02_convolutional_denoising_autoencoders

September 29, 2021

1 Convolutional & Denoising Autoencoders

The insights from Chapter 17, Convolutional Neural Networks, suggest we incorporate convolutional layers into the autoencoder to extract information characteristic of the grid-like structure of image data.

Source: https://blog.keras.io/building-autoencoders-in-keras.html

1.1 Imports & Settings

[1]:
```
%matplotlib inline

from pathlib import Path
import pandas as pd
import numpy as np
from numpy.random import choice

import tensorflow as tf
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.datasets import fashion_mnist

import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
gpu_devices = tf.config.experimental.list_physical_devices('GPU')
if gpu_devices:
    print('Using GPU')
    tf.config.experimental.set_memory_growth(gpu_devices[0], True)
else:
    print('Using CPU')
```

```
Using CPU
```

[3]:
```
n_classes = 10 # all examples have 10 classes
```

[4]:
```
sns.set_style('white')
cmap = sns.color_palette('Paired', n_classes)
```

[5]:
```
results_path = Path('results', 'fashion_mnist')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.2 Fashion MNIST Data

[6]:
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

[7]:
```
X_train.shape, X_test.shape
```

[7]:
```
((60000, 28, 28), (10000, 28, 28))
```

[8]:
```
class_dict = {0: 'T-shirt/top',
              1: 'Trouser',
              2: 'Pullover',
              3: 'Dress',
              4: 'Coat',
              5: 'Sandal',
              6: 'Shirt',
              7: 'Sneaker',
              8: 'Bag',
              9: 'Ankle boot'}
classes = list(class_dict.keys())
```

1.3 Reshape & normalize Fashion MNIST data

[9]:
```
image_size = 28
```

[10]:
```
def data_prep_conv(x, size=image_size):
    return x.reshape(-1, size, size, 1).astype('float32')/255
```

[11]:
```
X_train_scaled = data_prep_conv(X_train)
X_test_scaled = data_prep_conv(X_test)
```

[12]:
```
X_train_scaled.shape, X_test_scaled.shape
```

[12]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

1.4 Combine training steps into function

[13]:
```
def train_autoencoder(path, model, x_train=X_train_scaled,␣
↪x_test=X_test_scaled):
    callbacks = [EarlyStopping(patience=5, restore_best_weights=True),
                 ModelCheckpoint(filepath=path, save_best_only=True,␣
↪save_weights_only=True)]
    model.fit(x=x_train, y=x_train, epochs=100, validation_split=.1,␣
↪callbacks=callbacks)
    model.load_weights(path)
    mse = model.evaluate(x=x_test, y=x_test)
    return model, mse
```

1.5 Convolutional Autoencoder

We define a three-layer encoder that uses 2D convolutions with 32, 16, and 8 filters, respectively, ReLU activations, and 'same' padding to maintain the input size. The resulting encoding size at the third layer is 4 x 4 x 8 = 128, higher than for the preceding examples:

1.5.1 3-dim input

[14]:
```
input_ = Input(shape=(28, 28, 1), name='Input_3D')
```

1.5.2 Encoding Layers

[15]:
```
x = Conv2D(filters=32,
           kernel_size=(3, 3),
           activation='relu',
           padding='same',
           name='Encoding_Conv_1')(input_)
x = MaxPooling2D(pool_size=(2, 2), padding='same', name='Encoding_Max_1')(x)
x = Conv2D(filters=16,
           kernel_size=(3, 3),
           activation='relu',
           padding='same',
           name='Encoding_Conv_2')(x)
x = MaxPooling2D(pool_size=(2, 2), padding='same', name='Encoding_Max_2')(x)
x = Conv2D(filters=8,
           kernel_size=(3, 3),
           activation='relu',
           padding='same',
           name='Encoding_Conv_3')(x)
encoded_conv = MaxPooling2D(pool_size=(2, 2),
                            padding='same',
                            name='Encoding_Max_3')(x)
```

We also define a matching decoder that reverses the number of filters and uses 2D upsampling instead of max pooling to reverse the reduction of the filter sizes. The three-layer autoencoder has 12,785 parameters, a little more than 5% of the capacity of the preceding deep autoencoder.

[16]:
```
x = Conv2D(filters=8,
           kernel_size=(3, 3),
           activation='relu',
           padding='same',
           name='Decoding_Conv_1')(encoded_conv)
x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_1')(x)
x = Conv2D(filters=16,
           kernel_size=(3, 3),
           activation='relu',
           padding='same',
           name='Decoding_Conv_2')(x)
x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_2')(x)
x = Conv2D(filters=32,
           kernel_size=(3, 3),
           activation='relu',
           name='Decoding_Conv_3')(x)
x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_3')(x)
decoded_conv = Conv2D(filters=1,
                      kernel_size=(3, 3),
                      activation='sigmoid',
                      padding='same',
                      name='Decoding_Conv_4')(x)
```

[17]:
```
autoencoder_conv = Model(input_, decoded_conv)
autoencoder_conv.compile(optimizer='adam', loss='mse')
```

[18]:
```
autoencoder_conv.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_3D (InputLayer)        [(None, 28, 28, 1)]       0
_________________________________________________________________
Encoding_Conv_1 (Conv2D)     (None, 28, 28, 32)        320
_________________________________________________________________
Encoding_Max_1 (MaxPooling2D (None, 14, 14, 32)        0
_________________________________________________________________
Encoding_Conv_2 (Conv2D)     (None, 14, 14, 16)        4624
_________________________________________________________________
Encoding_Max_2 (MaxPooling2D (None, 7, 7, 16)          0
_________________________________________________________________
Encoding_Conv_3 (Conv2D)     (None, 7, 7, 8)           1160
_________________________________________________________________
Encoding_Max_3 (MaxPooling2D (None, 4, 4, 8)           0
```

