Assignment_13(Try_various_CNN_networks_on_MNIST_dataset)

September 18, 2018

1 OBJECTIVE :- Try various CNN networks on MNIST dataset

In [1]:
```
# Importing libraries
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.initializers import he_normal
from keras.layers.normalization import BatchNormalization
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Using TensorFlow backend.
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
```

In [2]:
```
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
    print('x_train shape:', x_train.shape)
    print(x_train.shape[0], 'train samples')
    print(x_test.shape[0], 'test samples')

    # 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)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
In [0]: # this function is used draw Categorical Crossentropy Loss VS No. of epochs plot
        def plt_dynamic(x, vy, ty):
          plt.figure(figsize=(10,5))
          plt.plot(x, vy, 'b', label="Validation Loss")
          plt.plot(x, ty, 'r', label="Train Loss")
          plt.xlabel('Epochs')
          plt.ylabel('Categorical Crossentropy Loss')
          plt.title('\nCategorical Crossentropy Loss VS Epochs')
          plt.legend()
          plt.grid()
          plt.show()
```

1.1 (1). CNN with 3 Convolutional layers and kernel size - (3X3)

```
In [5]: # Initialising the model
        model_3 = Sequential()

        # Adding first conv layer
        model_3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))

        # Adding second conv layer
        model_3.add(Conv2D(64, (3, 3), activation='relu'))

        # Adding Maxpooling layer
        model_3.add(MaxPooling2D(pool_size=(2, 2)))
```

```
        # Adding Dropout
        model_3.add(Dropout(0.25))

        # Adding third conv layer
        model_3.add(Conv2D(128, (3, 3), activation='relu'))

        # Adding Maxpooling layer
        model_3.add(MaxPooling2D(pool_size=(2, 2)))

        # Adding Dropout
        model_3.add(Dropout(0.25))

        # Adding flatten layer
        model_3.add(Flatten())

        # Adding first hidden layer
        model_3.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))

        # Adding Dropout
        model_3.add(Dropout(0.5))

        # Adding output layer
        model_3.add(Dense(num_classes, activation='softmax'))

        # Printing model Summary
        print(model_3.summary())

        # Compiling the model
        model_3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']

        # Fitting the data to the model
        history_3 = model_3.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1
```

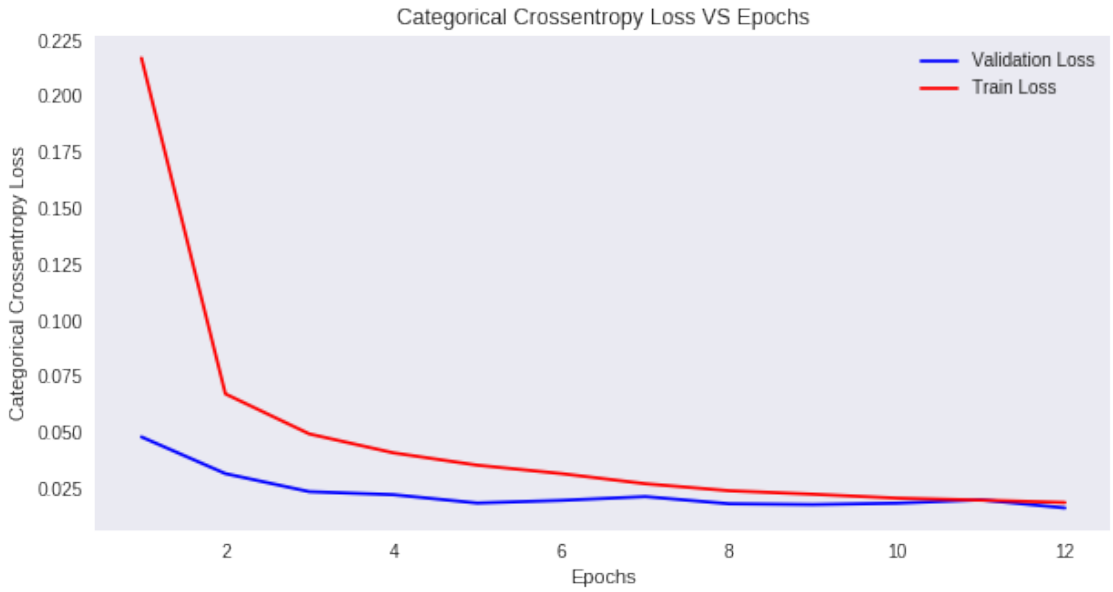
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 128)         0
```

