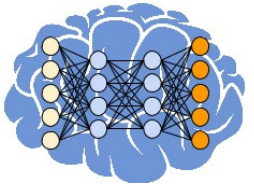


RWTH AACHEN UNIVERSITY

Deep Learning Train-the-Trainer Workshop

Autoencoders

Uwe Klemradt

RWTH Aachen University

Starting point: standard network

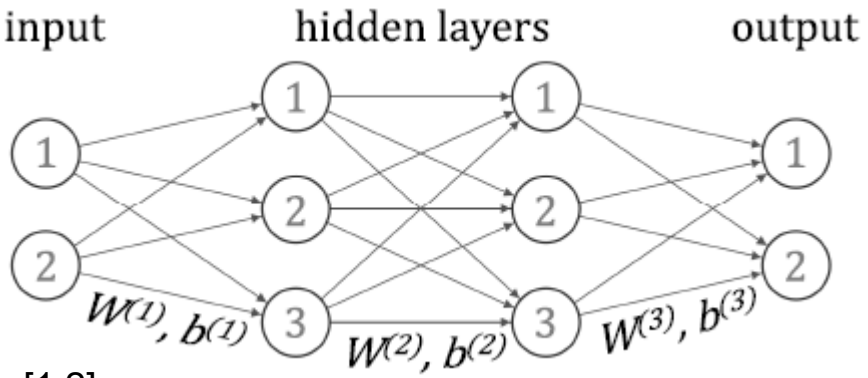
- Fully-connected networks:

  matrix multiplication between layers / bias vectors / nonlinear activation functions

  weights and biases are fixed by training

- Input data are processed from left to right (feedforward processing)

  network ready for new input after each output, results are not kept



[1,2]

Autoencoders

- Elementary feedforward networks for unsupervised learning (= no labeled data necessary)
- Origins in data compression and dimensionality reduction
- Idea: use bottleneck structure to force network to learn essential features

  two network parts: encoder + decoder
- Training can be difficult
- Many applications nowadays in a wide range of fields, well beyond original purpose

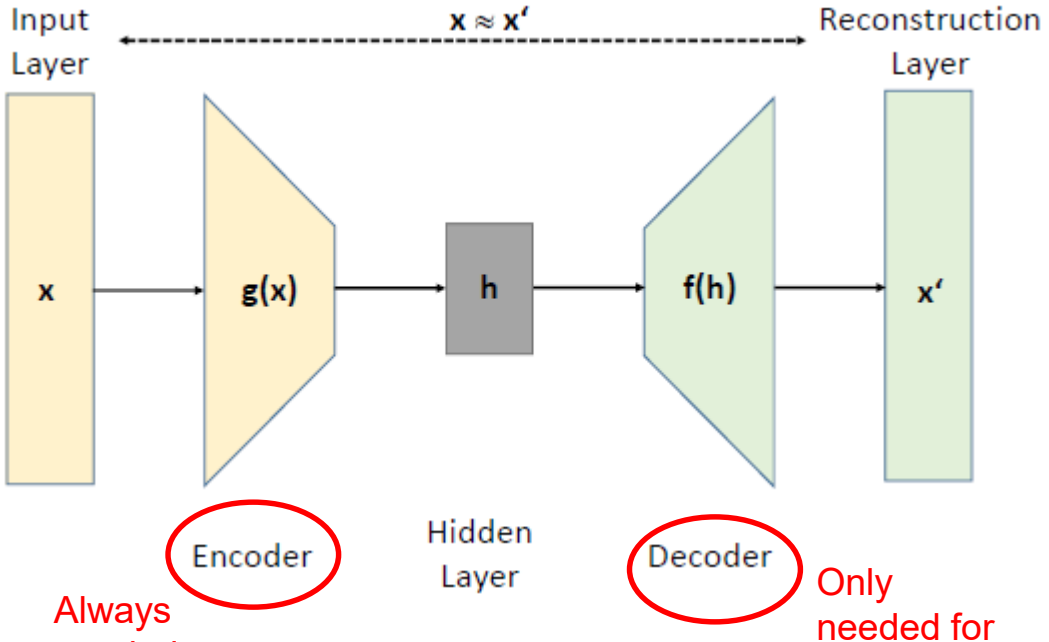
  (e.g., outlier detection, de-noising, data generation…)

- Core task: reproduce given input **x** at output **x'** (layers of same dimensionality)

  measure similarity by appropriate loss function

  avoid trivial solution (identity function) by bottleneck structure

Autoencoders



Also called:
Compressed representation
Latent variables
Latent vector
Latent space

Loss function as similarity measure:

$$\mathcal{L}(\mathrm{x}, f(g(\mathrm{x})) = \mathcal{L}(\mathrm{x}, \mathrm{x}')$$

Examples:

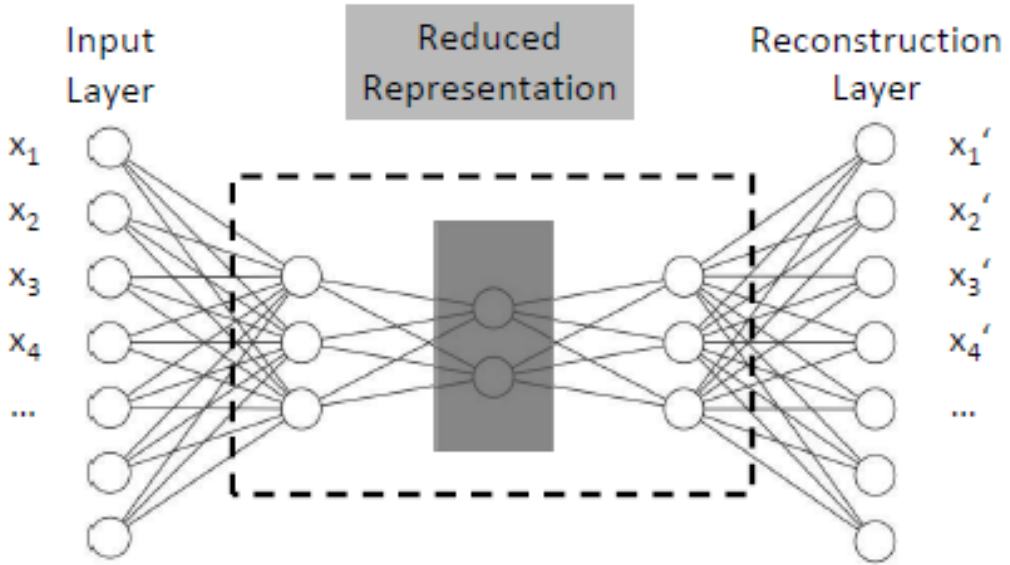
Mean-squared-error (real-valued input):

