CNN Assignment : Apply 3 different CNN's on the MNIST dataset

In [1]:

```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py
#Refer this link for making better CNN networks
#https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architecturespart-ii-hyper-parameter-42efca01e
import warnings
warnings.filterwarnings("ignore")
#from __future__ import print_function
exec('from __future__ import absolute_import, division, 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
# Preparing trainining and testing data
# 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()
#print(x_train.shape)
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 [==============================] - 3s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

In [2]:

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time
# https://gist.github.com/greydanus/f6eee59eaf1d90fcb3b534a25362cea4
# https://stackoverflow.com/a/14434334 # this function is used to update the plots for each epoch and error
def plt_dynamic(x, vy, ty, ax, colors=['b']):
  ax.plot(x, vy, 'b', label="Validation Loss")
  ax.plot(x, ty, 'r', label="Train Loss")
  plt.legend()
  plt.grid()
  fig.canvas.draw()
```

Models with Conv , Max Pool and Dense Layer

Model 1 : 2 conv + 2 maxpoll+ 3 dense layers

In [3]:

```
import warnings
warnings.filterwarnings("ignore")
# In this (First Model) lets follow the general structure of the lenet we will make a simple model
# Network Architecture
# input -> conv -> polling -> conv -> polling -> FC -> FC -> output
# 8 16 120 84 10
model = Sequential()
model.add(Conv2D(8, kernel_size=(3, 3),activation='relu',padding='same',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(16, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.adam(),
              metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
WARNING:tensorflow:From C:\Anaconda\lib\site-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with (from tens
orflow.python.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 8)         80        
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 8)         0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 16)        3216      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_2 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850       
=================================================================
Total params: 62,430
Trainable params: 62,430
Non-trainable params: 0
_________________________________________________________________
```

In [4]:

```
import warnings
warnings.filterwarnings("ignore")
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('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
WARNING:tensorflow:From C:\Anaconda\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.op
s.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 17s 287us/step - loss: 0.2721 - accuracy: 0.9219 - val_loss: 0.0698 - val_accuracy:
0.9765
Epoch 2/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0740 - accuracy: 0.9771 - val_loss: 0.0515 - val_accuracy:
0.9838
Epoch 3/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0528 - accuracy: 0.9838 - val_loss: 0.0404 - val_accuracy:
0.9859
Epoch 4/12
60000/60000 [==============================] - 18s 306us/step - loss: 0.0425 - accuracy: 0.9868 - val_loss: 0.0338 - val_accuracy:
0.9889
Epoch 5/12
60000/60000 [==============================] - 20s 332us/step - loss: 0.0353 - accuracy: 0.9890 - val_loss: 0.0345 - val_accuracy:
0.9885
Epoch 6/12
60000/60000 [==============================] - 18s 300us/step - loss: 0.0294 - accuracy: 0.9903 - val_loss: 0.0421 - val_accuracy:
0.9864
Epoch 7/12
60000/60000 [==============================] - 18s 293us/step - loss: 0.0252 - accuracy: 0.9920 - val_loss: 0.0321 - val_accuracy:
0.9888
Epoch 8/12
60000/60000 [==============================] - 18s 296us/step - loss: 0.0205 - accuracy: 0.9931 - val_loss: 0.0373 - val_accuracy:
0.9887
Epoch 9/12
60000/60000 [==============================] - 17s 289us/step - loss: 0.0184 - accuracy: 0.9937 - val_loss: 0.0352 - val_accuracy:
0.9888
Epoch 10/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.0172 - accuracy: 0.9942 - val_loss: 0.0415 - val_accuracy:
0.9859
Epoch 11/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0136 - accuracy: 0.9957 - val_loss: 0.0342 - val_accuracy:
0.9892
Epoch 12/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0132 - accuracy: 0.9956 - val_loss: 0.0464 - val_accuracy:
0.9871
Test loss: 0.046447735224399364
Test accuracy: 0.9871000051498413
```

