Name: Shrinika Telu

Enroll No: 700741742

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

In class programming:

1. Add one more hidden layer to autoencoder

```python
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', 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:])))

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 [==============================] - 1s 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 [==============================] - 14s 41ms/step - loss: 0.6970 - accuracy: 6.8333e-04 - val_loss: 0.6969 - val_accuracy: 4.0000e-04
Epoch 2/5
235/235 [==============================] - 7s 28ms/step - loss: 0.6967 - accuracy: 7.1667e-04 - val_loss: 0.6966 - val_accuracy: 3.0000e-04
Epoch 3/5
235/235 [==============================] - 6s 26ms/step - loss: 0.6965 - accuracy: 7.3333e-04 - val_loss: 0.6963 - val_accuracy: 3.0000e-04
Epoch 4/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6962 - accuracy: 7.3333e-04 - val_loss: 0.6961 - val_accuracy: 4.0000e-04
Epoch 5/5
235/235 [==============================] - 3s 15ms/step - loss: 0.6959 - accuracy: 7.3333e-04 - val_loss: 0.6958 - val_accuracy: 4.0000e-04
<keras.callbacks.History at 0x7f5afa406560>
```

[1]
```
from keras.layers import Input, Dense
from keras.models import Model

# This is the size of our encoded representation
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
encoded1 = Dense(128, activation='relu')(input_img)
encoded2 = Dense(encoding_dim, activation='relu')(encoded1)

# "decoded" is the lossy reconstruction of the input
decoded1 = Dense(128, activation='relu')(encoded2)
decoded2 = Dense(784, activation='sigmoid')(decoded1)

# This model maps an input to its reconstruction
autoencoder = Model(input_img, decoded2)

# This model maps an input to its encoded representation
encoder = Model(input_img, encoded2)

# This is our decoder model
encoded_input = Input(shape=(encoding_dim,))
decoder_layer1 = autoencoder.layers[-2]
decoder_layer2 = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer2(decoder_layer1(encoded_input)))

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

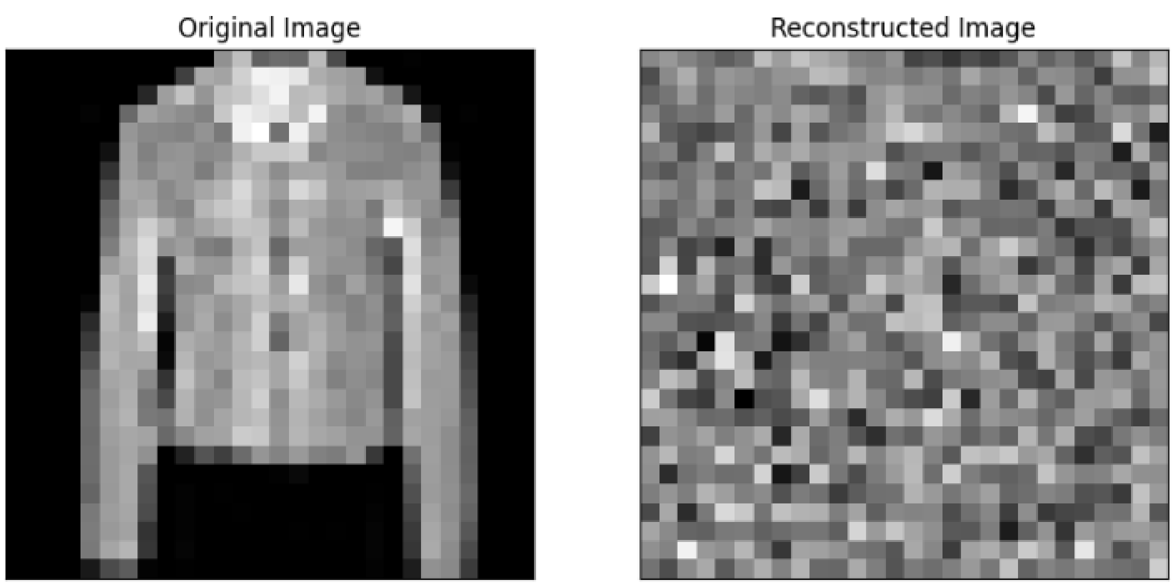
```
# Load the MNIST dataset
from keras.datasets import mnist, fashion_mnist
import numpy as np
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# Normalize and flatten the 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 the autoencoder
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 [==============================] - 17s 50ms/step - loss: 0.6933 - accuracy: 9.3333e-04 - val_loss: 0.6932 - val_accuracy: 7.0000e-04
Epoch 2/5
235/235 [==============================] - 8s 34ms/step - loss: 0.6932 - accuracy: 9.8333e-04 - val_loss: 0.6931 - val_accuracy: 7.0000e-04
Epoch 3/5
235/235 [==============================] - 5s 21ms/step - loss: 0.6930 - accuracy: 0.0010 - val_loss: 0.6929 - val_accuracy: 6.0000e-04
Epoch 4/5
235/235 [==============================] - 5s 23ms/step - loss: 0.6928 - accuracy: 0.0010 - val_loss: 0.6928 - val_accuracy: 6.0000e-04
Epoch 5/5
235/235 [==============================] - 5s 23ms/step - loss: 0.6927 - accuracy: 0.0011 - val_loss: 0.6926 - val_accuracy: 7.0000e-04
<keras.callbacks.History at 0x7e247cda1cc0>
```

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

