# ASSIGNMENT – 8

**Name:** Naga Phaneendra Kumara Gupta Mogili

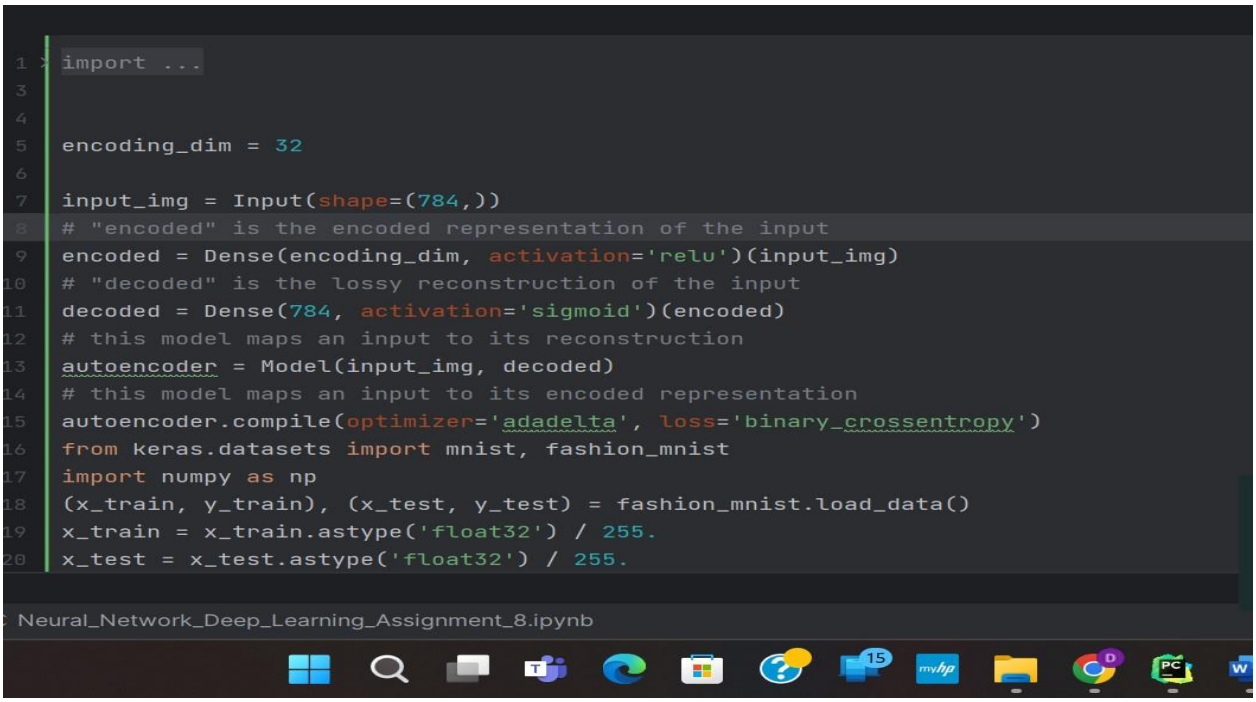
700757977

**GitHub link:**

https://github.com/nagaphaneendra2001/Deep_Learning_Neural_Networks.git

Python program:

1.Add one more hidden layer to autoencoder

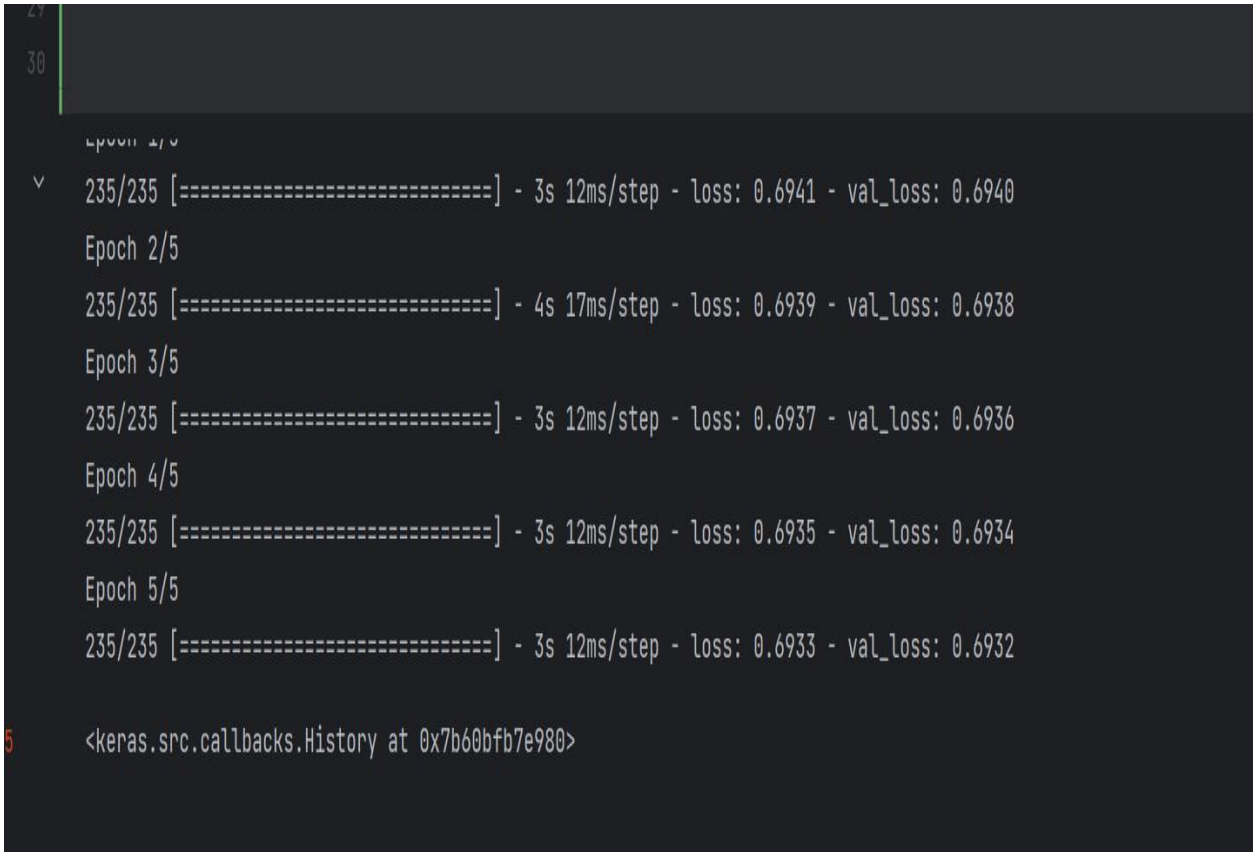


```
import ...


encoding_dim = 32

input_img = Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
# this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
```

Neural_Network_Deep_Learning_Assignment_8.ipynb

```
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

Output:




2. Do the prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

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

encoding_dim = 32

input_img = Input(shape=(784,))

hidden_1 = Dense(256, activation='relu')(input_img)

encoded = Dense(encoding_dim, activation='relu')(hidden_1)

hidden_2 = Dense(256, activation='relu')(encoded)

# Define the output layer
decoded = Dense(784, activation='sigmoid')(hidden_2)

# Define the autoencoder model
```

```python
autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics=['accuracy'])

# Load the fashion MNIST dataset
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

history = autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))

decoded_imgs = autoencoder.predict(x_test)
```

