Keras on MNIST datset

Objective : To apply multiple hidden layer [3,5,7,9...] MLP architecture on MNIST dataset

```python
import warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

# For plotting purposes
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from keras.utils import to_categorical
from keras.models import Sequential
from keras.initializers import he_normal
from keras.layers import BatchNormalization, Dense, Dropout

# Import MNIST Dataset
from keras.datasets import mnist
```

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

```python
print("x_train shape: ", x_train.shape)
print("x_test shape: ", x_test.shape)
print("Number of training examples :", x_train.shape[0], "and each image is of shape (%d, %d)"%(x_train
print("Number of testing examples :", x_test.shape[0], "and each image is of shape (%d, %d)"%(x_test.sha
```

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

```
#converting from 3d to 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])

# after converting the input images from 3d to 2d vectors
print("x_train shape: ", x_train.shape)
print("x_test shape: ", x_test.shape)
print("Number of training examples :", x_train.shape[0], "and each image is of shape (%d)"%(x_train.sha
print("Number of training examples :", x_test.shape[0], "and each image is of shape (%d)"%(x_test.shape
```

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

```
#Normalising data
minMaxScaler = MinMaxScaler()

x_train = minMaxScaler.fit_transform(x_train)
x_test = minMaxScaler.transform(x_test)

# x_train 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.
```

```
temp = y_train[0]
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)

print("After converting the output {0} into a vector : {1}".format(temp,y_train[0]))
```

```
After converting the output 5 into a vector : [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```python
# Function for plotting  train and cross validation loss
def plot_train_cv_loss(trained_model, epochs, colors=['b']):
    fig, ax = plt.subplots(1,1)
    ax.set_xlabel('epoch')
    ax.set_ylabel('Categorical Crossentropy Loss')
    x_axis_values = list(range(1,epochs+1))

    validation_loss = trained_model.history['val_loss']
    train_loss = trained_model.history['loss']

    ax.plot(x_axis_values, validation_loss, 'b', label="Validation Loss")
    ax.plot(x_axis_values, train_loss, 'r', label="Train Loss")
    plt.legend()
    plt.grid()
    fig.canvas.draw()
```

```python
# Defining batch size and epochs
# Batch size
batch_size = 128

# Number of time whole data is trained
epochs =50

# Input layer dimension
input_dimension = x_train.shape[1]

# Output layer dimension
output_dimension = y_train.shape[1]
```

Try 3 Hidden layer architecture

```
# Instantiate sequential model
model = Sequential()

# Add 1st hidden layer : dense Layer
dense_layer1 = Dense(750,
                       activation="relu",
                       input_shape=(input_dimension,),
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer1)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 2nd hidden layer : dense Layer
dense_layer2 = Dense(500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer2)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))


# Add 3rd hidden layer : dense Layer
dense_layer3 = Dense(250,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer3)
```

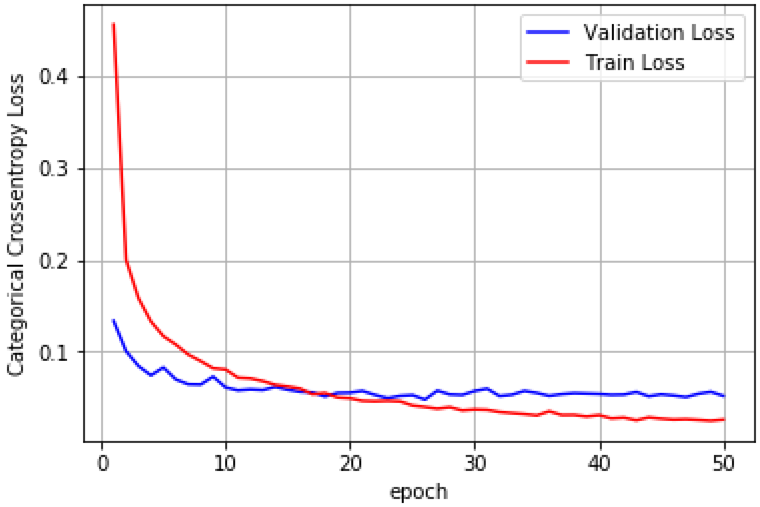
```

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add output layer : dense Layer
dense_layer4 = Dense(output_dimension, activation='softmax')
model.add(dense_layer4)

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, valida
```

```
60000/60000 [==============================] - 4s 61us/step - loss: 0.0399 - acc: 0.9876 - val_loss: 0.0480 - val_acc: 0.9
862
Epoch 27/50
60000/60000 [==============================] - 4s 73us/step - loss: 0.0381 - acc: 0.9876 - val_loss: 0.0578 - val_acc: 0.9
844
Epoch 28/50
60000/60000 [==============================] - 4s 60us/step - loss: 0.0398 - acc: 0.9874 - val_loss: 0.0536 - val_acc: 0.9
847
Epoch 29/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0362 - acc: 0.9882 - val_loss: 0.0530 - val_acc: 0.9
859
Epoch 30/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0372 - acc: 0.9879 - val_loss: 0.0574 - val_acc: 0.9
843
Epoch 31/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0368 - acc: 0.9877 - val_loss: 0.0599 - val_acc: 0.9
```

```
833
Epoch 32/50
60000/60000 [==============================] - 4s 62us/step - loss: 0.0345 - acc: 0.9889 - val_loss: 0.0518 - val_acc: 0.9
860
Epoch 33/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.0333 - acc: 0.9895 - val_loss: 0.0533 - val_acc: 0.9
```

```python
# Accuracy and plotting train & validation loss
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy: {0:.2f}%'.format(score[1]*100))


# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```

