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]:
```
from os.path import join
import pandas as pd

import numpy as np
from numpy.random import choice
from numpy.linalg import norm
import seaborn as sns

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from matplotlib.offsetbox import AnnotationBbox, OffsetImage
from mpl_toolkits.axes_grid1 import make_axes_locatable

from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras import regularizers
from keras.models import Model, model_from_json
from keras.callbacks import TensorBoard, EarlyStopping, ModelCheckpoint
from keras.datasets import fashion_mnist
from keras import backend as K

from sklearn.preprocessing import minmax_scale
from sklearn.manifold import TSNE
from sklearn.model_selection import train_test_split

from scipy.spatial.distance import pdist, cdist
```

```
Using TensorFlow backend.
```

[2]:
```
%matplotlib inline
plt.style.use('ggplot')
n_classes = 10 # all examples have 10 classes
cmap = sns.color_palette('Paired', n_classes)
pd.options.display.float_format = '{:,.2f}'.format
```

1.2 Fashion MNIST Data

[3]:
```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

[4]:
```
X_train.shape, X_test.shape
```

[4]:
```
((60000, 28, 28), (10000, 28, 28))
```

[5]:
```
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

[6]:
```
image_size = 28
```

[7]:
```
def data_prep_conv(x, size=image_size):
    return x.reshape(-1, size, size, 1).astype('float32')/255
```

[8]:
```
X_train_scaled = data_prep_conv(X_train)
X_test_scaled = data_prep_conv(X_test)
```

[9]:
```
X_train_scaled.shape, X_test_scaled.shape
```

[9]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

1.4 Combine training steps into function

[10]:
```
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

[11]:
```
input_ = Input(shape=(28, 28, 1), name='Input_3D')
```

1.5.2 Encoding Layers

[12]:
```
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)
```

```
WARNING:tensorflow:From
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.6/site-
packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

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.

[13]:
```
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)
```

[14]:
```
autoencoder_conv = Model(input_, decoded_conv)
autoencoder_conv.compile(optimizer='adam', loss='mse')
```

[15]:
```
autoencoder_conv.summary()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

[16]:
```python
path = 'models/fashion_mnist.autencoder_conv.32.weights.hdf5'
```

[17]:
```python
autoencoder_deep, mse = train_autoencoder(path,
                                          autoencoder_conv,
                                          x_train=X_train_scaled,
                                          x_test=X_test_scaled)
```

```
WARNING:tensorflow:From
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.6/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Train on 54000 samples, validate on 6000 samples
Epoch 1/100
54000/54000 [==============================] - 21s 398us/step - loss: 0.0233 -
val_loss: 0.0160
Epoch 2/100
54000/54000 [==============================] - 19s 361us/step - loss: 0.0149 -
val_loss: 0.0140
Epoch 3/100
54000/54000 [==============================] - 20s 371us/step - loss: 0.0134 -
val_loss: 0.0129
Epoch 4/100
54000/54000 [==============================] - 22s 401us/step - loss: 0.0125 -
val_loss: 0.0122
Epoch 5/100
54000/54000 [==============================] - 22s 410us/step - loss: 0.0120 -
val_loss: 0.0118
Epoch 6/100
54000/54000 [==============================] - 21s 389us/step - loss: 0.0115 -
val_loss: 0.0114
Epoch 7/100
54000/54000 [==============================] - 22s 412us/step - loss: 0.0112 -
val_loss: 0.0111
Epoch 8/100
54000/54000 [==============================] - 21s 383us/step - loss: 0.0109 -
val_loss: 0.0111
Epoch 9/100
54000/54000 [==============================] - 20s 364us/step - loss: 0.0107 -
val_loss: 0.0107
Epoch 10/100
54000/54000 [==============================] - 20s 368us/step - loss: 0.0106 -
val_loss: 0.0104
```