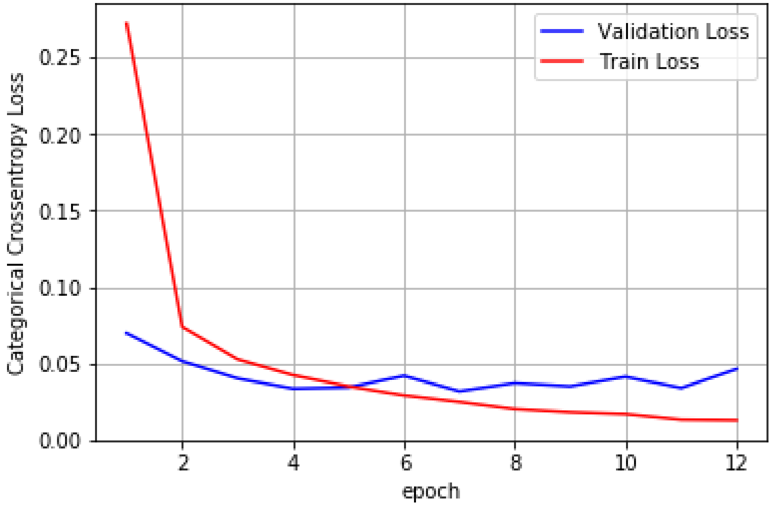
In [5]:

```
score = model.evaluate(x_train, y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************* **********************\n')
#test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
# plot
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch');
ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,12+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.015934466881301449
Train accuracy: 99.4533360004425

************************* **********************

Test score: 0.046447735224399364
Test accuracy: 98.71000051498413
```



Model 2 : 3 conv + 3 maxpoll+ 2 dense layers

In [6]:

```
import warnings
warnings.filterwarnings("ignore")
# go basic model to deep layer model
# Network Architecture
# input -> conv -> polling -> conv -> polling -> conv -> polling -> FC -> output
# 8 32 128 64
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.adam(),
metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 128)         73856     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 128)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650       
=================================================================
Total params: 101,578
Trainable params: 101,578
```

```
Non-trainable params: 0
_________________________________________________________________
```

In [7]:

```
import warnings
warnings.filterwarnings("ignore")
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('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 48s 796us/step - loss: 0.3081 - accuracy: 0.9097 - val_loss: 0.0952 - val_accuracy:
0.9729
Epoch 2/12
60000/60000 [==============================] - 46s 766us/step - loss: 0.0917 - accuracy: 0.9721 - val_loss: 0.0805 - val_accuracy:
0.9736
Epoch 3/12
60000/60000 [==============================] - 45s 758us/step - loss: 0.0675 - accuracy: 0.9797 - val_loss: 0.0624 - val_accuracy:
0.9813
Epoch 4/12
60000/60000 [==============================] - 45s 750us/step - loss: 0.0539 - accuracy: 0.9836 - val_loss: 0.0514 - val_accuracy:
0.9841
Epoch 5/12
60000/60000 [==============================] - 45s 757us/step - loss: 0.0433 - accuracy: 0.9865 - val_loss: 0.0573 - val_accuracy:
0.9827
Epoch 6/12
60000/60000 [==============================] - 45s 757us/step - loss: 0.0359 - accuracy: 0.9889 - val_loss: 0.0682 - val_accuracy:
0.9794
Epoch 7/12
60000/60000 [==============================] - 45s 755us/step - loss: 0.0318 - accuracy: 0.9900 - val_loss: 0.0545 - val_accuracy:
0.9834
Epoch 8/12
60000/60000 [==============================] - 46s 764us/step - loss: 0.0269 - accuracy: 0.9910 - val_loss: 0.0499 - val_accuracy:
0.9866
Epoch 9/12
60000/60000 [==============================] - 46s 761us/step - loss: 0.0228 - accuracy: 0.9928 - val_loss: 0.0533 - val_accuracy:
0.9857
Epoch 10/12
60000/60000 [==============================] - 46s 759us/step - loss: 0.0197 - accuracy: 0.9937 - val_loss: 0.0485 - val_accuracy:
0.9868
Epoch 11/12
60000/60000 [==============================] - 49s 823us/step - loss: 0.0169 - accuracy: 0.9944 - val_loss: 0.0503 - val_accuracy:
```

```
0.9874
Epoch 12/12
60000/60000 [==============================] - 48s 793us/step - loss: 0.0150 - accuracy: 0.9950 - val_loss: 0.0501 - val_accuracy:
0.9851
Test loss: 0.05009871911372902
Test accuracy: 0.9850999712944031
```