$$\mathcal{L}_{MSE}(\mathrm{x}, \mathrm{x}') = ||\mathrm{x} - \mathrm{x}'||^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - x_i')^2$$

Cross-entropy (discrete-valued input):

$$\mathcal{L}_{CE}(\mathrm{x}, \mathrm{x}') = \frac{1}{N}\sum_{i=1}^{N}[\,-x_i \log x_i' - (1 - x_i)\log(1 - x_i')\,]$$

Autoencoder example



Network example of a simple autoencoder:

- Two nodes in the compressed representation (may be used for 2D data visualization)
- Encoding performed by dimensionality reduction (matrix multiplication) + nonlinear activation functions

One hidden layer: *shallow* autoencoder

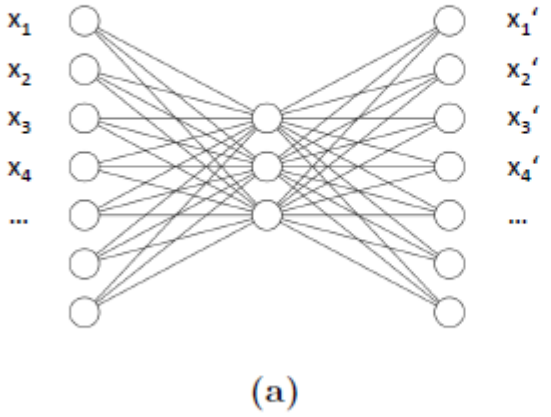
Multiple hidden layers: *deep* autoencoder

Autoencoder creates a representation of the data by mapping them onto a lower-dimensional surface.

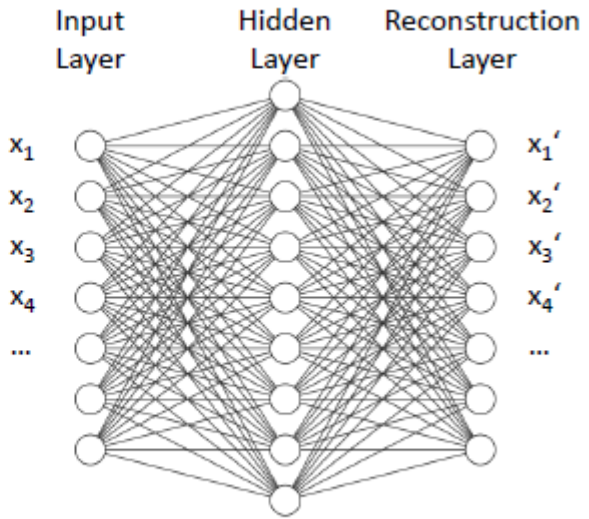
Works well for internally correlated data!

Autoencoder types

- Shallow / deep
- Undercomplete vs. overcomplete:



(a)



(b)

Bottleneck?

→ Yes – e.g. by training (de-activation of nodes)

Autoencoder types

- Regularized

  = additional constraint on the loss function (def.)

Subtypes:

sparse $\mathcal{L}(\mathbf{x}, f(g(\mathbf{x})) \rightarrow \mathcal{L}(\mathbf{x}, \mathbf{x}') + \lambda\,\Omega(\mathbf{h})$ various sparsity penalty terms possible

contractive $$\Omega(\mathbf{h}) = \sum_i ||\nabla_x h_i||^2$$

derivatives $\rightarrow$ learning of features that change only slightly

de-noising $\mathcal{L}(\mathbf{x}, f(g(\mathbf{x})) = \mathcal{L}(\mathbf{x}, \mathbf{x}')$ *replaced* $\rightarrow$ *minimize instead* $\mathcal{L}(\mathbf{x}, f(g(\tilde{\mathbf{x}}))$

De-noising autoencoders

- $\tilde{x}$ is a distorted ('corrupted') version of x
- Examples:

- Basic autoencoder: trained to replicate input
- De-noising autoencoder: trained to undo corruption

Modified training: add specific noise to uncorrupted data
autoencoder learns to reconstruct
uncorrupted data from noisy (corrupted) input

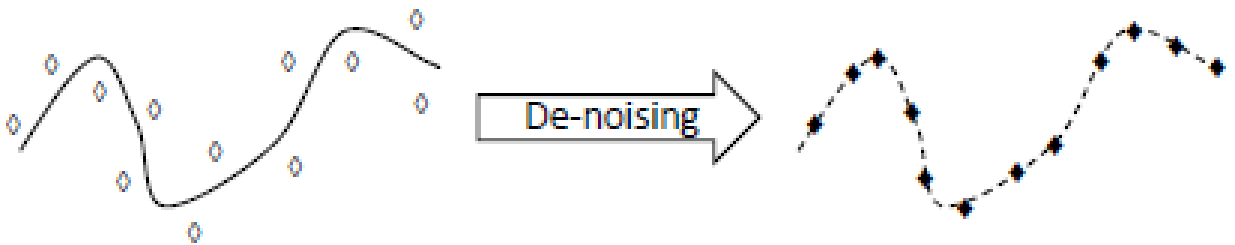
De-noising autoencoders

Idea:

Training = assessment of true manifold from which uncorrupted data originate

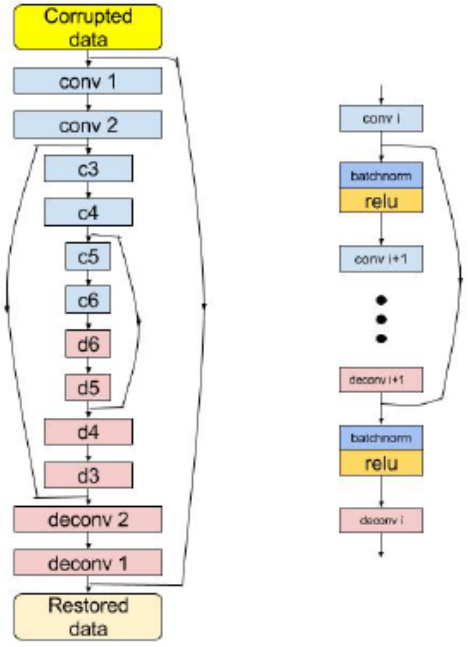
Application to data not seen before = data are treated as corrupted,

projection on manifold as estimated from previous training



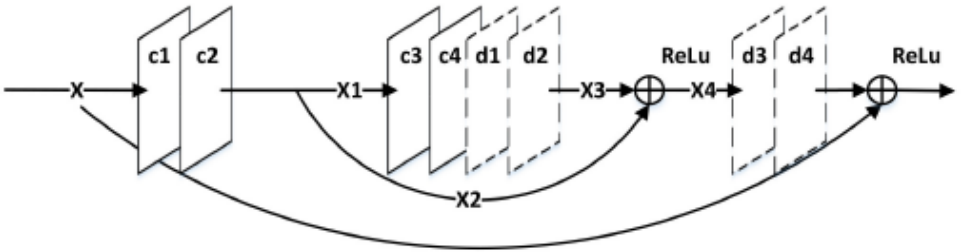
Deep autoencoder: shortcuts

- Signal can be backpropagated directly (solves vanishing gradient problem)
- Passes images details from convolution to deconvolution layers
  $\rightarrow$ better image reconstruction

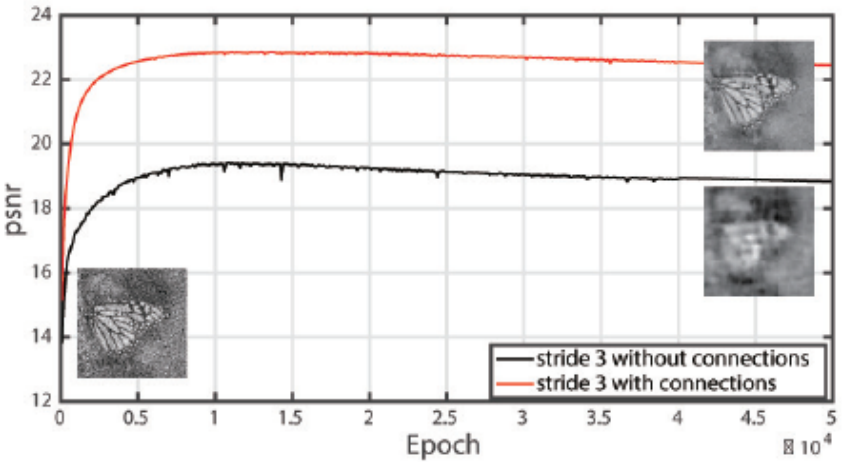


(a) Network architecture (b) Shortcut connection

Deep autoencoder: shortcuts




```
X1 = ...
c3 = Conv2D(...)(X1)
... # some (de-)convolution
    operations
X3 = Conv2D(...)(d2)
    # X1.shape == X3.shape
sc1 = add()([X1, X3])
X4 = Activation('relu')(sc1)
```



Useful functions for convolutional autoencoders

Decoding

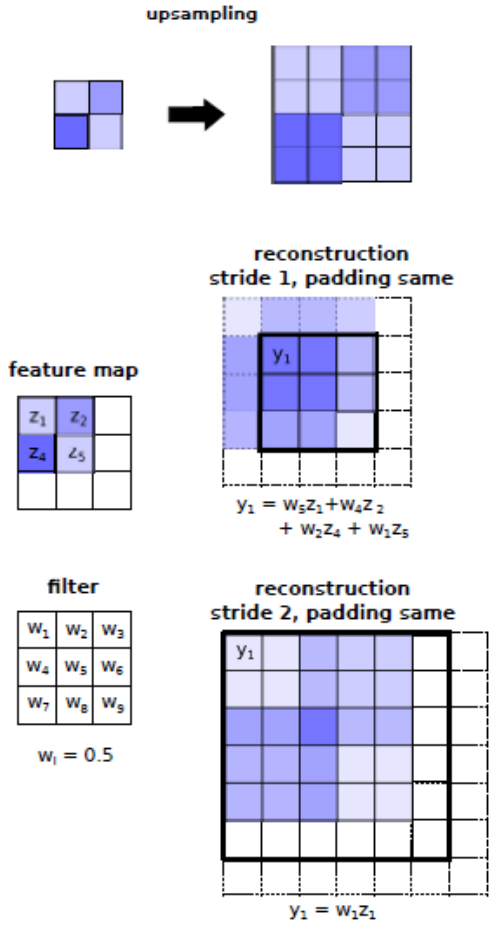
- Upsampling

```
UpSampling2D((2,2))
```

