

NNDL: ICP6

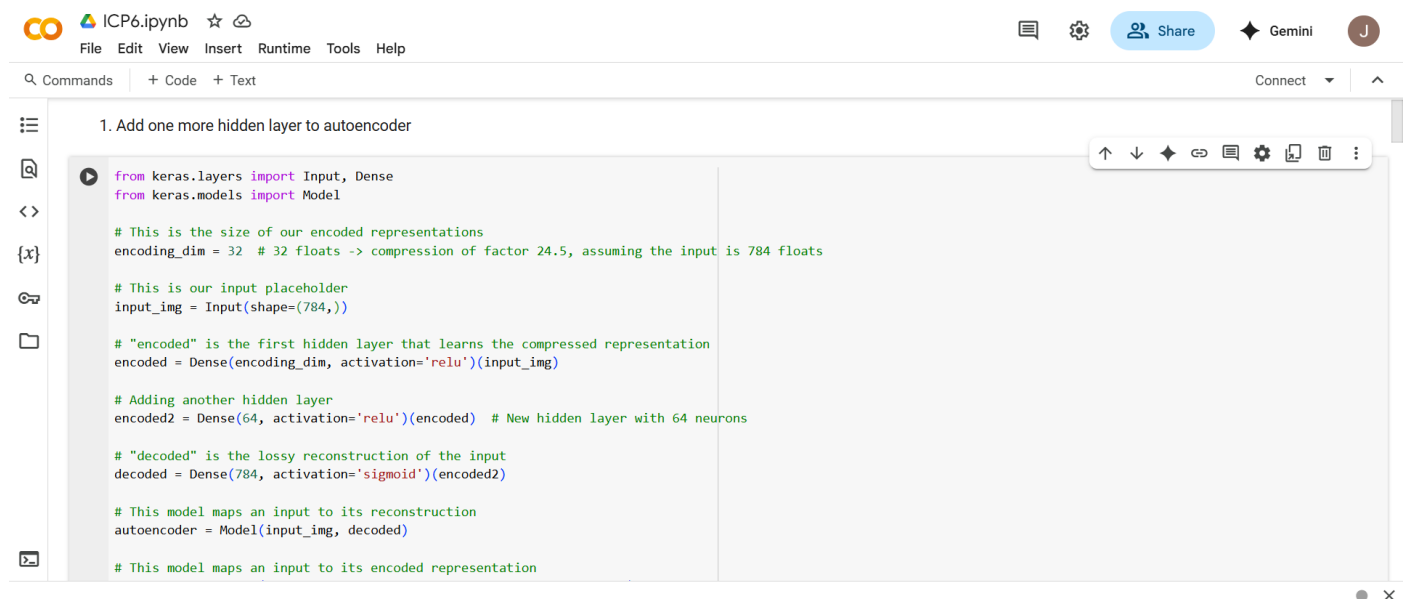
Jyothi Kiran Boddeda
700769023

GITHUB LINK : <https://github.com/jyothikiranboddeda/Neural-Network-Deep-Learning.git>

VIDEO LINK :

<https://drive.google.com/file/d/1NCZKv3f3vv-nUEZ1Kz7WkQJmrk9x1t51/view?usp=sharing>

1. Add one more hidden layer to autoencoder



```
ICP6.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text
Connect ^

1. Add one more hidden layer to autoencoder

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 first hidden layer that learns the compressed representation
encoded = Dense(encoding_dim, activation='relu')(input_img)

# Adding another hidden layer
encoded2 = Dense(64, activation='relu')(encoded) # New hidden layer with 64 neurons

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

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

# This model maps an input to its encoded representation
```



```
ICP6.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text
Connect ^

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

# 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')

# Loading the data
from keras.datasets import 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:])))

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

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

ICP6.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text

Connect

```
[ ] import matplotlib.pyplot as plt
import numpy as np

# Predict the reconstruction of the test data
reconstructed_imgs = autoencoder.predict(x_test)

# Choose an index to visualize
index = 10 # You can change this to any index you'd like

# Plot the original test image
plt.figure(figsize=(10, 5))

# Plot the original image
plt.subplot(1, 2, 1)
plt.imshow(x_test[index].reshape(28, 28), cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Plot the reconstructed image
plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[index].reshape(28, 28), cmap='gray')
```



3. Repeat the question 2 on the denoising autoencoder

```
ICP6.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
3.Repeat the question 2 on the denoising autoencoder

[ ] import matplotlib.pyplot as plt
import numpy as np

# Assuming that the model (autoencoder) has already been trained with the noisy data

# Step 2: Make predictions on the noisy test data
reconstructed_imgs = autoencoder.predict(X_test_noisy)

# Step 3: Choose an index to visualize
index = 10 # You can choose any index from the test data

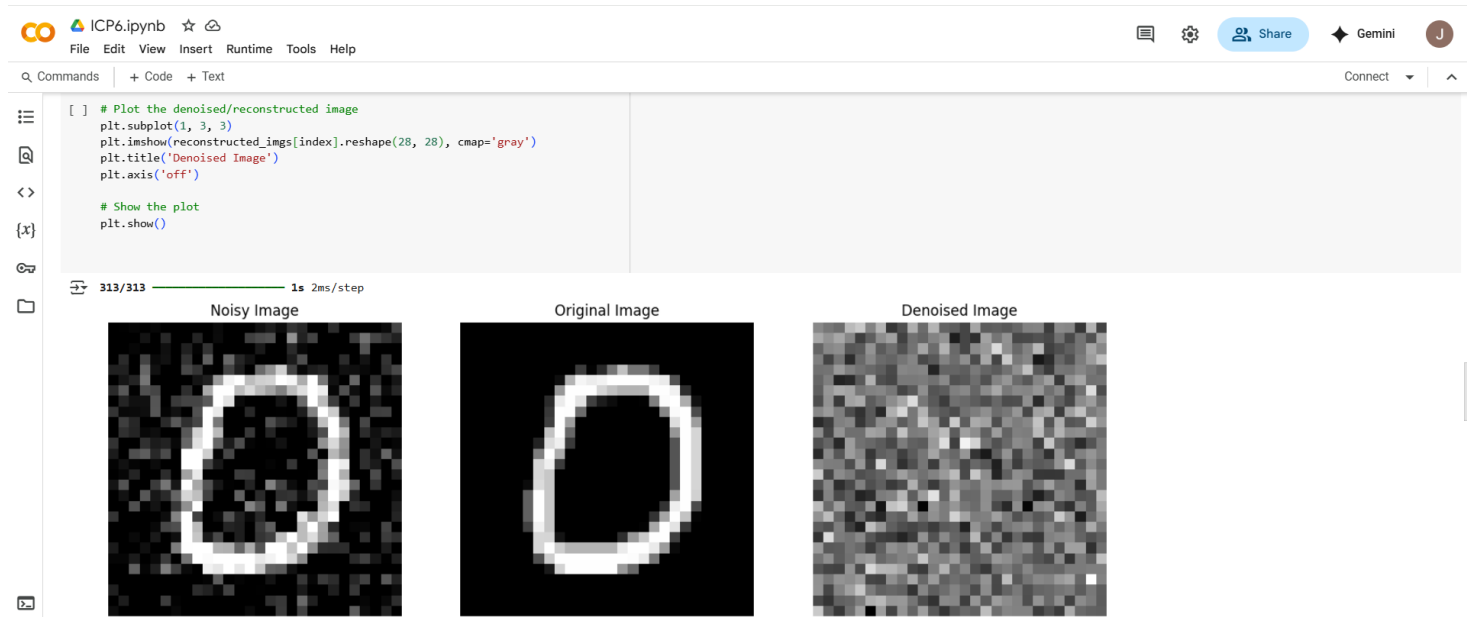
# Plot the noisy test image, original (clean) image, and reconstructed image

plt.figure(figsize=(15, 5))

# Plot the noisy test image
plt.subplot(1, 3, 1)
plt.imshow(X_test_noisy[index].reshape(28, 28), cmap='gray')
plt.title('Noisy Image')
plt.axis('off')

