In [1]:

```python
import keras
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal
from keras import backend as K
from keras.layers import Dense, Dropout, Flatten, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
import seaborn as sns
```

```
Using TensorFlow backend.
```

In [2]:

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

In [3]:

```python
# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [4]:

```python
print("Number of training examples :", X_train.shape[0], "and each image is o
print("Number of training examples :", X_test.shape[0], "and each image is of
```

```
Number of training examples : 60000 and each image is of shape
(28, 28)
Number of training examples : 10000 and each image is of shape
(28, 28)
```

In [5]:

```
# if you observe the input shape its 2 dimensional vector
# for each image we have a (28*28) vector
# we will convert the (28*28) vector into single dimensional vector of 1 * 78

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1]*X_train.shape[2]
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

In [6]:

```
# after converting the input images from 3d to 2d vectors

print("Number of training examples :", X_train.shape[0], "and each image is o
print("Number of test examples :", X_test.shape[0], "and each image is of sha
```

```
Number of training examples : 60000 and each image is of shape
(784)
Number of test examples : 10000 and each image is of shape (78
4)
```

In [7]:

```
# An example data point
print(X_train[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   3  18  18  18 126 136 17
5  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0   3
0  36  94 154
 170 253 253 253 253 253 225 172 253 242 195  64   0   0
0   0   0   0
   0   0   0   0   0  49 238 253 253 253 253 253 253 253 25
3 251  93  82
  82  56  39   0   0   0   0   0   0   0   0   0   0   0
0  18 219 253
 253 253 253 253 198 182 247 241   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0  80 156 107 253 253 205   1
1   0  43 154
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0  14   1 154 253  90   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0 139 25
3 190   2   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0  11 190 253  70   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0  35 241
 225 160 108   1   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0  81 240 253 253 119   2
5   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
0   0   0   0
   0   0  45 186 253 253 150  27   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0  16   9
3 252 253 187
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0 249 253 249  64   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4
6 130 183 253
 253 207   2   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0  39 148 229 253 253 253 250 182   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0  24 114 22
1 253 253 253
 253 201  78   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0  23  66 213 253 253 253 253 198  81   2   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0  18 171 219 253 25
3 253 253 195
  80   9   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
  55 172 226 253 253 253 253 244 133  11   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0 136 253 253 253 21
2 135 132  16
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0
   0   0   0   0   0   0   0   0   0   0]
```

In [8]:

```python
# if we observe the above matrix each cell is having a value between 0-255
# before we move to apply machine learning algorithms lets try to normalize t
# X => (X - Xmin)/(Xmax-Xmin) = X/255

X_train = X_train/255
X_test = X_test/255
```

In [9]:

```
# example data point after normlizing
print(X_train[0])
```

```
[0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
 0.          0.          0.          0.          0.          0.
```

In [10]:

```
# here we are having a class number for each image
print("Class label of first image :", y_train[0])

# lets convert this into a 10 dimensional vector
# ex: consider an image is 5 convert it into 5 => [0, 0, 0, 0, 0, 1, 0, 0, 0,
# this conversion needed for MLPs

Y_train = np_utils.to_categorical(y_train, 10)
Y_test = np_utils.to_categorical(y_test, 10)

print("After converting the output into a vector : ",Y_train[0])
```

```
Class label of first image : 5
After converting the output into a vector :  [0. 0. 0. 0. 0.
1. 0. 0. 0. 0.]
```

In [11]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation

# some model parameters

batch_size = 128
num_classes = 10
epochs = 20

# input image dimensions
img_rows, img_cols = 28, 28
```

In [12]:

```
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)
```

In [13]:

```
x_train[0]
```

Out[13]:

```
array([[[0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
        [0.        ],
```

Model 1 : 2 Convolutional Layers with 3X3 Filters, 1 Dense layer (ReLU Activation) and Adam Optimizer

In [14]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac
history = model.fit(x_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, Y_test))
```

```
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
==============================================================
===
```

