Convolutional & Denoising Autoencoders

SOUFIANE BOULAHCEN

The insights, 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

Imports & Settings

```
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.  %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
```

Fashion MNIST Data

```
(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()  X_train.shape, X_test.shape  ((60000,
28, 28), (10000, 28, 28))  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())
```

Reshape & normalize Fashion MNIST data

image_size = 28 **def** data_prep_conv(x, size=image_size):
return x.reshape(-1, size, size, 1).astype('float32')/255 X_train_scaled = data_prep_conv(X_train)
X_test_scaled = data_prep_conv(X_test) X_train_scaled.shape, X_test_scaled.shape ((60000, 28, 28,
1), (10000, 28, 28, 1))

Combine training steps into function

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

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 **4x4x8**, higher than for the preceding examples:

3-dim input

input_ = Input(shape=(28, 28, 1), name='Input_3D')

Encoding Layers

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. x = Conv2D(filters=8, kernel$_size$ = $(3, 3)$, $activation = '$ $relu'$, $padding$='same', name='Decoding_Conv_1')(encoded

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)

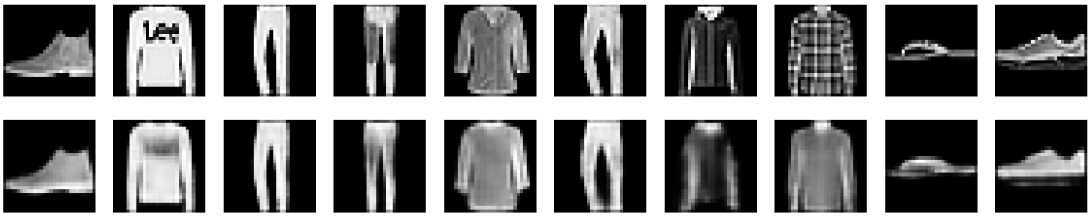
```
    x = UpSampling2D(size=(2, 2), name='Decoding_Upsample_3')(x)

    decoded_conv = Conv2D(filters=1, kernel_size=(3, 3), activation='sigmoid', padding='same', name='Decoding_Conv_4
    autoencoder_conv = Model(input_, decoded_conv)
autoencoder_conv.compile(optimizer='adam', loss='mse')
    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
_________________________________________________________________
 path = 'models/fashion_mnist.autencoder_conv.32.weights.hdf5' autoencoder_conv, 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
[... training log continues ...]
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. f'MSE: {mse:.4f}| RMSE {mse**.5:.4f}' 'MSE: 0.0084 | RMSE 0.0916' autoencoder_conv.load_weights(path)
reconstructed_images = autoencoder_conv.predict(X_test_scaled)

reconstructed_images.shape (10000, 28, 28, 1)
fig, axes = plt.subplots(ncols=n_classes, nrows=2, figsize=(20, 4))
for i **in** range(n_classes):

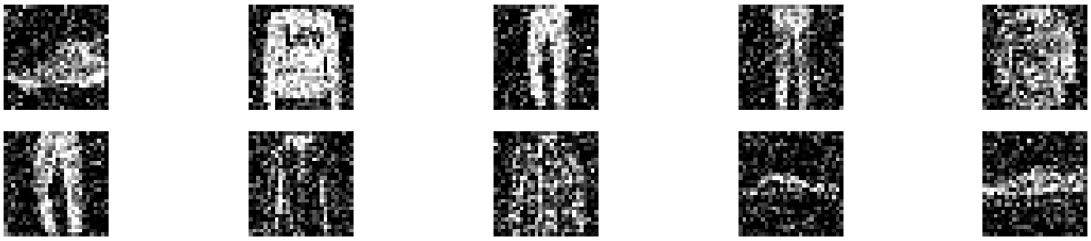
axes[0, i].imshow(X_test_scaled[i].reshape(image_size, image_size), cmap='gray')
axes[0, i].axis('off')

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



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: **def** add_noise(x, noise_factor=.3):
return np.clip(x + noise_factor * np.random.normal(size=x.shape), 0, 1) X_train_noisy = add_noise(X_train_scaled)
X_test_noisy = add_noise(X_test_scaled)
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')



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)

encoded_conv = MaxPooling2D(pool_size=(2, 2), padding='same', name='Encoding_Max_2')(x)
x = Conv2D(filters=16, kernel_size = (3, 3), activation =' relu', padding='same', name='Decoding_Conv_1')(encoded_c

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_3')(
autoencoder_denoise = Model(input_, decoded_conv)
autoencoder_denoise.compile(optimizer='adam', loss='mse')
path = 'models/fashion_mnist.autencoder_denoise.32.weights.hdf5'
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: 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
[... training log continues ...]
Epoch 70/100
54000/54000 [==============================] - 16s 292us/step - loss: 0.0085 - val_loss: 0.0085 <keras.callbacks.History at 0x7f2a0d71a400> autoencoder_denoise.load_weights(path) 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 'MSE: 0.0086 | RMSE 0.0925'

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: reconstructed_images = autoencoder_denoise.predict(X_test_noisy)
reconstructed_images.shape (10000, 28, 28, 1) 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)
```