Higher value of epochs usually gives high accuracy on training data, but can be less on test data.

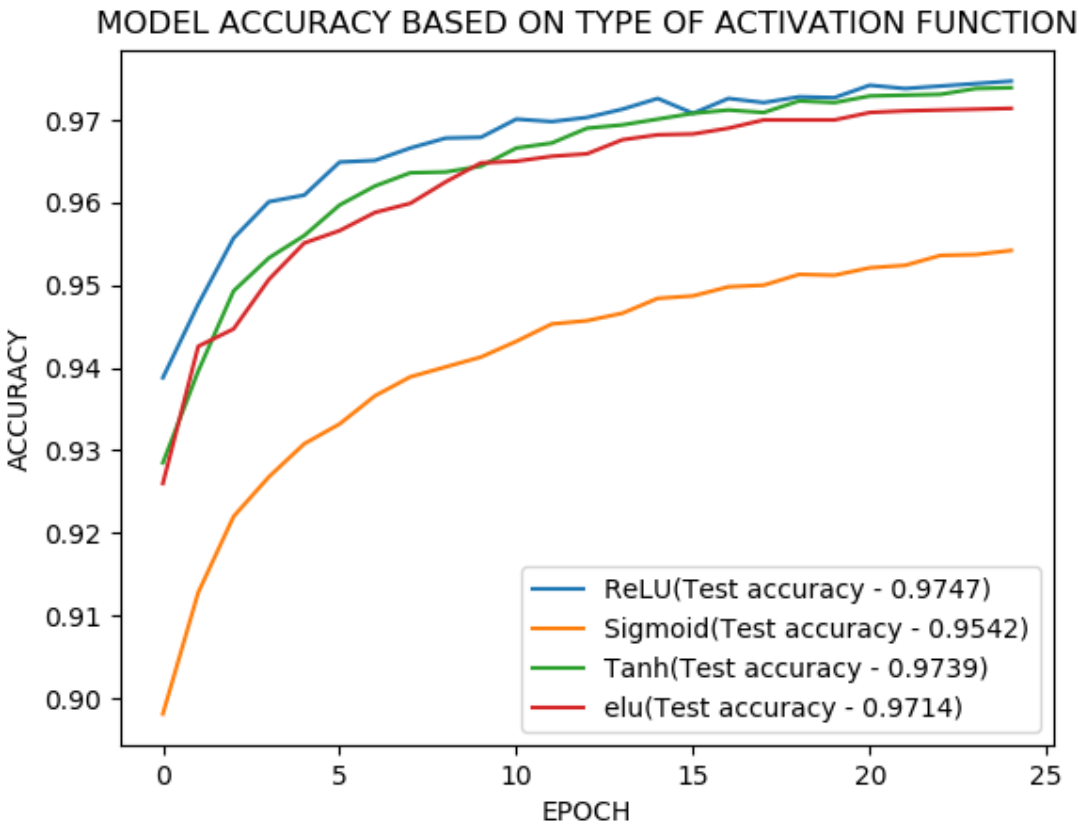


With higher no. of Epoch Values, there is no significant change in accuracy.

TYPE OF ACTIVATION FUNCTION

The activation function of a node defines the output of that node given an input or set of inputs.
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.

For Hidden layers in neural networks, we have trained using 4 different Activation Functions. For example Sigmoid, Tanh, ELU and ReLU.



Accuracy of Sigmoid function is very less as compared to other functions. Gradient in Sigmoid function at extreme values is close to 0 and thus less improvement is observed.

OVERFITTING PROBLEM

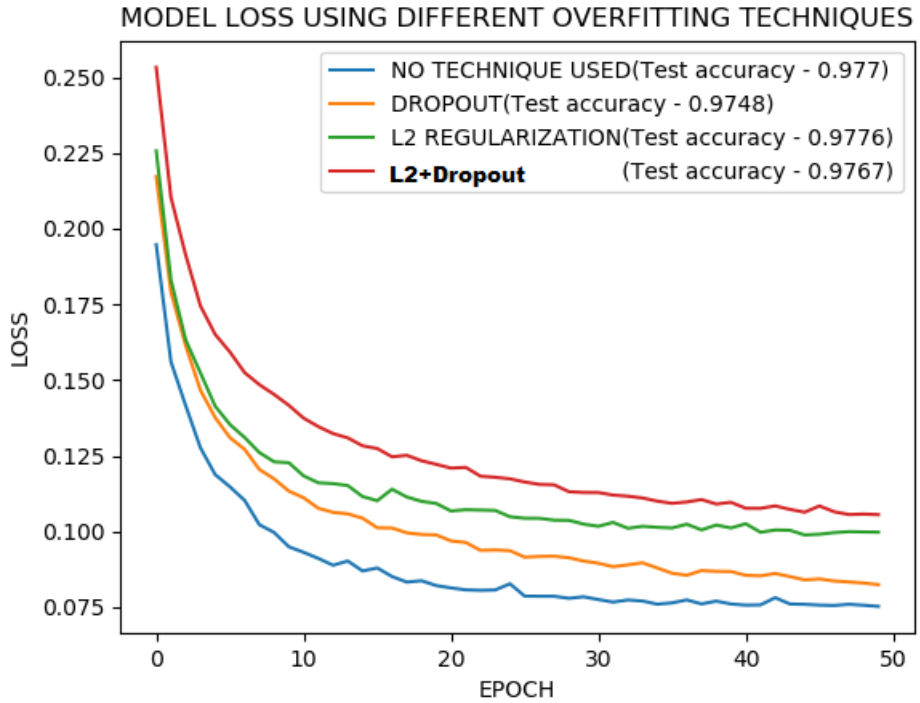
Overfitting is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.