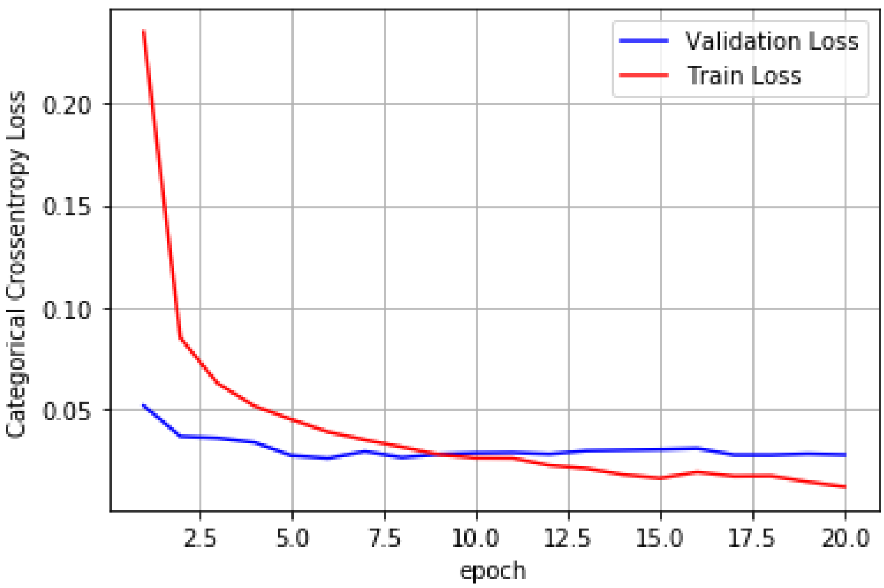
```
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
___
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 82us/step -
loss: 0.2350 - accuracy: 0.9279 - val_loss: 0.0516 - val_accur
acy: 0.9830
Epoch 2/20
60000/60000 [==============================] - 3s 51us/step -
loss: 0.0848 - accuracy: 0.9741 - val_loss: 0.0365 - val_accur
acy: 0.9879
Epoch 3/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0626 - accuracy: 0.9808 - val_loss: 0.0357 - val_accur
acy: 0.9886
Epoch 4/20
60000/60000 [==============================] - 3s 53us/step -
loss: 0.0514 - accuracy: 0.9841 - val_loss: 0.0336 - val_accur
acy: 0.9878
Epoch 5/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0448 - accuracy: 0.9861 - val_loss: 0.0272 - val_accur
acy: 0.9911
Epoch 6/20
60000/60000 [==============================] - 3s 53us/step -
loss: 0.0387 - accuracy: 0.9876 - val_loss: 0.0258 - val_accur
acy: 0.9909
Epoch 7/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0348 - accuracy: 0.9888 - val_loss: 0.0292 - val_accur
acy: 0.9908
Epoch 8/20
60000/60000 [==============================] - 3s 51us/step -
loss: 0.0313 - accuracy: 0.9898 - val_loss: 0.0262 - val_accur
acy: 0.9920
Epoch 9/20
60000/60000 [==============================] - 3s 51us/step -
loss: 0.0276 - accuracy: 0.9911 - val_loss: 0.0278 - val_accur
acy: 0.9919
Epoch 10/20
60000/60000 [==============================] - 3s 51us/step -
loss: 0.0260 - accuracy: 0.9917 - val_loss: 0.0284 - val_accur
acy: 0.9920
Epoch 11/20
60000/60000 [==============================] - 3s 53us/step -
loss: 0.0258 - accuracy: 0.9918 - val_loss: 0.0286 - val_accur
acy: 0.9920
Epoch 12/20
60000/60000 [==============================] - 3s 53us/step -
```

```
loss: 0.0223 - accuracy: 0.9927 - val_loss: 0.0278 - val_accur
acy: 0.9924
Epoch 13/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0207 - accuracy: 0.9930 - val_loss: 0.0295 - val_accur
acy: 0.9927
Epoch 14/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0177 - accuracy: 0.9940 - val_loss: 0.0298 - val_accur
acy: 0.9926
Epoch 15/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0160 - accuracy: 0.9945 - val_loss: 0.0301 - val_accur
acy: 0.9918
Epoch 16/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0190 - accuracy: 0.9938 - val_loss: 0.0306 - val_accur
acy: 0.9929
Epoch 17/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0170 - accuracy: 0.9942 - val_loss: 0.0274 - val_accur
acy: 0.9932
Epoch 18/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0171 - accuracy: 0.9942 - val_loss: 0.0274 - val_accur
acy: 0.9931
Epoch 19/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0142 - accuracy: 0.9955 - val_loss: 0.0280 - val_accur
acy: 0.9927
Epoch 20/20
60000/60000 [==============================] - 3s 52us/step -
loss: 0.0118 - accuracy: 0.9962 - val_loss: 0.0275 - val_accur
acy: 0.9936
```

In [15]:

```
score = model.evaluate(x_test, Y_test, verbose=0)
score1 = score[0]
accuracy1 = score[1]
print('Test score:', score1)
print('Test accuracy:', accuracy1)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

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

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

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


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

```
Test score: 0.027506851389828297
Test accuracy: 0.9936000108718872
```



Model 2 : 2 Convolutional Layers with 5X5 Filters, 2 Dense layers (ReLU Activation) and Adam Optimizer

In [16]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5,5), padding='same', activation='relu', ir
model.add(Conv2D(64, (5,5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac
history = model.fit(x_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, Y_test))
```

```
Model: "sequential_2"
_________________________________________________________________
___
Layer (type)                 Output Shape              Param #
=================================================================
===
conv2d_3 (Conv2D)            (None, 28, 28, 32)        832
_________________________________________________________________
___
conv2d_4 (Conv2D)            (None, 24, 24, 64)        51264
_________________________________________________________________
___
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
___
dropout_3 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
___
flatten_2 (Flatten)          (None, 9216)              0
_________________________________________________________________
___
dense_3 (Dense)              (None, 256)               2359552
_________________________________________________________________
___
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
___
dense_4 (Dense)              (None, 256)               65792
_________________________________________________________________
```