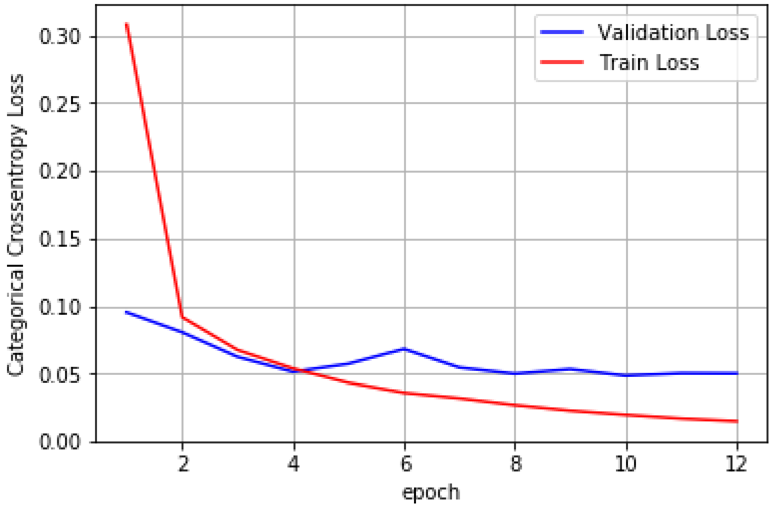
In [8]:

```
score = model.evaluate(x_train, y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************ *********************\n')
#test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
# plot
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch');
ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,12+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.011942215900942877
Train accuracy: 99.6150016784668

************************ *********************

Test score: 0.05009871911372902
Test accuracy: 98.50999712944031
```



Finally we train a model with the trend Conv-Conv-Pool-Conv-Conv-Pool

Model 3 : 4 conv+ 2 maxpoll + 2 dense Layer

In [9]:

```
# go basic model to deep layer model
# Network Architecture
# input -> conv -> conv -> polling -> conv -> conv -> polling -> FC -> output
# 16 16 32 32 512
model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3),activation='relu',padding='same',input_shape=input_shape))
model.add(Conv2D(16,(3, 3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.adam(),
metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
Model: "sequential_3"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 16)        160       
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 16)        2320      
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 12, 12, 32)        4640      
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 32)        9248      
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 5, 5, 32)          0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 800)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 512)               410112    
_________________________________________________________________
dense_7 (Dense)              (None, 10)                5130      
=================================================================
Total params: 431,610
Trainable params: 431,610
Non-trainable params: 0
_________________________________________________________________
```

In [10]:

```python
import warnings
warnings.filterwarnings("ignore")
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('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 83s 1ms/step - loss: 0.1849 - accuracy: 0.9430 - val_loss: 0.0541 - val_accuracy:
0.9824
Epoch 2/12
60000/60000 [==============================] - 85s 1ms/step - loss: 0.0485 - accuracy: 0.9848 - val_loss: 0.0321 - val_accuracy:
0.9892
Epoch 3/12
60000/60000 [==============================] - 83s 1ms/step - loss: 0.0321 - accuracy: 0.9897 - val_loss: 0.0336 - val_accuracy:
0.9899
```

```
Epoch 4/12
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0236 - accuracy: 0.9922 - val_loss: 0.0330 - val_accuracy:
0.9887
Epoch 5/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0188 - accuracy: 0.9942 - val_loss: 0.0259 - val_accuracy:
0.9913
Epoch 6/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0151 - accuracy: 0.9948 - val_loss: 0.0294 - val_accuracy:
0.9916
Epoch 7/12
60000/60000 [==============================] - 84s 1ms/step - loss: 0.0123 - accuracy: 0.9959 - val_loss: 0.0371 - val_accuracy:
0.9890
Epoch 8/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.0106 - accuracy: 0.9963 - val_loss: 0.0303 - val_accuracy:
0.9907
Epoch 9/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0084 - accuracy: 0.9974 - val_loss: 0.0289 - val_accuracy:
0.9927
Epoch 10/12
60000/60000 [==============================] - 90s 2ms/step - loss: 0.0099 - accuracy: 0.9968 - val_loss: 0.0282 - val_accuracy:
0.9929
Epoch 11/12
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0089 - accuracy: 0.9970 - val_loss: 0.0301 - val_accuracy:
0.9917
Epoch 12/12
60000/60000 [==============================] - 86s 1ms/step - loss: 0.0052 - accuracy: 0.9984 - val_loss: 0.0326 - val_accuracy:
0.9930
Test loss: 0.032560050917742964
Test accuracy: 0.9929999709129333
```

