CNN_MNIST

July 5, 2019

In [2]:
```
# 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.layers.normalization import BatchNormalization
from keras.layers.advanced_activations import LeakyReLU


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)
```

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

In [0]:
```
%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()
    plt.show()
```

1 Reference

https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7

2 Using 3*3 ConvNet, Optimizer 'AdaDelta'

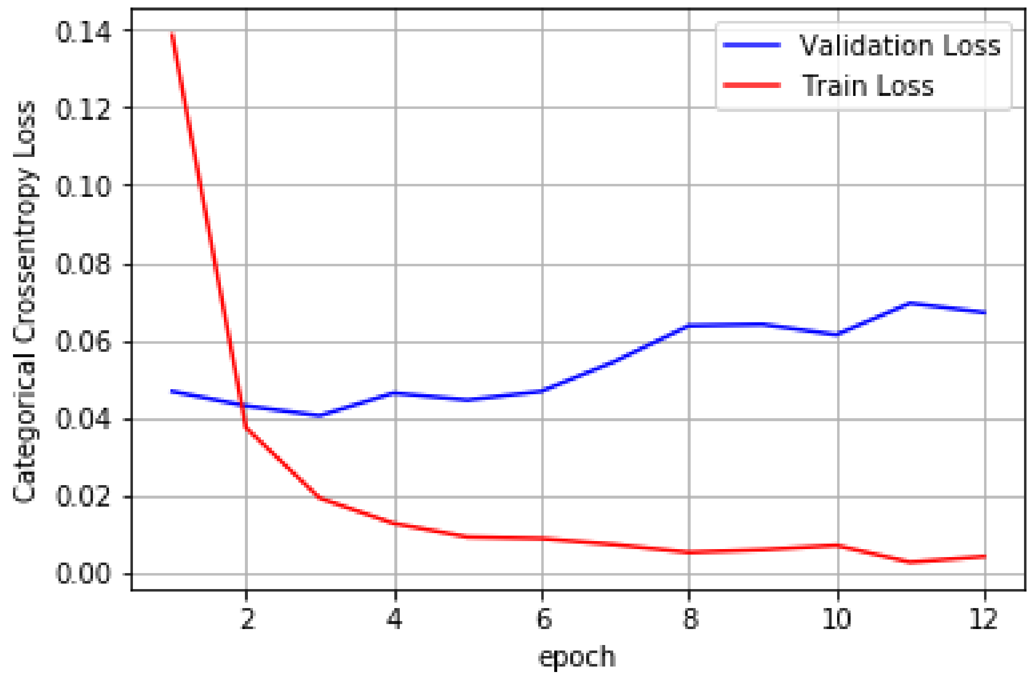
In [10]:
```
model = Sequential()
model.add(Conv2D(512, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 26, 26, 512)       5120
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 26, 26, 256)       1179904
_________________________________________________________________
flatten_4 (Flatten)          (None, 173056)            0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               22151296
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 23,337,610
Trainable params: 23,337,610
Non-trainable params: 0
_________________________________________________________________
```

```
In [11]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'])

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

         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))

         # print(history.history.keys())
         # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
         # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

         # we will get val_loss and val_acc only when you pass the paramter validation_data
         # val_loss : validation loss
         # val_acc : validation accuracy

         # loss : training loss
```

```
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 50s 829us/step - loss: 0.1386 - acc: 0.9589 - va
Epoch 2/12
60000/60000 [==============================] - 46s 765us/step - loss: 0.0376 - acc: 0.9880 - va
Epoch 3/12
60000/60000 [==============================] - 46s 768us/step - loss: 0.0194 - acc: 0.9937 - va
Epoch 4/12
60000/60000 [==============================] - 46s 761us/step - loss: 0.0129 - acc: 0.9960 - va
Epoch 5/12
60000/60000 [==============================] - 46s 759us/step - loss: 0.0094 - acc: 0.9969 - va
Epoch 6/12
60000/60000 [==============================] - 46s 758us/step - loss: 0.0090 - acc: 0.9970 - va
Epoch 7/12
60000/60000 [==============================] - 45s 756us/step - loss: 0.0074 - acc: 0.9976 - va
Epoch 8/12
60000/60000 [==============================] - 45s 754us/step - loss: 0.0054 - acc: 0.9982 - va
Epoch 9/12
60000/60000 [==============================] - 45s 753us/step - loss: 0.0061 - acc: 0.9981 - va
Epoch 10/12
60000/60000 [==============================] - 45s 753us/step - loss: 0.0072 - acc: 0.9978 - va
Epoch 11/12
60000/60000 [==============================] - 45s 750us/step - loss: 0.0029 - acc: 0.9991 - va
Epoch 12/12
60000/60000 [==============================] - 45s 750us/step - loss: 0.0044 - acc: 0.9986 - va
Test score: 0.06720557982521506
Test accuracy: 0.987
```



2.1 Evaluating Model

We can clearly see that our model is **overfitting**. If we kept filter width large at initial stage then model tends to **Overfit**,

Now we will try with less number of filter at initial stage and gradually increasing the same.

```
In [0]: model = Sequential()
        model.add(Conv2D(16, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
        model.add(Conv2D(16, kernel_size=(3, 3), activation='relu', padding='same'))


        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Conv2D(16, (3, 3), activation='relu'))

        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(BatchNormalization())
        model.add(Dropout(0.15))

        model.add(Conv2D(32, (3, 3), activation='relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(BatchNormalization())
        model.add(Dropout(0.15))


        model.add(Flatten())
```

```
        model.add(Dense(128, activation='relu'))
        model.add(Dense(num_classes, activation='softmax'))

        model.summary()
```

