In [0]:
```python
# if you keras is not using tensorflow as backend set "KERAS_BACKEND=te
nsorflow" use this command
from keras.utils import np_utils
from keras.datasets import mnist
import seaborn as sns
from keras.initializers import RandomNormal

import warnings
warnings.filterwarnings("ignore")
```

In [0]:
```python
%matplotlib notebook
import matplotlib.pyplot as plt
import numpy as np
import time
%matplotlib inline

def plotting_graph(x,y1,y2):
  x = list(range(1,nb_epoch+1))

  y1 = history.history['val_loss']#validation loss
  y2 = history.history['loss']#training loss

  plt.plot(x, y1, label='validation loss')
  plt.plot(x, y2, label='train loss')

  plt.xlabel('epoch')
  plt.ylabel('Categorical Crossentropy Loss')

  plt.title("graph")

  plt.legend()

  plt.show()
```

In [0]:
```python
# the data, shuffled and split between train and test sets
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [49]:

```
#printing actual shapes of the splitted data
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
(60000,)
(10000,)
```

In [50]:

```
print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d, %d)"%(X_train.shape[1], X_train.shape[2]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d, %d)"%(X_test.shape[1], X_test.shape[2]))
```

```
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 [0]:

```
# 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 * 784

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

In [52]:

```
#printing shapes after reshaping of the splitted data
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(60000, 784)
```

```
(10000, 784)
(60000,)
(10000,)
```

In [53]:

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

print("Number of training examples :", X_train.shape[0], "and each imag
e is of shape (%d)"%(X_train.shape[1]))
print("Number of training examples :", X_test.shape[0], "and each image
 is of shape (%d)"%(X_test.shape[1]))
```

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

In [0]:

```
# 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 norma
lize the data
# X => (X - Xmin)/(Xmax-Xmin) = X/255

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

In [55]:

```
# 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, 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 [0]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation
```

In [0]:
```
# some model parameters

output_dim = 10
input_dim = X_train.shape[1]

batch_size = 128
nb_epoch = 20
```

Models that we are building here

In [0]:
```
#(1)MLP_WITH_2_LAYER_+_RELU_+_ADAM
#(2)MLP_WITH_2_LAYER_+_RELU_+_ADAM_+BN
#(3)MLP_WITH_2_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT
#(4)MLP_WITH_3_LAYER_+_RELU_+_ADAM
#(5)MLP_WITH_3_LAYER_+_RELU_+_ADAM+BN
#(6)MLP_WITH_3_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT
#(7)MLP_WITH_5_LAYER_+_RELU_+_ADAM
#(8)MLP_WITH_5_LAYER_+_RELU_+_ADAM_+_BM
#(9)MLP_WITH_5_LAYER_+_RELU_+_ADAM_+_BM+_Dropout
```

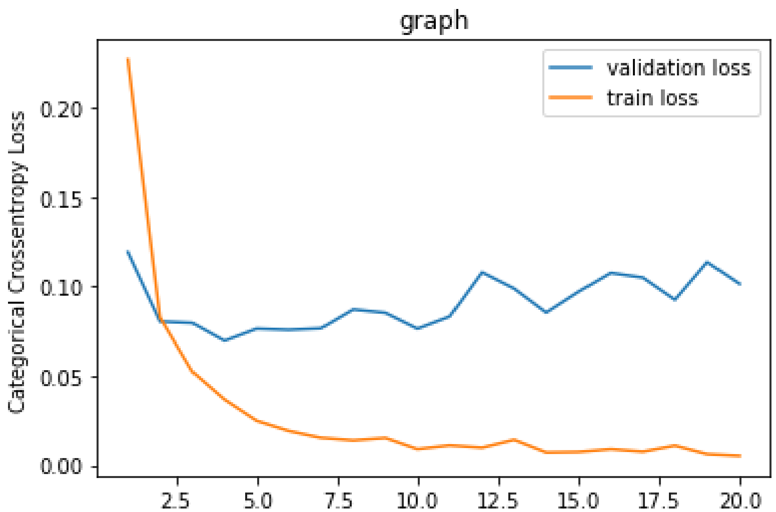
(1)MLP_WITH_2_LAYER_+_RELU_+_ADAM

In [59]:
```
#model building

#model 2:mlp with 2 layer + relu activation function +adam optimizer

#using 448 neurons and 262 neurons in h1 layer and h2 layer
#using relu function in h1 and h2
#using softmax layer in output layer
```

```
# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.066  => N(0,σ) = N(0,0.066)
# h2 =>  σ=√(2/(fan_in) = 0.087  => N(0,σ) = N(0,0.087)

model_1 = Sequential()
model_1.add(Dense(448, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.066, seed=None)))
model_1.add(Dense(262, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.087, seed=None)) )
model_1.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_1.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_1.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_1.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s 113us/step - loss:
0.2269 - acc: 0.9322 - val_loss: 0.1193 - val_acc: 0.9619
Epoch 2/20
60000/60000 [==============================] - 4s 62us/step - loss:
0.0829 - acc: 0.9749 - val_loss: 0.0804 - val_acc: 0.9759
Epoch 3/20
60000/60000 [==============================] - 4s 60us/step - loss:
0.0525 - acc: 0.9841 - val_loss: 0.0796 - val_acc: 0.9754
Epoch 4/20
60000/60000 [==============================] - 4s 61us/step - loss:
0.0368 - acc: 0.9891 - val_loss: 0.0697 - val_acc: 0.9789
Epoch 5/20
60000/60000 [==============================] - 4s 62us/step - loss:
0.0249 - acc: 0.9918 - val_loss: 0.0764 - val_acc: 0.9788
Epoch 6/20
60000/60000 [==============================] - 4s 62us/step - loss:
0.0192 - acc: 0.9939 - val_loss: 0.0758 - val_acc: 0.9787
Epoch 7/20
60000/60000 [==============================] - 4s 63us/step - loss:
0.0154 - acc: 0.9953 - val_loss: 0.0766 - val_acc: 0.9793
Epoch 8/20
60000/60000 [==============================] - 4s 62us/step - loss:
0.0139 - acc: 0.9955 - val_loss: 0.0871 - val_acc: 0.9783
Epoch 9/20
60000/60000 [==============================] - 4s 60us/step - loss:
0.0153 - acc: 0.9947 - val_loss: 0.0853 - val_acc: 0.9788
Epoch 10/20
60000/60000 [==============================] - 4s 61us/step - loss:
0.0090 - acc: 0.9971 - val_loss: 0.0764 - val_acc: 0.9805
Epoch 11/20
60000/60000 [==============================] - 4s 61us/step - loss:
0.0111 - acc: 0.9961 - val_loss: 0.0831 - val_acc: 0.9806
Epoch 12/20
60000/60000 [==============================] - 4s 59us/step - loss:
0.0098 - acc: 0.9966 - val_loss: 0.1080 - val_acc: 0.9759
Epoch 13/20
60000/60000 [==============================] - 3s 58us/step - loss:
```

