Various CNN Networks On MNIST Data Set :

In [2]:

```
from __future__ import print_function
from datetime import datetime
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
from keras.initializers import he_normal
from keras.layers.normalization import BatchNormalization
batch_size = 128
num_classes = 10
epochs = 12
# input image dimensions
img_rows, img_cols = 28, 28
```

```
Using TensorFlow backend.
```

In [4]:

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [5]:

```
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 [6]:

```
%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):
  fig = plt.figure( facecolor='y', edgecolor='k')
  plt.plot(x, vy, 'b', label="Validation Loss")
  plt.plot(x, ty, 'r', label="Train Loss")
  plt.xlabel('Epochs')
  plt.ylabel('Categorical Crossentropy Loss')
  plt.legend()
  plt.grid()
  plt.show()
```

1st Model With 3 ConvNet & 3*3 Kernel Size :

In [7]:

```
convnet=Sequential() # Initializing the model
convnet.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=input_shape))
convnet.add(Conv2D(64,kernel_size=(3,3),activation='relu'))
convnet.add(Dropout(0.25))
convnet.add(Conv2D(128,kernel_size=(3,3),activation='relu'))
#maxpooling by (2,2 ) ,dropout,flattening
convnet.add(MaxPooling2D(pool_size=(2,2)))
convnet.add(Dropout(0.25))
convnet.add(Flatten())
#hidden_layer
convnet.add(Dense(256,activation='relu',kernel_initializer=he_normal(seed=None)))
convnet.add(Dropout(0.5))
convnet.add(Dense(num_classes,activation='softmax'))
print(convnet.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 22, 128)       73856     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 11, 11, 128)       0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 11, 11, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 15488)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               3965184   
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570      
=================================================================
Total params: 4,060,426
Trainable params: 4,060,426
Non-trainable params: 0
_________________________________________________________________
None
```

In [8]:

```
convnet.compile(optimizer=keras.optimizers.Adam(),loss=keras.losses.categorical_crossentropy,metric
s=['accuracy'])
convnet_history=convnet.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validatio
n_data=(x_test, y_test))
```

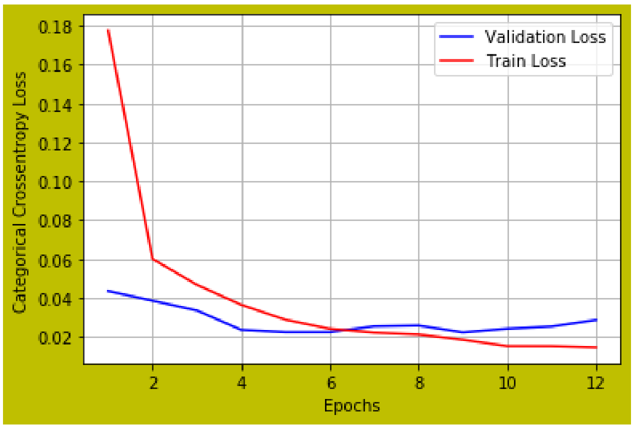
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 231s 4ms/step - loss: 0.1775 - accuracy: 0.9453 - v
al_loss: 0.0435 - val_accuracy: 0.9857
Epoch 2/12
60000/60000 [==============================] - 228s 4ms/step - loss: 0.0601 - accuracy: 0.9823 - v
al_loss: 0.0385 - val_accuracy: 0.9874
Epoch 3/12
60000/60000 [==============================] - 226s 4ms/step - loss: 0.0467 - accuracy: 0.9857 - v
al_loss: 0.0335 - val_accuracy: 0.9892
Epoch 4/12
60000/60000 [==============================] - 228s 4ms/step - loss: 0.0365 - accuracy: 0.9886 - v
```

```
al_loss: 0.0235 - val_accuracy: 0.9918
Epoch 5/12
60000/60000 [==============================] - 237s 4ms/step - loss: 0.0288 - accuracy: 0.9912 - v
al_loss: 0.0224 - val_accuracy: 0.9924
Epoch 6/12
60000/60000 [==============================] - 237s 4ms/step - loss: 0.0241 - accuracy: 0.9923 - v
al_loss: 0.0224 - val_accuracy: 0.9929
Epoch 7/12
60000/60000 [==============================] - 231s 4ms/step - loss: 0.0221 - accuracy: 0.9929 - v
al_loss: 0.0254 - val_accuracy: 0.9927
Epoch 8/12
60000/60000 [==============================] - 225s 4ms/step - loss: 0.0212 - accuracy: 0.9932 - v
al_loss: 0.0258 - val_accuracy: 0.9927
Epoch 9/12
60000/60000 [==============================] - 230s 4ms/step - loss: 0.0184 - accuracy: 0.9944 - v
al_loss: 0.0222 - val_accuracy: 0.9936
Epoch 10/12
60000/60000 [==============================] - 233s 4ms/step - loss: 0.0152 - accuracy: 0.9951 - v
al_loss: 0.0241 - val_accuracy: 0.9936
Epoch 11/12
60000/60000 [==============================] - 222s 4ms/step - loss: 0.0151 - accuracy: 0.9952 - v
al_loss: 0.0253 - val_accuracy: 0.9922
Epoch 12/12
60000/60000 [==============================] - 222s 4ms/step - loss: 0.0145 - accuracy: 0.9953 - v
al_loss: 0.0286 - val_accuracy: 0.9926
```

In [10]:

```
score=convnet.evaluate(x_test,y_test,verbose=0)
test_score=score[0]
test_accuracy=score[1]
train_accuracy=max(convnet_history.history['accuracy'])
print('test score :',test_score)
print('test sccuracy :',test_accuracy)
# error plot
x=list(range(1,epochs+1))
vy=convnet_history.history['val_loss'] #validation loss
ty=convnet_history.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
test score : 0.028591023394818865
test sccuracy : 0.9926000237464905
```



2nd Model CNN with 5 ConvNet & 5*5 Kernel Size :

In [11]:

```
convnet2=Sequential() # Initializing the model
# First ConvNet
convnet2.add(Conv2D(32,kernel_size=(5,5),activation='relu',padding='same',input_shape=input_shape))
convnet2.add(Conv2D(64,kernel_size=(5,5),padding='same',activation='relu'))#Second Convnet
convnet2.add(MaxPooling2D(pool_size=(2,2)))
convnet2.add(Dropout(0.25))
convnet2.add(Conv2D(96,kernel_size=(5,5),padding='same',activation='relu'))  # 3rd ConvNet
#maxpooling by (2,2 ) ,dropout,flattening
convnet2.add(MaxPooling2D(pool_size=(2,2)))
```

```
convnet2.add(Dropout(0.25))
convnet2.add(Conv2D(128,kernel_size=(5,5),padding='same',activation='relu'))#fourth Convnet
convnet2.add(MaxPooling2D(pool_size=(2,2)))
convnet2.add(Dropout(0.25))
convnet2.add(Conv2D(164,kernel_size=(5,5),padding='same',activation='relu'))#fifth Convnet
convnet2.add(MaxPooling2D(pool_size=(2,2)))
convnet2.add(Dropout(0.25))
convnet2.add(Flatten())
#hidden_layer
convnet2.add(Dense(256,activation='relu',kernel_initializer=he_normal(seed=None)))
convnet2.add(BatchNormalization())
convnet2.add(Dropout(0.5))
convnet2.add(Dense(num_classes,activation='softmax'))
print(convnet2.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 96)        153696
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 96)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 7, 7, 96)          0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 128)         307328
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 128)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 3, 128)         0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 3, 3, 164)         524964
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 1, 1, 164)         0
_________________________________________________________________
dropout_7 (Dropout)          (None, 1, 1, 164)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 164)               0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               42240
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_8 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,083,918
Trainable params: 1,083,406
Non-trainable params: 512
_________________________________________________________________
None
```