```python
import matplotlib.pyplot as plt

# Get the reconstructed images for the test set
reconstructed_imgs = autoencoder.predict(x_test)

# Choose a random image from the test set
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))

# Plot the original image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Original Image")

# Plot the reconstructed image
ax = plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")

plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```



3. Repeat the question 2 on the denoisening autoencoder

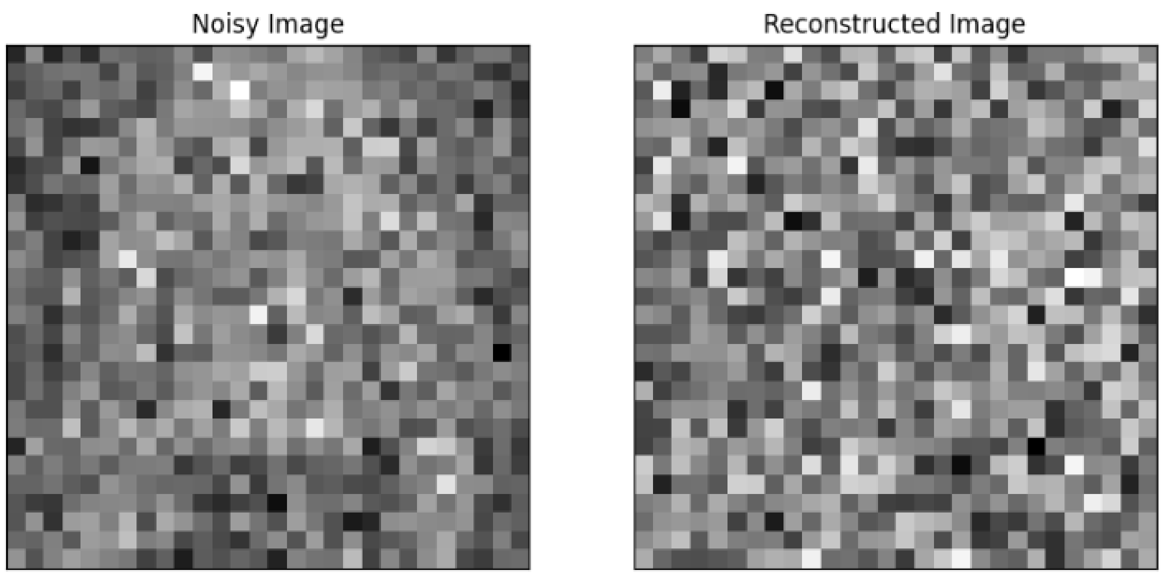
```
[3] encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

    # 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',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)

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

