nn-autoencoder-image-compression

January 31, 2024

0.1 Autoencoder

An autoencoder is an unsupervised learning technique for neural networks that learns efficient data representations (encoding) by training the network to ignore signal "noise." Autoencoders can be used for image denoising, image compression, and, in some cases, even generation of image data.

0.2 Flow of Autoencoder

Input Image -> Encoder -> Compressed Representation -> Decoder -> Reconstruct Input Image

0.3 Import Modules

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
from keras import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.datasets import mnist
```

0.4 Load the Dataset

[3]:
```
(x_train, _), (x_test, _) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

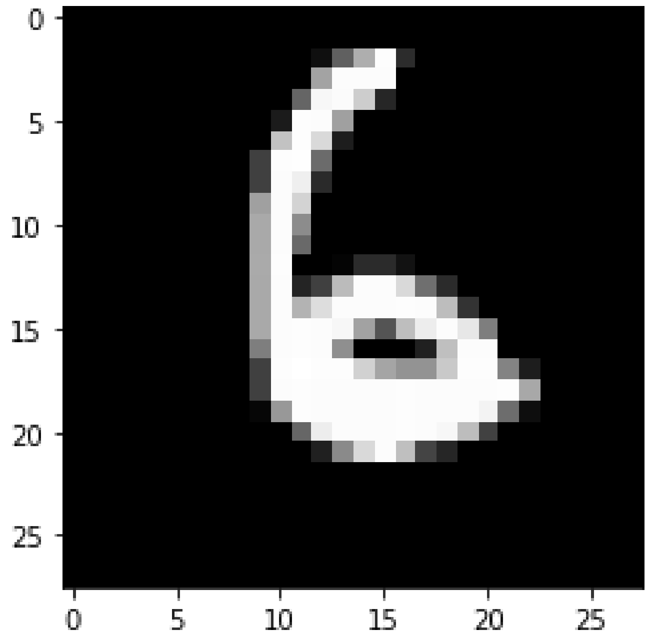
[4]:
```
# normalize the image data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

[6]:
```
# reshape in the input data for the model
x_train = x_train.reshape(len(x_train), 28, 28, 1)
x_test = x_test.reshape(len(x_test), 28, 28, 1)
x_test.shape
```

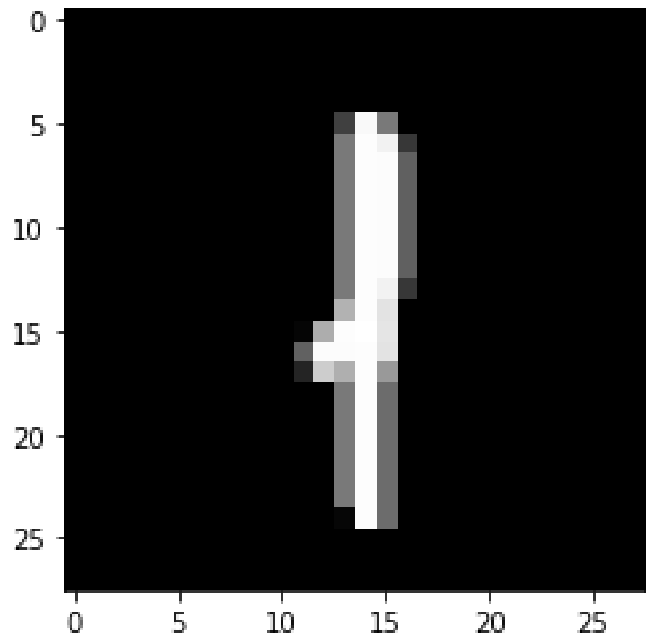
[6]:
```
(10000, 28, 28, 1)
```

0.5 Exploratory Data Analysis

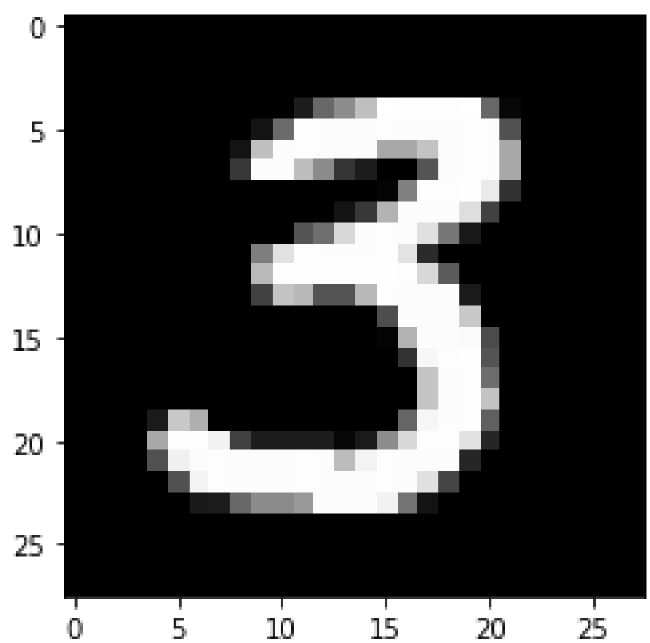
[15]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```



[16]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```



[18]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```



[ ]:

```

```

0.6 Model Creation

[20]:

```
model = Sequential([
                    # encoder network
                    Conv2D(32, 3, activation='relu', padding='same',␣
 ↪input_shape=(28, 28, 1)),
                    MaxPooling2D(2, padding='same'),
                    Conv2D(16, 3, activation='relu', padding='same'),
                    MaxPooling2D(2, padding='same'),
                    # decoder network
                    Conv2D(16, 3, activation='relu', padding='same'),
                    UpSampling2D(2),
                    Conv2D(32, 3, activation='relu', padding='same'),
                    UpSampling2D(2),
                    # output layer
                    Conv2D(1, 3, activation='sigmoid', padding='same')
])

model.compile(optimizer='adam', loss='binary_crossentropy')
model.summary()
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_16 (Conv2D)          (None, 14, 14, 16)        4624

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 16)         0
 2D)

 conv2d_17 (Conv2D)          (None, 7, 7, 16)          2320

 up_sampling2d_6 (UpSampling  (None, 14, 14, 16)       0
 2D)

 conv2d_18 (Conv2D)          (None, 14, 14, 32)        4640

 up_sampling2d_7 (UpSampling  (None, 28, 28, 32)       0
 2D)
```

```
 conv2d_19 (Conv2D)          (None, 28, 28, 1)         289

=================================================================
Total params: 12,193
Trainable params: 12,193
Non-trainable params: 0
_________________________________________________________________
```

[21]:
```
# train the model
model.fit(x_train, x_train, epochs=20, batch_size=256, validation_data=(x_test,
 ↪x_test))
```

```
Epoch 1/20
235/235 [==============================] - 15s 23ms/step - loss: 0.1729 -
val_loss: 0.0876
Epoch 2/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0837 -
val_loss: 0.0793
Epoch 3/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0785 -
val_loss: 0.0761
Epoch 4/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0759 -
val_loss: 0.0743
Epoch 5/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0744 -
val_loss: 0.0730
Epoch 6/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0733 -
val_loss: 0.0727
Epoch 7/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0725 -
val_loss: 0.0716
Epoch 8/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0718 -
val_loss: 0.0708
Epoch 9/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0712 -
val_loss: 0.0704
Epoch 10/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0708 -
val_loss: 0.0699
Epoch 11/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0704 -
val_loss: 0.0696
Epoch 12/20
235/235 [==============================] - 5s 20ms/step - loss: 0.0700 -
```

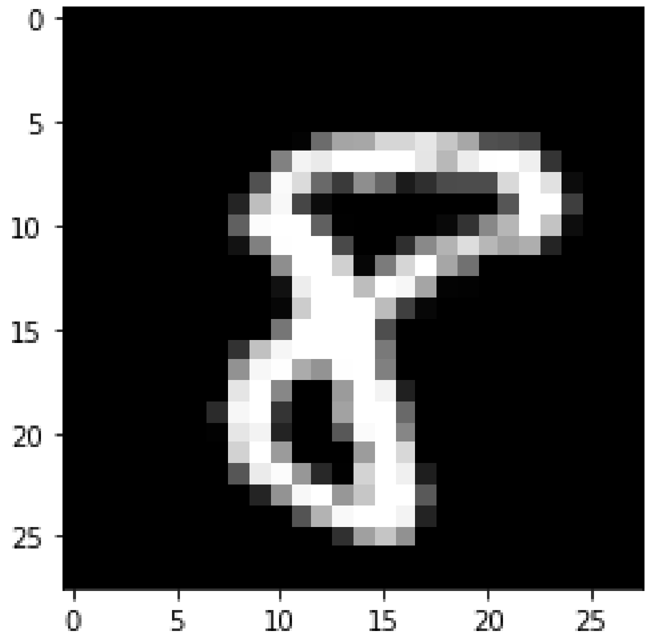
```
val_loss: 0.0693
Epoch 13/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0698 -
val_loss: 0.0693
Epoch 14/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0695 -
val_loss: 0.0688
Epoch 15/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0693 -
val_loss: 0.0686
Epoch 16/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0691 -
val_loss: 0.0684
Epoch 17/20
235/235 [==============================] - 5s 22ms/step - loss: 0.0688 -
val_loss: 0.0682
Epoch 18/20
235/235 [==============================] - 5s 21ms/step - loss: 0.0687 -
val_loss: 0.0682
Epoch 19/20
235/235 [==============================] - 5s 20ms/step - loss: 0.0685 -
val_loss: 0.0679
Epoch 20/20
235/235 [==============================] - 5s 20ms/step - loss: 0.0683 -
val_loss: 0.0677
```

[21]:
```
<keras.callbacks.History at 0x7fb87c4e5b50>
```

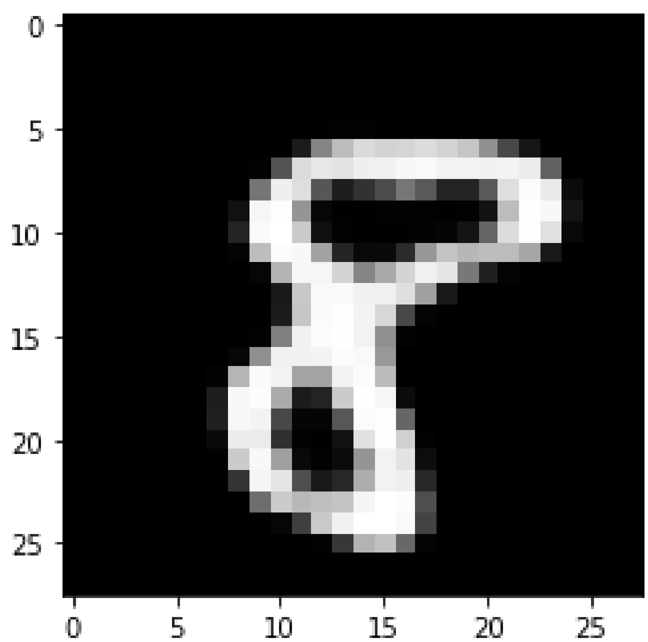
0.7 Visualize the Results

[23]:
```python
# predict the results from model (get compressed images)
pred = model.predict(x_test)
```

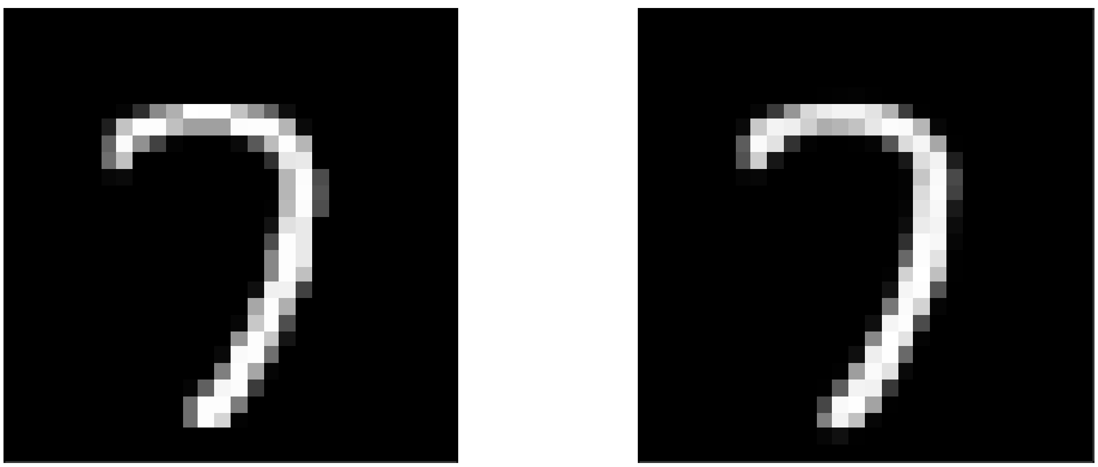
[22]:
```python
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```



```
[24]: # visualize compressed image
plt.imshow(pred[index].reshape(28,28))
plt.gray()
```



[28]:
```
index = np.random.randint(len(x_test))
plt.figure(figsize=(10, 4))
# display original image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display compressed image
ax = plt.subplot(1, 2, 2)
plt.imshow(pred[index].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
plt.show()
```



[29]:
```
index = np.random.randint(len(x_test))
plt.figure(figsize=(10, 4))
# display original image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display compressed image
ax = plt.subplot(1, 2, 2)
plt.imshow(pred[index].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
```

```
plt.show()
```