```
0.0142 - acc: 0.9951 - val_loss: 0.0989 - val_acc: 0.9776
Epoch 14/20
60000/60000 [==============================] - 4s 58us/step - loss:
0.0071 - acc: 0.9976 - val_loss: 0.0854 - val_acc: 0.9820
Epoch 15/20
60000/60000 [==============================] - 4s 59us/step - loss:
0.0074 - acc: 0.9974 - val_loss: 0.0970 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] - 4s 59us/step - loss:
0.0090 - acc: 0.9970 - val_loss: 0.1076 - val_acc: 0.9787
Epoch 17/20
60000/60000 [==============================] - 4s 60us/step - loss:
0.0076 - acc: 0.9974 - val_loss: 0.1050 - val_acc: 0.9790
Epoch 18/20
60000/60000 [==============================] - 4s 61us/step - loss:
0.0110 - acc: 0.9962 - val_loss: 0.0926 - val_acc: 0.9813
Epoch 19/20
60000/60000 [==============================] - 4s 61us/step - loss:
0.0062 - acc: 0.9980 - val_loss: 0.1137 - val_acc: 0.9754
Epoch 20/20
60000/60000 [==============================] - 4s 62us/step - loss:
0.0052 - acc: 0.9980 - val_loss: 0.1015 - val_acc: 0.9794
Test score: 0.10150725200319671
Test accuracy: 0.9794
```



epoch

(2)MLP_WITH_2_LAYER_+_RELU_+_ADAM_+BN

In [62]:

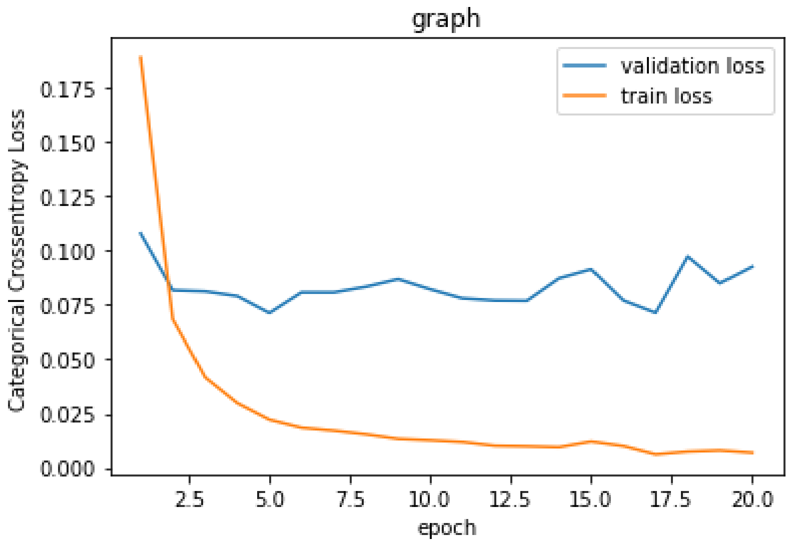
```
#model building and adding batch normalization

#model 2:mlp with 2 layer + relu activation function +adam optimizer +
 batch normalization

#using 448 neurons and 262 neurons in h1 layer and h2 layer
#using relu function in h1 and h2
#using softmax layer in output layer

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.066  => N(0,σ) = N(0,0.066)
# h2 =>  σ=√(2/(fan_in) = 0.087  => N(0,σ) = N(0,0.087)
from keras.layers.normalization import BatchNormalization

#adding BN to h1 and h2

model_2 = Sequential()
model_2.add(Dense(448, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.066, seed=None)))
model_2.add(BatchNormalization())
model_2.add(Dense(262, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.087, seed=None)) )
model_2.add(BatchNormalization())
model_2.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
```

```
history = model_2.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_2.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 148us/step - loss: 0.
1888 - acc: 0.9424 - val_loss: 0.1080 - val_acc: 0.9653
Epoch 2/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
685 - acc: 0.9796 - val_loss: 0.0818 - val_acc: 0.9749
Epoch 3/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
417 - acc: 0.9868 - val_loss: 0.0812 - val_acc: 0.9733
Epoch 4/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
298 - acc: 0.9906 - val_loss: 0.0791 - val_acc: 0.9756
Epoch 5/20
60000/60000 [==============================] - 6s 95us/step - loss: 0.0
222 - acc: 0.9927 - val_loss: 0.0713 - val_acc: 0.9787
Epoch 6/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0
185 - acc: 0.9941 - val_loss: 0.0808 - val_acc: 0.9766
```

```
Epoch 7/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
171 - acc: 0.9946 - val_loss: 0.0808 - val_acc: 0.9780
Epoch 8/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
154 - acc: 0.9952 - val_loss: 0.0834 - val_acc: 0.9762
Epoch 9/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
133 - acc: 0.9956 - val_loss: 0.0869 - val_acc: 0.9754
Epoch 10/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
127 - acc: 0.9957 - val_loss: 0.0822 - val_acc: 0.9803
Epoch 11/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.0
119 - acc: 0.9962 - val_loss: 0.0780 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
102 - acc: 0.9966 - val_loss: 0.0770 - val_acc: 0.9789
Epoch 13/20
60000/60000 [==============================] - 6s 93us/step - loss: 0.0
099 - acc: 0.9969 - val_loss: 0.0769 - val_acc: 0.9811
Epoch 14/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
095 - acc: 0.9971 - val_loss: 0.0872 - val_acc: 0.9801
Epoch 15/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.0
121 - acc: 0.9958 - val_loss: 0.0914 - val_acc: 0.9793
Epoch 16/20
60000/60000 [==============================] - 6s 94us/step - loss: 0.0
100 - acc: 0.9965 - val_loss: 0.0770 - val_acc: 0.9814
Epoch 17/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
062 - acc: 0.9981 - val_loss: 0.0714 - val_acc: 0.9834
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.0
075 - acc: 0.9977 - val_loss: 0.0973 - val_acc: 0.9769
Epoch 19/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.0
080 - acc: 0.9973 - val_loss: 0.0850 - val_acc: 0.9805
```

```
Epoch 20/20
60000/60000 [==============================] - 5s 89us/step - loss: 0.0
069 - acc: 0.9977 - val_loss: 0.0926 - val_acc: 0.9793
Test score: 0.09259701214867527
Test accuracy: 0.9793
```



(3)MLP_WITH_2_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT

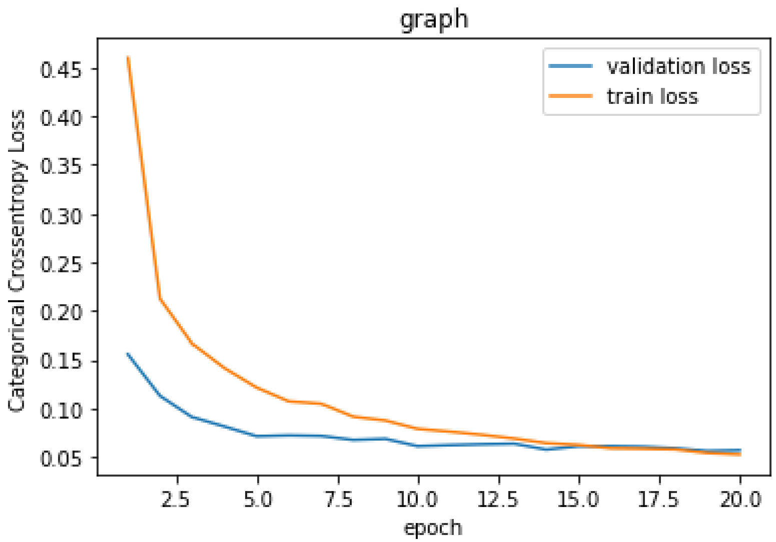
In [63]:
```
#model building and adding batch normalization and dropouts