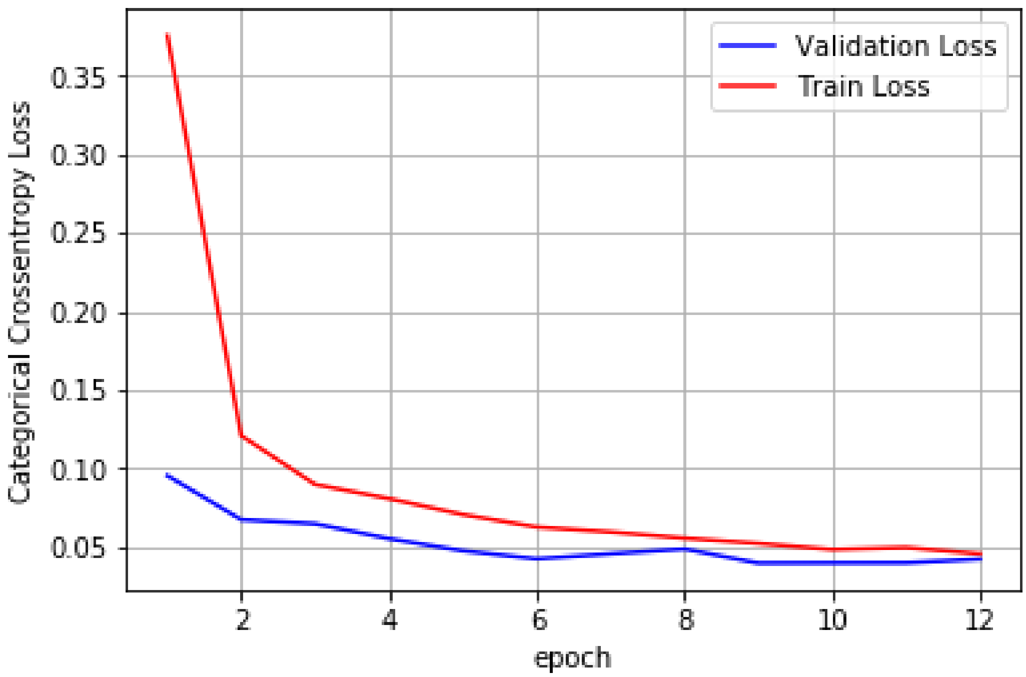
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 26, 26, 16)        2320
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 16)        0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 11, 11, 16)        2320
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
batch_normalization_7 (Batch (None, 5, 5, 16)          64
_________________________________________________________________
dropout_7 (Dropout)          (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 3, 3, 32)          4640
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 1, 1, 32)          0
_________________________________________________________________
batch_normalization_8 (Batch (None, 1, 1, 32)          128
_________________________________________________________________
dropout_8 (Dropout)          (None, 1, 1, 32)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               4224
_________________________________________________________________
dense_10 (Dense)             (None, 10)                1290
=================================================================
Total params: 15,146
Trainable params: 15,050
Non-trainable params: 96
_________________________________________________________________
```

```
In [0]: model.compile(loss=keras.losses.categorical_crossentropy,
                      optimizer=keras.optimizers.Adam(),
                      metrics=['accuracy'])

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

    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))

    # print(history.history.keys())
    # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
    # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, v

    # we will get val_loss and val_acc only when you pass the paramter validation_data
    # val_loss : validation loss
    # val_acc : validation accuracy

    # loss : training loss
    # acc : train accuracy
    # for each key in histrory.histrory we will have a list of length equal to number of e

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 5s 88us/step - loss: 0.3755 - acc: 0.8907 - val_
Epoch 2/12
60000/60000 [==============================] - 4s 65us/step - loss: 0.1209 - acc: 0.9630 - val_
Epoch 3/12
60000/60000 [==============================] - 4s 65us/step - loss: 0.0898 - acc: 0.9718 - val_
Epoch 4/12
60000/60000 [==============================] - 4s 65us/step - loss: 0.0807 - acc: 0.9749 - val_
Epoch 5/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0706 - acc: 0.9776 - val_
Epoch 6/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0626 - acc: 0.9805 - val_
Epoch 7/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0594 - acc: 0.9811 - val_
Epoch 8/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0555 - acc: 0.9823 - val_
Epoch 9/12
```

```
60000/60000 [==============================] - 4s 65us/step - loss: 0.0522 - acc: 0.9832 - val_
Epoch 10/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0483 - acc: 0.9844 - val_
Epoch 11/12
60000/60000 [==============================] - 4s 63us/step - loss: 0.0495 - acc: 0.9841 - val_
Epoch 12/12
60000/60000 [==============================] - 4s 64us/step - loss: 0.0455 - acc: 0.9857 - val_
Test score: 0.042175486064370486
Test accuracy: 0.9877
```



2.2 Evaluating Model

We can clearly observe that we initially we kept filter size is less and gradually increasing gives our perfect model.

Model is not overfitting.

3 5*5 ConvNet, Optimizer 'ADAM'

```
In [20]: model = Sequential()

