Autoencoder

Diane Lingrand

UNIVERSITÉ
CÔTE D'AZUR

Master Data Science DSAI M1

2021 - 2022

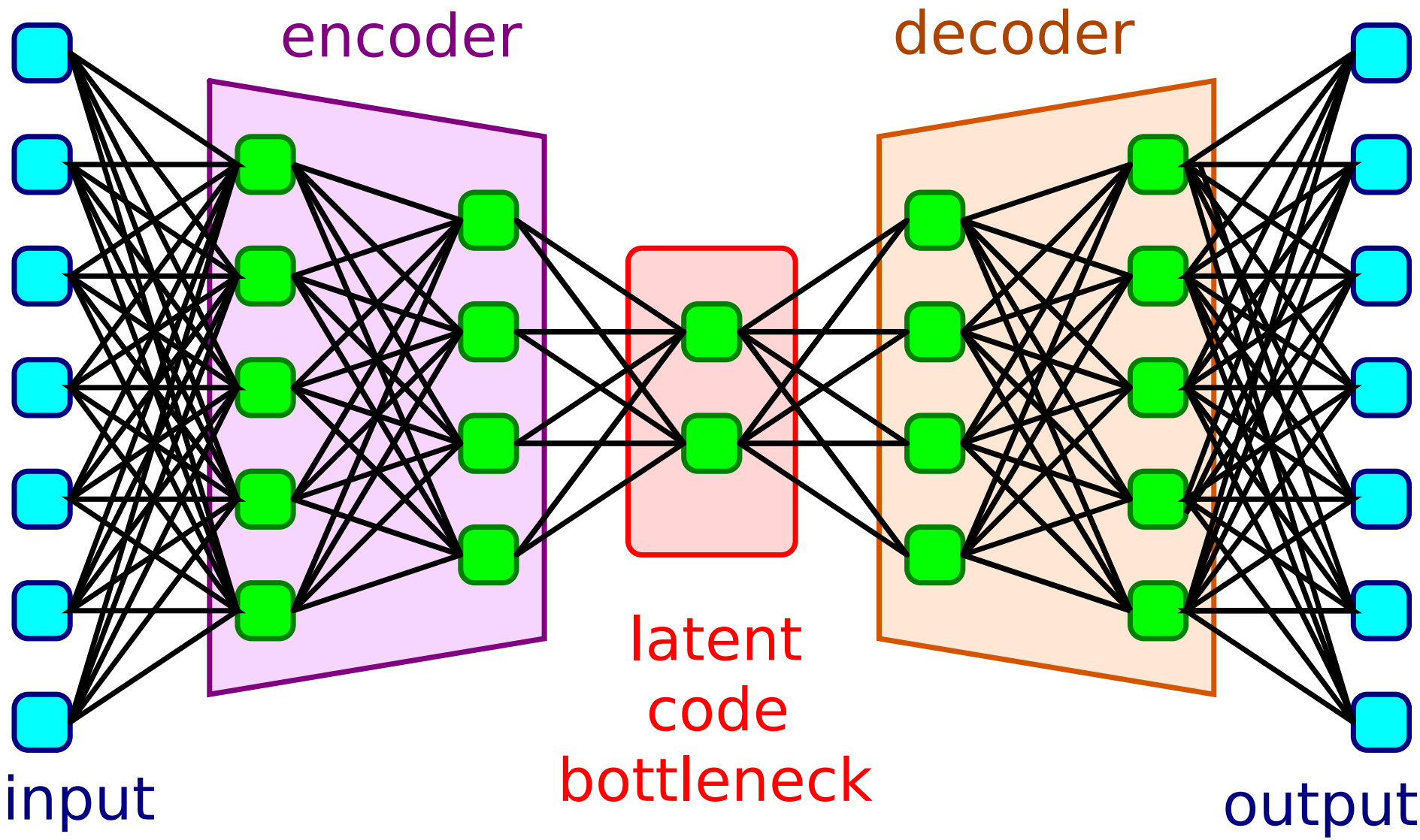
Outline

Context

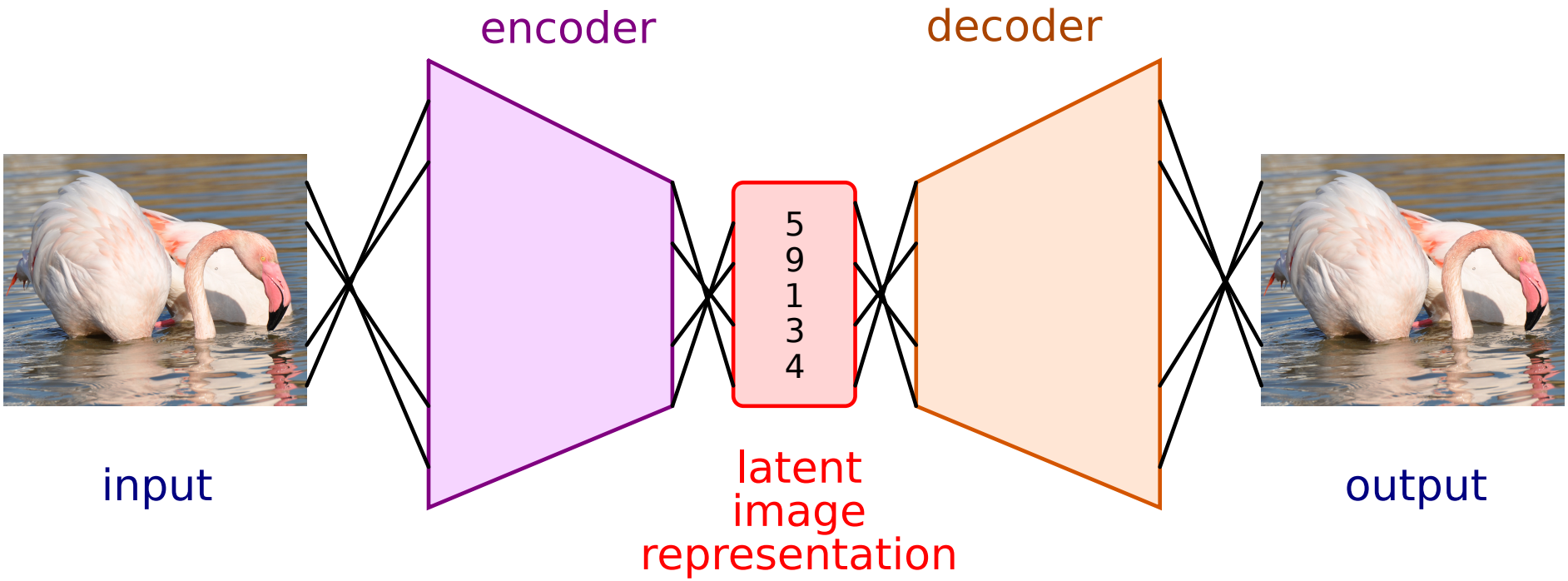
- non supervised
- reduction of the size of the data
- applications :
  - denoising / inpainting
  - segmentation
  - data representation
  - data compression (not as good as JPEG for images)
  - data generation

