# cnn-autoencoder-denoising-image

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

Noisy Image -> Encoder -> Compressed Representation -> Decoder -> Reconstruct Clear Image

## 0.3 Import Modules

[1]:
```
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

[2]:
```
(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
```

[3]:
```
# normalize the image data
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
```

[4]:
```
# 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
```

[4]:
```
(10000, 28, 28, 1)
```

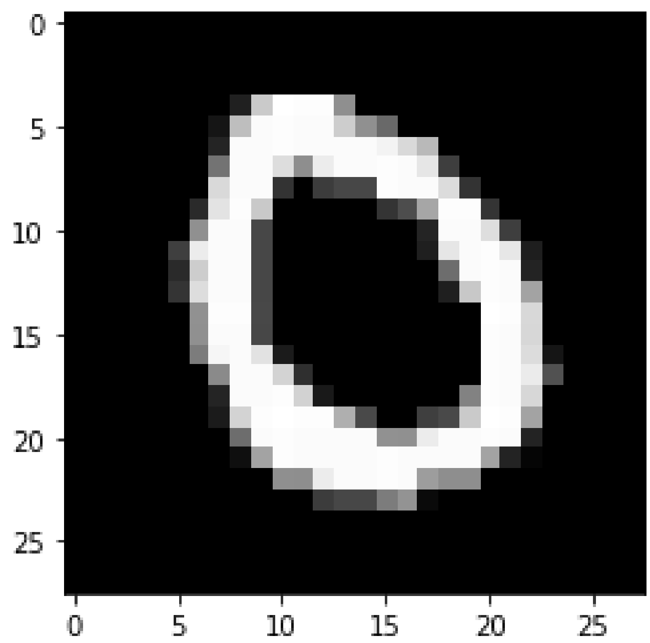
### 0.5 Add Noise to the Image

[5]:
```
# add noise
noise_factor = 0.6
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)
```

[6]:
```
# clip the values in the range of 0-1
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

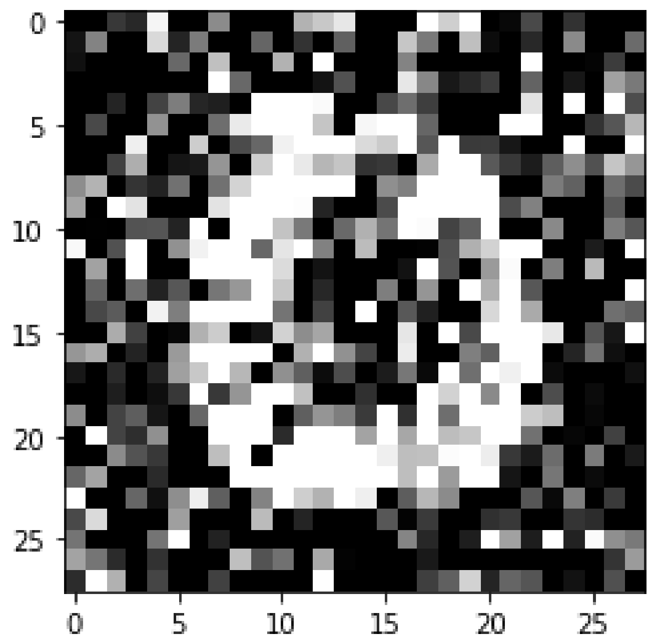
### 0.6 Exploratory Data Analysis

[7]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```



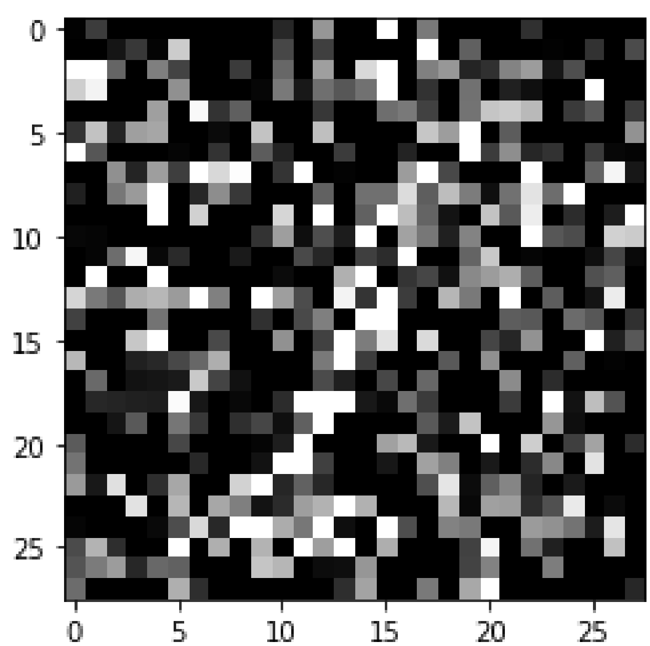

[8]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test_noisy[index].reshape(28,28))
```