         #1st laeyr
         model.add(Conv2D(1, kernel_size=(5, 5),activation='relu',padding='same', input_shape=
         model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

```
model.add(Dropout(0.10))


# 2nd layer
model.add(Conv2D(2, kernel_size=(5, 5),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(4, activation='relu'))
model.add(Dense(8, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

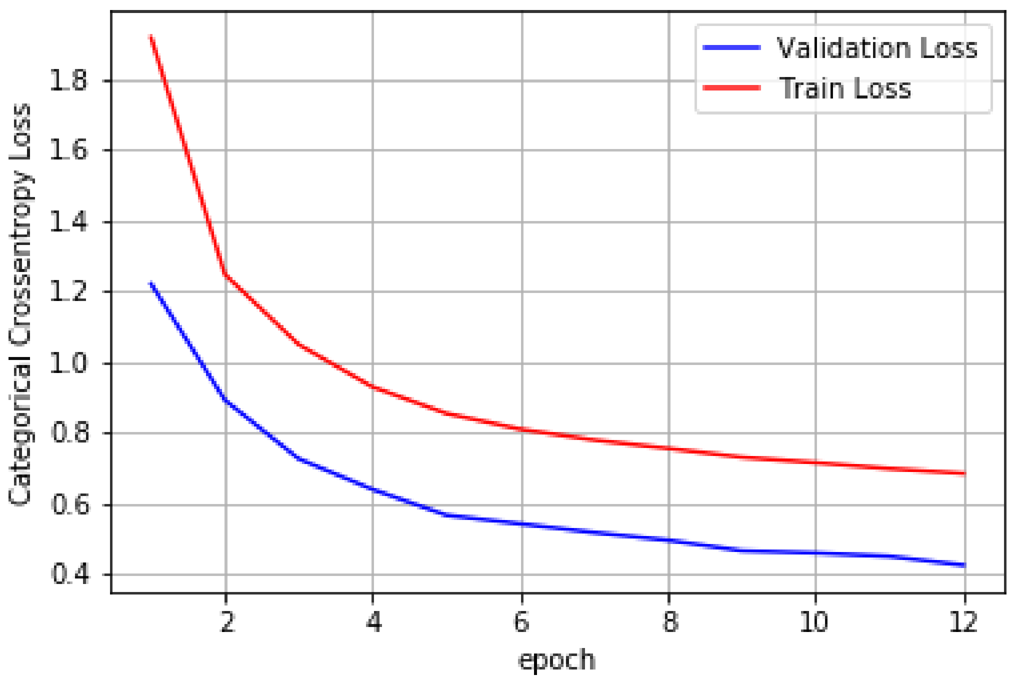
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_25 (Conv2D)           (None, 28, 28, 1)         26
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 14, 14, 1)         0
_________________________________________________________________
dropout_15 (Dropout)         (None, 14, 14, 1)         0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 10, 10, 2)         52
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 5, 5, 2)           0
_________________________________________________________________
dropout_16 (Dropout)         (None, 5, 5, 2)           0
_________________________________________________________________
flatten_9 (Flatten)          (None, 50)                0
_________________________________________________________________
dense_20 (Dense)             (None, 4)                 204
_________________________________________________________________
dense_21 (Dense)             (None, 8)                 40
_________________________________________________________________
dense_22 (Dense)             (None, 10)                90
=================================================================
Total params: 412
Trainable params: 412
Non-trainable params: 0
_________________________________________________________________
```

```
In [21]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
```

```
                      metrics=['accuracy'])

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

        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))

        # print(history.history.keys())
        # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
        # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

        # we will get val_loss and val_acc only when you pass the paramter validation_data
        # val_loss : validation loss
        # val_acc : validation accuracy

        # loss : training loss
        # acc : train accuracy
        # for each key in histrory.histrory we will have a list of length equal to number of

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 4s 70us/step - loss: 1.9163 - acc: 0.3024 - val_
Epoch 2/12
60000/60000 [==============================] - 3s 47us/step - loss: 1.2467 - acc: 0.5540 - val_
Epoch 3/12
60000/60000 [==============================] - 3s 45us/step - loss: 1.0484 - acc: 0.6328 - val_
Epoch 4/12
60000/60000 [==============================] - 3s 47us/step - loss: 0.9283 - acc: 0.6795 - val_
Epoch 5/12
60000/60000 [==============================] - 3s 46us/step - loss: 0.8530 - acc: 0.7091 - val_
Epoch 6/12
60000/60000 [==============================] - 3s 45us/step - loss: 0.8090 - acc: 0.7254 - val_
Epoch 7/12
```

```
60000/60000 [==============================] - 3s 45us/step - loss: 0.7784 - acc: 0.7375 - val_
Epoch 8/12
60000/60000 [==============================] - 3s 44us/step - loss: 0.7546 - acc: 0.7474 - val_
Epoch 9/12
60000/60000 [==============================] - 3s 44us/step - loss: 0.7296 - acc: 0.7583 - val_
Epoch 10/12
60000/60000 [==============================] - 3s 44us/step - loss: 0.7143 - acc: 0.7627 - val_
Epoch 11/12
60000/60000 [==============================] - 3s 45us/step - loss: 0.6974 - acc: 0.7701 - val_
Epoch 12/12
60000/60000 [==============================] - 3s 44us/step - loss: 0.6839 - acc: 0.7731 - val_
Test score: 0.424343634724617
Test accuracy: 0.8811
```



3.1 Evaluating Model

As we observe training error is large and validation error is small way smaller then model is **underfit**. We have used extremely small number of filters, which cause model tends to underfit.

How we will increase slightly number of filters ie increasing filter width

```
In [36]: model = Sequential()

         #1st laeyr
         model.add(Conv2D(8, kernel_size=(5, 5),activation='relu',padding='same', input_shape=i
```

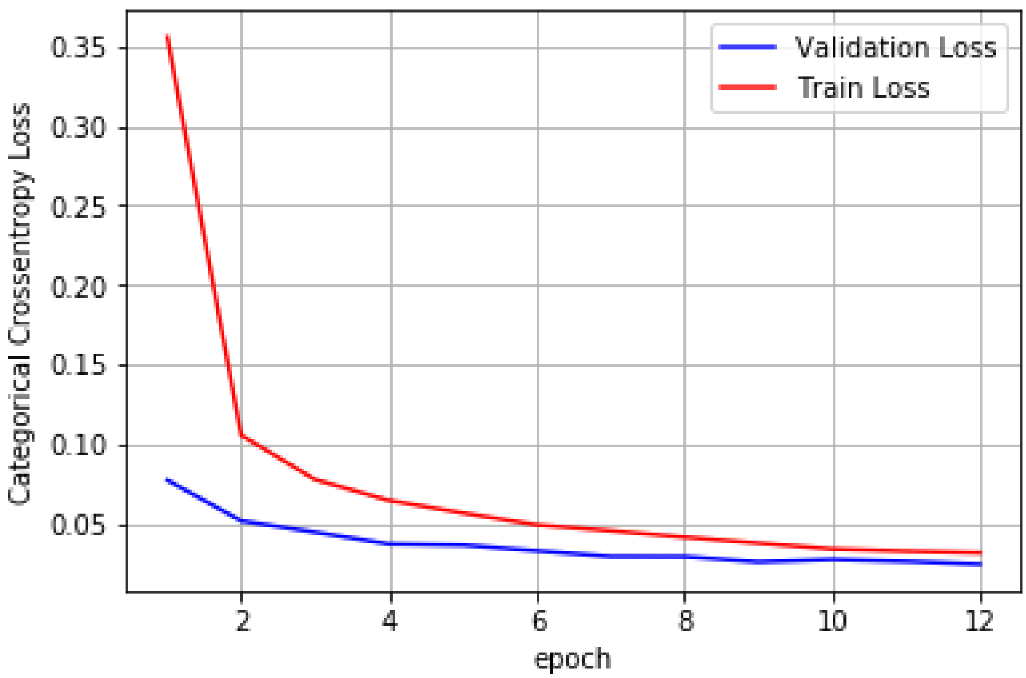
```python
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Dropout(0.10))


# 2nd layer
model.add(Conv2D(16, kernel_size=(5, 5),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Dropout(0.25))


model.add(Flatten())
model.add(Dense(200, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_41 (Conv2D)           (None, 28, 28, 8)         208
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 14, 14, 8)         0
_________________________________________________________________
dropout_31 (Dropout)         (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 10, 10, 16)        3216
_________________________________________________________________
max_pooling2d_36 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
dropout_32 (Dropout)         (None, 5, 5, 16)          0
_________________________________________________________________
flatten_17 (Flatten)         (None, 400)               0
_________________________________________________________________
dense_51 (Dense)             (None, 200)               80200
_________________________________________________________________
dense_52 (Dense)             (None, 128)               25728
_________________________________________________________________
dense_53 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_54 (Dense)             (None, 10)                650
=================================================================
Total params: 118,258
Trainable params: 118,258
Non-trainable params: 0
_________________________________________________________________
```

```
In [37]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'])

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

         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))

         # print(history.history.keys())
         # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
         # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

         # we will get val_loss and val_acc only when you pass the paramter validation_data
         # val_loss : validation loss
         # val_acc : validation accuracy

         # loss : training loss
         # acc : train accuracy
         # for each key in histrory.histrory we will have a list of length equal to number of

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 6s 102us/step - loss: 0.3559 - acc: 0.8856 - va
Epoch 2/12
60000/60000 [==============================] - 3s 52us/step - loss: 0.1058 - acc: 0.9670 - val
Epoch 3/12
60000/60000 [==============================] - 3s 52us/step - loss: 0.0779 - acc: 0.9755 - val
Epoch 4/12
60000/60000 [==============================] - 3s 51us/step - loss: 0.0647 - acc: 0.9801 - val
Epoch 5/12
```

```
60000/60000 [==============================] - 3s 52us/step - loss: 0.0568 - acc: 0.9820 - val_
Epoch 6/12
60000/60000 [==============================] - 3s 52us/step - loss: 0.0493 - acc: 0.9844 - val_
Epoch 7/12
60000/60000 [==============================] - 3s 53us/step - loss: 0.0456 - acc: 0.9851 - val_
Epoch 8/12
60000/60000 [==============================] - 3s 55us/step - loss: 0.0416 - acc: 0.9866 - val_
Epoch 9/12
60000/60000 [==============================] - 3s 55us/step - loss: 0.0378 - acc: 0.9876 - val_
Epoch 10/12
60000/60000 [==============================] - 3s 54us/step - loss: 0.0342 - acc: 0.9892 - val_
Epoch 11/12
60000/60000 [==============================] - 3s 53us/step - loss: 0.0328 - acc: 0.9894 - val_
Epoch 12/12
60000/60000 [==============================] - 3s 52us/step - loss: 0.0318 - acc: 0.9896 - val_
Test score: 0.024570954338563024
Test accuracy: 0.9916
```