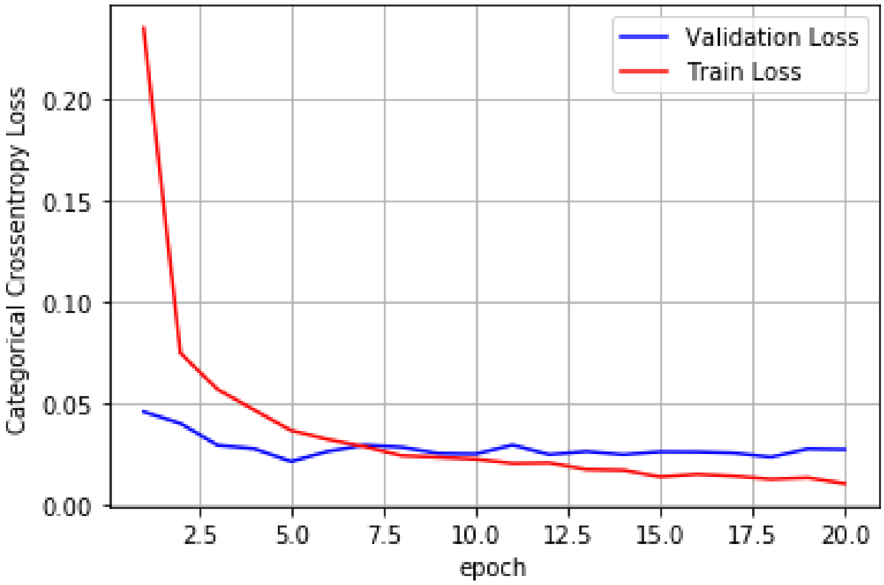
```
___
batch_normalization_1 (Batch (None, 256)              1024
_________________________________________________________________
___
dropout_5 (Dropout)          (None, 256)              0
_________________________________________________________________
___
dense_5 (Dense)              (None, 10)               2570
=================================================================
===
Total params: 2,481,034
Trainable params: 2,480,522
Non-trainable params: 512
_________________________________________________________________
___
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 77us/step -
loss: 0.2347 - accuracy: 0.9287 - val_loss: 0.0455 - val_accur
acy: 0.9859
Epoch 2/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0744 - accuracy: 0.9793 - val_loss: 0.0397 - val_accur
acy: 0.9880
Epoch 3/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0566 - accuracy: 0.9827 - val_loss: 0.0290 - val_accur
acy: 0.9909
Epoch 4/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0463 - accuracy: 0.9863 - val_loss: 0.0272 - val_accur
acy: 0.9916
Epoch 5/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0362 - accuracy: 0.9892 - val_loss: 0.0209 - val_accur
acy: 0.9935
Epoch 6/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0318 - accuracy: 0.9905 - val_loss: 0.0259 - val_accur
acy: 0.9920
Epoch 7/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0282 - accuracy: 0.9913 - val_loss: 0.0290 - val_accur
acy: 0.9920
Epoch 8/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0238 - accuracy: 0.9925 - val_loss: 0.0280 - val_accur
acy: 0.9922
Epoch 9/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0230 - accuracy: 0.9933 - val_loss: 0.0249 - val_accur
acy: 0.9924
```

```
Epoch 10/20
60000/60000 [==============================] - 4s 66us/step -
loss: 0.0221 - accuracy: 0.9932 - val_loss: 0.0246 - val_accur
acy: 0.9921
Epoch 11/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0200 - accuracy: 0.9941 - val_loss: 0.0291 - val_accur
acy: 0.9934
Epoch 12/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0202 - accuracy: 0.9937 - val_loss: 0.0243 - val_accur
acy: 0.9939
Epoch 13/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0170 - accuracy: 0.9947 - val_loss: 0.0259 - val_accur
acy: 0.9931
Epoch 14/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0166 - accuracy: 0.9947 - val_loss: 0.0243 - val_accur
acy: 0.9932
Epoch 15/20
60000/60000 [==============================] - 4s 66us/step -
loss: 0.0133 - accuracy: 0.9963 - val_loss: 0.0258 - val_accur
acy: 0.9938
Epoch 16/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0146 - accuracy: 0.9955 - val_loss: 0.0257 - val_accur
acy: 0.9929
Epoch 17/20
60000/60000 [==============================] - 4s 66us/step -
loss: 0.0137 - accuracy: 0.9960 - val_loss: 0.0251 - val_accur
acy: 0.9941
Epoch 18/20
60000/60000 [==============================] - 4s 66us/step -
loss: 0.0121 - accuracy: 0.9962 - val_loss: 0.0231 - val_accur
acy: 0.9942
Epoch 19/20
60000/60000 [==============================] - 4s 66us/step -
loss: 0.0129 - accuracy: 0.9958 - val_loss: 0.0272 - val_accur
acy: 0.9934
Epoch 20/20
60000/60000 [==============================] - 4s 67us/step -
loss: 0.0100 - accuracy: 0.9970 - val_loss: 0.0269 - val_accur
acy: 0.9936
```