```
_________________________________________________________________
dropout_5 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               819456
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 914,698
Trainable params: 914,698
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 222s 4ms/step - loss: 0.2171 - acc: 0.9305 - val
Epoch 2/12
60000/60000 [==============================] - 240s 4ms/step - loss: 0.0674 - acc: 0.9788 - val
Epoch 3/12
60000/60000 [==============================] - 237s 4ms/step - loss: 0.0496 - acc: 0.9850 - val
Epoch 4/12
60000/60000 [==============================] - 237s 4ms/step - loss: 0.0412 - acc: 0.9880 - val
Epoch 5/12
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0357 - acc: 0.9896 - val
Epoch 6/12
60000/60000 [==============================] - 209s 3ms/step - loss: 0.0319 - acc: 0.9901 - val
Epoch 7/12
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0274 - acc: 0.9912 - val
Epoch 8/12
60000/60000 [==============================] - 209s 3ms/step - loss: 0.0243 - acc: 0.9923 - val
Epoch 9/12
60000/60000 [==============================] - 205s 3ms/step - loss: 0.0227 - acc: 0.9927 - val
Epoch 10/12
60000/60000 [==============================] - 213s 4ms/step - loss: 0.0210 - acc: 0.9932 - val
Epoch 11/12
60000/60000 [==============================] - 206s 3ms/step - loss: 0.0201 - acc: 0.9935 - val
Epoch 12/12
60000/60000 [==============================] - 208s 3ms/step - loss: 0.0190 - acc: 0.9941 - val
```

```
In [6]: # Evaluating the model
        score = model_3.evaluate(x_test, y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])
```

```
        # Test and train accuracy of the model
        model_3_test = score[1]
        model_3_train = max(history_3.history['acc'])

        # Plotting Train and Test Loss VS no. of epochs
        # list of epoch numbers
        x = list(range(1,epochs+1))

        # Validation loss
        vy = history_3.history['val_loss']
        # Training loss
        ty = history_3.history['loss']

        # Calling the function to draw the plot
        plt_dynamic(x, vy, ty)
```

```
Test score: 0.01658615784635249
Test accuracy: 0.994
```



1.2 (2). CNN with 5 Convolutional layers and kernel size - (5X5)

```
In [10]: # Initialising the model
         model_5 = Sequential()

         # Adding first conv layer
         model_5.add(Conv2D(8, kernel_size=(5, 5),padding='same',activation='relu',input_shape=
```

```
# Adding second conv layer
model_5.add(Conv2D(16, (5, 5), activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding third conv layer
model_5.add(Conv2D(32, (5, 5),padding='same', activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding fourth conv layer
model_5.add(Conv2D(64, (5, 5),padding='same',activation='relu'))

# Adding fifth conv layer
model_5.add(Conv2D(64, (5, 5), activation='relu'))

# Adding Maxpooling layer
model_5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))

# Adding Dropout
model_5.add(Dropout(0.25))

# Adding flatten layer
model_5.add(Flatten())

# Adding first hidden layer
model_5.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))

# Adding Batch Normalization
model_5.add(BatchNormalization())

# Adding Dropout
model_5.add(Dropout(0.5))

# Adding output layer
model_5.add(Dense(num_classes, activation='softmax'))

# Printing model Summary
print(model_5.summary())
```

```
        # Compiling the model
        model_5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