4 7*7 ConvNet, Optimizer 'Adam'

In [38]:
```
model = Sequential()

    # 1st layer
```

```python
model.add(Conv2D(16, kernel_size=(7, 7),activation='relu',padding='same', input_shape=
model.add(MaxPooling2D(pool_size=(3, 3),strides=1))
model.add(BatchNormalization())
model.add(Dropout(0.15))


# 2nd layer
model.add(Conv2D(32, (7, 7), activation='relu'))
model.add(MaxPooling2D(pool_size=(3,3 ),strides=2))
model.add(BatchNormalization())
model.add(Dropout(0.20))


model.add(Flatten())
model.add(Dense(3200, activation='relu'))
model.add(Dense(800, activation='relu'))
model.add(Dense(400, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

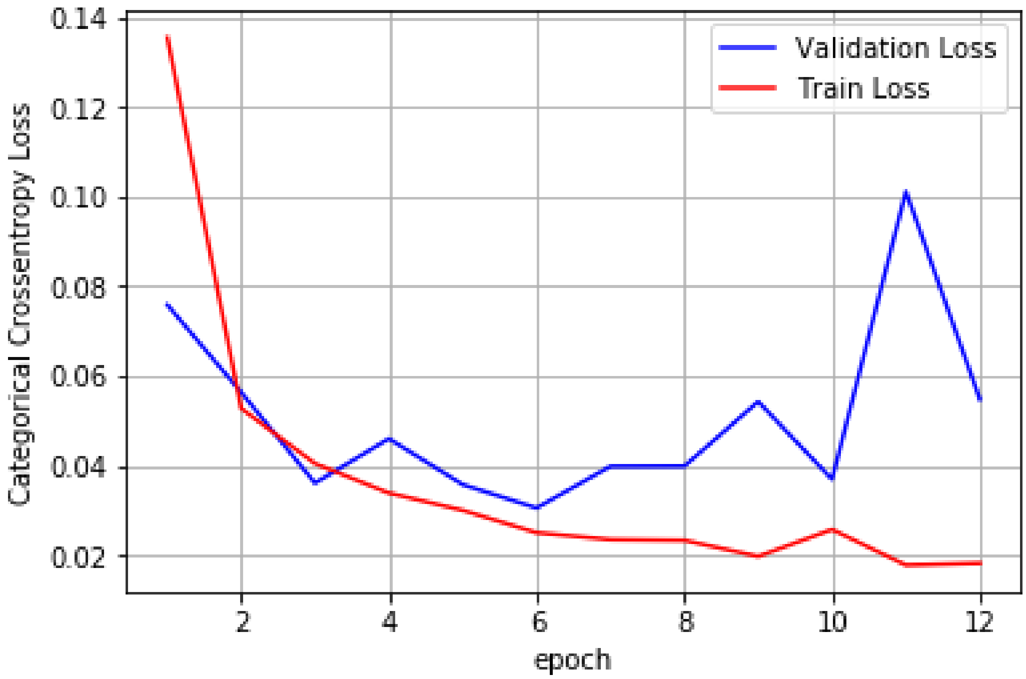
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_43 (Conv2D)           (None, 28, 28, 16)        800       
_________________________________________________________________
max_pooling2d_37 (MaxPooling (None, 26, 26, 16)        0         
_________________________________________________________________
batch_normalization_21 (Batc (None, 26, 26, 16)        64        
_________________________________________________________________
dropout_33 (Dropout)         (None, 26, 26, 16)        0         
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 20, 20, 32)        25120     
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 9, 9, 32)          0         
_________________________________________________________________
batch_normalization_22 (Batc (None, 9, 9, 32)          128       
_________________________________________________________________
dropout_34 (Dropout)         (None, 9, 9, 32)          0         
_________________________________________________________________
flatten_18 (Flatten)         (None, 2592)              0         
_________________________________________________________________
dense_55 (Dense)             (None, 3200)              8297600   
_________________________________________________________________
dense_56 (Dense)             (None, 800)               2560800   
_________________________________________________________________
dense_57 (Dense)             (None, 400)               320400    
```

```
-----------------------------------------------------------------
dense_58 (Dense)             (None, 10)               4010
=================================================================
Total params: 11,208,922
Trainable params: 11,208,826
Non-trainable params: 96
-----------------------------------------------------------------
```

```
In [39]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'])

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

         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))

         # print(history.history.keys())
         # dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
         # history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

         # we will get val_loss and val_acc only when you pass the paramter validation_data
         # val_loss : validation loss
         # val_acc : validation accuracy

         # loss : training loss
         # acc : train accuracy
         # for each key in histrory.histrory we will have a list of length equal to number of 

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 12s 208us/step - loss: 0.1354 - acc: 0.9626 - va
```

```
Epoch 2/12
60000/60000 [==============================] - 9s 146us/step - loss: 0.0528 - acc: 0.9850 - val
Epoch 3/12
60000/60000 [==============================] - 9s 147us/step - loss: 0.0405 - acc: 0.9891 - val
Epoch 4/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0339 - acc: 0.9907 - val
Epoch 5/12
60000/60000 [==============================] - 9s 147us/step - loss: 0.0301 - acc: 0.9917 - val
Epoch 6/12
60000/60000 [==============================] - 9s 147us/step - loss: 0.0251 - acc: 0.9927 - val
Epoch 7/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0235 - acc: 0.9933 - val
Epoch 8/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0234 - acc: 0.9935 - val
Epoch 9/12
60000/60000 [==============================] - 9s 149us/step - loss: 0.0198 - acc: 0.9947 - val
Epoch 10/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0258 - acc: 0.9931 - val
Epoch 11/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0178 - acc: 0.9951 - val
Epoch 12/12
60000/60000 [==============================] - 9s 148us/step - loss: 0.0183 - acc: 0.9956 - val
Test score: 0.054796922633474424
Test accuracy: 0.9884
```



