In [5]:

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [7]:

```
(XTrain, YTrain), (XTest, YTest) = mnist.load_data()
print('XTrain class = ',type(XTrain))
print('YTrain class = ',type(YTrain))
# shape of our dataset.
print('XTrain shape = ',XTrain. shape)
print('XTest shape = ',XTest.shape)
print('YTrain shape = ',YTrain.shape)
print('YTest shape = ',YTest.shape)
# Number of distinct values of our MNIST target
print('YTrain values = ', np. unique (YTrain))
print('YTest values = ', np. unique (YTest))
# Distribution of classes in our dataset.
unique, counts = np. unique (YTrain, return_counts=True)
print('YTrain distribution = ',dict (zip(unique, counts)))
unique, counts = np. unique (YTest, return_counts=True)
print('YTest distribution = ',dict(zip(unique, counts)))
```

```
XTrain class =  <class 'numpy.ndarray'>
YTrain class =  <class 'numpy.ndarray'>
XTrain shape =  (60000, 28, 28)
XTest shape =  (10000, 28, 28)
YTrain shape =  (60000,)
YTest shape =  (10000,)
YTrain values =  [0 1 2 3 4 5 6 7 8 9]
YTest values =  [0 1 2 3 4 5 6 7 8 9]
YTrain distribution =  {0: 5923, 1: 6742, 2: 5958, 3: 6131, 4: 5842, 5: 5421, 6: 5
918, 7: 6265, 8: 5851, 9: 5949}
YTest distribution =  {0: 980, 1: 1135, 2: 1032, 3: 1010, 4: 982, 5: 892, 6: 958,
7: 1028, 8: 974, 9: 1009}
```

In [11]:

```
XTrain = XTrain.astype('float32') / 255
XTest = XTest.astype('float32') / 255
# data reshapping.
XTrain = XTrain.reshape((len(XTrain), np. prod (XTrain.shape[1:])))
XTest = XTest.reshape((len(XTest), np.prod(XTest.shape[1:])))
print (XTrain.shape)
print (XTest.shape)
```

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

In [15]:

```
InputModel = Input (shape=(784,))
EncodedLayer = Dense(32, activation='relu')(InputModel)
DecodedLayer = Dense(784, activation='sigmoid') (EncodedLayer)
AutoencoderModel = Model (InputModel, DecodedLayer)
# we can summarize our model.
AutoencoderModel.summary()
```

```
Model: "functional"
```

Layer (type)	Output Shape
input_layer (InputLayer)	(None, 784)
dense (Dense)	(None, 32)
dense_1 (Dense)	(None, 784)

```
Total params: 50,992 (199.19 KB)
Trainable params: 50,992 (199.19 KB)
Non-trainable params: 0 (0.00 B)
```

In [19]:

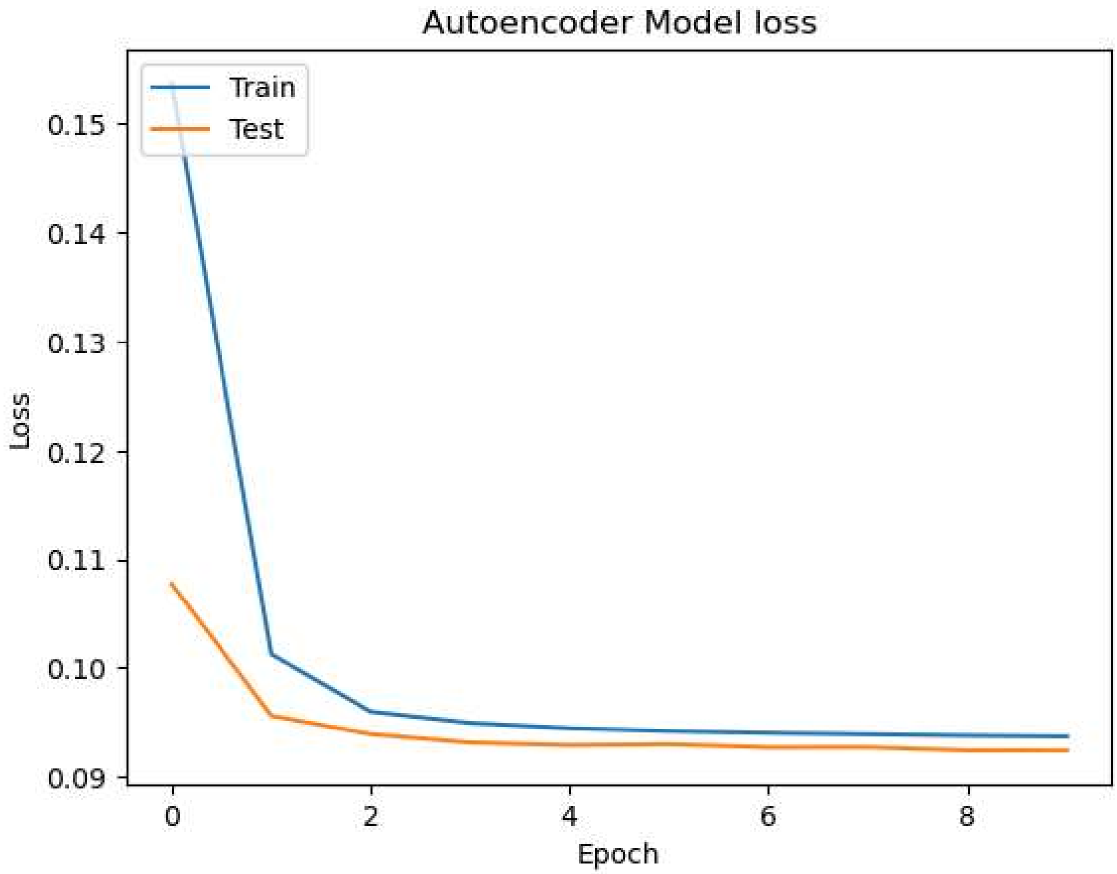
```
AutoencoderModel.compile(optimizer='adam', loss='binary_crossentropy')
history = AutoencoderModel.fit(XTrain, XTrain,
                               batch_size=25,
                               epochs=10,
                               shuffle=True,
                               validation_data= (XTest, XTest))
# Make prediction to decode the digits
DecodedDigits = AutoencoderModel.predict(XTest)
```

```
Epoch 1/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - loss: 0.2118 - val_loss: 0.1077
Epoch 2/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.1043 - val_loss: 0.0956
Epoch 3/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 0.0962 - val_loss: 0.0939
Epoch 4/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 0.0952 - val_loss: 0.0931
Epoch 5/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.0944 - val_loss: 0.0929
Epoch 6/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 0.0944 - val_loss: 0.0930
Epoch 7/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 0.0940 - val_loss: 0.0927
Epoch 8/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - loss: 0.0938 - val_loss: 0.0927
Epoch 9/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - loss: 0.0939 - val_loss: 0.0924
Epoch 10/10
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 0.0937 - val_loss: 0.0924
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [21]:

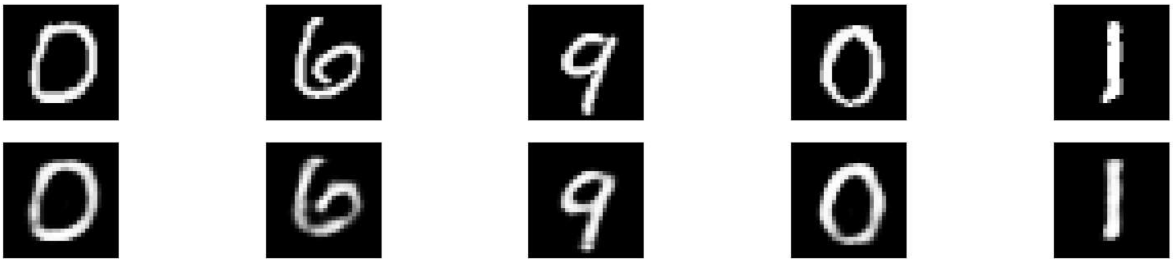
```
def plotmodelhistory(history):
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Autoencoder Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend (['Train', 'Test'], loc='upper left')
    plt.show()
# List all data in history
print (history.history.keys())
#visualization of the Loss minimization during the training process
plotmodelhistory (history)
```

```
dict_keys(['loss', 'val_loss'])
```



In [25]:

```
n=5
plt.figure(figsize=(20, 4))
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    # input image
    plt.imshow(XTest[i+10].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(DecodedDigits[i+10].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



In [ ]: