Exercise : Try various CNN networks on MNIST dataset

Step By Step Process

- Lets Start working on this assignment and as we know the main objective of this assignmenst is to Try various CNN networks on MNIST dataset
- So in this assignment we will try CNN with different Models with different conv layers and with different kernels and in this we will also use dropouts and Batch Normalization.
- Now lets start with Importing some important libraries and after this we will import our MNIST Data set.
- After loading MNIST Dataset with 100000 data points and then we will split our dataset into train and test and we know our dataset and after that we are doing something like

> K.image_data_format() == 'channels_first'

- Let me explain you in brief ->

1. “keras.backend.image_data_format()” : Returns the default image data format convention.The image_data_format parameter affects how each of the backends treat the data dimensions when working with multi-dimensional convolution layers (such as Conv2D, Conv3D, Conv2DTranspose, Copping2D, … and any other 2D or 3D layer). Specifically, it defines where the ‘channels’ dimension is in the input data.
2. And Both TensorFlow and Theano expects a four dimensional tensor as input. But where TensorFlow expects the ‘channels’ dimension as the last dimension (index 3, where the first is index 0) of the tensor – i.e. tensor with shape (samples, rows, cols, channels) – Theano will expect ‘channels’ at the second dimension (index 1) – i.e. tensor with shape (samples, channels, rows, cols). The outputs of the convolutional layers will also follow this pattern.

- And before we move to apply CNN lets try to normalize the data.
- And as we know our class lables in this dataset is in numbers between {0,1,2,3.....9} and now we will convert it into one hot encoded vector
- Now after doing all above lets start with our CNN models in this we will try various CNN networks on MNIST dataset.
- Here we will try conv with various layers and also with different kernel size and here we will also try dropout and Batch norm and In which as a Activation function we will use reLu and as a optimizer we will use adam and most important we will work with softmax classifier with multiple hidden layers that means we will works with softmax with multiple hidden layers and try to observe the performance by changing the no of layers with different kernel size and in this we will also work with dropout and batch normalization.
- And after doing all this we will try to observe the performance of train and test val so that we will be able to know our model should not overfit or underfit.

In [1]:

```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

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()

# it defines where the 'channels' dimension is in the input data
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)
```

```
C:\Users\nisha\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the s
econd argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be
treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

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

In [2]:

```
# utility function
def plt_dynamic_model(x, vy, ty):
    plt.figure(figsize=(10,5))
    plt.plot(x, vy, 'b', label="Val 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.show()
```

1. CNN with 3 conv layers and with (3X3) kernel size

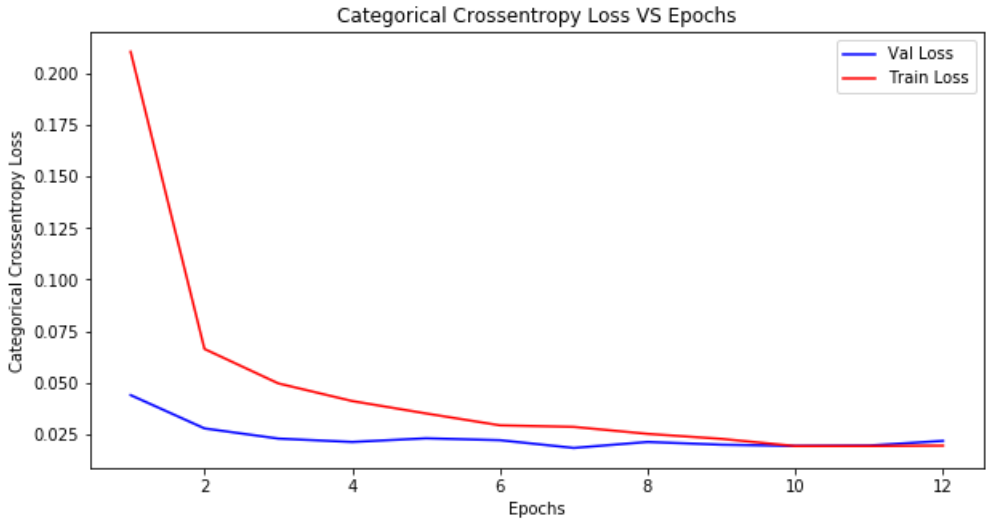
In [3]:

```
model3 = Sequential()

model3.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))
model3.add(Conv2D(128, (3, 3), activation='relu'))
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Flatten())

model3.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))
model3.add(Dropout(0.5))
model3.add(Dense(num_classes, activation='softmax'))

print(model3.summary())
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history3 = model3.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_dat
a=(x_test, y_test))

#--------------------------------------------------------------------plot-------------------------
x = list(range(1,epochs+1))
vy = history3.history['val_loss']
ty = history3.history['loss']
```

```
# function call
plt_dynamic_model(x, vy, ty)

model_score = model3.evaluate(x_test, y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving accuracy of the model
model3_test_acc = model_score[1]
model3_train_acc = max(history3.history['acc'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3200)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               819456
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (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 [==============================] - 318s 5ms/step - loss: 0.2105 - acc: 0.9341 - val_lo
ss: 0.0440 - val_acc: 0.9848
Epoch 2/12
60000/60000 [==============================] - 307s 5ms/step - loss: 0.0663 - acc: 0.9799 - val_lo
ss: 0.0278 - val_acc: 0.9907
Epoch 3/12
60000/60000 [==============================] - 310s 5ms/step - loss: 0.0496 - acc: 0.9850 - val_lo
ss: 0.0229 - val_acc: 0.9925
Epoch 4/12
60000/60000 [==============================] - 316s 5ms/step - loss: 0.0411 - acc: 0.9877 - val_lo
ss: 0.0213 - val_acc: 0.9926
Epoch 5/12
60000/60000 [==============================] - 287s 5ms/step - loss: 0.0351 - acc: 0.9889 - val_lo
ss: 0.0230 - val_acc: 0.9923
Epoch 6/12
60000/60000 [==============================] - 283s 5ms/step - loss: 0.0293 - acc: 0.9909 - val_lo
ss: 0.0221 - val_acc: 0.9928
Epoch 7/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.0286 - acc: 0.9915 - val_lo
ss: 0.0184 - val_acc: 0.9936
Epoch 8/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.0252 - acc: 0.9919 - val_lo
ss: 0.0213 - val_acc: 0.9935
Epoch 9/12
60000/60000 [==============================] - 279s 5ms/step - loss: 0.0228 - acc: 0.9932 - val_lo
ss: 0.0199 - val_acc: 0.9938
Epoch 10/12
60000/60000 [==============================] - 280s 5ms/step - loss: 0.0193 - acc: 0.9943 - val_lo
ss: 0.0193 - val_acc: 0.9945
Epoch 11/12
60000/60000 [==============================] - 294s 5ms/step - loss: 0.0193 - acc: 0.9934 - val_lo
ss: 0.0195 - val_acc: 0.9937
Epoch 12/12
```

```
60000/60000 [==============================] - 293s 5ms/step - loss: 0.0195 - acc: 0.9938 - val_lo
ss: 0.0218 - val_acc: 0.9933
```




```
Test score: 0.02178811355105845
Test accuracy: 0.9933
```

2. CNN with 5 conv layers and with (5X5) kernel size

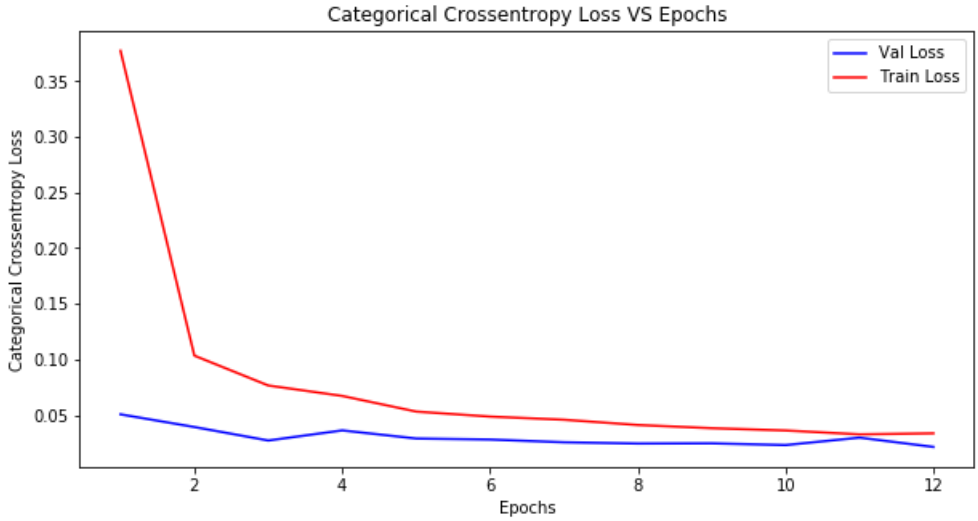
In [4]:

```
model5 = Sequential()

model5.add(Conv2D(8, kernel_size=(5, 5),padding='same',activation='relu',input_shape=input_shape))

model5.add(Conv2D(16, (5, 5), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model5.add(Dropout(0.25))

model5.add(Conv2D(32, (5, 5),padding='same', activation='relu'))
model5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model5.add(Dropout(0.25))

model5.add(Conv2D(64, (5, 5),padding='same',activation='relu'))

model5.add(Conv2D(64, (5, 5), activation='relu'))
model5.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model5.add(Dropout(0.25))

model5.add(Flatten())
model5.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(num_classes, activation='softmax'))
print(model5.summary())
model5.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history5 = model5.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_dat
a=(x_test, y_test))

#--------------------------------------------------------------plot-------------------------
x = list(range(1,epochs+1))
vy = history5.history['val_loss']
ty = history5.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

model_score = model5.evaluate(x_test, y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving accuracy of the model
model5_test_acc = model_score[1]
```

```
model5_train_acc = max(history5.history['acc'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 8)         208
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 24, 24, 16)        3216
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 12, 12, 32)        12832
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 32)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 6, 6, 32)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 6, 6, 64)          51264
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 2, 2, 64)          102464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 1, 1, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               16640
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_7 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (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 [==============================] - 200s 3ms/step - loss: 0.3774 - acc: 0.8789 - val_lo
ss: 0.0506 - val_acc: 0.9845
Epoch 2/12
60000/60000 [==============================] - 209s 3ms/step - loss: 0.1034 - acc: 0.9700 - val_lo
ss: 0.0392 - val_acc: 0.9878
Epoch 3/12
60000/60000 [==============================] - 195s 3ms/step - loss: 0.0765 - acc: 0.9782 - val_lo
ss: 0.0271 - val_acc: 0.9915
Epoch 4/12
60000/60000 [==============================] - 181s 3ms/step - loss: 0.0672 - acc: 0.9815 - val_lo
ss: 0.0362 - val_acc: 0.9887
Epoch 5/12
60000/60000 [==============================] - 176s 3ms/step - loss: 0.0531 - acc: 0.9854 - val_lo
ss: 0.0289 - val_acc: 0.9915
Epoch 6/12
60000/60000 [==============================] - 176s 3ms/step - loss: 0.0486 - acc: 0.9864 - val_lo
ss: 0.0280 - val_acc: 0.9920
Epoch 7/12
60000/60000 [==============================] - 183s 3ms/step - loss: 0.0458 - acc: 0.9871 - val_lo
ss: 0.0255 - val_acc: 0.9925
Epoch 8/12
60000/60000 [==============================] - 193s 3ms/step - loss: 0.0411 - acc: 0.9883 - val_lo
ss: 0.0244 - val_acc: 0.9924
Epoch 9/12
60000/60000 [==============================] - 185s 3ms/step - loss: 0.0382 - acc: 0.9895 - val_lo
ss: 0.0246 - val_acc: 0.9937
Epoch 10/12
60000/60000 [==============================] - 196s 3ms/step - loss: 0.0361 - acc: 0.9896 - val_lo
ss: 0.0230 - val_acc: 0.9935
Epoch 11/12
60000/60000 [==============================] - 211s 4ms/step - loss: 0.0326 - acc: 0.9911 - val_lo
ss: 0.0297 - val_acc: 0.9920
```