```
Epoch 11/100
54000/54000 [==============================] - 19s 361us/step - loss: 0.0104 -
val_loss: 0.0105
Epoch 12/100
54000/54000 [==============================] - 21s 396us/step - loss: 0.0103 -
val_loss: 0.0103
Epoch 13/100
54000/54000 [==============================] - 20s 364us/step - loss: 0.0102 -
val_loss: 0.0101
Epoch 14/100
54000/54000 [==============================] - 21s 393us/step - loss: 0.0101 -
val_loss: 0.0100
Epoch 15/100
54000/54000 [==============================] - 18s 337us/step - loss: 0.0100 -
val_loss: 0.0101
Epoch 16/100
54000/54000 [==============================] - 18s 334us/step - loss: 0.0099 -
val_loss: 0.0100
Epoch 17/100
54000/54000 [==============================] - 18s 331us/step - loss: 0.0098 -
val_loss: 0.0098
Epoch 18/100
54000/54000 [==============================] - 19s 360us/step - loss: 0.0097 -
val_loss: 0.0096
Epoch 19/100
54000/54000 [==============================] - 22s 398us/step - loss: 0.0096 -
val_loss: 0.0096
Epoch 20/100
54000/54000 [==============================] - 21s 387us/step - loss: 0.0096 -
val_loss: 0.0095
Epoch 21/100
54000/54000 [==============================] - 20s 368us/step - loss: 0.0095 -
val_loss: 0.0094
Epoch 22/100
54000/54000 [==============================] - 21s 389us/step - loss: 0.0095 -
val_loss: 0.0097
Epoch 23/100
54000/54000 [==============================] - 21s 392us/step - loss: 0.0094 -
val_loss: 0.0094
Epoch 24/100
54000/54000 [==============================] - 19s 356us/step - loss: 0.0094 -
val_loss: 0.0093
Epoch 25/100
54000/54000 [==============================] - 19s 354us/step - loss: 0.0093 -
val_loss: 0.0093
Epoch 26/100
54000/54000 [==============================] - 21s 384us/step - loss: 0.0093 -
val_loss: 0.0093
```

```
Epoch 27/100
54000/54000 [==============================] - 19s 350us/step - loss: 0.0092 -
val_loss: 0.0092
Epoch 28/100
54000/54000 [==============================] - 19s 358us/step - loss: 0.0092 -
val_loss: 0.0092
Epoch 29/100
54000/54000 [==============================] - 19s 354us/step - loss: 0.0092 -
val_loss: 0.0091
Epoch 30/100
54000/54000 [==============================] - 19s 350us/step - loss: 0.0091 -
val_loss: 0.0094
Epoch 31/100
54000/54000 [==============================] - 19s 351us/step - loss: 0.0091 -
val_loss: 0.0093
Epoch 32/100
54000/54000 [==============================] - 19s 351us/step - loss: 0.0091 -
val_loss: 0.0092
Epoch 33/100
54000/54000 [==============================] - 19s 359us/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 34/100
54000/54000 [==============================] - 22s 398us/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 35/100
54000/54000 [==============================] - 20s 377us/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 36/100
54000/54000 [==============================] - 20s 373us/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 37/100
54000/54000 [==============================] - 22s 414us/step - loss: 0.0089 -
val_loss: 0.0092
Epoch 38/100
54000/54000 [==============================] - 23s 430us/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 39/100
54000/54000 [==============================] - 25s 462us/step - loss: 0.0089 -
val_loss: 0.0093
Epoch 40/100
54000/54000 [==============================] - 23s 419us/step - loss: 0.0088 -
val_loss: 0.0089
Epoch 41/100
54000/54000 [==============================] - 20s 366us/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 42/100
54000/54000 [==============================] - 20s 374us/step - loss: 0.0088 -
val_loss: 0.0088
```

```
Epoch 43/100
54000/54000 [==============================] - 24s 442us/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 44/100
54000/54000 [==============================] - 24s 450us/step - loss: 0.0088 -
val_loss: 0.0087
Epoch 45/100
54000/54000 [==============================] - 21s 385us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 46/100
54000/54000 [==============================] - 48s 884us/step - loss: 0.0087 -
val_loss: 0.0086
Epoch 47/100
54000/54000 [==============================] - 48s 886us/step - loss: 0.0087 -
val_loss: 0.0086
Epoch 48/100
54000/54000 [==============================] - 50s 925us/step - loss: 0.0087 -
val_loss: 0.0089
Epoch 49/100
54000/54000 [==============================] - 50s 935us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 50/100
54000/54000 [==============================] - 36s 666us/step - loss: 0.0086 -
val_loss: 0.0089
Epoch 51/100
54000/54000 [==============================] - 21s 381us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 52/100
54000/54000 [==============================] - 20s 369us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 53/100
54000/54000 [==============================] - 22s 404us/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 54/100
54000/54000 [==============================] - 19s 356us/step - loss: 0.0086 -
val_loss: 0.0085
Epoch 55/100
54000/54000 [==============================] - 19s 344us/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 56/100
54000/54000 [==============================] - 19s 358us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 57/100
54000/54000 [==============================] - 19s 358us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 58/100
54000/54000 [==============================] - 19s 348us/step - loss: 0.0085 -
val_loss: 0.0085
```

```
Epoch 59/100
54000/54000 [==============================] - 20s 366us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 60/100
54000/54000 [==============================] - 20s 363us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 61/100
54000/54000 [==============================] - 20s 367us/step - loss: 0.0085 -
val_loss: 0.0084
Epoch 62/100
54000/54000 [==============================] - 19s 356us/step - loss: 0.0085 -
val_loss: 0.0084
Epoch 63/100
54000/54000 [==============================] - 20s 373us/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 64/100
54000/54000 [==============================] - 21s 392us/step - loss: 0.0084 -
val_loss: 0.0085
Epoch 65/100
54000/54000 [==============================] - 19s 360us/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 66/100
54000/54000 [==============================] - 19s 353us/step - loss: 0.0084 -
val_loss: 0.0085
Epoch 67/100
54000/54000 [==============================] - 20s 364us/step - loss: 0.0084 -
val_loss: 0.0084
Epoch 68/100
54000/54000 [==============================] - 20s 364us/step - loss: 0.0084 -
val_loss: 0.0089
Epoch 69/100
54000/54000 [==============================] - 19s 355us/step - loss: 0.0084 -
val_loss: 0.0085
Epoch 70/100
54000/54000 [==============================] - 20s 373us/step - loss: 0.0084 -
val_loss: 0.0084
10000/10000 [==============================] - 1s 112us/step
```

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.

[18]:
```python
f'MSE: {mse:.4f} | RMSE {mse**.5:.4f}'
```

[18]: 'MSE: 0.0084 | RMSE 0.0916'

[19]:
```python
autoencoder_conv.load_weights(path)
```

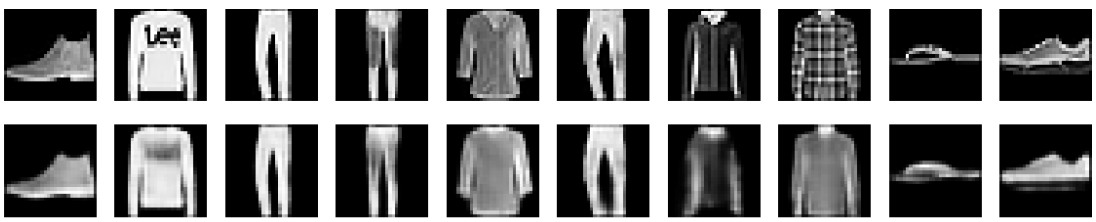
[20]:
```
reconstructed_images = autoencoder_deep.predict(X_test_scaled)
reconstructed_images.shape
```

[20]: (10000, 28, 28, 1)

[21]:
```
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')
```



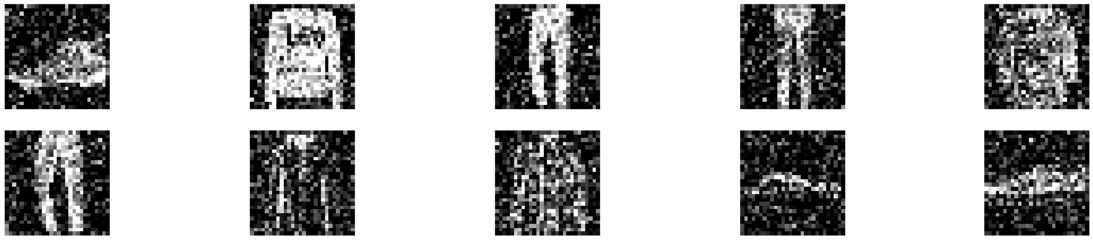
1.6 Denoising Autoencoder

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:

[22]:
```
def add_noise(x, noise_factor=.3):
    return np.clip(x  + noise_factor * np.random.normal(size=x.shape), 0, 1)
```

[23]:
```
X_train_noisy = add_noise(X_train_scaled)
X_test_noisy = add_noise(X_test_scaled)
```

[24]:
```
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')
```



[25]:
```
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)
```

[26]:
```
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)
```

[27]:
```
autoencoder_denoise = Model(input_, decoded_conv)
autoencoder_denoise.compile(optimizer='adam', loss='mse')
```

[28]:
```
path = 'models/fashion_mnist.autencoder_denoise.32.weights'
```

[29]:
```
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:

[30]:
```
autoencoder_denoise.fit(x=X_train_noisy,
                        y=X_train_scaled,
                        epochs=100,
                        batch_size=128,
                        shuffle=True,
                        validation_split=.1,
                        callbacks=callbacks)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/100
54000/54000 [==============================] - 17s 322us/step - loss: 0.0251 -
val_loss: 0.0158
Epoch 2/100
54000/54000 [==============================] - 17s 313us/step - loss: 0.0147 -
val_loss: 0.0141
Epoch 3/100
54000/54000 [==============================] - 17s 319us/step - loss: 0.0133 -
val_loss: 0.0127
Epoch 4/100
54000/54000 [==============================] - 17s 324us/step - loss: 0.0124 -
val_loss: 0.0119
Epoch 5/100
54000/54000 [==============================] - 17s 309us/step - loss: 0.0117 -
val_loss: 0.0114
Epoch 6/100
54000/54000 [==============================] - 17s 320us/step - loss: 0.0112 -
val_loss: 0.0109
Epoch 7/100
54000/54000 [==============================] - 17s 316us/step - loss: 0.0108 -
val_loss: 0.0105
Epoch 8/100
54000/54000 [==============================] - 17s 318us/step - loss: 0.0105 -
val_loss: 0.0103
Epoch 9/100
54000/54000 [==============================] - 18s 324us/step - loss: 0.0103 -
val_loss: 0.0102
Epoch 10/100
54000/54000 [==============================] - 16s 304us/step - loss: 0.0101 -
val_loss: 0.0100
Epoch 11/100
54000/54000 [==============================] - 17s 306us/step - loss: 0.0100 -
val_loss: 0.0099
Epoch 12/100
54000/54000 [==============================] - 17s 318us/step - loss: 0.0099 -
val_loss: 0.0098
Epoch 13/100
54000/54000 [==============================] - 17s 310us/step - loss: 0.0097 -
val_loss: 0.0097
Epoch 14/100
54000/54000 [==============================] - 17s 310us/step - loss: 0.0096 -
val_loss: 0.0095
Epoch 15/100
54000/54000 [==============================] - 17s 314us/step - loss: 0.0095 -
val_loss: 0.0095
Epoch 16/100
54000/54000 [==============================] - 17s 320us/step - loss: 0.0095 -
```

```
val_loss: 0.0094
Epoch 17/100
54000/54000 [==============================] - 18s 336us/step - loss: 0.0094 -
val_loss: 0.0094
Epoch 18/100
54000/54000 [==============================] - 19s 343us/step - loss: 0.0093 -
val_loss: 0.0093
Epoch 19/100
54000/54000 [==============================] - 18s 339us/step - loss: 0.0093 -
val_loss: 0.0092
Epoch 20/100
54000/54000 [==============================] - 17s 321us/step - loss: 0.0092 -
val_loss: 0.0092
Epoch 21/100
54000/54000 [==============================] - 20s 370us/step - loss: 0.0092 -
val_loss: 0.0091
Epoch 22/100
54000/54000 [==============================] - 18s 329us/step - loss: 0.0091 -
val_loss: 0.0091
Epoch 23/100
54000/54000 [==============================] - 19s 353us/step - loss: 0.0091 -
val_loss: 0.0091
Epoch 24/100
54000/54000 [==============================] - 17s 306us/step - loss: 0.0091 -
val_loss: 0.0091
Epoch 25/100
54000/54000 [==============================] - 17s 310us/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 26/100
54000/54000 [==============================] - 16s 303us/step - loss: 0.0090 -
val_loss: 0.0089
Epoch 27/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0090 -
val_loss: 0.0090
Epoch 28/100
54000/54000 [==============================] - 16s 298us/step - loss: 0.0090 -
val_loss: 0.0089
Epoch 29/100
54000/54000 [==============================] - 20s 361us/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 30/100
54000/54000 [==============================] - 16s 300us/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 31/100
54000/54000 [==============================] - 16s 299us/step - loss: 0.0089 -
val_loss: 0.0089
Epoch 32/100
54000/54000 [==============================] - 16s 296us/step - loss: 0.0089 -
```

```
val_loss: 0.0088
Epoch 33/100
54000/54000 [==============================] - 19s 352us/step - loss: 0.0088 -
val_loss: 0.0089
Epoch 34/100
54000/54000 [==============================] - 17s 320us/step - loss: 0.0088 -
val_loss: 0.0089
Epoch 35/100
54000/54000 [==============================] - 18s 325us/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 36/100
54000/54000 [==============================] - 17s 316us/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 37/100
