In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
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

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()

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)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

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('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Using TensorFlow backend.
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 260s 4ms/step - loss: 0.
2648 - acc: 0.9174 - val_loss: 0.0556 - val_acc: 0.9811
```

```
Epoch 2/12
60000/60000 [==============================] - 265s 4ms/step - loss: 0.
0897 - acc: 0.9741 - val_loss: 0.0389 - val_acc: 0.9865
Epoch 3/12
60000/60000 [==============================] - 262s 4ms/step - loss: 0.
0676 - acc: 0.9798 - val_loss: 0.0331 - val_acc: 0.9882
Epoch 4/12
60000/60000 [==============================] - 266s 4ms/step - loss: 0.
0558 - acc: 0.9838 - val_loss: 0.0307 - val_acc: 0.9896
Epoch 5/12
60000/60000 [==============================] - 4762s 79ms/step - loss:
0.0485 - acc: 0.9852 - val_loss: 0.0331 - val_acc: 0.9892
Epoch 6/12
60000/60000 [==============================] - 260s 4ms/step - loss: 0.
0434 - acc: 0.9868 - val_loss: 0.0274 - val_acc: 0.9905
Epoch 7/12
60000/60000 [==============================] - 260s 4ms/step - loss: 0.
0384 - acc: 0.9883 - val_loss: 0.0273 - val_acc: 0.9915
Epoch 8/12
60000/60000 [==============================] - 261s 4ms/step - loss: 0.
0358 - acc: 0.9892 - val_loss: 0.0259 - val_acc: 0.9923
Epoch 9/12
60000/60000 [==============================] - 256s 4ms/step - loss: 0.
0340 - acc: 0.9894 - val_loss: 0.0273 - val_acc: 0.9911
Epoch 10/12
60000/60000 [==============================] - 259s 4ms/step - loss: 0.
0301 - acc: 0.9912 - val_loss: 0.0255 - val_acc: 0.9920
Epoch 11/12
60000/60000 [==============================] - 275s 5ms/step - loss: 0.
0290 - acc: 0.9911 - val_loss: 0.0258 - val_acc: 0.9915
Epoch 12/12
60000/60000 [==============================] - 274s 5ms/step - loss: 0.
0269 - acc: 0.9917 - val_loss: 0.0285 - val_acc: 0.9908
Test loss: 0.028484841597173318
Test accuracy: 0.9908
```

In [6]:
```
from keras.layers.normalization import BatchNormalization
model_3layers=Sequential()
model_3layers.add(Conv2D(32,kernel_size=(3,3),activation='relu',kernel_
```

```
initializer='he_normal',input_shape=input_shape))
model_3layers.add(Conv2D(64,kernel_size=(3,3),activation='relu',kernel_
initializer='he_normal'))
model_3layers.add(MaxPooling2D(pool_size=(2,2)))
model_3layers.add(Dropout(0.25))
model_3layers.add(Conv2D(64,kernel_size=(3,3),padding="same",activation
='relu',kernel_initializer='he_normal'))
model_3layers.add(MaxPooling2D(pool_size=(2,2)))
model_3layers.add(Dropout(0.25))
model_3layers.add(Flatten())
model_3layers.add(Dense(128,activation='relu'))
model_3layers.add(BatchNormalization())
model_3layers.add(Dropout(0.25))
model_3layers.add(Dense(num_classes,activation='softmax'))

model_3layers.compile(loss=keras.losses.categorical_crossentropy,optimi
zer=keras.optimizers.Adadelta(),metrics=['accuracy'])
h=model_3layers.fit(x_train,y_train,batch_size=batch_size,epochs=epochs
,verbose=1,validation_data=(x_test,y_test))
score=model_3layers.evaluate(x_test,y_test,verbose=0)
print('Test loss :',score[0])
print('test accuracy:',score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 193s 3ms/step - loss: 0.
1813 - acc: 0.9446 - val_loss: 0.0740 - val_acc: 0.9774
Epoch 2/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0571 - acc: 0.9826 - val_loss: 0.0362 - val_acc: 0.9876
Epoch 3/12
60000/60000 [==============================] - 191s 3ms/step - loss: 0.
0450 - acc: 0.9858 - val_loss: 0.0271 - val_acc: 0.9909
Epoch 4/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0352 - acc: 0.9894 - val_loss: 0.0227 - val_acc: 0.9932
Epoch 5/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0314 - acc: 0.9901 - val_loss: 0.0200 - val_acc: 0.9933