```
Epoch 1/10
235/235 [==============================] - 5s 19ms/step - loss: 0.6973 - accuracy: 0.0010 - val_loss: 0.6972 - val_accuracy: 7.0000e-04
Epoch 2/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6971 - accuracy: 0.0010 - val_loss: 0.6970 - val_accuracy: 7.0000e-04
Epoch 3/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6969 - accuracy: 0.0010 - val_loss: 0.6968 - val_accuracy: 7.0000e-04
Epoch 4/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6967 - accuracy: 0.0011 - val_loss: 0.6966 - val_accuracy: 7.0000e-04
Epoch 5/10
235/235 [==============================] - 4s 18ms/step - loss: 0.6965 - accuracy: 0.0011 - val_loss: 0.6964 - val_accuracy: 7.0000e-04
Epoch 6/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6963 - accuracy: 0.0011 - val_loss: 0.6962 - val_accuracy: 7.0000e-04
Epoch 7/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6961 - accuracy: 0.0011 - val_loss: 0.6960 - val_accuracy: 7.0000e-04
Epoch 8/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6959 - accuracy: 0.0011 - val_loss: 0.6959 - val_accuracy: 7.0000e-04
Epoch 9/10
235/235 [==============================] - 4s 17ms/step - loss: 0.6958 - accuracy: 0.0011 - val_loss: 0.6957 - val_accuracy: 8.0000e-04
Epoch 10/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6956 - accuracy: 0.0011 - val_loss: 0.6955 - val_accuracy: 8.0000e-04
```

[4]

```python
import matplotlib.pyplot as plt

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

# Choose a random image from the test set
n = 10  # index of the image to be plotted
plt.figure(figsize=(10, 5))

# Plot the original noisy image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Noisy Image")

# Plot the reconstructed image
ax = plt.subplot(1, 2, 2)
plt.imshow(reconstructed_imgs[n].reshape(28, 28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
ax.set_title("Reconstructed Image")

plt.show()
```

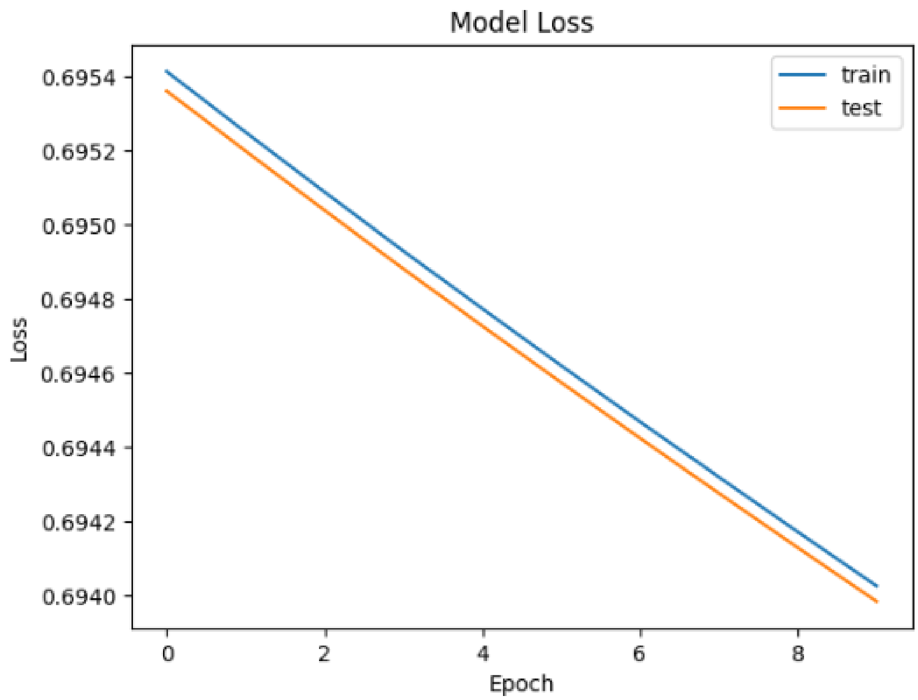
```
313/313 [==============================] - 1s 2ms/step
```

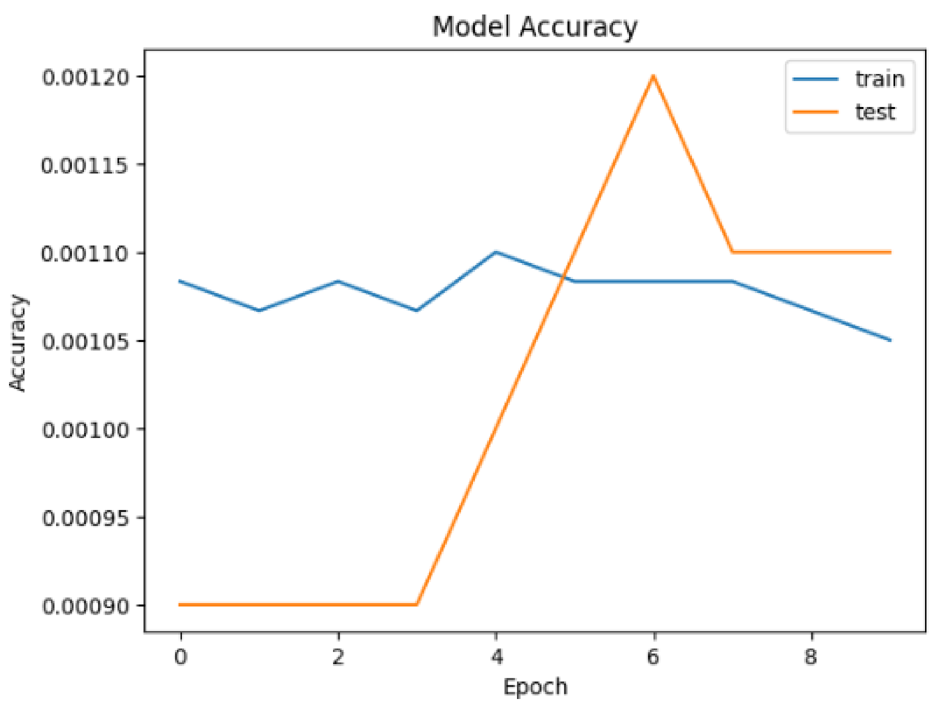


4. plot loss and accuracy using the history object

