# 7 layers (32-42-58-64-64-128-152) CNN

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 = 5

# 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.
```

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

In [2]:

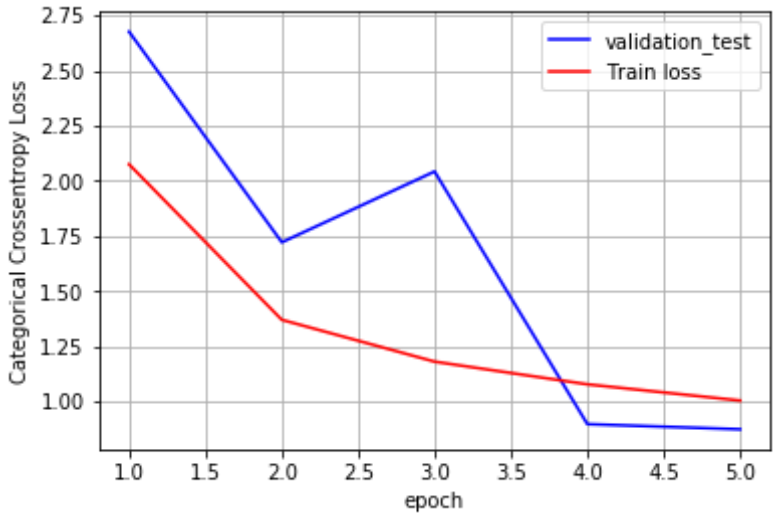
```
from keras.layers.normalization import BatchNormalization
from keras.layers.normalization import BatchNormalization
model_7layers=Sequential()
model_7layers.add(Conv2D(32,kernel_size=(7,7),activation='relu',kernel_
initializer='he_normal',input_shape=input_shape))#1st layer
model_7layers.add(Conv2D(42,kernel_size=(7,7),activation='relu',kernel_
initializer='he_normal'))#2nd layer
model_7layers.add(MaxPooling2D(pool_size=(2,2)))
model_7layers.add(Dropout(0.25))
model_7layers.add(Conv2D(58,kernel_size=(7,7),padding="same",activation
='relu',kernel_initializer='he_normal'))#3rd layer
model_7layers.add(Conv2D(64,kernel_size=(7,7),padding="same",activation
='relu',kernel_initializer='he_normal'))#4th layer
model_7layers.add(MaxPooling2D(pool_size=(2,2)))
model_7layers.add(Dropout(0.25))
model_7layers.add(Conv2D(64,kernel_size=(7,7),padding='same',activation
='relu',kernel_initializer='he_normal'))#5th layer
model_7layers.add(Conv2D(128,kernel_size=(7,7),padding='same',activatio
n='relu',kernel_initializer='he_normal'))#6th layer
model_7layers.add(Conv2D(152,kernel_size=(7,7),padding='same',activatio
n='relu',kernel_initializer='he_normal'))
model_7layers.add(MaxPooling2D(pool_size=(2,2)))#7th layer
model_7layers.add(Dropout(0.20))
model_7layers.add(Flatten())
```

```
model_7layers.add(Dense(212,activation='relu'))
model_7layers.add(BatchNormalization())
model_7layers.add(Dropout(0.5))
model_7layers.add(Dense(num_classes,activation='softmax'))

model_7layers.compile(loss=keras.losses.categorical_crossentropy,optimi
zer=keras.optimizers.Adadelta(),metrics=['accuracy'])
h=model_7layers.fit(x_train,y_train,batch_size=batch_size,epochs=epochs
,verbose=1,validation_data=(x_test,y_test))
score=model_7layers.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 1/5
60000/60000 [==============================] - 3523s 59ms/step - loss:
1.9028 - acc: 0.3082 - val_loss: 4.7795 - val_acc: 0.1216
Epoch 2/5
60000/60000 [==============================] - 19483s 325ms/step - los
s: 0.5265 - acc: 0.8292 - val_loss: 0.2505 - val_acc: 0.9205
Epoch 3/5
60000/60000 [==============================] - 1273s 21ms/step - loss:
0.1724 - acc: 0.9499 - val_loss: 0.0903 - val_acc: 0.9748
Epoch 4/5
60000/60000 [==============================] - 1314s 22ms/step - loss:
0.1097 - acc: 0.9684 - val_loss: 0.0752 - val_acc: 0.9779
Epoch 5/5
60000/60000 [==============================] - 2619s 44ms/step - loss:
0.0788 - acc: 0.9768 - val_loss: 0.0680 - val_acc: 0.9818
Test loss : 0.06796135781658813
test accuracy: 0.9818
```

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 [4]:
```
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 [1]:
```
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
```

1. IN CNN 7layer neural network
2. we have different architecture of 32-42-58-64-128-152 .
3. with kernel size of (7,7)
4. we are using maxpooling,batch normalization and regularization as dropout.
5. at first epochs the the test loss is more and test accuracy is 12 percent.
6. but at second epochs the weights are learnt iits give more efficient huge change in accracy suddenly rise to 92 percent accuracy.
7. but at end of 5 epochs the accuracy and test loss is improved alot.
8. with kernel size and layers increase the test loss and accuracy increse.