```
Test score: 0.05205892696186711
Test accuracy: 98.61%
```



Around 17 Epoch train error and validation error meets.

With 5 Hidden layer

```
# Instantiate sequential model
model = Sequential()

# Add 1st hidden layer : dense Layer
dense_layer1 = Dense(1150,
                       activation="relu",
                       input_shape=(input_dimension,),
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer1)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 2nd hidden layer : dense Layer
dense_layer2 = Dense(900,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer2)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))


# Add 3rd hidden layer : dense Layer
dense_layer3 = Dense(750,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer3)
```

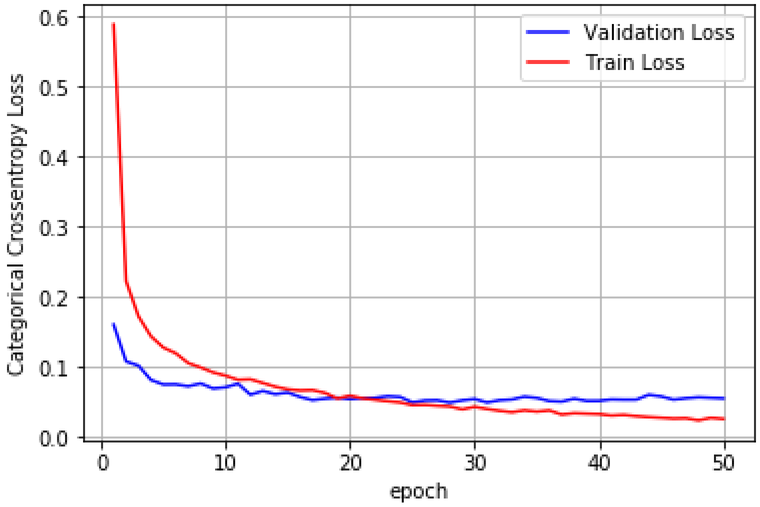
```

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 4th hidden layer : dense Layer
dense_layer4 = Dense(500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer4)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 5th hidden layer : dense Layer
dense_layer5 = Dense(250,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer5)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add output layer : dense Layer
dense_layer6 = Dense(output_dimension, activation='softmax')
model.add(dense_layer6)

```

```
# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, valida
```

```
Model Summary: 

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 1150)              902750    
_________________________________________________________________
batch_normalization_6 (Batch (None, 1150)              4600      
_________________________________________________________________
dropout_6 (Dropout)          (None, 1150)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 900)               1035900   
_________________________________________________________________
batch_normalization_7 (Batch (None, 900)               3600      
_________________________________________________________________
dropout_7 (Dropout)          (None, 900)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 750)               675750    
_________________________________________________________________
batch_normalization_8 (Batch (None, 750)               3000      
_________________________________________________________________
```

```
# Accuracy and plotting train & validation loss
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy: {0:.2f}%'.format(score[1]*100))


# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```

```
Test score: 0.05508785108429729
Test accuracy: 98.68%
```



Around 20 Epoch train error and validation error meets.

Try 7 Hidden Layer

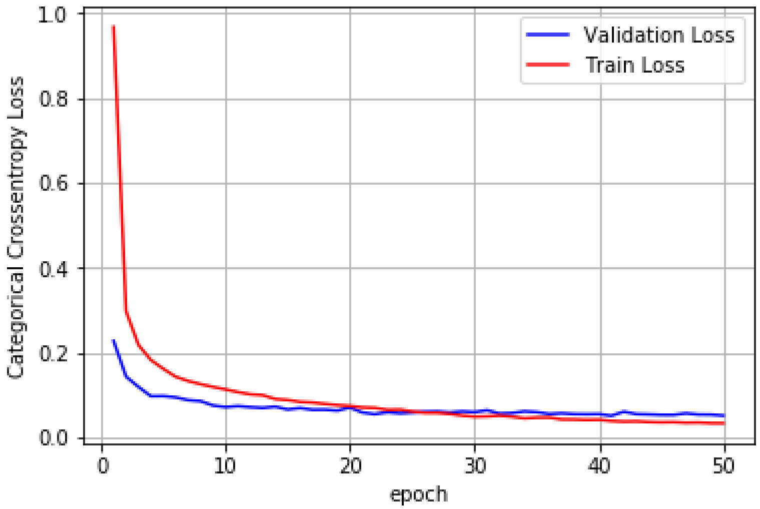
```
# Instantiate sequential model
model = Sequential()

# Add 1st hidden layer : dense Layer
dense_layer1 = Dense(1250,
                       activation="relu",
                       input_shape=(input_dimension,),
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer1)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 2nd hidden layer : dense Layer
dense_layer2 = Dense(1000,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer2)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))


# Add 3rd hidden layer : dense Layer
dense_layer3 = Dense(750,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer3)
```

```

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 4th hidden layer : dense Layer
dense_layer4 = Dense(500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer4)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 5th hidden layer : dense Layer
dense_layer5 = Dense(750,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer5)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 6th hidden layer : dense Layer
dense_layer6 = Dense(500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
```

```
model.add(dense_layer6)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 7th hidden layer : dense Layer
dense_layer7 = Dense(250,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer7)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add output layer : dense Layer
dense_layer8 = Dense(output_dimension, activation='softmax')
model.add(dense_layer8)

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, valida
60000/60000 [==============================] - 7s 111us/step - loss: 0.0464 - acc: 0.9865 - val_loss: 0.0547 - val_acc: 0.
9863
Epoch 37/50
60000/60000 [==============================] - 7s 112us/step - loss: 0.0418 - acc: 0.9880 - val_loss: 0.0563 - val_acc: 0.
9868
Epoch 38/50
60000/60000 [==============================] - 7s 113us/step - loss: 0.0416 - acc: 0.9876 - val_loss: 0.0544 - val_acc: 0.
9866
Epoch 39/50
60000/60000 [==============================] - 7s 113us/step - loss: 0.0406 - acc: 0.9882 - val_loss: 0.0542 - val_acc: 0.
9862
Epoch 40/50
60000/60000 [==============================] - 7s 112us/step - loss: 0.0411 - acc: 0.9880 - val_loss: 0.0545 - val_acc: 0.
9867
Epoch 41/50
```

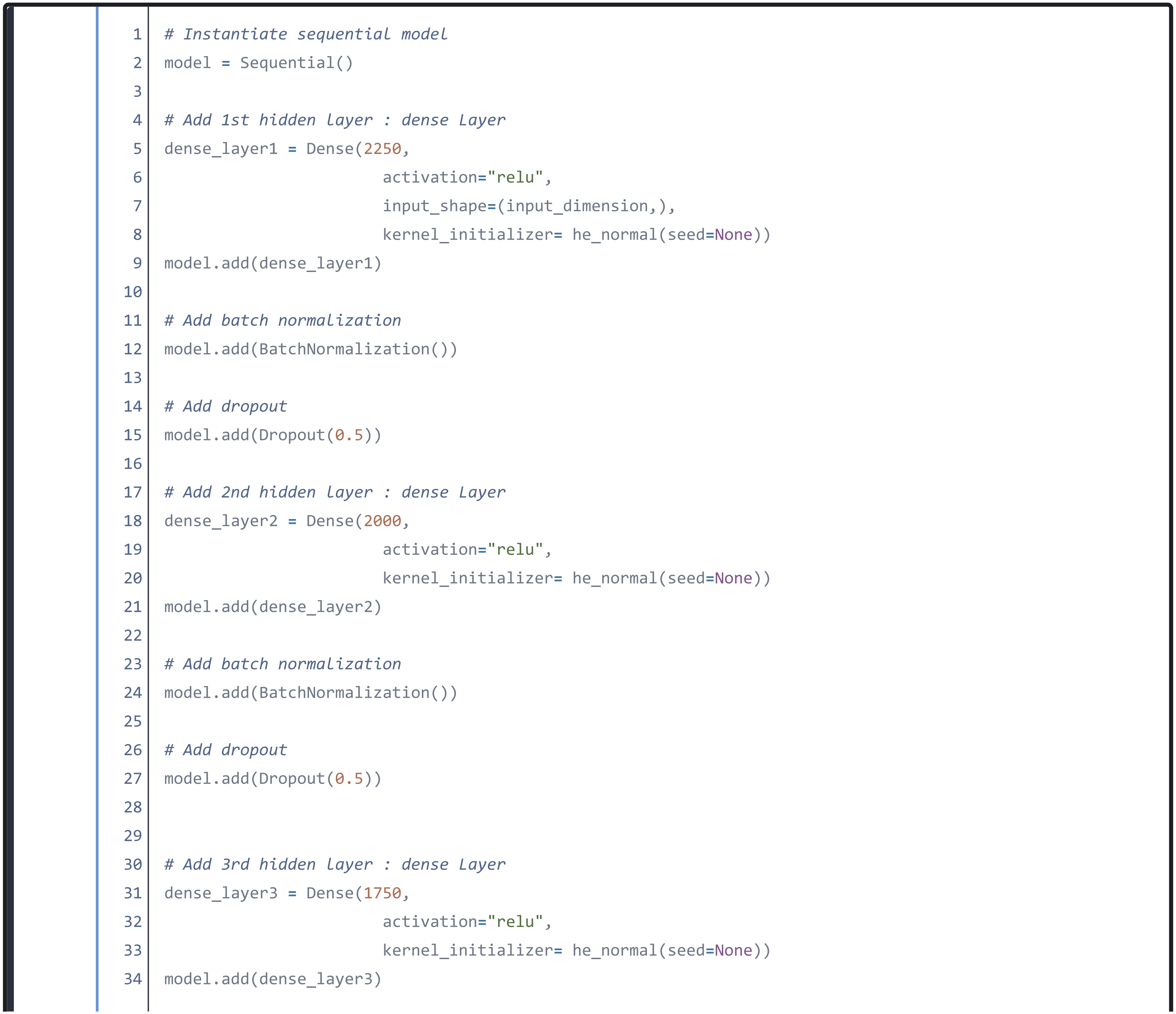
```
# Accuracy and plotting train & validation loss
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy: {0:.2f}%'.format(score[1]*100))


# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```

```
Test score: 0.050907338776078 54
Test accuracy: 98.70%
```



Around 22 Epoch train error and validation error meets.

Try 9 Hidden layer



```
# Instantiate sequential model
model = Sequential()

# Add 1st hidden layer : dense Layer
dense_layer1 = Dense(2250,
                       activation="relu",
                       input_shape=(input_dimension,),
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer1)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 2nd hidden layer : dense Layer
dense_layer2 = Dense(2000,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer2)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))


# Add 3rd hidden layer : dense Layer
dense_layer3 = Dense(1750,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer3)
```

```

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 4th hidden layer : dense Layer
dense_layer4 = Dense(1500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer4)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 5th hidden layer : dense Layer
dense_layer5 = Dense(1250,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer5)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 6th hidden layer : dense Layer
dense_layer6 = Dense(1000,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
```

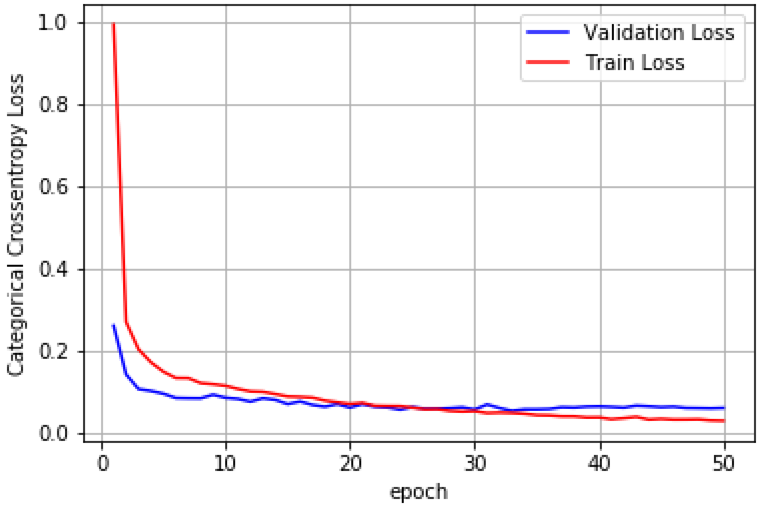
```
model.add(dense_layer6)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 7th hidden layer : dense Layer
dense_layer7 = Dense(750,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer7)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 8th hidden layer : dense Layer
dense_layer8 = Dense(500,
                       activation="relu",
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer8)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add 9th hidden layer : dense Layer
dense_layer9 = Dense(250,
                       activation="relu",
```

```python
                       kernel_initializer= he_normal(seed=None))
model.add(dense_layer9)

# Add batch normalization
model.add(BatchNormalization())

# Add dropout
model.add(Dropout(0.5))

# Add output layer : dense Layer
dense_layer10 = Dense(output_dimension, activation='softmax')
model.add(dense_layer10)

# Summary of the model
print("Model Summary: \n")
model.summary()
print()
print()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Run the model
trained_model = model.fit(x_train, y_train, batch_size = batch_size, epochs = epochs, verbose=1, valida
```

```
_________________________________________________________________
batch_normalization_20 (Batc (None, 1750)              7000      
_________________________________________________________________
dropout_20 (Dropout)         (None, 1750)              0         
_________________________________________________________________
dense_25 (Dense)             (None, 1500)              2626500   
_________________________________________________________________
batch_normalization_21 (Batc (None, 1500)              6000      
_________________________________________________________________
dropout_21 (Dropout)         (None, 1500)              0         
_________________________________________________________________
dense_26 (Dense)             (None, 1250)              1876250   
_________________________________________________________________
batch_normalization_22 (Batc (None, 1250)              5000      
batch_normalization_22 (Batc (None, 1250)              5000
_________________________________________________________________
dropout_22 (Dropout)         (None, 1250)              0
_________________________________________________________________
dense_27 (Dense)             (None, 1000)              1251000
_________________________________________________________________
batch_normalization_23 (Batc (None, 1000)              4000
_________________________________________________________________
dropout_23 (Dropout)         (None, 1000)              0
```

```
# Accuracy and plotting train & validation loss
score = model.evaluate(x_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy: {0:.2f}%'.format(score[1]*100))


# Plot train and cross validation error
plot_train_cv_loss(trained_model, epochs)
```

```
Test score: 0.060523610947560 52
Test accuracy: 98.62%
```



Around 18 Epoch train error and validation error meets.

Summary

```
print("Activation function= Relu")
print("Epochs= 50")
print("Batch size= 128")
print("Dropout = 0.5")
print("Batch Normalization")

from prettytable import PrettyTable
x=PrettyTable()
x.field_names = ["Hidden layers","Test accuracy","Optimal_epochs"]
x.add_row(["3 Hidden layers","98.61%","17"])
x.add_row(["5 Hidden layers","98.68%","20"])
x.add_row(["7 Hidden layers","98.70%","22"])
x.add_row(["9 Hidden layers","98.62%","18"])

print(x)
```

```
Activation function= Relu
Epochs= 50
Batch size= 128
Dropout = 0.5
Batch Normalization
+-----------------+---------------+----------------+
|  Hidden layers  | Test accuracy | Optimal_epochs |
+-----------------+---------------+----------------+
| 3 Hidden layers |     98.61%    |       17       |
| 5 Hidden layers |     98.68%    |       20       |
| 7 Hidden layers |     98.70%    |       22       |
| 9 Hidden layers |     98.62%    |       18       |
+-----------------+---------------+----------------+
```