14_keras_mnist

November 30, 2020

0.1 2,3,5 hidden layer architecture on MNIST dataset

```
In [1]: import tensorflow as tf
        from tensorflow.keras import utils
        from tensorflow.keras.datasets import mnist
        import seaborn as sns
        from tensorflow.keras.initializers import he_normal
```

```
In [2]: # the data, shuffled and split between train and test sets
        (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
```

```
In [3]: print("Number of X_train points :", X_train.shape)
        print("Number of y_train points :", y_train.shape)
        print("Number of X_test points :", X_test.shape)
        print("Number of y_test points :", y_test.shape)
```

```
Number of X_train points : (60000, 28, 28)
Number of y_train points : (60000,)
Number of X_test points : (10000, 28, 28)
Number of y_test points : (10000,)
```

```
In [4]: # 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.shape[2])
        X_test = X_test.reshape(X_test.shape[0], X_test.shape[1]*X_test.shape[2])
```

```
In [5]: # after converting the input images from 3d to 2d vectors

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

```
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 [6]: # 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  18 126 136 175  26 166 255
 247 127   0   0   0   0   0   0   0   0   0   0   0   0  30  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  49 238 253 253 253 253 253 253 253 253 251  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  11   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   0 139 253 190   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  25   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  93 252 253 187
   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  46 130 183 253
 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  24 114 221 253 253 253
 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  18 171 219 253 253 253 253 195
  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   0 136 253 253 253 212 135 132  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 [7]: # 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 the data
        # X => (X - Xmin)/(Xmax-Xmin) = X/255

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

```
In [8]: # 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.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333333 0.68627451 0.10196078 0.65098039 1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.11764706 0.14117647 0.36862745 0.60392157
 0.66666667 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.88235294 0.6745098  0.99215686 0.94901961 0.76470588 0.25098039
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.19215686
 0.93333333 0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
 0.99215686 0.99215686 0.99215686 0.98431373 0.36470588 0.32156863
 0.32156863 0.21960784 0.15294118 0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

```
0.         0.         0.         0.07058824 0.85882353 0.99215686
0.99215686 0.99215686 0.99215686 0.99215686 0.77647059 0.71372549
0.96862745 0.94509804 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.31372549 0.61176471 0.41960784 0.99215686
0.99215686 0.80392157 0.04313725 0.         0.16862745 0.60392157
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.05490196 0.00392157 0.60392157 0.99215686 0.35294118
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.54509804 0.99215686 0.74509804 0.00784314 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.04313725
0.74509804 0.99215686 0.2745098  0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.1372549  0.94509804
0.88235294 0.62745098 0.42352941 0.00392157 0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.31764706 0.94117647 0.99215686
0.99215686 0.46666667 0.09803922 0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.17647059 0.72941176 0.99215686 0.99215686
0.58823529 0.10588235 0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.0627451  0.36470588 0.98823529 0.99215686 0.73333333
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.97647059 0.99215686 0.97647059 0.25098039 0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
0.         0.         0.         0.         0.         0.
```

```
 0.         0.         0.18039216 0.50980392 0.71764706 0.99215686
 0.99215686 0.81176471 0.00784314 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.15294118 0.58039216
 0.89803922 0.99215686 0.99215686 0.99215686 0.98039216 0.71372549
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.09411765 0.44705882 0.86666667 0.99215686 0.99215686 0.99215686
 0.99215686 0.78823529 0.30588235 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882353 0.83529412 0.99215686
 0.99215686 0.99215686 0.99215686 0.77647059 0.31764706 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.07058824 0.67058824
 0.85882353 0.99215686 0.99215686 0.99215686 0.99215686 0.76470588
 0.31372549 0.03529412 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.21568627 0.6745098  0.88627451 0.99215686 0.99215686 0.99215686
 0.99215686 0.95686275 0.52156863 0.04313725 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.53333333 0.99215686
 0.99215686 0.99215686 0.83137255 0.52941176 0.51764706 0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         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 [9]: # 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 = utils.to_categorical(y_train, 10)
         Y_test = 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 [10]: # some model parameters
         from tensorflow.keras.models import Sequential
         from tensorflow.keras.layers import Dense, Activation

         output_dim = 10
         input_dim = X_train.shape[1]

         batch_size = 128
         nb_epoch = 20
```

0.2 Two hidden layer

0.2.1 MLP + ReLU + ADAM + BN + w/o Dropout +2Layer

```
In [11]: from tensorflow.keras.layers import BatchNormalization
         initializer = tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.066, seed=None)

         model_2lbn = Sequential() #2lbn=2 layer batch normalization

         model_2lbn.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
         model_2lbn.add(BatchNormalization())

         model_2lbn.add(Dense(128, activation='relu', kernel_initializer=initializer) )
         model_2lbn.add(BatchNormalization())

         model_2lbn.add(Dense(output_dim, activation='softmax'))


         model_2lbn.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
batch_normalization (BatchNo (None, 512)               2048
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
batch_normalization_1 (Batch (None, 128)               512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
_________________________________________________________________
```