In [12]:

```
convnet2.compile(optimizer=keras.optimizers.Adam(),loss=keras.losses.categorical_crossentropy,metrics=['accuracy'])
convnet2_history=convnet2.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

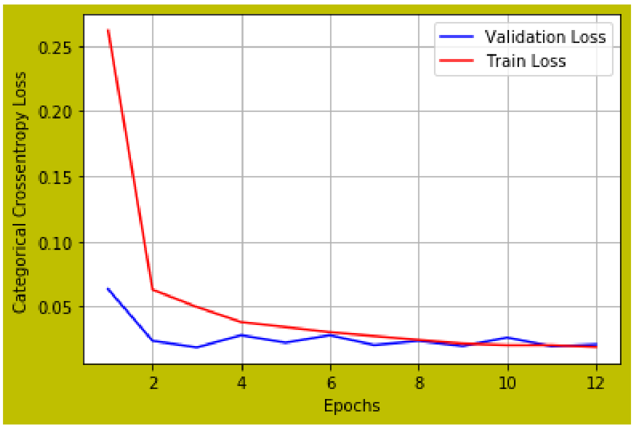
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 294s 5ms/step - loss: 0.2618 - accuracy: 0.9147 - val_loss: 0.0633 - val_accuracy: 0.9804
Epoch 2/12
60000/60000 [==============================] - 290s 5ms/step - loss: 0.0630 - accuracy: 0.9814 - val_loss: 0.0236 - val_accuracy: 0.9920
```

```
al_loss: 0.0230 - val_accuracy: 0.9920
Epoch 3/12
60000/60000 [==============================] - 288s 5ms/step - loss: 0.0497 - accuracy: 0.9856 - v
al_loss: 0.0186 - val_accuracy: 0.9933
Epoch 4/12
60000/60000 [==============================] - 290s 5ms/step - loss: 0.0381 - accuracy: 0.9888 - v
al_loss: 0.0280 - val_accuracy: 0.9924
Epoch 5/12
60000/60000 [==============================] - 289s 5ms/step - loss: 0.0342 - accuracy: 0.9896 - v
al_loss: 0.0222 - val_accuracy: 0.9934
Epoch 6/12
60000/60000 [==============================] - 290s 5ms/step - loss: 0.0303 - accuracy: 0.9914 - v
al_loss: 0.0280 - val_accuracy: 0.9925
Epoch 7/12
60000/60000 [==============================] - 289s 5ms/step - loss: 0.0273 - accuracy: 0.9920 - v
al_loss: 0.0204 - val_accuracy: 0.9945
Epoch 8/12
60000/60000 [==============================] - 290s 5ms/step - loss: 0.0244 - accuracy: 0.9930 - v
al_loss: 0.0236 - val_accuracy: 0.9930
Epoch 9/12
60000/60000 [==============================] - 291s 5ms/step - loss: 0.0215 - accuracy: 0.9937 - v
al_loss: 0.0196 - val_accuracy: 0.9943
Epoch 10/12
60000/60000 [==============================] - 291s 5ms/step - loss: 0.0203 - accuracy: 0.9939 - v
al_loss: 0.0262 - val_accuracy: 0.9932
Epoch 11/12
60000/60000 [==============================] - 289s 5ms/step - loss: 0.0204 - accuracy: 0.9941 - v
al_loss: 0.0194 - val_accuracy: 0.9943
Epoch 12/12
60000/60000 [==============================] - 291s 5ms/step - loss: 0.0189 - accuracy: 0.9943 - v
al_loss: 0.0209 - val_accuracy: 0.9945
```

In [13]:

```
#evaluating model
score=convnet2.evaluate(x_test,y_test,verbose=0)
test_score2=score[0]
test_accuracy2=score[1]
train_accuracy2=max(convnet2_history.history['accuracy'])
print('test score :',test_score2)
print('test Accuracy :',test_accuracy2)
# error plot
x=list(range(1,epochs+1))
vy=convnet2_history.history['val_loss'] #validation loss
ty=convnet2_history.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
test score : 0.020858116581862943
test Accuracy : 0.9944999814033508
```



3rd Model CNN with 7 ConvNet & 2*2 kernel size :