4.1 Evaluating Model

It can be observe that model is overfitting because we have used Max Pooling matrices (3,3) , for ConvNet (7,7).

We have using 2*2 matrics for Max Pooling.

```
In [40]: model = Sequential()

         # 1st layer
         model.add(Conv2D(16, kernel_size=(7, 7),activation='relu',padding='same', input_shape=
         model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
         model.add(BatchNormalization())
         model.add(Dropout(0.15))


         # 2nd layer
         model.add(Conv2D(32, (7, 7), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2,2 ),strides=2))
         model.add(BatchNormalization())
         model.add(Dropout(0.20))


         model.add(Flatten())
         model.add(Dense(3200, activation='relu'))
         model.add(Dense(800, activation='relu'))
         model.add(Dense(400, activation='relu'))

         model.add(Dense(num_classes, activation='softmax'))

         model.summary()
```

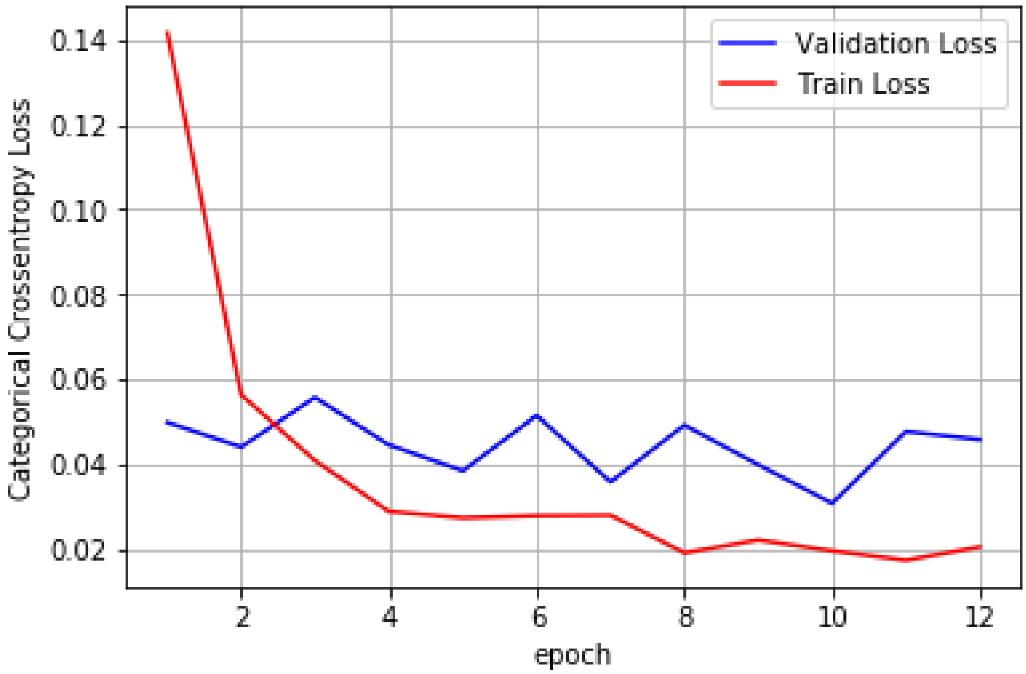
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_45 (Conv2D)           (None, 28, 28, 16)        800
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 27, 27, 16)        0
_________________________________________________________________
batch_normalization_23 (Batc (None, 27, 27, 16)        64
_________________________________________________________________
dropout_35 (Dropout)         (None, 27, 27, 16)        0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 21, 21, 32)        25120
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
batch_normalization_24 (Batc (None, 10, 10, 32)        128
_________________________________________________________________
dropout_36 (Dropout)         (None, 10, 10, 32)        0
```

```
_________________________________________________________________
flatten_19 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_59 (Dense)             (None, 3200)              10243200
_________________________________________________________________
dense_60 (Dense)             (None, 800)               2560800
_________________________________________________________________
dense_61 (Dense)             (None, 400)               320400
_________________________________________________________________
dense_62 (Dense)             (None, 10)                4010
=================================================================
Total params: 13,154,522
Trainable params: 13,154,426
Non-trainable params: 96
_________________________________________________________________
```

In [41]:
```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

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

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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of 
```

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 14s 231us/step - loss: 0.1416 - acc: 0.9612 - va
Epoch 2/12
60000/60000 [==============================] - 11s 177us/step - loss: 0.0563 - acc: 0.9841 - va
Epoch 3/12
60000/60000 [==============================] - 10s 173us/step - loss: 0.0409 - acc: 0.9884 - va
Epoch 4/12
60000/60000 [==============================] - 10s 172us/step - loss: 0.0289 - acc: 0.9917 - va
Epoch 5/12
60000/60000 [==============================] - 10s 173us/step - loss: 0.0273 - acc: 0.9925 - va
Epoch 6/12
60000/60000 [==============================] - 10s 174us/step - loss: 0.0279 - acc: 0.9924 - va
Epoch 7/12
60000/60000 [==============================] - 10s 173us/step - loss: 0.0280 - acc: 0.9923 - va
Epoch 8/12
60000/60000 [==============================] - 10s 173us/step - loss: 0.0191 - acc: 0.9948 - va
Epoch 9/12
60000/60000 [==============================] - 10s 173us/step - loss: 0.0221 - acc: 0.9939 - va
Epoch 10/12
60000/60000 [==============================] - 10s 174us/step - loss: 0.0196 - acc: 0.9948 - va
Epoch 11/12
60000/60000 [==============================] - 10s 174us/step - loss: 0.0173 - acc: 0.9955 - va
Epoch 12/12
60000/60000 [==============================] - 10s 174us/step - loss: 0.0205 - acc: 0.9951 - va
Test score: 0.04580583838448993
Test accuracy: 0.9914
```



