Name:konyala sai vardhan reddy

Student id:700745733

Github link: https://github.com/vardhan141/icp4deeplearning

Video link:
https://drive.google.com/file/d/1qK2hDV52YyH7hSWD_tv4s1bOmuZ44Gya/view?usp=sharing

Icp4

Given sample code for autoencoder in ppt

```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')
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:])))
```

```python
autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/5
235/235 [==============================] - 7s 9ms/step - loss: 0.6956 - val_loss: 0.6955
Epoch 2/5
235/235 [==============================] - 2s 8ms/step - loss: 0.6954 - val_loss: 0.6953
Epoch 3/5
235/235 [==============================] - 2s 9ms/step - loss: 0.6952 - val_loss: 0.6951
Epoch 4/5
235/235 [==============================] - 1s 4ms/step - loss: 0.6950 - val_loss: 0.6949
Epoch 5/5
235/235 [==============================] - 1s 5ms/step - loss: 0.6948 - val_loss: 0.6947
<keras.callbacks.History at 0x7deee7f3add0>
```

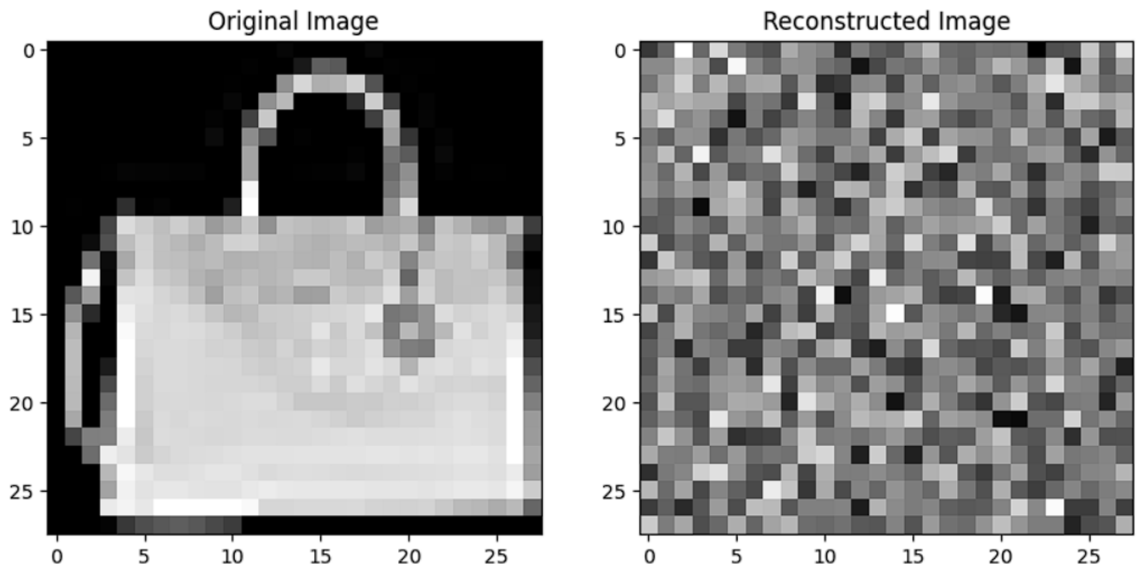
Adding one more hidden layer to encoder

```python
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)
# "encoded" is the encoded representation of the input
encoded1 = Dense(encoding_dim1, activation='relu')(input_img)
#adding one more hidden layer
encoded2 = Dense(encoding_dim2, activation='relu')(encoded1)
#"decoded" is the lossy reconstruction of the input
decoded1 = Dense(encoding_dim1, activation='relu')(encoded2)
#adding one more hidden layer
decoded2 = Dense(input_shape[0], activation='sigmoid')(decoded1)
autoencoder = Model(input_img, decoded2)
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.
```

```python
# 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 = 30

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

# Reshape the reconstructed image
reconstructed_img = decoded_imgs[idx].reshape(28, 28)
```

```python
# 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()
```

```
Epoch 19/20
235/235 [==============================] - 1s 5ms/step - loss: 0.6927 - val_loss: 0.6927
Epoch 20/20
235/235 [==============================] - 1s 5ms/step - loss: 0.6926 - val_loss: 0.6926
313/313 [==============================] - 1s 2ms/step
```



Here we plotted the 30th image from test dataset and printed original image and reconstructed message.

Repeat the question 2 on the denoisening autoencoder

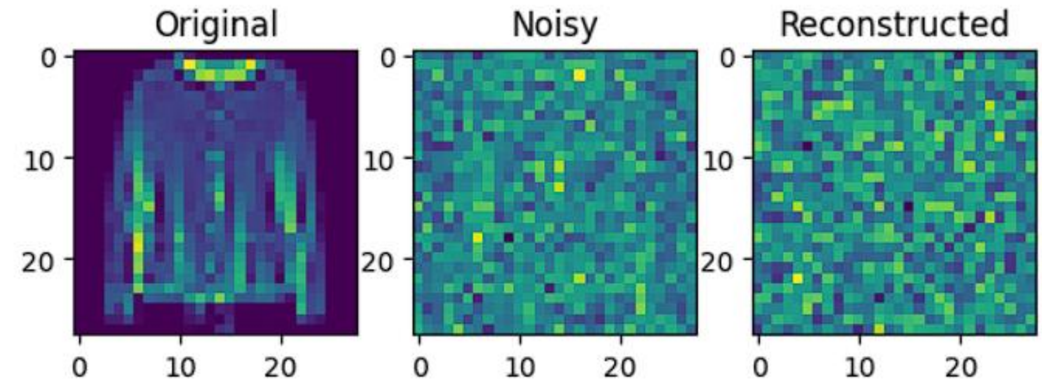
```
from keras.layers import Input, Dense
from keras.models import Model

# 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', 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:])))
```

```python
#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))
```

```
Epoch 1/10
235/235 [==============================] - 3s 8ms/step - loss: 0.6989 - accuracy: 9.0000e-04 - val_loss: 0.6987 - val_accuracy: 0.0010
Epoch 2/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6985 - accuracy: 8.6667e-04 - val_loss: 0.6984 - val_accuracy: 0.0012
Epoch 3/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6982 - accuracy: 9.0000e-04 - val_loss: 0.6980 - val_accuracy: 0.0011
Epoch 4/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6978 - accuracy: 9.0000e-04 - val_loss: 0.6977 - val_accuracy: 0.0012
Epoch 5/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6975 - accuracy: 9.1667e-04 - val_loss: 0.6974 - val_accuracy: 0.0012
Epoch 6/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6972 - accuracy: 9.3333e-04 - val_loss: 0.6971 - val_accuracy: 0.0012
Epoch 7/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6969 - accuracy: 8.8333e-04 - val_loss: 0.6968 - val_accuracy: 0.0012
Epoch 8/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6966 - accuracy: 8.6667e-04 - val_loss: 0.6965 - val_accuracy: 0.0013
Epoch 9/10
235/235 [==============================] - 1s 5ms/step - loss: 0.6963 - accuracy: 9.0000e-04 - val_loss: 0.6962 - val_accuracy: 0.0013
Epoch 10/10
235/235 [==============================] - 1s 6ms/step - loss: 0.6960 - accuracy: 9.1667e-04 - val_loss: 0.6959 - val_accuracy: 0.0013
```

```python
import matplotlib.pyplot as plt

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

# Select one image to display
img_to_display = 25

# 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()
```

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



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()
```