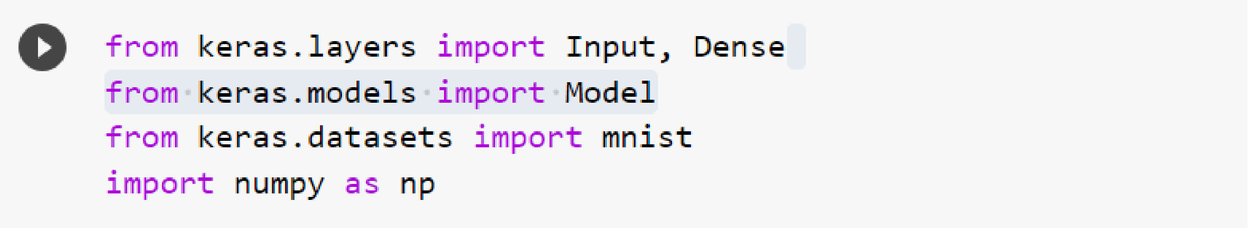
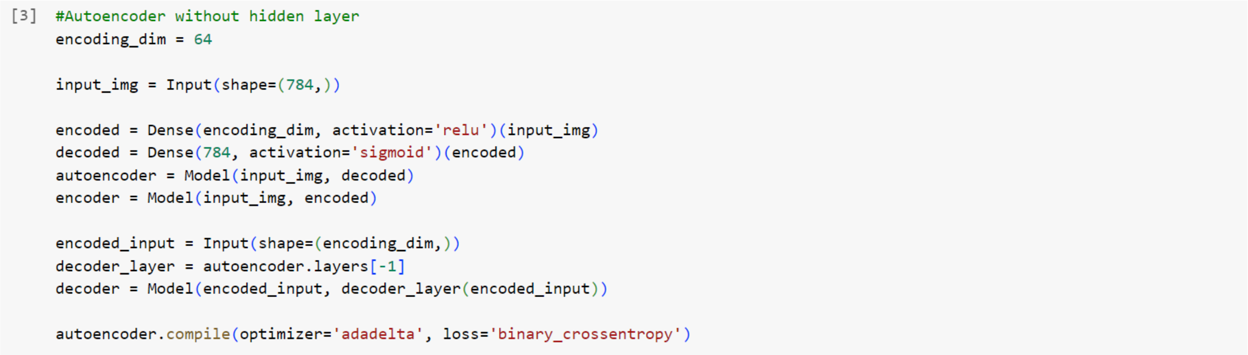
Fall 2023: CS5720 Neural Networks & Deep Learning - ICP-8
Assignment-8
NAME:RAJYALAKSHMI GOTTIPATI
STUDENT ID:700745186

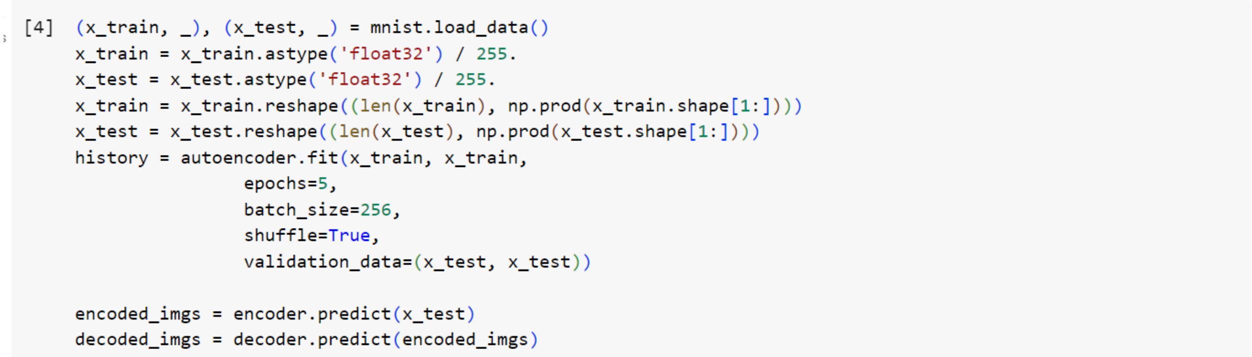
Github link: https://github.com/rajigottipati/icp_8.git
Videolink:https://drive.google.com/file/d/10iPCWPHfiS8dJzQjwuJvVTgY5KqRK4QT/view?usp=share_link



```
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np
```



```
[3] #Autoencoder without hidden layer
    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')
```