        # Fitting the data to the model
        history_5 = model_5.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=
```

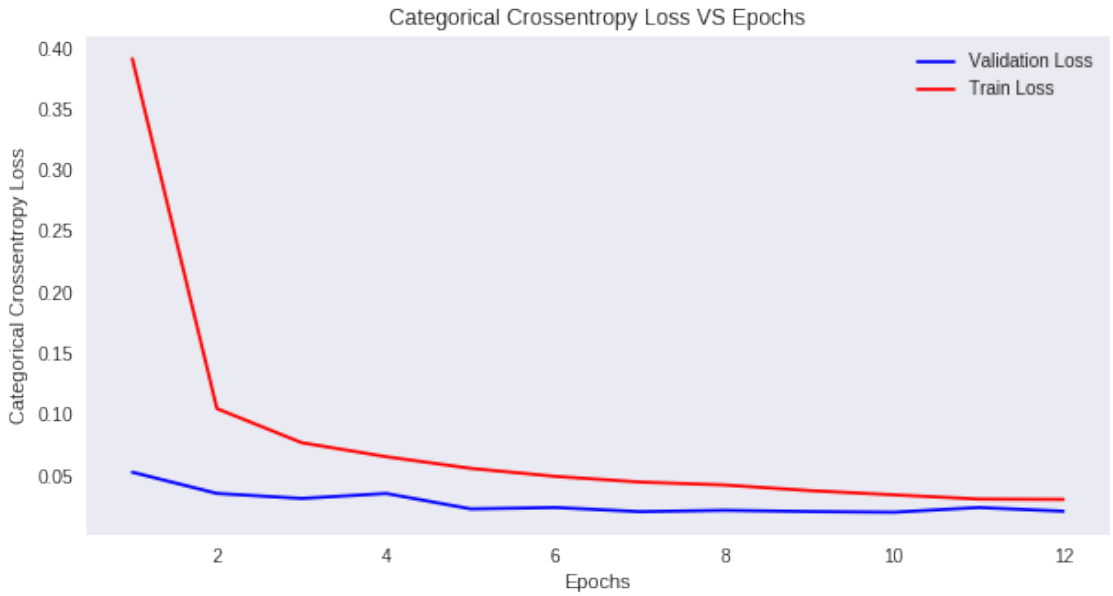
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 28, 28, 8)         208
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 24, 24, 16)        3216
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 12, 12, 16)        0
_________________________________________________________________
dropout_13 (Dropout)         (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 12, 12, 32)        12832
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 6, 6, 32)          0
_________________________________________________________________
dropout_14 (Dropout)         (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 6, 6, 64)          51264
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 2, 2, 64)          102464
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 1, 1, 64)          0
_________________________________________________________________
dropout_15 (Dropout)         (None, 1, 1, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               16640
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_16 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 190,218
Trainable params: 189,706
Non-trainable params: 512
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/12
60000/60000 [==============================] - 143s 2ms/step - loss: 0.3913 - acc: 0.8772 - val
Epoch 2/12
60000/60000 [==============================] - 144s 2ms/step - loss: 0.1050 - acc: 0.9703 - val
Epoch 3/12
60000/60000 [==============================] - 139s 2ms/step - loss: 0.0771 - acc: 0.9785 - val
Epoch 4/12
60000/60000 [==============================] - 135s 2ms/step - loss: 0.0656 - acc: 0.9815 - val
Epoch 5/12
60000/60000 [==============================] - 127s 2ms/step - loss: 0.0560 - acc: 0.9843 - val
Epoch 6/12
60000/60000 [==============================] - 136s 2ms/step - loss: 0.0495 - acc: 0.9868 - val
Epoch 7/12
60000/60000 [==============================] - 143s 2ms/step - loss: 0.0448 - acc: 0.9879 - val
Epoch 8/12
60000/60000 [==============================] - 132s 2ms/step - loss: 0.0424 - acc: 0.9882 - val
Epoch 9/12
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0379 - acc: 0.9891 - val
Epoch 10/12
60000/60000 [==============================] - 126s 2ms/step - loss: 0.0344 - acc: 0.9902 - val
Epoch 11/12
60000/60000 [==============================] - 135s 2ms/step - loss: 0.0310 - acc: 0.9913 - val
Epoch 12/12
60000/60000 [==============================] - 138s 2ms/step - loss: 0.0306 - acc: 0.9912 - val
```

In [11]:
```
# Evaluating the model
score = model_5.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_5_test = score[1]
model_5_train = max(history_5.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,epochs+1))

# Validation loss
vy = history_5.history['val_loss']
# Training loss
ty = history_5.history['loss']

# Calling the function to draw the plot
plt_dynamic(x, vy, ty)
```

```
Test score: 0.020986396820967274
```

```
Test accuracy: 0.9931
```



1.3 (3). CNN with 7 Convolutional layers and kernel size - (2X2)

