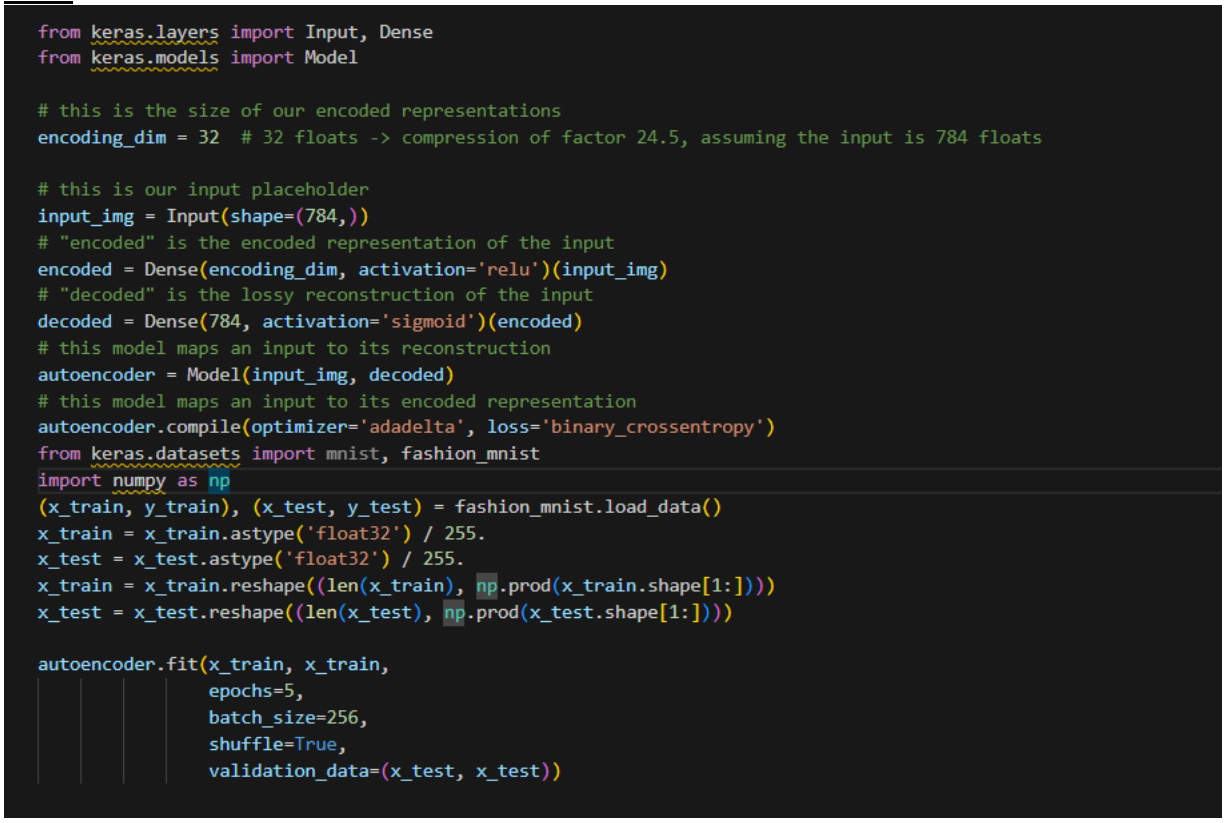
Neural ICP6

Nithin Thota,
700764916

GitHub Link: https://github.com/NithinThota9/ICP6
Video Link: https://drive.google.com/file/d/18D1abAuQDrXN4xCUJzjjO6Nqy1f4hKHw/view?usp=sharing

Question 1:
Code:



```
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))
```

Explanation:
1. Autoencoder Architecture: The network is structured with an input layer, an encoded layer to compress the input, an additional hidden layer for potentially capturing more complex patterns, and a decoded layer to reconstruct the input from its encoded representation.

2. Encoding Dimension: The encoded layer compresses the input to 32 floating point numbers, significantly reducing the dimensionality from the original 784 floats (assuming the input images are 28x28 pixels, flattened to 784 floats for a fully connected network).

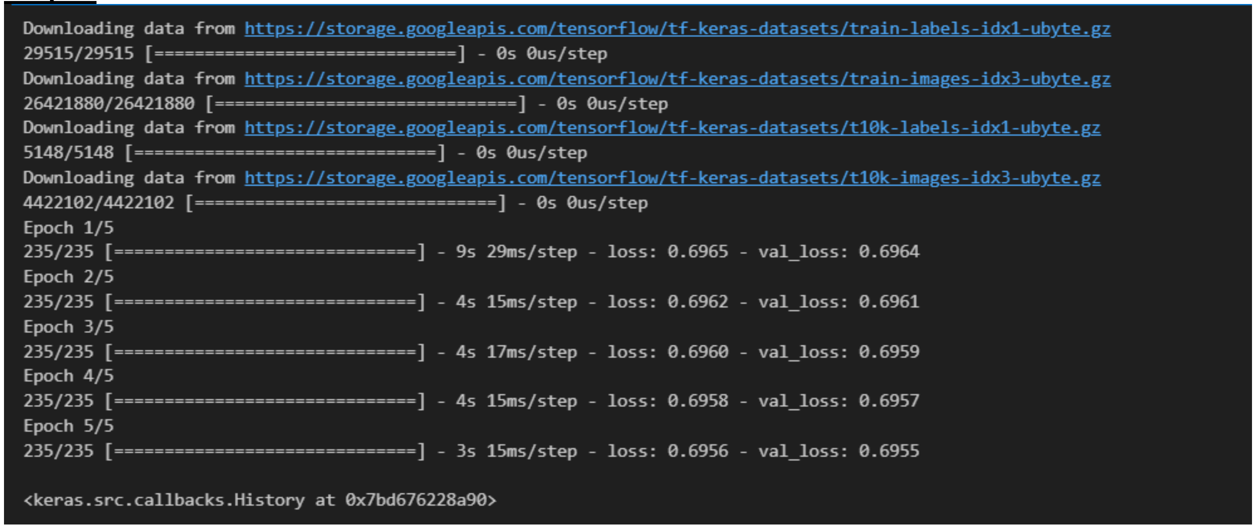
3. Hidden Layer: An additional hidden layer with 64 units is introduced after the encoding layer. This can help in learning more complex representations and aid in the decoding process. The activation function for both the encoded layer and the hidden layer is ReLU (Rectified Linear Unit), which introduces non-linearity into the model.

4. Loss Function and Optimizer: The model uses binary crossentropy as the loss function, which is common for reconstruction tasks, and the 'adadelta' optimizer, which is an adaptive learning rate method.

5. Data Preparation: The Fashion MNIST dataset is loaded, normalized to have pixel values between 0 and 1 (by dividing by 255), and reshaped to fit the model's input requirements. The dataset consists of 28x28 pixel grayscale images of clothing items, which are flattened to 784 dimensions to match the input layer of the model.

6. Model Training: The autoencoder is trained on the Fashion MNIST dataset for 5 epochs with a batch size of 256, using both the training and validation datasets. Here, the model learns to compress (encode) the input data and then reconstruct (decode) it as closely as possible to the original input.
7. Purpose and Application: Autoencoders like this one are used for dimensionality reduction, feature learning, and denoising images. By learning to reconstruct the input data from a compressed representation, the model can discover important features and patterns in the data.

Output:

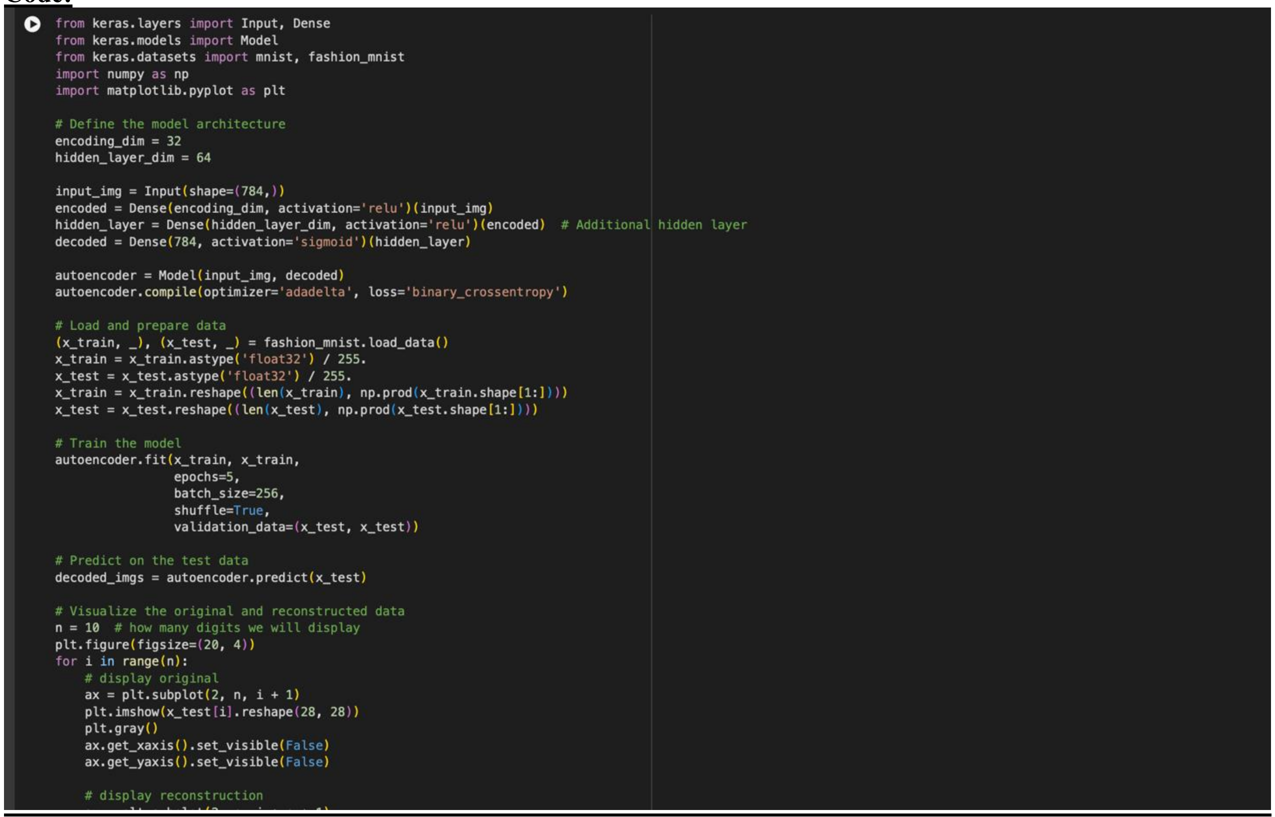


```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 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 [==============================] - 9s 29ms/step - loss: 0.6965 - val_loss: 0.6964
Epoch 2/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6962 - val_loss: 0.6961
Epoch 3/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6960 - val_loss: 0.6959
Epoch 4/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6958 - val_loss: 0.6957
Epoch 5/5
235/235 [==============================] - 3s 15ms/step - loss: 0.6956 - val_loss: 0.6955

<keras.src.callbacks.History at 0x7bd676228a90>
```

Question 2:

Code:



```python
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:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

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

# Predict on the test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize the original and reconstructed data
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
```

Explanation:

1. Model Architecture: The model consists of an input layer that accepts flattened 28x28 grayscale images (784 pixels), an encoding layer to compress the input to a 32-dimensional representation, an additional hidden layer with 64 units for more complex feature extraction, and a decoding layer to reconstruct the original image from the compressed form.

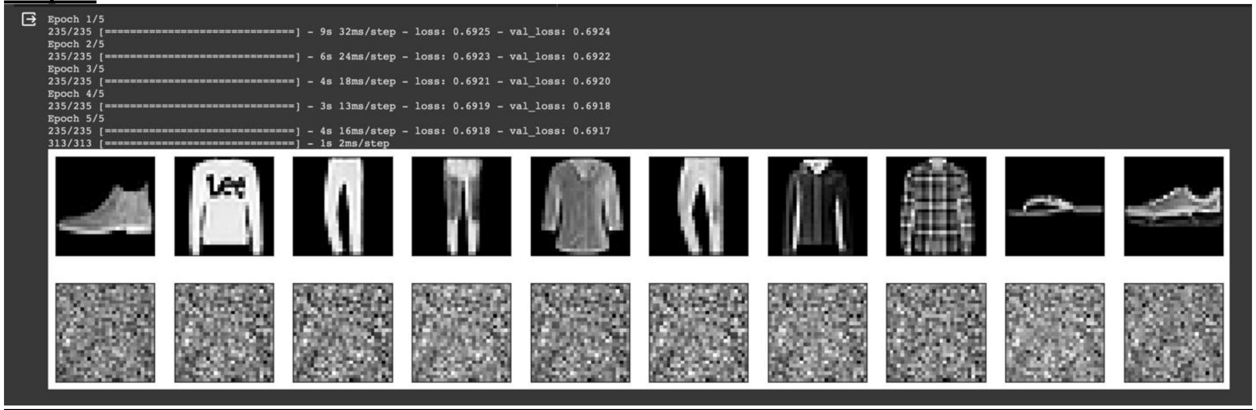
2. Compilation: The autoencoder is compiled with the Adadelta optimizer and binary crossentropy loss, a common setup for autoencoders since the task is to reproduce the input image as closely as possible.