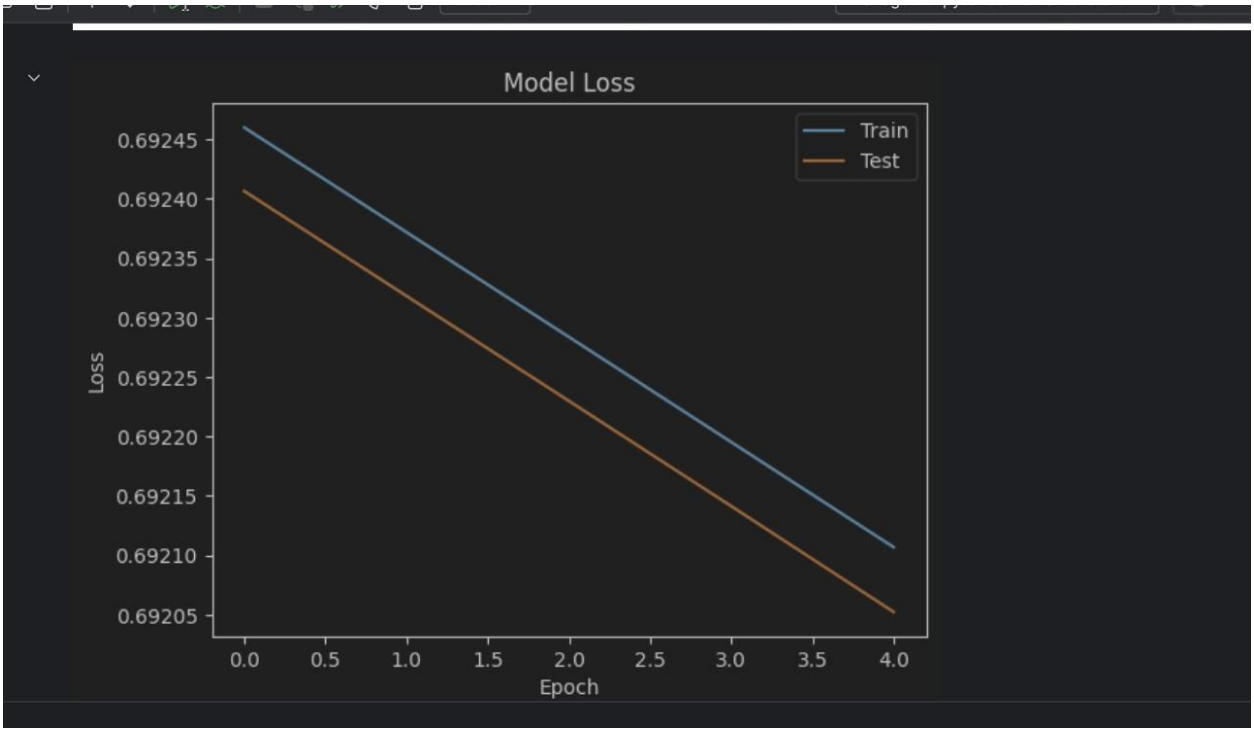
```python
n = 10  # number of images to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original test image
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstructed test image
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

plt.plot(history.history['loss'])
```

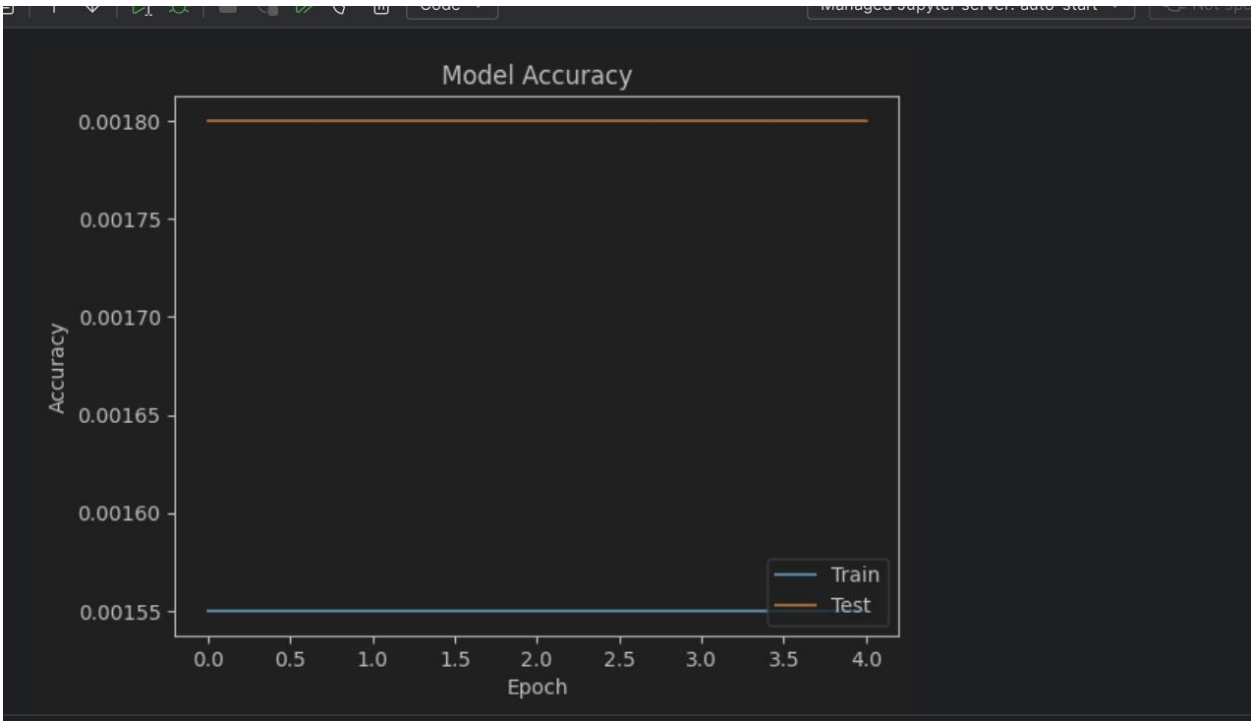
```python
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

```
Epoch 1/5
```

Output:


Epoch 1/5
235/235 [==============================] - 8s 29ms/step - loss: 0.6925 - accuracy: 0.0016 - val_loss: 0.6924 - val_accuracy:
0.0018
Epoch 2/5
235/235 [==============================] - 7s 29ms/step - loss: 0.6924 - accuracy: 0.0016 - val_loss: 0.6923 - val_accuracy:
0.0018
Epoch 3/5
235/235 [==============================] - 7s 29ms/step - loss: 0.6923 - accuracy: 0.0016 - val_loss: 0.6922 - val_accuracy:
0.0018


Model Loss
Train
Test
0.69245
0.69240
0.69235
0.69230
0.69225
0.69220
0.69215
0.69210
0.69205
Loss
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch


Model Accuracy
0.00180
0.00175
0.00170
0.00165
0.00160
0.00155
Accuracy
Train
Test
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

3. Repeat the question 2 on the denoisening autoencoder

```
Add Code Cell | Add Markdown Cell

from keras.layers import Input, Dense
from keras.models import Model

encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

input_img = Input(shape=(784,))
encoded = Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_img, decoded)
# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
from keras.datasets import fashion_mnist
import numpy as np
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
```

```
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))
```

Output:

```
Epoch 6/10
235/235 [==============================] - 4s 17ms/step - loss: 0.6951 - val_loss: 0.6950
Epoch 7/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6949 - val_loss: 0.6948
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6946 - val_loss: 0.6945
Epoch 9/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6943 - val_loss: 0.6942
Epoch 10/10
235/235 [==============================] - 4s 17ms/step - loss: 0.6941 - val_loss: 0.6940

<keras.src.callbacks.History at 0x7b60d1610940>
```

4. plot loss and accuracy using the history object

```python
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

encoding_dim = 32

input_img = Input(shape=(784,))

encoded = Dense(encoding_dim, activation='relu')(input_img)

decoded = Dense(784, activation='sigmoid')(encoded)

autoencoder = Model(input_img, decoded)