In [17]:

```
score = model.evaluate(x_test, Y_test, verbose=0)
score2 = score[0]
accuracy2 = score[1]
print('Test score:', score2)
print('Test accuracy:', accuracy2)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

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

```
Test score: 0.0268641913799965
Test accuracy: 0.9936000108718872
```



Model 3 : 2 Convolutional Layers with 11X11 Filters, 2 Dense layer (ReLU Activation) and Adam Optimizer

In [18]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(11, 11), padding='same', activation='relu',
model.add(Conv2D(64, (11, 11), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac
history = model.fit(x_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, Y_test))
```

```
Model: "sequential_3"
___________________________________________________________
______
Layer (type)                 Output Shape              Para
m #
===========================================================
======
conv2d_5 (Conv2D)            (None, 28, 28, 32)        3904
___________________________________________________________
______
conv2d_6 (Conv2D)            (None, 18, 18, 64)        2478
72
___________________________________________________________
______
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 64)          0
___________________________________________________________
______
dropout_6 (Dropout)          (None, 9, 9, 64)          0
___________________________________________________________
______
flatten_3 (Flatten)          (None, 5184)              0
___________________________________________________________
______
dense_6 (Dense)              (None, 256)               1327
360
___________________________________________________________
______
dropout_7 (Dropout)          (None, 256)               0
```

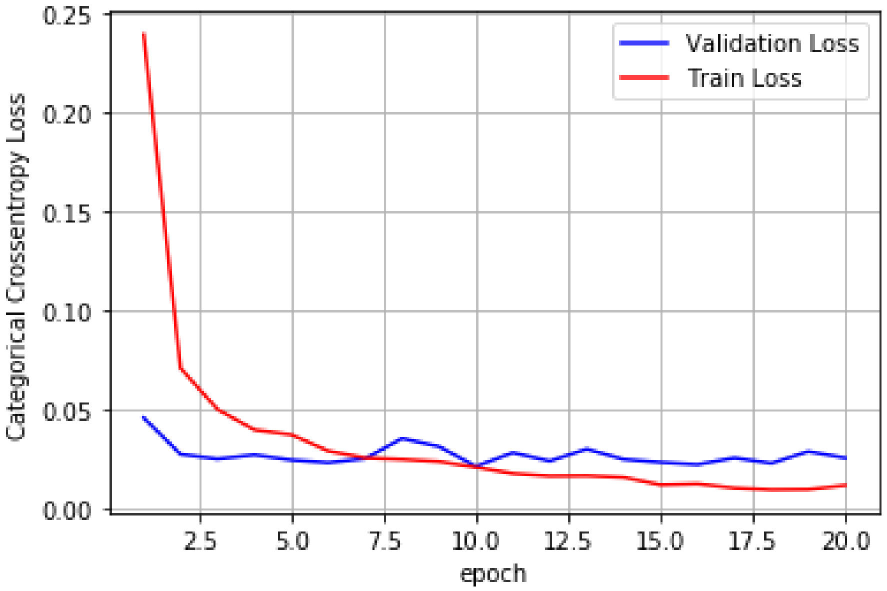
```
___________________________________________________________
______
dense_7 (Dense)              (None, 256)              6579
2

___________________________________________________________
______
batch_normalization_2 (Batch (None, 256)              1024
___________________________________________________________
______
dropout_8 (Dropout)          (None, 256)              0
___________________________________________________________
______
dense_8 (Dense)              (None, 10)               2570
===========================================================
======
Total params: 1,648,522
Trainable params: 1,648,010
Non-trainable params: 512
___________________________________________________________
______
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 5s 83us/step
- loss: 0.2395 - accuracy: 0.9278 - val_loss: 0.0454 - val_
accuracy: 0.9848
Epoch 2/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0704 - accuracy: 0.9804 - val_loss: 0.0269 - val_
accuracy: 0.9911
Epoch 3/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0496 - accuracy: 0.9856 - val_loss: 0.0245 - val_
accuracy: 0.9924
Epoch 4/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0392 - accuracy: 0.9884 - val_loss: 0.0266 - val_
accuracy: 0.9922
Epoch 5/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0368 - accuracy: 0.9894 - val_loss: 0.0240 - val_
accuracy: 0.9929
Epoch 6/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0285 - accuracy: 0.9918 - val_loss: 0.0226 - val_
accuracy: 0.9935
Epoch 7/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0250 - accuracy: 0.9925 - val_loss: 0.0247 - val_
accuracy: 0.9923
Epoch 8/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0243 - accuracy: 0.9928 - val_loss: 0.0349 - val_
```

```
accuracy: 0.9904
Epoch 9/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0232 - accuracy: 0.9931 - val_loss: 0.0308 - val_
accuracy: 0.9918
Epoch 10/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0203 - accuracy: 0.9940 - val_loss: 0.0208 - val_
accuracy: 0.9949
Epoch 11/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0171 - accuracy: 0.9950 - val_loss: 0.0277 - val_
accuracy: 0.9925
Epoch 12/20
60000/60000 [==============================] - 4s 74us/step
- loss: 0.0159 - accuracy: 0.9951 - val_loss: 0.0235 - val_
accuracy: 0.9935
Epoch 13/20
60000/60000 [==============================] - 4s 73us/step
- loss: 0.0160 - accuracy: 0.9952 - val_loss: 0.0295 - val_
accuracy: 0.9930
Epoch 14/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0152 - accuracy: 0.9952 - val_loss: 0.0243 - val_
accuracy: 0.9933
Epoch 15/20
60000/60000 [==============================] - 4s 72us/step
- loss: 0.0114 - accuracy: 0.9964 - val_loss: 0.0228 - val_
accuracy: 0.9930
Epoch 16/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0119 - accuracy: 0.9965 - val_loss: 0.0217 - val_
accuracy: 0.9936
Epoch 17/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0098 - accuracy: 0.9971 - val_loss: 0.0251 - val_
accuracy: 0.9935
Epoch 18/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0089 - accuracy: 0.9973 - val_loss: 0.0223 - val_
accuracy: 0.9940
Epoch 19/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0091 - accuracy: 0.9969 - val_loss: 0.0283 - val_
accuracy: 0.9934
Epoch 20/20
60000/60000 [==============================] - 4s 71us/step
- loss: 0.0111 - accuracy: 0.9966 - val_loss: 0.0250 - val_
accuracy: 0.9942
```

In [19]:

```
score = model.evaluate(x_test, Y_test, verbose=0)
score3 = score[0]
accuracy3 = score[1]
print('Test score:', score3)
print('Test accuracy:', accuracy3)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

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

```
Test score: 0.025038751071174595
Test accuracy: 0.9941999912261963
```



Model 4 : 2 Convolutional Layers with 11X11 Filters (No Padding), 2 Dense layer (ReLU Activation) and Adam Optimizer

In [20]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(11, 11), activation='relu', input_shape=inp
model.add(Conv2D(64, (11, 11), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac
history = model.fit(x_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, Y_test))
```

```
Model: "sequential_4"
_________________________________________________________________
______
Layer (type)                 Output Shape              Para
m #
=================================================================
======
conv2d_7 (Conv2D)            (None, 18, 18, 32)        3904
_________________________________________________________________
______
conv2d_8 (Conv2D)            (None, 8, 8, 64)          2478
72
_________________________________________________________________
______
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
______
dropout_9 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
______
flatten_4 (Flatten)          (None, 1024)              0
_________________________________________________________________
______
dense_9 (Dense)              (None, 256)               2624
00
_________________________________________________________________
______
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
```

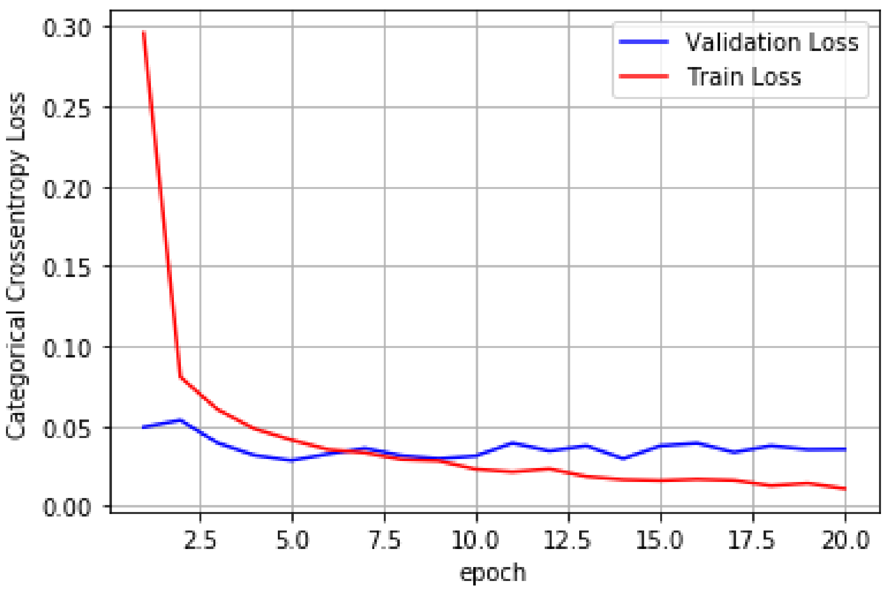
```
______
dense_10 (Dense)             (None, 256)             6579
2
__________________________________________________________
______
batch_normalization_3 (Batch (None, 256)             1024
__________________________________________________________
______
dropout_11 (Dropout)         (None, 256)             0
__________________________________________________________
______
dense_11 (Dense)             (None, 10)              2570
==========================================================
======
Total params: 583,562
Trainable params: 583,050
Non-trainable params: 512
__________________________________________________________
______
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 4s 62us/step
- loss: 0.2960 - accuracy: 0.9097 - val_loss: 0.0492 - val_
accuracy: 0.9846
Epoch 2/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0806 - accuracy: 0.9767 - val_loss: 0.0536 - val_
accuracy: 0.9844
Epoch 3/20
60000/60000 [==============================] - 3s 53us/step
- loss: 0.0604 - accuracy: 0.9830 - val_loss: 0.0394 - val_
accuracy: 0.9896
Epoch 4/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0482 - accuracy: 0.9865 - val_loss: 0.0317 - val_
accuracy: 0.9905
Epoch 5/20
60000/60000 [==============================] - 3s 53us/step
- loss: 0.0412 - accuracy: 0.9880 - val_loss: 0.0284 - val_
accuracy: 0.9927
Epoch 6/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0352 - accuracy: 0.9898 - val_loss: 0.0324 - val_
accuracy: 0.9907
Epoch 7/20
60000/60000 [==============================] - 3s 52us/step
- loss: 0.0332 - accuracy: 0.9901 - val_loss: 0.0360 - val_
accuracy: 0.9901
Epoch 8/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0290 - accuracy: 0.9918 - val_loss: 0.0314 - val_
accuracy: 0.9908
```

```
Epoch 9/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0281 - accuracy: 0.9915 - val_loss: 0.0295 - val_
accuracy: 0.9926
Epoch 10/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0229 - accuracy: 0.9936 - val_loss: 0.0312 - val_
accuracy: 0.9922
Epoch 11/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0212 - accuracy: 0.9936 - val_loss: 0.0392 - val_
accuracy: 0.9909
Epoch 12/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0231 - accuracy: 0.9930 - val_loss: 0.0344 - val_
accuracy: 0.9915
Epoch 13/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0183 - accuracy: 0.9945 - val_loss: 0.0375 - val_
accuracy: 0.9906
Epoch 14/20
60000/60000 [==============================] - 3s 53us/step
- loss: 0.0164 - accuracy: 0.9952 - val_loss: 0.0295 - val_
accuracy: 0.9920
Epoch 15/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0158 - accuracy: 0.9954 - val_loss: 0.0374 - val_
accuracy: 0.9904
Epoch 16/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0166 - accuracy: 0.9951 - val_loss: 0.0392 - val_
accuracy: 0.9897
Epoch 17/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0159 - accuracy: 0.9952 - val_loss: 0.0335 - val_
accuracy: 0.9928
Epoch 18/20
60000/60000 [==============================] - 3s 55us/step
- loss: 0.0125 - accuracy: 0.9962 - val_loss: 0.0375 - val_
accuracy: 0.9921
Epoch 19/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0141 - accuracy: 0.9959 - val_loss: 0.0352 - val_
accuracy: 0.9916
Epoch 20/20
60000/60000 [==============================] - 3s 54us/step
- loss: 0.0107 - accuracy: 0.9967 - val_loss: 0.0354 - val_
accuracy: 0.9924
```

In [21]:

```
score = model.evaluate(x_test, Y_test, verbose=0)
score4 = score[0]
accuracy4 = score[1]
print('Test score:', score4)
print('Test accuracy:', accuracy4)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

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

```
Test score: 0.0353821564207712
Test accuracy: 0.9923999905586243
```



Model 5 : 3 Convolutional Layers with 11X11 Filters, 2 Dense layer (ReLU Activation), 2 MaxPool and Adam Optimizer

In [22]:

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(11,11), padding='same', activation='relu',
model.add(Conv2D(64, (11, 11), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (11, 11), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['ac
history = model.fit(x_train, Y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, Y_test))
```

```
Model: "sequential_5"
___________________________________________________________________________
______
Layer (type)                 Output Shape              Para
m #
===========================================================================
======
conv2d_9 (Conv2D)            (None, 28, 28, 32)        3904
___________________________________________________________________________
______
conv2d_10 (Conv2D)           (None, 28, 28, 64)        2478
72
___________________________________________________________________________
______
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 64)        0
___________________________________________________________________________
______
conv2d_11 (Conv2D)           (None, 4, 4, 64)          4956
80
___________________________________________________________________________
______
max_pooling2d_6 (MaxPooling2 (None, 2, 2, 64)          0
___________________________________________________________________________
______
dropout_12 (Dropout)         (None, 2, 2, 64)          0
___________________________________________________________________________
______
```

```
flatten_5 (Flatten)          (None, 256)             0
_________________________________________________________________
dense_12 (Dense)             (None, 256)             6579
2
_________________________________________________________________
dropout_13 (Dropout)         (None, 256)             0
_________________________________________________________________
dense_13 (Dense)             (None, 256)             6579
2
_________________________________________________________________
batch_normalization_4 (Batch (None, 256)             1024
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)             0
_________________________________________________________________
dense_14 (Dense)             (None, 10)              2570
=================================================================
Total params: 882,634
Trainable params: 882,122
Non-trainable params: 512
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 120us/ste
p - loss: 0.2782 - accuracy: 0.9136 - val_loss: 0.0603 - va
l_accuracy: 0.9805
Epoch 2/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0703 - accuracy: 0.9808 - val_loss: 0.0282 - va
l_accuracy: 0.9919
Epoch 3/20
60000/60000 [==============================] - 7s 108us/ste
p - loss: 0.0471 - accuracy: 0.9869 - val_loss: 0.0236 - va
l_accuracy: 0.9923
Epoch 4/20
60000/60000 [==============================] - 7s 109us/ste
p - loss: 0.0407 - accuracy: 0.9891 - val_loss: 0.0485 - va
l_accuracy: 0.9878
Epoch 5/20
60000/60000 [==============================] - 6s 108us/ste
p - loss: 0.0351 - accuracy: 0.9904 - val_loss: 0.0258 - va
l_accuracy: 0.9927
Epoch 6/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0279 - accuracy: 0.9921 - val_loss: 0.0248 - va
```

