2_AutoEncoder

May 28, 2024

1 Autoencoder CNN for Time Series Denoising

As a second example, we will create another convolutional neural network (CNN), but this time for time series denoising. The type of neural network architecture we ar using for that purpose is the one of an **autoencoder**.

1.1 Autoencoder Structure and Purpose

Autoencoders are a type of unsupervised neural network. They do not use labeled classes or any labeled data. They are in general used to

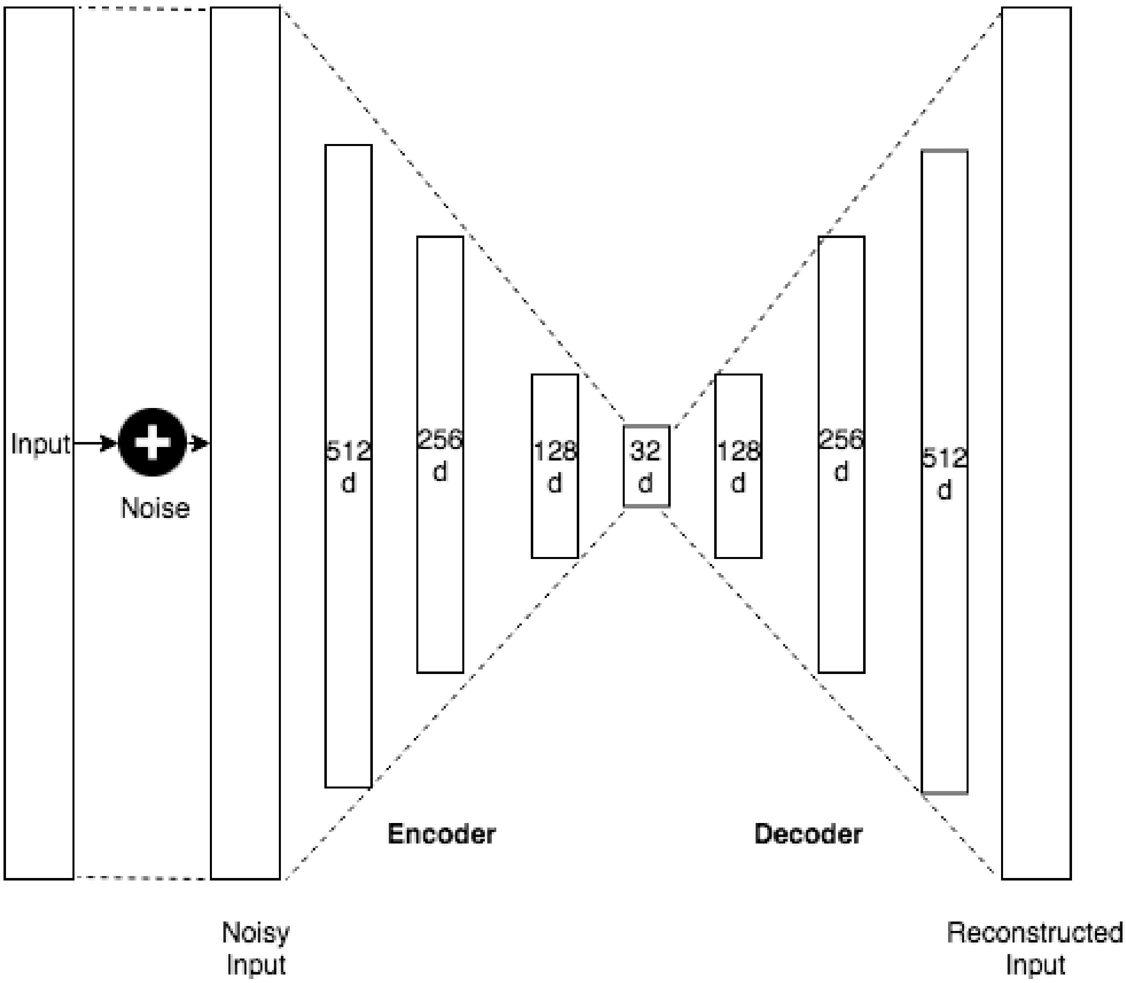
- Accept an input set of data
- Internally compress the input data into a latent-space representation
- Reconstruct the input data from this latent representation

An autoencoder is having two components:

- **Encoder:** Accepts the input data and compresses it into the latent-space. If we denote our input data as x and the encoder as E, then the output latent-space representation, s, would be $s = E(x)$.
- **Decoder:** The decoder is responsible for accepting the latent-space representation s and then reconstructing the original input. If we denote the decoder function as D and the output of the detector as o, then we can represent the decoder as $o = D(s)$.