3. Data Preparation: The Fashion MNIST dataset, comprising 28x28 grayscale images of clothing items, is loaded, normalized (pixel values scaled between 0 and 1), and reshaped to fit the model.

4. Training: The model is trained on the Fashion MNIST training data for 5 epochs, using a batch size of 256. Validation is performed using the test set.

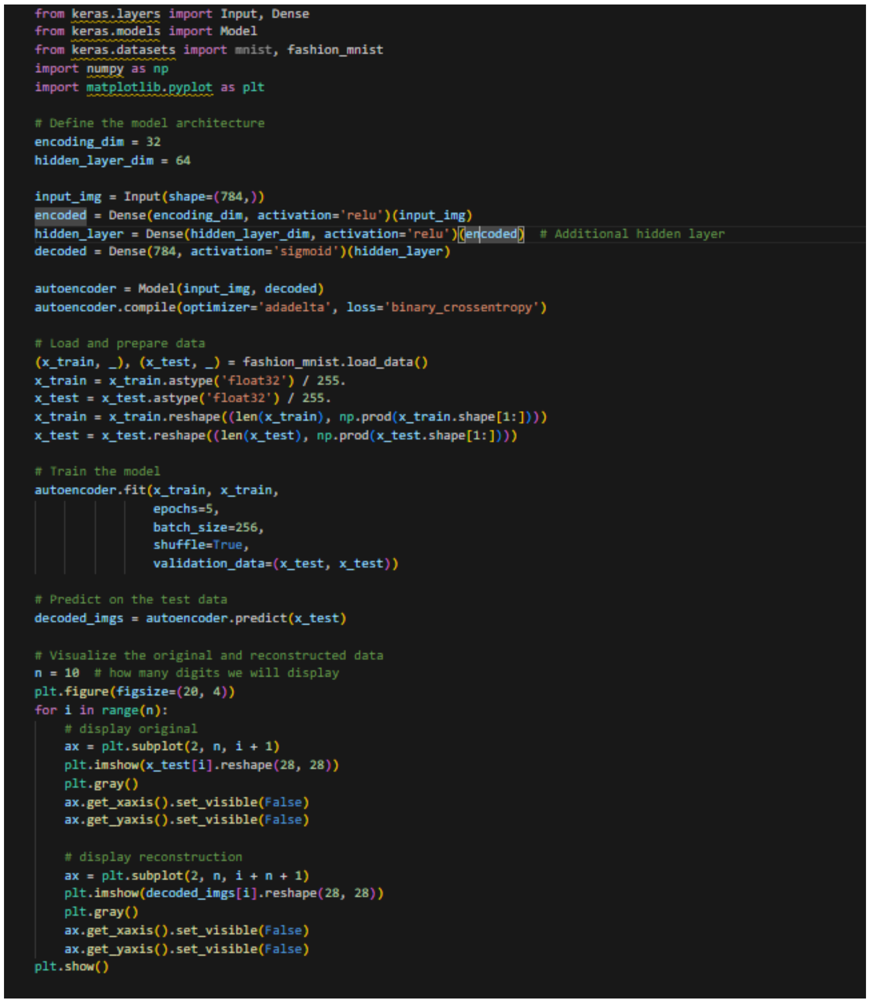
5. Visualization: After training, the model predicts on the test set, and a comparison between original images and their reconstructed counterparts is visualized for 10 example cases. This step helps to visually assess the model's performance in terms of how well it can reconstruct the input images after compression and decompression.