```
_________________________________________________________________
Decoding_Conv_1 (Conv2D)     (None, 4, 4, 8)           584
_________________________________________________________________
Decoding_Upsample_1 (UpSampl (None, 8, 8, 8)           0
_________________________________________________________________
Decoding_Conv_2 (Conv2D)     (None, 8, 8, 16)          1168
_________________________________________________________________
Decoding_Upsample_2 (UpSampl (None, 16, 16, 16)        0
_________________________________________________________________
Decoding_Conv_3 (Conv2D)     (None, 14, 14, 32)        4640
_________________________________________________________________
Decoding_Upsample_3 (UpSampl (None, 28, 28, 32)        0
_________________________________________________________________
Decoding_Conv_4 (Conv2D)     (None, 28, 28, 1)         289
=================================================================
Total params: 12,785
Trainable params: 12,785
Non-trainable params: 0
_________________________________________________________________
```

[19]:
```python
path = (results_path / 'autencoder_conv.32.weights.hdf5').as_posix()
```

[20]:
```python
autoencoder_deep, mse = train_autoencoder(path,
                                          autoencoder_conv,
                                          x_train=X_train_scaled,
                                          x_test=X_test_scaled)
```

```
Epoch 1/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0227 -
val_loss: 0.0164
Epoch 2/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0152 -
val_loss: 0.0142
Epoch 3/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0136 -
val_loss: 0.0130
Epoch 4/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0127 -
val_loss: 0.0124
Epoch 5/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0120 -
val_loss: 0.0117
Epoch 6/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0116 -
val_loss: 0.0114
Epoch 7/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0112 -
val_loss: 0.0113
```

```
Epoch 8/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0109 -
val_loss: 0.0111
Epoch 9/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0107 -
val_loss: 0.0105
Epoch 10/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0105 -
val_loss: 0.0104
Epoch 11/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0104 -
val_loss: 0.0104
Epoch 12/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0102 -
val_loss: 0.0101
Epoch 13/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0101 -
val_loss: 0.0102
Epoch 14/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0100 -
val_loss: 0.0099
Epoch 15/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0099 -
val_loss: 0.0097
Epoch 16/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0098 -
val_loss: 0.0097
Epoch 17/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0097 -
val_loss: 0.0096
Epoch 18/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0096 -
val_loss: 0.0096
Epoch 19/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0095 -
val_loss: 0.0095
Epoch 20/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0095 -
val_loss: 0.0094
Epoch 21/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0094 -
val_loss: 0.0095
Epoch 22/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0093 -
val_loss: 0.0094
Epoch 23/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0093 -
val_loss: 0.0095
```

```
Epoch 24/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0092 -
val_loss: 0.0092
Epoch 25/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0092 -
val_loss: 0.0093
Epoch 26/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0091 -
val_loss: 0.0091
Epoch 27/100
1688/1688 [==============================] - 4s 3ms/step - loss: 0.0091 -
val_loss: 0.0093
Epoch 28/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0091 -
val_loss: 0.0090
Epoch 29/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0090 -
val_loss: 0.0089
Epoch 30/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 31/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0089 -
val_loss: 0.0092
Epoch 32/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0089 -
val_loss: 0.0090
Epoch 33/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0089 -
val_loss: 0.0088
Epoch 34/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 35/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 36/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 37/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0088 -
val_loss: 0.0087
Epoch 38/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 39/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0087 -
val_loss: 0.0087
```

```
Epoch 40/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 41/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 42/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 43/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 44/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0086 -
val_loss: 0.0088
Epoch 45/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0086 -
val_loss: 0.0092
Epoch 46/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 47/100
1688/1688 [==============================] - 4s 3ms/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 48/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 49/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 50/100
1688/1688 [==============================] - 4s 3ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 51/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0085 -
val_loss: 0.0088
Epoch 52/100
1688/1688 [==============================] - 4s 3ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 53/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 54/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 55/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0085 -
val_loss: 0.0084
```

```
Epoch 56/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0085
Epoch 57/100
1688/1688 [==============================] - 5s 3ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 58/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 59/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0086
Epoch 60/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 61/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 62/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 63/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 64/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0084
Epoch 65/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0083
Epoch 66/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0083
Epoch 67/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0087
Epoch 68/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0083
Epoch 69/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0083
Epoch 70/100
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0083 -
val_loss: 0.0083
313/313 [==============================] - 0s 1ms/step - loss: 0.0083
```

Training stops after 75 epochs and results in a further 9% reduction of the test RMSE, due to a combination of the ability of convolutional filters to learn more efficiently from image data and the larger encoding size.

[21]:
```
f'MSE: {mse:.4f} | RMSE {mse**.5:.4f}'
```

[21]: 'MSE: 0.0083 | RMSE 0.0910'

[22]:
```
autoencoder_conv.load_weights(path)
```

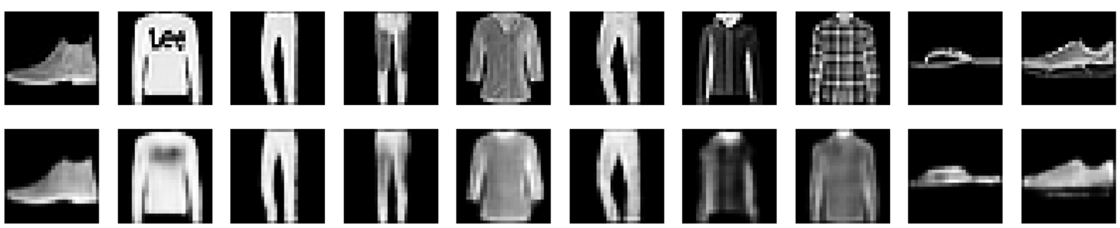
[23]:
```
reconstructed_images = autoencoder_deep.predict(X_test_scaled)
reconstructed_images.shape
```

[23]: (10000, 28, 28, 1)

[24]:
```
fig, axes = plt.subplots(ncols=n_classes, nrows=2, figsize=(20, 4))
for i in range(n_classes):

    axes[0, i].imshow(X_test_scaled[i].reshape(image_size, image_size), cmap='gray')
    axes[0, i].axis('off')

    axes[1, i].imshow(reconstructed_images[i].reshape(image_size, image_size) , cmap='gray')
    axes[1, i].axis('off')
```



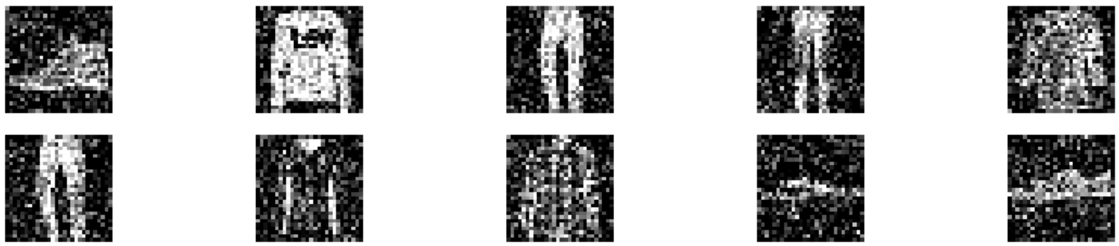
1.6 Denoising Autoencoder

The application of an autoencoder to a denoising task only affects the training stage. In this example, we add noise to the Fashion MNIST data from a standard normal distribution while maintaining the pixel values in the range of [0, 1], as follows:

[25]:
```
def add_noise(x, noise_factor=.3):
    return np.clip(x  + noise_factor * np.random.normal(size=x.shape), 0, 1)
```

[26]:
```
X_train_noisy = add_noise(X_train_scaled)
X_test_noisy = add_noise(X_test_scaled)
```

[27]:
```
fig, axes = plt.subplots(nrows=2, ncols=5, figsize=(20, 4))
axes = axes.flatten()
for i, ax in enumerate(axes):
    ax.imshow(X_test_noisy[i].reshape(28, 28), cmap='gray')
    ax.axis('off')
```



[28]:
```
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', name='Encoding_Conv_1')(input_)
x = MaxPooling2D(pool_size=(2, 2), padding='same', name='Encoding_Max_1')(x)
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', name='Encoding_Conv_2')(x)
encoded_conv = MaxPooling2D(pool_size=(2, 2), padding='same', name='Encoding_Max_3')(x)
```

[29]:
```
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', name='Decoding_Conv_1')(encoded_conv)
x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_1')(x)
x = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding='same', name='Decoding_Conv_2')(x)
x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_2')(x)
decoded_conv = Conv2D(filters=1, kernel_size=(3, 3), activation='sigmoid', padding='same', name='Decoding_Conv_4')(x)
```

[30]:
```
autoencoder_denoise = Model(input_, decoded_conv)
autoencoder_denoise.compile(optimizer='adam', loss='mse')
```

[31]:
```
path = (results_path / 'autencoder_denoise.32.weights.hdf5').as_posix()
```

[32]:
```
callbacks = [EarlyStopping(patience=5,
                           restore_best_weights=True),
             ModelCheckpoint(filepath=path,
                             save_best_only=True,
                             save_weights_only=True)]
```

We then proceed to train the convolutional autoencoder on noisy input with the objective to learn how to generate the uncorrupted originals:

[33]:

```
autoencoder_denoise.fit(x=X_train_noisy,
                        y=X_train_scaled,
                        epochs=100,
                        batch_size=128,
                        shuffle=True,
                        validation_split=.1,
                        callbacks=callbacks)
```

```
Epoch 1/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0243 -
val_loss: 0.0150
Epoch 2/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0140 -
val_loss: 0.0133
Epoch 3/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0126 -
val_loss: 0.0120
Epoch 4/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0117 -
val_loss: 0.0114
Epoch 5/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0111 -
val_loss: 0.0108
Epoch 6/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0106 -
val_loss: 0.0105
Epoch 7/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0103 -
val_loss: 0.0103
Epoch 8/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0101 -
val_loss: 0.0100
Epoch 9/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0099 -
val_loss: 0.0098
Epoch 10/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0097 -
val_loss: 0.0097
Epoch 11/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0096 -
val_loss: 0.0095
Epoch 12/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0095 -
val_loss: 0.0095
Epoch 13/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0094 -
val_loss: 0.0094
```

```
Epoch 14/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0093 -
val_loss: 0.0093
Epoch 15/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0093 -
val_loss: 0.0093
Epoch 16/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0092 -
val_loss: 0.0093
Epoch 17/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0092 -
val_loss: 0.0091
Epoch 18/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0091 -
val_loss: 0.0091
Epoch 19/100
422/422 [==============================] - 2s 6ms/step - loss: 0.0091 -
val_loss: 0.0090
Epoch 20/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 21/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 22/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0090 -
val_loss: 0.0091
Epoch 23/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 24/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0089 -
val_loss: 0.0090
Epoch 25/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 26/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0089 -
val_loss: 0.0090
Epoch 27/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0088 -
val_loss: 0.0089
Epoch 28/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0088 -
val_loss: 0.0089
Epoch 29/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0088 -
val_loss: 0.0088
```

```
Epoch 30/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 31/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 32/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 33/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 34/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 35/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 36/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 37/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 38/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 39/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 40/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 41/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 42/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 43/100
422/422 [==============================] - 2s 5ms/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 44/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 45/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0086
```

```
Epoch 46/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 47/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 48/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 49/100
422/422 [==============================] - 2s 4ms/step - loss: 0.0086 -
val_loss: 0.0086
```

[33]:
```
<tensorflow.python.keras.callbacks.History at 0x7f401c27fdc0>
```

[34]:
```
autoencoder_denoise.load_weights(path)
```

[35]:
```
mse = autoencoder_denoise.evaluate(x=X_test_noisy, y=X_test_scaled)
f'MSE: {mse:.4f} | RMSE {mse**.5:.4f}'
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.0087
```

[35]:
```
'MSE: 0.0087 | RMSE 0.0930'
```

1.7 Visualize Reconstructed Images

The following figure shows, from top to bottom, the original images as well as the noisy and denoised versions. It illustrates that the autoencoder is successful in producing compressed encodings from the noisy images that are quite similar to those produced from the original images:

[36]:
```
reconstructed_images = autoencoder_denoise.predict(X_test_noisy)
reconstructed_images.shape
```

[36]:
```
(10000, 28, 28, 1)
```

[37]:
```
fig, axes = plt.subplots(ncols=n_classes, nrows=3, figsize=(20, 6))
for i in range(n_classes):
    axes[0, i].imshow(X_test[i].reshape(image_size, image_size), cmap='gray')
    axes[0, i].axis('off')

    axes[1, i].imshow(X_test_noisy[i].reshape(image_size, image_size),␣
↪cmap='gray')
    axes[1, i].axis('off')

    axes[2, i].imshow(reconstructed_images[i].reshape(image_size, image_size) ,␣
↪cmap='gray')
    axes[2, i].axis('off')
fig.suptitle('Originals, Corrupted and Reconstructed Images', fontsize=20)
```

```
fig.tight_layout()
fig.subplots_adjust(top=.9)
fig.savefig(results_path / 'autoencoder_denoising', dpi=300)
```