ICP – 6

Student name – Venkata sai prasad

Student id – 700755524

Github link - https://github.com/venkat137222/week-8---ICP6

Video link – https://drive.google.com/file/d/1jCsDmp4kppSUglpgZuMinPDATtB4-0k7/view?usp=sharing

1. 1. Add one more hidden layer to autoencoder

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
[1] 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,))
```

Adding one more hidden layer

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
[2] # "encoded" is the encoded representation of the input
    encoded = Dense(encoding_dim, activation='relu')(input_img)
    encoded2 = Dense(encoding_dim2, activation='relu')(encoded)

    # "decoded" is the lossy reconstruction of the input
    decoded = Dense(784, activation='sigmoid')(encoded)
    decoded2 = Dense(784, activation='sigmoid')(decoded)
```

Working with a basic autoencoder model in keras for image reconstruction using the fashion MNIST dataset and code setup for unsupervised learning, where we are training an autoencoder to encode and then decode the input data.

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
# 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))
```

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.

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

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

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

    # Choosing an index to a test image for visualizing
    idx = 10

    # Reshaping the test image
    test_image = x_test[idx].reshape(28, 28)

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

    # Plotting the original and reconstructed images side by side
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(test_image, cmap='Blues_r')
    plt.title('Original Image')
    plt.subplot(1, 2, 2)
    plt.imshow(reconstructed_image, cmap='Blues_r')
    plt.title('Reconstructed Image')
    plt.show()
```

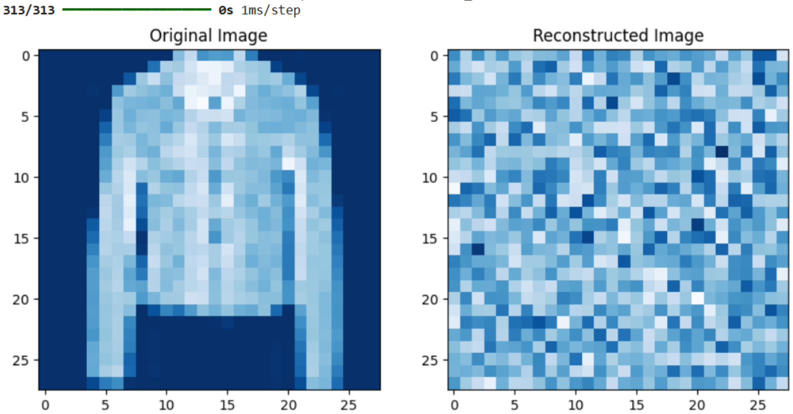
Output:

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
Epoch 1/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.6957 - val_loss: 0.6955
Epoch 2/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.6954 - val_loss: 0.6951
Epoch 3/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.6951 - val_loss: 0.6948
Epoch 4/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - loss: 0.6947 - val_loss: 0.6945
Epoch 5/5
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - loss: 0.6944 - val_loss: 0.6942
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```



3. Repeat the question 2 on the denoisening autoencoder

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
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 the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
```

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
# 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:])))
```

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
#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))
```

Output :

NNDL ICP-6-700755524

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
Epoch 1/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 9.2768e-04 - loss: 0.6954 - val_accuracy: 0.0014 - val_loss: 0.6954
Epoch 2/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.0011 - loss: 0.6953 - val_accuracy: 0.0014 - val_loss: 0.6953
Epoch 3/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 8.9726e-04 - loss: 0.6952 - val_accuracy: 0.0015 - val_loss: 0.6952
Epoch 4/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.0011 - loss: 0.6951 - val_accuracy: 0.0015 - val_loss: 0.6951
Epoch 5/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - accuracy: 0.0010 - loss: 0.6950 - val_accuracy: 0.0015 - val_loss: 0.6950
Epoch 6/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 6s 14ms/step - accuracy: 0.0010 - loss: 0.6949 - val_accuracy: 0.0015 - val_loss: 0.6949
Epoch 7/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.0010 - loss: 0.6948 - val_accuracy: 0.0014 - val_loss: 0.6948
Epoch 8/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.0011 - loss: 0.6947 - val_accuracy: 0.0014 - val_loss: 0.6947
Epoch 9/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 0.0012 - loss: 0.6946 - val_accuracy: 0.0013 - val_loss: 0.6946
Epoch 10/10
235/235 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 0.0010 - loss: 0.6945 - val_accuracy: 0.0013 - val_loss: 0.6946
```

Visualizing the test data using Matplotlib

NNDL ICP-6-700755524

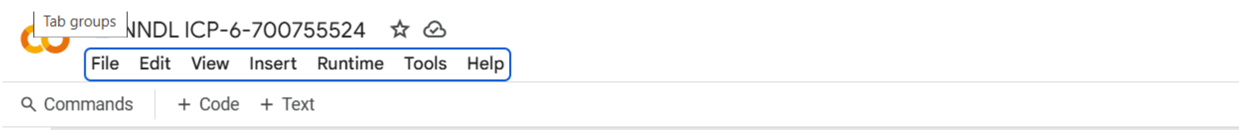
File Edit View Insert Runtime Tools Help

Commands + Code + Text

```python
import matplotlib.pyplot as plt

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

# Select one image to display
img_to_display = 0

# 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 Image')

plt.subplot(1, 3, 2)
plt.imshow(x_test_noisy[img_to_display].reshape(28, 28))
plt.title('Noisy Image')

plt.subplot(1, 3, 3)
plt.imshow(reconstructed_images[img_to_display].reshape(28, 28))
plt.title('Reconstructed Image')

plt.show()
```

Output:

4. 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()
```

Output: