NITEESH KUMAR

700763258

ASSIGNMENT_8

NEURAL NETWORKS AND DEEP LEARNING

https://github.com/niteesh0301/Assignment_8.git

1.HIDDEN LAYER TO THE AUTOENCODER AND VISUALIZING THE DATA. ADDING

CODE :-

```
[35] from keras.layers import Input, Dense
     from keras.models import Model
     import matplotlib.pyplot as plt
```

```
# Define the size of encoded representations and the additional hidden layer size
encoded_representation_size = 32
hidden_layer_size = 32
```

```
[37] # Input placeholder
     input_img = Input(shape=(28*28,))
```

```
[38] # First encoding layer
     encoded1_layer = Dense(hidden_layer_size, activation='relu')(input_img)
```

```
[39] # Second encoding layer
     encoded2_layer = Dense(encoded_representation_size, activation='relu')(encoded1_layer)
```

```
[40] # First decoding layer
     decoded1_layer = Dense(hidden_layer_size, activation='relu')(encoded2_layer)
```

[41]
```
# Second decoding layer
decoded_output = Dense(784, activation='sigmoid')(decoded1_layer)
```

[42]
```
# Create the autoencoder model
autoencoder_model = Model(input_img, decoded_output)
```

[43]
```
# Compile the autoencoder model
autoencoder_model.compile(optimizer='adadelta', loss='binary_crossentropy')
```

[44]
```
# Load and preprocess the data
import tensorflow.keras.datasets.fashion_mnist as fashion_mnist
import numpy as np
```

[45]
```
(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:])))
```

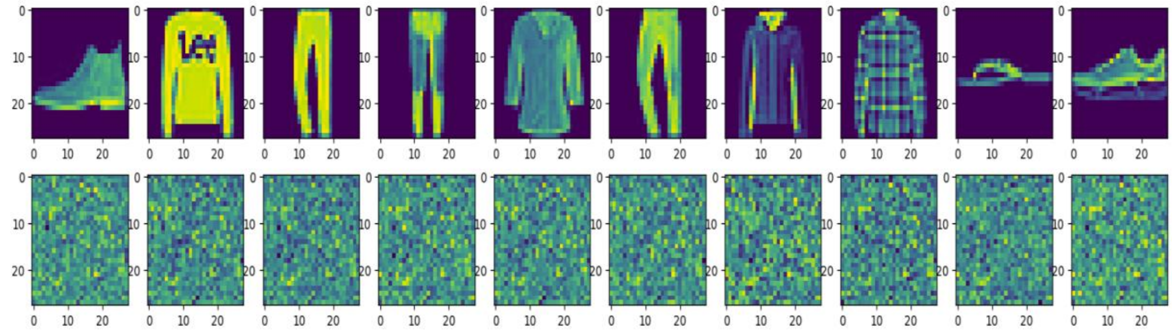
[46]
```
# Train the autoencoder model
training_history = autoencoder_model.fit(x_train, x_train, epochs=5, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

# Predict and visualize reconstructed test data
```

✓ 0s completed at 8:28 PM

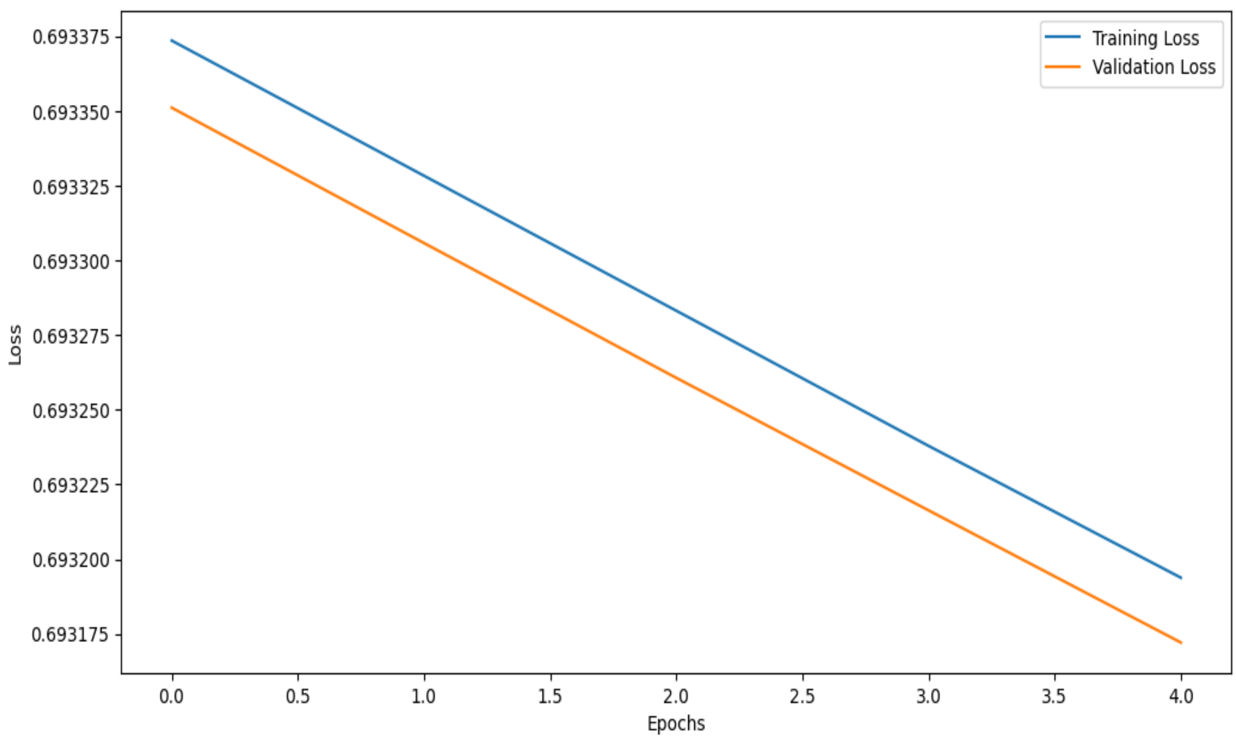
```
# Train the autoencoder model
training_history = autoencoder_model.fit(x_train, x_train, epochs=5, batch_size=256, shuffle=True, validation_data=(x_test, x_test))

# Predict and visualize reconstructed test data
decoded_images = autoencoder_model.predict(x_test)

# Display original and reconstructed images
num_display = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(num_display):
    # Display original images
    ax = plt.subplot(2, num_display, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))

    # Display reconstructed images
    ax = plt.subplot(2, num_display, i + 1 + num_display)
    plt.imshow(decoded_images[i].reshape(28, 28))

plt.show()

# Visualize the training and validation loss
plt.figure(figsize=(12, 6))
plt.plot(training_history.history['loss'], label='Training Loss')
plt.plot(training_history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output:-

```
Epoch 1/5
235/235 [==============================] - 5s 17ms/step - loss: 0.6934 - val_loss: 0.6934
Epoch 2/5
235/235 [==============================] - 3s 13ms/step - loss: 0.6933 - val_loss: 0.6933
Epoch 3/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6933 - val_loss: 0.6933
Epoch 4/5
235/235 [==============================] - 3s 12ms/step - loss: 0.6932 - val_loss: 0.6932
Epoch 5/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6932 - val_loss: 0.6932
313/313 [==============================] - 1s 3ms/step
```



Graph :-



2. ADDING HIDDEN LAYER TO THE DENOISING AUTOENCODER AND VISUALIZING THE DATA

CODE:-

```
[47] from keras.layers import Input, Dense
     from keras.models import Model
     import matplotlib.pyplot as plt
     import numpy as np
```

```
[48] # Define the size of the encoded representations and the size of the additional hidden layer
     encoding_dim = 32
     hidden_dim = 32
```

```
[49] # Define the input placeholder for the noisy data
     input_img = Input(shape=(28*28,))
```

```
# Define the first encoding layer
encoded1 = Dense(hidden_dim, activation='relu')(input_img)
```

```
[51] # Define the second encoding layer
     encoded2 = Dense(encoding_dim, activation='relu')(encoded1)
```

```
[52] # Define the first decoding layer
     decoded1 = Dense(hidden_dim, activation='relu')(encoded2)
```

```
[53] # Define the second decoding layer
     decoded = Dense(784, activation='sigmoid')(decoded1)
```

```
[54] # Create the denoising autoencoder model
     autoencoder = Model(input_img, decoded)
```

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

```
[56] # Load and preprocess the Fashion MNIST dataset
     from keras.datasets import fashion_mnist
     (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:])))
```

```python
# Load and preprocess the Fashion MNIST dataset
from keras.datasets import fashion_mnist
(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:])))
```

```python
# Introduce noise to the training and test data
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)
```

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

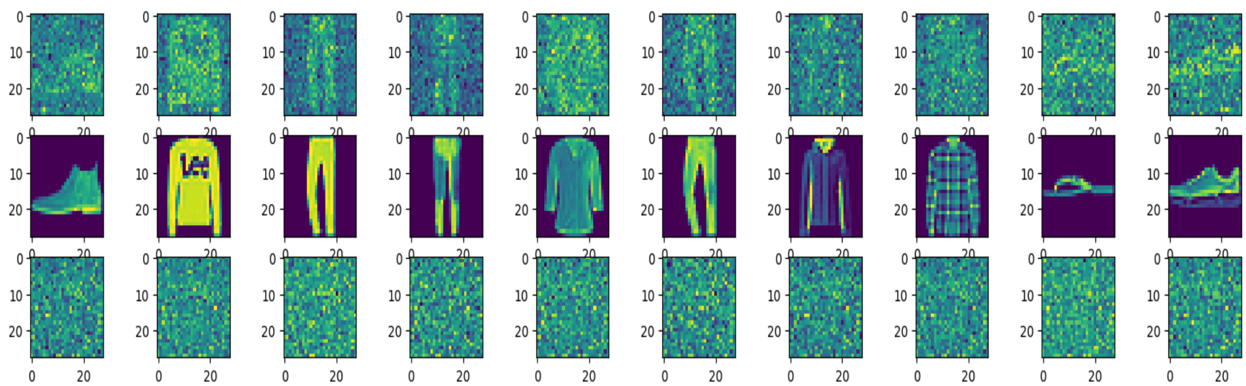
Output:-