```
[5] import matplotlib.pyplot as plt

    # Train the autoencoder
    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')
    plt.legend()
    plt.show()
```

```
Epoch 1/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6954 - accuracy: 0.0011 - val_loss: 0.6954 - val_accuracy: 9.0000e-04
Epoch 2/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6952 - accuracy: 0.0011 - val_loss: 0.6952 - val_accuracy: 9.0000e-04
Epoch 3/10
235/235 [==============================] - 4s 18ms/step - loss: 0.6951 - accuracy: 0.0011 - val_loss: 0.6950 - val_accuracy: 9.0000e-04
Epoch 4/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6949 - accuracy: 0.0011 - val_loss: 0.6949 - val_accuracy: 9.0000e-04
Epoch 5/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6948 - accuracy: 0.0011 - val_loss: 0.6947 - val_accuracy: 0.0010
Epoch 6/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6946 - accuracy: 0.0011 - val_loss: 0.6946 - val_accuracy: 0.0011
Epoch 7/10
235/235 [==============================] - 4s 18ms/step - loss: 0.6945 - accuracy: 0.0011 - val_loss: 0.6944 - val_accuracy: 0.0012
Epoch 8/10
235/235 [==============================] - 3s 13ms/step - loss: 0.6943 - accuracy: 0.0011 - val_loss: 0.6943 - val_accuracy: 0.0011
Epoch 9/10
235/235 [==============================] - 3s 14ms/step - loss: 0.6942 - accuracy: 0.0011 - val_loss: 0.6941 - val_accuracy: 0.0011
Epoch 10/10
235/235 [==============================] - 4s 15ms/step - loss: 0.6940 - accuracy: 0.0010 - val_loss: 0.6940 - val_accuracy: 0.0011
```


Model Loss
Loss
Epoch
train
test
0.6954
0.6952
0.6950
0.6948
0.6946
0.6944
0.6942
0.6940
0
2
4
6
8


Model Accuracy
Accuracy
Epoch
train
test
0.00120
0.00115
0.00110
0.00105
0.00100
0.00095
0.00090
0
2
4
6
8