

**Angel Reddy Nakkala**

**700748217**

## **Assignment-8**

**GitHub Link:**

<https://github.com/angelreddy09/DNN/tree/main/ICP8>

Lesson Overview:

In this lesson, we are going to discuss types and applications of Autoencoder.

Programming elements:

1. Basics of Autoencoders
2. Role of Autoencoders in unsupervised learning
3. Types of Autoencoders
4. Use case: Simple autoencoder-Reconstructing the existing image, which will contain most important features of the image
5. Use case: Stacked 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 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))

```

The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo and the notebook name 'Untitled0.ipynb'. The code cell contains the same Python code as the first block. The output cell shows the following text:

```

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 [=====] - 4s 13ms/step - loss: 0.6963 - val_loss: 0.6962
Epoch 2/5
235/235 [=====] - 5s 21ms/step - loss: 0.6961 - val_loss: 0.6960
Epoch 3/5
235/235 [=====] - 4s 18ms/step - loss: 0.6958 - val_loss: 0.6957
Epoch 4/5
235/235 [=====] - 4s 19ms/step - loss: 0.6956 - val_loss: 0.6955
Epoch 5/5
235/235 [=====] - 5s 21ms/step - loss: 0.6954 - val_loss: 0.6953
<keras.src.callbacks.History at 0x7a8a6940e4d0>

```

The bottom of the notebook shows a status bar indicating the code was completed at 1:27 PM.

## 1. Add one more hidden layer to autoencoder

```

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)

# Visualize reconstructed image and original image
import matplotlib.pyplot as plt

# Choose an index of a test image to visualize
idx = 10

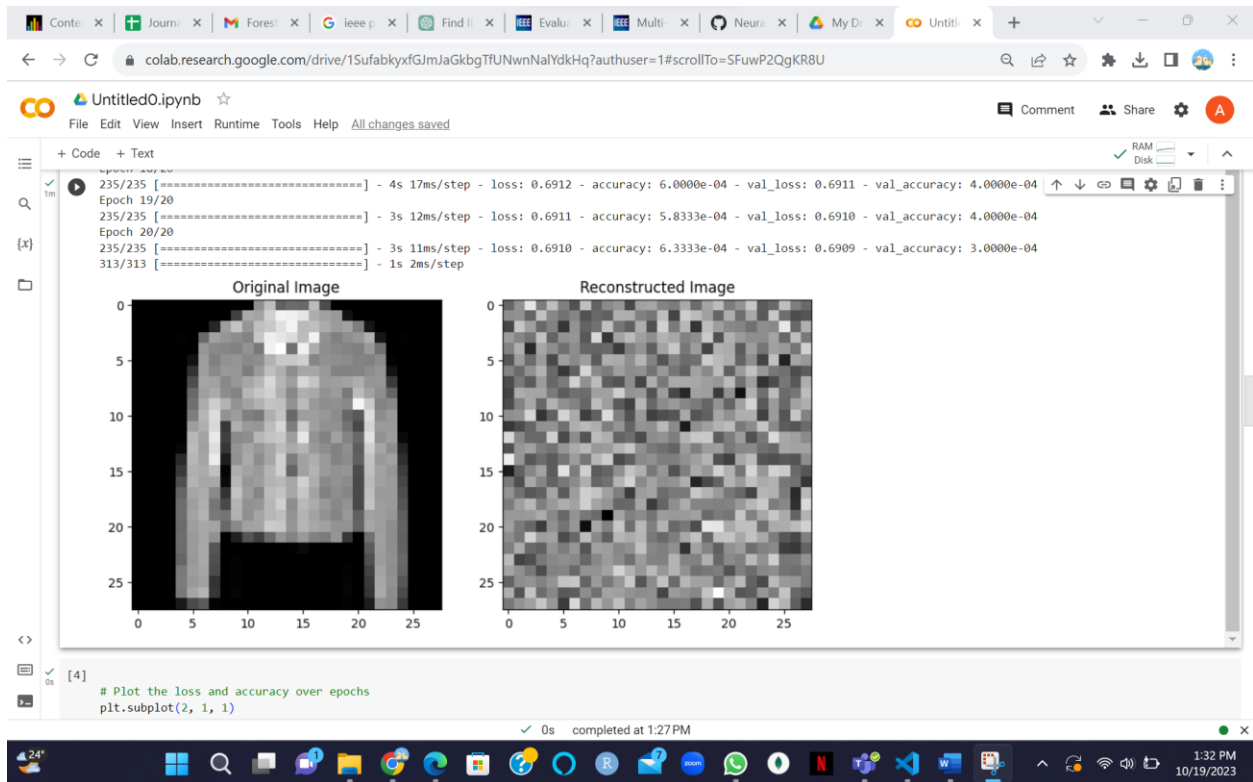
# Reshape the test image
test_img = x_test[idx].reshape(28, 28)