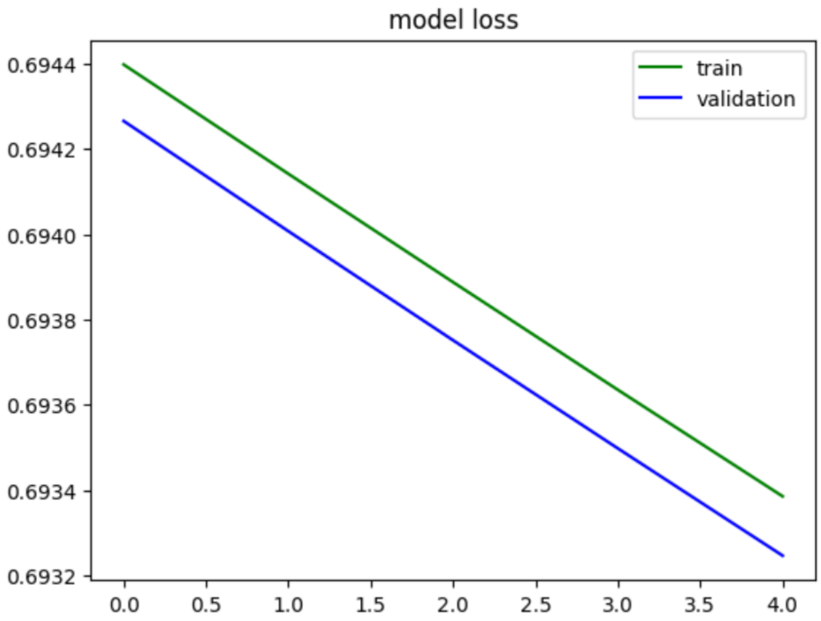


```
[4] (x_train, _), (x_test, _) = mnist.load_data()
    x_train = x_train.astype('float32') / 255.
    x_test = x_test.astype('float32') / 255.
    x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
    x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
    history = 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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/5
235/235 [==============================] - 10s 35ms/step - loss: 0.6944 - val_loss: 0.6943
Epoch 2/5
235/235 [==============================] - 4s 18ms/step - loss: 0.6941 - val_loss: 0.6940
Epoch 3/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6939 - val_loss: 0.6938
Epoch 4/5
235/235 [==============================] - 5s 19ms/step - loss: 0.6936 - val_loss: 0.6935
Epoch 5/5
235/235 [==============================] - 4s 15ms/step - loss: 0.6934 - val_loss: 0.6932
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 3ms/step
```

[5]

```
# 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()
```



[7]

```
#Autoencoder with hidden layer

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')
```

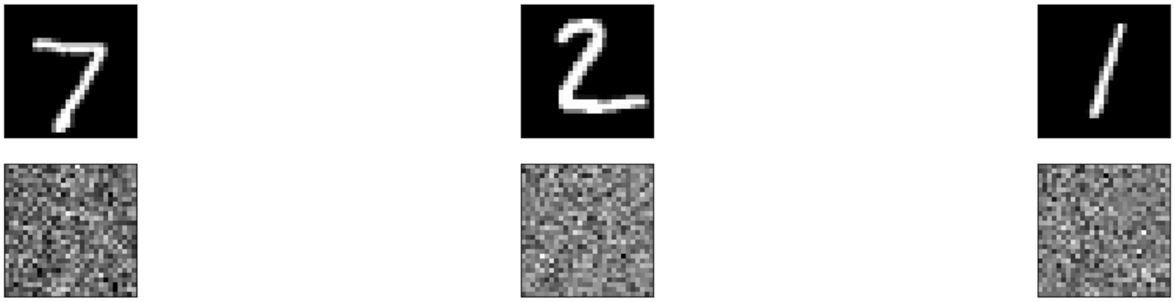
```
(x_train, _), (x_test, _) = mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
history = autoencoder.fit(x_train,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

```
Epoch 1/5
235/235 [==============================] - 7s 24ms/step - loss: 0.2307 - val_loss: 0.1505
Epoch 2/5
235/235 [==============================] - 4s 19ms/step - loss: 0.1350 - val_loss: 0.1214
Epoch 3/5
235/235 [==============================] - 5s 20ms/step - loss: 0.1173 - val_loss: 0.1111
Epoch 4/5
235/235 [==============================] - 5s 22ms/step - loss: 0.1096 - val_loss: 0.1054
Epoch 5/5
235/235 [==============================] - 6s 27ms/step - loss: 0.1051 - val_loss: 0.1024
```

[9]

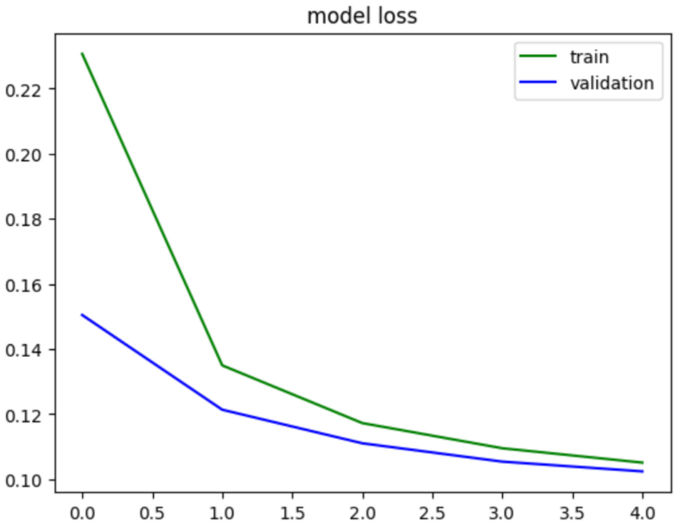
```
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)

import matplotlib.pyplot as plt

n = 3
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 + 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 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
```

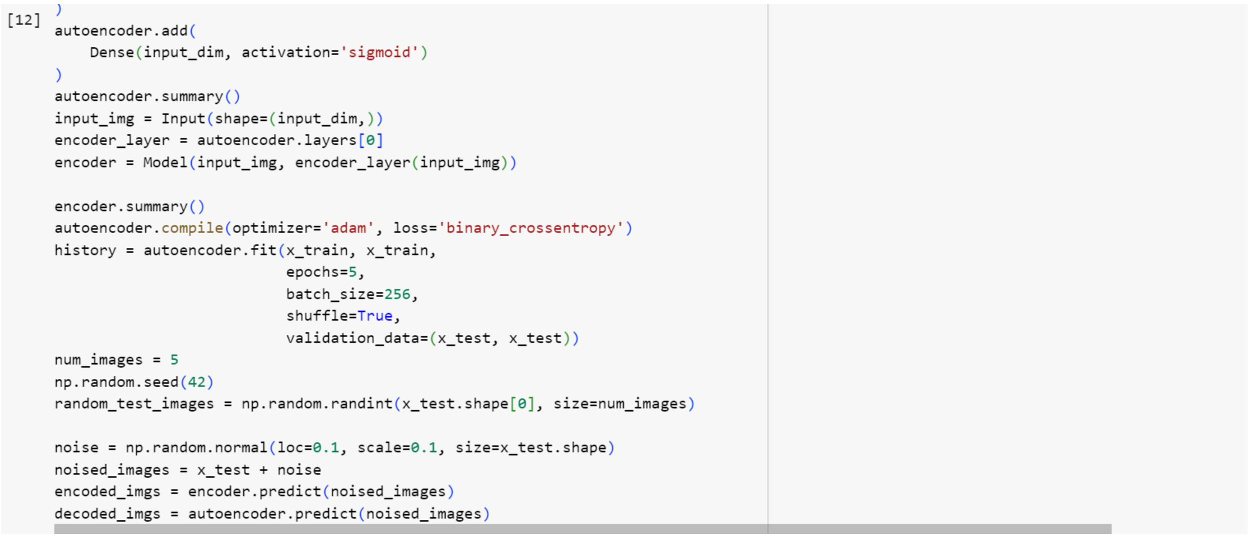


[10]

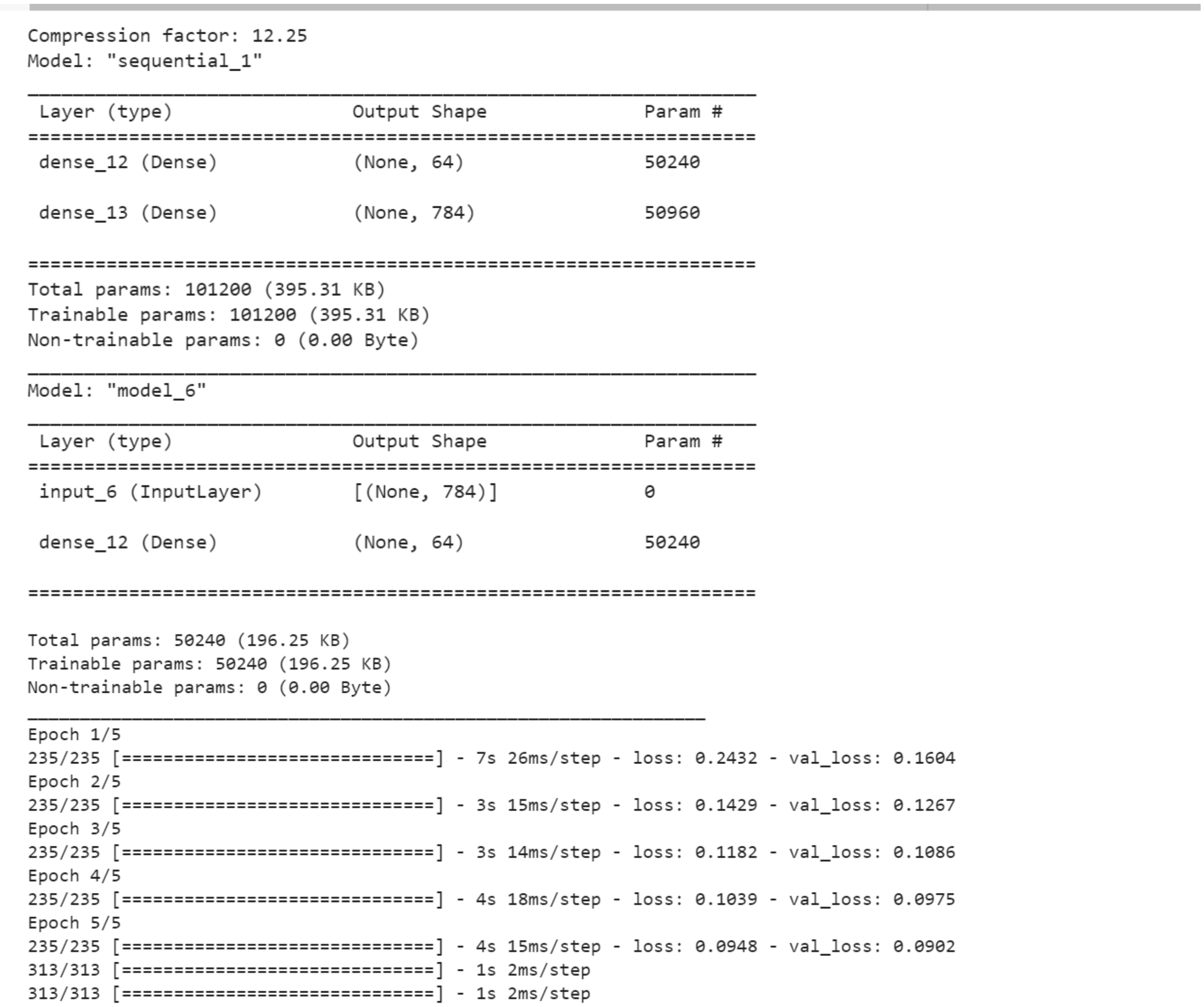
```
# graph
plt.plot(history.history['loss'], color="green")
plt.plot(history.history['val_loss'], color="blue")
plt.title('model loss')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



```
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')
```



```
[12]  )
      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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_12 (Dense)            (None, 64)                50240

 dense_13 (Dense)            (None, 784)               50960

=================================================================
Total params: 101200 (395.31 KB)
Trainable params: 101200 (395.31 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 784)]             0

 dense_12 (Dense)            (None, 64)                50240

=================================================================

Total params: 50240 (196.25 KB)
Trainable params: 50240 (196.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
235/235 [==============================] - 7s 26ms/step - loss: 0.2432 - val_loss: 0.1604
Epoch 2/5
235/235 [==============================] - 3s 15ms/step - loss: 0.1429 - val_loss: 0.1267
Epoch 3/5
235/235 [==============================] - 3s 14ms/step - loss: 0.1182 - val_loss: 0.1086
Epoch 4/5
235/235 [==============================] - 4s 18ms/step - loss: 0.1039 - val_loss: 0.0975
Epoch 5/5
235/235 [==============================] - 4s 15ms/step - loss: 0.0948 - val_loss: 0.0902
313/313 [==============================] - 1s 2ms/step
313/313 [==============================] - 1s 2ms/step
```