In [15]:

```
convnet3=Sequential() # Initializing the model
# First ConvNet
convnet3.add(Conv2D(16,kernel_size=(2,2),activation='relu',padding='same',strides=(1,1),input_shape
convnet3.add(Conv2D(16,kernel_size=(2,2),activation='relu',padding='same',strides=(1,1),input_shape
=input_shape))
convnet3.add(Conv2D(32,kernel_size=(2,2),padding='same',strides=(2,2),activation='relu'))#Second Co
nvnet
convnet3.add(Conv2D(64,kernel_size=(2,2),padding='same',activation='relu'))   # 3rd ConvNet
convnet3.add(Dropout(0.15))
convnet3.add(Conv2D(96,kernel_size=(2,2),padding='same',activation='relu'))#fourth Convnet
convnet3.add(MaxPooling2D(pool_size=(2,2)))
convnet3.add(Dropout(0.39))
convnet3.add(Conv2D(128,kernel_size=(2,2),padding='same',activation='relu'))#fifth Convnet
convnet3.add(MaxPooling2D(pool_size=(2,2)))
convnet3.add(Dropout(0.3))
convnet3.add(Conv2D(164,kernel_size=(2,2),padding='same',activation='relu'))#sixth Convnet
convnet3.add(Conv2D(164,kernel_size=(2,2),padding='same',strides=(1,1),activation='relu'))#seventh
Convnet
convnet3.add(MaxPooling2D(pool_size=(2,2)))
convnet3.add(Dropout(0.4))
convnet3.add(Flatten())
#hidden_layer
convnet3.add(Dense(256,activation='relu',kernel_initializer=he_normal(seed=None)))#1 hidden layer
convnet3.add(BatchNormalization())
convnet3.add(Dropout(0.4))
convnet3.add(Dense(148,activation='relu',kernel_initializer=he_normal(seed=None)))#2 hidden layer
convnet3.add(BatchNormalization())
convnet3.add(Dropout(0.3))
convnet3.add(Dense(128,activation='relu',kernel_initializer=he_normal(seed=None)))#3 hidden layer
convnet3.add(BatchNormalization())
convnet3.add(Dropout(0.4))
convnet3.add(Dense(num_classes,activation='softmax'))
print(convnet3.summary())
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_16 (Conv2D)           (None, 28, 28, 16)        80        
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 14, 14, 32)        2080      
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 14, 14, 64)        8256      
_________________________________________________________________
dropout_12 (Dropout)         (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 14, 14, 96)        24672     
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 7, 7, 96)          0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 7, 7, 96)          0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 7, 7, 128)         49280     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 3, 3, 128)         0         
_________________________________________________________________
dropout_14 (Dropout)         (None, 3, 3, 128)         0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 3, 3, 164)         84132     
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 3, 3, 164)         107748    
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 1, 1, 164)         0         
_________________________________________________________________
dropout_15 (Dropout)         (None, 1, 1, 164)         0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 164)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 256)               42240     
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024      
_________________________________________________________________
dropout_16 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 148)               38036     
_________________________________________________________________
batch_normalization_3 (Batch (None, 148)               592       
_________________________________________________________________
dropout_17 (Dropout)         (None, 148)               0         
```

```
_________________________________________________________________
dense_7 (Dense)              (None, 128)               19072
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_18 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 379,014
Trainable params: 377,950
Non-trainable params: 1,064
_________________________________________________________________
None
```

In [16]:

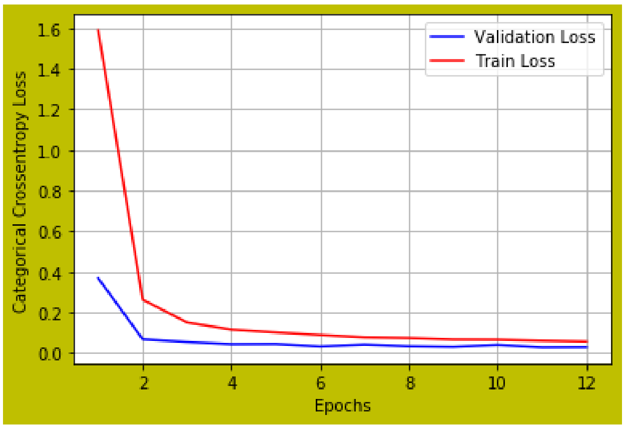
```
convnet3.compile(optimizer=keras.optimizers.Adam(),loss=keras.losses.categorical_crossentropy,metrics=['accuracy'])
convnet3_history=convnet3.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(x_test, y_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 121s 2ms/step - loss: 1.5887 - accuracy: 0.4850 - val_loss: 0.3675 - val_accuracy: 0.8998
Epoch 2/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.2623 - accuracy: 0.9277 - val_loss: 0.0670 - val_accuracy: 0.9809
Epoch 3/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.1499 - accuracy: 0.9601 - val_loss: 0.0532 - val_accuracy: 0.9863
Epoch 4/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.1145 - accuracy: 0.9696 - val_loss: 0.0424 - val_accuracy: 0.9885
Epoch 5/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.1007 - accuracy: 0.9742 - val_loss: 0.0431 - val_accuracy: 0.9892
Epoch 6/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0878 - accuracy: 0.9768 - val_loss: 0.0310 - val_accuracy: 0.9908
Epoch 7/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0761 - accuracy: 0.9794 - val_loss: 0.0404 - val_accuracy: 0.9898
Epoch 8/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0729 - accuracy: 0.9812 - val_loss: 0.0320 - val_accuracy: 0.9913
Epoch 9/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0658 - accuracy: 0.9834 - val_loss: 0.0293 - val_accuracy: 0.9922
Epoch 10/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0651 - accuracy: 0.9829 - val_loss: 0.0388 - val_accuracy: 0.9895
Epoch 11/12
60000/60000 [==============================] - 119s 2ms/step - loss: 0.0597 - accuracy: 0.9844 - val_loss: 0.0268 - val_accuracy: 0.9927
Epoch 12/12
60000/60000 [==============================] - 118s 2ms/step - loss: 0.0555 - accuracy: 0.9857 - val_loss: 0.0276 - val_accuracy: 0.9928
```

In [17]:

```
score=convnet3.evaluate(x_test,y_test,verbose=0)
test_score3=score[0]
test_accuracy3=score[1]
train_accuracy3=max(convnet3_history.history['accuracy'])
print('test score :',test_score3)
print('test Accuracy :',test_accuracy3)
# error plot
x=list(range(1,epochs+1))
vy=convnet3_history.history['val_loss'] #validation loss
ty=convnet3_history.history['loss'] # train loss
plt_dynamic(x, vy, ty)
```

```
test score : 0.027557410120103372
test score : 0.027557419139193372
test Accuracy : 0.9927999973297119
```



Conclusion :

In [18]:

```python
from prettytable import PrettyTable
models=['3ConvNet with kernel 3x3',
        '5ConvNet with kernel 5x5',
        '7ConvNet with kernel 2x2']
training_accuracy=[train_accuracy,train_accuracy2,train_accuracy3]
test_accuracy=[test_accuracy,test_accuracy2,test_accuracy3]
INDEX = [1,2,3]
# Initializing prettytable
Model_Performance = PrettyTable()
# Adding columns
Model_Performance.add_column("INDEX.",INDEX)
Model_Performance.add_column("MODEL_NAME",models)
Model_Performance.add_column("TRAINING ACCURACY",training_accuracy)
Model_Performance.add_column("TESTING ACCURACY",test_accuracy)
#Model_Performance.add_column("TEST SCORE",test_score)

# Printing the Model_Performance
print(Model_Performance)
```

```
+--------+--------------------------+-------------------+--------------------+
| INDEX. |        MODEL_NAME        | TRAINING ACCURACY |  TESTING ACCURACY  |
+--------+--------------------------+-------------------+--------------------+
|   1    | 3ConvNet with kernel 3x3 |     0.9952833     | 0.9926000237464905 |
|   2    | 5ConvNet with kernel 5x5 |     0.99431664    | 0.9944999814033508 |
|   3    | 7ConvNet with kernel 2x2 |     0.9856667     | 0.9927999973297119 |
+--------+--------------------------+-------------------+--------------------+
```