

# NEURAL NETWORK & DEEP LEARNING

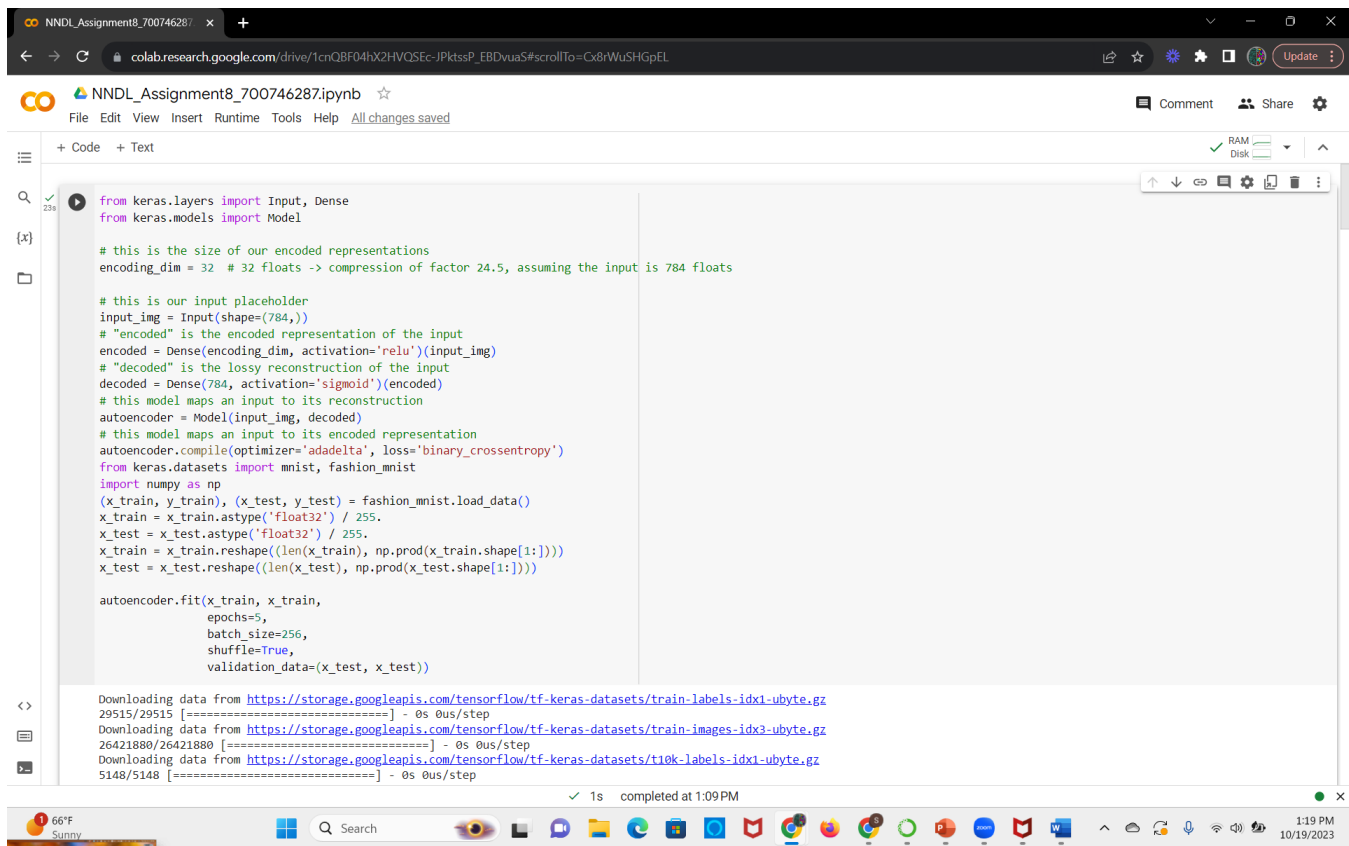
## ASSIGNMENT 8

Name : SAI SNUSHA NAKKA

Student ID : 700746287

Git hub Link: [https://github.com/NSnusha/NNDL\\_Assignment8](https://github.com/NSnusha/NNDL_Assignment8)

Video link: [https://drive.google.com/file/d/1E-dWl3fXiBBdrr8L7x9bWuABILK52GVg/view?usp=share\\_link](https://drive.google.com/file/d/1E-dWl3fXiBBdrr8L7x9bWuABILK52GVg/view?usp=share_link)



```
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')
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.
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))
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>  
29515/29515 [=====] - 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>  
26421880/26421880 [=====] - 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz>  
5148/5148 [=====] - 0s 0us/step

✓ 1s completed at 1:09 PM