- 2D-Convolution
- "Deconvolutions" (Transposed Convolutions)
  → Same as backward pass for normal 2D-Convolution

```
Conv2DTranspose(filters, kernel_size,
                strides=(1, 1), padding='same')
```

```
Conv2DTranspose(filters, kernel_size,
                strides=(2, 2), padding='same')
```



Coding examples

- Example 1

```
input_data = Input(shape=(32,32,3))
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
encoded = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(3, (3, 3), activation='linear', padding='same')(x)
```

- Example 2

```
input_img = Input(shape=(32, 32, 3))
x = Conv2D(32, (3, 3), strides=(2, 2), activation='relu', padding='same')
    (input_img)
encoded = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
decoded = Conv2DTranspose(3, (3, 3), strides=(2, 2), activation='linear',
     padding='same')(encoded)
```

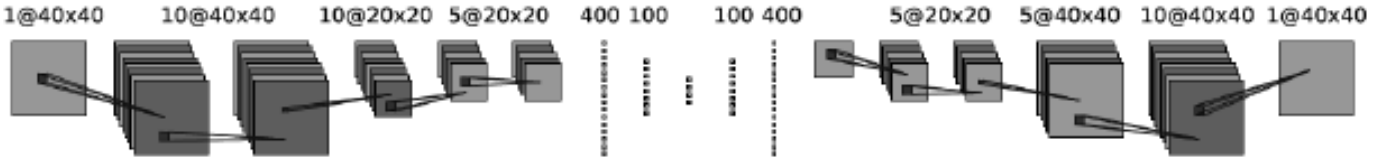
Autoencoders applications

- Finding relevant subspaces of multidimensional data
  (nonlinear generalization of principal component analysis)
- Visualization of multidimensional data
- Use loss function beyond training to detect outliers
- …

Application example in physics:

Jet analysis in high energy physics

- search for anomalies in jets of collision experiments (outliers = new physics?)
- automatic monitoring of jet substructure from decays of heavy resonances
- unsupervised comparison of experimental data to compressed representation of an autoencoder

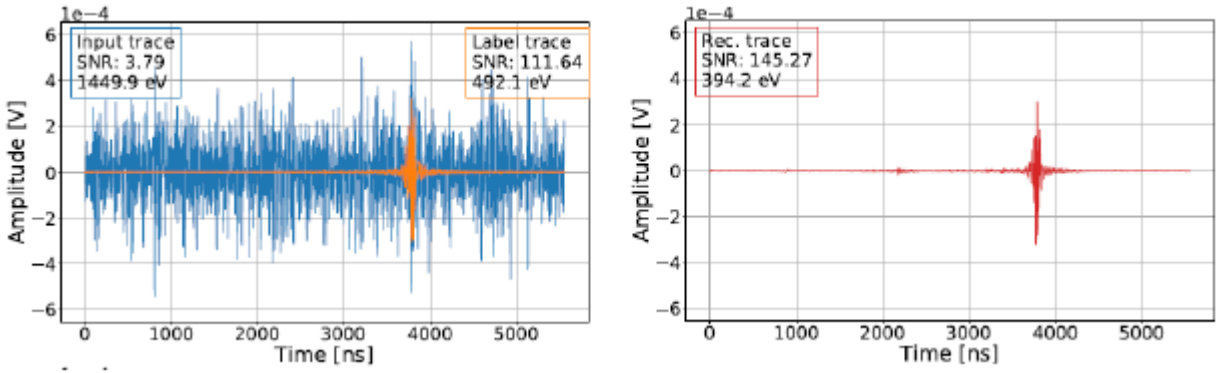


T. Heimel, G. Kasieczka, T. Plehn and J. Thompson, QCD or what? *SciPost Physics* **6**, 3 (2019), doi:10.21468/scipostphys.6.3.030, `http://dx.doi.org/10.21468/SciPostPhys.6.3.030`.

Application example in physics:

De-noising of air showers from cosmic rays

- disentanglement of signal and simultaneously recorded noise
- can be suitable for problems with unfavorable signal-to-noise ratios
- fine details of signal and noise are learned during training; features associated with the background are reconstructed



M. Erdmann et al.,
J. Instrumentation 14, 04005 (2019)

Autoencoder summary

- Autoencoders consist of an input and output layer with the same dimensionality and one (or more) hidden layers.
- Encoder: maps the input to a hidden representation
- Decoder: maps the hidden representation to the output
- Autoencoder training: maximizing the similarity between input and output
  ($\rightarrow$ allows unsupervised learning)
- Typical architecture: bottleneck (reduced data representation)
- Decoder only needed for training (but important in special applications)
- Deep autoencoders: multiple layers create hierarchically reduced representations of the data
- Many variants: sparse, contractive, variational, de-noising
  ($\rightarrow$ de-noising type does not maximize similarity between input and output, but reconstructs clean versions from noisy input)

References

- I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, Chapter 14, MIT Press, 2016
- M. Erdmann, J. Glombitza, G. Kasieczka, U. Klemradt, *Deep Learning for Physics Research*, Chapter 17, World Scientific, 2021
- G.E. Hinton, R.R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science 313, 504 (2006)
- J. Schmidhuber, Deep Learning in Neural Networks: An Overview, Neural Networks 61, 85 (2015)
- P. Vincent et al., Stacked Denoising Autoencoders: Learning Useful Representation in a Deep Network with a Local Denoising Criterion, J. Machine Learning Res. 11, 3371 (2010)
- X.-J. Mao et al., Image Restoration Using Convolutional Auto-Encoders with Symmetric Skip Connections, (2016) https://arxiv.org/pdf/1606.08921.pdf
- J. Dong et al., Learning Deep Representations Using Convolutional Auto-Encoders with Symmetric Skip Connections, (2017) https://arxiv.org/abs/1611.09119