#model 2:mlp with 2 layer + relu activation function +adam optimizer +
 batch normalization + dropout

#using 448 neurons and 262 neurons in h1 layer and h2 layer
#using relu function in h1 and h2
#using softmax layer in output layer
#using bn and dropout in h1 and h2

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
```

```
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) = 0.066  => N(0,σ) = N(0,0.066)
# h2 =>  σ=√(2/(fan_in) = 0.087  => N(0,σ) = N(0,0.087)
from keras.layers import Dropout

#adding BN to h1 and h2

model_3 = Sequential()
model_3.add(Dense(448, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.066, seed=None)))
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(262, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.087, seed=None)) )
model_3.add(BatchNormalization())
model_3.add(Dropout(0.5))
model_3.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_3.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_3.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_3.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s 175us/step - loss:
0.4594 - acc: 0.8606 - val_loss: 0.1556 - val_acc: 0.9522
Epoch 2/20
60000/60000 [==============================] - 6s 100us/step - loss:
0.2124 - acc: 0.9351 - val_loss: 0.1128 - val_acc: 0.9651
Epoch 3/20
60000/60000 [==============================] - 6s 103us/step - loss:
0.1662 - acc: 0.9491 - val_loss: 0.0908 - val_acc: 0.9707
Epoch 4/20
60000/60000 [==============================] - 6s 103us/step - loss:
0.1412 - acc: 0.9572 - val_loss: 0.0811 - val_acc: 0.9740
Epoch 5/20
60000/60000 [==============================] - 6s 101us/step - loss:
0.1213 - acc: 0.9617 - val_loss: 0.0711 - val_acc: 0.9779
Epoch 6/20
60000/60000 [==============================] - 6s 104us/step - loss:
0.1071 - acc: 0.9673 - val_loss: 0.0720 - val_acc: 0.9782
Epoch 7/20
60000/60000 [==============================] - 6s 101us/step - loss:
0.1047 - acc: 0.9671 - val_loss: 0.0714 - val_acc: 0.9782
Epoch 8/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0913 - acc: 0.9714 - val_loss: 0.0673 - val_acc: 0.9809
Epoch 9/20
60000/60000 [==============================] - 6s 105us/step - loss:
0.0873 - acc: 0.9718 - val_loss: 0.0684 - val_acc: 0.9796
Epoch 10/20
60000/60000 [==============================] - 6s 103us/step - loss:
0.0788 - acc: 0.9759 - val_loss: 0.0609 - val_acc: 0.9812
Epoch 11/20
60000/60000 [==============================] - 6s 105us/step - loss:
0.0759 - acc: 0.9755 - val_loss: 0.0620 - val_acc: 0.9808
```

```
Epoch 12/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0726 - acc: 0.9762 - val_loss: 0.0628 - val_acc: 0.9806
Epoch 13/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0688 - acc: 0.9780 - val_loss: 0.0635 - val_acc: 0.9810
Epoch 14/20
60000/60000 [==============================] - 6s 100us/step - loss:
0.0640 - acc: 0.9797 - val_loss: 0.0573 - val_acc: 0.9824
Epoch 15/20
60000/60000 [==============================] - 6s 101us/step - loss:
0.0622 - acc: 0.9803 - val_loss: 0.0608 - val_acc: 0.9816
Epoch 16/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0587 - acc: 0.9812 - val_loss: 0.0608 - val_acc: 0.9819
Epoch 17/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0583 - acc: 0.9812 - val_loss: 0.0603 - val_acc: 0.9832
Epoch 18/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0576 - acc: 0.9819 - val_loss: 0.0587 - val_acc: 0.9832
Epoch 19/20
60000/60000 [==============================] - 6s 102us/step - loss:
0.0539 - acc: 0.9829 - val_loss: 0.0560 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 6s 101us/step - loss:
0.0524 - acc: 0.9827 - val_loss: 0.0567 - val_acc: 0.9848
Test score: 0.056680836588794775
Test accuracy: 0.9848
```



(4)MLP_WITH_3_LAYER_+_RELU_+_ADAM

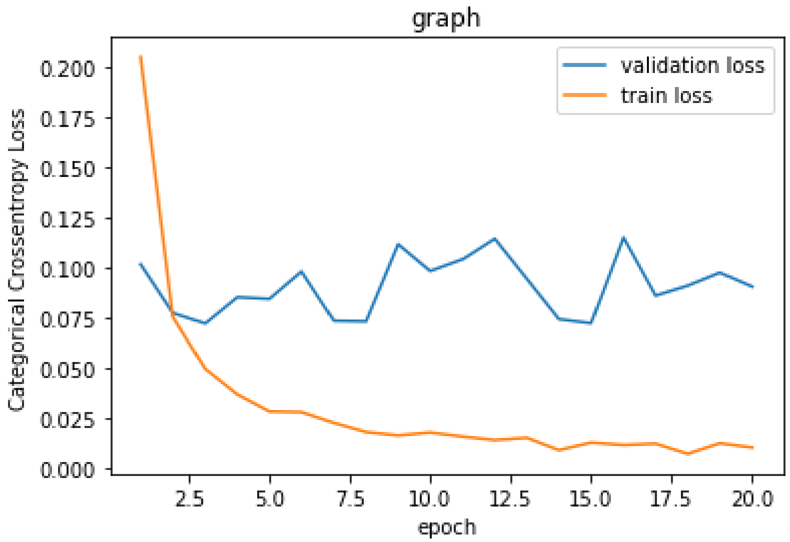
In [64]:
```
#model building

#model 4:mlp with 3 layer + relu activation function +adam optimizer

#using 660 neurons and 410 neurons and 230 neurons in h1 layer and h2 l
ayer and h3 layers
#using relu function in h1 , h2 , h3
#using softmax layer in output layer

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(660)= 0.055  => N(0,σ) = N(0,0.055)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(410)= 0.069  => N(0,σ) = N(0,0.069)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(230)= 0.093  => N(0,σ) = N(0,0.093)
```

```
model_4 = Sequential()
model_4.add(Dense(660, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.055, seed=None)))
model_4.add(Dense(410, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.069, seed=None)))
model_4.add(Dense(230, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.093, seed=None)))
model_4.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_4.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_4.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_4.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 9s 146us/step - loss: 0.
2050 - acc: 0.9370 - val_loss: 0.1018 - val_acc: 0.9663
Epoch 2/20
```

```
60000/60000 [==============================] - 5s 77us/step - loss: 0.0
757 - acc: 0.9769 - val_loss: 0.0776 - val_acc: 0.9754
Epoch 3/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0
498 - acc: 0.9845 - val_loss: 0.0723 - val_acc: 0.9768
Epoch 4/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0
370 - acc: 0.9878 - val_loss: 0.0854 - val_acc: 0.9758
Epoch 5/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0
285 - acc: 0.9903 - val_loss: 0.0846 - val_acc: 0.9753
Epoch 6/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0
282 - acc: 0.9910 - val_loss: 0.0982 - val_acc: 0.9733
Epoch 7/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0
228 - acc: 0.9925 - val_loss: 0.0736 - val_acc: 0.9811
Epoch 8/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0
182 - acc: 0.9942 - val_loss: 0.0733 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0
165 - acc: 0.9949 - val_loss: 0.1119 - val_acc: 0.9727
Epoch 10/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0
180 - acc: 0.9939 - val_loss: 0.0985 - val_acc: 0.9770
Epoch 11/20
60000/60000 [==============================] - 4s 73us/step - loss: 0.0
159 - acc: 0.9947 - val_loss: 0.1043 - val_acc: 0.9756
Epoch 12/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0
142 - acc: 0.9954 - val_loss: 0.1146 - val_acc: 0.9763
Epoch 13/20
60000/60000 [==============================] - 4s 74us/step - loss: 0.0
153 - acc: 0.9955 - val_loss: 0.0946 - val_acc: 0.9768
Epoch 14/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0
092 - acc: 0.9971 - val_loss: 0.0745 - val_acc: 0.9828
Epoch 15/20
60000/60000 [==============================] - 4s 75us/step - loss: 0.0
```

```
129 - acc: 0.9960 - val_loss: 0.0724 - val_acc: 0.9840

Epoch 16/20
60000/60000 [==============================] - 5s 75us/step - loss: 0.0
117 - acc: 0.9966 - val_loss: 0.1152 - val_acc: 0.9755
Epoch 17/20
60000/60000 [==============================] - 5s 78us/step - loss: 0.0
124 - acc: 0.9960 - val_loss: 0.0862 - val_acc: 0.9831
Epoch 18/20
60000/60000 [==============================] - 5s 80us/step - loss: 0.0
073 - acc: 0.9979 - val_loss: 0.0912 - val_acc: 0.9801
Epoch 19/20
60000/60000 [==============================] - 5s 77us/step - loss: 0.0
126 - acc: 0.9962 - val_loss: 0.0976 - val_acc: 0.9796
Epoch 20/20
60000/60000 [==============================] - 5s 76us/step - loss: 0.0
104 - acc: 0.9969 - val_loss: 0.0907 - val_acc: 0.9817
Test score: 0.09068295588794917
Test accuracy: 0.9817
```



(5)MLP_WITH_3_LAYER_+_RELU_+_ADAM+BN

In [65]:

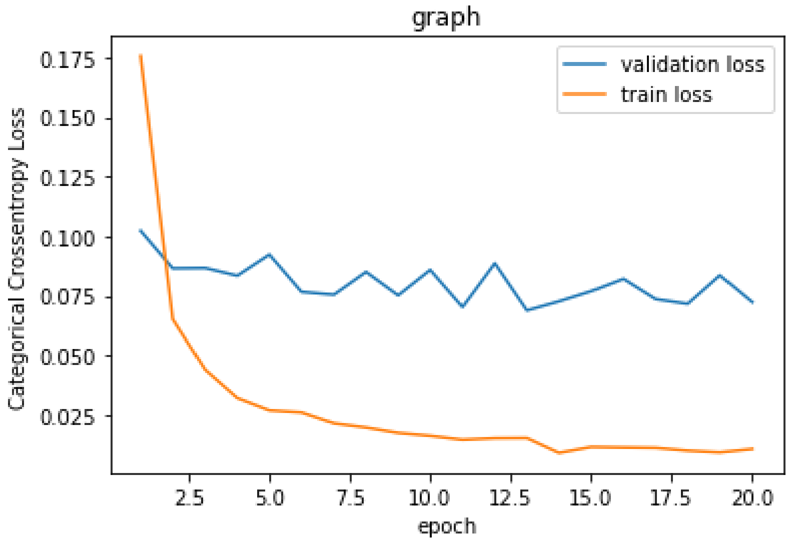
```
#model building

#model 5:mlp with 3 layer + relu activation function +adam optimizer +B
N

#using 660 neurons and 410 neurons and 230 neurons in h1 layer and h2 l
ayer and h3 layers
#using relu function in h1 , h2 , h3
#using softmax layer in output layer
#adding batch normalizaiton to h1 ,h2 and h3

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(660)= 0.055  => N(0,σ) = N(0,0.055)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(410)= 0.069  => N(0,σ) = N(0,0.069)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(230)= 0.093  => N(0,σ) = N(0,0.093)

model_5 = Sequential()
model_5.add(Dense(660, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.055, seed=None)))
model_5.add(BatchNormalization())
model_5.add(Dense(410, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.069, seed=None)))
model_5.add(BatchNormalization())
model_5.add(Dense(230, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.093, seed=None)))
model_5.add(BatchNormalization())
model_5.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_5.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_5.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))
```

```
##plotting train ,test vs epoch graph
##calculating accuracy

score = model_5.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 13s 219us/step - loss:
0.1755 - acc: 0.9463 - val_loss: 0.1023 - val_acc: 0.9683
Epoch 2/20
60000/60000 [==============================] - 8s 137us/step - loss:
0.0651 - acc: 0.9793 - val_loss: 0.0864 - val_acc: 0.9735
Epoch 3/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0439 - acc: 0.9851 - val_loss: 0.0866 - val_acc: 0.9720
Epoch 4/20
60000/60000 [==============================] - 8s 137us/step - loss:
0.0320 - acc: 0.9896 - val_loss: 0.0834 - val_acc: 0.9757
Epoch 5/20
60000/60000 [==============================] - 8s 137us/step - loss:
0.0268 - acc: 0.9907 - val_loss: 0.0922 - val_acc: 0.9728
Epoch 6/20
60000/60000 [==============================] - 8s 137us/step - loss:
0.0260 - acc: 0.9911 - val_loss: 0.0766 - val_acc: 0.9783
Epoch 7/20
60000/60000 [==============================] - 8s 136us/step - loss:
```

```
0.0214 - acc: 0.9928 - val_loss: 0.0754 - val_acc: 0.9779
Epoch 8/20
60000/60000 [==============================] - 8s 134us/step - loss:
0.0196 - acc: 0.9934 - val_loss: 0.0849 - val_acc: 0.9752
Epoch 9/20
60000/60000 [==============================] - 8s 136us/step - loss:
0.0173 - acc: 0.9944 - val_loss: 0.0751 - val_acc: 0.9798
Epoch 10/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0161 - acc: 0.9948 - val_loss: 0.0858 - val_acc: 0.9770
Epoch 11/20
60000/60000 [==============================] - 8s 134us/step - loss:
0.0145 - acc: 0.9954 - val_loss: 0.0703 - val_acc: 0.9818
Epoch 12/20
60000/60000 [==============================] - 8s 137us/step - loss:
0.0151 - acc: 0.9948 - val_loss: 0.0886 - val_acc: 0.9798
Epoch 13/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0152 - acc: 0.9949 - val_loss: 0.0688 - val_acc: 0.9828
Epoch 14/20
60000/60000 [==============================] - 8s 134us/step - loss:
0.0089 - acc: 0.9969 - val_loss: 0.0726 - val_acc: 0.9812
Epoch 15/20
60000/60000 [==============================] - 8s 132us/step - loss:
0.0114 - acc: 0.9960 - val_loss: 0.0768 - val_acc: 0.9815
Epoch 16/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0112 - acc: 0.9963 - val_loss: 0.0820 - val_acc: 0.9811
Epoch 17/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0110 - acc: 0.9963 - val_loss: 0.0735 - val_acc: 0.9826
Epoch 18/20
60000/60000 [==============================] - 8s 134us/step - loss:
0.0098 - acc: 0.9966 - val_loss: 0.0715 - val_acc: 0.9823
Epoch 19/20
60000/60000 [==============================] - 8s 135us/step - loss:
0.0091 - acc: 0.9969 - val_loss: 0.0835 - val_acc: 0.9813
Epoch 20/20
60000/60000 [==============================] - 8s 138us/step - loss:
```

```
0.0106 - acc: 0.9969 - val_loss: 0.0723 - val_acc: 0.9840
Test score: 0.07225036431260887
Test accuracy: 0.984
```



(6)MLP_WITH_3_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT

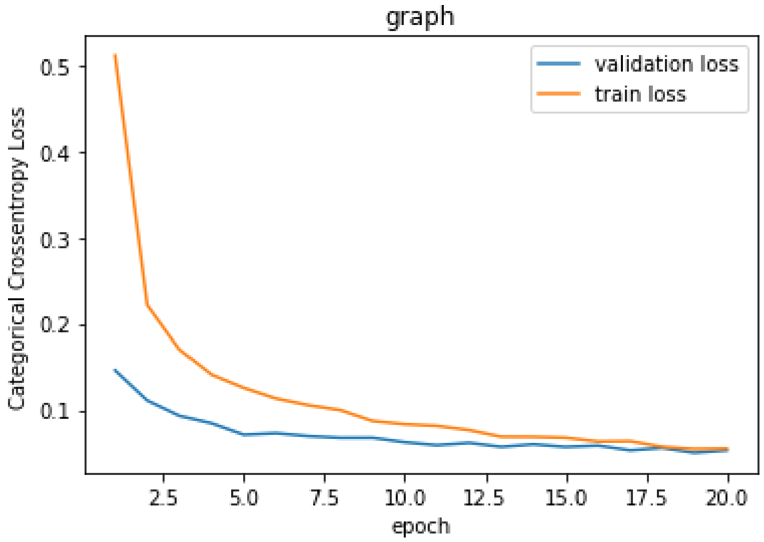
In [66]:
```
#model building

#model 6:mlp with 3 layer + relu activation function +adam optimizer +B
N + dropouts

#using 660 neurons and 410 neurons and 230 neurons in h1 layer and h2 l
ayer and h3 layers
#using relu function in h1 , h2 , h3
#using softmax layer in output layer
#adding batch normalizaiton to h1 ,h2 and h3

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
```

```
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(660)= 0.055  => N(0,σ) = N(0,0.055)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(410)= 0.069  => N(0,σ) = N(0,0.069)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(230)= 0.093  => N(0,σ) = N(0,0.093)

model_6 = Sequential()
model_6.add(Dense(660, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.055, seed=None)))
model_6.add(BatchNormalization())
model_6.add(Dropout(0.5))
model_6.add(Dense(410, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.069, seed=None)))
model_6.add(BatchNormalization())
model_6.add(Dropout(0.5))
model_6.add(Dense(230, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.093, seed=None)))
model_6.add(BatchNormalization())
model_6.add(Dropout(0.5))
model_6.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_6.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_6.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_6.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

# list of epoch numbers
```

```
x = list(range(1,nb_epoch+1))

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 14s 235us/step - loss: 
0.5110 - acc: 0.8441 - val_loss: 0.1460 - val_acc: 0.9543
Epoch 2/20
60000/60000 [==============================] - 9s 142us/step - loss: 
0.2218 - acc: 0.9328 - val_loss: 0.1110 - val_acc: 0.9657
Epoch 3/20
60000/60000 [==============================] - 9s 142us/step - loss: 
0.1700 - acc: 0.9478 - val_loss: 0.0932 - val_acc: 0.9719
Epoch 4/20
60000/60000 [==============================] - 8s 140us/step - loss: 
0.1409 - acc: 0.9567 - val_loss: 0.0844 - val_acc: 0.9741
Epoch 5/20
60000/60000 [==============================] - 8s 140us/step - loss: 
0.1256 - acc: 0.9616 - val_loss: 0.0711 - val_acc: 0.9777
Epoch 6/20
60000/60000 [==============================] - 8s 141us/step - loss: 
0.1133 - acc: 0.9649 - val_loss: 0.0729 - val_acc: 0.9770
Epoch 7/20
60000/60000 [==============================] - 9s 142us/step - loss: 
0.1056 - acc: 0.9671 - val_loss: 0.0695 - val_acc: 0.9797
Epoch 8/20
60000/60000 [==============================] - 9s 142us/step - loss: 
0.0997 - acc: 0.9697 - val_loss: 0.0675 - val_acc: 0.9803
Epoch 9/20
60000/60000 [==============================] - 8s 139us/step - loss: 
0.0871 - acc: 0.9731 - val_loss: 0.0676 - val_acc: 0.9799
Epoch 10/20
60000/60000 [==============================] - 9s 142us/step - loss: 
0.0833 - acc: 0.9739 - val_loss: 0.0624 - val_acc: 0.9800
Epoch 11/20
60000/60000 [==============================] - 8s 141us/step - loss: 
0.0814 - acc: 0.9741 - val_loss: 0.0589 - val_acc: 0.9827
```

```
Epoch 12/20
60000/60000 [==============================] - 8s 141us/step - loss:
0.0765 - acc: 0.9763 - val_loss: 0.0616 - val_acc: 0.9822
Epoch 13/20
60000/60000 [==============================] - 9s 142us/step - loss:
0.0688 - acc: 0.9788 - val_loss: 0.0570 - val_acc: 0.9830
Epoch 14/20
60000/60000 [==============================] - 9s 144us/step - loss:
0.0686 - acc: 0.9784 - val_loss: 0.0600 - val_acc: 0.9825
Epoch 15/20
60000/60000 [==============================] - 9s 148us/step - loss:
0.0677 - acc: 0.9785 - val_loss: 0.0569 - val_acc: 0.9840
Epoch 16/20
60000/60000 [==============================] - 8s 141us/step - loss:
0.0632 - acc: 0.9799 - val_loss: 0.0585 - val_acc: 0.9824
Epoch 17/20
60000/60000 [==============================] - 9s 143us/step - loss:
0.0636 - acc: 0.9798 - val_loss: 0.0529 - val_acc: 0.9841
Epoch 18/20
60000/60000 [==============================] - 8s 139us/step - loss:
0.0570 - acc: 0.9823 - val_loss: 0.0557 - val_acc: 0.9848
Epoch 19/20
60000/60000 [==============================] - 8s 141us/step - loss:
0.0545 - acc: 0.9826 - val_loss: 0.0501 - val_acc: 0.9857
Epoch 20/20
60000/60000 [==============================] - 9s 145us/step - loss:
0.0549 - acc: 0.9826 - val_loss: 0.0529 - val_acc: 0.9842
Test score: 0.052923913336900295
Test accuracy: 0.9842
```



(7)MLP_WITH_5_LAYER_+_RELU_+_ADAM

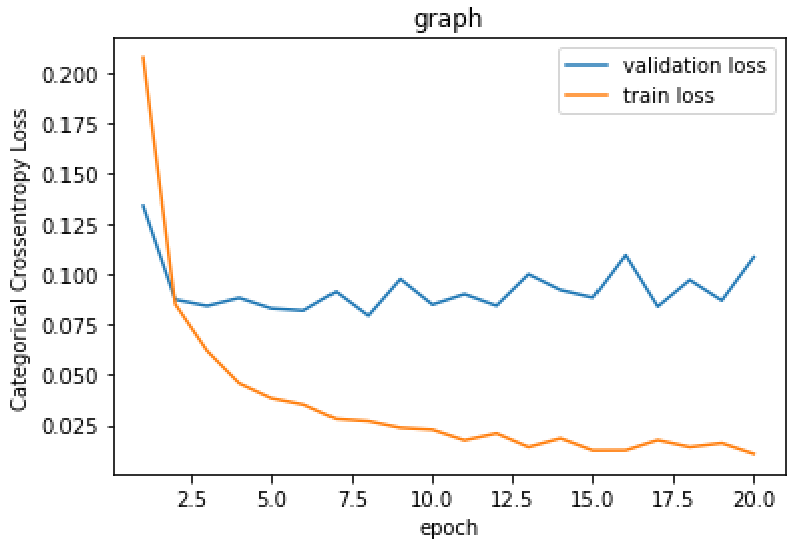
In [67]:
```
#model building

#model 7:mlp with 5 layer + relu activation function +adam optimizer

#using 712 neurons ,556 neurons ,438 neurons ,312 neurons ,144 neurons
 in h1 layer h2 layer ,h3 layers ,h4 layer and h5 layer
#using relu function in h1 , h2 , h3, h4 , h5
#using softmax layer in output layer

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(712)= 0.052  => N(0,σ) = N(0,0.052)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(556)= 0.059  => N(0,σ) = N(0,0.059)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(438)= 0.067  => N(0,σ) = N(0,0.067)
```

```
# h4 =>  σ=√(2/(fan_in) =σ=√(2/(312)= 0.080  => N(0,σ) = N(0,0.080)
# h5 =>  σ=√(2/(fan_in) =σ=√(2/(144)= 0.11  => N(0,σ) = N(0,0.11)

model_7 = Sequential()
model_7.add(Dense(712, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.052, seed=None)))
model_7.add(Dense(556, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.059, seed=None)))
model_7.add(Dense(438, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.067, seed=None)))
model_7.add(Dense(312, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.080, seed=None)))
model_7.add(Dense(144, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.11, seed=None)))
model_7.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_7.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_7.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_7.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
```

```
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.2077 - acc: 0.9368 - val_loss: 0.1343 - val_acc: 0.9582
Epoch 2/20
60000/60000 [==============================] - 6s 95us/step - loss:
0.0853 - acc: 0.9738 - val_loss: 0.0875 - val_acc: 0.9728
Epoch 3/20
60000/60000 [==============================] - 6s 95us/step - loss:
0.0619 - acc: 0.9810 - val_loss: 0.0844 - val_acc: 0.9740
Epoch 4/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0458 - acc: 0.9858 - val_loss: 0.0884 - val_acc: 0.9735
Epoch 5/20
60000/60000 [==============================] - 6s 92us/step - loss:
0.0384 - acc: 0.9880 - val_loss: 0.0831 - val_acc: 0.9771
Epoch 6/20
60000/60000 [==============================] - 6s 93us/step - loss:
0.0352 - acc: 0.9896 - val_loss: 0.0821 - val_acc: 0.9775
Epoch 7/20
60000/60000 [==============================] - 6s 94us/step - loss:
0.0281 - acc: 0.9916 - val_loss: 0.0915 - val_acc: 0.9775
Epoch 8/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0270 - acc: 0.9922 - val_loss: 0.0795 - val_acc: 0.9779
Epoch 9/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0236 - acc: 0.9927 - val_loss: 0.0977 - val_acc: 0.9776
Epoch 10/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0227 - acc: 0.9933 - val_loss: 0.0850 - val_acc: 0.9788
Epoch 11/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0174 - acc: 0.9946 - val_loss: 0.0902 - val_acc: 0.9809
Epoch 12/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0209 - acc: 0.9935 - val_loss: 0.0845 - val_acc: 0.9802
```

```
Epoch 13/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0140 - acc: 0.9959 - val_loss: 0.1002 - val_acc: 0.9796
Epoch 14/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0184 - acc: 0.9948 - val_loss: 0.0923 - val_acc: 0.9801
Epoch 15/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0124 - acc: 0.9968 - val_loss: 0.0885 - val_acc: 0.9823
Epoch 16/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0124 - acc: 0.9965 - val_loss: 0.1097 - val_acc: 0.9808
Epoch 17/20
60000/60000 [==============================] - 6s 96us/step - loss:
0.0175 - acc: 0.9951 - val_loss: 0.0841 - val_acc: 0.9819
Epoch 18/20
60000/60000 [==============================] - 6s 98us/step - loss:
0.0140 - acc: 0.9958 - val_loss: 0.0973 - val_acc: 0.9805
Epoch 19/20
60000/60000 [==============================] - 6s 97us/step - loss:
0.0160 - acc: 0.9956 - val_loss: 0.0871 - val_acc: 0.9814
Epoch 20/20
60000/60000 [==============================] - 6s 94us/step - loss:
0.0106 - acc: 0.9971 - val_loss: 0.1087 - val_acc: 0.9803
Test score: 0.10868040132140352
Test accuracy: 0.9803
```



(8)MLP_WITH_5_LAYER_+_RELU_+_ADAM_+_BM

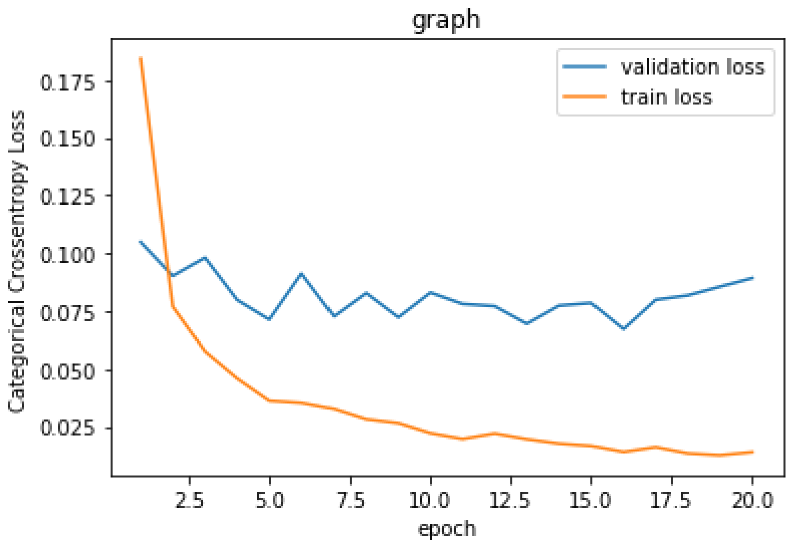
In [68]:
```
#model building

