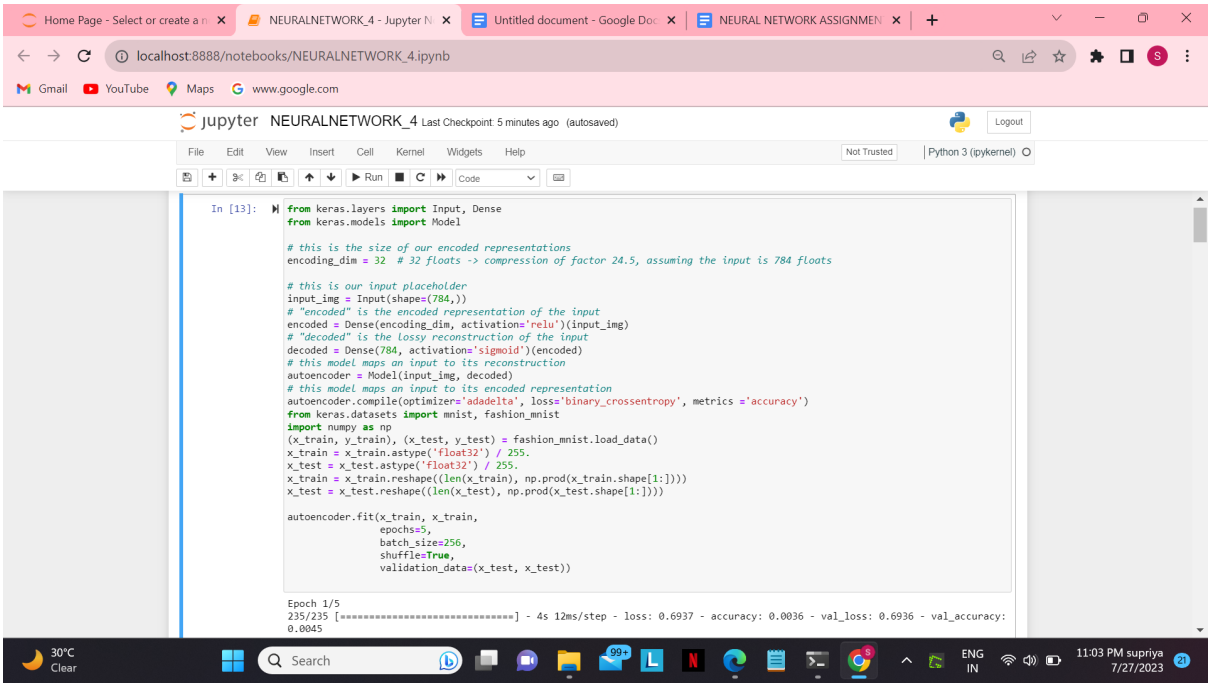
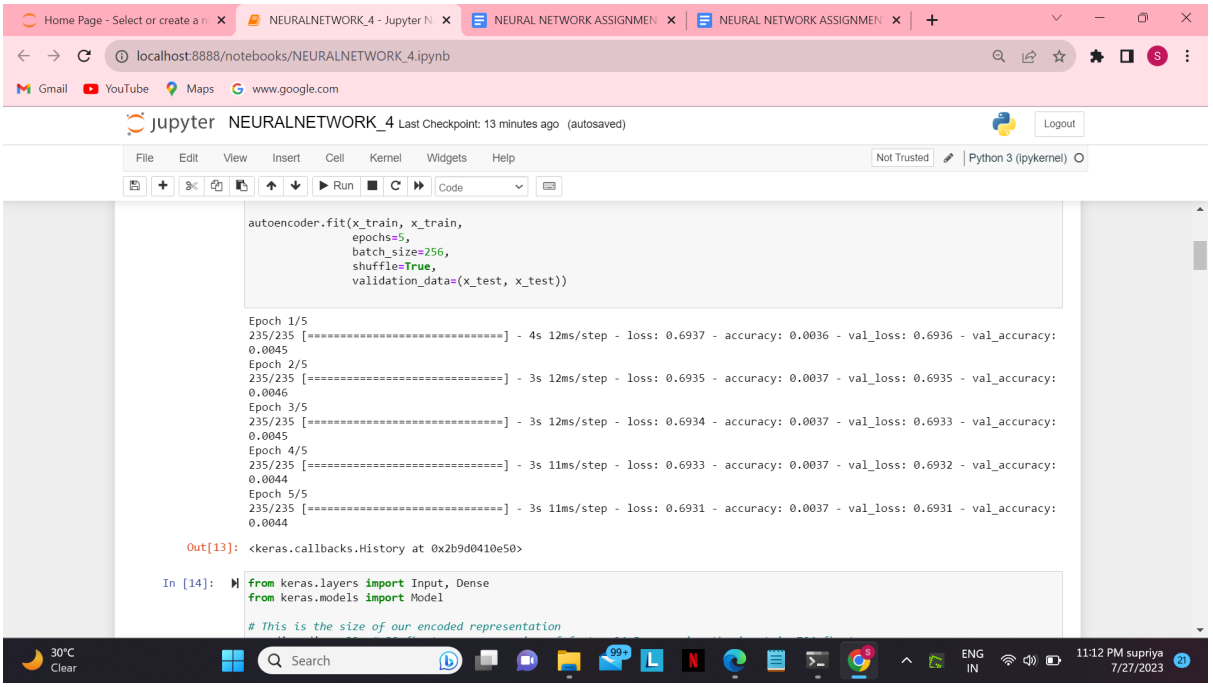
NEURAL NETWORK ASSIGNMENT 4

NAME: SUPRIYA SAMA
700744510

Video Link:
https://drive.google.com/file/d/1qfKP7_oLaT-EA4XuJthFmwE1FCJFlqjn/view?usp=sharing

1. Let's import the Dense function from the keras.





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

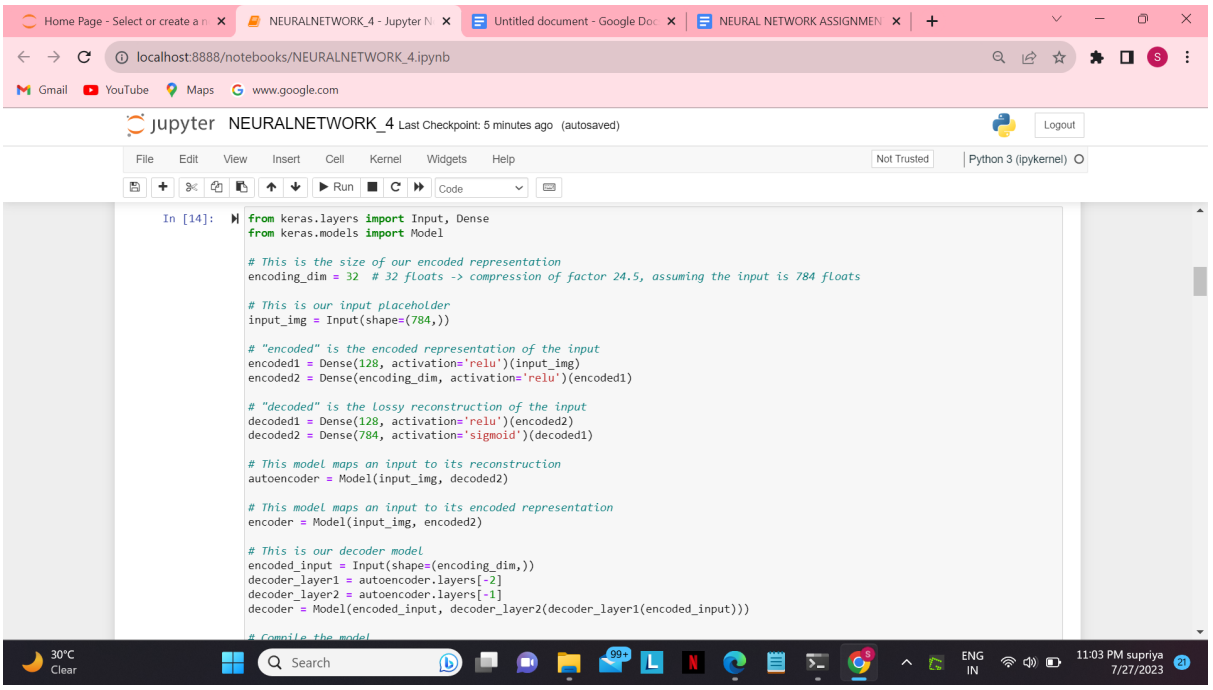
```
Epoch 1/5
235/235 [==============================] - 4s 12ms/step - loss: 0.6937 - accuracy: 0.0036 - val_loss: 0.6936 - val_accuracy:
0.0045
Epoch 2/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 0.0037 - val_loss: 0.6935 - val_accuracy:
0.0046
Epoch 3/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6934 - accuracy: 0.0037 - val_loss: 0.6933 - val_accuracy:
0.0045
Epoch 4/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6933 - accuracy: 0.0037 - val_loss: 0.6932 - val_accuracy:
0.0044
Epoch 5/5
235/235 [==============================] - 3s 11ms/step - loss: 0.6931 - accuracy: 0.0037 - val_loss: 0.6931 - val_accuracy:
0.0044
```

Out[13]: <keras.callbacks.History at 0x2b9d0410e50>

In [14]:

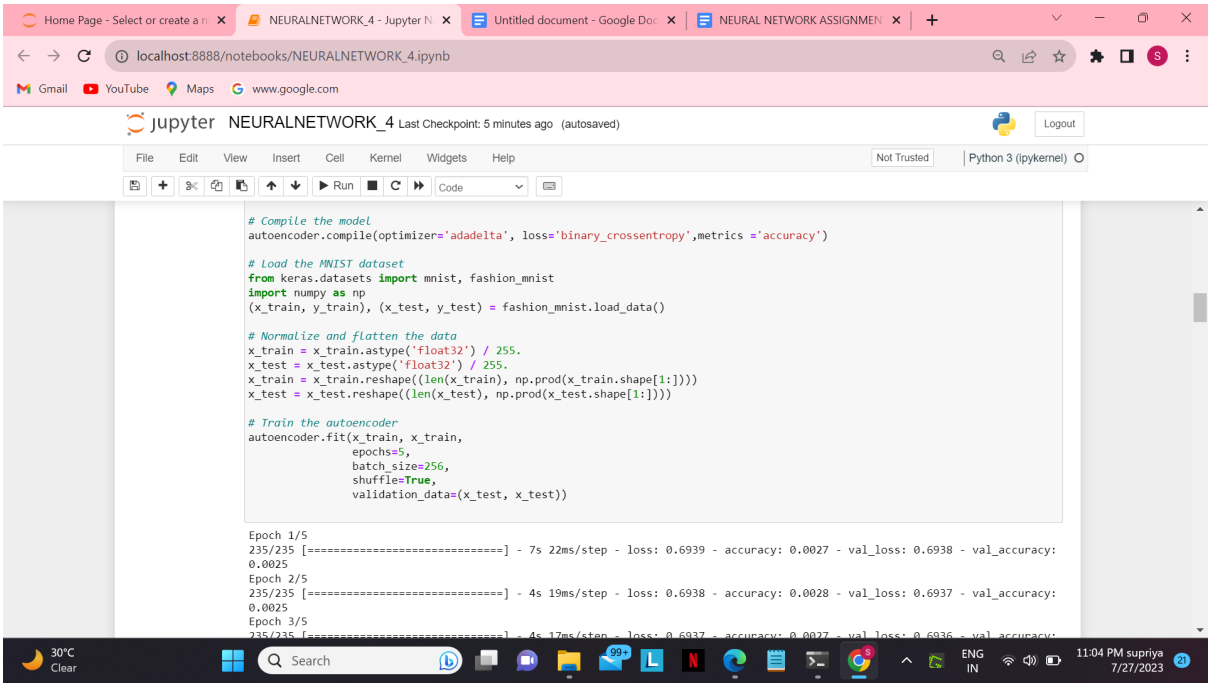
```
from keras.layers import Input, Dense
from keras.models import Model

# This is the size of our encoded representation
```

Here I have added hidden layers to the autoencoder.



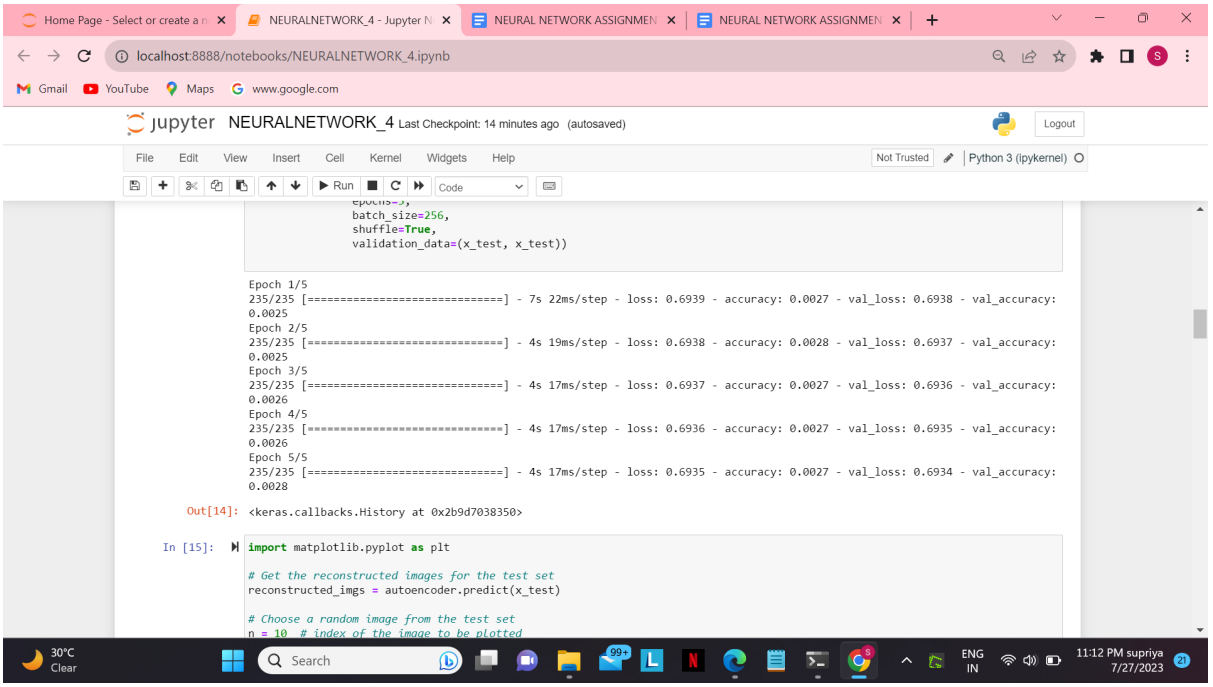
In [14]:

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

# This is the size of our encoded representation
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# This is our input placeholder
input_img = Input(shape=(784,))

# "encoded" is the encoded representation of the input
encoded1 = Dense(128, activation='relu')(input_img)
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

# "decoded" is the lossy reconstruction of the input
decoded1 = Dense(128, activation='relu')(encoded2)
decoded2 = Dense(784, activation='sigmoid')(decoded1)

# This model maps an input to its reconstruction
autoencoder = Model(input_img, decoded2)

# This model maps an input to its encoded representation
encoder = Model(input_img, encoded2)

# This is our decoder model
encoded_input = Input(shape=(encoding_dim,))
decoder_layer1 = autoencoder.layers[-2]
decoder_layer2 = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer2(decoder_layer1(encoded_input)))

# Compile the model
```



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

# Load the MNIST dataset
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize and flatten the 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:])))

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

```
Epoch 1/5
235/235 [==============================] - 7s 22ms/step - loss: 0.6939 - accuracy: 0.0027 - val_loss: 0.6938 - val_accuracy:
0.0025
Epoch 2/5
235/235 [==============================] - 4s 19ms/step - loss: 0.6938 - accuracy: 0.0028 - val_loss: 0.6937 - val_accuracy:
0.0025
Epoch 3/5
```



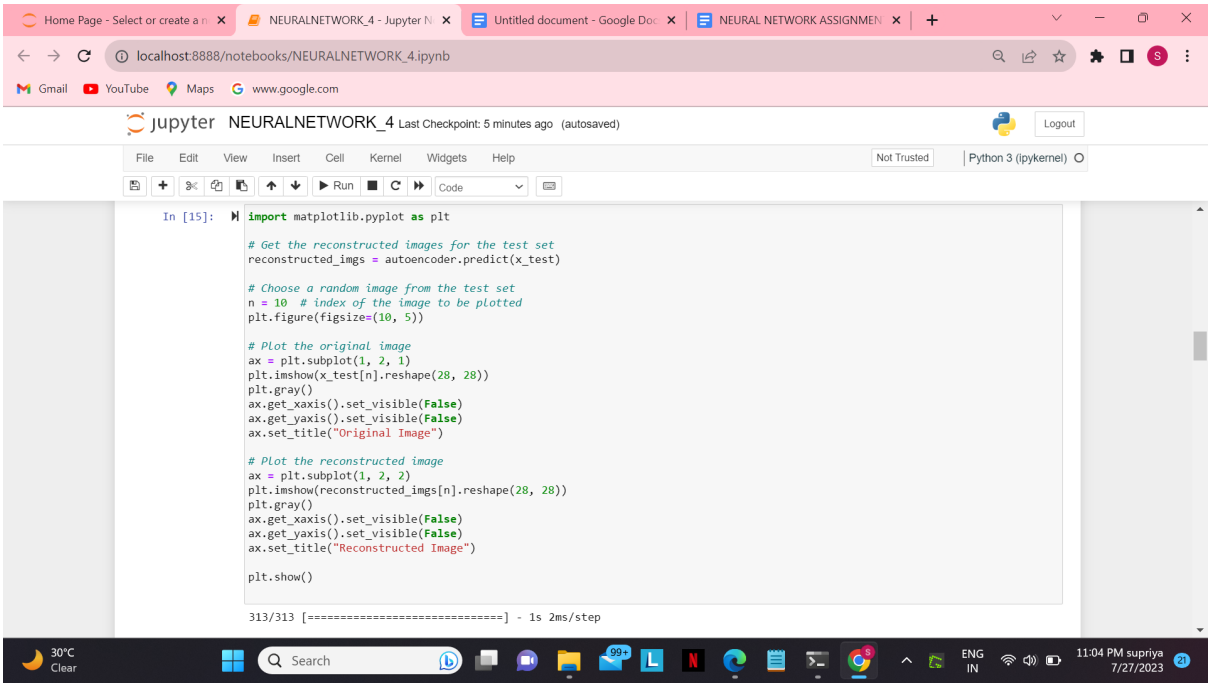
```
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

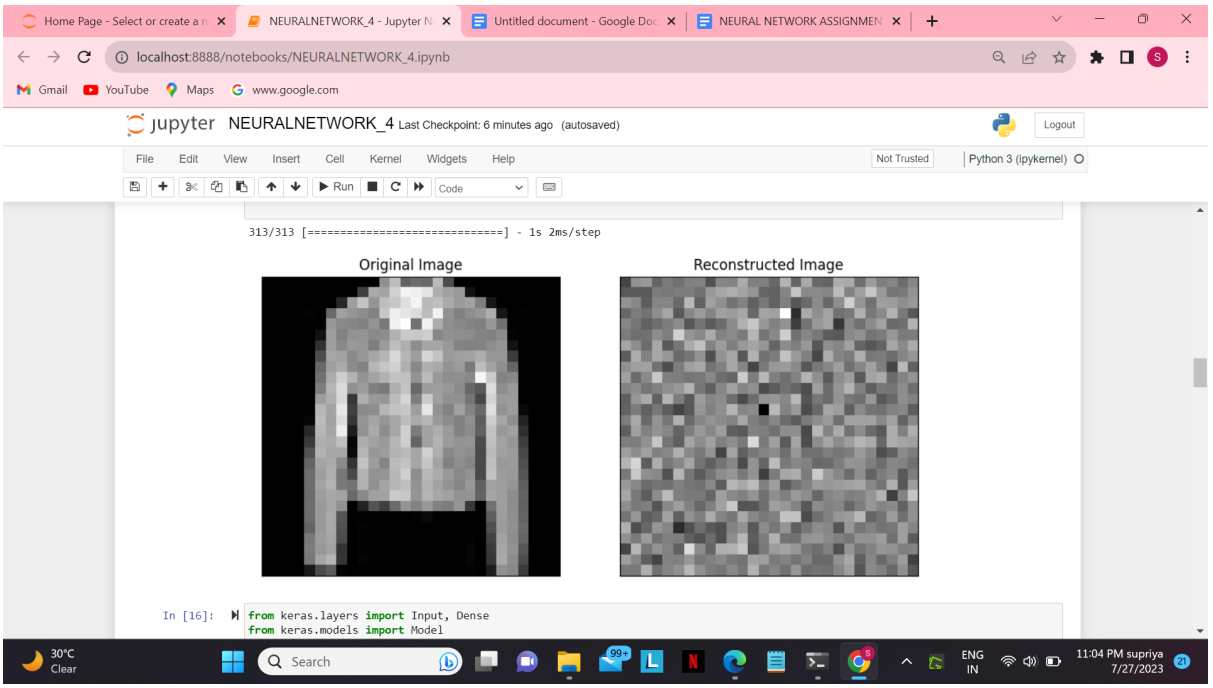
```
Epoch 1/5
235/235 [==============================] - 7s 22ms/step - loss: 0.6939 - accuracy: 0.0027 - val_loss: 0.6938 - val_accuracy:
0.0025
Epoch 2/5
235/235 [==============================] - 4s 19ms/step - loss: 0.6938 - accuracy: 0.0028 - val_loss: 0.6937 - val_accuracy:
0.0025
Epoch 3/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6937 - accuracy: 0.0027 - val_loss: 0.6936 - val_accuracy:
0.0026
Epoch 4/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6936 - accuracy: 0.0027 - val_loss: 0.6935 - val_accuracy:
0.0026
Epoch 5/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6935 - accuracy: 0.0027 - val_loss: 0.6934 - val_accuracy:
0.0028
```

Out[14]: <keras.callbacks.History at 0x2b9d7038350>

In [15]:
```
import matplotlib.pyplot as plt

# Get the reconstructed images for the test set
reconstructed_imgs = autoencoder.predict(x_test)

# Choose a random image from the test set
n = 10  # index of the image to be plotted
```

2. I have imported matplotlib as plt and plotted original and reconstructed images.





We can see the output of original and reconstructed images.

3. Let's repeat the process on the denoising autoencoder.

Home Page - Select or create a n | NEURALNETWORK_4 - Jupyter N | Untitled document - Google Doc | NEURAL NETWORK ASSIGNMEN

localhost:8888/notebooks/NEURALNETWORK_4.ipynb

Gmail YouTube Maps www.google.com

jupyter NEURALNETWORK_4 Last Checkpoint: 6 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

In [16]:

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

# this is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# this is our input placeholder
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',metrics ='accuracy')
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.reshape((len(x_test), np.prod(x_test.shape[1:])))

#introducing 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)

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

30°C Clear | Search | ENG IN | 11:04 PM supriya 7/27/2023

Home Page - Select or create a n | NEURALNETWORK_4 - Jupyter N | NEURAL NETWORK ASSIGNMEN | NEURAL NETWORK ASSIGNMEN

localhost:8888/notebooks/NEURALNETWORK_4.ipynb

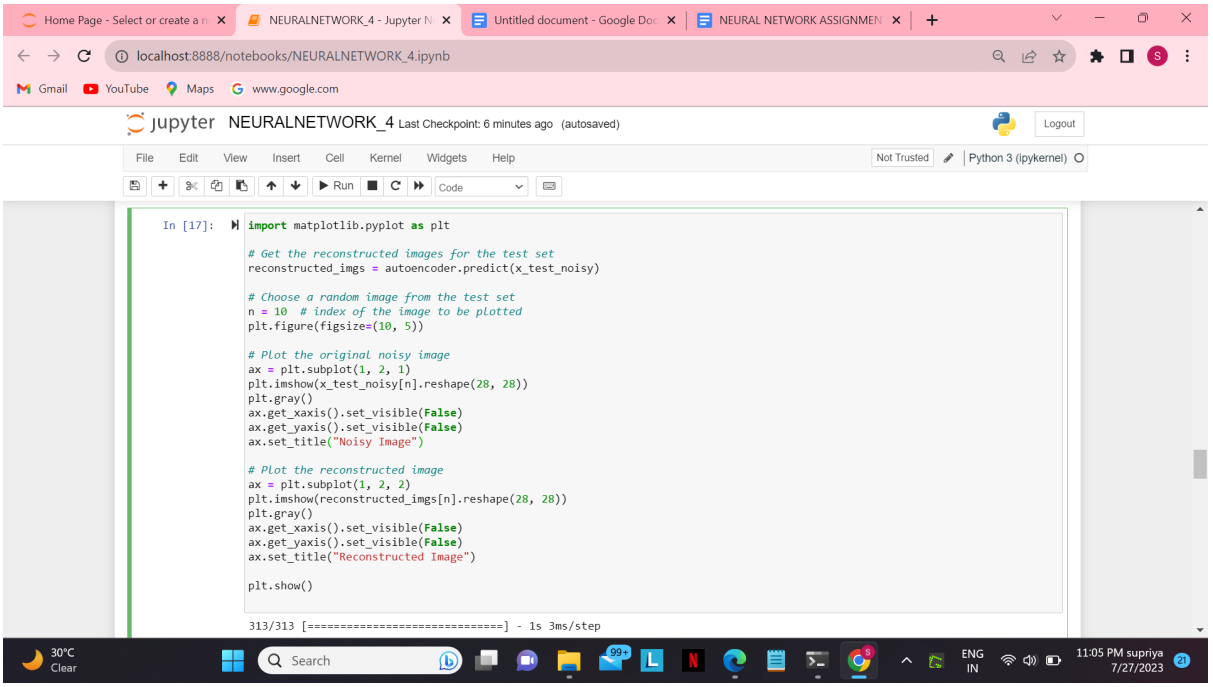
Gmail YouTube Maps www.google.com

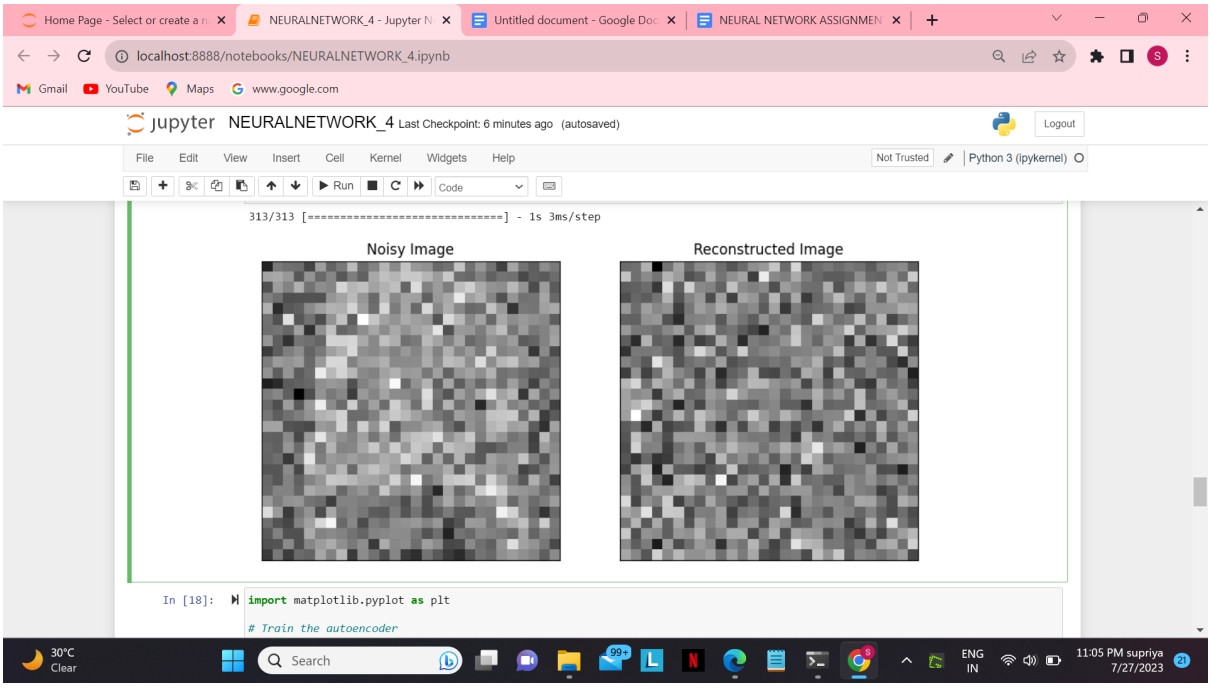
jupyter NEURALNETWORK_4 Last Checkpoint: 14 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

```
Epoch 1/10
235/235 [==============================] - 4s 14ms/step - loss: 0.6964 - accuracy: 7.8333e-04 - val_loss: 0.6963 - val_accur
acy: 8.0000e-04
Epoch 2/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6962 - accuracy: 8.0000e-04 - val_loss: 0.6961 - val_accur
acy: 8.0000e-04
Epoch 3/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6959 - accuracy: 8.1667e-04 - val_loss: 0.6959 - val_accur
acy: 8.0000e-04
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6957 - accuracy: 8.6667e-04 - val_loss: 0.6956 - val_accur
acy: 7.0000e-04
Epoch 5/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6955 - accuracy: 8.6667e-04 - val_loss: 0.6954 - val_accur
acy: 7.0000e-04
Epoch 6/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6952 - accuracy: 8.6667e-04 - val_loss: 0.6952 - val_accur
acy: 8.0000e-04
Epoch 7/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6950 - accuracy: 9.0000e-04 - val_loss: 0.6950 - val_accur
acy: 9.0000e-04
Epoch 8/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6948 - accuracy: 8.6667e-04 - val_loss: 0.6948 - val_accur
acy: 0.0011
Epoch 9/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6946 - accuracy: 8.6667e-04 - val_loss: 0.6946 - val_accur
acy: 0.0013
Epoch 10/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6944 - accuracy: 8.3333e-04 - val_loss: 0.6944 - val_accur
acy: 0.0013
```

30°C Clear | Search | ENG IN | 11:12 PM supriya 7/27/2023





We can see the output of noisy and reconstructed images.

4. Let's train the autoencoder and plot the loss and accuracy.

Home Page - Select or create a n | NEURALNETWORK_4 - Jupyter N | Untitled document - Google Doc | NEURAL NETWORK ASSIGNMEN

localhost:8888/notebooks/NEURALNETWORK_4.ipynb

Gmail YouTube Maps www.google.com

jupyter NEURALNETWORK_4 Last Checkpoint: 6 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

In [18]:

```
import matplotlib.pyplot as plt

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

# Plot the loss
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Plot the accuracy
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6942 - accuracy: 8.5000e-04 - val_loss: 0.6942 - val_accur
```

30°C Clear | Search | ENG IN | 11:05 PM supriya 7/27/2023

Home Page - Select or create a n | NEURALNETWORK_4 - Jupyter N | NEURAL NETWORK ASSIGNMEN | NEURAL NETWORK ASSIGNMEN

localhost:8888/notebooks/NEURALNETWORK_4.ipynb

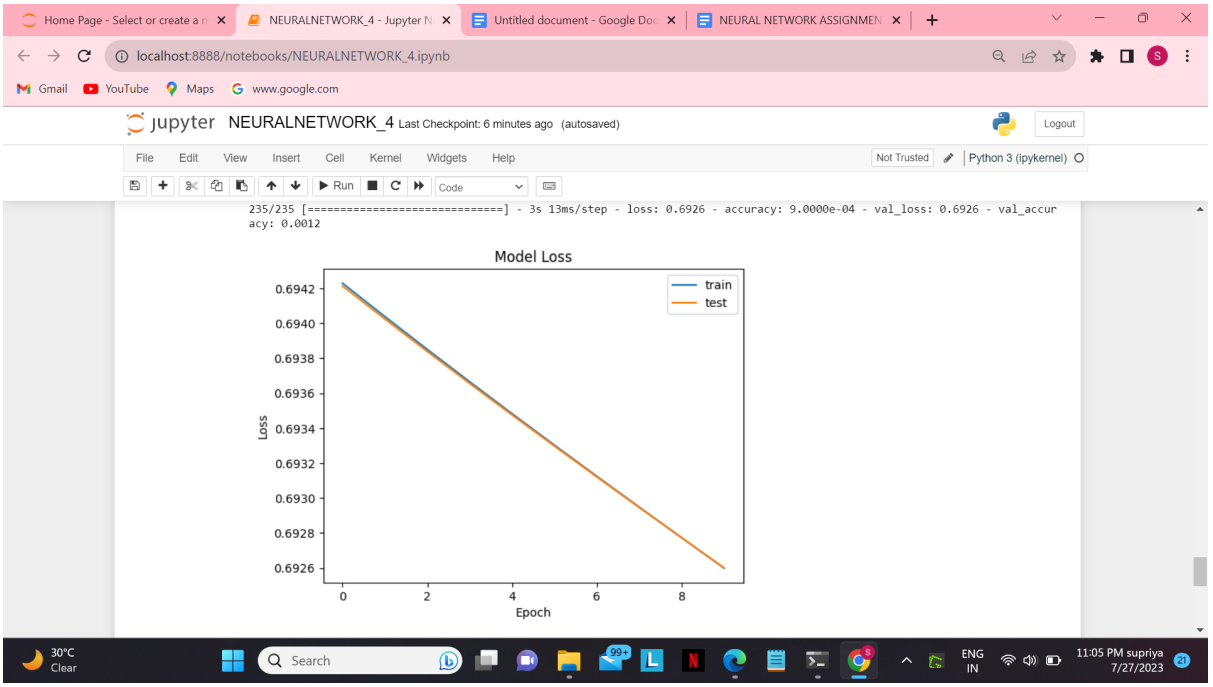
Gmail YouTube Maps www.google.com

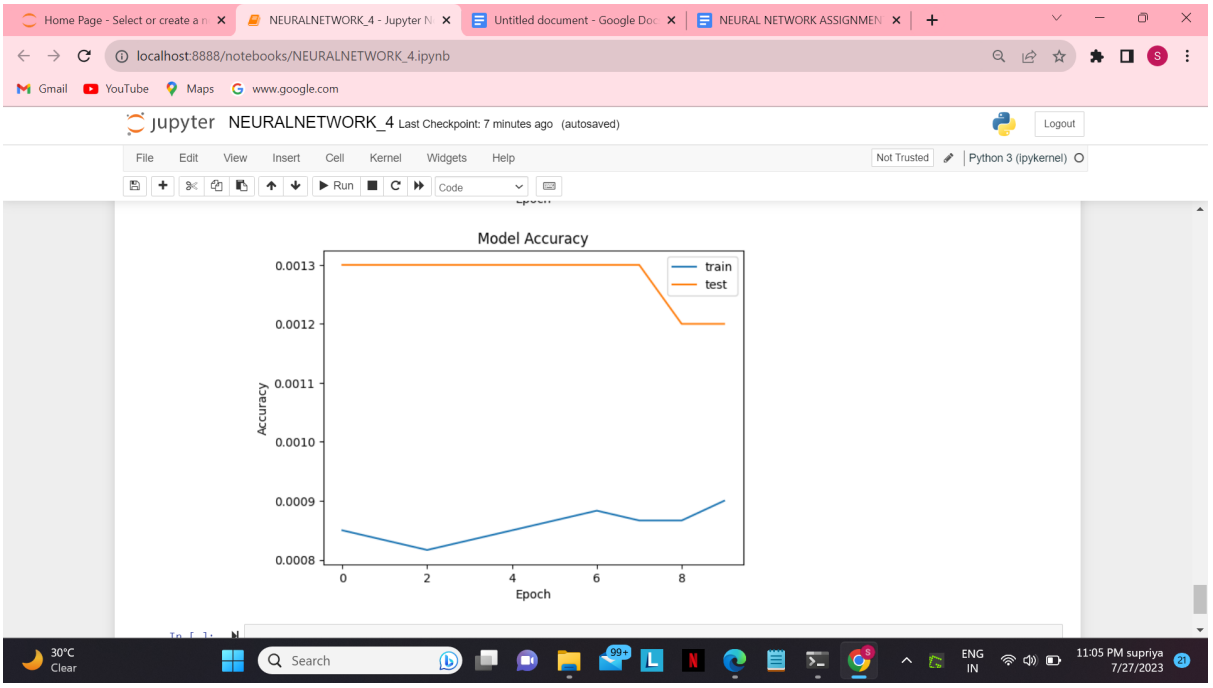
jupyter NEURALNETWORK_4 Last Checkpoint: 14 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Not Trusted | Python 3 (ipykernel)

```
Epoch 1/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6942 - accuracy: 8.5000e-04 - val_loss: 0.6942 - val_accur
acy: 0.0013
Epoch 2/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6940 - accuracy: 8.3333e-04 - val_loss: 0.6940 - val_accur
acy: 0.0013
Epoch 3/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6939 - accuracy: 8.1667e-04 - val_loss: 0.6938 - val_accur
acy: 0.0013
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6937 - accuracy: 8.3333e-04 - val_loss: 0.6937 - val_accur
acy: 0.0013
Epoch 5/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6935 - accuracy: 8.5000e-04 - val_loss: 0.6935 - val_accur
acy: 0.0013
Epoch 6/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6933 - accuracy: 8.6667e-04 - val_loss: 0.6933 - val_accur
acy: 0.0013
Epoch 7/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6931 - accuracy: 8.8333e-04 - val_loss: 0.6931 - val_accur
acy: 0.0013
Epoch 8/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6929 - accuracy: 8.6667e-04 - val_loss: 0.6929 - val_accur
acy: 0.0013
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6928 - accuracy: 8.6667e-04 - val_loss: 0.6928 - val_accur
acy: 0.0012
Epoch 10/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6926 - accuracy: 9.0000e-04 - val_loss: 0.6926 - val_accur
acy: 0.0012
```

30°C Clear | Search | ENG IN | 11:12 PM supriya 7/27/2023





We can see the output of both loss and accuracy.