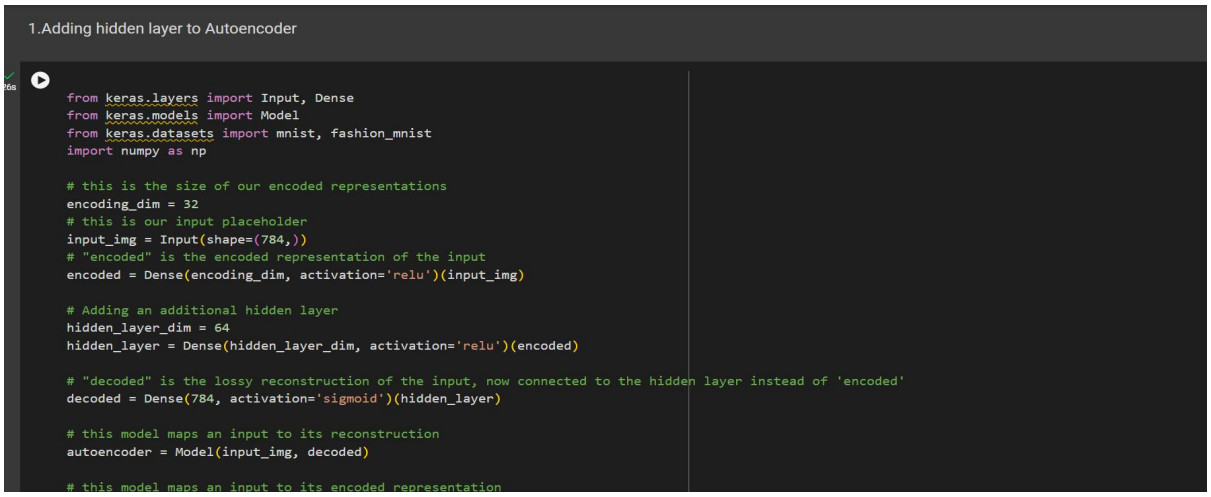
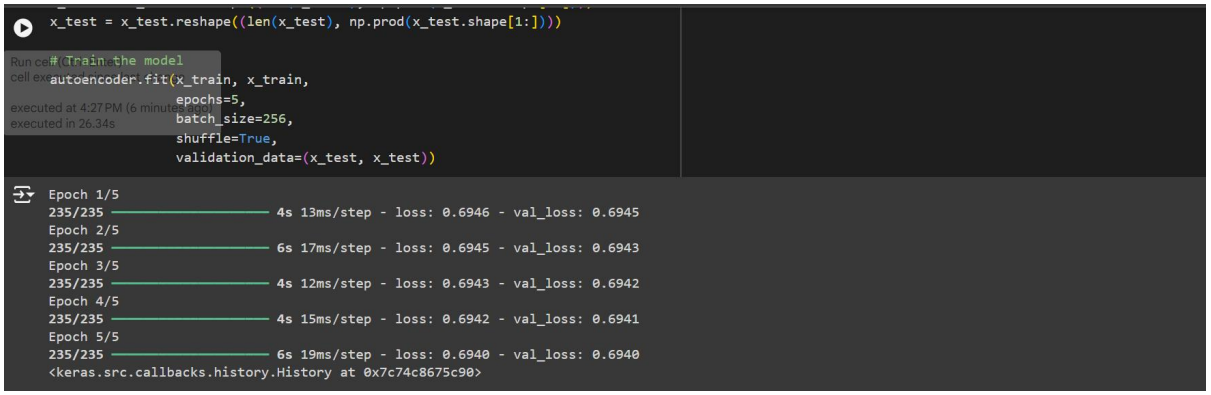
NEURAL NETWORK DEEP LEARNING
ICP 6
700765227
MANDHA SAIKUMAR REDDY

GitHub:
Repository URL for the source code:
https://github.com/saikumarreddyMandha/Assignment-6

Video Link:
https://drive.google.com/file/d/1q1zHUeda_FuybVgnOwEO_j8beHwvWwWw/view?usp=drive_link

1.Adding hidden layer to Autoencoder



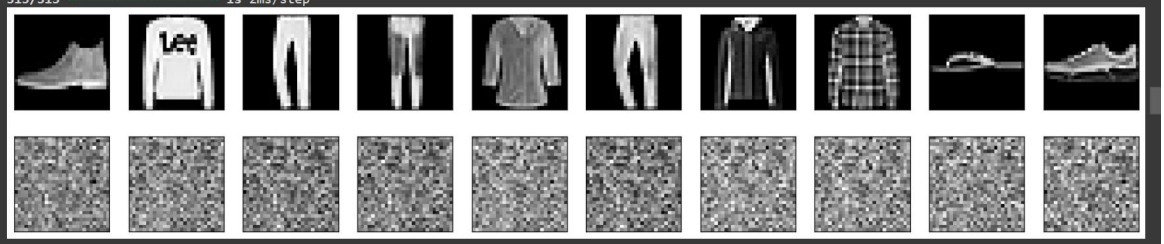


2.Prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

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

     # Define the model architecture
     encoding_dim = 32
     hidden_layer_dim = 64

     input_img = Input(shape=(784,))
     encoded = Dense(encoding_dim, activation='relu')(input_img)
     hidden_layer = Dense(hidden_layer_dim, activation='relu')(encoded)  # Additional hidden layer
     decoded = Dense(784, activation='sigmoid')(hidden_layer)

     autoencoder = Model(input_img, decoded)
     autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')