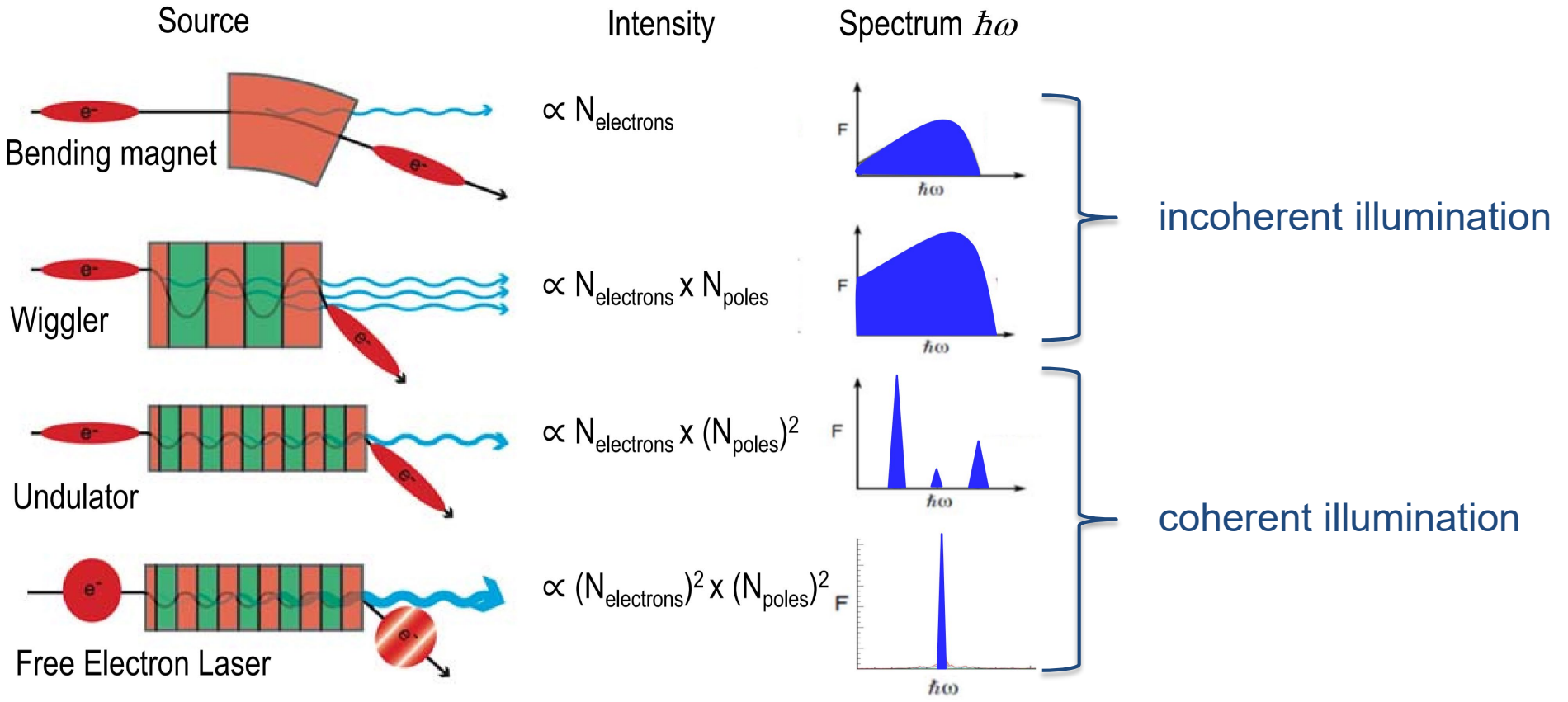
Exercise: Synchrotron Radiation Data

- highly brilliant X-ray photon sources
- broad spectrum of energies
- allows large variety of experiments in condensed matter physics, materials science, chemistry, life sciences, etc.

Apparent noise in data sets can have many causes:

- low intensity at extended ranges
- cross-sections for incoherent scattering from sample features (e.g., roughness)
- cross sections for coherent scattering (speckles, e.g. from waveguide illumination)

Exercise: Synchrotron Radiation Data



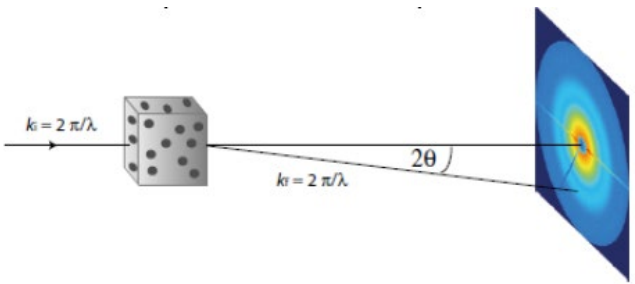
Source: https://physics.stackexchange.com/q/514819

Exercise: Synchrotron Radiation Data

Application example: Small Angle X-ray Scattering (SAXS)

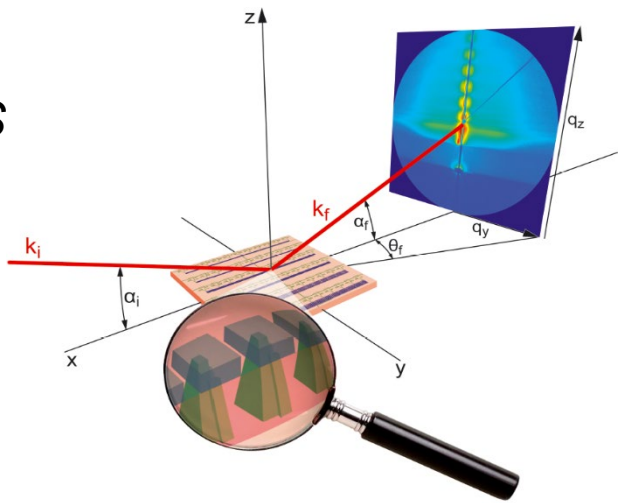
- elastic, diffuse scattering from variations in electron density (= structural or chemical inhomogeneities)
- scattering angle $2\theta \leq 5°$ allows resolving structures up to 1000 nm $(1/d \sim \sin\theta)$
- Focused beams (100 nm x 100 nm) from undulators allow position-dependent measurements
- Transmission and reflection geometry:

SAXS



Grazing Incidence SAXS

(GISAXS)



Exercise

http://deeplearningphysics.org/ or Google Colab: https://colab.research.google.com/github/DeepLearningForPhysicsResearchBook/deep-learning-physics

Exercise 17.1

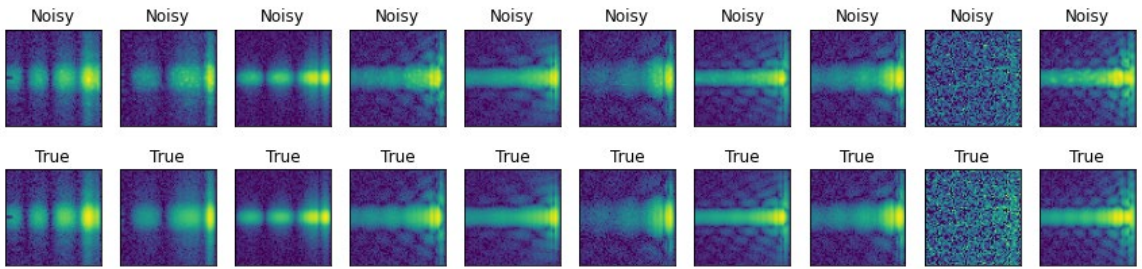
Speckle removal with denoising autoencoders (DAEs)

Small-angle scattering of X-rays or neutrons enables insights into properties of nanostructured materials. In the case of X-rays from undulators at synchrotron radiation sources, extreme beam focusing can result in a high degree of coherence. Interference effects called speckles then appear naturally in the recorded data. This may be an unwanted effect that makes the measurements appear noisy.

Try to remove the speckles from the given test samples:

1. Set up and train a deep convolutional autoencoder.
2. State the test loss and comment on the reconstruction results for some test images.
3. Apply the autoencoder to experimental data from partially coherent illumination and describe your observations.

Training DAEs on the provided data can be computationally demanding, thus, we recommend to use a GPU for this task.



Exercise 17.1 walkthrough

```
import numpy as np
import h5py
import matplotlib.pyplot as plt
from tensorflow import keras

layers = keras.layers

print(keras.__version__)
```

```
2.4.0
```

```
import gdown
import os
url = "https://drive.google.com/u/0/uc?export=download&confirm=HgGH&id=1CJVbIkLmzDTfC6ftaIUPiuqjoC0ay-wr"
output = 'speckles.npz'

if os.path.exists(output) == False:
    gdown.download(url, output, quiet=True)
```

Exercise 17.1 walkthrough

RWTH AACHEN UNIVERSITY

Preprocess data

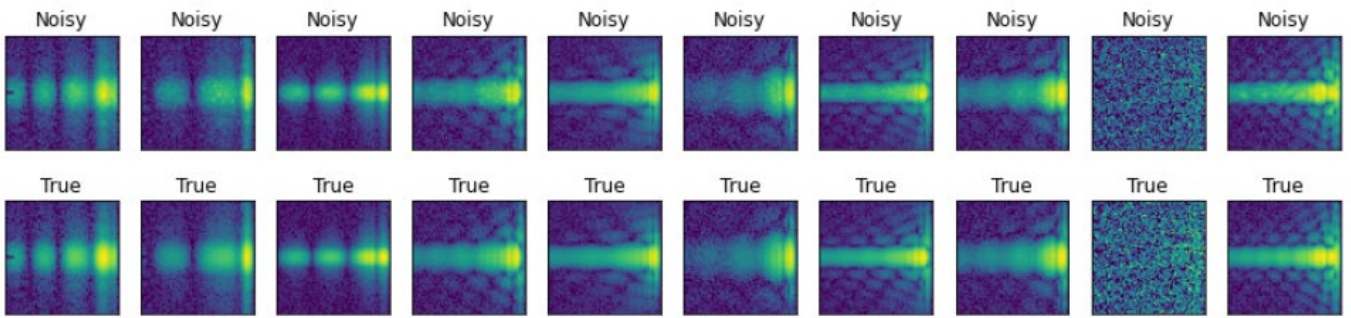
```
f = np.load(output)
image_normal = f['target_images']
image_speckle = f['speckle_images']

# logarithmic intensity values
image_normal = np.log10(image_normal + 1.)
image_speckle = np.log10(image_speckle + 1.)

# norm input data to max. value of distorted scattering pattern
max_val = np.max(image_speckle, axis=(1, 2, 3), keepdims=True)

image_normal = image_normal / max_val
image_normal = np.clip(image_normal, 0., 1.1)  # limit maximum intensity values

image_speckle = image_speckle / max_val
image_speckle = np.clip(image_speckle, 0., 1.1)  # limit maximum intensity values
```

```
n_train = 20000
n = image_normal.shape[0]

x_train_noisy, x_test_noisy = image_speckle[0:n_train], image_speckle[n_train:]
x_train, x_test = image_normal[0:n_train], image_normal[n_train:]
```

Exercise 17.1 walkthrough

Plot example data

```
plots = 10
plt.figure(1, (15, 3.5))
idx = np.random.choice(n, 10)

for i in range(plots):
    plt.subplot(2, plots, i+1)
    plt.imshow(image_speckle[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("Noisy")

    plt.subplot(2, plots, plots+i+1)
    plt.imshow(image_normal[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("True")

plt.show()
```