54000/54000 [==============================] - 17s 306us/step - loss: 0.0088 -
val_loss: 0.0088
Epoch 38/100
54000/54000 [==============================] - 16s 304us/step - loss: 0.0088 -
val_loss: 0.0087
Epoch 39/100
54000/54000 [==============================] - 17s 306us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 40/100
54000/54000 [==============================] - 17s 321us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 41/100
54000/54000 [==============================] - 17s 308us/step - loss: 0.0087 -
val_loss: 0.0088
Epoch 42/100
54000/54000 [==============================] - 17s 324us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 43/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 44/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 45/100
54000/54000 [==============================] - 16s 288us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 46/100
54000/54000 [==============================] - 16s 288us/step - loss: 0.0087 -
val_loss: 0.0086
Epoch 47/100
54000/54000 [==============================] - 16s 290us/step - loss: 0.0087 -
val_loss: 0.0087
Epoch 48/100
54000/54000 [==============================] - 16s 292us/step - loss: 0.0086 -
```

```
val_loss: 0.0088
Epoch 49/100
54000/54000 [==============================] - 16s 290us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 50/100
54000/54000 [==============================] - 16s 288us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 51/100
54000/54000 [==============================] - 15s 285us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 52/100
54000/54000 [==============================] - 16s 287us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 53/100
54000/54000 [==============================] - 16s 288us/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 54/100
54000/54000 [==============================] - 16s 288us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 55/100
54000/54000 [==============================] - 16s 289us/step - loss: 0.0086 -
val_loss: 0.0087
Epoch 56/100
54000/54000 [==============================] - 15s 286us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 57/100
54000/54000 [==============================] - 16s 294us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 58/100
54000/54000 [==============================] - 16s 295us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 59/100
54000/54000 [==============================] - 16s 292us/step - loss: 0.0086 -
val_loss: 0.0086
Epoch 60/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 61/100
54000/54000 [==============================] - 16s 294us/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 62/100
54000/54000 [==============================] - 16s 295us/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 63/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 64/100
54000/54000 [==============================] - 16s 293us/step - loss: 0.0085 -
```

```
val_loss: 0.0085
Epoch 65/100
54000/54000 [==============================] - 16s 291us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 66/100
54000/54000 [==============================] - 16s 291us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 67/100
54000/54000 [==============================] - 16s 291us/step - loss: 0.0085 -
val_loss: 0.0086
Epoch 68/100
54000/54000 [==============================] - 16s 294us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 69/100
54000/54000 [==============================] - 16s 292us/step - loss: 0.0085 -
val_loss: 0.0085
Epoch 70/100
54000/54000 [==============================] - 16s 292us/step - loss: 0.0085 -
val_loss: 0.0085
```

[30]:
```
<keras.callbacks.History at 0x7f2a0d71a400>
```

[31]:
```python
autoencoder_denoise.load_weights(path)
```

[32]:
```python
mse = autoencoder_denoise.evaluate(x=X_test_noisy, y=X_test_scaled)
f'MSE: {mse:.4f} | RMSE {mse**.5:.4f}'
```

```
10000/10000 [==============================] - 1s 106us/step
```

[32]:
```
'MSE: 0.0086 | RMSE 0.0925'
```

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:

[33]:
```python
reconstructed_images = autoencoder_denoise.predict(X_test_noisy)
reconstructed_images.shape
```

[33]:
```
(10000, 28, 28, 1)
```

[34]:
```python
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('figures/autoencoder_denoising', dpi=300)
```