Epoch 6/12
```

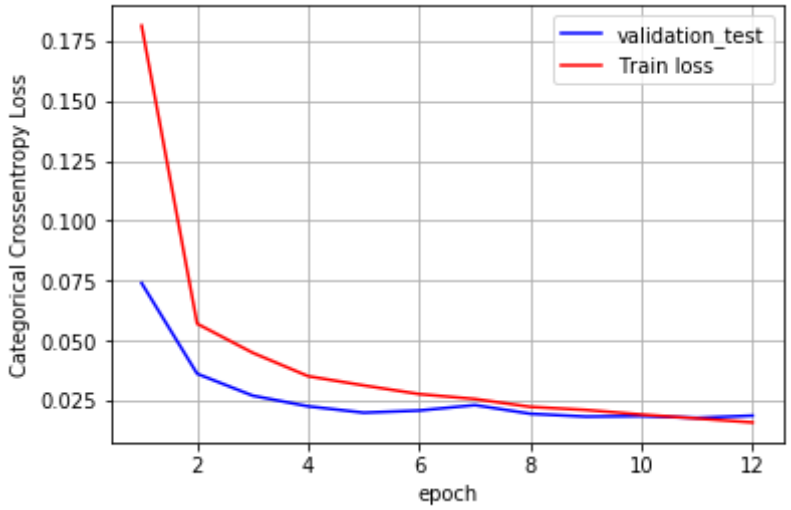
```
Epoch 6/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0278 - acc: 0.9909 - val_loss: 0.0209 - val_acc: 0.9935
Epoch 7/12
60000/60000 [==============================] - 193s 3ms/step - loss: 0.
0257 - acc: 0.9921 - val_loss: 0.0232 - val_acc: 0.9928
Epoch 8/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0224 - acc: 0.9929 - val_loss: 0.0196 - val_acc: 0.9935
Epoch 9/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0212 - acc: 0.9935 - val_loss: 0.0184 - val_acc: 0.9945
Epoch 10/12
60000/60000 [==============================] - 192s 3ms/step - loss: 0.
0192 - acc: 0.9935 - val_loss: 0.0186 - val_acc: 0.9943
Epoch 11/12
60000/60000 [==============================] - 191s 3ms/step - loss: 0.
0176 - acc: 0.9947 - val_loss: 0.0179 - val_acc: 0.9942
Epoch 12/12
60000/60000 [==============================] - 191s 3ms/step - loss: 0.
0159 - acc: 0.9947 - val_loss: 0.0188 - val_acc: 0.9940
Test loss : 0.018790835521311236
test accuracy: 0.994
```

In [0]:
```
def plt_dynamic(X,vy,ty,ax,colors=['b']):
    ax.plot(X,vy,'b',label='validation_test')
    ax.plot(X,ty,'r',label='Train loss')
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [8]:
```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))
vy = h.history['val_loss']
```

```
ty = h.history['loss']
plt_dynamic(x, vy, ty, ax)
```



In [2]:
```
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

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()

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)
```

```
Using TensorFlow backend.
```

```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [4]:

```
from keras.layers.normalization import BatchNormalization
model_5layers=Sequential()
model_5layers.add(Conv2D(32,kernel_size=(5,5),activation='relu',kernel_
initializer='he_normal',input_shape=input_shape))#1st layer
model_5layers.add(Conv2D(32,kernel_size=(5,5),activation='relu',kernel_
initializer='he_normal'))#2nd layer
model_5layers.add(MaxPooling2D(pool_size=(2,2)))
model_5layers.add(Dropout(0.25))
model_5layers.add(Conv2D(64,kernel_size=(5,5),padding="same",activation
='relu',kernel_initializer='he_normal'))#3rd layer
model_5layers.add(Conv2D(128,kernel_size=(5,5),padding="same",activatio
n='relu',kernel_initializer='he_normal'))#4th layer
model_5layers.add(MaxPooling2D(pool_size=(2,2)))
```

```
model_5layers.add(Dropout(0.25))
model_5layers.add(Conv2D(190,kernel_size=(5,5),padding='same',activatio
n='relu',kernel_initializer='he_normal'))#5th layer
model_5layers.add(Dropout(0.20))
model_5layers.add(Flatten())
model_5layers.add(Dense(256,activation='relu'))
model_5layers.add(BatchNormalization())
model_5layers.add(Dropout(0.5))
model_5layers.add(Dense(num_classes,activation='softmax'))

model_5layers.compile(loss=keras.losses.categorical_crossentropy,optimi
zer=keras.optimizers.Adadelta(),metrics=['accuracy'])
h=model_5layers.fit(x_train,y_train,batch_size=batch_size,epochs=epochs
,verbose=1,validation_data=(x_test,y_test))
score=model_5layers.evaluate(x_test,y_test,verbose=0)
print('Test loss :',score[0])
print('test accuracy:',score[1])
```

```
W0622 02:48:15.504623 140558645417856 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/optimizers.py:790: The nam
e tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optim
izer instead.