#model 8:mlp with 5 layer + relu activation function +adam optimizer +B
N

#using 712 neurons ,556 neurons ,438 neurons ,312 neurons ,144 neurons
 in h1 layer h2 layer ,h3 layers ,h4 layer and h5 layer
#using relu function in h1 , h2 , h3, h4 , h5
#using softmax layer in output layer

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(712)= 0.052  => N(0,σ) = N(0,0.052)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(556)= 0.059  => N(0,σ) = N(0,0.059)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(438)= 0.067  => N(0,σ) = N(0,0.067)
```

```
# h4 =>  σ=√(2/(fan_in) =σ=√(2/(312)= 0.080  => N(0,σ) = N(0,0.080)
# h5 =>  σ=√(2/(fan_in) =σ=√(2/(144)= 0.11  => N(0,σ) = N(0,0.11)

model_8 = Sequential()
model_8.add(Dense(712, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.052, seed=None)))
model_8.add(BatchNormalization())
model_8.add(Dense(556, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.059, seed=None)))
model_8.add(BatchNormalization())
model_8.add(Dense(438, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.067, seed=None)))
model_8.add(BatchNormalization())
model_8.add(Dense(312, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.080, seed=None)))
model_8.add(BatchNormalization())
model_8.add(Dense(144, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.11, seed=None)))
model_8.add(BatchNormalization())
model_8.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_8.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_8.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_8.evaluate(X_test, Y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 18s 298us/step - loss:
0.1842 - acc: 0.9432 - val_loss: 0.1049 - val_acc: 0.9652
Epoch 2/20
60000/60000 [==============================] - 11s 190us/step - loss:
0.0770 - acc: 0.9762 - val_loss: 0.0902 - val_acc: 0.9714
Epoch 3/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.0576 - acc: 0.9816 - val_loss: 0.0981 - val_acc: 0.9699
Epoch 4/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.0459 - acc: 0.9845 - val_loss: 0.0798 - val_acc: 0.9766
Epoch 5/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0362 - acc: 0.9882 - val_loss: 0.0713 - val_acc: 0.9786
Epoch 6/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0353 - acc: 0.9885 - val_loss: 0.0912 - val_acc: 0.9724
Epoch 7/20
60000/60000 [==============================] - 11s 192us/step - loss:
0.0326 - acc: 0.9892 - val_loss: 0.0727 - val_acc: 0.9793
Epoch 8/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0282 - acc: 0.9904 - val_loss: 0.0828 - val_acc: 0.9776
Epoch 9/20
60000/60000 [==============================] - 11s 188us/step - loss:
0.0264 - acc: 0.9912 - val_loss: 0.0723 - val_acc: 0.9813
Epoch 10/20
60000/60000 [==============================] - 11s 190us/step - loss:
0.0221 - acc: 0.9927 - val_loss: 0.0830 - val_acc: 0.9771
Epoch 11/20
```

```
60000/60000 [==============================] - 12s 193us/step - loss:
0.0196 - acc: 0.9934 - val_loss: 0.0781 - val_acc: 0.9785
Epoch 12/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0219 - acc: 0.9927 - val_loss: 0.0771 - val_acc: 0.9801
Epoch 13/20
60000/60000 [==============================] - 12s 192us/step - loss:
0.0195 - acc: 0.9933 - val_loss: 0.0695 - val_acc: 0.9814
Epoch 14/20
60000/60000 [==============================] - 11s 191us/step - loss:
0.0175 - acc: 0.9936 - val_loss: 0.0773 - val_acc: 0.9795
Epoch 15/20
60000/60000 [==============================] - 12s 194us/step - loss:
0.0165 - acc: 0.9945 - val_loss: 0.0785 - val_acc: 0.9801
Epoch 16/20
60000/60000 [==============================] - 12s 192us/step - loss:
0.0139 - acc: 0.9955 - val_loss: 0.0672 - val_acc: 0.9833
Epoch 17/20
60000/60000 [==============================] - 11s 189us/step - loss:
0.0160 - acc: 0.9950 - val_loss: 0.0799 - val_acc: 0.9806
Epoch 18/20
60000/60000 [==============================] - 11s 187us/step - loss:
0.0133 - acc: 0.9953 - val_loss: 0.0817 - val_acc: 0.9790
Epoch 19/20
60000/60000 [==============================] - 11s 190us/step - loss:
0.0125 - acc: 0.9959 - val_loss: 0.0855 - val_acc: 0.9800
Epoch 20/20
60000/60000 [==============================] - 11s 189us/step - loss:
0.0139 - acc: 0.9957 - val_loss: 0.0892 - val_acc: 0.9783
Test score: 0.08923894531318219
Test accuracy: 0.9783
```



(9)MLP_WITH_5_LAYER_+_RELU_+_ADAM_+_BM+_Dropout

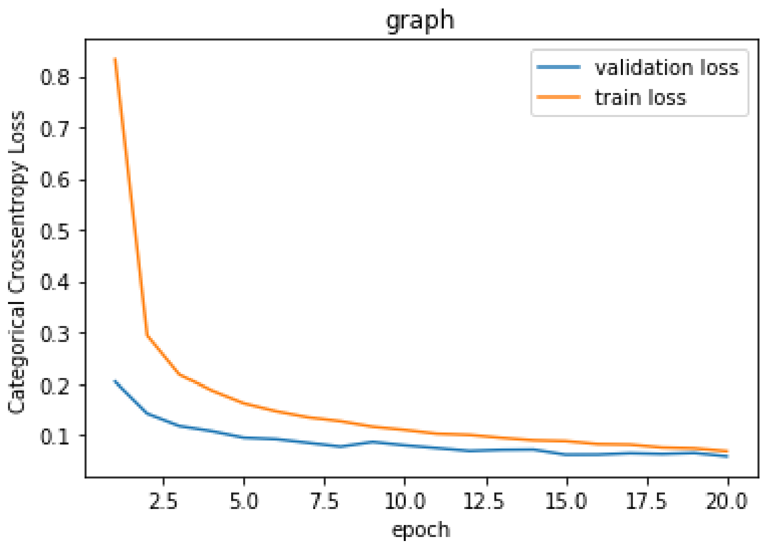
In [69]:
```
#model building

#model 9:mlp with 5 layer + relu activation function +adam optimizer +B
N

#using 712 neurons ,556 neurons ,438 neurons ,312 neurons ,144 neurons
 in h1 layer h2 layer ,h3 layers ,h4 layer and h5 layer
#using relu function in h1 , h2 , h3, h4 , h5
#using softmax layer in output layer

