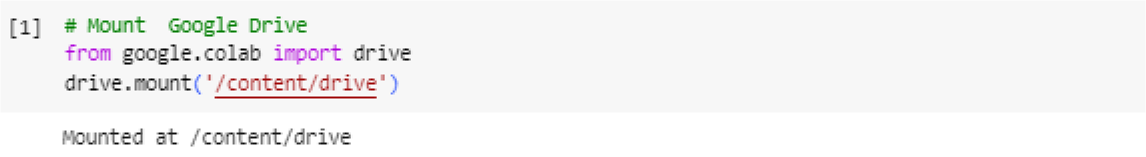
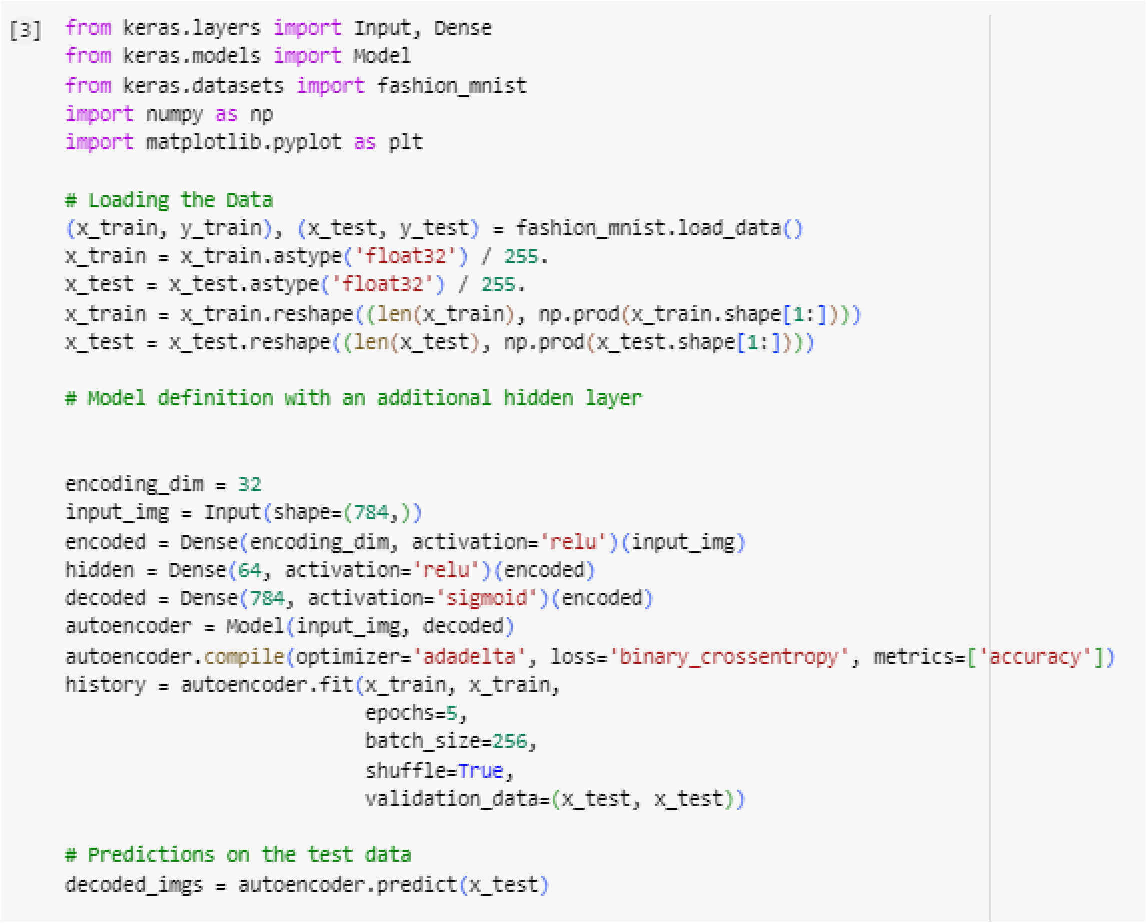
ICP-8

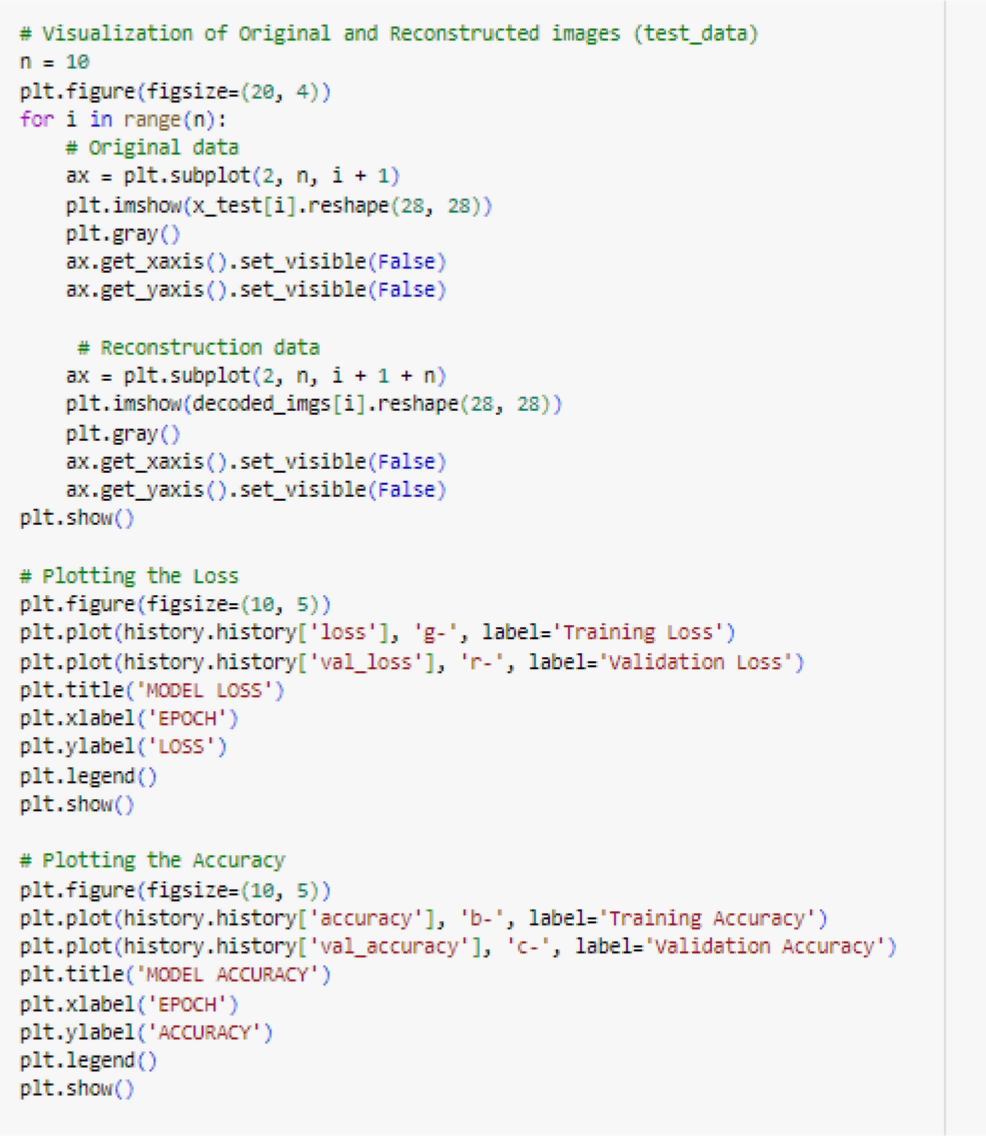
GitHub: https://github.com/Harishwar-reddi/ICP-8
YouTube: https://youtu.be/S9WWXNdJiGo



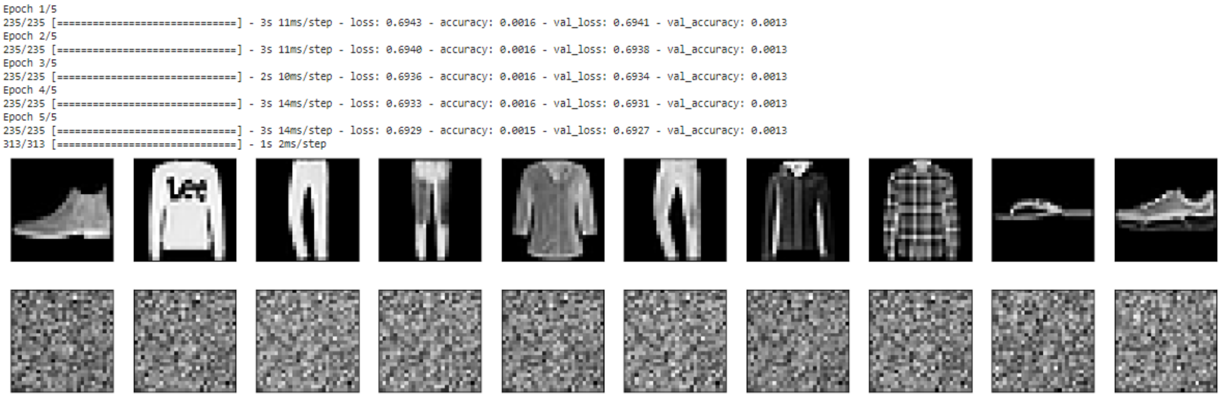
```
[1] # Mount  Google Drive
    from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```



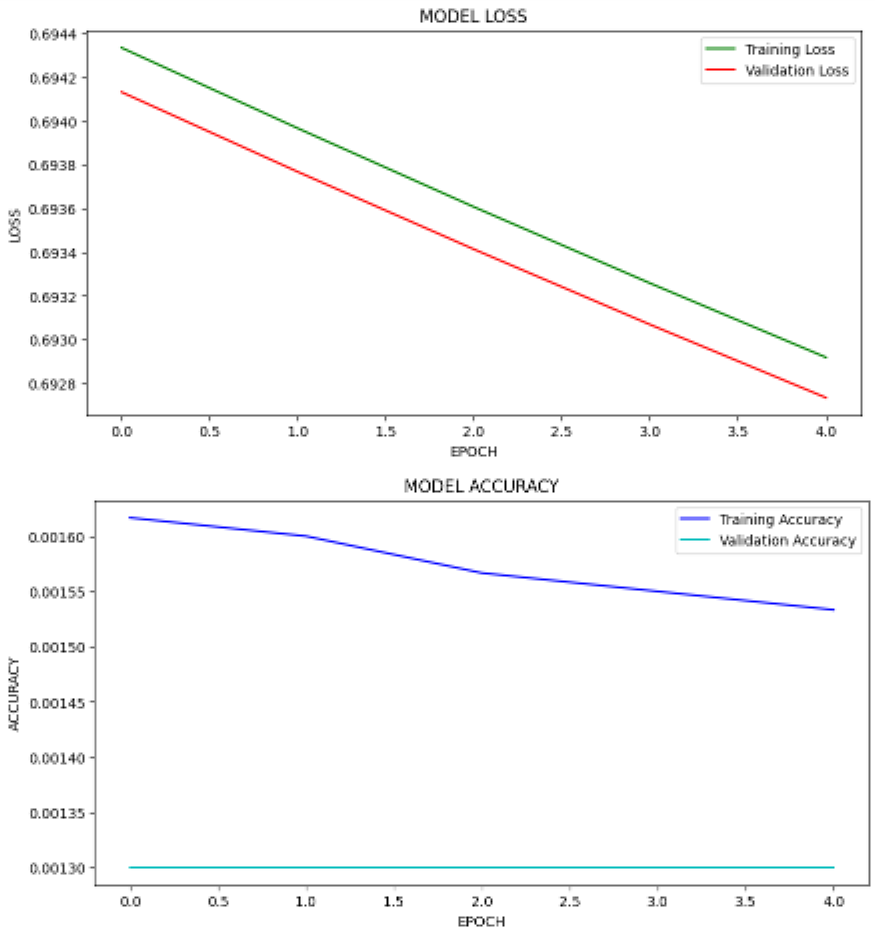
```
[3] from keras.layers import Input, Dense
    from keras.models import Model
    from keras.datasets import fashion_mnist
    import numpy as np
    import matplotlib.pyplot as plt

    # Loading the Data
    (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.
    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:])))

    # Model definition with an additional hidden layer


    encoding_dim = 32
    input_img = Input(shape=(784,))
    encoded = Dense(encoding_dim, activation='relu')(input_img)
    hidden = Dense(64, activation='relu')(encoded)
    decoded = Dense(784, activation='sigmoid')(encoded)
    autoencoder = Model(input_img, decoded)
    autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
    history = autoencoder.fit(x_train, x_train,
                              epochs=5,
                              batch_size=256,
                              shuffle=True,
                              validation_data=(x_test, x_test))

    # Predictions on the test data
    decoded_imgs = autoencoder.predict(x_test)
```

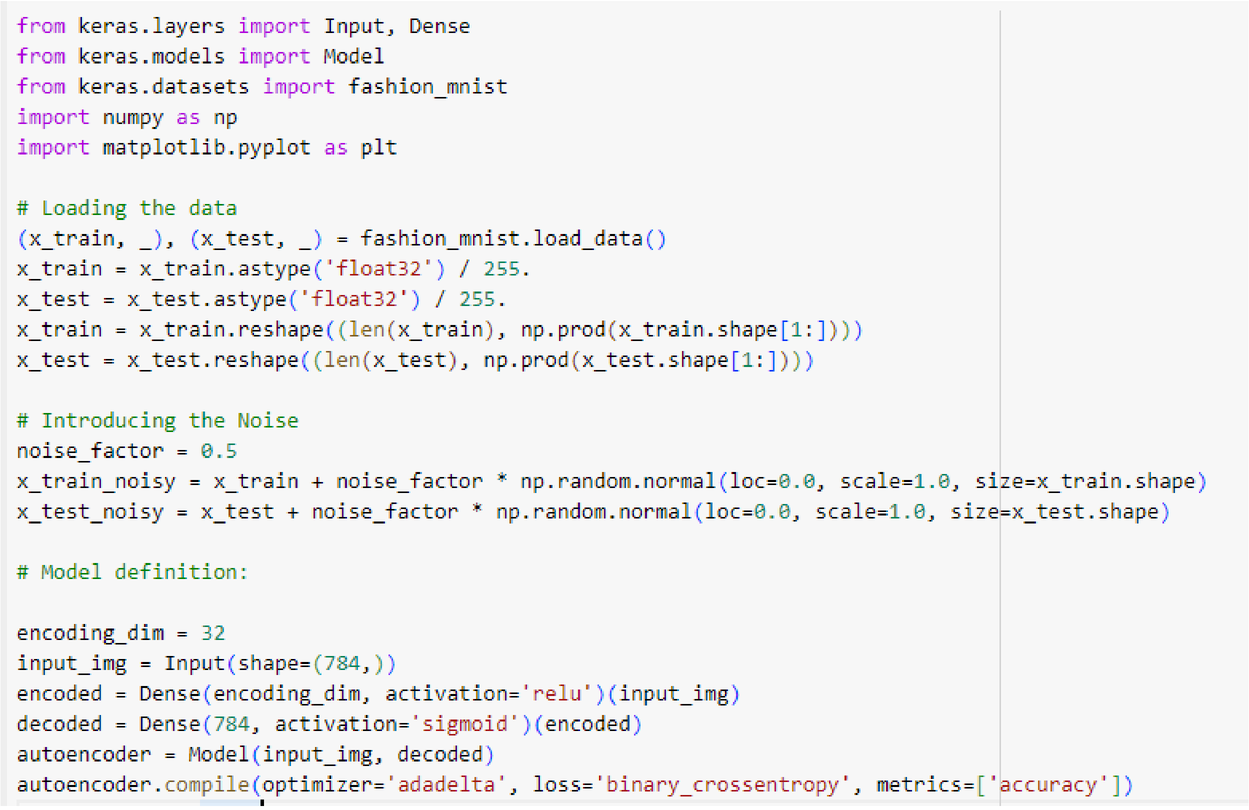


```
# Visualization of Original and Reconstructed images (test_data)
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original data
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

     # Reconstruction data
    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()

# Plotting the Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], 'g-', label='Training Loss')
plt.plot(history.history['val_loss'], 'r-', label='Validation Loss')
plt.title('MODEL LOSS')
plt.xlabel('EPOCH')
plt.ylabel('LOSS')
plt.legend()
plt.show()

# Plotting the Accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], 'b-', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'c-', label='Validation Accuracy')
plt.title('MODEL ACCURACY')
plt.xlabel('EPOCH')
plt.ylabel('ACCURACY')
plt.legend()
plt.show()
```

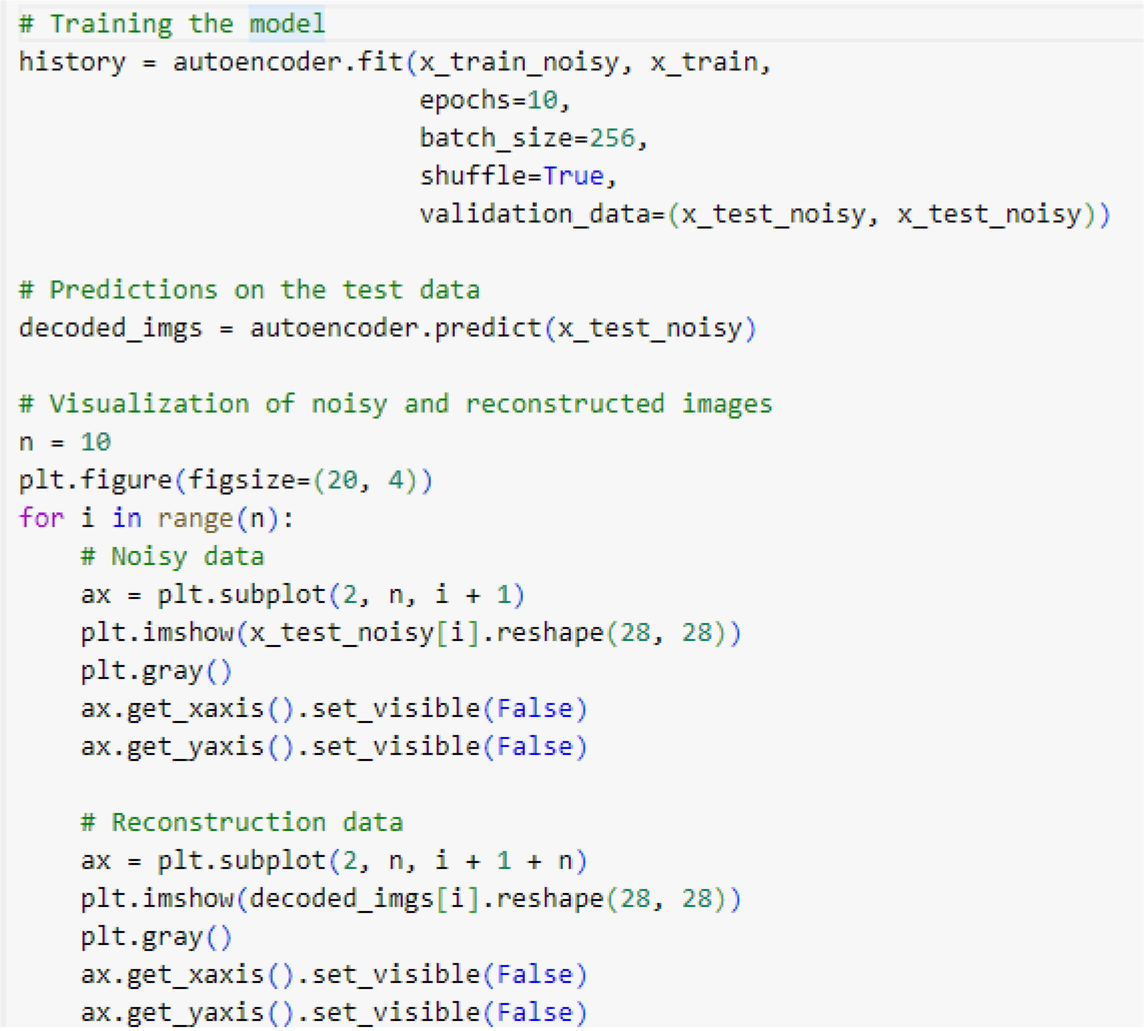


```
Epoch 1/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6943 - accuracy: 0.0016 - val_loss: 0.6941 - val_accuracy: 0.0013
Epoch 2/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6940 - accuracy: 0.0016 - val_loss: 0.6938 - val_accuracy: 0.0013
Epoch 3/5
235/235 [==============================] - 2s 10ms/step - loss: 0.6936 - accuracy: 0.0016 - val_loss: 0.6934 - val_accuracy: 0.0013
Epoch 4/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6933 - accuracy: 0.0016 - val_loss: 0.6931 - val_accuracy: 0.0013
Epoch 5/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6929 - accuracy: 0.0015 - val_loss: 0.6927 - val_accuracy: 0.0013
313/313 [==============================] - 1s 2ms/step
```

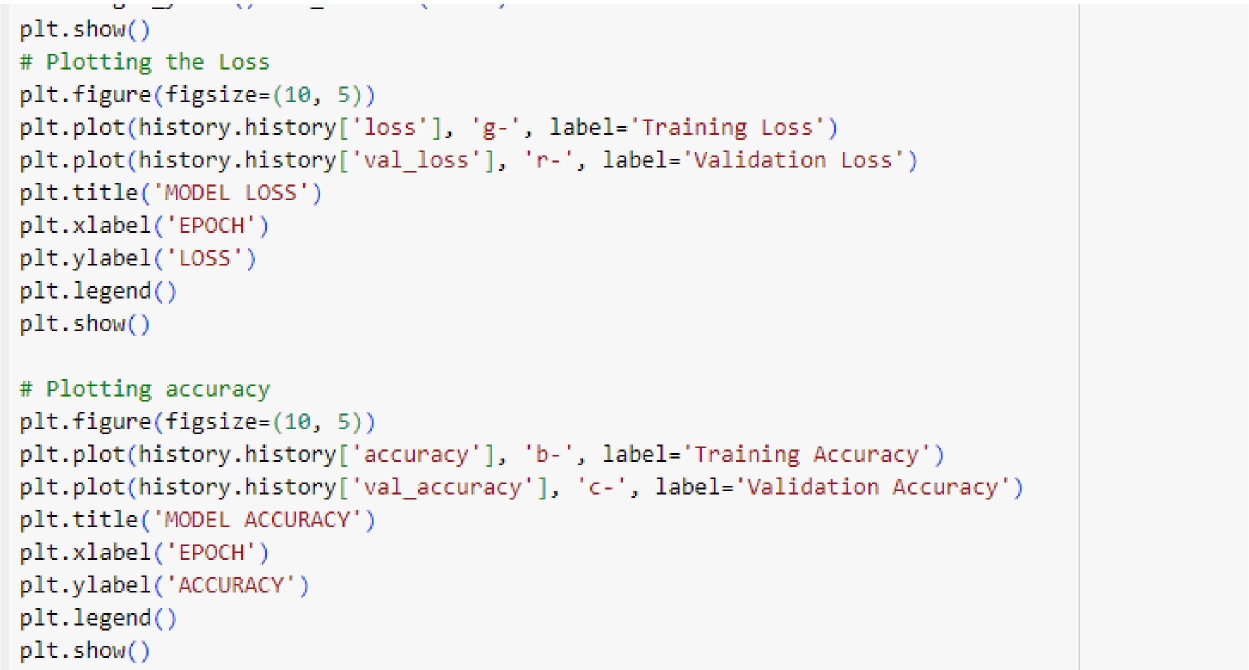

MODEL LOSS
Training Loss
Validation Loss
LOSS
0.6944
0.6942
0.6940
0.6938
0.6936
0.6934
0.6932
0.6930
0.6928
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
EPOCH
MODEL ACCURACY
Training Accuracy
Validation Accuracy
ACCURACY
0.00160
0.00155
0.00150
0.00145
0.00140
0.00135
0.00130
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
EPOCH



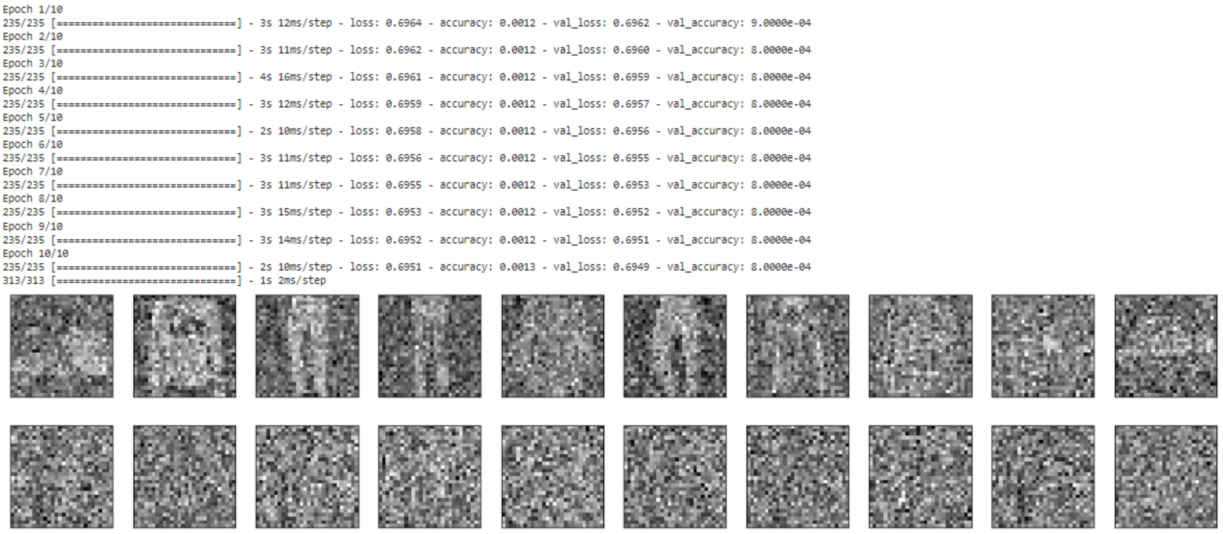
```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import fashion_mnist
import numpy as np
import matplotlib.pyplot as plt