```
In [12]: model_2lbn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

         history = model_2lbn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Epoch 1/20
469/469 [==============================] - 4s 9ms/step - loss: 0.1957 - accuracy: 0.9419 - val_
Epoch 2/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0712 - accuracy: 0.9786 - val_
Epoch 3/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0444 - accuracy: 0.9864 - val_
Epoch 4/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0308 - accuracy: 0.9905 - val_
Epoch 5/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0220 - accuracy: 0.9932 - val_
Epoch 6/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0213 - accuracy: 0.9932 - val_
Epoch 7/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0201 - accuracy: 0.9932 - val_
Epoch 8/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0136 - accuracy: 0.9956 - val_
Epoch 9/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0156 - accuracy: 0.9947 - val_
Epoch 10/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0140 - accuracy: 0.9952 - val_
Epoch 11/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0090 - accuracy: 0.9973 - val_
Epoch 12/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0115 - accuracy: 0.9964 - val_
Epoch 13/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0123 - accuracy: 0.9961 - val_
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0088 - accuracy: 0.9973 - val_
```

```
Epoch 15/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0084 - accuracy: 0.9973 - val_
Epoch 16/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0065 - accuracy: 0.9977 - val_
Epoch 17/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0116 - accuracy: 0.9963 - val_
Epoch 18/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0094 - accuracy: 0.9970 - val_
Epoch 19/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0064 - accuracy: 0.9978 - val_
Epoch 20/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0073 - accuracy: 0.9975 - val_
```

```
In [13]: #plotting function
         %matplotlib notebook
         import matplotlib.pyplot as plt
         %matplotlib inline
         import numpy as np

         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()
```

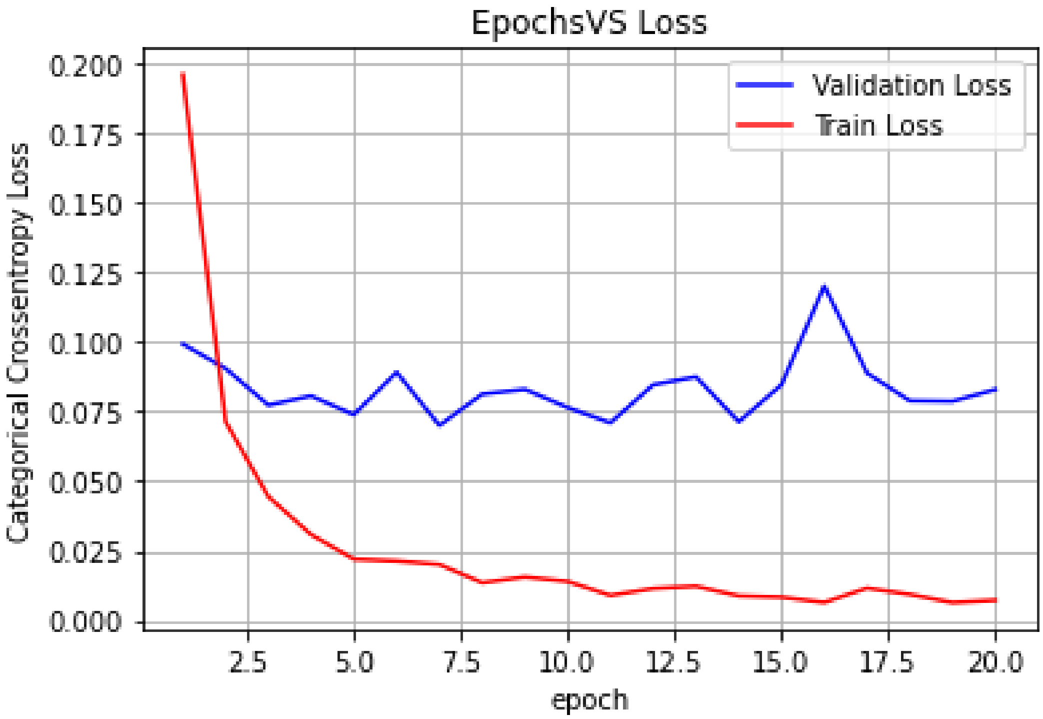
```
In [14]: score = model_2lbn.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])


         fig,ax = plt.subplots(1,1)
         ax.set_title('EpochsVS Loss')
         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']
         ty = history.history['loss']
         plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08280453830957413
Test accuracy: 0.9803000092506409
```



0.2.2 MLP + ReLU + ADAM + BN + with Dropout + 2Layer

```
In [15]: from tensorflow.keras.layers import Dropout
         from tensorflow.keras.layers import BatchNormalization
         initializer = tf.keras.initializers.he_normal(seed=None)

         model_2lbnd = Sequential() #model_2lbnd=2 layer batch normalization dropout

         model_2lbnd.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initial
         model_2lbnd.add(BatchNormalization())
         model_2lbnd.add(Dropout(0.5))

         model_2lbnd.add(Dense(128, activation='relu', kernel_initializer=initializer) )
         model_2lbnd.add(BatchNormalization())
         model_2lbnd.add(Dropout(0.5))

         model_2lbnd.add(Dense(output_dim, activation='softmax'))


         model_2lbnd.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_3 (Dense)              (None, 512)               401920
-----------------------------------------------------------------
batch_normalization_2 (Batch (None, 512)               2048
-----------------------------------------------------------------
dropout (Dropout)            (None, 512)               0
-----------------------------------------------------------------
dense_4 (Dense)              (None, 128)               65664
-----------------------------------------------------------------
batch_normalization_3 (Batch (None, 128)               512
-----------------------------------------------------------------
dropout_1 (Dropout)          (None, 128)               0
-----------------------------------------------------------------
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 471,434
Trainable params: 470,154
Non-trainable params: 1,280
-----------------------------------------------------------------
```