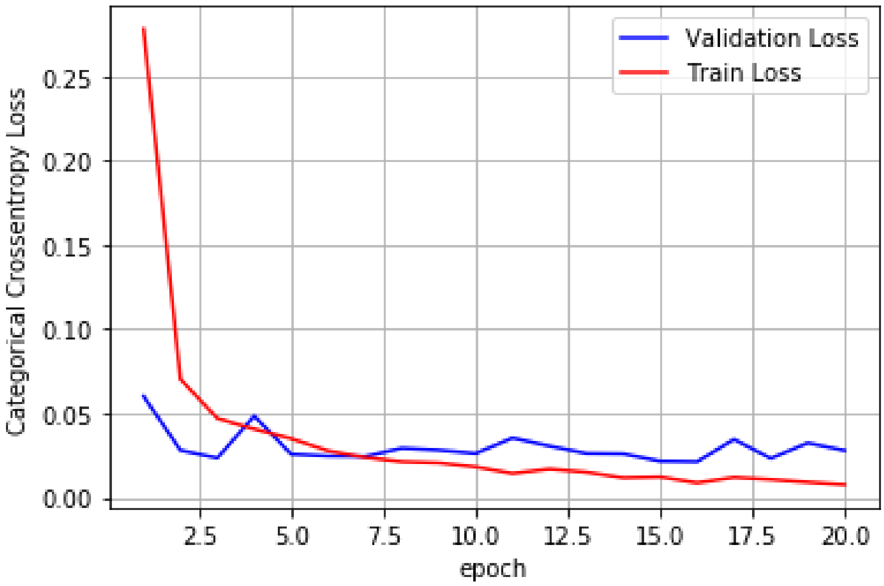
```
l_accuracy: 0.9930
Epoch 7/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0240 - accuracy: 0.9932 - val_loss: 0.0245 - va
l_accuracy: 0.9928
Epoch 8/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0213 - accuracy: 0.9942 - val_loss: 0.0294 - va
l_accuracy: 0.9925
Epoch 9/20
60000/60000 [==============================] - 6s 106us/ste
p - loss: 0.0207 - accuracy: 0.9940 - val_loss: 0.0283 - va
l_accuracy: 0.9928
Epoch 10/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0183 - accuracy: 0.9948 - val_loss: 0.0263 - va
l_accuracy: 0.9926
Epoch 11/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0144 - accuracy: 0.9960 - val_loss: 0.0355 - va
l_accuracy: 0.9903
Epoch 12/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0170 - accuracy: 0.9953 - val_loss: 0.0307 - va
l_accuracy: 0.9925
Epoch 13/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0151 - accuracy: 0.9958 - val_loss: 0.0263 - va
l_accuracy: 0.9928
Epoch 14/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0118 - accuracy: 0.9967 - val_loss: 0.0261 - va
l_accuracy: 0.9937
Epoch 15/20
60000/60000 [==============================] - 6s 106us/ste
p - loss: 0.0123 - accuracy: 0.9966 - val_loss: 0.0217 - va
l_accuracy: 0.9936
Epoch 16/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0088 - accuracy: 0.9976 - val_loss: 0.0214 - va
l_accuracy: 0.9945
Epoch 17/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0120 - accuracy: 0.9968 - val_loss: 0.0346 - va
l_accuracy: 0.9924
Epoch 18/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0108 - accuracy: 0.9967 - val_loss: 0.0234 - va
l_accuracy: 0.9940
Epoch 19/20
60000/60000 [==============================] - 6s 106us/ste
p - loss: 0.0092 - accuracy: 0.9976 - val_loss: 0.0325 - va
l_accuracy: 0.9925
```

```
Epoch 20/20
60000/60000 [==============================] - 6s 107us/ste
p - loss: 0.0078 - accuracy: 0.9979 - val_loss: 0.0280 - va
l_accuracy: 0.9934
```

In [23]:

```
score = model.evaluate(x_test, Y_test, verbose=0)
score5 = score[0]
accuracy5 = score[1]
print('Test score:', score5)
print('Test accuracy:', accuracy5)

fig,ax = plt.subplots(1,1)
ax.set_xlabel('epoch') ; ax.set_ylabel('Categorical Crossentropy Loss')

# list of epoch numbers
x = list(range(1,epochs+1))

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

```
Test score: 0.028024690563664523
Test accuracy: 0.993399977684021
```



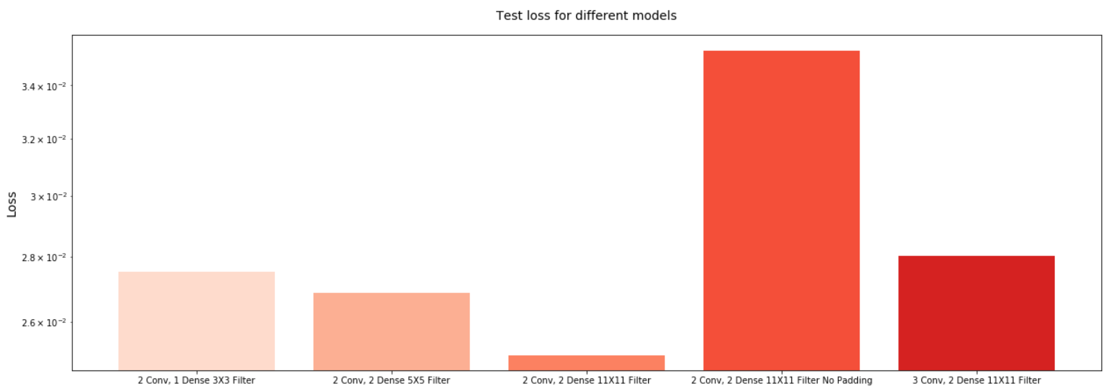
Vizualizing Results

In [24]:

```
import seaborn as sns
models = ['2 Conv, 1 Dense 3X3 Filter', '2 Conv, 2 Dense 5X5 Filter', '2 Conv
scores = [score1, score2, score3, score4, score5]
accuracies = [accuracy1, accuracy2, accuracy3, accuracy4, accuracy5]
```

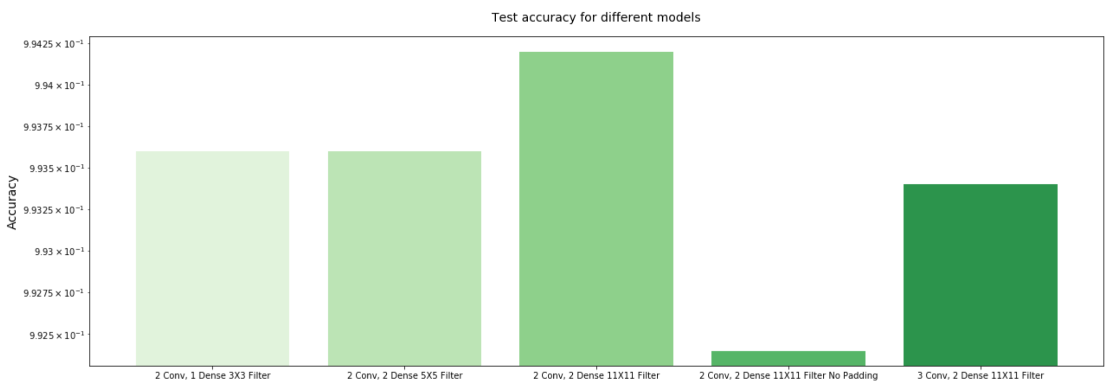
In [25]:

```
plt.figure(figsize=(21,7))
fig = plt.bar(models, scores, color=sns.color_palette("Reds",6))
plt.ylabel('Loss', size = 14)
plt.yscale('log')
plt.title('Test loss for different models', size = 14, y = 1.03)
plt.show()
```



In [26]:

```
plt.figure(figsize=(21,7))
fig = plt.bar(models, accuracies, color=sns.color_palette("Greens",6))
plt.ylabel('Accuracy', size = 14)
plt.yscale('log')
plt.title('Test accuracy for different models', size = 14, y = 1.03)
plt.show()
```



Conclusion

In [27]:

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Model", "Loss", "Accuracy"]

table.add_row([models[0], score1, accuracy1])
table.add_row([models[1], score2, accuracy2])
table.add_row([models[2], score3, accuracy3])
table.add_row([models[3], score4, accuracy4])
table.add_row([models[4], score5, accuracy5])
print(table)
```

```
+-----------------------------------------+-------------------
---+--------------------+
|                  Model                  |         Loss
|      Accuracy      |
+-----------------------------------------+-------------------
---+--------------------+
|         2 Conv, 1 Dense 3X3 Filter      | 0.0275068513898282
97 | 0.9936000108718872 |
|         2 Conv, 2 Dense 5X5 Filter      |  0.026864191379996
5  | 0.9936000108718872 |
|        2 Conv, 2 Dense 11X11 Filter     | 0.0250387510711745
95 | 0.9941999912261963 |
| 2 Conv, 2 Dense 11X11 Filter No Padding |  0.035382156420771
2  | 0.9923999905586243 |
|        3 Conv, 2 Dense 11X11 Filter     | 0.0280246905636645
23 | 0.993399977684021  |
+-----------------------------------------+-------------------
---+--------------------+
```