# Loading the data
(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:])))

# Introducing the Noise
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)

# Model definition:

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



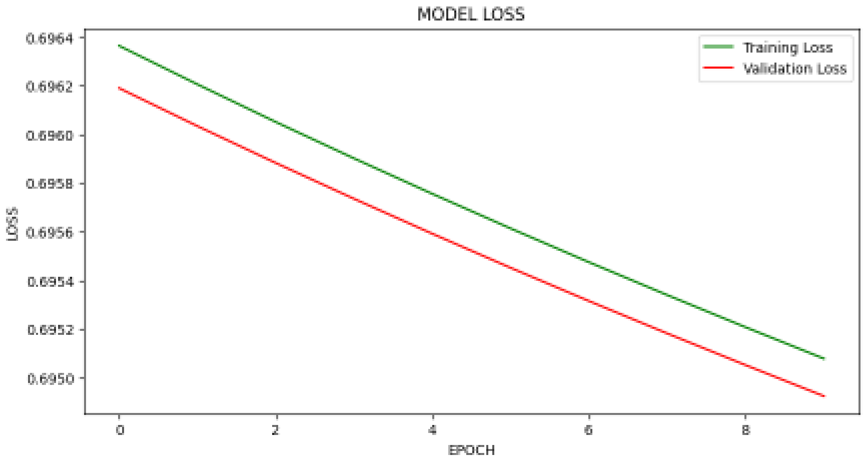
```
# Training the model
history = autoencoder.fit(x_train_noisy, x_train,
                          epochs=10,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test_noisy, x_test_noisy))

# Predictions on the test data
decoded_imgs = autoencoder.predict(x_test_noisy)

# Visualization of noisy and reconstructed images
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Noisy data
    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)

    # Reconstruction data
    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)
```

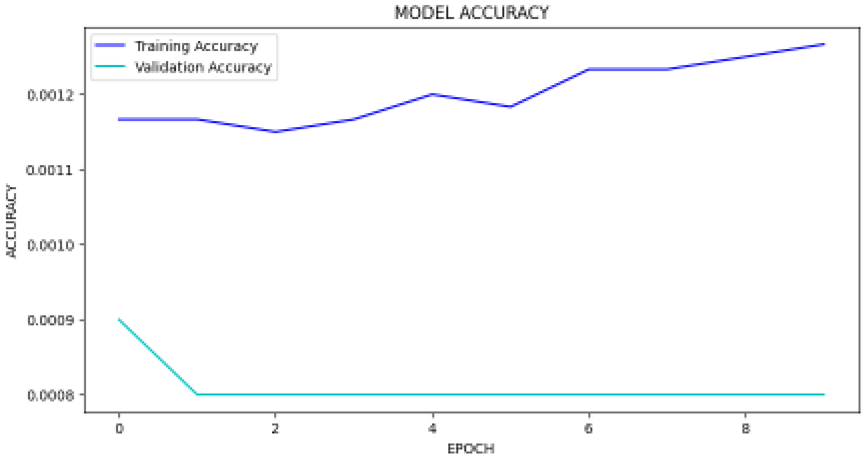


```
plt.show()
# Plotting the Loss
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], 'g-', label='Training Loss')
plt.plot(history.history['val_loss'], 'r-', label='Validation Loss')
plt.title('MODEL LOSS')
plt.xlabel('EPOCH')
plt.ylabel('LOSS')
plt.legend()
plt.show()

# Plotting accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], 'b-', label='Training Accuracy')
plt.plot(history.history['val_accuracy'], 'c-', label='Validation Accuracy')
plt.title('MODEL ACCURACY')
plt.xlabel('EPOCH')
plt.ylabel('ACCURACY')
plt.legend()
plt.show()
```



```
Epoch 1/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6964 - accuracy: 0.0012 - val_loss: 0.6962 - val_accuracy: 9.0000e-04
Epoch 2/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6962 - accuracy: 0.0012 - val_loss: 0.6960 - val_accuracy: 8.0000e-04
Epoch 3/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6961 - accuracy: 0.0012 - val_loss: 0.6959 - val_accuracy: 8.0000e-04
Epoch 4/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6959 - accuracy: 0.0012 - val_loss: 0.6957 - val_accuracy: 8.0000e-04
Epoch 5/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6958 - accuracy: 0.0012 - val_loss: 0.6956 - val_accuracy: 8.0000e-04
Epoch 6/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6956 - accuracy: 0.0012 - val_loss: 0.6955 - val_accuracy: 8.0000e-04
Epoch 7/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6955 - accuracy: 0.0012 - val_loss: 0.6953 - val_accuracy: 8.0000e-04
Epoch 8/10
235/235 [==============================] - 3s 15ms/step - loss: 0.6953 - accuracy: 0.0012 - val_loss: 0.6952 - val_accuracy: 8.0000e-04
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6952 - accuracy: 0.0012 - val_loss: 0.6951 - val_accuracy: 8.0000e-04
Epoch 10/10
235/235 [==============================] - 2s 10ms/step - loss: 0.6951 - accuracy: 0.0013 - val_loss: 0.6949 - val_accuracy: 8.0000e-04
313/313 [==============================] - 1s 2ms/step
```


MODEL LOSS
Training Loss
Validation Loss
LOSS
EPOCH


MODEL ACCURACY
Training Accuracy
Validation Accuracy
ACCURACY
EPOCH