In [18]:
```
# Initialising the model
model_7 = Sequential()

# Adding first conv layer
model_7.add(Conv2D(32, kernel_size=(2, 2),padding='same',activation='relu',input_shape

# Adding second conv layer
model_7.add(Conv2D(32, (2, 2), activation='relu'))

# Adding Maxpooling layer
model_7.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))

# Adding Dropout
model_7.add(Dropout(0.3))

# Adding third conv layer
model_7.add(Conv2D(64, (2, 2), activation='relu'))

# Adding Maxpooling layer
model_7.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
```

```
# Adding fourth conv layer
model_7.add(Conv2D(64, (2, 2),padding='same',activation='relu'))

# Adding fifth conv layer
model_7.add(Conv2D(128, (2, 2), activation='relu'))

# Adding Maxpooling layer
model_7.add(MaxPooling2D(pool_size=(3, 3),padding='same'))

# Adding Dropout
model_7.add(Dropout(0.25))

# Adding sixth conv layer
model_7.add(Conv2D(128, (2, 2),padding='same',activation='relu'))

# Adding seventh conv layer
model_7.add(Conv2D(256, (2, 2), activation='relu'))

# Adding Maxpooling layer
model_7.add(MaxPooling2D(pool_size=(2, 2), strides=(1,1)))

# Adding Dropout
model_7.add(Dropout(0.25))

# Adding flatten layer
model_7.add(Flatten())

# Adding first hidden layer
model_7.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))

# Adding Batch Normalization
model_7.add(BatchNormalization())

# Adding Dropout
model_7.add(Dropout(0.5))

# Adding second hidden layer
model_7.add(Dense(128, activation='relu',kernel_initializer=he_normal(seed=None)))

# Adding Dropout
model_7.add(Dropout(0.25))

# Adding output layer
model_7.add(Dense(num_classes, activation='softmax'))

# Printing model Summary
print(model_7.summary())
```

```
        # Compiling the model
        model_7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy

        # Fitting the data to the model
        history_7 = model_7.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=
```

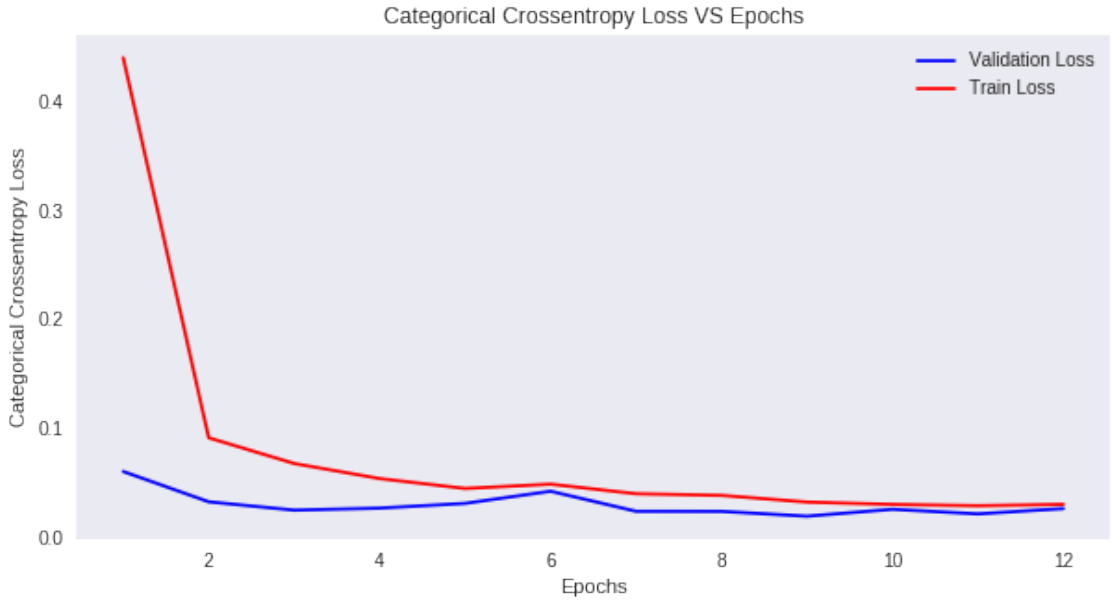
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_60 (Conv2D)           (None, 28, 28, 32)        160
_________________________________________________________________
conv2d_61 (Conv2D)           (None, 27, 27, 32)        4128
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 25, 25, 32)        0
_________________________________________________________________
dropout_42 (Dropout)         (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_62 (Conv2D)           (None, 24, 24, 64)        8256
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_63 (Conv2D)           (None, 12, 12, 64)        16448
_________________________________________________________________
conv2d_64 (Conv2D)           (None, 11, 11, 128)       32896
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
dropout_43 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_65 (Conv2D)           (None, 4, 4, 128)         65664
_________________________________________________________________
conv2d_66 (Conv2D)           (None, 3, 3, 256)         131328
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 2, 2, 256)         0
_________________________________________________________________
dropout_44 (Dropout)         (None, 2, 2, 256)         0
_________________________________________________________________
flatten_8 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_19 (Dense)             (None, 256)               262400
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dropout_45 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_20 (Dense)             (None, 128)               32896
_________________________________________________________________
dropout_46 (Dropout)         (None, 128)               0
```