```
In [16]: model_2lbnd.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accu

         history = model_2lbnd.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Epoch 1/20
469/469 [==============================] - 5s 10ms/step - loss: 0.4309 - accuracy: 0.8677 - val
Epoch 2/20
469/469 [==============================] - 5s 10ms/step - loss: 0.2066 - accuracy: 0.9382 - val
Epoch 3/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1592 - accuracy: 0.9523 - val
Epoch 4/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1346 - accuracy: 0.9592 - val
Epoch 5/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1214 - accuracy: 0.9624 - val
Epoch 6/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1101 - accuracy: 0.9665 - val
Epoch 7/20
469/469 [==============================] - 5s 10ms/step - loss: 0.1000 - accuracy: 0.9691 - val
Epoch 8/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0965 - accuracy: 0.9702 - val
Epoch 9/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0853 - accuracy: 0.9739 - val
Epoch 10/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0831 - accuracy: 0.9740 - val
Epoch 11/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0774 - accuracy: 0.9759 - val
Epoch 12/20
```

```
469/469 [==============================] - 5s 10ms/step - loss: 0.0754 - accuracy: 0.9763 - val
Epoch 13/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0697 - accuracy: 0.9776 - val
Epoch 14/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0695 - accuracy: 0.9785 - val
Epoch 15/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0665 - accuracy: 0.9792 - val
Epoch 16/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0628 - accuracy: 0.9807 - val
Epoch 17/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0636 - accuracy: 0.9801 - val
Epoch 18/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0578 - accuracy: 0.9810 - val
Epoch 19/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0558 - accuracy: 0.9821 - val
Epoch 20/20
469/469 [==============================] - 5s 10ms/step - loss: 0.0542 - accuracy: 0.9831 - val
```

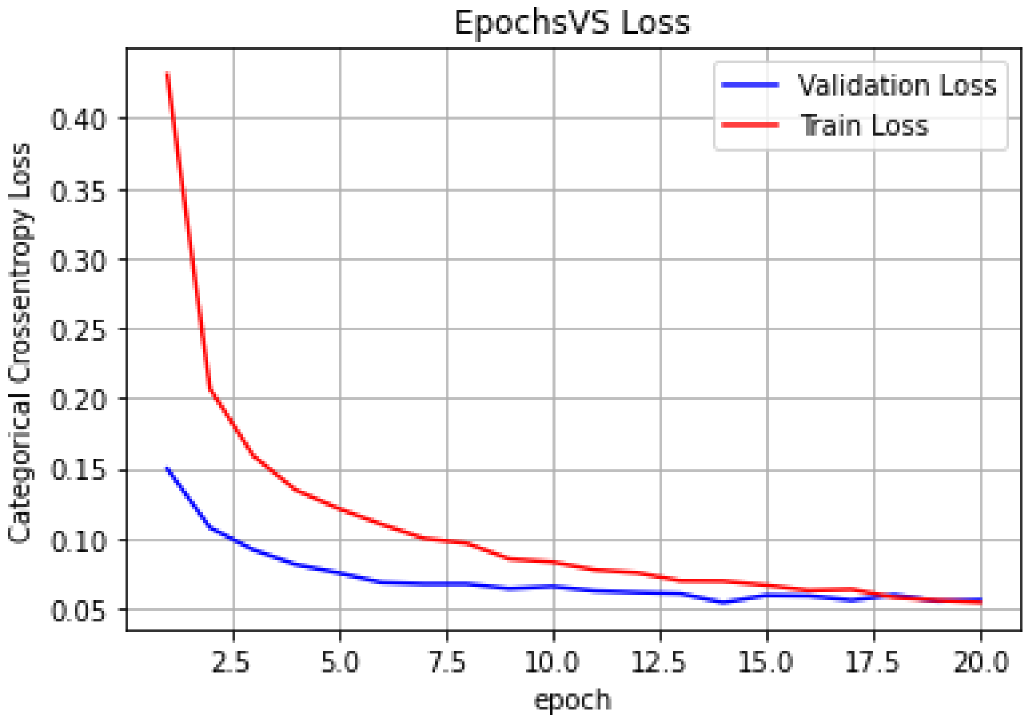
```
In [17]: score = model_2lbnd.evaluate(X_test, Y_test, verbose=0)
         print('Test score:', score[0])
         print('Test accuracy:', score[1])


         fig,ax = plt.subplots(1,1)
         ax.set_title('EpochsVS Loss')
         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']
         ty = history.history['loss']
         plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.05603070184588432
Test accuracy: 0.9840999841690063
```



0.3 Three hidden layer

0.3.1 MLP + ReLU + ADAM + w/o Dropout + BN + 3Layer

```
In [ ]: initializer = tf.keras.initializers.he_normal(seed=None)

        model_3lbn = Sequential()

        model_3lbn.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_3lbn.add(BatchNormalization())

        model_3lbn.add(Dense(128, activation='relu', kernel_initializer=initializer) )  #layer
        model_3lbn.add(BatchNormalization())

        model_3lbn.add(Dense(100, activation='relu', kernel_initializer=initializer) ) #layer
        model_3lbn.add(BatchNormalization())

        model_3lbn.add(Dense(output_dim, activation='softmax'))


        model_3lbn.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 512)               401920

_________________________________________________________________
batch_normalization_9 (Batch (None, 512)               2048

_________________________________________________________________
dense_15 (Dense)             (None, 128)               65664

_________________________________________________________________
batch_normalization_10 (Batc (None, 128)               512

_________________________________________________________________
dense_16 (Dense)             (None, 100)               12900

_________________________________________________________________
batch_normalization_11 (Batc (None, 100)               400

_________________________________________________________________
dense_17 (Dense)             (None, 10)                1010
=================================================================
Total params: 484,454
Trainable params: 482,974
Non-trainable params: 1,480

_________________________________________________________________
```

```
In [ ]: model_3lbn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_3lbn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, verl
```

```
Epoch 1/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2015 - accuracy: 0.9404 - val
Epoch 2/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0771 - accuracy: 0.9771 - val
Epoch 3/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0515 - accuracy: 0.9839 - val
Epoch 4/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0359 - accuracy: 0.9886 - val
Epoch 5/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0292 - accuracy: 0.9908 - val
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0296 - accuracy: 0.9899 - val
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0211 - accuracy: 0.9932 - val
Epoch 8/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0184 - accuracy: 0.9937 - val
Epoch 9/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0213 - accuracy: 0.9929 - val
Epoch 10/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0150 - accuracy: 0.9950 - val
Epoch 11/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0137 - accuracy: 0.9956 - val
```

```
Epoch 12/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0162 - accuracy: 0.9944 - val
Epoch 13/20
469/469 [==============================] - 5s 12ms/step - loss: 0.0132 - accuracy: 0.9957 - val
Epoch 14/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0112 - accuracy: 0.9960 - val
Epoch 15/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0091 - accuracy: 0.9970 - val
Epoch 16/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0129 - accuracy: 0.9957 - val
Epoch 17/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0111 - accuracy: 0.9963 - val
Epoch 18/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0086 - accuracy: 0.9973 - val
Epoch 19/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0076 - accuracy: 0.9976 - val
Epoch 20/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0093 - accuracy: 0.9970 - val
```

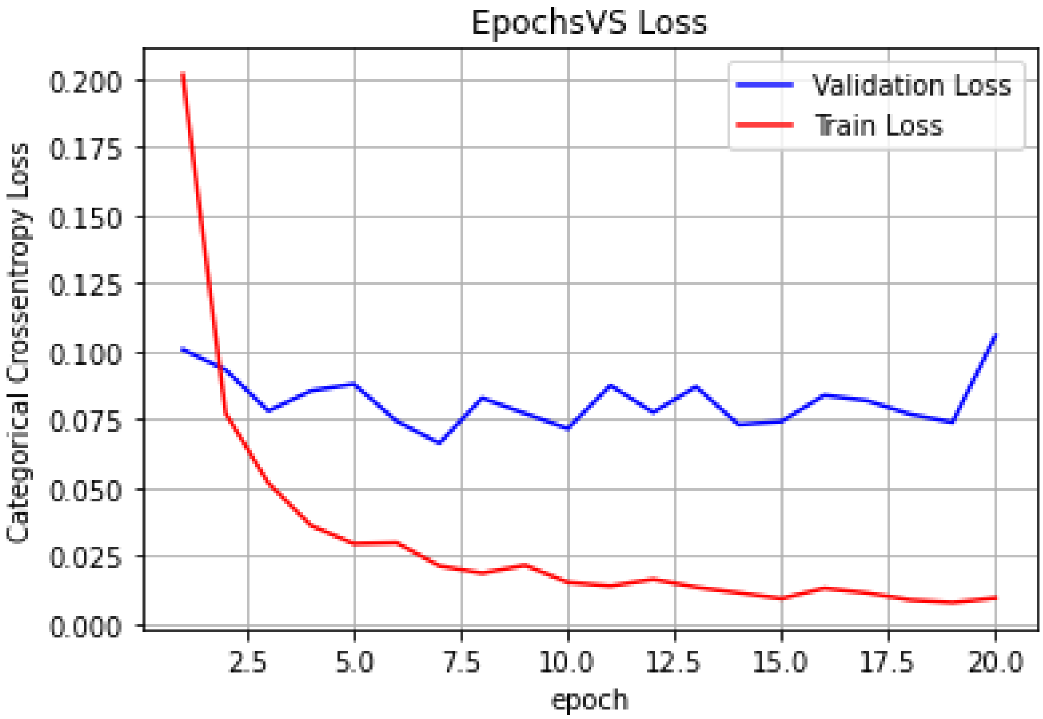
```
In [ ]: score = model_3lbn.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        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']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.105586476624012
Test accuracy: 0.9764000177383423
```



0.3.2 MLP + ReLU + ADAM + with Dropout + BN + 3Layer

```
In [ ]: initializer = tf.keras.initializers.he_normal(seed=None)

        model_3lbnd = Sequential()

        model_3lbnd.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_3lbnd.add(BatchNormalization())
        model_3lbnd.add(Dropout(0.5))

        model_3lbnd.add(Dense(128, activation='relu', kernel_initializer=initializer) )  #laye
        model_3lbnd.add(BatchNormalization())
        model_3lbnd.add(Dropout(0.5))

        model_3lbnd.add(Dense(100, activation='relu', kernel_initializer=initializer) ) #layer
        model_3lbnd.add(BatchNormalization())
        model_3lbnd.add(Dropout(0.5))

        model_3lbnd.add(Dense(output_dim, activation='softmax'))


        model_3lbnd.summary()
```

```
Model: "sequential_7"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 512)               401920
_________________________________________________________________
batch_normalization_12 (Batc (None, 512)               2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_20 (Dense)             (None, 100)               12900
_________________________________________________________________
batch_normalization_14 (Batc (None, 100)               400
_________________________________________________________________
dropout_4 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_21 (Dense)             (None, 10)                1010
=================================================================
Total params: 484,454
Trainable params: 482,974
Non-trainable params: 1,480
_________________________________________________________________
```

