Constructing Different CNN Architectures on MNIST Dataset

Data Overview

The MNIST Dataset is a dataset of Handwritten Characters pertaining to 10 integers from 0 to 9, which is used for Training in the case of Many Image Processing Tasks. We have an input square image of size (28 px* 28 px), which makes the corresponding vector that we obtain to be 784-dimensional. After this, we obtain a Matrix of this dimensionality where each cell in the Matrix corresponds to an integral number from 0 to 255 -> The Higher is the value of this number, the darker that particular pixel value is.

There are a total of 60,000 Training Datapoints and a total of 10,000 Test Datapoints in MNIST. We build various models in order to try and minimize our Test Accuracy and Test Log Loss values. We train each of our models on a total of 85 epochs so that the value is not too small for SGD type Optimizations to try and achieve convergence.

In [1]:

```
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.utils import np_utils
from keras.utils.np_utils import to_categorical
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

In [2]:

```
# Credits: https://github.com/keras-team/keras/blob/master/examples/mni
st_cnn.py
```

```
from sklearn.model_selection import train_test_split

batch_size = 128
num_classes = 10
epochs = 30

# 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

# 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)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

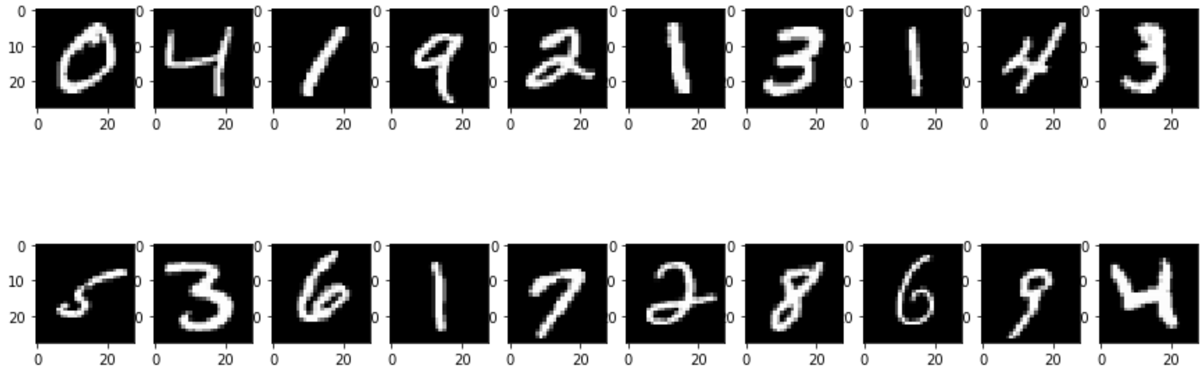
```
60000 train samples
10000 test samples
```

In [3]:
```
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()
```

This is how our Images look like in the MNIST Dataset :

In [4]:
```
#Display or plot a number from the MNIST train dataset. Find the corres
ponding labels below the images.

import warnings
warnings.filterwarnings("ignore")

plt.figure(figsize=(15,15))
for i in range(1,21):
    row = i
    grid_data = x_train[row].reshape(28,28) #Reshape from 1d to 2d pixe
l array
    plt.subplot(5,10,row)
    plt.imshow(grid_data, interpolation = "none", cmap = "gray")
plt.show()
```



1. Number of Convolution Layers in the Neural Network = 2

1.1 Constructing the Neural Network

In [5]:

```
model1 = Sequential()

model1.add(Conv2D(32, kernel_size=(3, 3),activation='relu', input_shape
=input_shape))
model1.add(Conv2D(64, (3, 3), activation='relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

model1.add(Flatten())
model1.add(Dense(128, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes, activation='softmax'))
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776   
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290      
=================================================================
Total params: 1,199,882
```

```
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

1.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [6]:
```
model1.compile(loss=keras.losses.categorical_crossentropy,optimizer=ker
as.optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

M1_history = model1.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 1/30
60000/60000 [==============================] - 94s 2ms/step - loss: 0.2
324 - accuracy: 0.9297 - val_loss: 0.0571 - val_accuracy: 0.9808
Epoch 2/30
60000/60000 [==============================] - 94s 2ms/step - loss: 0.0
834 - accuracy: 0.9754 - val_loss: 0.0446 - val_accuracy: 0.9855
Epoch 3/30
60000/60000 [==============================] - 90s 2ms/step - loss: 0.0
627 - accuracy: 0.9812 - val_loss: 0.0330 - val_accuracy: 0.9885
Epoch 4/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
513 - accuracy: 0.9841 - val_loss: 0.0316 - val_accuracy: 0.9895
Epoch 5/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
440 - accuracy: 0.9862 - val_loss: 0.0297 - val_accuracy: 0.9908
Epoch 6/30
60000/60000 [==============================] - 88s 1ms/step - loss: 0.0
383 - accuracy: 0.9881 - val_loss: 0.0303 - val_accuracy: 0.9896
Epoch 7/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
329 - accuracy: 0.9893 - val_loss: 0.0284 - val_accuracy: 0.9913
Epoch 8/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
```

```
297 - accuracy: 0.9905 - val_loss: 0.0307 - val_accuracy: 0.9908
Epoch 9/30
60000/60000 [==============================] - 93s 2ms/step - loss: 0.0
293 - accuracy: 0.9912 - val_loss: 0.0296 - val_accuracy: 0.9906
Epoch 10/30
60000/60000 [==============================] - 94s 2ms/step - loss: 0.0
267 - accuracy: 0.9917 - val_loss: 0.0306 - val_accuracy: 0.9905
Epoch 11/30
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0
226 - accuracy: 0.9927 - val_loss: 0.0320 - val_accuracy: 0.9911
Epoch 12/30
60000/60000 [==============================] - 90s 2ms/step - loss: 0.0
207 - accuracy: 0.9932 - val_loss: 0.0266 - val_accuracy: 0.9916
Epoch 13/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
193 - accuracy: 0.9936 - val_loss: 0.0332 - val_accuracy: 0.9910
Epoch 14/30
60000/60000 [==============================] - 90s 1ms/step - loss: 0.0
205 - accuracy: 0.9930 - val_loss: 0.0296 - val_accuracy: 0.9916
Epoch 15/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
182 - accuracy: 0.9941 - val_loss: 0.0318 - val_accuracy: 0.9907
Epoch 16/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
168 - accuracy: 0.9947 - val_loss: 0.0284 - val_accuracy: 0.9916
Epoch 17/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
166 - accuracy: 0.9940 - val_loss: 0.0276 - val_accuracy: 0.9921
Epoch 18/30
60000/60000 [==============================] - 90s 1ms/step - loss: 0.0
131 - accuracy: 0.9956 - val_loss: 0.0299 - val_accuracy: 0.9937
Epoch 19/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
143 - accuracy: 0.9947 - val_loss: 0.0323 - val_accuracy: 0.9915
Epoch 20/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
135 - accuracy: 0.9957 - val_loss: 0.0309 - val_accuracy: 0.9926
Epoch 21/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
```

```
135 - accuracy: 0.9954 - val_loss: 0.0295 - val_accuracy: 0.9927
Epoch 22/30
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0
129 - accuracy: 0.9956 - val_loss: 0.0280 - val_accuracy: 0.9928
Epoch 23/30
60000/60000 [==============================] - 107s 2ms/step - loss: 0.
0117 - accuracy: 0.9962 - val_loss: 0.0277 - val_accuracy: 0.9928
Epoch 24/30
60000/60000 [==============================] - 95s 2ms/step - loss: 0.0
109 - accuracy: 0.9962 - val_loss: 0.0329 - val_accuracy: 0.9916
Epoch 25/30
60000/60000 [==============================] - 96s 2ms/step - loss: 0.0
121 - accuracy: 0.9959 - val_loss: 0.0314 - val_accuracy: 0.9925
Epoch 26/30
60000/60000 [==============================] - 95s 2ms/step - loss: 0.0
098 - accuracy: 0.9966 - val_loss: 0.0338 - val_accuracy: 0.9927
Epoch 27/30
60000/60000 [==============================] - 92s 2ms/step - loss: 0.0
108 - accuracy: 0.9964 - val_loss: 0.0297 - val_accuracy: 0.9923
Epoch 28/30
60000/60000 [==============================] - 88s 1ms/step - loss: 0.0
106 - accuracy: 0.9965 - val_loss: 0.0344 - val_accuracy: 0.9918
Epoch 29/30
60000/60000 [==============================] - 87s 1ms/step - loss: 0.0
119 - accuracy: 0.9963 - val_loss: 0.0335 - val_accuracy: 0.9926
Epoch 30/30
60000/60000 [==============================] - 89s 1ms/step - loss: 0.0
101 - accuracy: 0.9966 - val_loss: 0.0323 - val_accuracy: 0.9920
```

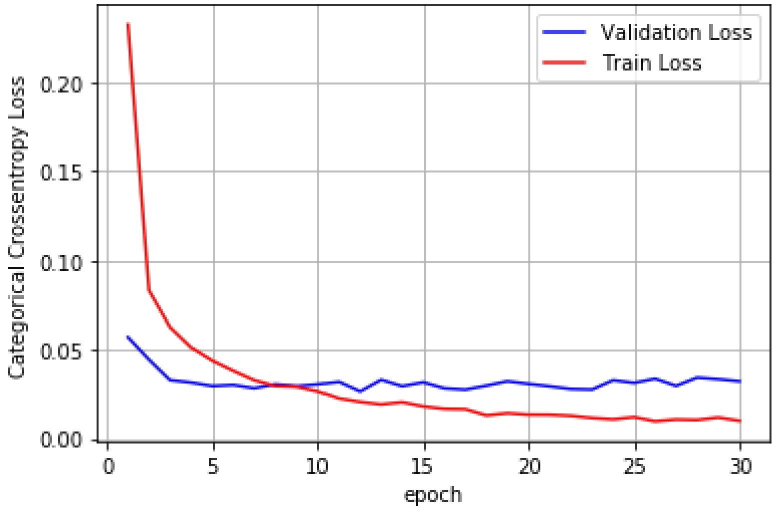
1.3 Number of Epochs vs Train Loss & Validation Loss

In [7]:

```
score1 = model1.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score1[0])
print('Test accuracy:', score1[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

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

vy = M1_history.history['val_loss']
ty = M1_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.032340202385733575
Test accuracy: 0.9919999837875366
```



2. Number of Convolution Layers in the Neural Network = 3

2.1 Model 2 :

2.1.1 Constructing the Neural Network

In [8]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model2 = Sequential()
model2.add(Conv2D(64, kernel_size=(3, 3),
                  activation='relu',
                  input_shape=input_shape, padding='same'))
model2.add(Conv2D(128, (3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.3))

model2.add(Conv2D(256, (3, 3), activation='relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Flatten())
model2.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model2.add(BatchNormalization())
model2.add(Dropout(0.4))

model2.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model2.add(BatchNormalization())
```

```
model2.add(Dropout(0.4))

model2.add(Dense(num_classes, activation='softmax'))
model2.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 64)        640       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 26, 26, 128)       73856     
_________________________________________________________________
batch_normalization_1 (Batch (None, 26, 26, 128)       512       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 13, 13, 128)       0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 13, 13, 128)       0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 256)       295168    
_________________________________________________________________
batch_normalization_2 (Batch (None, 11, 11, 256)       1024      
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 256)         0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 6400)              0         
_________________________________________________________________
dense_3 (Dense)              (None, 512)               3277312   
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048      
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664     
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512       
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290      
```

```
=================================================================
Total params: 3,718,026

Trainable params: 3,715,978
Non-trainable params: 2,048

_________________________________________________________________
```

2.1.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [10]:

```python
from keras import optimizers
model2.compile(loss=keras.losses.categorical_crossentropy, optimizer=op
timizers.Adam(lr=0.001),
              metrics=['accuracy'])

M2_history = model2.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 1/30
60000/60000 [==============================] - 583s 10ms/step - loss:
0.1467 - accuracy: 0.9556 - val_loss: 6.3708 - val_accuracy: 0.4519
Epoch 2/30
60000/60000 [==============================] - 599s 10ms/step - loss:
0.0546 - accuracy: 0.9836 - val_loss: 0.0326 - val_accuracy: 0.9903
Epoch 3/30
60000/60000 [==============================] - 602s 10ms/step - loss:
0.0433 - accuracy: 0.9870 - val_loss: 0.0268 - val_accuracy: 0.9915
Epoch 4/30
60000/60000 [==============================] - 587s 10ms/step - loss:
0.0339 - accuracy: 0.9901 - val_loss: 0.0263 - val_accuracy: 0.9911
Epoch 5/30
60000/60000 [==============================] - 589s 10ms/step - loss:
0.0300 - accuracy: 0.9909 - val_loss: 0.0307 - val_accuracy: 0.9909
Epoch 6/30
60000/60000 [==============================] - 589s 10ms/step - loss:
0.0239 - accuracy: 0.9930 - val_loss: 0.0217 - val_accuracy: 0.9917
Epoch 7/30
60000/60000 [==============================] - 563s 9ms/step - loss: 0.
```

```
0216 - accuracy: 0.9933 - val_loss: 0.0274 - val_accuracy: 0.9925
Epoch 8/30
60000/60000 [==============================] - 563s 9ms/step - loss: 0.
0200 - accuracy: 0.9936 - val_loss: 0.0229 - val_accuracy: 0.9927
Epoch 9/30
60000/60000 [==============================] - 567s 9ms/step - loss: 0.
0176 - accuracy: 0.9943 - val_loss: 0.0213 - val_accuracy: 0.9940
Epoch 10/30
60000/60000 [==============================] - 613s 10ms/step - loss:
0.0170 - accuracy: 0.9947 - val_loss: 0.0242 - val_accuracy: 0.9931
Epoch 11/30
60000/60000 [==============================] - 625s 10ms/step - loss:
0.0141 - accuracy: 0.9953 - val_loss: 0.0223 - val_accuracy: 0.9927
Epoch 12/30
60000/60000 [==============================] - 625s 10ms/step - loss:
0.0140 - accuracy: 0.9956 - val_loss: 0.0219 - val_accuracy: 0.9933
Epoch 13/30
60000/60000 [==============================] - 623s 10ms/step - loss:
0.0121 - accuracy: 0.9964 - val_loss: 0.0231 - val_accuracy: 0.9938
Epoch 14/30
60000/60000 [==============================] - 627s 10ms/step - loss:
0.0105 - accuracy: 0.9966 - val_loss: 0.0258 - val_accuracy: 0.9934
Epoch 15/30
60000/60000 [==============================] - 619s 10ms/step - loss:
0.0107 - accuracy: 0.9966 - val_loss: 0.0201 - val_accuracy: 0.9944
Epoch 16/30
60000/60000 [==============================] - 621s 10ms/step - loss:
0.0086 - accuracy: 0.9974 - val_loss: 0.0237 - val_accuracy: 0.9944
Epoch 17/30
60000/60000 [==============================] - 611s 10ms/step - loss:
0.0091 - accuracy: 0.9969 - val_loss: 0.0257 - val_accuracy: 0.9927
Epoch 18/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0095 - accuracy: 0.9969 - val_loss: 0.0215 - val_accuracy: 0.9938
Epoch 19/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0072 - accuracy: 0.9978 - val_loss: 0.0229 - val_accuracy: 0.9939
Epoch 20/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
```

```
0061 - accuracy: 0.9979 - val_loss: 0.0213 - val_accuracy: 0.9943
Epoch 21/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0068 - accuracy: 0.9978 - val_loss: 0.0219 - val_accuracy: 0.9939
Epoch 22/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0059 - accuracy: 0.9980 - val_loss: 0.0237 - val_accuracy: 0.9937
Epoch 23/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0056 - accuracy: 0.9981 - val_loss: 0.0270 - val_accuracy: 0.9935
Epoch 24/30
60000/60000 [==============================] - 597s 10ms/step - loss:
0.0062 - accuracy: 0.9982 - val_loss: 0.0233 - val_accuracy: 0.9939
Epoch 25/30
60000/60000 [==============================] - 603s 10ms/step - loss:
0.0057 - accuracy: 0.9982 - val_loss: 0.0242 - val_accuracy: 0.9947
Epoch 26/30
60000/60000 [==============================] - 561s 9ms/step - loss: 0.
0051 - accuracy: 0.9985 - val_loss: 0.0251 - val_accuracy: 0.9944
Epoch 27/30
60000/60000 [==============================] - 560s 9ms/step - loss: 0.
0050 - accuracy: 0.9982 - val_loss: 0.0236 - val_accuracy: 0.9950
Epoch 28/30
60000/60000 [==============================] - 561s 9ms/step - loss: 0.
0036 - accuracy: 0.9988 - val_loss: 0.0205 - val_accuracy: 0.9940
Epoch 29/30
60000/60000 [==============================] - 561s 9ms/step - loss: 0.
0034 - accuracy: 0.9989 - val_loss: 0.0219 - val_accuracy: 0.9947
Epoch 30/30
60000/60000 [==============================] - 561s 9ms/step - loss: 0.
0041 - accuracy: 0.9985 - val_loss: 0.0256 - val_accuracy: 0.9946
```

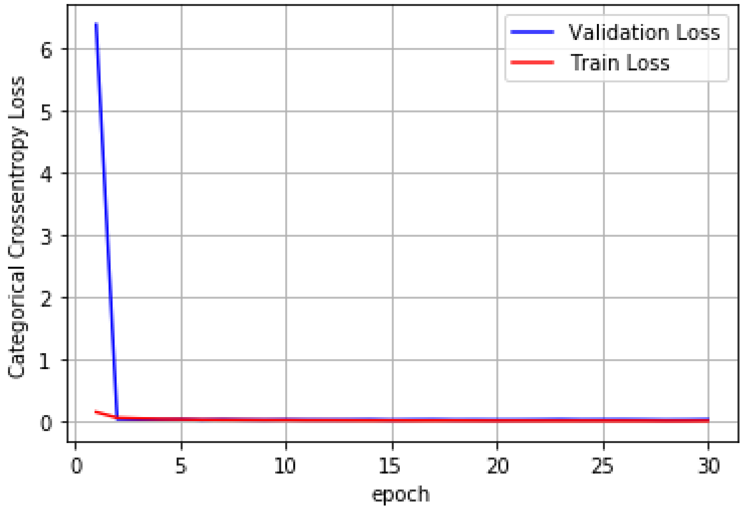
2.1.3 Number of Epochs vs Train Loss & Validation Loss

In [11]:

```
score2 = model2.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score2[0])
print('Test accuracy:', score2[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

# we will get val_loss and val_acc only when you pass the parameter val
idation_data
# val_loss : validation loss
# val_acc : validation accuracy

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M2_history.history['val_loss']
ty = M2_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.025601864825535336
Test accuracy: 0.9945999979972839
```



2.2 Model 3 :

2.2.1 Constructing the Neural Network

In [12]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model3 = Sequential()
model3.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu', kernel_initializer='he_uniform',
                 input_shape=input_shape, padding='same'))
model3.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.4))
```

```
model3.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.4))

model3.add(Flatten())

model3.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(256, activation='relu',kernel_initializer='he_uniform'
))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(64, activation='relu',kernel_initializer='he_uniform'
))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))

model3.add(Dense(num_classes, activation='softmax',kernel_initializer=
'he_uniform'))
model3.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
```

```
batch_normalization_5 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 64)        36928
_________________________________________________________________
batch_normalization_6 (Batch (None, 14, 14, 64)        256
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout_7 (Dropout)          (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               1606144
_________________________________________________________________
batch_normalization_7 (Batch (None, 512)               2048
_________________________________________________________________
dropout_8 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024
_________________________________________________________________
dropout_9 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512
_________________________________________________________________
dropout_10 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_10 (Batc (None, 64)                256
_________________________________________________________________
dropout_11 (Dropout)         (None, 64)                0
_________________________________________________________________
```

```
dense_10 (Dense)             (None, 10)                650
=================================================================
Total params: 1,858,122
Trainable params: 1,855,946

Non-trainable params: 2,176
_________________________________________________________________
```

2.2.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [13]:

```python
model3.compile(loss=keras.losses.categorical_crossentropy, optimizer=optimizers.Adam(lr=0.001),
              metrics=['accuracy'])

M3_history = model3.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 1/30
60000/60000 [==============================] - 309s 5ms/step - loss: 0.8541 - accuracy: 0.7377 - val_loss: 0.1921 - val_accuracy: 0.9421
Epoch 2/30
60000/60000 [==============================] - 305s 5ms/step - loss: 0.2103 - accuracy: 0.9434 - val_loss: 0.0673 - val_accuracy: 0.9807
Epoch 3/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.1360 - accuracy: 0.9647 - val_loss: 0.0527 - val_accuracy: 0.9846
Epoch 4/30
60000/60000 [==============================] - 296s 5ms/step - loss: 0.1158 - accuracy: 0.9695 - val_loss: 0.0461 - val_accuracy: 0.9877
Epoch 5/30
60000/60000 [==============================] - 295s 5ms/step - loss: 0.1000 - accuracy: 0.9749 - val_loss: 0.0330 - val_accuracy: 0.9909
Epoch 6/30
60000/60000 [==============================] - 295s 5ms/step - loss: 0.0862 - accuracy: 0.9784 - val_loss: 0.0372 - val_accuracy: 0.9902
Epoch 7/30
60000/60000 [==============================] - 295s 5ms/step - loss: 0.
```

```
60000/60000 [==============================] - 295s 5ms/step - loss: 0.
0752 - accuracy: 0.9804 - val_loss: 0.0298 - val_accuracy: 0.9916
Epoch 8/30
60000/60000 [==============================] - 296s 5ms/step - loss: 0.
0759 - accuracy: 0.9810 - val_loss: 0.0340 - val_accuracy: 0.9907
Epoch 9/30
60000/60000 [==============================] - 296s 5ms/step - loss: 0.
0641 - accuracy: 0.9834 - val_loss: 0.0304 - val_accuracy: 0.9930
Epoch 10/30
60000/60000 [==============================] - 296s 5ms/step - loss: 0.
0622 - accuracy: 0.9840 - val_loss: 0.0277 - val_accuracy: 0.9927
Epoch 11/30
60000/60000 [==============================] - 296s 5ms/step - loss: 0.
0607 - accuracy: 0.9847 - val_loss: 0.0297 - val_accuracy: 0.9920
Epoch 12/30
60000/60000 [==============================] - 297s 5ms/step - loss: 0.
0562 - accuracy: 0.9859 - val_loss: 0.0238 - val_accuracy: 0.9932
Epoch 13/30
60000/60000 [==============================] - 299s 5ms/step - loss: 0.
0512 - accuracy: 0.9867 - val_loss: 0.0239 - val_accuracy: 0.9938
Epoch 14/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.
0523 - accuracy: 0.9867 - val_loss: 0.0273 - val_accuracy: 0.9921
Epoch 15/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.
0460 - accuracy: 0.9885 - val_loss: 0.0209 - val_accuracy: 0.9948
Epoch 16/30
60000/60000 [==============================] - 302s 5ms/step - loss: 0.
0453 - accuracy: 0.9885 - val_loss: 0.0316 - val_accuracy: 0.9921
Epoch 17/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.
0426 - accuracy: 0.9893 - val_loss: 0.0228 - val_accuracy: 0.9941
Epoch 18/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.
0420 - accuracy: 0.9898 - val_loss: 0.0283 - val_accuracy: 0.9934
Epoch 19/30
60000/60000 [==============================] - 302s 5ms/step - loss: 0.
0403 - accuracy: 0.9901 - val_loss: 0.0238 - val_accuracy: 0.9940
Epoch 20/30
60000/60000 [==============================] - 302s 5ms/step - loss: 0.
```

```
0405 - accuracy: 0.9895 - val_loss: 0.0198 - val_accuracy: 0.9947
Epoch 21/30
60000/60000 [==============================] - 301s 5ms/step - loss: 0.
0368 - accuracy: 0.9908 - val_loss: 0.0184 - val_accuracy: 0.9945
Epoch 22/30
60000/60000 [==============================] - 303s 5ms/step - loss: 0.
0363 - accuracy: 0.9911 - val_loss: 0.0249 - val_accuracy: 0.9937
Epoch 23/30
60000/60000 [==============================] - 299s 5ms/step - loss: 0.
0358 - accuracy: 0.9909 - val_loss: 0.0236 - val_accuracy: 0.9945
Epoch 24/30
60000/60000 [==============================] - 291s 5ms/step - loss: 0.
0352 - accuracy: 0.9912 - val_loss: 0.0191 - val_accuracy: 0.9958
Epoch 25/30
60000/60000 [==============================] - 292s 5ms/step - loss: 0.
0293 - accuracy: 0.9926 - val_loss: 0.0171 - val_accuracy: 0.9958
Epoch 26/30
60000/60000 [==============================] - 291s 5ms/step - loss: 0.
0333 - accuracy: 0.9912 - val_loss: 0.0163 - val_accuracy: 0.9952
Epoch 27/30
60000/60000 [==============================] - 290s 5ms/step - loss: 0.
0327 - accuracy: 0.9916 - val_loss: 0.0218 - val_accuracy: 0.9953
Epoch 28/30
60000/60000 [==============================] - 291s 5ms/step - loss: 0.
0307 - accuracy: 0.9923 - val_loss: 0.0244 - val_accuracy: 0.9943
Epoch 29/30
60000/60000 [==============================] - 291s 5ms/step - loss: 0.
0287 - accuracy: 0.9926 - val_loss: 0.0188 - val_accuracy: 0.9952
Epoch 30/30
60000/60000 [==============================] - 292s 5ms/step - loss: 0.
0275 - accuracy: 0.9923 - val_loss: 0.0231 - val_accuracy: 0.9947
```

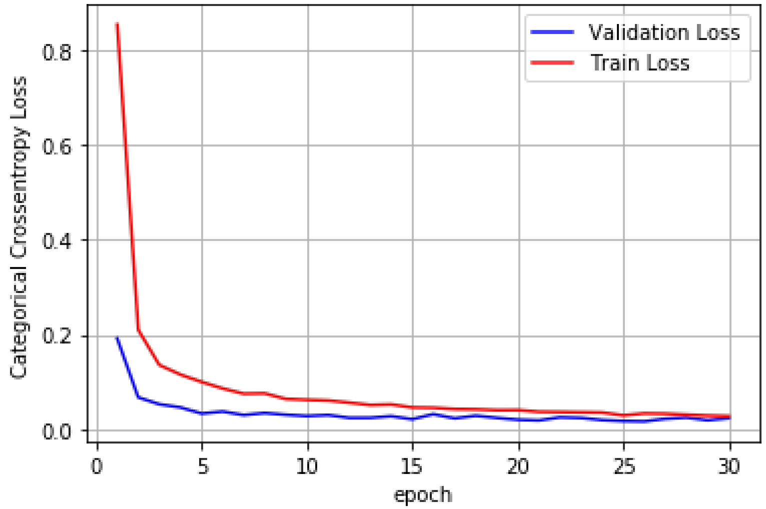
2.2.3 Number of Epochs vs Train Loss & Validation Loss

In [14]:

```
score3 = model3.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score3[0])
print('Test accuracy:', score3[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M3_history.history['val_loss']
ty = M3_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.023077930994681084
Test accuracy: 0.994700014591217
```



3. Number of Convolution Layers in the Neural Network = 5

3.1 Model 4 :

3.1.1 Constructing the Neural Network

In [5]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model4 = Sequential()
model4.add(Conv2D(64, kernel_size=(3, 3),
                  activation='relu', kernel_initializer='he_uniform',
                  input_shape=input_shape, padding='same'))
```

```
model4.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model4.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.5))

model4.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model4.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model4.add(BatchNormalization())
model4.add(MaxPooling2D(pool_size=(2, 2)))
model4.add(Dropout(0.5))

model4.add(Flatten())

model4.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))

model4.add(Dense(256, activation='relu',kernel_initializer='he_uniform'
))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))

model4.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))

model4.add(Dense(64, activation='relu',kernel_initializer='he_uniform'
))
model4.add(BatchNormalization())
model4.add(Dropout(0.5))

model4.add(Dense(num_classes, activation='softmax',kernel_initializer=
```

```
'he_uniform'))
model4.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
batch_normalization_3 (Batch (None, 512)               2048
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)               1024
```

```
dropout_4 (Dropout)          (None, 256)              0
_________________________________________________________________
dense_3 (Dense)              (None, 128)              32896
_________________________________________________________________
batch_normalization_5 (Batch (None, 128)              512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)              0
_________________________________________________________________
dense_4 (Dense)              (None, 64)               8256
_________________________________________________________________
batch_normalization_6 (Batch (None, 64)               256
_________________________________________________________________
dropout_6 (Dropout)          (None, 64)               0
_________________________________________________________________
dense_5 (Dense)              (None, 10)               650
=================================================================
Total params: 3,685,450
Trainable params: 3,683,146
Non-trainable params: 2,304
_________________________________________________________________
```

3.1.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [6]:
```
from keras import optimizers
model4.compile(loss=keras.losses.categorical_crossentropy, optimizer=op
timizers.Adam(lr=0.001),
              metrics=['accuracy'])

M4_history = model4.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 1/30
60000/60000 [==============================] - 603s 10ms/step - loss:
0.8482 - accuracy: 0.7383 - val_loss: 0.0863 - val_accuracy: 0.9739
```

```
Epoch 2/30
60000/60000 [==============================] - 596s 10ms/step - loss:
0.1835 - accuracy: 0.9514 - val_loss: 0.0493 - val_accuracy: 0.9857
Epoch 3/30
60000/60000 [==============================] - 628s 10ms/step - loss:
0.1292 - accuracy: 0.9658 - val_loss: 0.0562 - val_accuracy: 0.9834
Epoch 4/30
60000/60000 [==============================] - 657s 11ms/step - loss:
0.1042 - accuracy: 0.9736 - val_loss: 0.0463 - val_accuracy: 0.9859
Epoch 5/30
60000/60000 [==============================] - 655s 11ms/step - loss:
0.0893 - accuracy: 0.9772 - val_loss: 0.0347 - val_accuracy: 0.9906
Epoch 6/30
60000/60000 [==============================] - 654s 11ms/step - loss:
0.0767 - accuracy: 0.9807 - val_loss: 0.0303 - val_accuracy: 0.9926
Epoch 7/30
60000/60000 [==============================] - 662s 11ms/step - loss:
0.0725 - accuracy: 0.9818 - val_loss: 0.0334 - val_accuracy: 0.9912
Epoch 8/30
60000/60000 [==============================] - 652s 11ms/step - loss:
0.0634 - accuracy: 0.9839 - val_loss: 0.0327 - val_accuracy: 0.9910
Epoch 9/30
60000/60000 [==============================] - 644s 11ms/step - loss:
0.0608 - accuracy: 0.9847 - val_loss: 0.0254 - val_accuracy: 0.9937
Epoch 10/30
60000/60000 [==============================] - 653s 11ms/step - loss:
0.0561 - accuracy: 0.9852 - val_loss: 0.0251 - val_accuracy: 0.9938
Epoch 11/30
60000/60000 [==============================] - 634s 11ms/step - loss:
0.0538 - accuracy: 0.9865 - val_loss: 0.0268 - val_accuracy: 0.9931
Epoch 12/30
60000/60000 [==============================] - 607s 10ms/step - loss:
0.0508 - accuracy: 0.9874 - val_loss: 0.0232 - val_accuracy: 0.9940
Epoch 13/30
60000/60000 [==============================] - 600s 10ms/step - loss:
0.0468 - accuracy: 0.9886 - val_loss: 0.0254 - val_accuracy: 0.9925
Epoch 14/30
60000/60000 [==============================] - 601s 10ms/step - loss:
0.0446 - accuracy: 0.9892 - val_loss: 0.0304 - val_accuracy: 0.9923
Epoch 15/30
```

```
60000/60000 [==============================] - 610s 10ms/step - loss:
0.0427 - accuracy: 0.9893 - val_loss: 0.0226 - val_accuracy: 0.9939
Epoch 16/30
60000/60000 [==============================] - 634s 11ms/step - loss:
0.0425 - accuracy: 0.9893 - val_loss: 0.0294 - val_accuracy: 0.9925
Epoch 17/30
60000/60000 [==============================] - 604s 10ms/step - loss:
0.0394 - accuracy: 0.9903 - val_loss: 0.0234 - val_accuracy: 0.9942
Epoch 18/30
60000/60000 [==============================] - 601s 10ms/step - loss:
0.0366 - accuracy: 0.9907 - val_loss: 0.0254 - val_accuracy: 0.9942
Epoch 19/30
60000/60000 [==============================] - 602s 10ms/step - loss:
0.0369 - accuracy: 0.9903 - val_loss: 0.0208 - val_accuracy: 0.9946
Epoch 20/30
60000/60000 [==============================] - 597s 10ms/step - loss:
0.0346 - accuracy: 0.9913 - val_loss: 0.0226 - val_accuracy: 0.9942
Epoch 21/30
60000/60000 [==============================] - 625s 10ms/step - loss:
0.0329 - accuracy: 0.9919 - val_loss: 0.0194 - val_accuracy: 0.9958
Epoch 22/30
60000/60000 [==============================] - 655s 11ms/step - loss:
0.0313 - accuracy: 0.9920 - val_loss: 0.0335 - val_accuracy: 0.9924
Epoch 23/30
60000/60000 [==============================] - 664s 11ms/step - loss:
0.0310 - accuracy: 0.9924 - val_loss: 0.0230 - val_accuracy: 0.9947
Epoch 24/30
60000/60000 [==============================] - 656s 11ms/step - loss:
0.0293 - accuracy: 0.9928 - val_loss: 0.0199 - val_accuracy: 0.9949
Epoch 25/30
60000/60000 [==============================] - 650s 11ms/step - loss:
0.0285 - accuracy: 0.9930 - val_loss: 0.0193 - val_accuracy: 0.9957
Epoch 26/30
60000/60000 [==============================] - 630s 11ms/step - loss:
0.0260 - accuracy: 0.9934 - val_loss: 0.0179 - val_accuracy: 0.9958
Epoch 27/30
60000/60000 [==============================] - 598s 10ms/step - loss:
0.0268 - accuracy: 0.9928 - val_loss: 0.0178 - val_accuracy: 0.9957
Epoch 28/30
```

```
60000/60000 [==============================] - 599s 10ms/step - loss:
0.0263 - accuracy: 0.9932 - val_loss: 0.0208 - val_accuracy: 0.9951
Epoch 29/30
60000/60000 [==============================] - 617s 10ms/step - loss:
0.0253 - accuracy: 0.9936 - val_loss: 0.0195 - val_accuracy: 0.9949
Epoch 30/30
60000/60000 [==============================] - 657s 11ms/step - loss:
0.0217 - accuracy: 0.9947 - val_loss: 0.0250 - val_accuracy: 0.9941
```

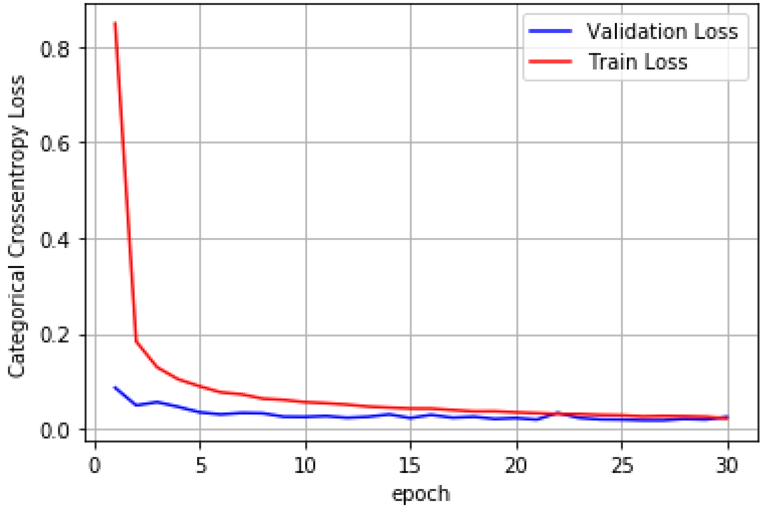
3.1.3 Number of Epochs vs Train Loss & Validation Loss

In [7]:

```
score4 = model4.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score4[0])
print('Test accuracy:', score4[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M4_history.history['val_loss']
```

```
ty = M4_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.024979012705250353
Test accuracy: 0.9940999746322632
```



3.2 Model 5 :

3.2.1 Constructing the Neural Network

In [8]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model5 = Sequential()
model5.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu', kernel_initializer='he_uniform',
                 input_shape=input_shape, padding='same'))
model5.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model5.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
```

```
itializer='he_uniform', padding='same'))
model5.add(BatchNormalization())
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.5))

model5.add(Conv2D(128, kernel_size = (5, 5), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model5.add(Conv2D(128, kernel_size = (5, 5), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model5.add(BatchNormalization())
model5.add(MaxPooling2D(pool_size=(2, 2)))
model5.add(Dropout(0.5))

model5.add(Flatten())

model5.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(256, activation='relu',kernel_initializer='he_uniform'
))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(64, activation='relu',kernel_initializer='he_uniform'
))
model5.add(BatchNormalization())
model5.add(Dropout(0.5))

model5.add(Dense(num_classes, activation='softmax',kernel_initializer=
'he_uniform'))
model5.summary()
```

```
Model: "sequential_2"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 28, 28, 64)        640

_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_7 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 14, 14, 128)       204928
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 128)       409728
_________________________________________________________________
batch_normalization_8 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_8 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               3211776
_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_10 (Batc (None, 256)               1024
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
```

```
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_11 (Batc (None, 128)               512
_________________________________________________________________
dropout_11 (Dropout)         (None, 128)               0

_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_12 (Batc (None, 64)                256
_________________________________________________________________
dropout_12 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                650
=================================================================
Total params: 4,078,666
Trainable params: 4,076,362
Non-trainable params: 2,304

_________________________________________________________________
```

3.2.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [9]:

```
model5.compile(loss=keras.losses.categorical_crossentropy, optimizer=op
timizers.Adam(lr=0.001),
              metrics=['accuracy'])

M5_history = model5.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 1/30
60000/60000 [==============================] - 1009s 17ms/step - loss:
0.7737 - accuracy: 0.7653 - val_loss: 0.0875 - val_accuracy: 0.9749
Epoch 2/30
60000/60000 [==============================] - 996s 17ms/step - loss:
0.1541 - accuracy: 0.9608 - val_loss: 0.1129 - val_accuracy: 0.9670
Epoch 3/30
60000/60000 [==============================] - 966s 16ms/step - loss:
```

```
0.1084 - accuracy: 0.9724 - val_loss: 0.0568 - val_accuracy: 0.9861
Epoch 4/30
60000/60000 [==============================] - 942s 16ms/step - loss:
0.0830 - accuracy: 0.9790 - val_loss: 0.1067 - val_accuracy: 0.9729
Epoch 5/30
60000/60000 [==============================] - 983s 16ms/step - loss:
0.0756 - accuracy: 0.9813 - val_loss: 0.0289 - val_accuracy: 0.9917
Epoch 6/30
60000/60000 [==============================] - 992s 17ms/step - loss:
0.0634 - accuracy: 0.9851 - val_loss: 0.0331 - val_accuracy: 0.9901
Epoch 7/30
60000/60000 [==============================] - 1011s 17ms/step - loss:
0.0581 - accuracy: 0.9860 - val_loss: 0.0295 - val_accuracy: 0.9925
Epoch 8/30
60000/60000 [==============================] - 1054s 18ms/step - loss:
0.0575 - accuracy: 0.9859 - val_loss: 0.0238 - val_accuracy: 0.9931
Epoch 9/30
60000/60000 [==============================] - 1016s 17ms/step - loss:
0.0511 - accuracy: 0.9876 - val_loss: 0.0371 - val_accuracy: 0.9892
Epoch 10/30
60000/60000 [==============================] - 1001s 17ms/step - loss:
0.0465 - accuracy: 0.9882 - val_loss: 0.0266 - val_accuracy: 0.9936
Epoch 11/30
60000/60000 [==============================] - 992s 17ms/step - loss:
0.0456 - accuracy: 0.9888 - val_loss: 0.0214 - val_accuracy: 0.9943
Epoch 12/30
60000/60000 [==============================] - 989s 16ms/step - loss:
0.0455 - accuracy: 0.9890 - val_loss: 0.0228 - val_accuracy: 0.9946
Epoch 13/30
60000/60000 [==============================] - 1011s 17ms/step - loss:
0.0381 - accuracy: 0.9907 - val_loss: 0.0260 - val_accuracy: 0.9935
Epoch 14/30
60000/60000 [==============================] - 1014s 17ms/step - loss:
0.0359 - accuracy: 0.9913 - val_loss: 0.0271 - val_accuracy: 0.9933
Epoch 15/30
60000/60000 [==============================] - 999s 17ms/step - loss:
0.0370 - accuracy: 0.9912 - val_loss: 0.0188 - val_accuracy: 0.9951
Epoch 16/30
60000/60000 [==============================] - 1000s 17ms/step - loss:
0.0355 - accuracy: 0.9915 - val_loss: 0.0204 - val_accuracy: 0.9947
```

```
Epoch 17/30
60000/60000 [==============================] - 1030s 17ms/step - loss:
0.0324 - accuracy: 0.9920 - val_loss: 0.0283 - val_accuracy: 0.9930
Epoch 18/30
60000/60000 [==============================] - 997s 17ms/step - loss:
0.0314 - accuracy: 0.9919 - val_loss: 0.0184 - val_accuracy: 0.9950
Epoch 19/30
60000/60000 [==============================] - 1023s 17ms/step - loss:
0.0277 - accuracy: 0.9930 - val_loss: 0.0211 - val_accuracy: 0.9949
Epoch 20/30
60000/60000 [==============================] - 1038s 17ms/step - loss:
0.0296 - accuracy: 0.9925 - val_loss: 0.0166 - val_accuracy: 0.9961
Epoch 21/30
60000/60000 [==============================] - 1010s 17ms/step - loss:
0.0251 - accuracy: 0.9939 - val_loss: 0.0203 - val_accuracy: 0.9955
Epoch 22/30
60000/60000 [==============================] - 1011s 17ms/step - loss:
0.0265 - accuracy: 0.9937 - val_loss: 0.0216 - val_accuracy: 0.9944
Epoch 23/30
60000/60000 [==============================] - 1017s 17ms/step - loss:
0.0237 - accuracy: 0.9942 - val_loss: 0.0219 - val_accuracy: 0.9946
Epoch 24/30
60000/60000 [==============================] - 1000s 17ms/step - loss:
0.0230 - accuracy: 0.9943 - val_loss: 0.0195 - val_accuracy: 0.9958
Epoch 25/30
60000/60000 [==============================] - 1019s 17ms/step - loss:
0.0216 - accuracy: 0.9945 - val_loss: 0.0175 - val_accuracy: 0.9954
Epoch 26/30
60000/60000 [==============================] - 1001s 17ms/step - loss:
0.0237 - accuracy: 0.9946 - val_loss: 0.0222 - val_accuracy: 0.9945
Epoch 27/30
60000/60000 [==============================] - 1027s 17ms/step - loss:
0.0242 - accuracy: 0.9942 - val_loss: 0.0158 - val_accuracy: 0.9965
Epoch 28/30
60000/60000 [==============================] - 1001s 17ms/step - loss:
0.0180 - accuracy: 0.9955 - val_loss: 0.0179 - val_accuracy: 0.9961
Epoch 29/30
60000/60000 [==============================] - 999s 17ms/step - loss:
0.0195 - accuracy: 0.9952 - val_loss: 0.0246 - val_accuracy: 0.9946
```

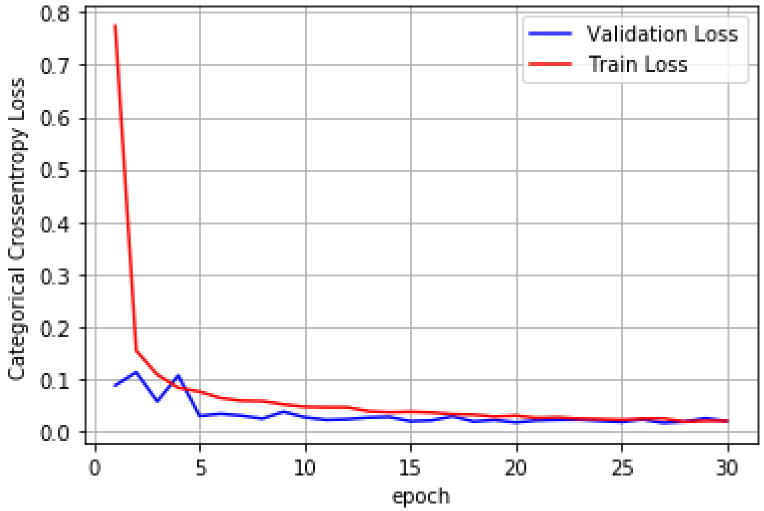
```
Epoch 30/30
60000/60000 [==============================] - 1001s 17ms/step - loss:
0.0195 - accuracy: 0.9953 - val_loss: 0.0189 - val_accuracy: 0.9959
```

3.2.3 Number of Epochs vs Train Loss & Validation Loss

In [10]:

```
score5 = model5.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score5[0])
print('Test accuracy:', score5[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M5_history.history['val_loss']
ty = M5_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.01890334000485018
Test accuracy: 0.9958999752998352
```



4. Number of Convolution Layers in the Neural Network = 7

4.1 Model 6 :

4.1.1 Constructing the Neural Network

In [5]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model6 = Sequential()
model6.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu', kernel_initializer='he_uniform',
                 input_shape=input_shape, padding='same'))
model6.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
```

```
itializer='he_uniform', padding='same'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2, 2)))
model6.add(Dropout(0.5))

model6.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model6.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2, 2)))
model6.add(Dropout(0.5))

model6.add(Conv2D(256, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model6.add(Conv2D(256, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model6.add(Conv2D(256, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model6.add(BatchNormalization())
model6.add(MaxPooling2D(pool_size=(2, 2)))
model6.add(Dropout(0.5))

model6.add(Flatten())

model6.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))

model6.add(Dense(256, activation='relu',kernel_initializer='he_uniform'
))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))

model6.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))
```

```
model6.add(Dense(64, activation='relu',kernel_initializer='he_uniform'
))
model6.add(BatchNormalization())
model6.add(Dropout(0.5))

model6.add(Dense(num_classes, activation='softmax',kernel_initializer=
'he_uniform'))
model6.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        640
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 7, 7, 256)         590080
_________________________________________________________________
```

```
batch_normalization_3 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 3, 3, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_7 (Batch (None, 64)                256
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 3,093,258
Trainable params: 3,090,442
Non-trainable params: 2,816
_________________________________________________________________
```

4.1.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [6]:
```
model6.compile(loss=keras.losses.categorical_crossentropy, optimizer=op
timizers.Adam(lr=0.001),
              metrics=['accuracy'])

M6_history = model6.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 1/30
60000/60000 [==============================] - 961s 16ms/step - loss:
1.3383 - accuracy: 0.5631 - val_loss: 3.9429 - val_accuracy: 0.1914
Epoch 2/30
60000/60000 [==============================] - 950s 16ms/step - loss:
0.2973 - accuracy: 0.9205 - val_loss: 0.0848 - val_accuracy: 0.9767
Epoch 3/30
60000/60000 [==============================] - 948s 16ms/step - loss:
0.1586 - accuracy: 0.9604 - val_loss: 0.0623 - val_accuracy: 0.9826
Epoch 4/30
60000/60000 [==============================] - 941s 16ms/step - loss:
0.1163 - accuracy: 0.9719 - val_loss: 0.0941 - val_accuracy: 0.9748
Epoch 5/30
60000/60000 [==============================] - 933s 16ms/step - loss:
0.0971 - accuracy: 0.9769 - val_loss: 0.0489 - val_accuracy: 0.9868
Epoch 6/30
60000/60000 [==============================] - 885s 15ms/step - loss:
0.0852 - accuracy: 0.9794 - val_loss: 0.0879 - val_accuracy: 0.9797
Epoch 7/30
60000/60000 [==============================] - 952s 16ms/step - loss:
0.0739 - accuracy: 0.9824 - val_loss: 0.0372 - val_accuracy: 0.9910
Epoch 8/30
60000/60000 [==============================] - 935s 16ms/step - loss:
0.0663 - accuracy: 0.9843 - val_loss: 0.0336 - val_accuracy: 0.9904
Epoch 9/30
60000/60000 [==============================] - 952s 16ms/step - loss:
0.0616 - accuracy: 0.9858 - val_loss: 0.0395 - val_accuracy: 0.9912
Epoch 10/30
60000/60000 [==============================] - 944s 16ms/step - loss:
```

```
0.0554 - accuracy: 0.9876 - val_loss: 0.0264 - val_accuracy: 0.9933
Epoch 11/30
60000/60000 [==============================] - 969s 16ms/step - loss:
0.0559 - accuracy: 0.9866 - val_loss: 0.0229 - val_accuracy: 0.9945
Epoch 12/30
60000/60000 [==============================] - 937s 16ms/step - loss:
0.0495 - accuracy: 0.9885 - val_loss: 0.0338 - val_accuracy: 0.9920
Epoch 13/30
60000/60000 [==============================] - 883s 15ms/step - loss:
0.0473 - accuracy: 0.9885 - val_loss: 0.0455 - val_accuracy: 0.9896
Epoch 14/30
60000/60000 [==============================] - 882s 15ms/step - loss:
0.0468 - accuracy: 0.9891 - val_loss: 0.0300 - val_accuracy: 0.9926
Epoch 15/30
60000/60000 [==============================] - 879s 15ms/step - loss:
0.0440 - accuracy: 0.9897 - val_loss: 0.0252 - val_accuracy: 0.9942
Epoch 16/30
60000/60000 [==============================] - 892s 15ms/step - loss:
0.0423 - accuracy: 0.9903 - val_loss: 0.0295 - val_accuracy: 0.9930
Epoch 17/30
60000/60000 [==============================] - 861s 14ms/step - loss:
0.0407 - accuracy: 0.9906 - val_loss: 0.0223 - val_accuracy: 0.9953
Epoch 18/30
60000/60000 [==============================] - 863s 14ms/step - loss:
0.0400 - accuracy: 0.9904 - val_loss: 0.0283 - val_accuracy: 0.9929
Epoch 19/30
60000/60000 [==============================] - 875s 15ms/step - loss:
0.0381 - accuracy: 0.9914 - val_loss: 0.0321 - val_accuracy: 0.9922
Epoch 20/30
60000/60000 [==============================] - 890s 15ms/step - loss:
0.0348 - accuracy: 0.9921 - val_loss: 0.0275 - val_accuracy: 0.9937
Epoch 21/30
60000/60000 [==============================] - 938s 16ms/step - loss:
0.0320 - accuracy: 0.9923 - val_loss: 0.0257 - val_accuracy: 0.9940
Epoch 22/30
60000/60000 [==============================] - 939s 16ms/step - loss:
0.0294 - accuracy: 0.9931 - val_loss: 0.0235 - val_accuracy: 0.9942
Epoch 23/30
60000/60000 [==============================] - 889s 15ms/step - loss:
```

```
0.0320 - accuracy: 0.9927 - val_loss: 0.0178 - val_accuracy: 0.9958
Epoch 24/30
60000/60000 [==============================] - 814s 14ms/step - loss:
0.0307 - accuracy: 0.9930 - val_loss: 0.0208 - val_accuracy: 0.9949
Epoch 25/30
60000/60000 [==============================] - 814s 14ms/step - loss:
0.0274 - accuracy: 0.9937 - val_loss: 0.0167 - val_accuracy: 0.9952
Epoch 26/30
60000/60000 [==============================] - 814s 14ms/step - loss:
0.0270 - accuracy: 0.9938 - val_loss: 0.0268 - val_accuracy: 0.9940
Epoch 27/30
60000/60000 [==============================] - 823s 14ms/step - loss:
0.0254 - accuracy: 0.9942 - val_loss: 0.0227 - val_accuracy: 0.9952
Epoch 28/30
60000/60000 [==============================] - 890s 15ms/step - loss:
0.0243 - accuracy: 0.9946 - val_loss: 0.0244 - val_accuracy: 0.9951
Epoch 29/30
60000/60000 [==============================] - 890s 15ms/step - loss:
0.0213 - accuracy: 0.9954 - val_loss: 0.0292 - val_accuracy: 0.9930
Epoch 30/30
60000/60000 [==============================] - 815s 14ms/step - loss:
0.0231 - accuracy: 0.9950 - val_loss: 0.0186 - val_accuracy: 0.9960
```

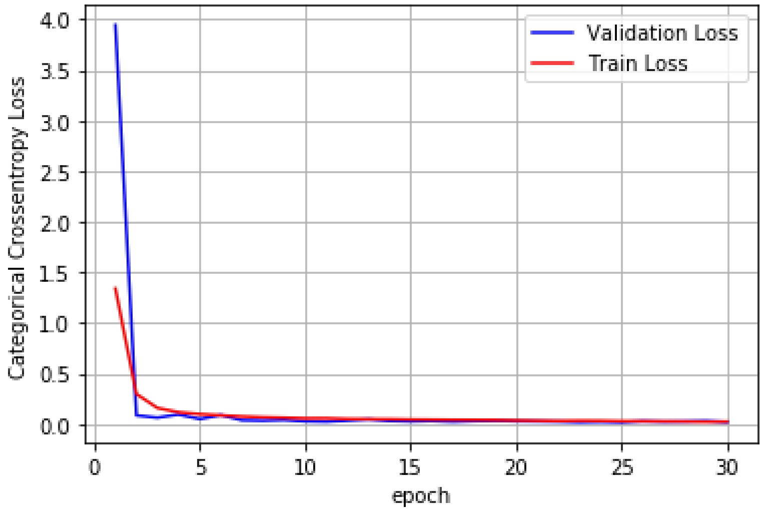
4.1.3 Number of Epochs vs Train Loss & Validation Loss

In [7]:

```
score6 = model6.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score6[0])
print('Test accuracy:', score6[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M6_history.history['val_loss']
ty = M6_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.018550323320517783
Test accuracy: 0.9959999918937683
```



4.2 Model 7 :

4.2.1 Constructing the Neural Network

In [7]:
```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout

model7 = Sequential()
model7.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu', kernel_initializer='he_uniform',
                 input_shape=input_shape, padding='same'))
model7.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model7.add(Conv2D(64, kernel_size = (3, 3), activation='relu',kernel_in
itializer='he_uniform', padding='same'))
model7.add(BatchNormalization())
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(Dropout(0.5))

model7.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model7.add(Conv2D(128, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model7.add(BatchNormalization())
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(Dropout(0.5))

model7.add(Conv2D(256, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model7.add(Conv2D(256, kernel_size = (3, 3), activation='relu',kernel_i
nitializer='he_uniform', padding='same'))
model7.add(BatchNormalization())
model7.add(MaxPooling2D(pool_size=(2, 2)))
model7.add(Dropout(0.5))

model7.add(Flatten())

model7.add(Dense(512, activation='relu',kernel_initializer='he_uniform'
))
model7.add(BatchNormalization())
model7.add(Dropout(0.5))
```

```
model7.add(Dense(256, activation='relu',kernel_initializer='he_uniform'
))
model7.add(BatchNormalization())
model7.add(Dropout(0.5))

model7.add(Dense(128, activation='relu',kernel_initializer='he_uniform'
))
model7.add(BatchNormalization())
model7.add(Dropout(0.5))

model7.add(Dense(64, activation='relu',kernel_initializer='he_uniform'
))
model7.add(BatchNormalization())
model7.add(Dropout(0.5))

model7.add(Dense(num_classes, activation='softmax',kernel_initializer=
'he_uniform'))
model7.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 64)        640       
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 64)        36928     
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 64)        36928     
_________________________________________________________________
batch_normalization_8 (Batch (None, 28, 28, 64)        256       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 64)        0         
_________________________________________________________________
dropout_8 (Dropout)          (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 128)       73856     
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 128)       147584    
_________________________________________________________________
```

```
batch_normalization_9 (Batch (None, 14, 14, 128)       512
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
dropout_9 (Dropout)          (None, 7, 7, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 7, 7, 256)         295168
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 7, 7, 256)         590080
_________________________________________________________________
batch_normalization_10 (Batc (None, 7, 7, 256)         1024
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
dropout_10 (Dropout)         (None, 3, 3, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_6 (Dense)              (None, 512)               1180160
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_12 (Batc (None, 256)               1024
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
batch_normalization_14 (Batc (None, 64)                256
_________________________________________________________________
```

```
dropout_14 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                650
=================================================================
Total params: 2,540,106
Trainable params: 2,537,290
Non-trainable params: 2,816
_________________________________________________________________
```

4.2.2 Running the Neural Network on Train & Validation Datasets for 30 Epochs

In [8]:
```
model7.compile(loss=keras.losses.categorical_crossentropy, optimizer=op
timizers.Adam(lr=0.001),
               metrics=['accuracy'])

M7_history = model7.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 1/30
60000/60000 [==============================] - 898s 15ms/step - loss:
1.4802 - accuracy: 0.5267 - val_loss: 0.8756 - val_accuracy: 0.7228
Epoch 2/30
60000/60000 [==============================] - 917s 15ms/step - loss:
0.3290 - accuracy: 0.9109 - val_loss: 0.1117 - val_accuracy: 0.9677
Epoch 3/30
60000/60000 [==============================] - 910s 15ms/step - loss:
0.1673 - accuracy: 0.9576 - val_loss: 0.0726 - val_accuracy: 0.9797
Epoch 4/30
60000/60000 [==============================] - 905s 15ms/step - loss:
0.1210 - accuracy: 0.9699 - val_loss: 0.0662 - val_accuracy: 0.9830
Epoch 5/30
60000/60000 [==============================] - 861s 14ms/step - loss:
0.1074 - accuracy: 0.9730 - val_loss: 0.0402 - val_accuracy: 0.9908
Epoch 6/30
60000/60000 [==============================] - 836s 14ms/step - loss:
0.0877 - accuracy: 0.9782 - val_loss: 0.0378 - val_accuracy: 0.9909
```

```
Epoch 7/30
60000/60000 [==============================] - 832s 14ms/step - loss:
0.0801 - accuracy: 0.9804 - val_loss: 0.0352 - val_accuracy: 0.9917
Epoch 8/30
60000/60000 [==============================] - 831s 14ms/step - loss:
0.0727 - accuracy: 0.9823 - val_loss: 0.0404 - val_accuracy: 0.9900
Epoch 9/30
60000/60000 [==============================] - 831s 14ms/step - loss:
0.0634 - accuracy: 0.9842 - val_loss: 0.0241 - val_accuracy: 0.9942
Epoch 10/30
60000/60000 [==============================] - 882s 15ms/step - loss:
0.0627 - accuracy: 0.9845 - val_loss: 0.0250 - val_accuracy: 0.9937
Epoch 11/30
60000/60000 [==============================] - 898s 15ms/step - loss:
0.0630 - accuracy: 0.9849 - val_loss: 0.0207 - val_accuracy: 0.9943
Epoch 12/30
60000/60000 [==============================] - 871s 15ms/step - loss:
0.0536 - accuracy: 0.9865 - val_loss: 0.0427 - val_accuracy: 0.9897
Epoch 13/30
60000/60000 [==============================] - 831s 14ms/step - loss:
0.0533 - accuracy: 0.9874 - val_loss: 0.0280 - val_accuracy: 0.9930
Epoch 14/30
60000/60000 [==============================] - 830s 14ms/step - loss:
0.0499 - accuracy: 0.9883 - val_loss: 0.0256 - val_accuracy: 0.9936
Epoch 15/30
60000/60000 [==============================] - 831s 14ms/step - loss:
0.0477 - accuracy: 0.9888 - val_loss: 0.0261 - val_accuracy: 0.9930
Epoch 16/30
60000/60000 [==============================] - 833s 14ms/step - loss:
0.0461 - accuracy: 0.9895 - val_loss: 0.0159 - val_accuracy: 0.9952
Epoch 17/30
60000/60000 [==============================] - 893s 15ms/step - loss:
0.0442 - accuracy: 0.9895 - val_loss: 0.0189 - val_accuracy: 0.9950
Epoch 18/30
60000/60000 [==============================] - 890s 15ms/step - loss:
0.0400 - accuracy: 0.9903 - val_loss: 0.0248 - val_accuracy: 0.9942
Epoch 19/30
60000/60000 [==============================] - 892s 15ms/step - loss:
0.0408 - accuracy: 0.9905 - val_loss: 0.0217 - val_accuracy: 0.9941
```

```
Epoch 20/30
60000/60000 [==============================] - 891s 15ms/step - loss:
0.0360 - accuracy: 0.9915 - val_loss: 0.0174 - val_accuracy: 0.9957
Epoch 21/30
60000/60000 [==============================] - 892s 15ms/step - loss:
0.0344 - accuracy: 0.9915 - val_loss: 0.0269 - val_accuracy: 0.9934
Epoch 22/30
60000/60000 [==============================] - 888s 15ms/step - loss:
0.0352 - accuracy: 0.9915 - val_loss: 0.0229 - val_accuracy: 0.9943
Epoch 23/30
60000/60000 [==============================] - 892s 15ms/step - loss:
0.0342 - accuracy: 0.9916 - val_loss: 0.0233 - val_accuracy: 0.9939
Epoch 24/30
60000/60000 [==============================] - 891s 15ms/step - loss:
0.0326 - accuracy: 0.9921 - val_loss: 0.0170 - val_accuracy: 0.9957
Epoch 25/30
60000/60000 [==============================] - 895s 15ms/step - loss:
0.0319 - accuracy: 0.9920 - val_loss: 0.0185 - val_accuracy: 0.9959
Epoch 26/30
60000/60000 [==============================] - 894s 15ms/step - loss:
0.0293 - accuracy: 0.9931 - val_loss: 0.0212 - val_accuracy: 0.9958
Epoch 27/30
60000/60000 [==============================] - 894s 15ms/step - loss:
0.0277 - accuracy: 0.9930 - val_loss: 0.0228 - val_accuracy: 0.9952
Epoch 28/30
60000/60000 [==============================] - 894s 15ms/step - loss:
0.0282 - accuracy: 0.9935 - val_loss: 0.0187 - val_accuracy: 0.9960
Epoch 29/30
60000/60000 [==============================] - 895s 15ms/step - loss:
0.0246 - accuracy: 0.9946 - val_loss: 0.0225 - val_accuracy: 0.9955
Epoch 30/30
60000/60000 [==============================] - 895s 15ms/step - loss:
0.0268 - accuracy: 0.9938 - val_loss: 0.0184 - val_accuracy: 0.9963
```

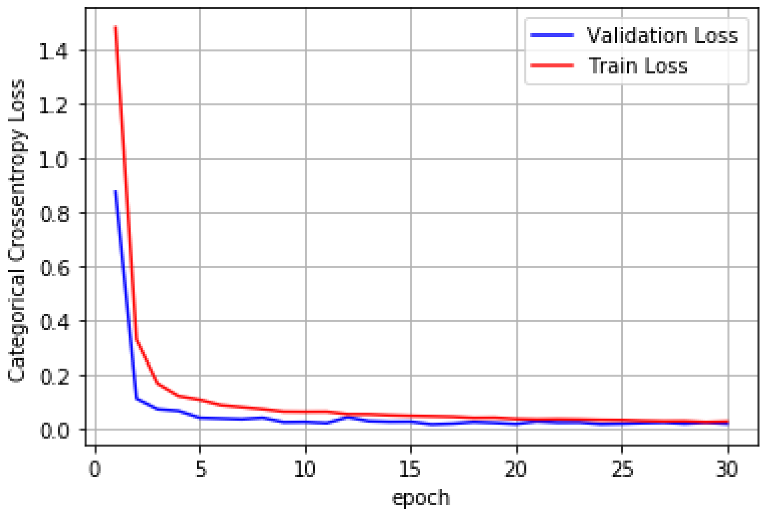
4.2.3 Number of Epochs vs Train Loss & Validation Loss

In [9]:

```
score7 = model7.evaluate(x_test, y_test, verbose=0)
```

```
print('Test loss:', score7[0])
print('Test accuracy:', score7[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, epo
chs=nb_epoch, verbose=1, validation_data=(X_test, Y_test))

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

# loss : training loss
# acc : train accuracy
# for each key in history.history we will have a list of length equal t
o number of epochs

vy = M7_history.history['val_loss']
ty = M7_history.history['loss']
plt_dynamic(x, vy, ty, ax)
```

```
Test loss: 0.018387412990181474
Test accuracy: 0.9962999820709229
```



5. Conclusion

The MNIST Dataset is a dataset of Handwritten Characters pertaining to 10 integers from 0 to 9, which is used for Training in the case of Many Image Processing Tasks. We have an input square image of size (28 px* 28 px), which makes the corresponding vector that we obtain to be 784-dimensional. After this, we obtain a Matrix of this dimensionality where each cell in the Matrix corresponds to an integral number from 0 to 255 -> The Higher is the value of this number, the darker that particular pixel value is.

There are a total of 60,000 Training Datapoints and a total of 10,000 Test Datapoints in MNIST. We build various models with various Convolution Layer Architectures in order to try and minimize our Test Accuracy and Test Log Loss values. We train each of our models on a total of 30 epochs.

First we work on an architecture with 2 Convolution layers and 1 Hidden Layer with Relu as the Activation Unit.{We are only using Relu as the Activation Unit and Adam as the optimizer with

He-Uniform initialization and a small enough learning rate of 0.001 in order to achieve the best possible convergence}. Also we are using Maxpooling, Batch Normalization as well as Dropoout whenever it is thought to be necessary. Note that here we are not tuning the Dropout Value because of the limit of time and computational resources.

This is followed by further new models with Convolutional Layers of 3,5 and 7 in number with Kernels of Various sizes. Since it is always recommended to have Kernels of odd sizes, we tried modelling with (3 *3), (5* 5) and (7 * 7) kernels and variation in the number of Hidden Layers. Also, we know that VGG-16 Architecture is state of the art. Therefore we have taken this architecture as inspiration and tried to replicate it as much as possible with our maximum 7 Convolutional Layers.

We are training all the models on a fix number of 30 epochs for each model for better comparison. However, for each model we also compare the Loss Value across different epoch numbers for Train and Validation Datasets, so as to the Epoch Number for which this loss is minimum. Along with this, we also need to ensure that the gap between the Train and Validation curves in this plot is not too big, which would indicate overfitting in such a scenario.

Also, we could have carried out Hyperparameter Tuning using Hyperas by splitting the data into Train, CV and Test but we have not done so because our dataset of 60K Training Datapoints is anyway not that big : If we split into Train and CV, Train will have only 48K Training Datapoints and 12K CV Datapoints. Also, Test anyway has 10K Datapoints. I tried following this approach but our Test Log Loss was coming out worse when we trained on 48K Datapoints.

The Summary of each of our models that we have built so far is as shown below :

In [6]:
```
from prettytable import PrettyTable

x=PrettyTable()
x.field_names=["Model #","Number of Convolutional Layers","Test Accurac
y","Test Log Loss"]

print("="*100)

x.add_row(["Model 1","Two", "99.19%","0.0323"])
```

```
x.add_row(["Model 2","Three", "99.45%","0.0256"])
x.add_row(["Model 3","Three", "99.47%","0.0230"])
x.add_row(["Model 4","Five", "99.40%","0.0249"])
x.add_row(["Model 5","Five", "99.59%","0.0189"])
x.add_row(["Model 6","Seven", "99.60%","0.0186"])
x.add_row(["Model 7","Seven", "99.63%","0.0183"])

print(x)
print('-'*100)
```

```
========================================================================
=============================
+---------+---------------------------------+--------------+-----------
----+
| Model # | Number of Convolutional Layers | Test Accuracy | Test Log L
oss |
+---------+---------------------------------+--------------+-----------
----+
| Model 1 |              Two               |    99.19%    |   0.0323
    |
| Model 2 |             Three              |    99.45%    |   0.0256
    |
| Model 3 |             Three              |    99.47%    |   0.0230
    |
| Model 4 |              Five              |    99.40%    |   0.0249
    |
| Model 5 |              Five              |    99.59%    |   0.0189
    |
| Model 6 |             Seven              |    99.60%    |   0.0186
    |
| Model 7 |             Seven              |    99.63%    |   0.0183
    |
+---------+---------------------------------+--------------+-----------
----+
------------------------------------------------------------------------
----------------------------
```