```
Epoch 1/10
235/235 [==============================] - 4s 12ms/step - loss: 0.6937 - val_loss: 0.6937
Epoch 2/10
235/235 [==============================] - 4s 16ms/step - loss: 0.6936 - val_loss: 0.6936
Epoch 3/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6935 - val_loss: 0.6935
Epoch 4/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6934 - val_loss: 0.6934
Epoch 5/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6933 - val_loss: 0.6933
Epoch 6/10
235/235 [==============================] - 3s 12ms/step - loss: 0.6932 - val_loss: 0.6932
Epoch 7/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6931 - val_loss: 0.6931
Epoch 8/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6931 - val_loss: 0.6930
Epoch 9/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6930 - val_loss: 0.6929
Epoch 10/10
235/235 [==============================] - 3s 11ms/step - loss: 0.6929 - val_loss: 0.6928
```

```
[59] # Predict and visualize the reconstructed test data
     decoded_imgs = autoencoder.predict(x_test_noisy)

     313/313 [==============================] - 1s 3ms/step
```

```
n = 10  # Number of digits to display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display noisy images
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(x_test_noisy[i].reshape(28, 28))

    # Display original images
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(x_test[i].reshape(28, 28))

    # Display reconstructed images
    ax = plt.subplot(3, n, i + 1 + 2 * n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
```

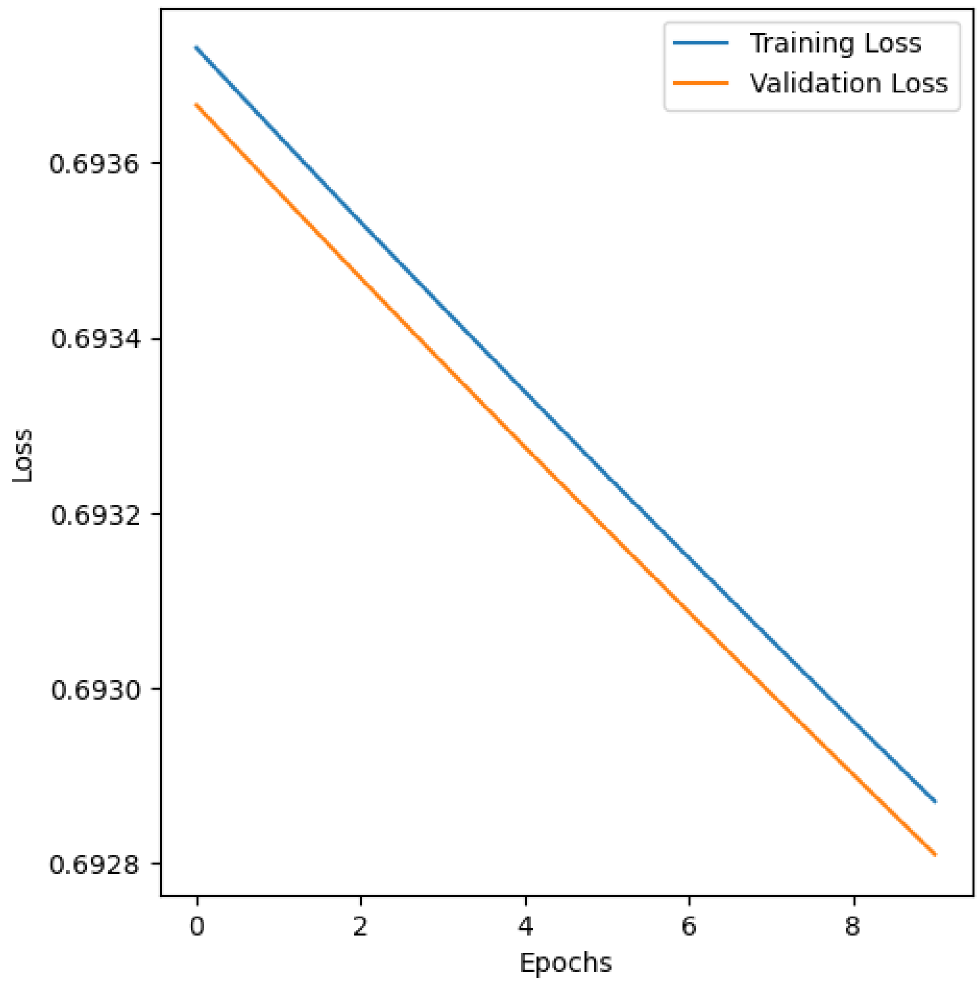
Output :-



Visualization:-

```
# Visualize the training and validation loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output :-



Training Loss
Validation Loss
0.6936
0.6934
0.6932
0.6930
0.6928
Loss
0
2
4
6
8
Epochs