NEURAL NETWORK DEEP LEARNING
ICP 8
700755861
SREEJA MADHAGONI

GitHub:

Repository URL for the source code:
https://github.com/SreejaMadhagoni/NNDL/tree/main/ICP%208

Video Link:

https://drive.google.com/file/d/1KPbTuzZf-uYAcDM8vMYC2It0vqC5Y_TQ/view?usp=share_link

Autoencoder without hidden layer.

```
In [ ]: encoding_dim = 64

        input_img = Input(shape=(784,))

        encoded = Dense(encoding_dim, activation='relu')(input_img)
        decoded = Dense(784, activation='sigmoid')(encoded)
        autoencoder = Model(input_img, decoded)
        encoder = Model(input_img, encoded)

        encoded_input = Input(shape=(encoding_dim,))
        decoder_layer = autoencoder.layers[-1]
        decoder = Model(encoded_input, decoder_layer(encoded_input))

        autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

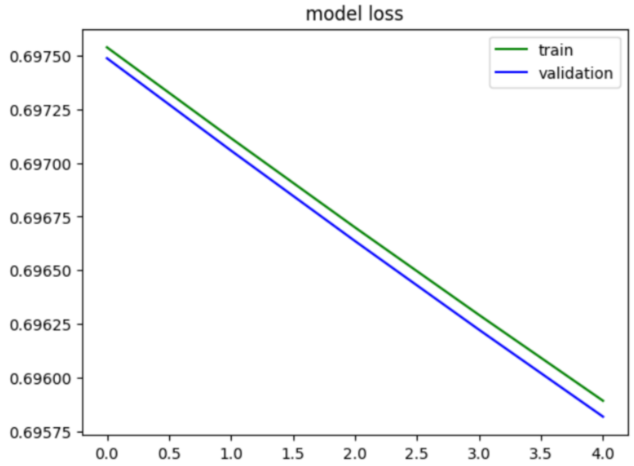
```
In [ ]: (x_train, _), (x_test, _) = mnist.load_data()
        x_train = x_train.astype('float32') / 255.
        x_test = x_test.astype('float32') / 255.
        x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
        x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
        history = autoencoder.fit(x_train, x_train,
                        epochs=5,
                        batch_size=256,
                        shuffle=True,
                        validation_data=(x_test, x_test))

        encoded_imgs = encoder.predict(x_test)
        decoded_imgs = decoder.predict(encoded_imgs)
```

```
Epoch 1/5
235/235 [==============================] - 8s 30ms/step - loss: 0.6975 - val_loss: 0.6975
Epoch 2/5
235/235 [==============================] - 7s 29ms/step - loss: 0.6971 - val_loss: 0.6971
Epoch 3/5
235/235 [==============================] - 6s 24ms/step - loss: 0.6967 - val_loss: 0.6966
Epoch 4/5
235/235 [==============================] - 4s 17ms/step - loss: 0.6963 - val_loss: 0.6962
Epoch 5/5
235/235 [==============================] - 3s 14ms/step - loss: 0.6959 - val_loss: 0.6958
313/313 [==============================] - 1s 1ms/step
313/313 [==============================] - 1s 2ms/step
```

Graph for validation and training.

```
In [ ]: # graph
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], color="green")
plt.plot(history.history['val_loss'], color="blue")
plt.title('model loss')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Autoencoder with hidden layer.

```
In [ ]: input_size = 784
hidden_size = 128
code_size = 32

input_img = Input(shape=(input_size,))
hidden_1 = Dense(hidden_size, activation='relu')(input_img)
code = Dense(code_size, activation='relu')(hidden_1)
hidden_2 = Dense(hidden_size, activation='relu')(code)
output_img = Dense(input_size, activation='sigmoid')(hidden_2)

autoencoder = Model(input_img, output_img)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

```
In [ ]: (x_train, _), (x_test, _) = 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:])))
history = autoencoder.fit(x_train, x_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

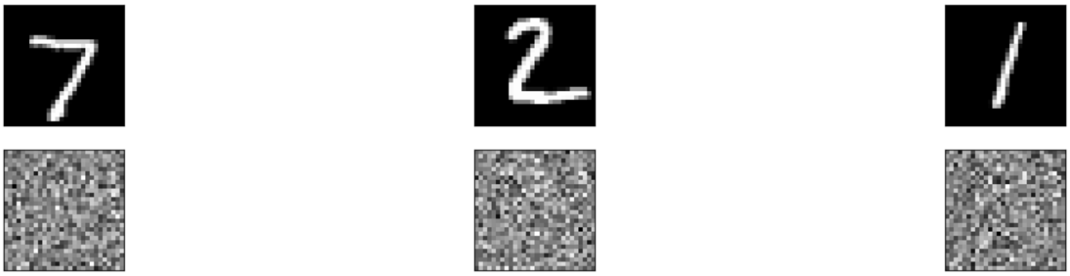
```
Epoch 1/5
235/235 [==============================] - 6s 23ms/step - loss: 0.2308 - val_loss: 0.1491
Epoch 2/5
235/235 [==============================] - 4s 17ms/step - loss: 0.1342 - val_loss: 0.1221
Epoch 3/5
235/235 [==============================] - 4s 18ms/step - loss: 0.1169 - val_loss: 0.1103
Epoch 4/5
235/235 [==============================] - 5s 20ms/step - loss: 0.1085 - val_loss: 0.1046
Epoch 5/5
235/235 [==============================] - 4s 17ms/step - loss: 0.1035 - val_loss: 0.1001
```

Printing original and reconstructed images

```
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 + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

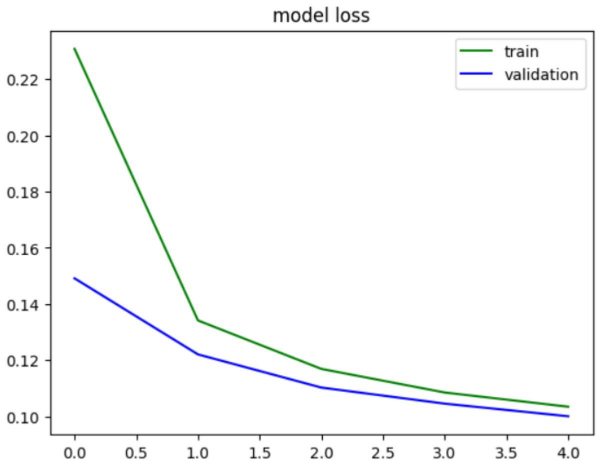
```
313/313 [==============================] - 0s 1ms/step
313/313 [==============================] - 1s 2ms/step
```



Printing model loss after adding hidden layer.

In [ ]:

```
# graph
plt.plot(history.history['loss'], color="green")
plt.plot(history.history['val_loss'], color="blue")
plt.title('model loss')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Do the prediction on the test data and then visualize one of the reconstructed versions of that test data. Also, visualize the same test data before reconstruction using Matplotlib

3. Use denoising autoencoder, to reconstruct the input,

4. Plot loss and accuracy using the history object.

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

# Scales the training and test data to range between 0 and 1.
max_value = float(x_train.max())
x_train = x_train.astype('float32') / max_value
x_test = x_test.astype('float32') / max_value
x_train.shape, x_test.shape
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:])))

(x_train.shape, x_test.shape)
input_dim = x_train.shape[1]
encoding_dim = 64

compression_factor = float(input_dim) / encoding_dim
print("Compression factor: %s" % compression_factor)

autoencoder = Sequential()
autoencoder.add(
    Dense(encoding_dim, input_shape=(input_dim,), activation='relu')
)
autoencoder.add(
    Dense(input_dim, activation='sigmoid')
)

autoencoder.summary()
input_img = Input(shape=(input_dim,))
encoder_layer = autoencoder.layers[0]
encoder = Model(input_img, encoder_layer(input_img))
```

```
encoder.summary()
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
history = autoencoder.fit(x_train, x_train,
                          epochs=5,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(x_test, x_test))

num_images = 5
np.random.seed(42)
random_test_images = np.random.randint(x_test.shape[0], size=num_images)

noise = np.random.normal(loc=0.1, scale=0.1, size=x_test.shape)
noised_images = x_test + noise
encoded_imgs = encoder.predict(noised_images)
decoded_imgs = autoencoder.predict(noised_images)
```

```
Compression factor: 12.25
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 64)                50240

 dense_7 (Dense)             (None, 784)               50960

=================================================================
Total params: 101,200
Trainable params: 101,200
Non-trainable params: 0
_________________________________________________________________
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 784)]             0

 dense_6 (Dense)             (None, 64)                50240

=================================================================
Total params: 50,240
Trainable params: 50,240
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
235/235 [==============================] - 4s 14ms/step - loss: 0.2444 - val_loss: 0.1629
Epoch 2/5
235/235 [==============================] - 4s 17ms/step - loss: 0.1450 - val_loss: 0.1287
Epoch 3/5
235/235 [==============================] - 3s 13ms/step - loss: 0.1202 - val_loss: 0.1103
Epoch 4/5
235/235 [==============================] - 3s 13ms/step - loss: 0.1054 - val_loss: 0.0986
Epoch 5/5
235/235 [==============================] - 3s 13ms/step - loss: 0.0957 - val_loss: 0.0908
313/313 [==============================] - 1s 1ms/step
313/313 [==============================] - 1s 2ms/step
```