Exercise 17.1 walkthrough

Model building

```
input_img = layers.Input(shape=(64, 64, 1))
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
c1 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(c1)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
c2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(c2)
encoded = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)

# decoder part
x = layers.UpSampling2D((2, 2))(encoded)
x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
c2_2 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = layers.Add()([c2, c2_2])              # shortcuts
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
c1_2 = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = layers.Add()([c1, c1_2])              # shortcuts
x = layers.Conv2D(32, (3, 3), activation='linear', padding='same')(x)
decoded = layers.Conv2D(1, (3, 3), activation='linear', padding='same')(x)

autoencoder = keras.models.Model(input_img, decoded)

print(autoencoder.summary())
```

Exercise 17.1 walkthrough

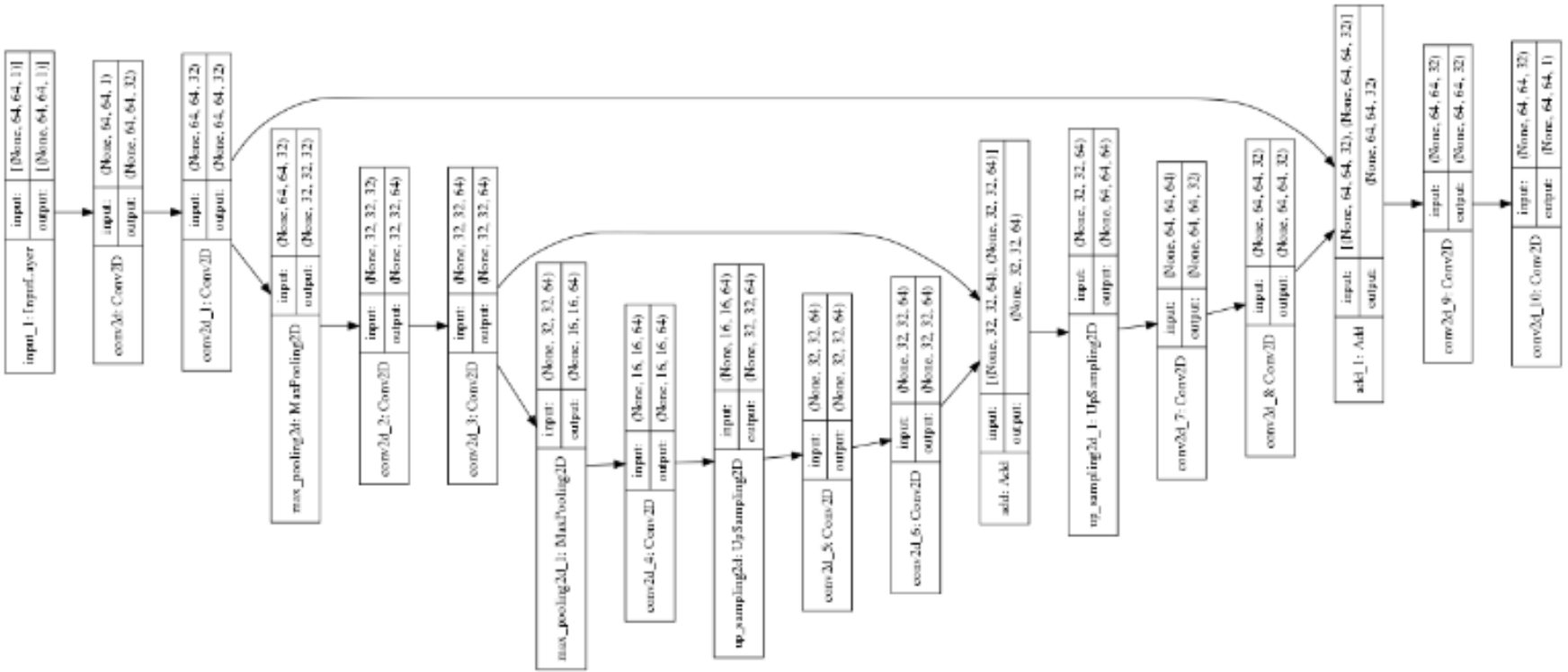
RWTH AACHEN UNIVERSITY

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 64, 64, 1)]  0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 64, 64, 32)   320         input_1[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 64, 64, 32)   9248        conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d (MaxPooling2D)    (None, 32, 32, 32)   0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 32, 32, 64)   18496       max_pooling2d[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 32, 32, 64)   36928       conv2d_2[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 16, 16, 64)   0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 16, 16, 64)   36928       max_pooling2d_1[0][0]
__________________________________________________________________________________________________
up_sampling2d (UpSampling2D)    (None, 32, 32, 64)   0           conv2d_4[0][0]
__________________________________________________________________________________________________
conv2d_5 (Conv2D)               (None, 32, 32, 64)   36928       up_sampling2d[0][0]
__________________________________________________________________________________________________
conv2d_6 (Conv2D)               (None, 32, 32, 64)   36928       conv2d_5[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 32, 32, 64)   0           conv2d_3[0][0]
                                                                 conv2d_6[0][0]
__________________________________________________________________________________________________
up_sampling2d_1 (UpSampling2D)  (None, 64, 64, 64)   0           add[0][0]
__________________________________________________________________________________________________
conv2d_7 (Conv2D)               (None, 64, 64, 32)   18464       up_sampling2d_1[0][0]
__________________________________________________________________________________________________
conv2d_8 (Conv2D)               (None, 64, 64, 32)   9248        conv2d_7[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 64, 64, 32)   0           conv2d_1[0][0]
                                                                 conv2d_8[0][0]
__________________________________________________________________________________________________
conv2d_9 (Conv2D)               (None, 64, 64, 32)   9248        add_1[0][0]
__________________________________________________________________________________________________
conv2d_10 (Conv2D)              (None, 64, 64, 1)    289         conv2d_9[0][0]
==================================================================================================
Total params: 213,025
Trainable params: 213,025
Non-trainable params: 0
```

Exercise 17.1 walkthrough

Plot model to visualize the shortcuts