Also not originally developed to denoise data, we will construct an autoencoder, which is learning to denoise a time series. Autoencoders are also often used to remove noise from images before applying a CNN to image classification.

The shape of the autoencoder network could be the following. We take a timeseries as input, which could contain 1024 data points. The datapoints are then compressed down to only 32 datapoints in the encoder steps and then decoded back into the original 1024 datapoint. Due to the compression and the action of the weights, a part of the noise is removed. In our example below, we will represent the autoencoder with a CNN.



1.2 Data Generation

We will apply (and train) the network to a data series containing a noisy sine wave. In a first step, we will generate data for that purpose. For the convolutional network, our data shall be two dimensional. We therefore squeeze our linear timeseries in a two dimensional array with 28 x 28 data points.

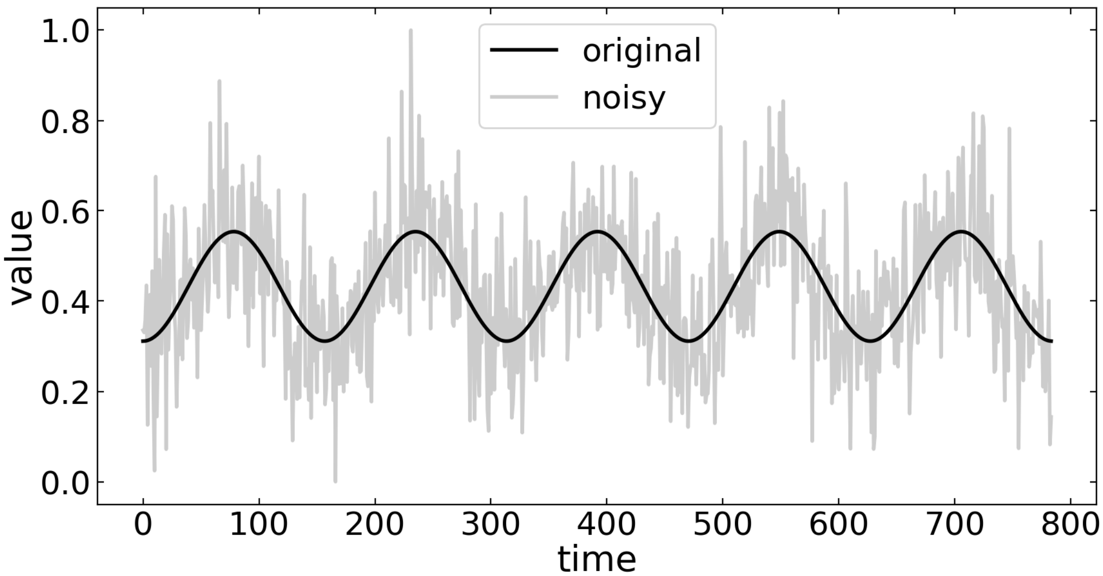
We create two series

- a noisy sample with noise, noisy_input
- a pure series without noise, pure_input

Overall we will generate 100000 datasets. We will take from that data the training, the testing and the validation data.

The following cell will generate all the data based on a sine function and add quite a lot of noise. It will also normalize the input data to have all in a restructed value range.

The plot below just depicts a random sample of our data.



1.3 Create the Autoencoder network

This creates an autoencoder. Two convolutional layers for the encoding and two for decoding.

1.3.1 Define model data

1.3.2 Encoder/Decoder Setup

The next cell defines the actual autoencoder network. The network consists of a

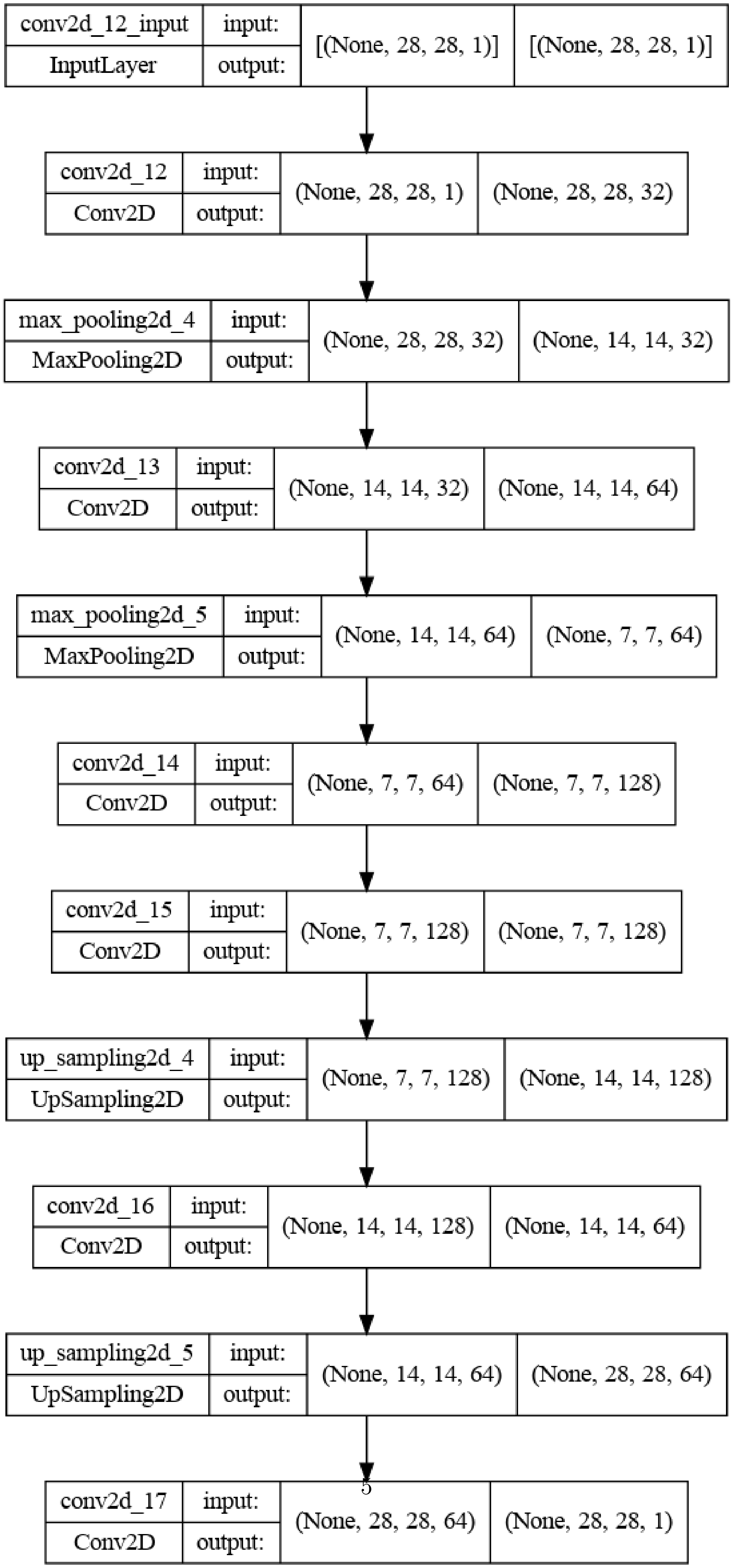
encoder - 28 x 28 datapoints input - convolutional layer with 32 kernels of 3 x 3 size and ReLU activation - pooling layer using the maxima of a 2 x 2 matrix - convolutional layer with 64 kernels of 3 x 3 size and ReLU activation - pooling layer using the maxima of a 2 x 2 matrix - convolutional layer with 128 kernels of 3 x 3 size and ReLU activation

decoder - convolutional layer with 128 kernels of 3 x 3 size and ReLU activation - upsampling layer increasing the data by a factor of 2 x 2 - convolutional layer with 64 kernels of 3 x 3 size and ReLU activation - upsampling layer increasing the data by a factor of 2 x 2 - convolutional layer with 1 kernels of 3 x 3 size and ReLU activation

The summary function shows the parameters of the network, especially the output shape of each layer.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 28, 28, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 14, 14, 32)       0
 2D)
```

```
 conv2d_13 (Conv2D)          (None, 14, 14, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 7, 7, 64)         0
 2D)

 conv2d_14 (Conv2D)          (None, 7, 7, 128)         73856

 conv2d_15 (Conv2D)          (None, 7, 7, 128)         147584

 up_sampling2d_4 (UpSampling  (None, 14, 14, 128)      0
 2D)

 conv2d_16 (Conv2D)          (None, 14, 14, 64)        73792

 up_sampling2d_5 (UpSampling  (None, 28, 28, 64)       0
 2D)