Output:



```
Epoch 1/5
235/235 [==============================] - 9s 32ms/step - loss: 0.6925 - val_loss: 0.6924
Epoch 2/5
235/235 [==============================] - 6s 24ms/step - loss: 0.6923 - val_loss: 0.6922
Epoch 3/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6921 - val_loss: 0.6920
Epoch 4/5
235/235 [==============================] - 3s 13ms/step - loss: 0.6919 - val_loss: 0.6918
Epoch 5/5
235/235 [==============================] - 4s 16ms/step - loss: 0.6918 - val_loss: 0.6917
313/313 [==============================] - 1s 2ms/step
```

Question 3:
Code:



```python
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:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

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

# Predict on the test data
decoded_imgs = autoencoder.predict(x_test)

# Visualize the original and reconstructed data
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(2, n, i + n + 1)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

Explanation:

1. Autoencoder Architecture: A simple neural network with an input layer, one hidden layer (encoded representation), and an output layer (decoded representation). The network compresses the input to a lower-dimensional encoded representation and then reconstructs the output from this encoding.

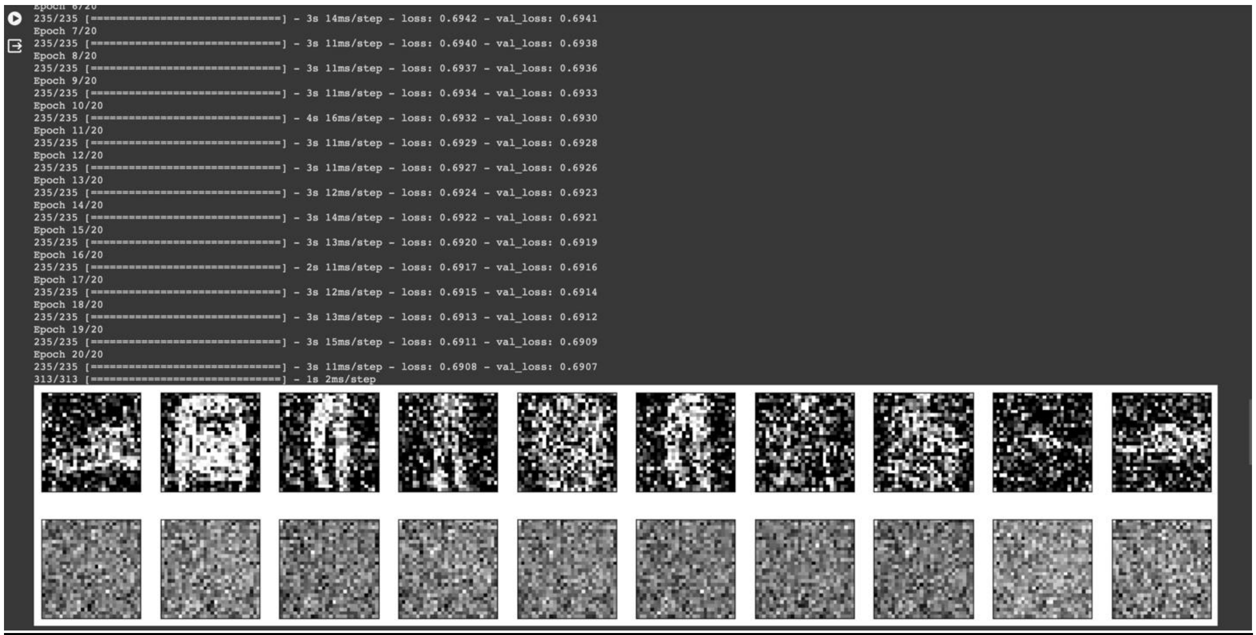
2. Data Preparation: The Fashion MNIST dataset, consisting of 28x28 grayscale images of clothing items, is loaded and normalized. The images are flattened and converted to floating-point arrays with values between 0 and 1.

3. Noise Addition: Artificial noise is added to the images to simulate corrupted input data. This is done by adding Gaussian noise to the training and test sets, followed by clipping to ensure the pixel values remain between 0 and 1.

4. Training: The autoencoder is trained using the noisy images as input and the original, clean images as the target for reconstruction. The model learns to filter out the noise and recover the original images from the noisy inputs.

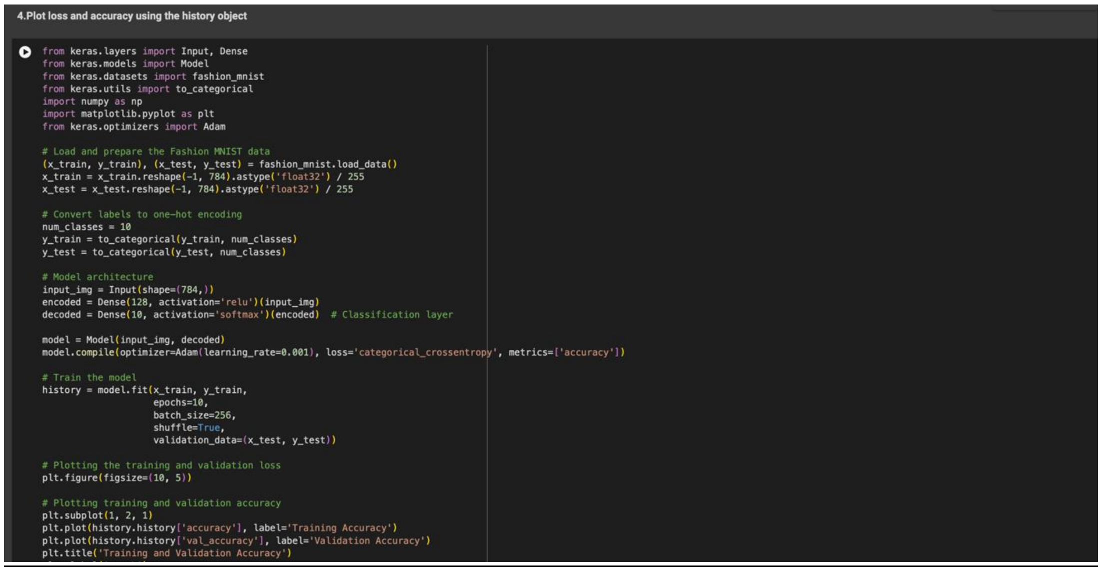
5. Evaluation and Visualization: After training, the autoencoder is used to predict (denoise) the test set images. The original noisy images and their denoised reconstructions are then visualized side by side for comparison.

Output:



```
Epoch 6/20
235/235 [==============================] - 3s 14ms/step - loss: 0.6942 - val_loss: 0.6941
Epoch 7/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6940 - val_loss: 0.6938
Epoch 8/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6937 - val_loss: 0.6936
Epoch 9/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6934 - val_loss: 0.6933
Epoch 10/20
235/235 [==============================] - 4s 16ms/step - loss: 0.6932 - val_loss: 0.6930
Epoch 11/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6929 - val_loss: 0.6928
Epoch 12/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6927 - val_loss: 0.6926
Epoch 13/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6924 - val_loss: 0.6923
Epoch 14/20
235/235 [==============================] - 3s 14ms/step - loss: 0.6922 - val_loss: 0.6921
Epoch 15/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6920 - val_loss: 0.6919
Epoch 16/20
235/235 [==============================] - 2s 11ms/step - loss: 0.6917 - val_loss: 0.6916
Epoch 17/20
235/235 [==============================] - 3s 12ms/step - loss: 0.6915 - val_loss: 0.6914
Epoch 18/20
235/235 [==============================] - 3s 13ms/step - loss: 0.6913 - val_loss: 0.6912
Epoch 19/20
235/235 [==============================] - 3s 15ms/step - loss: 0.6911 - val_loss: 0.6909
Epoch 20/20
235/235 [==============================] - 3s 11ms/step - loss: 0.6908 - val_loss: 0.6907
313/313 [==============================] - 1s 2ms/step
```

Question- 4:
Code:



4.Plot loss and accuracy using the history object

```python
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'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=256,
                    shuffle=True,
                    validation_data=(x_test, y_test))

# Plotting the training and validation loss
plt.figure(figsize=(10, 5))

# Plotting training and validation accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
```

Explanation:

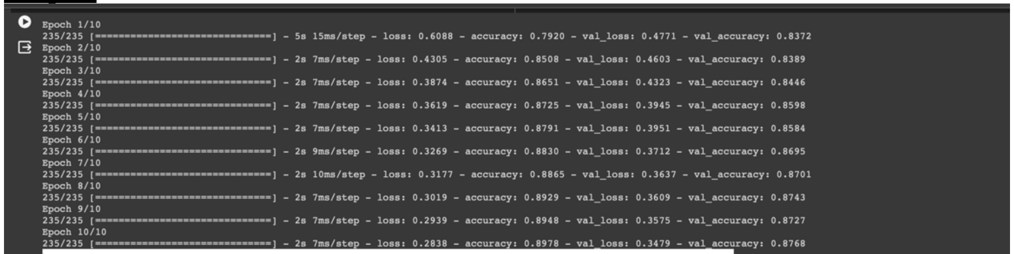
1. Data Preparation: The Fashion MNIST dataset, consisting of 28x28 grayscale images of fashion items, is loaded, normalized, and flattened to prepare for training. Labels are one-hot encoded to fit the classification model.

2. Model Architecture: A basic neural network with an input layer, a hidden layer of 128 neurons with ReLU activation, and an output classification layer of 10 neurons (one for each class) with softmax activation is defined. This architecture is suitable for multi-class classification tasks.

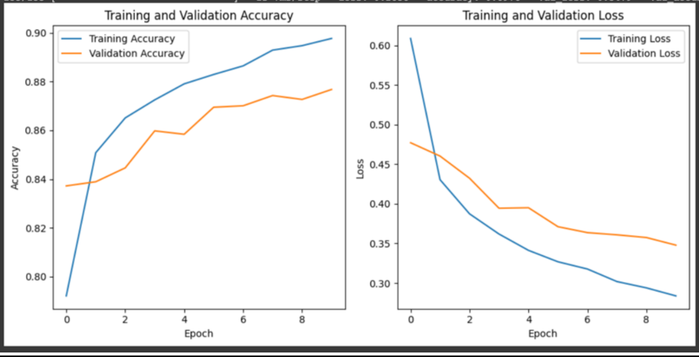
3. Compilation and Training: The model is compiled with the Adam optimizer and categorical crossentropy loss, which are standard choices for classification tasks. It also tracks accuracy as a performance metric. The model is trained for 10 epochs with a batch size of 256, using both training and validation datasets to monitor performance.

4. Performance Visualization: After training, the code plots the training and validation accuracy and loss over epochs. This visualization helps in understanding how well the model is learning and generalizing to unseen data, indicated by its performance on the validation set.

Output:



```
Epoch 1/10
235/235 [==============================] - 5s 15ms/step - loss: 0.6088 - accuracy: 0.7920 - val_loss: 0.4771 - val_accuracy: 0.8372
Epoch 2/10
235/235 [==============================] - 2s 7ms/step - loss: 0.4305 - accuracy: 0.8508 - val_loss: 0.4603 - val_accuracy: 0.8389
Epoch 3/10
235/235 [==============================] - 2s 7ms/step - loss: 0.3874 - accuracy: 0.8651 - val_loss: 0.4323 - val_accuracy: 0.8446
Epoch 4/10
235/235 [==============================] - 2s 7ms/step - loss: 0.3619 - accuracy: 0.8725 - val_loss: 0.3945 - val_accuracy: 0.8598
Epoch 5/10
235/235 [==============================] - 2s 7ms/step - loss: 0.3413 - accuracy: 0.8791 - val_loss: 0.3951 - val_accuracy: 0.8584
Epoch 6/10
235/235 [==============================] - 2s 9ms/step - loss: 0.3269 - accuracy: 0.8830 - val_loss: 0.3712 - val_accuracy: 0.8695
Epoch 7/10
235/235 [==============================] - 2s 10ms/step - loss: 0.3177 - accuracy: 0.8865 - val_loss: 0.3637 - val_accuracy: 0.8701
Epoch 8/10
235/235 [==============================] - 2s 7ms/step - loss: 0.3019 - accuracy: 0.8929 - val_loss: 0.3609 - val_accuracy: 0.8743
Epoch 9/10
235/235 [==============================] - 2s 7ms/step - loss: 0.2939 - accuracy: 0.8948 - val_loss: 0.3575 - val_accuracy: 0.8727
Epoch 10/10
235/235 [==============================] - 2s 7ms/step - loss: 0.2838 - accuracy: 0.8978 - val_loss: 0.3479 - val_accuracy: 0.8768
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
Training and Validation Loss
Training Loss
Validation Loss
Loss
Epoch