4.2 Evaluating model

It can be observe that, it is better than previous one, But can we experiment on dense layer?

```
In [42]: model = Sequential()

         # 1st layer
         model.add(Conv2D(16, kernel_size=(7, 7),activation='relu',padding='same', input_shape=
         model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
         model.add(BatchNormalization())
         model.add(Dropout(0.15))


         # 2nd layer
         model.add(Conv2D(32, (7, 7), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2,2 ),strides=2))
         model.add(BatchNormalization())
         model.add(Dropout(0.20))


         model.add(Flatten())
         model.add(Dense(400, activation='relu'))

         model.add(Dense(num_classes, activation='softmax'))
```

```
model.summary()
```

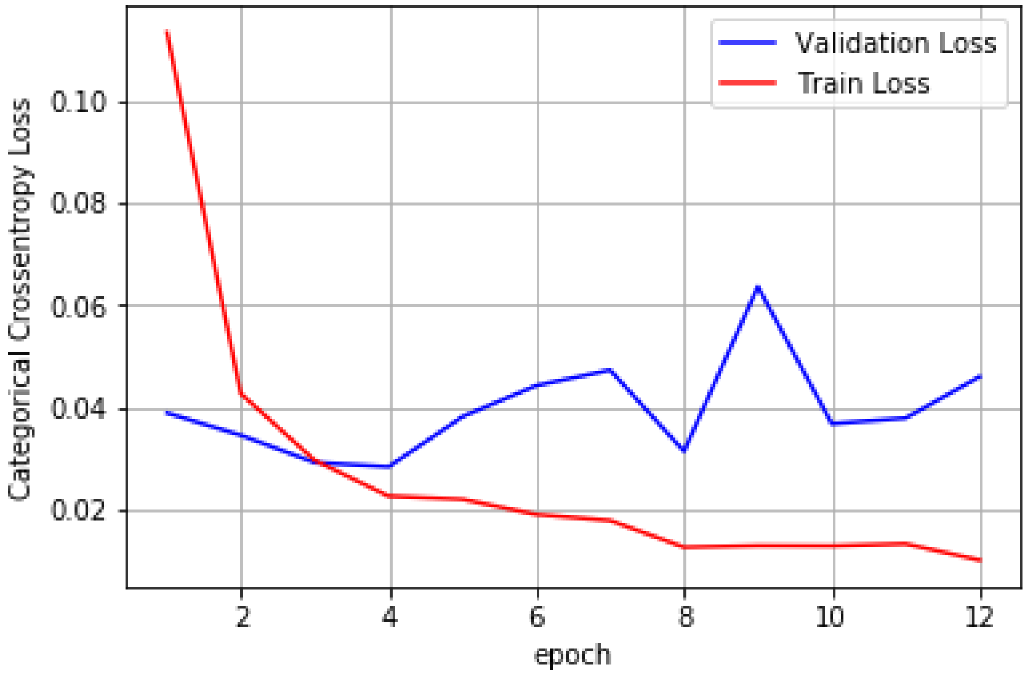
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_47 (Conv2D)           (None, 28, 28, 16)        800       
_________________________________________________________________
max_pooling2d_41 (MaxPooling (None, 27, 27, 16)        0         
_________________________________________________________________
batch_normalization_25 (Batc (None, 27, 27, 16)        64        
_________________________________________________________________
dropout_37 (Dropout)         (None, 27, 27, 16)        0         
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 21, 21, 32)        25120     
_________________________________________________________________
max_pooling2d_42 (MaxPooling (None, 10, 10, 32)        0         
_________________________________________________________________
batch_normalization_26 (Batc (None, 10, 10, 32)        128       
_________________________________________________________________
dropout_38 (Dropout)         (None, 10, 10, 32)        0         
_________________________________________________________________
flatten_20 (Flatten)         (None, 3200)              0         
_________________________________________________________________
dense_63 (Dense)             (None, 400)               1280400   
_________________________________________________________________
dense_64 (Dense)             (None, 10)                4010      
=================================================================
Total params: 1,310,522
Trainable params: 1,310,426
Non-trainable params: 96
_________________________________________________________________
```

```
In [43]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'])

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

```
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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of 

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 9s 156us/step - loss: 0.1133 - acc: 0.9667 - val
Epoch 2/12
60000/60000 [==============================] - 6s 94us/step - loss: 0.0427 - acc: 0.9869 - val_
Epoch 3/12
60000/60000 [==============================] - 6s 94us/step - loss: 0.0297 - acc: 0.9913 - val_
Epoch 4/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0226 - acc: 0.9928 - val_
Epoch 5/12
60000/60000 [==============================] - 6s 94us/step - loss: 0.0220 - acc: 0.9932 - val_
Epoch 6/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0190 - acc: 0.9939 - val_
Epoch 7/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0179 - acc: 0.9947 - val_
Epoch 8/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0126 - acc: 0.9960 - val_
Epoch 9/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0129 - acc: 0.9960 - val_
Epoch 10/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0129 - acc: 0.9959 - val_
Epoch 11/12
60000/60000 [==============================] - 6s 95us/step - loss: 0.0133 - acc: 0.9959 - val_
Epoch 12/12
60000/60000 [==============================] - 6s 97us/step - loss: 0.0101 - acc: 0.9970 - val_
Test score: 0.045987277704650886
```

```
Test accuracy: 0.99
```