Reasons for overfitting :

- It occurs when our model does not generalize well to real-world cases although it fits the training data well.
- It happens when the model's complexity is unnecessarily increased
- Model is trained for high epochs than required. After a point of time, test accuracy ceases to increase as shown in the below figure.

WAYS TO TACKLE OVERFITTING PROBLEM :

- Dropout – It is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks.
- L2 regularization – L2 regularization, also called Ridge regression, adds squared magnitude of coefficient as penalty term to the loss function.



These techniques reduce the loss in test data as can be seen in the above plot. Higher loss means higher level of overfitting.

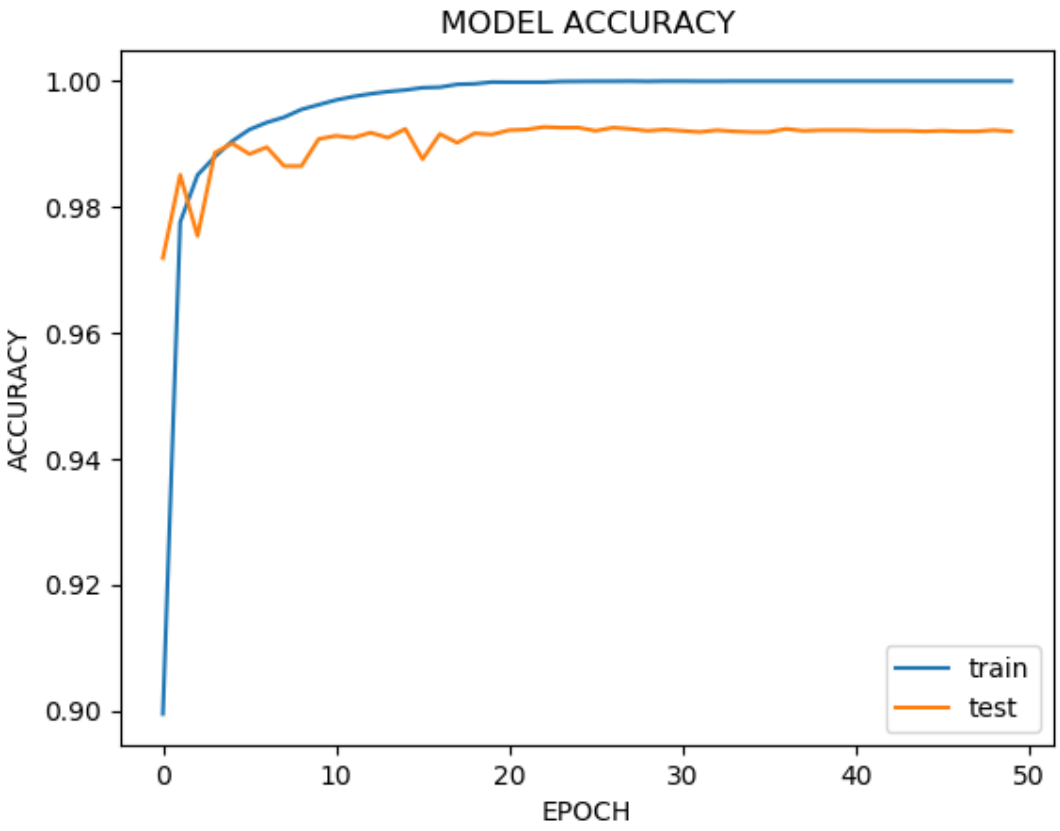
SOFTMAX REGRESSION WITH CONVOLUTIONAL LAYERS

This model contains following layers :

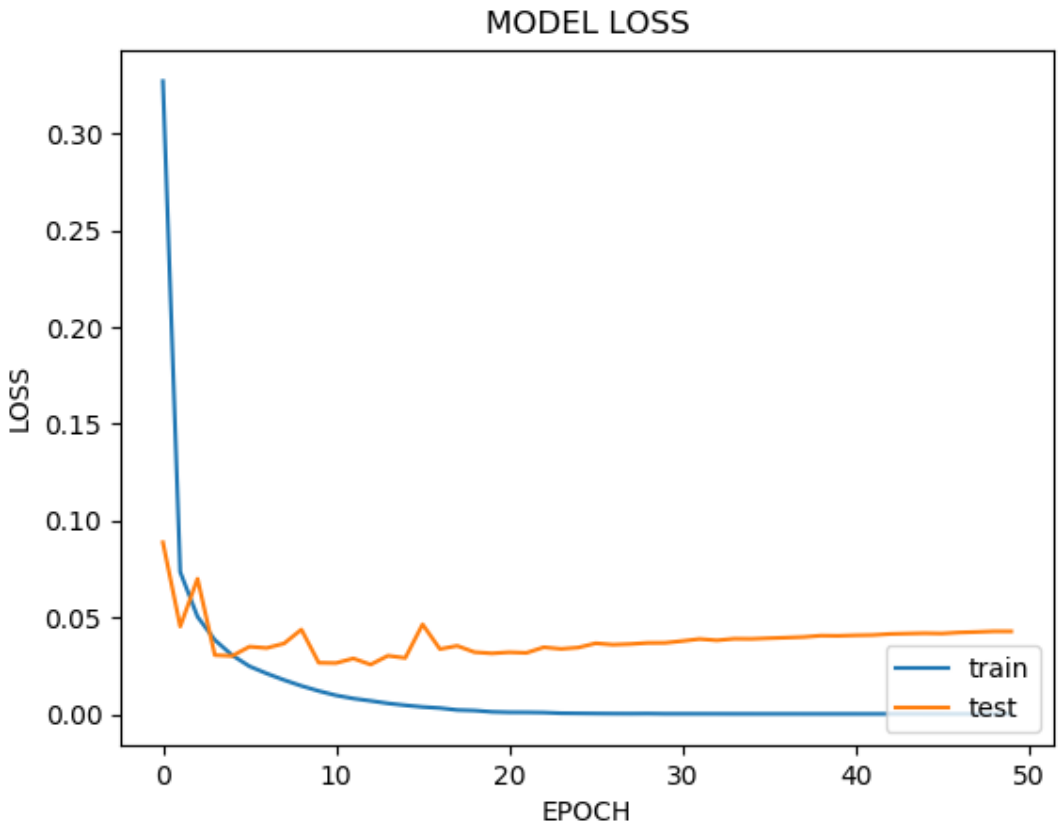
- Conv2D Layer - This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
- MaxPooling2D Layer – This layer performs down-sampling by dividing the input into rectangular pooling regions, and computing the maximum of each region. We added a 2D max pooling layer of 2x2 over our convolutional layer.