     # Load and prepare 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:])))
```

```
plt.show()
```

```
Epoch 1/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.6959 - val_loss: 0.6957
Epoch 2/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.6956 - val_loss: 0.6954
Epoch 3/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step - loss: 0.6953 - val_loss: 0.6951
Epoch 4/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.6950 - val_loss: 0.6948
Epoch 5/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - loss: 0.6948 - val_loss: 0.6946
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



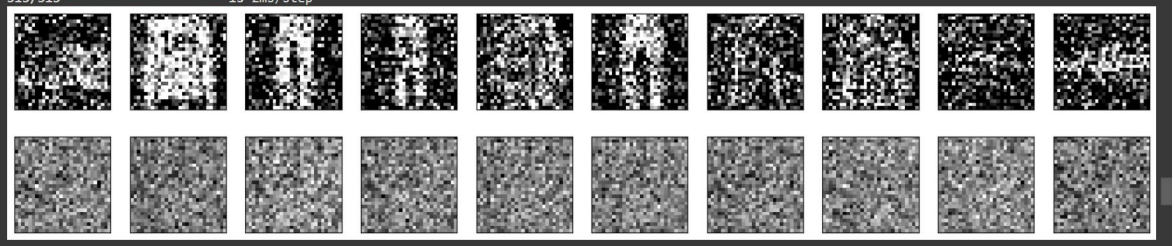
3.Denoising Autoencoder - prediction on the test data and then visualize one of the reconstructed version of that test data. Also, visualize the same test data before reconstruction using Matplotlib

```
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

# Define the model architecture
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')
# Load 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 noise
```

```
    # Display reconstruction
    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()
```

```
Epoch 1/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.6963 - val_loss: 0.6960
Epoch 2/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.6960 - val_loss: 0.6958
Epoch 3/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.6957 - val_loss: 0.6955
Epoch 4/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.6955 - val_loss: 0.6953
Epoch 5/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.6952 - val_loss: 0.6950
Epoch 6/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.6950 - val_loss: 0.6948
Epoch 7/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.6947 - val_loss: 0.6946
Epoch 8/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 11ms/step - loss: 0.6945 - val_loss: 0.6944
Epoch 9/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.6943 - val_loss: 0.6942
Epoch 10/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.6941 - val_loss: 0.6940
Epoch 11/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.6939 - val_loss: 0.6938
```

```
[4] Epoch 15/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 13ms/step - loss: 0.6932 - val_loss: 0.6931
Epoch 16/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.6930 - val_loss: 0.6929
Epoch 17/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.6928 - val_loss: 0.6927
Epoch 18/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.6927 - val_loss: 0.6926
Epoch 19/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.6925 - val_loss: 0.6924
Epoch 20/20
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - loss: 0.6924 - val_loss: 0.6922
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```



4.Plot loss and accuracy using the history object

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

# Load and prepare the Fashion MNIST data
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train.reshape(-1, 784).astype('float32') / 255
x_test = x_test.reshape(-1, 784).astype('float32') / 255

# Convert labels to one-hot encoding
num_classes = 10
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)

# Model architecture
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
decoded = Dense(10, activation='softmax')(encoded)  # Classification layer

model = Model(input_img, decoded)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.7185 - loss: 0.8426 - val_accuracy: 0.8283 - val_loss: 0.4965
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8500 - loss: 0.4343 - val_accuracy: 0.8524 - val_loss: 0.4266
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8652 - loss: 0.3853 - val_accuracy: 0.8607 - val_loss: 0.3955
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.8712 - loss: 0.3647 - val_accuracy: 0.8534 - val_loss: 0.4032
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8813 - loss: 0.3394 - val_accuracy: 0.8654 - val_loss: 0.3798
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.8846 - loss: 0.3226 - val_accuracy: 0.8665 - val_loss: 0.3677
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.8900 - loss: 0.3061 - val_accuracy: 0.8719 - val_loss: 0.3612
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.8919 - loss: 0.2978 - val_accuracy: 0.8725 - val_loss: 0.3539
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.8954 - loss: 0.2897 - val_accuracy: 0.8742 - val_loss: 0.3509
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9000 - loss: 0.2799 - val_accuracy: 0.8742 - val_loss: 0.3474
```

235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - accuracy: 0.9000 - loss: 0.2799 - val_accuracy: 0.8742 - val_loss: 0.3474