# 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()
```



plot loss and accuracy using the history object

```
# 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()
```



### 3. Repeat the question 2 on the denoising 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 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='adadelat', 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)

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

```

The screenshot shows a Google Colab notebook titled 'Untitled0.ipynb'. The code editor contains the same Python code as the first block. The output of the code is displayed in the lower half of the notebook, showing the training progress over 10 epochs. The output includes the following information for each epoch:

- Epoch 1/10: 235/235 [=====] - 4s 15ms/step - loss: 0.7004 - accuracy: 0.0013 - val\_loss: 0.7003 - val\_accuracy: 0.0012
- Epoch 2/10: 235/235 [=====] - 3s 13ms/step - loss: 0.7000 - accuracy: 0.0013 - val\_loss: 0.6998 - val\_accuracy: 0.0013
- Epoch 3/10: 235/235 [=====] - 2s 10ms/step - loss: 0.6995 - accuracy: 0.0013 - val\_loss: 0.6993 - val\_accuracy: 0.0013
- Epoch 4/10: 235/235 [=====] - 3s 11ms/step - loss: 0.6991 - accuracy: 0.0013 - val\_loss: 0.6989 - val\_accuracy: 0.0013
- Epoch 5/10: 235/235 [=====] - 3s 11ms/step - loss: 0.6987 - accuracy: 0.0013 - val\_loss: 0.6985 - val\_accuracy: 0.0014
- Epoch 6/10: 235/235 [=====] - 3s 13ms/step - loss: 0.6982 - accuracy: 0.0014 - val\_loss: 0.6981 - val\_accuracy: 0.0014
- Epoch 7/10: 235/235 [=====] - 3s 11ms/step - loss: 0.6979 - accuracy: 0.0014 - val\_loss: 0.6977 - val\_accuracy: 0.0014
- Epoch 8/10: 235/235 [=====] - 3s 11ms/step - loss: 0.6975 - accuracy: 0.0014 - val\_loss: 0.6973 - val\_accuracy: 0.0014
- Epoch 9/10: 235/235 [=====] - 2s 10ms/step - loss: 0.6971 - accuracy: 0.0014 - val\_loss: 0.6970 - val\_accuracy: 0.0014
- Epoch 10/10: 235/235 [=====] - 2s 10ms/step - loss: 0.6968 - accuracy: 0.0014 - val\_loss: 0.6966 - val\_accuracy: 0.0015

The final output shows the import of matplotlib and the prediction of reconstructed images:

```

[6] import matplotlib.pyplot as plt

# Get the reconstructed images
reconstructed_imgs = autoencoder.predict(x_test_noisy)

```

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

```
import matplotlib.pyplot as plt

# Get the reconstructed images
reconstructed_imgs = autoencoder.predict(x_test_noisy)

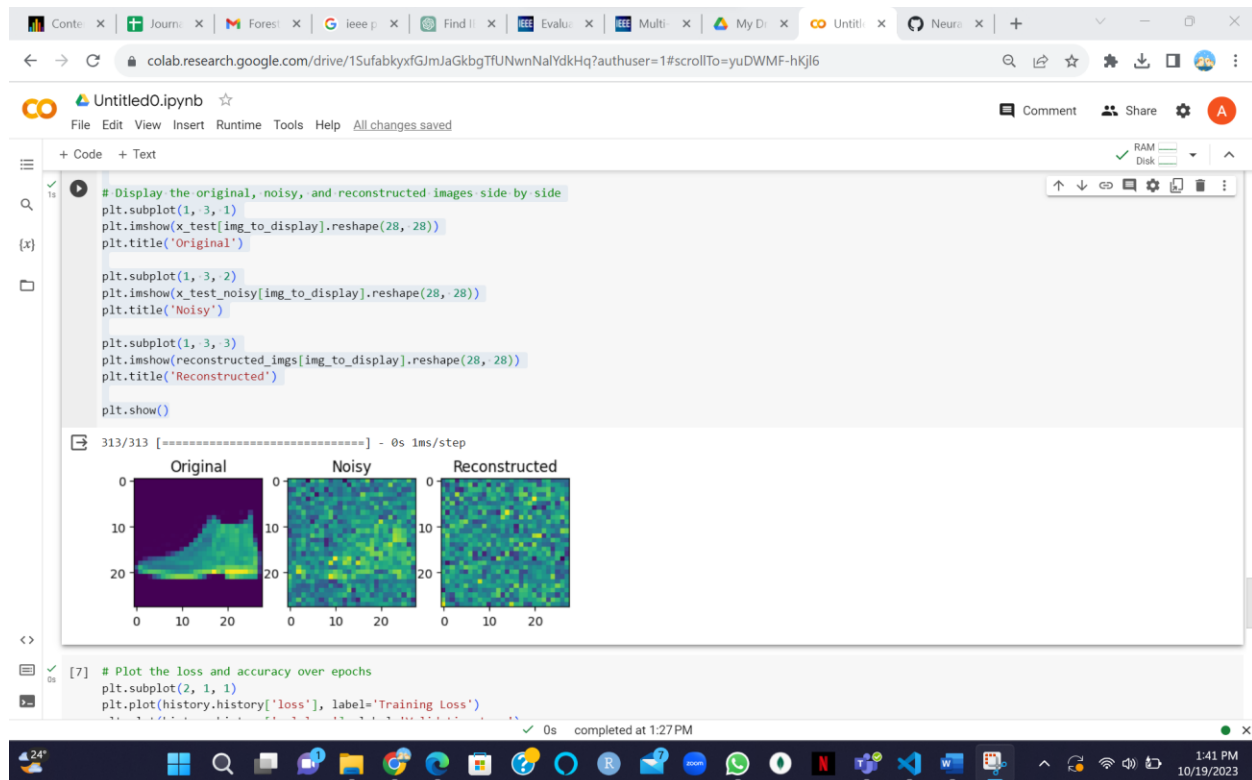
# Select one image to display
img_to_display = 0

# Display the original, noisy, and reconstructed images side by side
plt.subplot(1, 3, 1)
plt.imshow(x_test[img_to_display].reshape(28, 28))
plt.title('Original')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy')

plt.subplot(1, 3, 3)
plt.imshow(reconstructed_imgs[img_to_display].reshape(28, 28))
plt.title('Reconstructed')

plt.show()
```