```
In [ ]: model_3lbnd.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_3lbnd.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Epoch 1/20
469/469 [==============================] - 6s 14ms/step - loss: 0.6442 - accuracy: 0.8026 - va
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2782 - accuracy: 0.9201 - val
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 0.2113 - accuracy: 0.9397 - val
Epoch 4/20
469/469 [==============================] - 6s 13ms/step - loss: 0.1809 - accuracy: 0.9477 - val
Epoch 5/20
469/469 [==============================] - 6s 13ms/step - loss: 0.1580 - accuracy: 0.9547 - val
Epoch 6/20
469/469 [==============================] - 6s 13ms/step - loss: 0.1457 - accuracy: 0.9582 - val
Epoch 7/20
469/469 [==============================] - 6s 13ms/step - loss: 0.1275 - accuracy: 0.9631 - val
Epoch 8/20
```

```
469/469 [==============================] - 7s 14ms/step - loss: 0.1243 - accuracy: 0.9639 - val
Epoch 9/20
469/469 [==============================] - 7s 14ms/step - loss: 0.1144 - accuracy: 0.9668 - val
Epoch 10/20
469/469 [==============================] - 7s 16ms/step - loss: 0.1059 - accuracy: 0.9696 - val
Epoch 11/20
469/469 [==============================] - 7s 14ms/step - loss: 0.1014 - accuracy: 0.9706 - val
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0948 - accuracy: 0.9727 - val
Epoch 13/20
469/469 [==============================] - 6s 14ms/step - loss: 0.0955 - accuracy: 0.9714 - val
Epoch 14/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0882 - accuracy: 0.9746 - val
Epoch 15/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0850 - accuracy: 0.9754 - val
Epoch 16/20
469/469 [==============================] - 6s 14ms/step - loss: 0.0779 - accuracy: 0.9770 - val
Epoch 17/20
469/469 [==============================] - 6s 14ms/step - loss: 0.0776 - accuracy: 0.9762 - val
Epoch 18/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0755 - accuracy: 0.9773 - val
Epoch 19/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0724 - accuracy: 0.9791 - val
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0678 - accuracy: 0.9804 - val
```

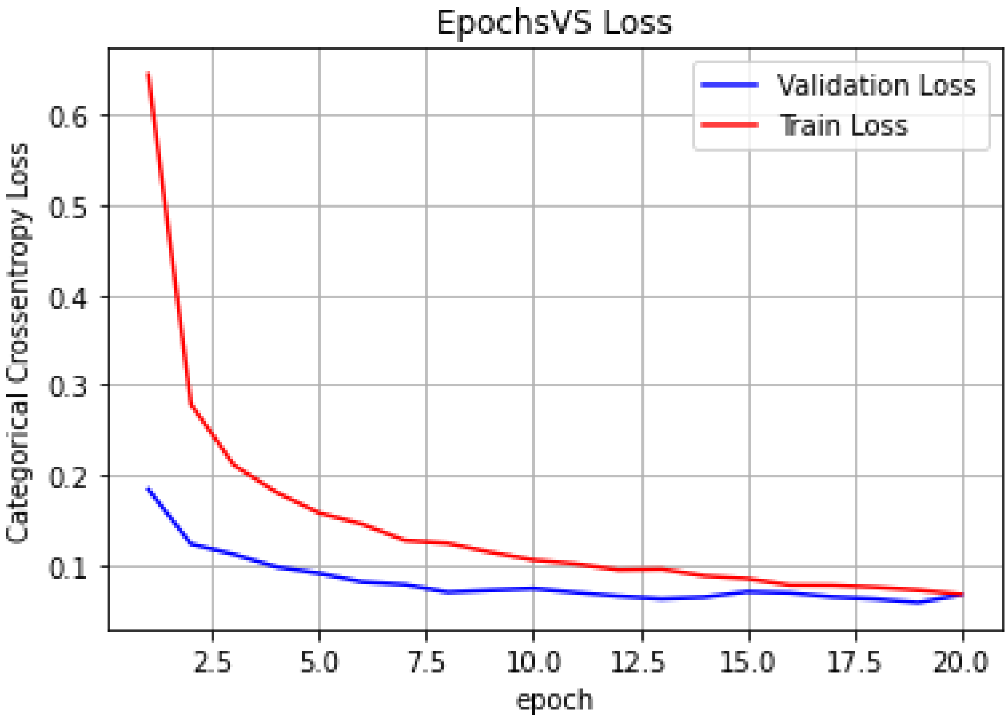
```
In [ ]: score = model_3lbnd.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        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']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.06718681752681732
Test accuracy: 0.9817000031471252
```



0.4 Five hidden layer

0.4.1 MLP + ReLU + ADAM + w/o Dropout + BN + 5Layer

```
In [ ]: initializer = tf.keras.initializers.he_normal(seed=None)

        model_5lbn = Sequential()

        model_5lbn.add(Dense(612, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_5lbn.add(BatchNormalization())

        model_5lbn.add(Dense(512, activation='relu', kernel_initializer=initializer) )  #layer
        model_5lbn.add(BatchNormalization())

        model_5lbn.add(Dense(312, activation='relu', kernel_initializer=initializer) ) #layer
        model_5lbn.add(BatchNormalization())

        model_5lbn.add(Dense(212, activation='relu', kernel_initializer=initializer) ) #layer
        model_5lbn.add(BatchNormalization())

        model_5lbn.add(Dense(15, activation='relu', kernel_initializer=initializer) ) #layer 3
        model_5lbn.add(BatchNormalization())

        model_5lbn.add(Dense(output_dim, activation='softmax'))
        model_5lbn.summary()

Model: "sequential_10"
-----------------------------------------------------------------
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 612)               480420
-----------------------------------------------------------------
batch_normalization_25 (Batc (None, 612)               2448
-----------------------------------------------------------------
dense_35 (Dense)             (None, 512)               313856
-----------------------------------------------------------------
batch_normalization_26 (Batc (None, 512)               2048
-----------------------------------------------------------------
dense_36 (Dense)             (None, 312)               160056
-----------------------------------------------------------------
batch_normalization_27 (Batc (None, 312)               1248
-----------------------------------------------------------------
dense_37 (Dense)             (None, 212)               66356
-----------------------------------------------------------------
batch_normalization_28 (Batc (None, 212)               848
-----------------------------------------------------------------
dense_38 (Dense)             (None, 15)                3195
-----------------------------------------------------------------
batch_normalization_29 (Batc (None, 15)                60
-----------------------------------------------------------------
dense_39 (Dense)             (None, 10)                160
=================================================================
Total params: 1,030,695
Trainable params: 1,027,369
Non-trainable params: 3,326
-----------------------------------------------------------------
```

```
In [ ]: model_5lbn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_5lbn.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ver
```

```
Epoch 1/20
469/469 [==============================] - 13s 27ms/step - loss: 0.2785 - accuracy: 0.9367 - va
Epoch 2/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0934 - accuracy: 0.9739 - va
Epoch 3/20
469/469 [==============================] - 12s 27ms/step - loss: 0.0626 - accuracy: 0.9814 - va
Epoch 4/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0506 - accuracy: 0.9841 - va
```

```
Epoch 5/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0405 - accuracy: 0.9875 - va
Epoch 6/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0332 - accuracy: 0.9893 - va
Epoch 7/20
469/469 [==============================] - 13s 28ms/step - loss: 0.0320 - accuracy: 0.9899 - va
Epoch 8/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0238 - accuracy: 0.9924 - va
Epoch 9/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0249 - accuracy: 0.9918 - va
Epoch 10/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0223 - accuracy: 0.9930 - va
Epoch 11/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0194 - accuracy: 0.9936 - va
Epoch 12/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0202 - accuracy: 0.9937 - va
Epoch 13/20
469/469 [==============================] - 13s 27ms/step - loss: 0.0172 - accuracy: 0.9945 - va
Epoch 14/20
469/469 [==============================] - 14s 29ms/step - loss: 0.0185 - accuracy: 0.9942 - va
Epoch 15/20
469/469 [==============================] - 13s 28ms/step - loss: 0.0138 - accuracy: 0.9954 - va
Epoch 16/20
469/469 [==============================] - 12s 26ms/step - loss: 0.0140 - accuracy: 0.9954 - va
Epoch 17/20
469/469 [==============================] - 13s 27ms/step - loss: 0.0140 - accuracy: 0.9954 - va
Epoch 18/20
469/469 [==============================] - 12s 27ms/step - loss: 0.0135 - accuracy: 0.9956 - va
Epoch 19/20
469/469 [==============================] - 12s 27ms/step - loss: 0.0131 - accuracy: 0.9958 - va
Epoch 20/20
469/469 [==============================] - 13s 28ms/step - loss: 0.0084 - accuracy: 0.9973 - va
```

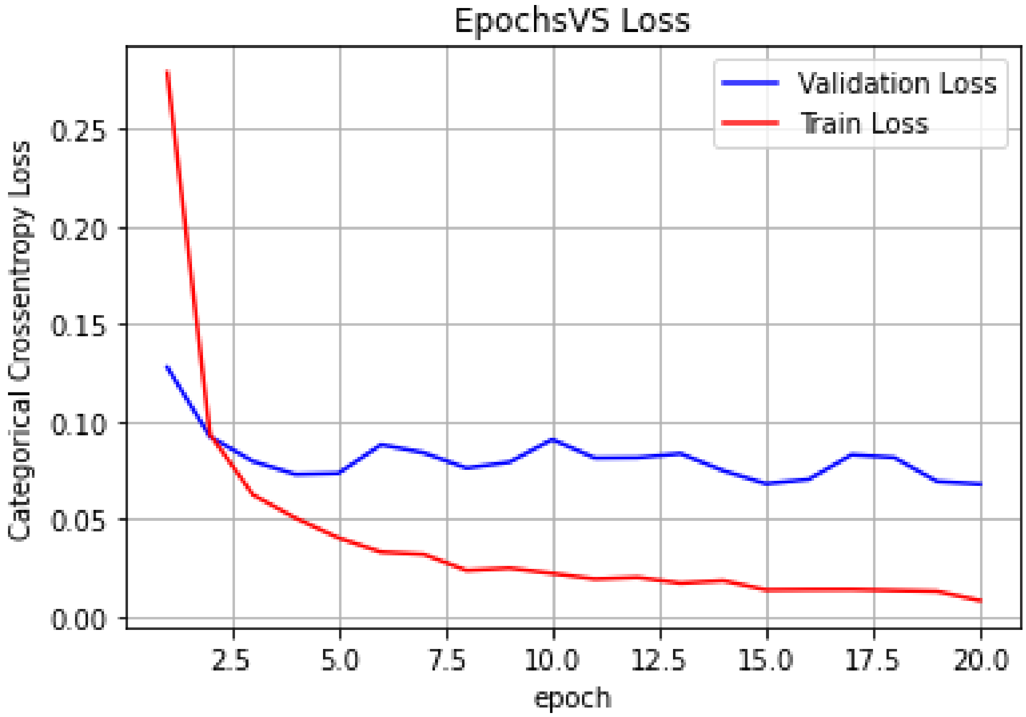
```
In [ ]: score = model_5lbn.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        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']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.0679909959435463
Test accuracy: 0.9822999835014343
```



0.4.2 MLP + ReLU + ADAM + BN + with Dropout + 5Layer

```
In [ ]: initializer = tf.keras.initializers.he_normal(seed=None)

        model_5lbnd = Sequential()

        model_5lbnd.add(Dense(612, activation='relu', input_shape=(input_dim,), kernel_initiali
        model_5lbnd.add(BatchNormalization())
        model_5lbnd.add(Dropout(0.5))

        model_5lbnd.add(Dense(512, activation='relu', kernel_initializer=initializer) )  #laye
        model_5lbnd.add(BatchNormalization())
        model_5lbnd.add(Dropout(0.5))

        model_5lbnd.add(Dense(312, activation='relu', kernel_initializer=initializer) ) #layer
        model_5lbnd.add(BatchNormalization())
        model_5lbnd.add(Dropout(0.5))
```

```
model_5lbnd.add(Dense(212, activation='relu', kernel_initializer=initializer) ) #layer
model_5lbnd.add(BatchNormalization())
model_5lbnd.add(Dropout(0.5))

model_5lbnd.add(Dense(15, activation='relu', kernel_initializer=initializer) ) #layer
model_5lbnd.add(BatchNormalization())
model_5lbnd.add(Dropout(0.5))

model_5lbnd.add(Dense(output_dim, activation='softmax'))


model_5lbnd.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_28 (Dense)             (None, 612)               480420

_________________________________________________________________
batch_normalization_20 (Batc (None, 612)               2448

_________________________________________________________________
dropout_5 (Dropout)          (None, 612)               0

_________________________________________________________________
dense_29 (Dense)             (None, 512)               313856

_________________________________________________________________
batch_normalization_21 (Batc (None, 512)               2048

_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0

_________________________________________________________________
dense_30 (Dense)             (None, 312)               160056

_________________________________________________________________
batch_normalization_22 (Batc (None, 312)               1248

_________________________________________________________________
dropout_7 (Dropout)          (None, 312)               0

_________________________________________________________________
dense_31 (Dense)             (None, 212)               66356

_________________________________________________________________
batch_normalization_23 (Batc (None, 212)               848

_________________________________________________________________
dropout_8 (Dropout)          (None, 212)               0

_________________________________________________________________
dense_32 (Dense)             (None, 15)                3195

_________________________________________________________________
batch_normalization_24 (Batc (None, 15)                60

_________________________________________________________________
dropout_9 (Dropout)          (None, 15)                0

_________________________________________________________________
```

```
dense_33 (Dense)             (None, 10)                160
=================================================================
Total params: 1,030,695
Trainable params: 1,027,369
Non-trainable params: 3,326
_________________________________________________________________
```

```
In [ ]: model_5lbnd.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accura

        history = model_5lbnd.fit(X_train, Y_train, batch_size=batch_size, epochs=nb_epoch, ve
```

```
Epoch 1/20
469/469 [==============================] - 14s 30ms/step - loss: 1.2392 - accuracy: 0.6089 - va
Epoch 2/20
469/469 [==============================] - 14s 29ms/step - loss: 0.5216 - accuracy: 0.8631 - va
Epoch 3/20
469/469 [==============================] - 14s 29ms/step - loss: 0.3929 - accuracy: 0.8996 - va
Epoch 4/20
469/469 [==============================] - 14s 29ms/step - loss: 0.3270 - accuracy: 0.9171 - va
Epoch 5/20
469/469 [==============================] - 13s 28ms/step - loss: 0.2912 - accuracy: 0.9244 - va
Epoch 6/20
469/469 [==============================] - 13s 29ms/step - loss: 0.2621 - accuracy: 0.9315 - va
Epoch 7/20
469/469 [==============================] - 14s 29ms/step - loss: 0.2509 - accuracy: 0.9339 - va
Epoch 8/20
469/469 [==============================] - 14s 29ms/step - loss: 0.2287 - accuracy: 0.9392 - va
Epoch 9/20
469/469 [==============================] - 13s 29ms/step - loss: 0.2177 - accuracy: 0.9425 - va
Epoch 10/20
469/469 [==============================] - 14s 29ms/step - loss: 0.2143 - accuracy: 0.9446 - va
Epoch 11/20
469/469 [==============================] - 14s 29ms/step - loss: 0.1945 - accuracy: 0.9482 - va
Epoch 12/20
469/469 [==============================] - 14s 29ms/step - loss: 0.1935 - accuracy: 0.9484 - va
Epoch 13/20
469/469 [==============================] - 13s 29ms/step - loss: 0.1842 - accuracy: 0.9506 - va
Epoch 14/20
469/469 [==============================] - 13s 29ms/step - loss: 0.1776 - accuracy: 0.9508 - va
Epoch 15/20
469/469 [==============================] - 14s 29ms/step - loss: 0.1720 - accuracy: 0.9523 - va
Epoch 16/20
469/469 [==============================] - 14s 29ms/step - loss: 0.1612 - accuracy: 0.9557 - va
Epoch 17/20
469/469 [==============================] - 13s 28ms/step - loss: 0.1627 - accuracy: 0.9561 - va
Epoch 18/20
469/469 [==============================] - 13s 28ms/step - loss: 0.1553 - accuracy: 0.9567 - va
```