# Compile the model
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy',metrics=['accuracy'])
```

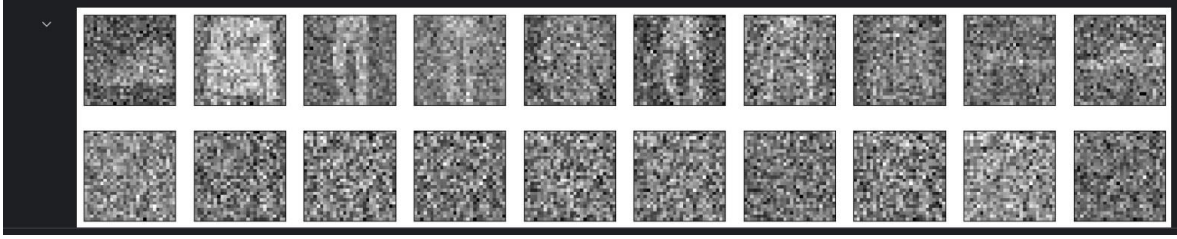
```python
# Load the fashion MNIST dataset
(x_train, _), (x_test, _) = fashion_mnist.load_data()

# Normalize the data and flatten the images
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

noise_factor = 0.5
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)

history = autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test_noisy, x_test_noisy))

decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualize one of the noisy test images
```

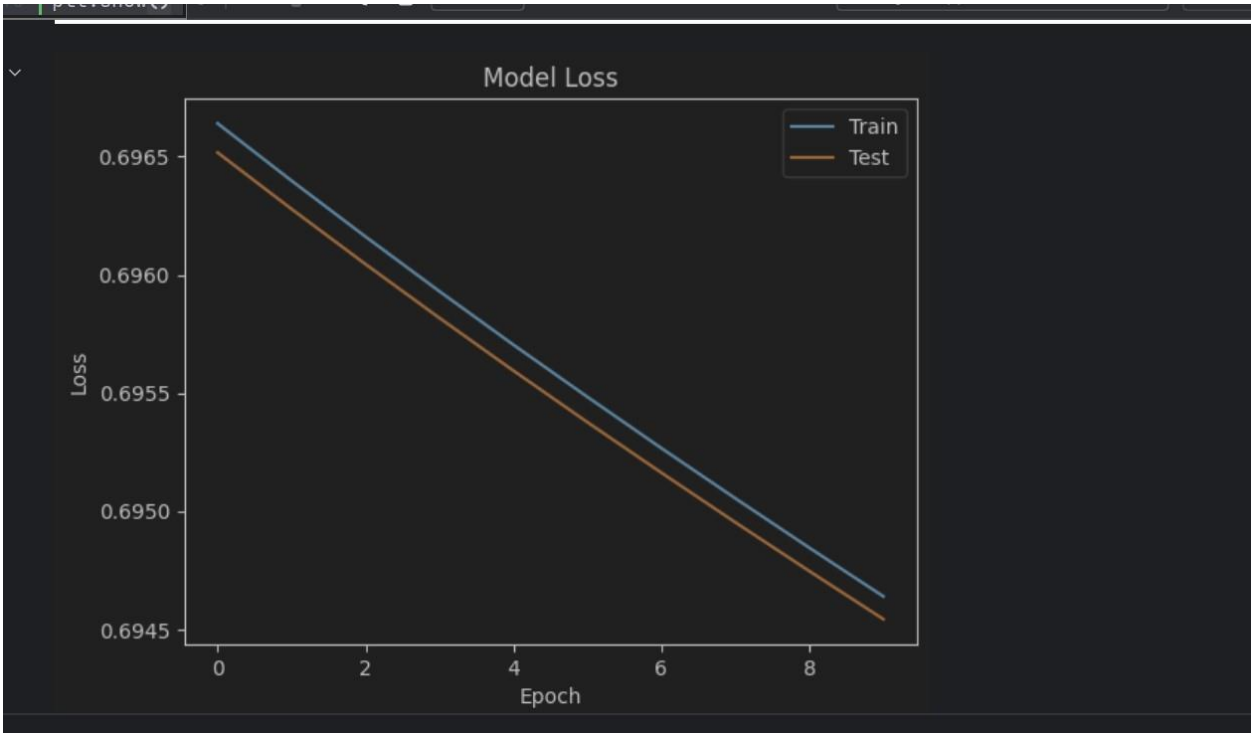
```
plt.figure(figsize=(20, 4))
n = 10
for i in range(n):
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