# Plot the original (clean) test image
plt.subplot(1, 3, 2)
plt.imshow(X_test[index].reshape(28, 28), cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Plot the denoised/reconstructed image
plt.subplot(1, 3, 3)
plt.imshow(reconstructed_imgs[index].reshape(28, 28), cmap='gray')
```



4. plot loss and accuracy using the history object



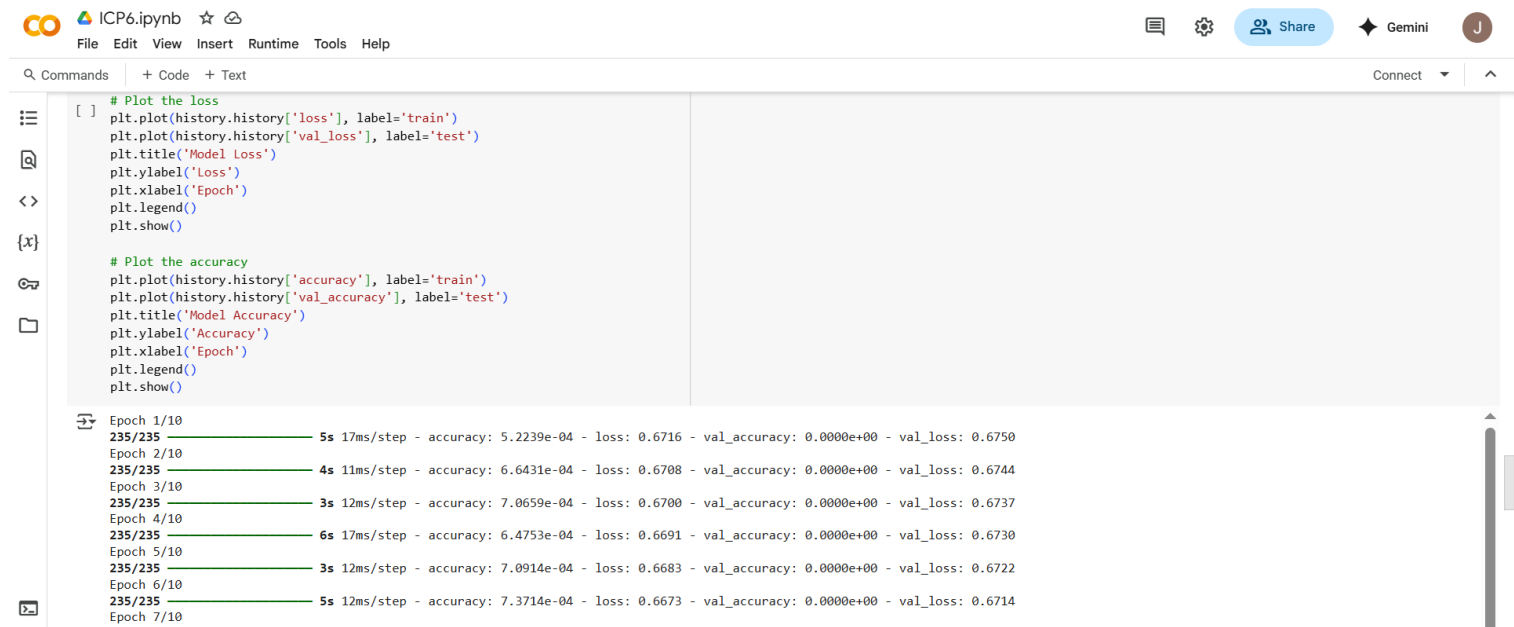
```
[ ] import matplotlib.pyplot as plt

# Train the autoencoder
# Include 'accuracy' in the metrics list during compilation
autoencoder.compile(optimizer='adadelata', loss='binary_crossentropy', metrics=['accuracy'])

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')
```



```
[ ] # 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	Time	Step	Accuracy	Loss	Val Accuracy	Val Loss
Epoch 1/10	235	235	5.2239e-04	0.6716	0.0000e+00	0.6750
Epoch 2/10	235	235	6.6431e-04	0.6708	0.0000e+00	0.6744
Epoch 3/10	235	235	7.0659e-04	0.6700	0.0000e+00	0.6737
Epoch 4/10	235	235	6.4753e-04	0.6691	0.0000e+00	0.6730
Epoch 5/10	235	235	7.0914e-04	0.6683	0.0000e+00	0.6722
Epoch 6/10	235	235	7.3714e-04	0.6673	0.0000e+00	0.6714
Epoch 7/10						