W0622 02:48:15.738141 140558645417856 deprecation.py:323] From /usr/loc
al/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array
_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 599s 10ms/step - loss:
0.8728 - acc: 0.7164 - val_loss: 0.5907 - val_acc: 0.8227
Epoch 2/12
60000/60000 [==============================] - 596s 10ms/step - loss:
0.4264 - acc: 0.8646 - val_loss: 0.2728 - val_acc: 0.9213
Epoch 3/12
60000/60000 [==============================] - 594s 10ms/step - loss:
0.3008 - acc: 0.9057 - val_loss: 0.2775 - val_acc: 0.9153
Epoch 4/12
```

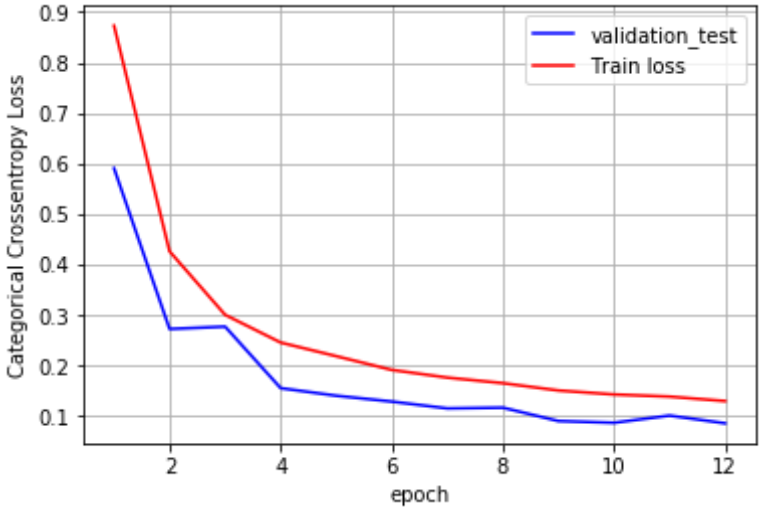
```
Epoch 4/12
60000/60000 [==============================] - 595s 10ms/step - loss:
0.2459 - acc: 0.9217 - val_loss: 0.1558 - val_acc: 0.9493
Epoch 5/12
60000/60000 [==============================] - 595s 10ms/step - loss:
0.2190 - acc: 0.9315 - val_loss: 0.1408 - val_acc: 0.9535
Epoch 6/12
60000/60000 [==============================] - 594s 10ms/step - loss:
0.1918 - acc: 0.9388 - val_loss: 0.1292 - val_acc: 0.9601
Epoch 7/12
60000/60000 [==============================] - 592s 10ms/step - loss:
0.1766 - acc: 0.9442 - val_loss: 0.1157 - val_acc: 0.9640
Epoch 8/12
60000/60000 [==============================] - 592s 10ms/step - loss:
0.1656 - acc: 0.9482 - val_loss: 0.1172 - val_acc: 0.9610
Epoch 9/12
60000/60000 [==============================] - 592s 10ms/step - loss:
0.1511 - acc: 0.9527 - val_loss: 0.0907 - val_acc: 0.9708
Epoch 10/12
60000/60000 [==============================] - 593s 10ms/step - loss:
0.1433 - acc: 0.9545 - val_loss: 0.0873 - val_acc: 0.9714
Epoch 11/12
60000/60000 [==============================] - 595s 10ms/step - loss:
0.1390 - acc: 0.9564 - val_loss: 0.1015 - val_acc: 0.9664
Epoch 12/12
60000/60000 [==============================] - 597s 10ms/step - loss:
0.1302 - acc: 0.9589 - val_loss: 0.0862 - val_acc: 0.9703
Test loss : 0.08624118152540178
test accuracy: 0.9703
```

In [0]:
```
def plt_dynamic(X,vy,ty,ax,colors=['b']):
    ax.plot(X,vy,'b',label='validation_test')
    ax.plot(X,ty,'r',label='Train loss')
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

In [8]:
```
import matplotlib.pyplot as plt
fig,ax = plt.subplots(1,1)
```

```
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))
vy = h.history['val_loss']
ty = h.history['loss']
plt_dynamic(x, vy, ty, ax)
```



CONCLUSION

TABLE WITH CNN OF DIFFERENT LAYERS

In [4]:
```
print("Convolution neural network ")
from tabulate import tabulate
print(tabulate ([['CNN(2LAYERS)', 0.028, 99.08],['CNN(3LAYERS)',0.018,9
9.48],['CNN(5LAYERS)',0.086,97.03] , ['CNN(7LAYERS)WITH 5 EPOCHS ',0.06
7,97.03]],    headers=['LAYERS', 'TEST LOSS','TEST ACCURACY']))
```

```
Convolution neural network
```

LAYERS	TEST LOSS	TEST ACCURACY
CNN(2LAYERS)	0.028	99.08
CNN(3LAYERS)	0.018	99.48
CNN(5LAYERS)	0.086	97.03
CNN(7LAYERS)WITH 5 EPOCHS	0.067	97.03

3rd layer

1. IN CNN 3 layer neural network
2. we have different architecture of 32-64-64.
3. with kernel size of (3,3)
4. we are using maxpooling,batch normalization and regularization as dropout.
5. The the test loss is less and test accuracy is also high .

5th layer

1. IN CNN 5 layer neural network
2. we have different architecture of 32-32-64-128-190 .
3. with kernel size of (5,5)
4. we are using maxpooling,batch normalization and regularization as dropout.
5. the test loss is decresing for every eochs and test accuracy is good .
6. but as compared to 3rd layer and 2nd layer the accuracy is less and test loss is more.