Softmax is used as the activation function for the output layer. ReLU activation function is used for hidden layers.
Batch size of 128 is used and model is run for 15 epochs.

Accuracy on test data - 98.47%



Loss on CNN Test Data 0.0461

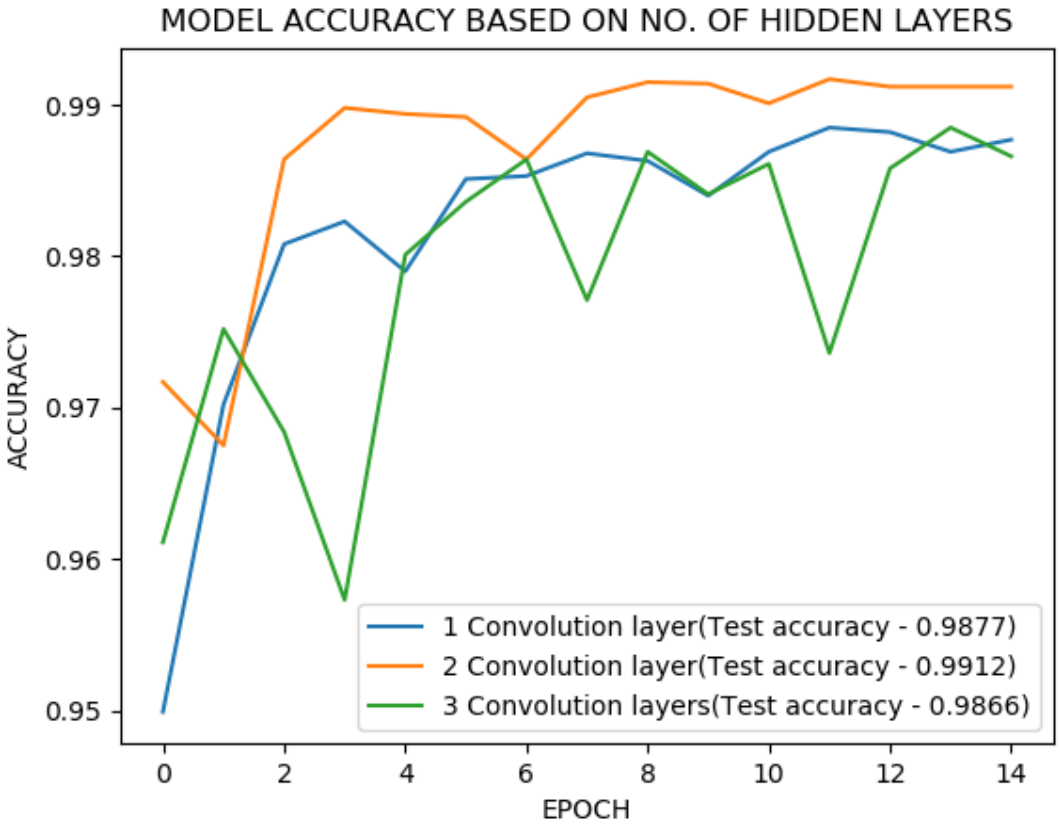


CHANGING HYPER-PARAMETERS

We change the values of hyper-parameters to improve the results.

NUMBER OF HIDDEN LAYERS

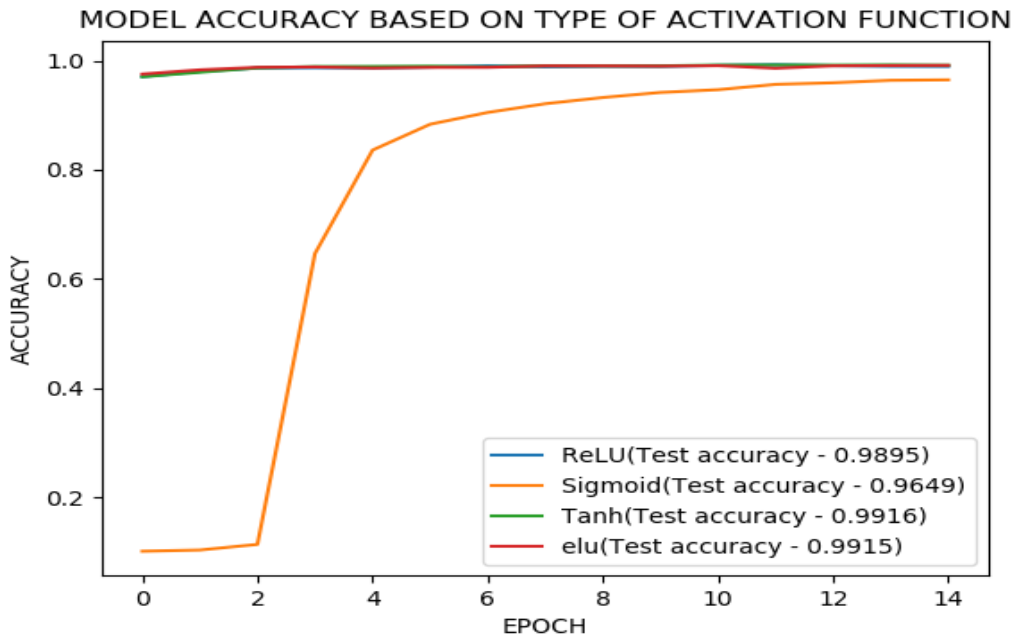
We try with three different combination of convolutional and dense layers.



Significant improvement in accuracy is observed when no. of hidden layers is changed from 1 to 2.
But when we further increase the no. of hidden layers, there is not much significant improvement.
Neural Network with 2 layers was complex enough and was able to predict them properly. Adding another layer does not add some extra complexity.

TYPE OF ACTIVATION FUNCTION

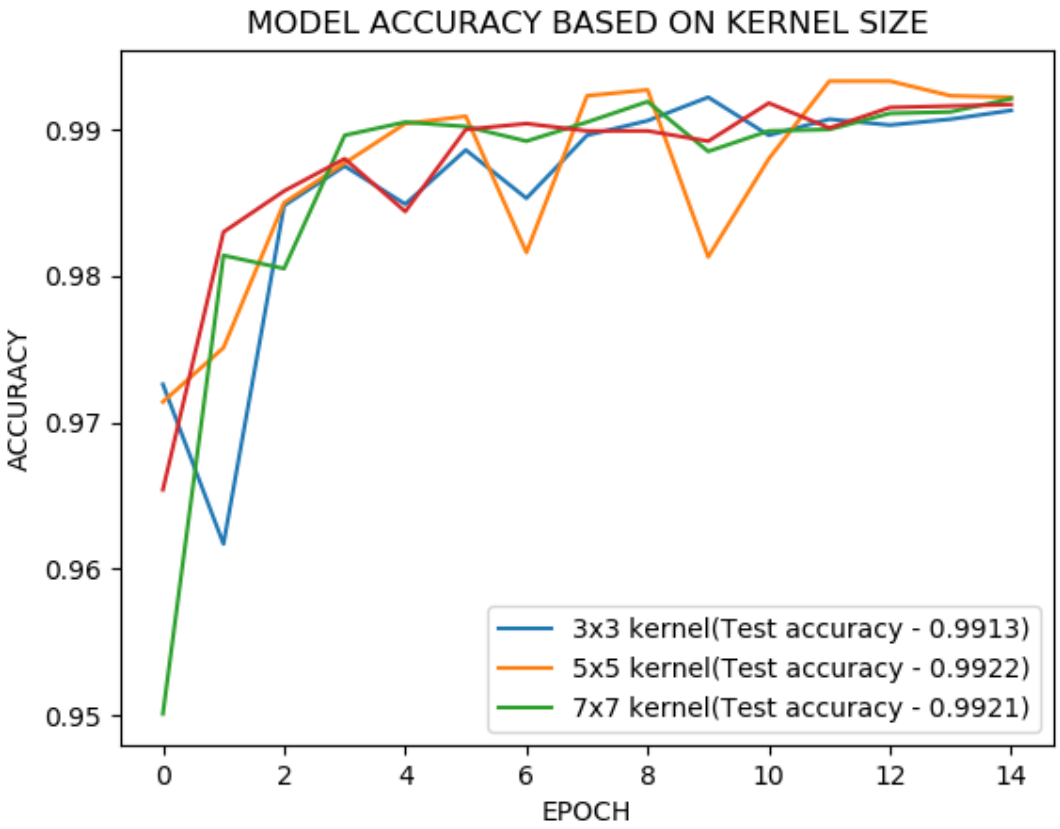
We tried to train our neural network using diff. activation functions. Sigmoid in this case also does not produce good results as in 1st case.



KERNEL SIZE

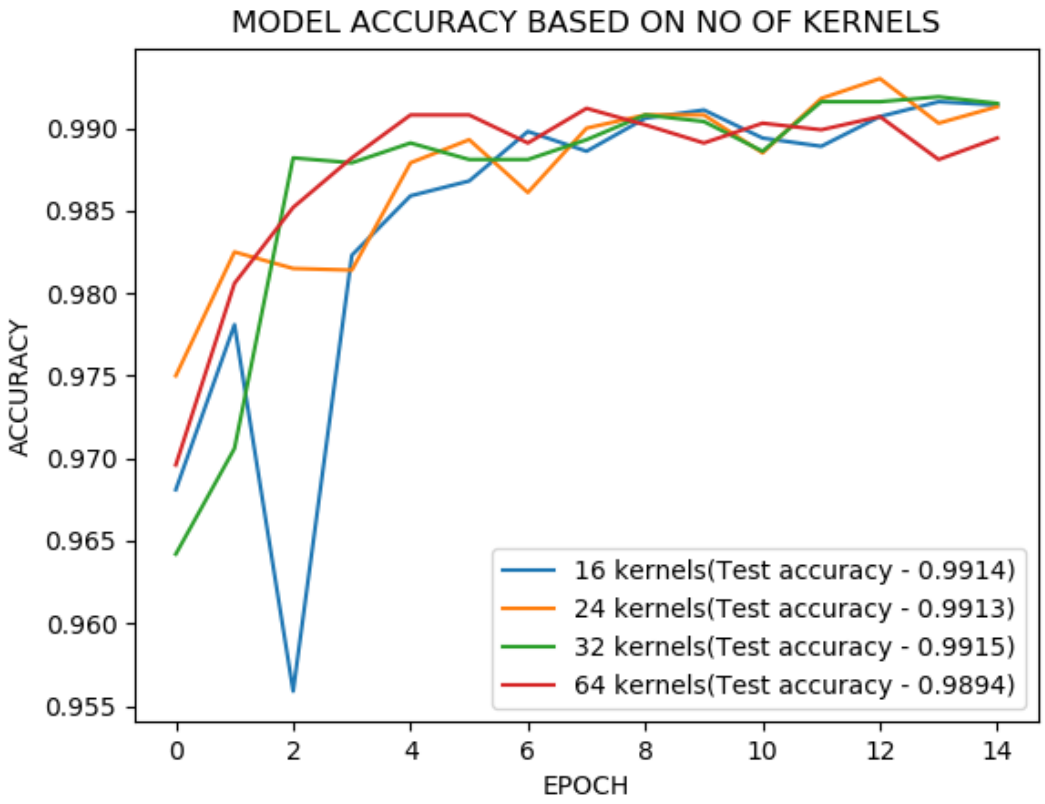
We experimented on 3 different kernel sizes, one being 3x3, 5x5 and 7x7. Accuracy in casr of 5x5 is observed to be better than 3x3 and 7x7.

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 intermediate value of filter size is giving best results.

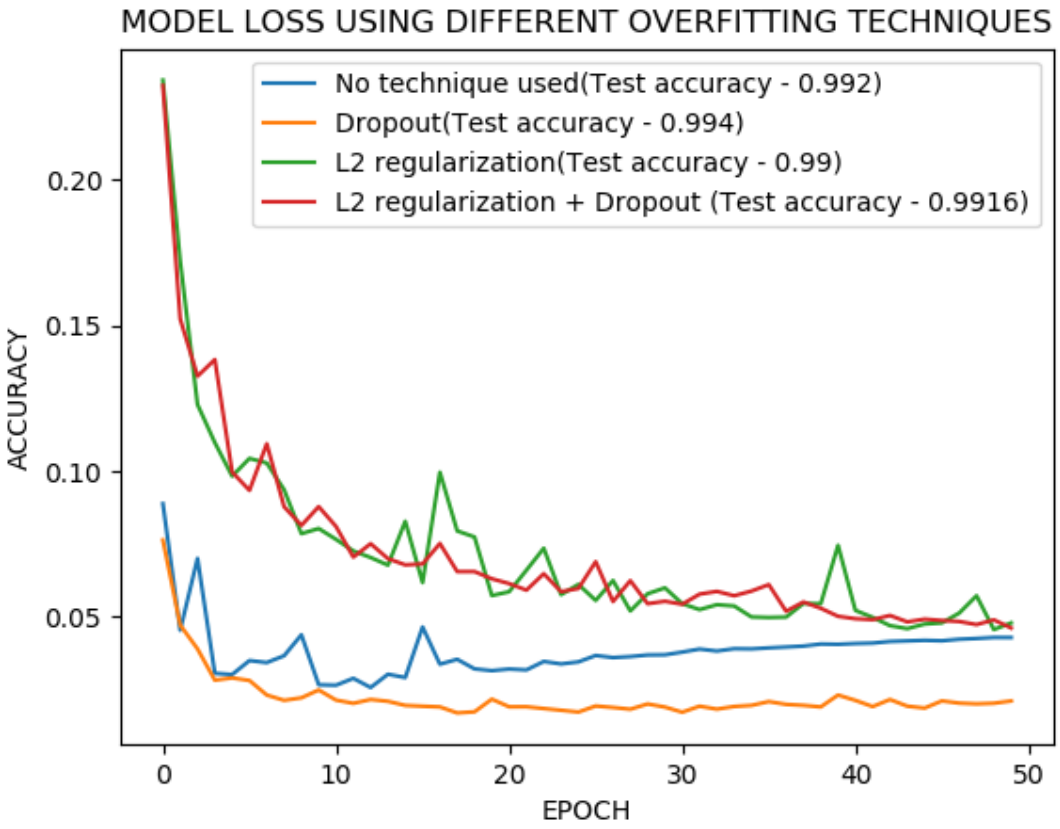


NUMBER OF KERNELS

We experiment with 16, 24,32 and 64 kernels in our model.
Number of kernels can be different for different layers also. Based on the task which we are doing, no. of kernels can be significantly different. With the higher number of Kernels, there are higher no. of parameters to learn and complexity of model is also higher.