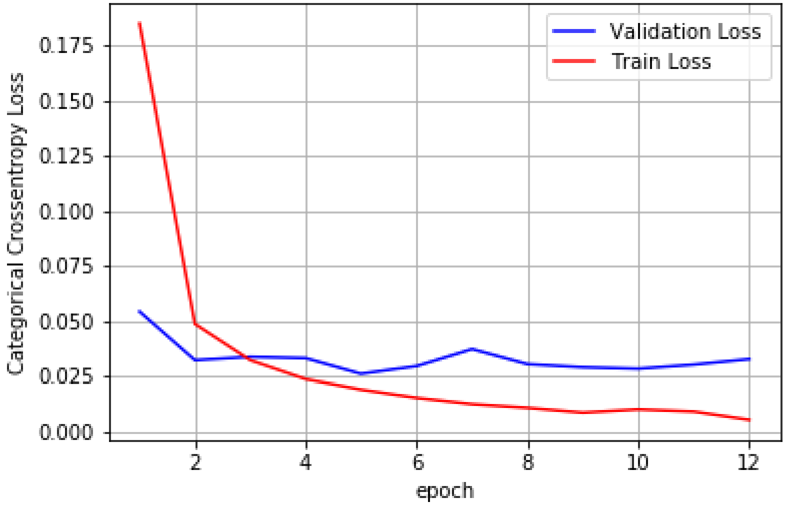
In [11]:

```
score = model.evaluate(x_train, y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************ *********************\n')
#test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
# plot
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch');
ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,12+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.003995988390061666
Train accuracy: 99.85666871070862

************************ *********************

Test score: 0.03256005091774296
Test accuracy: 99.29999709129333
```



Models included Dropout

Model 1 : 2 conv + 2 maxpoll+ 3 dense layer +Dropout (0.5)

In [12]:

```
#Same models with Dropouts
import warnings
warnings.filterwarnings("ignore")
# In this (First Model) lets follow the general structure of the lenet we will make a simple model
# Network Architecture
# input -> conv -> polling -> conv -> polling ->droupout-> FC -> FC -> output
# 8 16 120 84 10
model = Sequential()
model.add(Conv2D(8, kernel_size=(3, 3),activation='relu',padding='same',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(16, (5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.adam(),
metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_10 (Conv2D)           (None, 28, 28, 8)         80        
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 14, 14, 8)         0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 16)        3216      
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 5, 5, 16)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 16)          0         
_________________________________________________________________
flatten_4 (Flatten)          (None, 400)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 120)               48120     
_________________________________________________________________
dense_9 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_10 (Dense)             (None, 10)                850       
=================================================================
Total params: 62,430
Trainable params: 62,430
Non-trainable params: 0
_________________________________________________________________
```

In [13]:

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

```
Train on 60000 samples, validate on 10000 samples
```

```
Epoch 1/12
60000/60000 [==============================] - 17s 276us/step - loss: 0.3910 - accuracy: 0.8756 - val_loss: 0.0814 - val_accuracy:
0.9738
Epoch 2/12
60000/60000 [==============================] - 18s 305us/step - loss: 0.1317 - accuracy: 0.9588 - val_loss: 0.0547 - val_accuracy:
0.9819
Epoch 3/12
60000/60000 [==============================] - 18s 303us/step - loss: 0.1035 - accuracy: 0.9679 - val_loss: 0.0398 - val_accuracy:
0.9868
Epoch 4/12
60000/60000 [==============================] - 18s 296us/step - loss: 0.0866 - accuracy: 0.9733 - val_loss: 0.0354 - val_accuracy:
0.9877
Epoch 5/12
60000/60000 [==============================] - 18s 296us/step - loss: 0.0780 - accuracy: 0.9757 - val_loss: 0.0317 - val_accuracy:
0.9898
Epoch 6/12
60000/60000 [==============================] - 18s 295us/step - loss: 0.0688 - accuracy: 0.9787 - val_loss: 0.0368 - val_accuracy:
0.9881
Epoch 7/12
60000/60000 [==============================] - 18s 299us/step - loss: 0.0631 - accuracy: 0.9796 - val_loss: 0.0324 - val_accuracy:
0.9896
Epoch 8/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0587 - accuracy: 0.9812 - val_loss: 0.0268 - val_accuracy:
0.9916
Epoch 9/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0553 - accuracy: 0.9825 - val_loss: 0.0283 - val_accuracy:
0.9910
Epoch 10/12
60000/60000 [==============================] - 18s 300us/step - loss: 0.0523 - accuracy: 0.9834 - val_loss: 0.0272 - val_accuracy:
0.9910
Epoch 11/12
60000/60000 [==============================] - 18s 298us/step - loss: 0.0515 - accuracy: 0.9837 - val_loss: 0.0269 - val_accuracy:
0.9906
Epoch 12/12
60000/60000 [==============================] - 18s 300us/step - loss: 0.0474 - accuracy: 0.9846 - val_loss: 0.0236 - val_accuracy:
0.9912
Test loss: 0.02355184760145494
Test accuracy: 0.9911999702453613
```

In [14]:

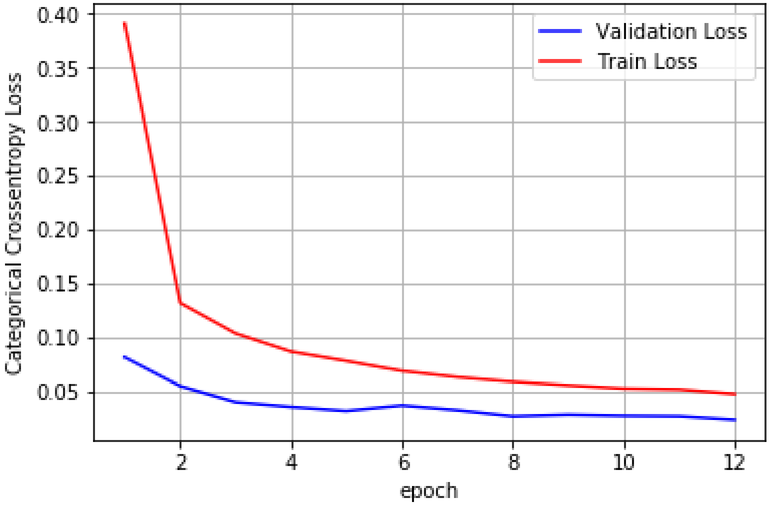
```
score = model.evaluate(x_train, y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************ *********************\n')
#test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
```

```
# plot
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch');
ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,12+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.017866180427525736
Train accuracy: 99.43666458129883

************************ *********************

Test score: 0.02355184760145494
Test accuracy: 99.11999702453613
```



Model 2 : 3 conv + 3 maxpoll+ 2 dense layers + Dropout (0.9)