# https://arxiv.org/pdf/1707.09725.pdf#page=95
# for relu layers
# If we sample weights from a normal distribution N(0,σ) we satisfy thi
s condition with σ=√(2/(ni).
# h1 =>  σ=√(2/(fan_in) =σ=√(2/(712)= 0.052  => N(0,σ) = N(0,0.052)
# h2 =>  σ=√(2/(fan_in) =σ=√(2/(556)= 0.059  => N(0,σ) = N(0,0.059)
# h3 =>  σ=√(2/(fan_in) =σ=√(2/(438)= 0.067  => N(0,σ) = N(0,0.067)
```

```
# h4 =>  σ=√(2/(fan_in) =σ=√(2/(312)= 0.080  => N(0,σ) = N(0,0.080)
# h5 =>  σ=√(2/(fan_in) =σ=√(2/(144)= 0.11  => N(0,σ) = N(0,0.11)

model_9 = Sequential()
model_9.add(Dense(712, activation='relu', input_shape=(input_dim,), ker
nel_initializer=RandomNormal(mean=0.0, stddev=0.052, seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5))
model_9.add(Dense(556, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.059, seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5))
model_9.add(Dense(438, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.067, seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5))
model_9.add(Dense(312, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.080, seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5))
model_9.add(Dense(144, activation='relu', kernel_initializer=RandomNorm
al(mean=0.0, stddev=0.11, seed=None)))
model_9.add(BatchNormalization())
model_9.add(Dropout(0.5))
model_9.add(Dense(output_dim, activation='softmax'))



#compiling model by adding adam optimizer
model_9.compile(optimizer='adam', loss='categorical_crossentropy', metr
ics=['accuracy'])
history = model_9.fit(X_train, Y_train, batch_size=batch_size, epochs=n
b_epoch, verbose=1, validation_data=(X_test, Y_test))


##plotting train ,test vs epoch graph
##calculating accuracy

score = model_9.evaluate(X_test, Y_test, verbose=0)
```

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

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

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

vy = history.history['val_loss']#validation loss
ty = history.history['loss']#training loss
plotting_graph(x,vy,ty)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 19s 322us/step - loss:
0.8326 - acc: 0.7453 - val_loss: 0.2039 - val_acc: 0.9392
Epoch 2/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.2941 - acc: 0.9151 - val_loss: 0.1418 - val_acc: 0.9580
Epoch 3/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.2176 - acc: 0.9374 - val_loss: 0.1175 - val_acc: 0.9663
Epoch 4/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.1868 - acc: 0.9469 - val_loss: 0.1074 - val_acc: 0.9700
Epoch 5/20
60000/60000 [==============================] - 12s 199us/step - loss:
0.1618 - acc: 0.9542 - val_loss: 0.0945 - val_acc: 0.9730
Epoch 6/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.1464 - acc: 0.9586 - val_loss: 0.0918 - val_acc: 0.9739
Epoch 7/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.1345 - acc: 0.9619 - val_loss: 0.0846 - val_acc: 0.9755
Epoch 8/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.1269 - acc: 0.9648 - val_loss: 0.0771 - val_acc: 0.9793
Epoch 9/20
60000/60000 [==============================] - 12s 193us/step - loss:
```

```
0.1161 - acc: 0.9664 - val_loss: 0.0863 - val_acc: 0.9754
Epoch 10/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.1098 - acc: 0.9682 - val_loss: 0.0797 - val_acc: 0.9775
Epoch 11/20
60000/60000 [==============================] - 12s 195us/step - loss:
0.1024 - acc: 0.9699 - val_loss: 0.0743 - val_acc: 0.9791
Epoch 12/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.1001 - acc: 0.9712 - val_loss: 0.0687 - val_acc: 0.9817
Epoch 13/20
60000/60000 [==============================] - 12s 197us/step - loss:
0.0943 - acc: 0.9733 - val_loss: 0.0708 - val_acc: 0.9800
Epoch 14/20
60000/60000 [==============================] - 12s 198us/step - loss:
0.0895 - acc: 0.9745 - val_loss: 0.0713 - val_acc: 0.9812
Epoch 15/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.0880 - acc: 0.9751 - val_loss: 0.0614 - val_acc: 0.9839
Epoch 16/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0821 - acc: 0.9767 - val_loss: 0.0615 - val_acc: 0.9825
Epoch 17/20
60000/60000 [==============================] - 12s 201us/step - loss:
0.0811 - acc: 0.9764 - val_loss: 0.0642 - val_acc: 0.9831
Epoch 18/20
60000/60000 [==============================] - 12s 200us/step - loss:
0.0754 - acc: 0.9784 - val_loss: 0.0628 - val_acc: 0.9843
Epoch 19/20
60000/60000 [==============================] - 12s 196us/step - loss:
0.0738 - acc: 0.9781 - val_loss: 0.0647 - val_acc: 0.9820
Epoch 20/20
60000/60000 [==============================] - 12s 201us/step - loss:
0.0682 - acc: 0.9800 - val_loss: 0.0581 - val_acc: 0.9841
Test score: 0.058116857163724486
Test accuracy: 0.9841
```



In [71]:
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model_Number","Architecture","Test_Score","Test_Accur
acy"]
x.add_row(["Model_1","2_LAYER_+_RELU_+_ADAM",0.101,0.9794])
x.add_row(["Model_2","2_LAYER_+_RELU_+_ADAM_+_BN",0.092,0.9793])
x.add_row(["Model_3","2_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT",0.056,0.984
8])
x.add_row(["Model_4","3_LAYER_+_RELU_+_ADAM",0.090,0.9817])
x.add_row(["Model_5","3_LAYER_+_RELU_+_ADAM_+_BN",0.072,0.9840])
x.add_row(["Model_6","3_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT",0.052,0.984
2])
x.add_row(["Model_7","5_LAYER_+_RELU_+_ADAM",0.108,0.9803])
x.add_row(["Model_8","5_LAYER_+_RELU_+_ADAM_+BN",0.089,0.9783])
x.add_row(["Model_9","5_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT",0.058,0.984
1])
print(x)
```

```
+--------------+------------------------------------+------------+---
------------+
```

Model_Number	Architecture	Test_Score	Test_Accuracy
Model_1	2_LAYER_+_RELU_+_ADAM	0.101	0.9794
Model_2	2_LAYER_+_RELU_+_ADAM_+_BN	0.092	0.9793
Model_3	2_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT	0.056	0.9848
Model_4	3_LAYER_+_RELU_+_ADAM	0.09	0.9817
Model_5	3_LAYER_+_RELU_+_ADAM_+_BN	0.072	0.984
Model_6	3_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT	0.052	0.9842
Model_7	5_LAYER_+_RELU_+_ADAM	0.108	0.9803
Model_8	5_LAYER_+_RELU_+_ADAM_+BN	0.089	0.9783
Model_9	5_LAYER_+_RELU_+_ADAM_+_BN_+_DROPOUT	0.058	0.9841