 conv2d_17 (Conv2D)          (None, 28, 28, 1)         577

=================================================================
Total params: 314,625
Trainable params: 314,625
Non-trainable params: 0
_________________________________________________________________
```


conv2d_12_input
InputLayer
input:
output:
[(None, 28, 28, 1)]
[(None, 28, 28, 1)]
conv2d_12
Conv2D
input:
output:
(None, 28, 28, 1)
(None, 28, 28, 32)
max_pooling2d_4
MaxPooling2D
input:
output:
(None, 28, 28, 32)
(None, 14, 14, 32)
conv2d_13
Conv2D
input:
output:
(None, 14, 14, 32)
(None, 14, 14, 64)
max_pooling2d_5
MaxPooling2D
input:
output:
(None, 14, 14, 64)
(None, 7, 7, 64)
conv2d_14
Conv2D
input:
output:
(None, 7, 7, 64)
(None, 7, 7, 128)
conv2d_15
Conv2D
input:
output:
(None, 7, 7, 128)
(None, 7, 7, 128)
up_sampling2d_4
UpSampling2D
input:
output:
(None, 7, 7, 128)
(None, 14, 14, 128)
conv2d_16
Conv2D
input:
output:
(None, 14, 14, 128)
(None, 14, 14, 64)
up_sampling2d_5
UpSampling2D
input:
output:
(None, 14, 14, 64)
(None, 28, 28, 64)
conv2d_17
Conv2D
input:
output:
(None, 28, 28, 64)
(None, 28, 28, 1)

1.3.3 Training the encoder

The only thing we have to do now, is to compile the model and train the network on our generated data.

```
Epoch 1/5
```

```
2023-07-11 15:17:25.371248: W
tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 175616000
exceeds 10% of free system memory.
2023-07-11 15:17:25.470598: W
tensorflow/core/framework/cpu_allocator_impl.cc:82] Allocation of 175616000
exceeds 10% of free system memory.
```

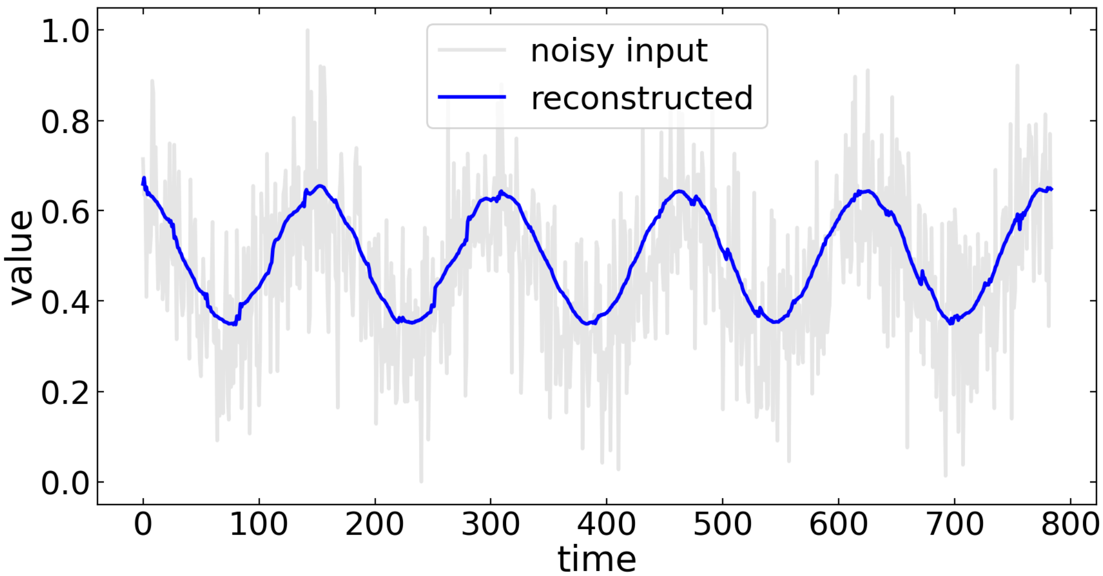
```
374/374 [==============================] - 73s 193ms/step - loss: 0.0193 -
val_loss: 0.0063
Epoch 2/5
374/374 [==============================] - 73s 194ms/step - loss: 0.0025 -
val_loss: 0.0015
Epoch 3/5
374/374 [==============================] - 73s 194ms/step - loss: 0.0016 -
val_loss: 7.3326e-04
Epoch 4/5
374/374 [==============================] - 73s 195ms/step - loss: 0.0012 -
val_loss: 0.0026
Epoch 5/5
374/374 [==============================] - 73s 194ms/step - loss: 0.0011 -
val_loss: 0.0013
```

```
<keras.callbacks.History at 0x7fae84ba6340>
```

1.4 Reconstruction of the Data

After 5 episodes of training, we are ready to test the model on a testing example of our data. That means we supply some timeseries that is unknown to the network. The cell below is doing 2000 reconstructions at a time.

Having done all these 2000 reconstructions, we can select an arbitrary one to plot the noisy input data together with the denoised data.


noisy input
reconstructed
value
time
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400
500
600
700
800