```
Epoch 12/12
60000/60000 [==============================] - 197s 3ms/step - loss: 0.0335 - acc: 0.9907 - val_lo
ss: 0.0213 - val_acc: 0.9936
```




```
Test score: 0.021333325604430683
Test accuracy: 0.9936
```

3. CNN with 7 conv layers and with (2X2) kernel size

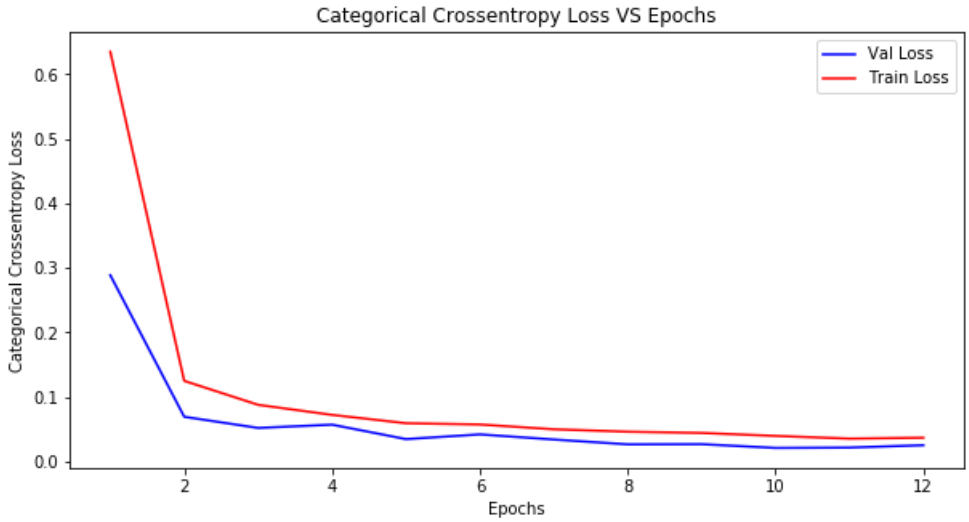
In [5]:

```
model7 = Sequential()

model7.add(Conv2D(32, kernel_size=(2, 2),padding='same',activation='relu',input_shape=input_shape))

model7.add(Conv2D(32, (2, 2), activation='relu'))
model7.add(MaxPooling2D(pool_size=(3, 3), strides=(1,1)))
model7.add(Dropout(0.4))

model7.add(Conv2D(64, (2, 2), activation='relu'))
model7.add(MaxPooling2D(pool_size=(2, 2),padding='same'))
model7.add(Dropout(0.3))

model7.add(Conv2D(64, (2, 2),padding='same',activation='relu'))

model7.add(Conv2D(128, (2, 2), activation='relu'))
model7.add(MaxPooling2D(pool_size=(3, 3),padding='same'))
model7.add(Dropout(0.25))

model7.add(Conv2D(128, (2, 2),padding='same',activation='relu'))

model7.add(Conv2D(256, (2, 2), activation='relu'))
model7.add(MaxPooling2D(pool_size=(2, 2), strides=(1,1)))
model7.add(Dropout(0.35))

model7.add(Flatten())

model7.add(Dense(256, activation='relu',kernel_initializer=he_normal(seed=None)))
model7.add(BatchNormalization())
model7.add(Dropout(0.5))

model7.add(Dense(128, activation='relu',kernel_initializer=he_normal(seed=None)))
model7.add(Dropout(0.25))

model7.add(Dense(num_classes, activation='softmax'))
print(model7.summary())
model7.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history7 = model7.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_dat
a=(x_test, y_test))

#--------------------------------------------------------------------plot-------------------------
x = list(range(1,epochs+1))
vy = history7.history['val_loss']
```

```
ty = history7.history['loss']

# function call
plt_dynamic_model(x, vy, ty)

model_score = model7.evaluate(x_test, y_test, verbose=0)
print('Test score:', model_score[0])
print('Test accuracy:', model_score[1])

# saving accuracy of the model
model7_test_acc = model_score[1]
model7_train_acc = max(history7.history['acc'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 28, 28, 32)        160
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 27, 27, 32)        4128
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 24, 24, 64)        8256
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 12, 12, 64)        16448
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 128)       32896
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 4, 4, 128)         0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 4, 4, 128)         65664
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 3, 3, 256)         131328
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_11 (Dropout)         (None, 2, 2, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               262400
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_7 (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 [==============================] - 419s 7ms/step - loss: 0.6351 - acc: 0.7844 - val_lo
ss: 0.2886 - val_acc: 0.9176
Epoch 2/12
60000/60000 [==============================] - 395s 7ms/step - loss: 0.1247 - acc: 0.9634 - val_lo
ss: 0.0690 - val_acc: 0.9801
Epoch 3/12
60000/60000 [==============================] - 410s 7ms/step - loss: 0.0876 - acc: 0.9749 - val_lo
ss: 0.0517 - val_acc: 0.9838
Epoch 4/12
```

```
60000/60000 [==============================] - 402s 7ms/step - loss: 0.0720 - acc: 0.9792 - val_lo
ss: 0.0568 - val_acc: 0.9848
Epoch 5/12
60000/60000 [==============================] - 426s 7ms/step - loss: 0.0590 - acc: 0.9829 - val_lo
ss: 0.0344 - val_acc: 0.9901
Epoch 6/12
60000/60000 [==============================] - 388s 6ms/step - loss: 0.0571 - acc: 0.9832 - val_lo
ss: 0.0417 - val_acc: 0.9886
Epoch 7/12
60000/60000 [==============================] - 380s 6ms/step - loss: 0.0497 - acc: 0.9855 - val_lo
ss: 0.0338 - val_acc: 0.9913
Epoch 8/12
60000/60000 [==============================] - 376s 6ms/step - loss: 0.0459 - acc: 0.9867 - val_lo
ss: 0.0263 - val_acc: 0.9914
Epoch 9/12
60000/60000 [==============================] - 381s 6ms/step - loss: 0.0440 - acc: 0.9872 - val_lo
ss: 0.0266 - val_acc: 0.9913
Epoch 10/12
60000/60000 [==============================] - 374s 6ms/step - loss: 0.0394 - acc: 0.9880 - val_lo
ss: 0.0207 - val_acc: 0.9927
Epoch 11/12
60000/60000 [==============================] - 385s 6ms/step - loss: 0.0352 - acc: 0.9896 - val_lo
ss: 0.0213 - val_acc: 0.9932
Epoch 12/12
60000/60000 [==============================] - 387s 6ms/step - loss: 0.0366 - acc: 0.9895 - val_lo
ss: 0.0250 - val_acc: 0.9927
```




```
Test score: 0.024968813701119506
Test accuracy: 0.9927
```

Conclusion

In [6]:

```
from prettytable import PrettyTable

print('Performance Table')
x = PrettyTable()
x.field_names =["Models","Train","Test"]

x.add_row(["CNN with 3 conv layers and with (3X3) kernel size",model3_train_acc,model3_test_acc])
x.add_row(["CNN with 5 conv layers and with (5X5) kernel size",model5_train_acc,model5_test_acc])
x.add_row(["CNN with 7 conv layers and with (2X2) kernel size",model7_train_acc,model7_test_acc])

print(x)
```

```
Performance Table
+-------------------------------------------------+--------------------+--------+
|                      Models                     |       Train        |  Test  |
+-------------------------------------------------+--------------------+--------+
| CNN with 3 conv layers and with (3X3) kernel size | 0.9942833333333333 | 0.9933 |
| CNN with 5 conv layers and with (5X5) kernel size | 0.9911166666666666 | 0.9936 |
| CNN with 7 conv layers and with (2X2) kernel size | 0.9895666666666667 | 0.9927 |
```

| CNN with 7 conv layers and with (2x2) kernel size | 0.9059000000000007 | 0.9927 |
+---+--------------------+--------+