# Visualize one of the reconstructed test images
for i in range(n):
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
```

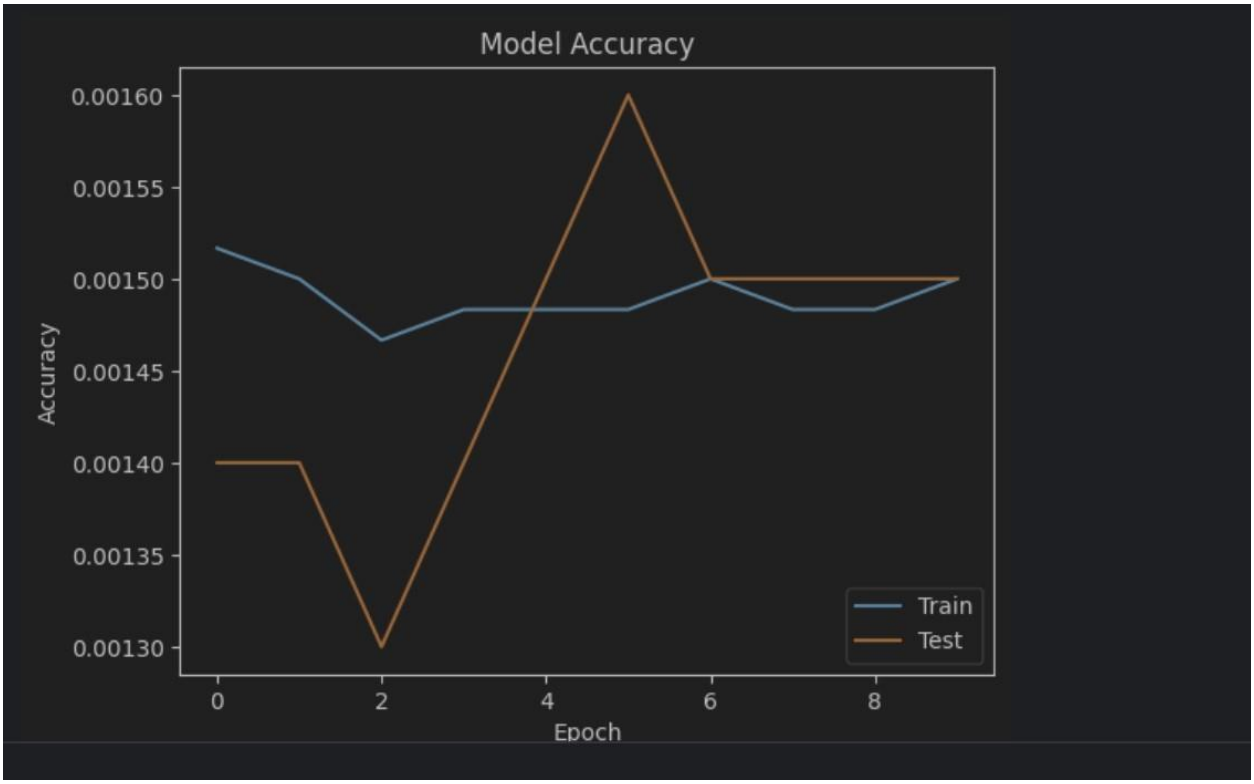
```
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')
plt.show()
```

Output:

```
235/235 [==============================] - 3s 12ms/step - loss: 0.6953 - accuracy: 0.0015 - val_loss: 0.6952 - val_accuracy:
 0.0015
Epoch 8/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6951 - accuracy: 0.0015 - val_loss: 0.6950 - val_accuracy:
 0.0015
Epoch 9/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6948 - accuracy: 0.0015 - val_loss: 0.6947 - val_accuracy:
 0.0015
Epoch 10/10
```






Model Accuracy
0.00160
0.00155
0.00150
0.00145
0.00140
0.00135
0.00130
Accuracy
0
2
4
6
8
Epoch
Train
Test