```
plt.gray()
```

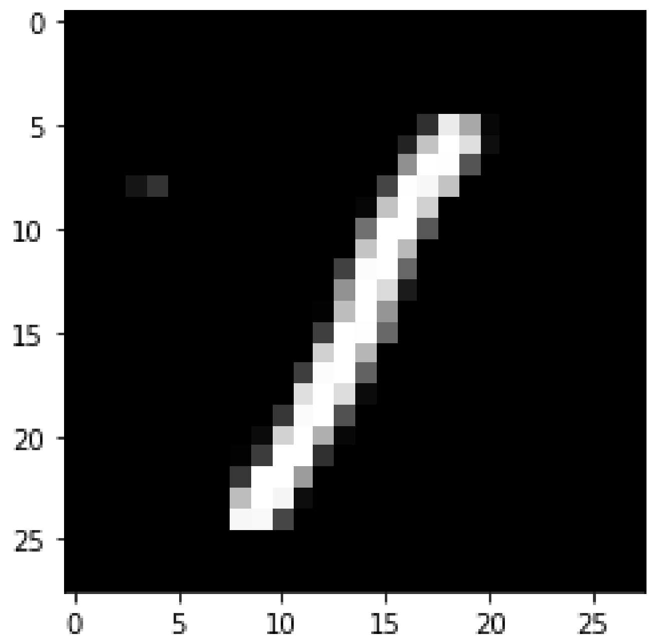


[9]:

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



[10]:
```
plt.imshow(x_test[index].reshape(28,28))
plt.gray()
```




## 0.7 Model Creation

[11]:
```
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"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 16)        4624

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 16)         0
 2D)

 conv2d_2 (Conv2D)           (None, 7, 7, 16)          2320

 up_sampling2d (UpSampling2D  (None, 14, 14, 16)       0
 )

 conv2d_3 (Conv2D)           (None, 14, 14, 32)        4640

 up_sampling2d_1 (UpSampling  (None, 28, 28, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 28, 28, 1)         289

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

[12]:

```
# train the model
model.fit(x_train_noisy, x_train, epochs=20, batch_size=256,
 ↪validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/20
235/235 [==============================] - 14s 12ms/step - loss: 0.2730 -
val_loss: 0.1604
Epoch 2/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1481 -
val_loss: 0.1400
Epoch 3/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1376 -
val_loss: 0.1336
Epoch 4/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1327 -
val_loss: 0.1305
```

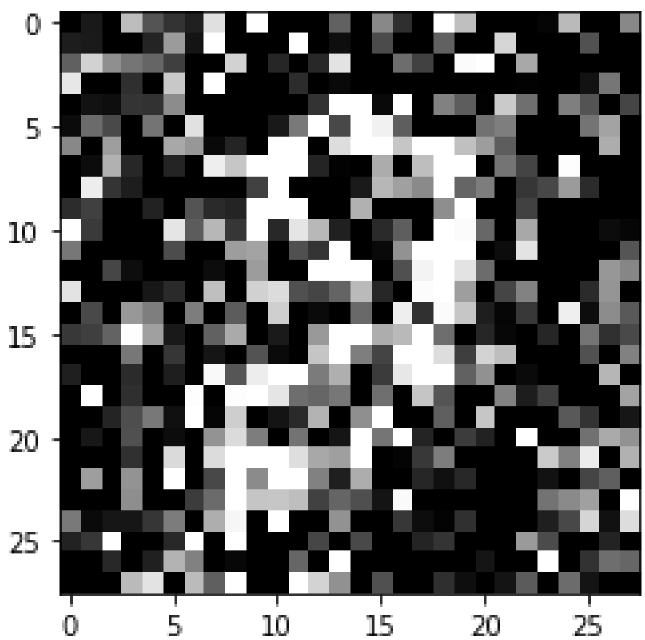
```
Epoch 5/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1298 -
val_loss: 0.1277
Epoch 6/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1276 -
val_loss: 0.1256
Epoch 7/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1256 -
val_loss: 0.1237
Epoch 8/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1240 -
val_loss: 0.1223
Epoch 9/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1227 -
val_loss: 0.1213
Epoch 10/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1217 -
val_loss: 0.1202
Epoch 11/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1209 -
val_loss: 0.1198
Epoch 12/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1200 -
val_loss: 0.1188
Epoch 13/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1193 -
val_loss: 0.1182
Epoch 14/20
235/235 [==============================] - 3s 11ms/step - loss: 0.1187 -
val_loss: 0.1173
Epoch 15/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1181 -
val_loss: 0.1168
Epoch 16/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1175 -
val_loss: 0.1164
Epoch 17/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1170 -
val_loss: 0.1157
Epoch 18/20
235/235 [==============================] - 2s 10ms/step - loss: 0.1164 -
val_loss: 0.1153
Epoch 19/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1162 -
val_loss: 0.1150
Epoch 20/20
235/235 [==============================] - 2s 11ms/step - loss: 0.1158 -
val_loss: 0.1156
```

[12]: <keras.callbacks.History at 0x7fd81036dd90>

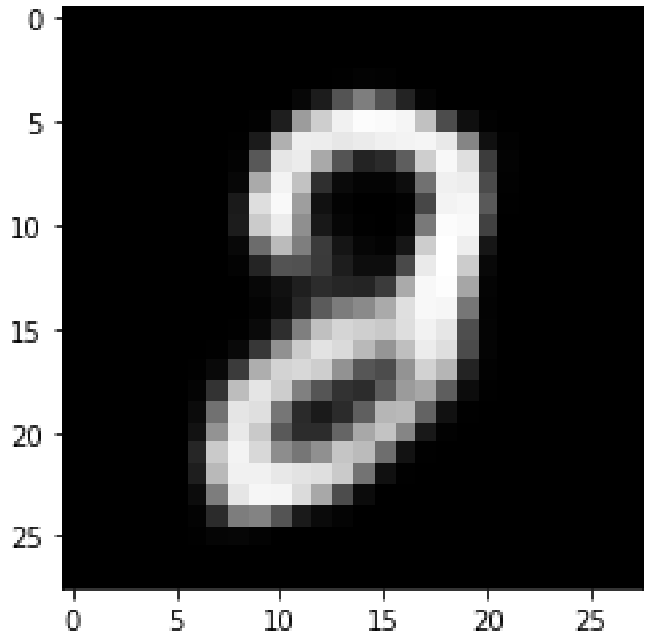
### 0.8 Visualize the Results

[13]:
```
# predict the results from model (get compressed images)
pred = model.predict(x_test_noisy)
```

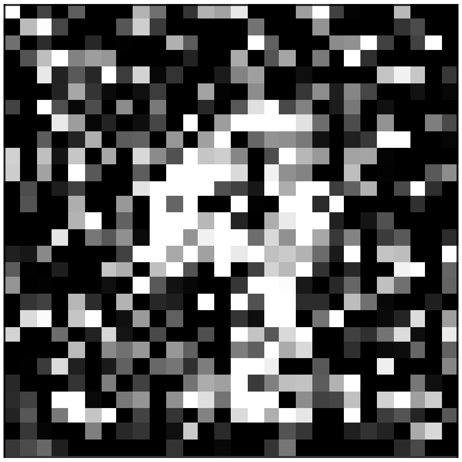
[14]:
```
# randomly select input image
index = np.random.randint(len(x_test))
# plot the image
plt.imshow(x_test_noisy[index].reshape(28,28))
plt.gray()
```

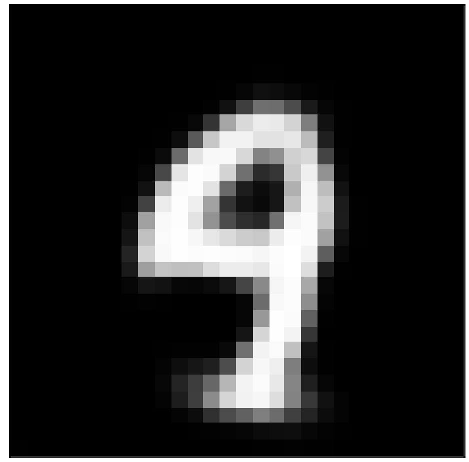



[15]:
```
# visualize compressed image
plt.imshow(pred[index].reshape(28,28))
plt.gray()
```

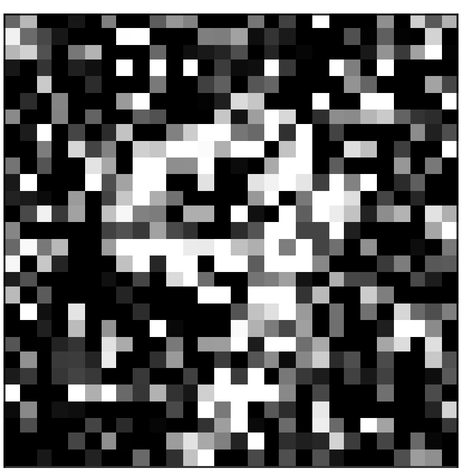


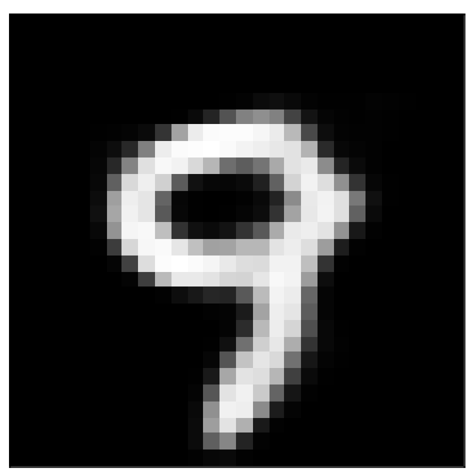

[16]:
```
index = np.random.randint(len(x_test))
plt.figure(figsize=(10, 4))
# display original image
ax = plt.subplot(1, 2, 1)
plt.imshow(x_test_noisy[index].reshape(28,28))
plt.gray()
ax.get_xaxis().set_visible(False)
ax.get_yaxis().set_visible(False)
# display compressed image
ax = plt.subplot(1, 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()
```





[17]:
```
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_noisy[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()
```





```
[ ]:
```