```
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1290
=================================================================
Total params: 556,490
Trainable params: 555,978
Non-trainable params: 512
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 263s 4ms/step - loss: 0.4402 - acc: 0.8525 - val
Epoch 2/12
60000/60000 [==============================] - 256s 4ms/step - loss: 0.0911 - acc: 0.9727 - val
Epoch 3/12
60000/60000 [==============================] - 262s 4ms/step - loss: 0.0675 - acc: 0.9802 - val
Epoch 4/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.0536 - acc: 0.9839 - val
Epoch 5/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.0446 - acc: 0.9867 - val
Epoch 6/12
60000/60000 [==============================] - 282s 5ms/step - loss: 0.0485 - acc: 0.9858 - val
Epoch 7/12
60000/60000 [==============================] - 287s 5ms/step - loss: 0.0397 - acc: 0.9882 - val
Epoch 8/12
60000/60000 [==============================] - 282s 5ms/step - loss: 0.0382 - acc: 0.9888 - val
Epoch 9/12
60000/60000 [==============================] - 281s 5ms/step - loss: 0.0321 - acc: 0.9902 - val
Epoch 10/12
60000/60000 [==============================] - 285s 5ms/step - loss: 0.0299 - acc: 0.9913 - val
Epoch 11/12
60000/60000 [==============================] - 275s 5ms/step - loss: 0.0288 - acc: 0.9919 - val
Epoch 12/12
60000/60000 [==============================] - 281s 5ms/step - loss: 0.0300 - acc: 0.9914 - val
```

In [19]:
```
# Evaluating the model
score = model_7.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

# Test and train accuracy of the model
model_7_test = score[1]
model_7_train = max(history_7.history['acc'])

# Plotting Train and Test Loss VS no. of epochs
# list of epoch numbers
x = list(range(1,epochs+1))
```

```
        # Validation loss
        vy = history_7.history['val_loss']
        # Training loss
        ty = history_7.history['loss']

        # Calling the function to draw the plot
        plt_dynamic(x, vy, ty)
```

```
Test score: 0.026096460982217105
Test accuracy: 0.9921
```



1.4 CONCLUSION

1.5 (a). Procedure Followed :

1. Load MNIST dataset
2. Split the dataset into train and test
3. Normalize the train and test data
4. Convert class variable into categorical data vector
5. Implement Softmax classifier with 3, 5 and 7 conv layers .
6. Use kernel -size (3X3) , (5X5) and (2,2) .
7. Draw Categorical Crossentropy Loss VS No.of Epochs plot .

1.6 (b) Table (Different models with their train and test accuracies):

```
In [22]: # Installing the library prettytable
         !pip install prettytable
```

```
# Creating table using PrettyTable library
from prettytable import PrettyTable

# Names of models
names = ['CNN(3-Conv layers) With Kernel-size = (3,3)','CNN(5-Conv layers) With Kernel
         'CNN(7-Conv layers) With Kernel-size = (2,2)']

# Training accuracies
train_acc = [model_3_train,model_5_train,model_7_train]

# Test accuracies
test_acc = [model_3_test,model_5_test,model_7_test]

numbering = [1,2,3]

# Initializing prettytable
ptable = PrettyTable()

# Adding columns
ptable.add_column("S.NO.",numbering)
ptable.add_column("MODEL",names)
ptable.add_column("Training Accuracy",train_acc)
ptable.add_column("Test Accuracy",test_acc)

# Printing the Table
print(ptable)
```

```
Requirement already satisfied: prettytable in /usr/local/lib/python3.6/dist-packages (0.7.2)
+-------+-----------------------------------------+--------------------+---------------+
| S.NO. |                  MODEL                  | Training Accuracy  | Test Accuracy |
+-------+-----------------------------------------+--------------------+---------------+
|   1   | CNN(3-Conv layers) With Kernel-size = (3,3) | 0.9941166666348775 |     0.994     |
|   2   | CNN(5-Conv layers) With Kernel-size = (5,5) |       0.9913       |     0.9931    |
|   3   | CNN(7-Conv layers) With Kernel-size = (2,2) |      0.99195       |     0.9921    |
+-------+-----------------------------------------+--------------------+---------------+
```