```
keras.utils.plot_model(autoencoder, show_shapes=True, dpi=60)
```



Exercise 17.1 walkthrough

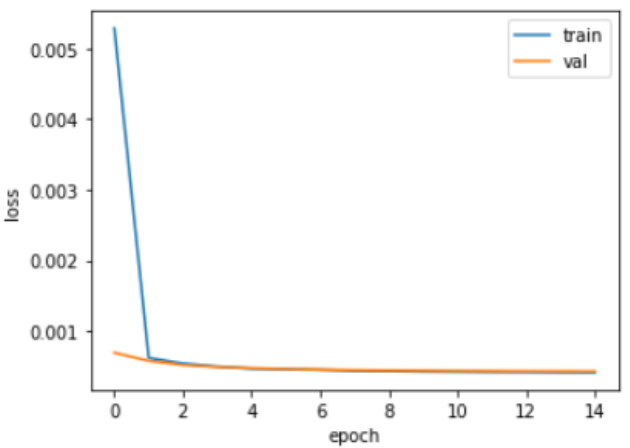
Compile and train the DAE

```
autoencoder.compile(optimizer=keras.optimizers.Adam(0.001), loss='mse')

earlystopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, verbose=1)
rl_on_plateau = keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.67, patience=2, verbose=1, min_lr=1e-5)
```

```
results = autoencoder.fit(x_train_noisy, x_train,
                          epochs=15,
                          batch_size=128,
                          validation_split=0.1,
                          verbose=1,
                          callbacks=[earlystopping, rl_on_plateau])
```

Plot training history

```
plt.figure(1, (12, 4))
plt.subplot(1, 2, 1)
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper right')
```

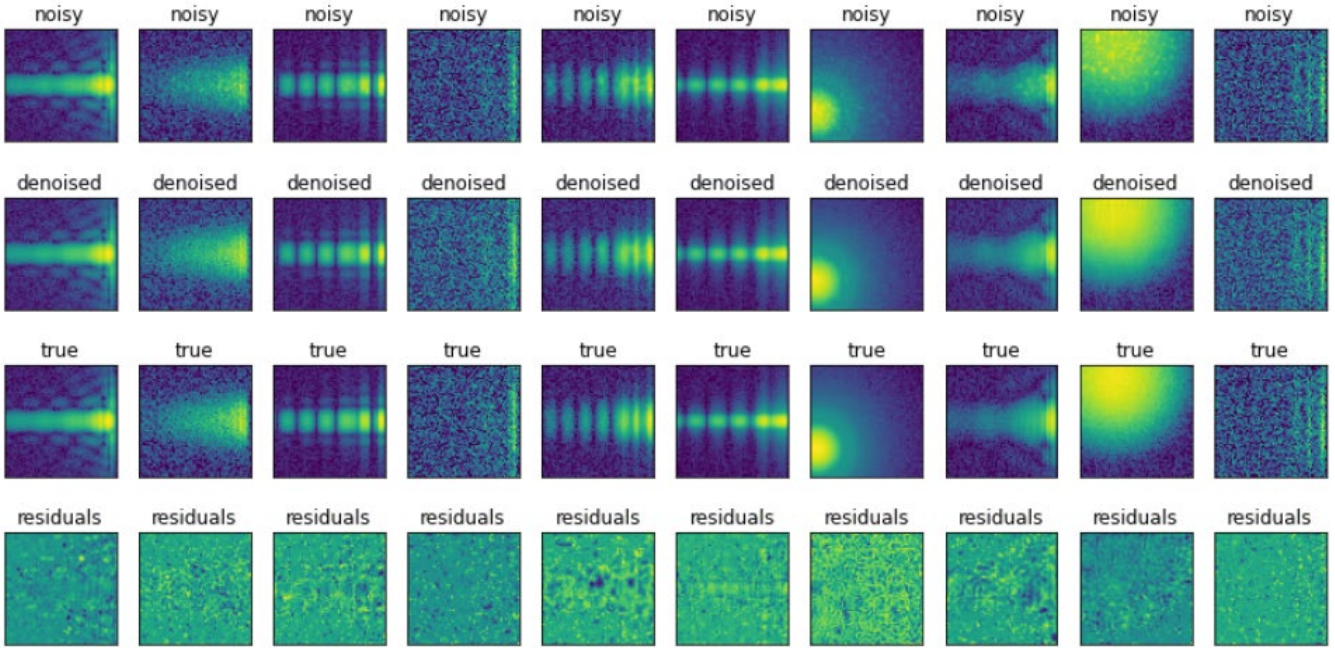


Exercise 17.1 walkthrough

Investigate the DAE performance using the test data

```
preds = autoencoder.predict(x_test_noisy, verbose=1)
```

```
plots = 10
n_test = x_test.shape[0]
plt.figure(1, (15, 7.5))
idx = np.random.choice(n_test, 10)

for i in range(plots):
    plt.subplot(4, plots, i+1)
    plt.imshow(x_test_noisy[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("noisy")

    plt.subplot(4, plots, plots+i+1)
    plt.imshow(preds[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("denoised")

    plt.subplot(4, plots, 2*plots+i+1)
    plt.imshow(x_test[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("true")

    plt.subplot(4, plots, 3*plots+i+1)
    plt.imshow(x_test[idx[i],:,:,0] - preds[idx[i],:,:,0])
    plt.xticks([])
    plt.yticks([])
    plt.title("residuals")

plt.show()
```



Exercise 17.1 walkthrough

Thank you very much for your attention!

Have a try for yourselves at Exercise 17.1:

http://deeplearningphysics.org/ or Google Colab:
https://colab.research.google.com/github/DeepLearningForPhysicsResearchBook/deep-learning-physics