MODEL ACCURACY BASED ON NO OF KERNELS
ACCURACY
EPOCH
0.990
0.985
0.980
0.975
0.970
0.965
0.960
0.955
0
2
4
6
8
10
12
14
16 kernels(Test accuracy - 0.9914)
24 kernels(Test accuracy - 0.9913)
32 kernels(Test accuracy - 0.9915)
64 kernels(Test accuracy - 0.9894)

Because of similar reasons, Convolutional Neural Networks also show the problem of overfitting. As done previously, here also we try using dropout and L2 regularisation to avoid overfitting.
Dropout in case of Neural Networks is helpful in many ways. First, it tries to remove overfitting and Second, it decreases the dependency on some of selective neurons in a layer which ensures that decision is based on the basis of entire image. This makes them less likely to pick local characteristics in training image.



CONCLUSION

- There is a trade-off between time and accuracy. Higher the time, for which we train our Neural Networks, they are able to achieve higher accuracy, but with increasing Epoch to much higher value can lead to overfitting.
- Convolutional Neural Networks perform better in case of images as they are able to map spacial features like edges, corners etc.
- It shows the importance of validation data. Different value of hyper-parameters e.g. - number of hidden layers, activation function etc. result in varied accuracy even for same neural network.
- It can be also be seen that Architecture of Neural Networks depend heavily on the problem statement. CNN works well for Computer Vision Problems but similar CNN architectures do not produce good results for NLP problems.

PART 1 (Without CNN layers)

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout ,Activation
from keras.optimizers import RMSprop
from keras.regularizers import l2
import matplotlib.pyplot as plt


class Cvml:
    def __init__(self):
        self.plots_dir = "resulting_plots2"

    def create_and_train_model(self,
                               batch_size=256,
                               num_classes=10,
                               epochs=25,
                               layers=2,
                               activation_fn='relu',
                               add_dropout=False,
                               l2_reg=0.0):
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        x_train = x_train.reshape(60000, 784)
        x_test = x_test.reshape(10000, 784)
        x_train = x_train.astype('float32')
        x_test = x_test.astype('float32')
        x_train /= 255
        x_test /= 255
        print('NO OF TRAINING EXAMPLES', x_train.shape[0],)
```

```
    print('NO OF TESTS EXAMPLES', x_test.shape[0],)

    # convert class vectors to binary class matrices
    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    model = Sequential()
    model.add(Dense(100,                    kernel_regularizer=l2(l2_reg),
input_shape=(784,)))
    model.add(Activation(activation_fn))
    if add_dropout:
      model.add(Dropout(0.25))

    if layers >= 2:
      model.add(Dense(50, kernel_regularizer=l2(l2_reg)))
      model.add(Activation(activation_fn))
      if add_dropout:
        model.add(Dropout(0.25))
    if layers >= 3:
      model.add(Dense(30, kernel_regularizer=l2(l2_reg)))
      model.add(Activation(activation_fn))
      if add_dropout:
        model.add(Dropout(0.25))

    model.add(Dense(10, activation='softmax'))

    model.summary()

    model.compile(loss='categorical_crossentropy',
              optimizer='adagrad',
              metrics=['accuracy'])

    history = 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('LOSS IN TEST DATA:', score[0])
    print('ACCURACY IN TEST DATA:', score[1])
```

```
        return history, score

    def test_no_of_layers(self,
                          plot_type='layers'):
        h1, s1 = self.create_and_train_model(layers=1)
        h2, s2 = self.create_and_train_model(layers=2)
        h3, s3 = self.create_and_train_model(layers=3)

        legend1 = '1 layer(Test accuracy - {})'.format(s1[1])
        legend2 = '2 layers(Test accuracy - {})'.format(s2[1])
        legend3 = '3 layers(Test accuracy - {})'.format(s3[1])

self.plot([h1.history['val_acc'],h2.history['val_acc'],h3.history['val_acc']],
                  'MODEL ACCURACY BASED ON NUMBER OF HIDDEN LAYERS',
                  'EPCOHS',
                  'ACCURACY',
                  [legend1, legend2, legend3],
                  self.plots_dir + '/accuracy_simple_{}.png'.format(plot_type)
                  )

    def test_epochs(self,
                    plot_type='epochs'):
        h1, s1 = self.create_and_train_model(epochs=100)
        legend1 = 'ACCURACY'
        legend2 = 'LOSS'

        self.plot(
            [h1.history['val_acc'], h1.history['val_loss']],
            'MODEL ACCURACY AND LOSS',
            'EPCOHS',
            'ACCURACY',
            [legend1, legend2],
            self.plots_dir + '/accuracy_simple_{}.png'.format(plot_type)
            )

    def test_activation_fun(self,
                            plot_type='activation'):
        h1, s1 = self.create_and_train_model(activation_fn='relu')
        h2, s2 = self.create_and_train_model(activation_fn='sigmoid')
        h3, s3 = self.create_and_train_model(activation_fn='tanh')
```

```
        h4, s4 = self.create_and_train_model(activation_fn='elu')

        legend1 = 'ReLU(Test accuracy - {})'.format(s1[1])
        legend2 = 'Sigmoid(Test accuracy - {})'.format(s2[1])
        legend3 = 'Tanh(Test accuracy - {})'.format(s3[1])
        legend4 = 'elu(Test accuracy - {})'.format(s4[1])

        self.plot(
            [h1.history['val_acc'], h2.history['val_acc'], h3.history['val_acc'],
h4.history['val_acc']],
            'MODEL ACCURACY BASED ON TYPE OF ACTIVATION FUNCTION',
            'EPCOHS',
            'ACCURACY',
            [legend1, legend2, legend3, legend4],
            self.plots_dir + '/accuracy_simple_{}.png'.format(plot_type)
            )

    def test_overfitting(self,
                         plot_type='overfitting'):
        h1, s1 = self.create_and_train_model(epochs=50)
        h2, s2 = self.create_and_train_model(epochs=50,
add5_dropout=True)
        h3, s3 = self.create_and_train_model(epochs=50, l2_5reg=0.0001)
        h4, s4 = self.create_and_train_model(epochs=50,
add_dropout=True,
                                             l2_reg=0.0001)

        self.plot(
            [h1.history['acc'], h1.history['val_acc']],
            'MODEL ACCURACY ',
            'EPCOHS',
            'ACCURACY',
            ['train', 'test'],
            self.plots_dir +
'/accuracy_simple_test_train.png'.format(plot_type)
        )

        self.plot(
            [h1.history['loss'], h1.history['val_loss']],
            'MODEL LOSS ',
```

```
            'EPCOHS',
            'LOSS',
            ['train', 'test'],
            self.plots_dir + '/loss_simple_test_train.png'.format(plot_type)
        )

        legend1 = 'NO TECHNIQUE USED(Test accuracy - {})'.format(s1[1])
        legend2 = 'DROPOUT(Test accuracy - {})'.format(s2[1])
        legend3 = 'L2 REGULARIZATION(Test accuracy - {})'.format(s3[1])
        legend4 = 'L2 REGULARIZATION with DROPOUT(Test accuracy - {})'.format(
            s4[1])

        self.plot(
            [h1.history['val_loss'], h2.history['val_loss'], h3.history['loss'], h4.history['val_loss']],
            'MODEL LOSS USING DIFFERENT OVERFITTING TECHNIQUES',
            'EPCOHS',
            'LOSS',
            [legend1, legend2, legend3, legend4],
            self.plots_dir + '/loss_simple_{}.png'.format(plot_type)
        )

    def plot(self, plots, title, xlabel, ylabel, legend, save_path):
        for i in plots:
            plt.plot(i)
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.legend(legend, loc="lower right")
        plt.savefig(save_path)
        plt.close()


if __name__ == '__main__':
    obj = Cvml()
    obj.test_no_of_layers()
    obj.test_epochs()
    obj.test_activation_fun()
    obj.test_overfitting()
```

PART 2 (With CNN layers)

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

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D, MaxPooling2D
from keras.regularizers import l2
from keras import backend as K
import matplotlib.pyplot as plt


class Cvml:
    def __init__(self):
        self.img_rows = 28
        self.img_cols = 28
        self.plots_dir = "resulting_plots"

    def create_and_train_model(self,
                               batch_size=256,
                               num_classes=10,
                               epochs=15,
                               layers=2,
                               activation_fn='relu',
                               add_dropout=False,
                               kernel_size=(3, 3),
                               no_of_kernels=32,
                               l2_reg=0.0):
        (x_train, y_train), (x_test, y_test) = mnist.load_data()

        x_train = x_train.reshape(x_train.shape[0], self.img_rows,
self.img_cols, 1)
```

```
    x_test = x_test.reshape(x_test.shape[0], self.img_rows, self.img_cols, 1)
    input_shape = (self.img_rows, self.img_cols, 1)

    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255
    print('X_TRAIN SHAPE:', x_train.shape)
    print('NO OF TRAINING EXAMPLES', x_train.shape[0],)
    print('NO OF TESTS EXAMPLES', x_test.shape[0],)

    y_train = keras.utils.to_categorical(y_train, num_classes)
    y_test = keras.utils.to_categorical(y_test, num_classes)

    model = Sequential()
    model.add(Conv2D(no_of_kernels,
                     kernel_size=kernel_size,
                     input_shape=input_shape,
                     kernel_regularizer=l2(l2_reg)))
    model.add(Activation(activation_fn))

    model.add(MaxPooling2D(pool_size=(2, 2)))

    for i in range(1,layers) :
      model.add(Conv2D(64, kernel_size, kernel_regularizer=l2(l2_reg)))
      model.add(Activation(activation_fn))
      model.add(MaxPooling2D(pool_size=(2, 2)))
      if add_dropout:
        model.add(Dropout(0.25))
    model.add(Flatten())

    model.add(Dense(128, activation=activation_fn))
    model.add(Activation(activation_fn))
    if add_dropout:
      model.add(Dropout(0.25))
    model.add(Dense(num_classes, activation='softmax'))

    model.compile(loss=keras.losses.categorical_crossentropy,
                  optimizer=keras.optimizers.Adadelta(),
```

```
                metrics=['accuracy'])

        history = 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('LOSS IN TEST DATA:', score[0])
        print('ACCURACY IN TEST DATA:', score[1])
        return history, score

    def test_no_of_layers(self, plot_type='layers'):
        h1, s1 = self.create_and_train_model(layers=1)
        h2, s2 = self.create_and_train_model(layers=2)
        h3, s3 = self.create_and_train_model(layers=3)

        legend1 = '1 Convolution layer(Test accuracy - {})'.format(s1[1])
        legend2 = '2 Convolution layer(Test accuracy - {})'.format(s2[1])
        legend3 = '3 Convolution layers(Test accuracy - {})'.format(s3[1])


self.plot([h1.history['val_acc'],h2.history['val_acc'],h3.history['val_acc']],
                "MODEL ACCURACY BASED ON NO. OF HIDDEN LAYERS",
                "EPOCH",
                "ACCURACY",
                [legend1, legend2, legend3],
                self.plots_dir + '/accuracy_cnn_{}.png'.format(plot_type)
                )

    def test_activation_fun(self, plot_type='activation'):
        h1, s1 = self.create_and_train_model(activation_fn='relu')
        h2, s2 = self.create_and_train_model(activation_fn='sigmoid')
        h3, s3 = self.create_and_train_model(activation_fn='tanh')
        h4, s4 = self.create_and_train_model(activation_fn='elu')

        legend1 = 'ReLU(Test accuracy - {})'.format(s1[1])
        legend2 = 'Sigmoid(Test accuracy - {})'.format(s2[1])
        legend3 = 'Tanh(Test accuracy - {})'.format(s3[1])
        legend4 = 'elu(Test accuracy - {})'.format(s4[1])
```

```
self.plot([h1.history['val_acc'],h2.history['val_acc'],h3.history['val_acc'],h4.history['val_acc']],
            "MODEL ACCURACY BASED ON TYPE OF ACTIVATION FUNCTION",
            "EPOCH",
            "ACCURACY",
            [legend1, legend2, legend3, legend4],
            self.plots_dir + '/accuracy_cnn_{}.png'.format(plot_type)
            )

    def test_kernel_size(self, plot_type='kernel_size'):
        h1, s1 = self.create_and_train_model(kernel_size=(3, 3))
        h2, s2 = self.create_and_train_model(kernel_size=(5, 5))
        h3, s3 = self.create_and_train_model(kernel_size=(7, 7))
        h4, s4 = self.create_and_train_model(kernel_size=(9, 9))

        legend1 = '3x3 kernel(Test accuracy - {})'.format(s1[1])
        legend2 = '5x5 kernel(Test accuracy - {})'.format(s2[1])
        legend3 = '7x7 kernel(Test accuracy - {})'.format(s3[1])
        legend4 = '9x9 kernel(Test accuracy - {})'.format(s4[1])

self.plot([h1.history['val_acc'],h2.history['val_acc'],h3.history['val_acc'],h4.history['val_acc']],
            "MODEL ACCURACY BASED ON KERNEL SIZE",
            "EPOCH",
            "ACCURACY",
            [legend1, legend2, legend3, legend4],
            self.plots_dir + '/accuracy_cnn_{}.png'.format(plot_type)
            )

    def test_no_of_kernels(self, plot_type = 'no_of_kernels'):
        h1,s1 = self.create_and_train_model(no_of_kernels=16)
        h2,s2 = self.create_and_train_model(no_of_kernels=24)
        h3,s3 = self.create_and_train_model(no_of_kernels=32)
        h4,s4 = self.create_and_train_model(no_of_kernels=64)

        legend1 = '16 kernels(Test accuracy - {})'.format(s1[1])
        legend2 = '24 kernels(Test accuracy - {})'.format(s2[1])
```

```
        legend3 = '32 kernels(Test accuracy - {})'.format(s3[1])
        legend4 = '64 kernels(Test accuracy - {})'.format(s4[1])

self.plot([h1.history['val_acc'],h2.history['val_acc'],h3.history['val_acc'],h
4.history['val_acc']],
                "MODEL ACCURACY BASED ON NO OF KERNELS",
                "EPOCH",
                "ACCURACY",
                [legend1, legend2, legend3, legend4],
                self.plots_dir + '/accuracy_cnn_{}.png'.format(plot_type)
                )

    def test_overfitting(self, plot_type='overfitting'):
        h1, s1 = self.create_and_train_model(epochs=50)
        h2,         s2         =         self.create_and_train_model(epochs=50,
add_dropout=True)
        h3, s3 = self.create_and_train_model(epochs=50, l2_reg=0.01)
        h4,         s4         =         self.create_and_train_model(epochs=50,
l2_reg=0.01,add_dropout=True)

        self.plot([h1.history['acc'], h1.history['val_acc']],
                'MODEL ACCURACY',
                'EPOCH',
                'ACCURACY',
                ['train', 'test'],
                self.plots_dir                                                  +
'/accuracy_cnn_test_train.png'.format(plot_type)
                )
        self.plot([h1.history['loss'], h1.history['val_loss']],
                'MODEL LOSS',
                'EPOCH',
                'LOSS',
                ['train', 'test'],
                self.plots_dir + '/loss_cnn_test_train.png'.format(plot_type)
                )

        legend1 = 'No technique used(Test accuracy - {})'.format(s1[1])
        legend2 = 'Dropout(Test accuracy - {})'.format(s2[1])
        legend3 = 'L2 regularization(Test accuracy - {})'.format(s3[1])
        legend4 = 'L2 + Dropout (Test accuracy - {})'.format(s4[1])
```

```
        self.plot([h1.history['loss'],h2.history['loss'],h3.history['loss'],
h4.history['loss']],
                  'MODEL LOSS USING DIFFERENT OVERFITTING TECHNIQUES',
                  'EPOCH',
                  'ACCURACY',
                  [legend1, legend2, legend3, legend4],
                  self.plots_dir + '/loss_cnn_{}.png'.format(plot_type)
                  )

    def plot(self, plots, title, xlabel, ylabel, legend, save_path):
        for i in plots:
            plt.plot(i)
        plt.title(title)
        plt.xlabel(xlabel)
        plt.ylabel(ylabel)
        plt.legend(legend, loc="lower right")
        plt.savefig(save_path)
        plt.close()


if __name__ == '__main__':
    obj = Cvml()
    obj.test_no_of_layers()
    obj.test_activation_fun()
    obj.test_kernel_size()
    obj.test_no_of_kernels()
    obj.test_overfitting()
    pass
```