```
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.
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))
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz> 29515/29515 [=====] - 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz> 26421880/26421880 [=====] - 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz> 5148/5148 [=====] - 0s 0us/step  
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz> 4422102/4422102 [=====] - 0s 0us/step  
Epoch 1/5  
235/235 [=====] - 6s 18ms/step - loss: 0.6952 - val\_loss: 0.6950  
Epoch 2/5  
235/235 [=====] - 3s 13ms/step - loss: 0.6949 - val\_loss: 0.6948  
Epoch 3/5  
235/235 [=====] - 2s 10ms/step - loss: 0.6947 - val\_loss: 0.6945  
Epoch 4/5  
235/235 [=====] - 2s 10ms/step - loss: 0.6944 - val\_loss: 0.6943  
Epoch 5/5  
235/235 [=====] - 2s 9ms/step - loss: 0.6942 - val\_loss: 0.6940  
<keras.src.callbacks.History at 0x7893502363e0>

```
[2] from keras.layers import Input, Dense
from keras.models import Model
```

1s completed at 1:09 PM

Here we added a new hidden layer to the encoder and the decoder

colab.research.google.com

colab.research.google.com/drive/1cnQB04hX2HVQSEc-JPktssP\_EBDvuaS#scrollTo=Cx8rWuSHGpEL

Update

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

RAM Disk

+ Code + Text

1m

[2]

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

# Define input shape
input_shape = (784,)

# Define encoding dimensions
encoding_dim1 = 64
encoding_dim2 = 32

# Define input layer
input_img = Input(shape=input_shape)

encoded1 = Dense(encoding_dim1, activation='relu')(input_img)
encoded2 = Dense(encoding_dim2, activation='relu')(encoded1)
decoded1 = Dense(encoding_dim1, activation='relu')(encoded2)
decoded2 = Dense(input_shape[0], activation='sigmoid')(decoded1)
autoencoder = Model(input_img, decoded2)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])
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.
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 model
history = autoencoder.fit(x_train, x_train,
                        epochs=20,
                        batch_size=256,
                        shuffle=True,
                        validation_data=(x_test, x_test))

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

1s

completed at 1:09 PM

colab.research.google.com

colab.research.google.com/drive/1cnQB04hX2HVQSEc-JPktssP\_EBDvuaS#scrollTo=Cx8rWuSHGpEL