```
Epoch 19/20
469/469 [==============================] - 13s 29ms/step - loss: 0.1518 - accuracy: 0.9589 - va
Epoch 20/20
469/469 [==============================] - 14s 29ms/step - loss: 0.1494 - accuracy: 0.9589 - va
```

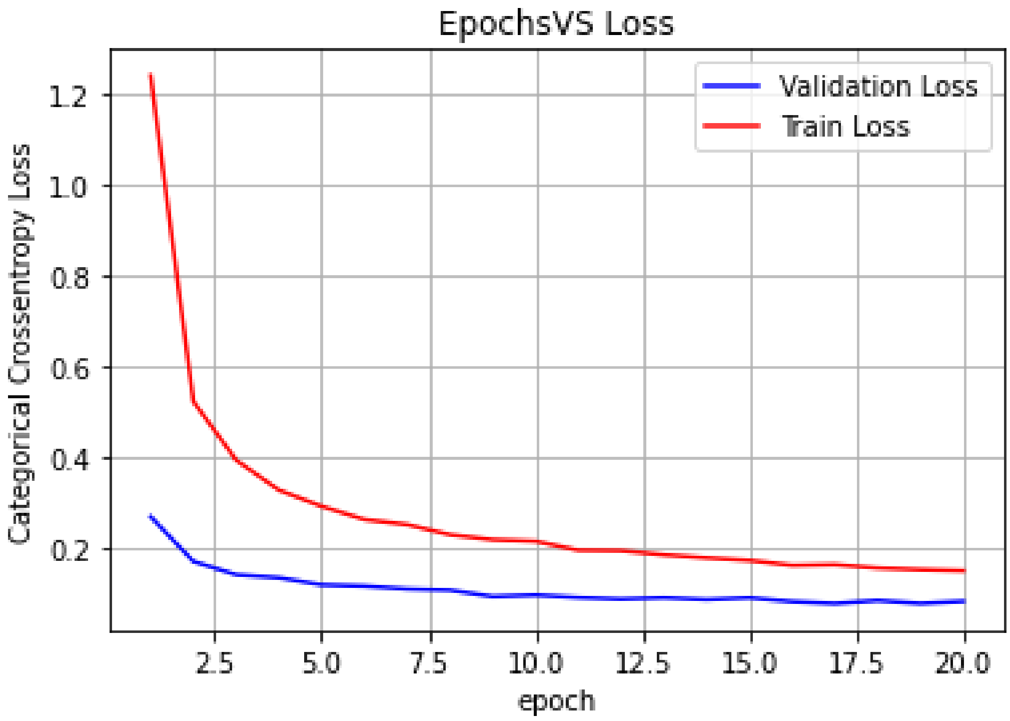
```
In [ ]: score = model_5lbnd.evaluate(X_test, Y_test, verbose=0)
        print('Test score:', score[0])
        print('Test accuracy:', score[1])


        fig,ax = plt.subplots(1,1)
        ax.set_title('EpochsVS Loss')
        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']
        ty = history.history['loss']
        plt_dynamic(x, vy, ty, ax)
```

```
Test score: 0.08210139721632004
Test accuracy: 0.9828000068664551
```



0.5 Conclusion

```
In [1]: from prettytable import PrettyTable
        x = PrettyTable()
        x.field_names = ["No. of layer","Dropout", "Accuracy %"]

        x.add_row(["2","NO","98.03"])
        x.add_row(["2","YES","98.40"])
        x.add_row(["3","NO","97.76"])
        x.add_row(["3","YES","98.17"])
        x.add_row(["5","NO","98.22"])
        x.add_row(["5","YES","98.28"])


        print(x)
```

```
+--------------+---------+------------+
| No. of layer | Dropout | Accuracy % |
+--------------+---------+------------+
|      2       |    NO   |   98.03    |
|      2       |   YES   |   98.40    |
|      3       |    NO   |   97.76    |
|      3       |   YES   |   98.17    |
|      5       |    NO   |   98.22    |
|      5       |   YES   |   98.28    |
+--------------+---------+------------+
```

1. Observed that using dropouts there is slight increasein the accuracy
2. 5 hidden layer model has given 98% accuracy and no much increment with/without dropout as dataset is small