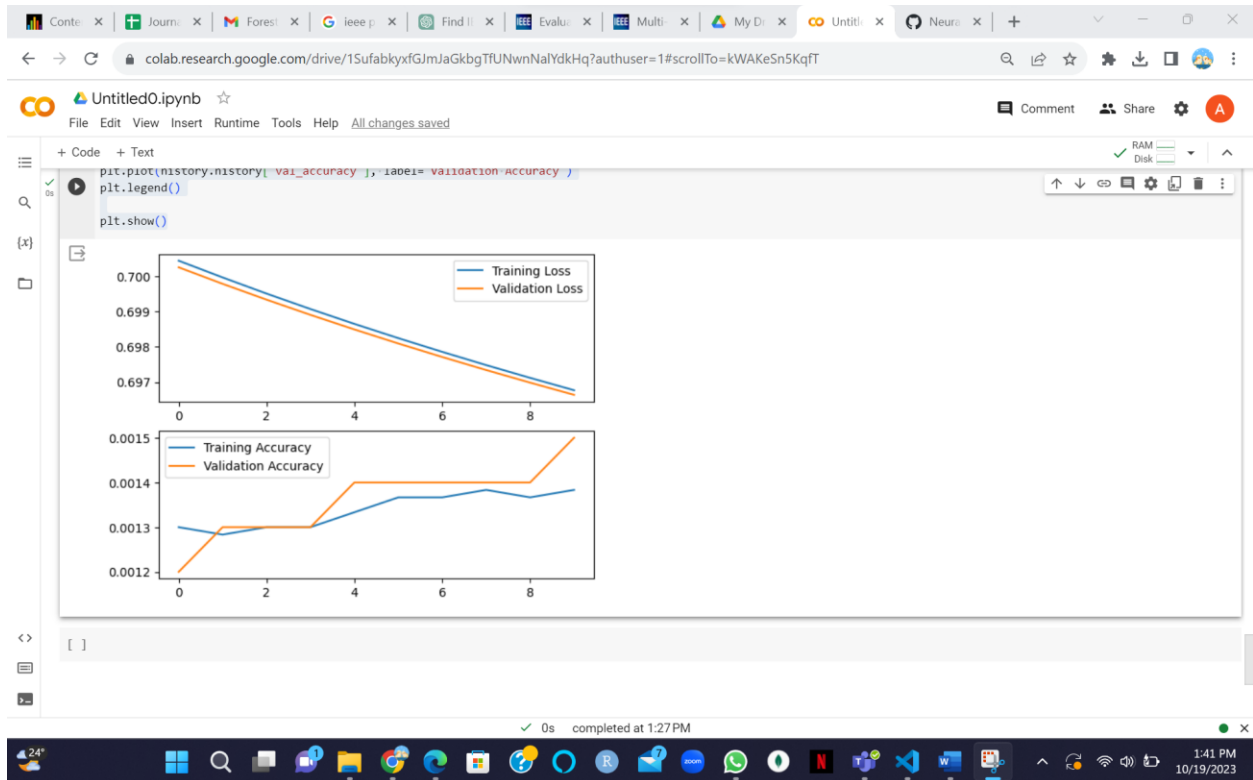
plot loss and accuracy using the history object

```
# 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()
```





GitHub Link:

<https://github.com/angelreddy09/DNN/tree/main/ICP8>