Update

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

RAM Disk

+ Code + Text

1m

[2]

```
# Train model
history = autoencoder.fit(x_train, x_train,
                        epochs=20,
                        batch_size=256,
                        shuffle=True,
                        validation_data=(x_test, x_test))

# Predict on test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize reconstructed image and original image
import matplotlib.pyplot as plt
# Choose an index of a test image to visualize
idx = 10

# Reshape the test image
test_img = x_test[idx].reshape(28, 28)

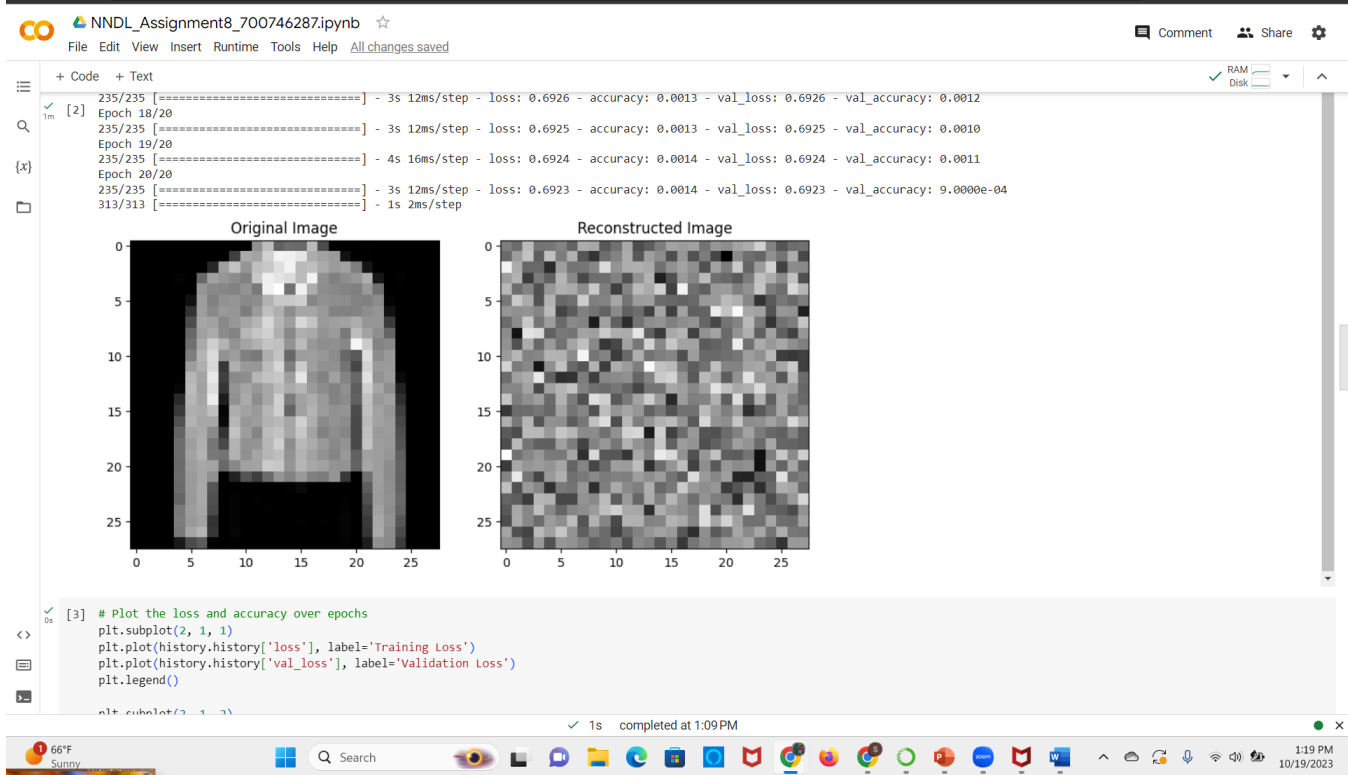
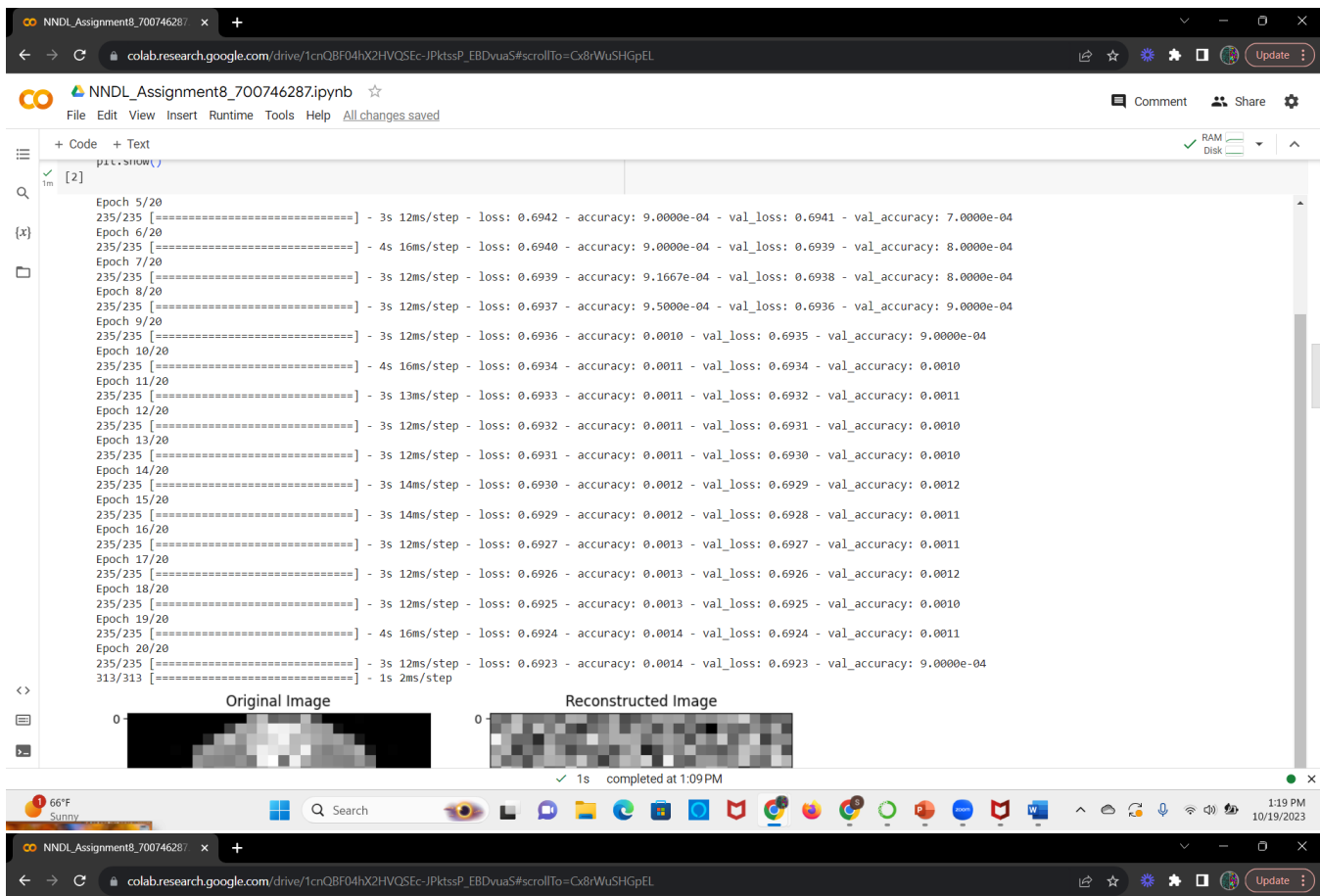
# Reshape the reconstructed image
reconstructed_img = decoded_imgs[idx].reshape(28, 28)

# Plot the original and reconstructed images side by side
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(test_img, cmap='gray')
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_img, cmap='gray')
plt.title('Reconstructed Image')
plt.show()

Epoch 5/20
235/235 [=====] - 3s 12ms/step - loss: 0.6942 - accuracy: 9.0000e-04 - val_loss: 0.6941 - val_accuracy: 7.0000e-04
Epoch 6/20
235/235 [=====] - 4s 16ms/step - loss: 0.6940 - accuracy: 9.0000e-04 - val_loss: 0.6939 - val_accuracy: 8.0000e-04
Epoch 7/20
235/235 [=====] - 3s 12ms/step - loss: 0.6939 - accuracy: 9.1667e-04 - val_loss: 0.6938 - val_accuracy: 8.0000e-04
Epoch 8/20
```

1s

completed at 1:09 PM



We are calculating the loss and accuracy of the model using the history object where in the compile metrics are set to accuracy

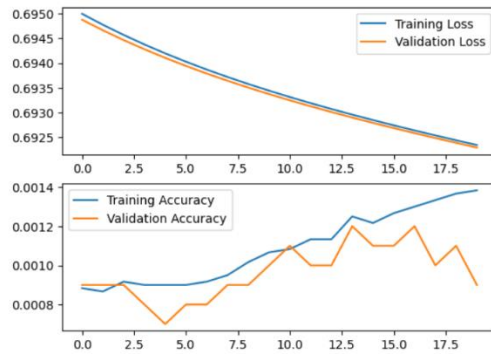
+ Code + Text

RAM  
Disk

```
[3] # Plot the loss and accuracy over epochs
plt.subplot(2, 1, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()

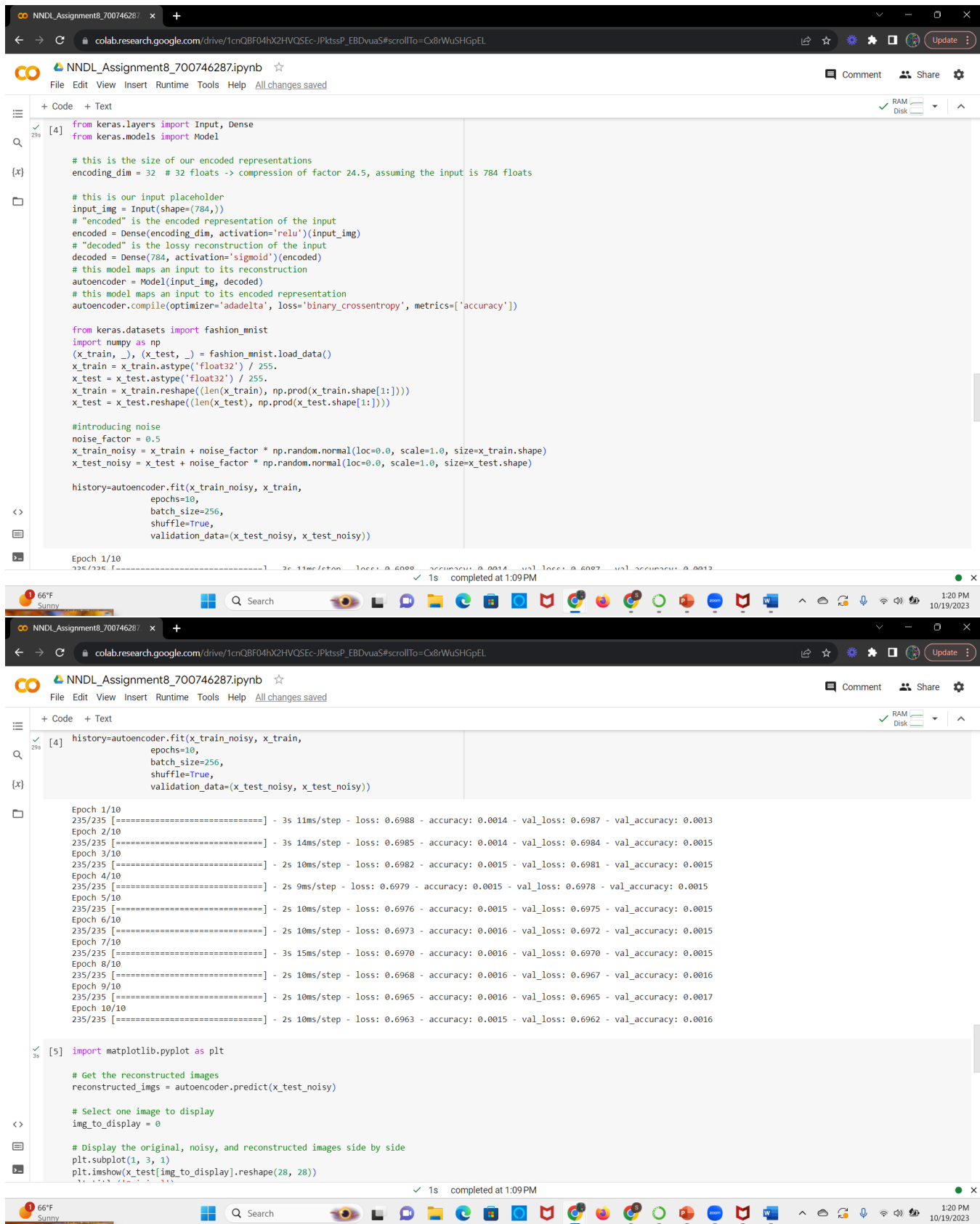
plt.subplot(2, 1, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()

plt.show()
```



1s completed at 1:09 PM

# For the denoising autoencoder we are adding the noise



The screenshot displays a Google Colab notebook titled "NNDL\_Assignment8\_700746287.ipynb". The notebook is open in a web browser, showing the code editor and the output of the code execution.

The code defines a denoising autoencoder model using Keras. It imports necessary libraries, defines the model architecture, and introduces noise to the training data. The model is trained for 10 epochs, and the output shows the training progress, including loss and accuracy metrics.

The output of the code execution shows the following results:

```
Epoch 1/10
235/235 [=====] - 3s 11ms/step - loss: 0.6988 - accuracy: 0.0014 - val_loss: 0.6987 - val_accuracy: 0.0013
Epoch 2/10
235/235 [=====] - 3s 14ms/step - loss: 0.6985 - accuracy: 0.0014 - val_loss: 0.6984 - val_accuracy: 0.0015
Epoch 3/10
235/235 [=====] - 2s 10ms/step - loss: 0.6982 - accuracy: 0.0015 - val_loss: 0.6981 - val_accuracy: 0.0015
Epoch 4/10
235/235 [=====] - 2s 9ms/step - loss: 0.6979 - accuracy: 0.0015 - val_loss: 0.6978 - val_accuracy: 0.0015
Epoch 5/10
235/235 [=====] - 2s 10ms/step - loss: 0.6976 - accuracy: 0.0015 - val_loss: 0.6975 - val_accuracy: 0.0015
Epoch 6/10
235/235 [=====] - 2s 10ms/step - loss: 0.6973 - accuracy: 0.0016 - val_loss: 0.6972 - val_accuracy: 0.0015
Epoch 7/10
235/235 [=====] - 3s 15ms/step - loss: 0.6970 - accuracy: 0.0016 - val_loss: 0.6970 - val_accuracy: 0.0015
Epoch 8/10
235/235 [=====] - 2s 10ms/step - loss: 0.6968 - accuracy: 0.0016 - val_loss: 0.6967 - val_accuracy: 0.0016
Epoch 9/10
235/235 [=====] - 2s 10ms/step - loss: 0.6965 - accuracy: 0.0016 - val_loss: 0.6965 - val_accuracy: 0.0017
Epoch 10/10
235/235 [=====] - 2s 10ms/step - loss: 0.6963 - accuracy: 0.0015 - val_loss: 0.6962 - val_accuracy: 0.0016
```

The code also includes a section for displaying the reconstructed images. It imports matplotlib.pyplot as plt, gets the reconstructed images, selects one image to display, and displays the original, noisy, and reconstructed images side by side.

## We are plotting the original image and the reconstructed image using the matplotlib library