4.3 Evaluating Model

It can be observe that if kept single dense layer after flatten operation model start overfitting.
Adding more Dense layer after flatten layer.

```
In [44]: model = Sequential()

         # 1st layer
         model.add(Conv2D(16, kernel_size=(7, 7),activation='relu',padding='same', input_shape=
         model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
         model.add(BatchNormalization())


         # 2nd layer
         model.add(Conv2D(32, (7, 7), activation='relu'))
         model.add(MaxPooling2D(pool_size=(2,2 ),strides=2))
         model.add(BatchNormalization())


         model.add(Flatten())
         model.add(Dense(3200, activation='relu'))
         model.add(Dense(1600, activation='relu'))
```

```
model.add(Dense(800, activation='relu'))
model.add(Dense(400, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

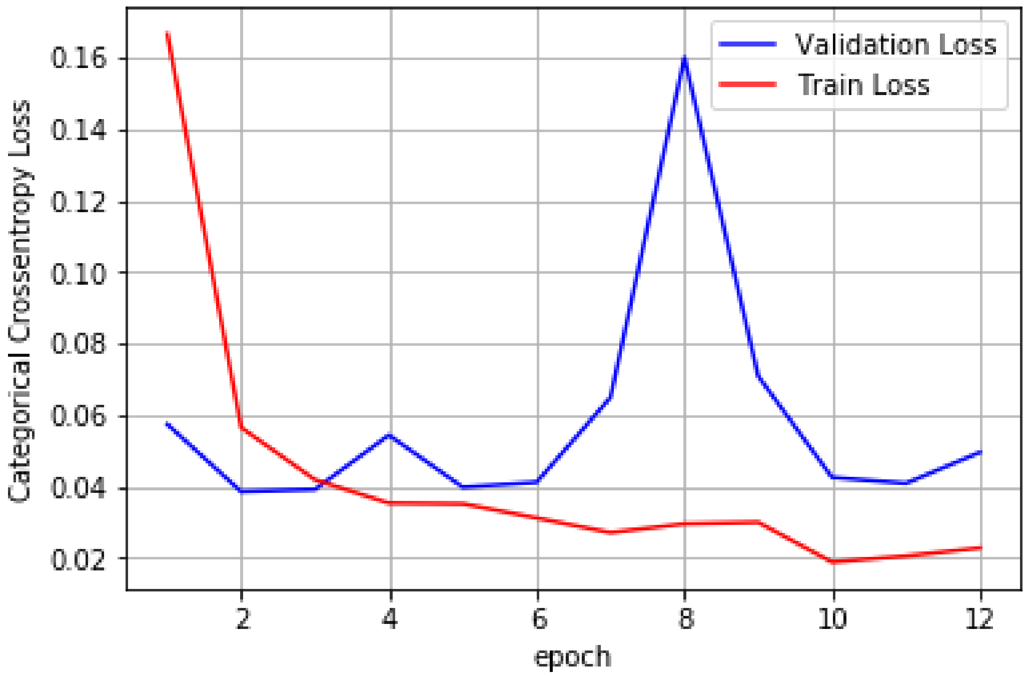
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_49 (Conv2D)           (None, 28, 28, 16)        800
_________________________________________________________________
max_pooling2d_43 (MaxPooling (None, 27, 27, 16)        0
_________________________________________________________________
batch_normalization_27 (Batc (None, 27, 27, 16)        64
_________________________________________________________________
dropout_39 (Dropout)         (None, 27, 27, 16)        0
_________________________________________________________________
conv2d_50 (Conv2D)           (None, 21, 21, 32)        25120
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
batch_normalization_28 (Batc (None, 10, 10, 32)        128
_________________________________________________________________
dropout_40 (Dropout)         (None, 10, 10, 32)        0
_________________________________________________________________
flatten_21 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_65 (Dense)             (None, 3200)              10243200
_________________________________________________________________
dense_66 (Dense)             (None, 1600)              5121600
_________________________________________________________________
dense_67 (Dense)             (None, 800)               1280800
_________________________________________________________________
dense_68 (Dense)             (None, 400)               320400
_________________________________________________________________
dense_69 (Dense)             (None, 10)                4010
=================================================================
Total params: 16,996,122
Trainable params: 16,996,026
Non-trainable params: 96
_________________________________________________________________
```

```
In [45]: model.compile(loss=keras.losses.categorical_crossentropy,
                       optimizer=keras.optimizers.Adam(),
                       metrics=['accuracy'])
```

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

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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, 

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of 

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 16s 271us/step - loss: 0.1666 - acc: 0.9560 - va
Epoch 2/12
60000/60000 [==============================] - 12s 202us/step - loss: 0.0565 - acc: 0.9846 - va
Epoch 3/12
60000/60000 [==============================] - 12s 202us/step - loss: 0.0418 - acc: 0.9886 - va
Epoch 4/12
60000/60000 [==============================] - 12s 204us/step - loss: 0.0353 - acc: 0.9909 - va
Epoch 5/12
60000/60000 [==============================] - 12s 203us/step - loss: 0.0351 - acc: 0.9910 - va
Epoch 6/12
60000/60000 [==============================] - 12s 202us/step - loss: 0.0311 - acc: 0.9924 - va
Epoch 7/12
60000/60000 [==============================] - 12s 202us/step - loss: 0.0270 - acc: 0.9930 - va
Epoch 8/12
```

```
60000/60000 [==============================] - 12s 204us/step - loss: 0.0295 - acc: 0.9929 - va
Epoch 9/12
60000/60000 [==============================] - 12s 204us/step - loss: 0.0299 - acc: 0.9931 - va
Epoch 10/12
60000/60000 [==============================] - 12s 203us/step - loss: 0.0188 - acc: 0.9954 - va
Epoch 11/12
60000/60000 [==============================] - 12s 204us/step - loss: 0.0205 - acc: 0.9950 - va
Epoch 12/12
60000/60000 [==============================] - 12s 206us/step - loss: 0.0227 - acc: 0.9949 - va
Test score: 0.04952140008617812
Test accuracy: 0.9918
```



4.4 Evaluating Model

Its highly Overfitting we can introduce drop out layer to reduce the overfit

In [66]:
```
model = Sequential()

# 1st layer
model.add(Conv2D(16, kernel_size=(7, 7),activation='relu',padding='same', input_shape=
model.add(MaxPooling2D(pool_size=(2, 2),strides=1))
model.add(Dropout(0.50))
```

```
# 2nd layer
model.add(Conv2D(32, (7, 7), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Dropout(0.33))


model.add(Flatten())
model.add(Dense(1600, activation='relu'))
model.add(Dense(800, activation='relu'))
model.add(Dense(400, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

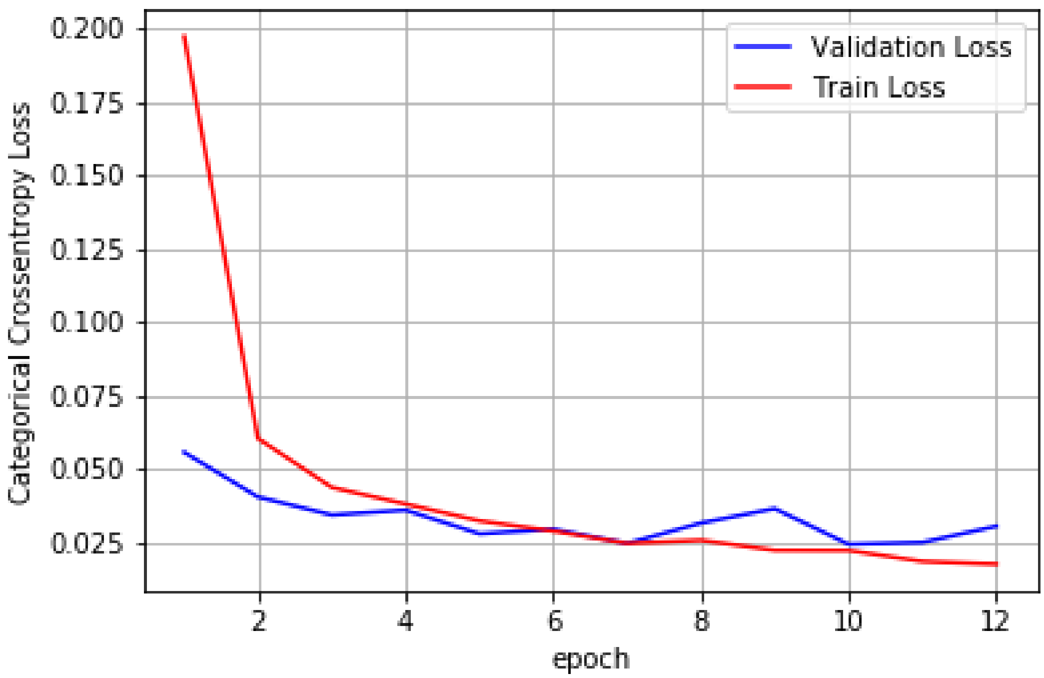
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_74 (Conv2D)           (None, 28, 28, 16)        800
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 27, 27, 16)        0
_________________________________________________________________
dropout_64 (Dropout)         (None, 27, 27, 16)        0
_________________________________________________________________
conv2d_75 (Conv2D)           (None, 21, 21, 32)        25120
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 10, 10, 32)        0
_________________________________________________________________
dropout_65 (Dropout)         (None, 10, 10, 32)        0
_________________________________________________________________
flatten_32 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_104 (Dense)            (None, 1600)              5121600
_________________________________________________________________
dense_105 (Dense)            (None, 800)               1280800
_________________________________________________________________
dense_106 (Dense)            (None, 400)               320400
_________________________________________________________________
dense_107 (Dense)            (None, 10)                4010
=================================================================
Total params: 6,752,730
Trainable params: 6,752,730
Non-trainable params: 0
_________________________________________________________________
```

In [67]:
```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
```

```
                metrics=['accuracy'])

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

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))

# print(history.history.keys())
# dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
# history = model_drop.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch,

# we will get val_loss and val_acc only when you pass the paramter validation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in histrory.histrory we will have a list of length equal to number of

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

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 13s 217us/step - loss: 0.1971 - acc: 0.9373 - va
Epoch 2/12
60000/60000 [==============================] - 8s 128us/step - loss: 0.0603 - acc: 0.9821 - val
Epoch 3/12
60000/60000 [==============================] - 8s 127us/step - loss: 0.0436 - acc: 0.9867 - val
Epoch 4/12
60000/60000 [==============================] - 8s 125us/step - loss: 0.0379 - acc: 0.9883 - val
Epoch 5/12
60000/60000 [==============================] - 7s 125us/step - loss: 0.0322 - acc: 0.9907 - val
Epoch 6/12
60000/60000 [==============================] - 8s 125us/step - loss: 0.0287 - acc: 0.9914 - val
Epoch 7/12
```

```
60000/60000 [==============================] - 7s 125us/step - loss: 0.0245 - acc: 0.9928 - va
Epoch 8/12
60000/60000 [==============================] - 8s 125us/step - loss: 0.0256 - acc: 0.9923 - va
Epoch 9/12
60000/60000 [==============================] - 8s 126us/step - loss: 0.0222 - acc: 0.9939 - va
Epoch 10/12
60000/60000 [==============================] - 8s 125us/step - loss: 0.0221 - acc: 0.9940 - va
Epoch 11/12
60000/60000 [==============================] - 8s 126us/step - loss: 0.0184 - acc: 0.9943 - va
Epoch 12/12
60000/60000 [==============================] - 8s 125us/step - loss: 0.0174 - acc: 0.9952 - va
Test score: 0.030353180483853794
Test accuracy: 0.9919
```



5 Conclusion

```
In [68]: import pandas as pd
         from prettytable import PrettyTable

         bold = '\033[1m'
         end = '\033[0m'
```

```python
print(bold+'\t\t\t Convolutional Neural Network  '+end)


x = PrettyTable()
x.field_names =  ['Metric','3*3 ConvNet','5*5 ConvNet', '7*7 Convnet']
x.add_row(["Optimizer ", 'AdaDelta','ADAM','ADAM'])


x.add_row(["Train Accuracy ", 0.9857,0.9896,0.9952])
x.add_row(["Train Loss ", 0.0455,0.0318,0.0174])
x.add_row(["Test Accuracy ",0.9877,0.9916,0.9919])
x.add_row(["Test Loss ", 0.0422,0.0246,0.0304])


print('\n')
print(x)
```

```
                        Convolutional Neural Network


+-----------------+-------------+-------------+-------------+
|      Metric     | 3*3 ConvNet | 5*5 ConvNet | 7*7 Convnet |
+-----------------+-------------+-------------+-------------+
|    Optimizer    |   AdaDelta  |     ADAM    |     ADAM    |
| Train Accuracy  |    0.9857   |    0.9896   |    0.9952   |
|   Train Loss    |    0.0455   |    0.0318   |    0.0174   |
|  Test Accuracy  |    0.9877   |    0.9916   |    0.9919   |
|    Test Loss    |    0.0422   |    0.0246   |    0.0304   |
+-----------------+-------------+-------------+-------------+
```

6 Summary

1.Always start by using smaller filters is to collect as much local information as possible, and then gradually increase the filter width to reduce the generated feature space width to represent more global, high-level and representative information

2. Large filter size at begining cause OVERFIT a model.

3. Smaller filter size causee UNDERFIT a model.

4. Additing Drop out layer can reduce overfitting of model.

5. using Sigmoid activation layer was worst output for above specific architecture.

6. Adding dense layer after flatten layer is important, It will change out dramatically

7. Larger and Complex Model take more time to train the model.