In [15]:

```
import warnings
warnings.filterwarnings("ignore")
# go basic model to deep layer model
# Network Architecture
# input -> conv -> polling -> conv -> polling -> conv -> polling ->dropout-> FC -> output
# 8 32 128 64
model = Sequential()
```

```
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Dropout(0.9))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.adam(),
metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 32)        320       
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 32)        0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 11, 11, 64)        18496     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 64)          0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 3, 3, 128)         73856     
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 1, 1, 128)         0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 1, 128)         0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 128)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 64)                8256      
_________________________________________________________________
dense_12 (Dense)             (None, 10)                650       
=================================================================
Total params: 101,578
Trainable params: 101,578
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 44s 741us/step - loss: 1.3356 - accuracy: 0.5181 - val_loss: 0.2980 - val_accuracy:
0.9365
Epoch 2/12
60000/60000 [==============================] - 45s 747us/step - loss: 0.8161 - accuracy: 0.7112 - val_loss: 0.1750 - val_accuracy:
0.9537
Epoch 3/12
60000/60000 [==============================] - 45s 752us/step - loss: 0.6775 - accuracy: 0.7623 - val_loss: 0.1483 - val_accuracy:
0.9616
Epoch 4/12
60000/60000 [==============================] - 45s 751us/step - loss: 0.6011 - accuracy: 0.7896 - val_loss: 0.1250 - val_accuracy:
0.9640
Epoch 5/12
60000/60000 [==============================] - 45s 748us/step - loss: 0.5565 - accuracy: 0.8058 - val_loss: 0.1123 - val_accuracy:
0.9697
Epoch 6/12
60000/60000 [==============================] - 46s 761us/step - loss: 0.5184 - accuracy: 0.8206 - val_loss: 0.1050 - val_accuracy:
0.9705
Epoch 7/12
60000/60000 [==============================] - 46s 760us/step - loss: 0.4900 - accuracy: 0.8301 - val_loss: 0.1024 - val_accuracy:
0.9713
Epoch 8/12
60000/60000 [==============================] - 45s 753us/step - loss: 0.4653 - accuracy: 0.8412 - val_loss: 0.1005 - val_accuracy:
0.9722
Epoch 9/12
60000/60000 [==============================] - 46s 762us/step - loss: 0.4491 - accuracy: 0.8457 - val_loss: 0.1117 - val_accuracy:
0.9685
Epoch 10/12
60000/60000 [==============================] - 45s 755us/step - loss: 0.4229 - accuracy: 0.8566 - val_loss: 0.1161 - val_accuracy:
0.9675
Epoch 11/12
60000/60000 [==============================] - 45s 754us/step - loss: 0.4140 - accuracy: 0.8597 - val_loss: 0.0912 - val_accuracy:
0.9745s - los
Epoch 12/12
60000/60000 [==============================] - 45s 754us/step - loss: 0.3991 - accuracy: 0.8645 - val_loss: 0.0948 - val_accuracy:
0.9748
Test loss: 0.09479323272332549
Test accuracy: 0.9747999906539917
```

In [17]:

```
keras.layers.BatchNormalization(axis=-1, momentum=0.99, epsilon=0.001, center=True, scale=True, beta_initializer='zeros', gamma_in
```

Out[17]:

```
<keras.layers.normalization.BatchNormalization at 0x1d744fe5a90>
```

Model 3 : 4 conv + 2 maxpoll+ 2 dense layers + Dropout (0.3)

In [18]:

```
# go basic model to deep layer model
# Network Architecture
# input -> conv -> conv -> polling -> conv -> conv -> polling ->dropout-> FC -> output
# 16 16 32 32 512
model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3),activation='relu',padding='same',input_shape=input_shape))
model.add(Conv2D(16,(3, 3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))# for the location invariants
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer=keras.optimizers.adam(),
metrics=['accuracy'])
# this will train the model and validate the model in this fit function
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_15 (Conv2D)           (None, 28, 28, 16)        160       
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 28, 28, 16)        2320      
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 12, 12, 32)        4640      
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 10, 10, 32)        9248      
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 5, 5, 32)          0         
_________________________________________________________________
```

```
dropout_3 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_13 (Dense)             (None, 512)               410112
_________________________________________________________________
dense_14 (Dense)             (None, 10)                5130
=================================================================
Total params: 431,610
Trainable params: 431,610
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 81s 1ms/step - loss: 0.2103 - accuracy: 0.9357 - val_loss: 0.0499 - val_accuracy:
0.9849
Epoch 2/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0618 - accuracy: 0.9807 - val_loss: 0.0332 - val_accuracy:
0.9881
Epoch 3/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0422 - accuracy: 0.9871 - val_loss: 0.0282 - val_accuracy:
0.9905
Epoch 4/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0350 - accuracy: 0.9891 - val_loss: 0.0250 - val_accuracy:
0.9914
Epoch 5/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0289 - accuracy: 0.9907 - val_loss: 0.0217 - val_accuracy:
0.9921
Epoch 6/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0241 - accuracy: 0.9919 - val_loss: 0.0277 - val_accuracy:
0.9923
Epoch 7/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0217 - accuracy: 0.9929 - val_loss: 0.0242 - val_accuracy:
0.9928
Epoch 8/12
60000/60000 [==============================] - 79s 1ms/step - loss: 0.0189 - accuracy: 0.9938 - val_loss: 0.0210 - val_accuracy:
0.9929
```

```
Epoch 9/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0166 - accuracy: 0.9948 - val_loss: 0.0256 - val_accuracy:
0.9926
Epoch 10/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0152 - accuracy: 0.9949 - val_loss: 0.0268 - val_accuracy:
0.9918
Epoch 11/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0136 - accuracy: 0.9954 - val_loss: 0.0235 - val_accuracy:
0.9924
Epoch 12/12
60000/60000 [==============================] - 80s 1ms/step - loss: 0.0124 - accuracy: 0.9960 - val_loss: 0.0179 - val_accuracy:
0.9943
Test loss: 0.017945763700317457
Test accuracy: 0.9943000078201294
```

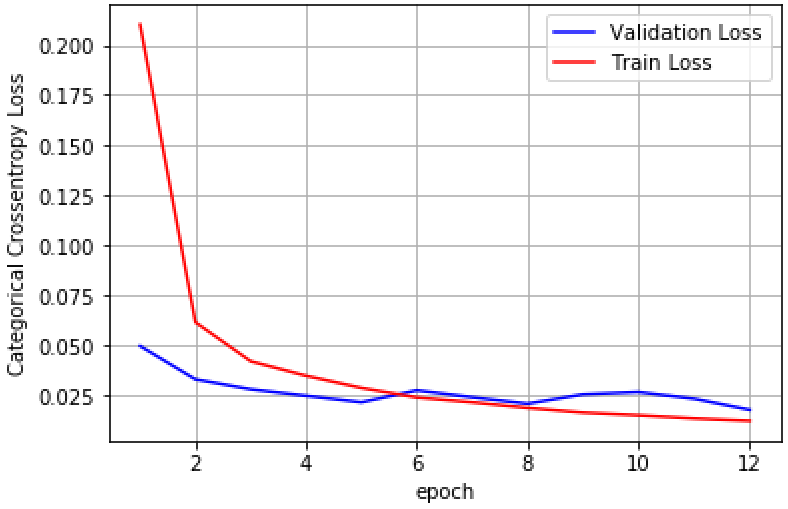
In [20]:

```
score = model.evaluate(x_train, y_train, verbose=0)
print('Train score:', score[0])
print('Train accuracy:', score[1]*100)
print('\n************************ *********************\n')
#test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1]*100)
# plot
fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch');
ax.set_ylabel('Categorical Crossentropy Loss')
x = list(range(1,12+1))
vy = history.history['val_loss']
ty = history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Train score: 0.0033918453480264966
Train accuracy: 99.91166591644287

************************ *********************

Test score: 0.017945763700317457
Test accuracy: 99.43000078201294
```



CONCLUSION:

In [21]:
```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= ("conv_layers", "MAxPoll_layers", "Dense_layers","Dropout","Accuracy")
tb.add_row(["2", "2","3","NO",98.71])
tb.add_row(["3", "3","2","NO",98.51])
tb.add_row(["4", "2","2","NO",99.29])
tb.add_row(["2", "2","3","0.5",99.12])
tb.add_row(["3", "3","2","0.9",97.47])
tb.add_row(["4", "2","2","0.3",99.43])

print(tb.get_string(titles = "CNN Models - Observations"))
```

```
+-------------+----------------+--------------+---------+----------+
| conv_layers | MAxPoll_layers | Dense_layers | Dropout | Accuracy |
+-------------+----------------+--------------+---------+----------+
|      2      |       2        |      3       |    NO   |  98.71   |
|      3      |       3        |      2       |    NO   |  98.51   |
|      4      |       2        |      2       |    NO   |  99.29   |
|      2      |       2        |      3       |   0.5   |  99.12   |
|      3      |       3        |      2       |   0.9   |  97.47   |
|      4      |       2        |      2       |   0.3   |  99.43   |
+-------------+----------------+--------------+---------+----------+
```

All the 3 different architectures performed good with accuracy of 98 % Plus and it is also observed that regularizers like drop out resulted in

accuracy of 99 % plus