Outline

Principle

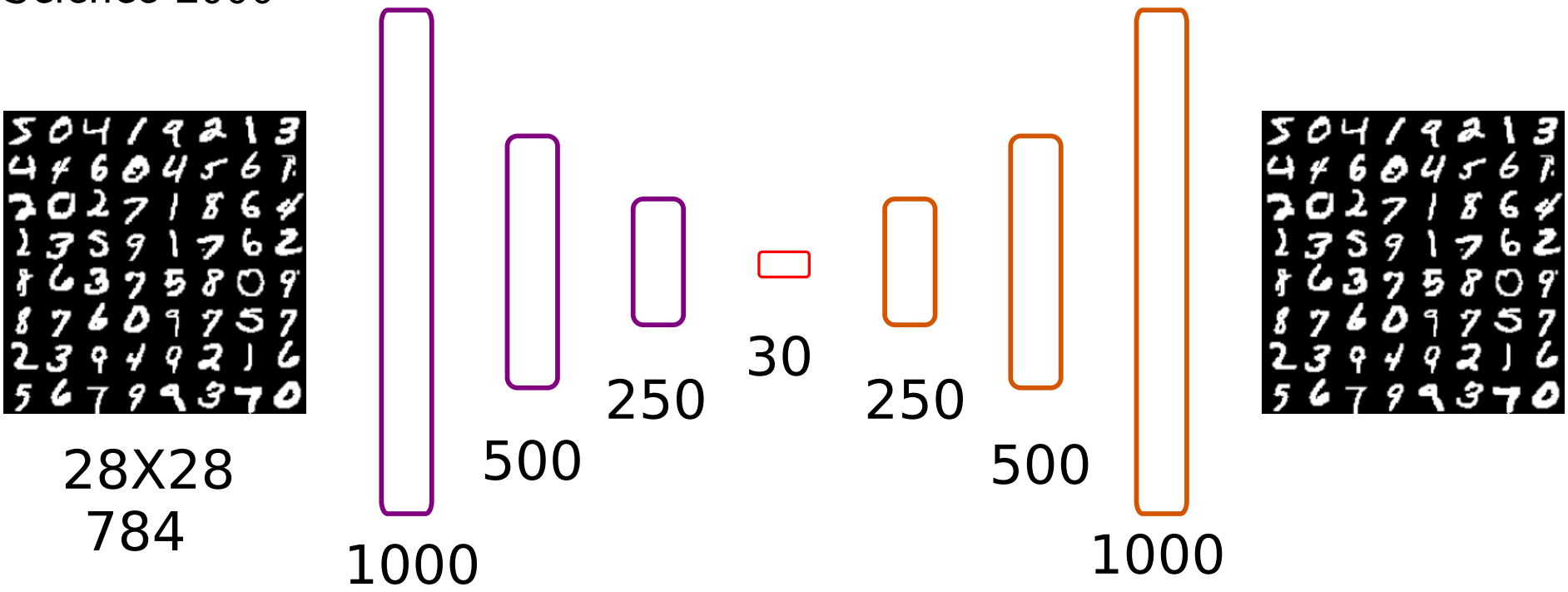


Application to images



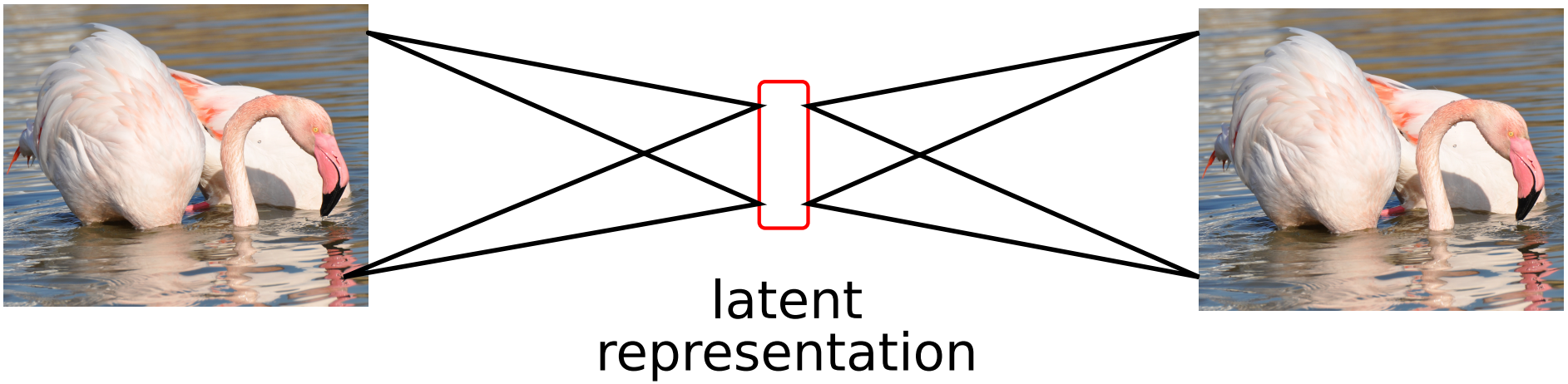
Historic autoencoder

The first really successfull autoencoder : Hinton and Salakhutdinov, Science 2006



The simplest autoencoder : Vanilla autoencoder

- one hidden layer
- similarities with PCA (linear functions of data, minimisation of square reconstruction error)
  - PCA : Covariance matrix $= [e_1 e_2].\begin{bmatrix}\lambda_1 & 0\\ 0 & \lambda_2\end{bmatrix}.[e_1 e_2]^T$
  - autoencoder : vectors not forced to be orthogonal, variances tend to be equal

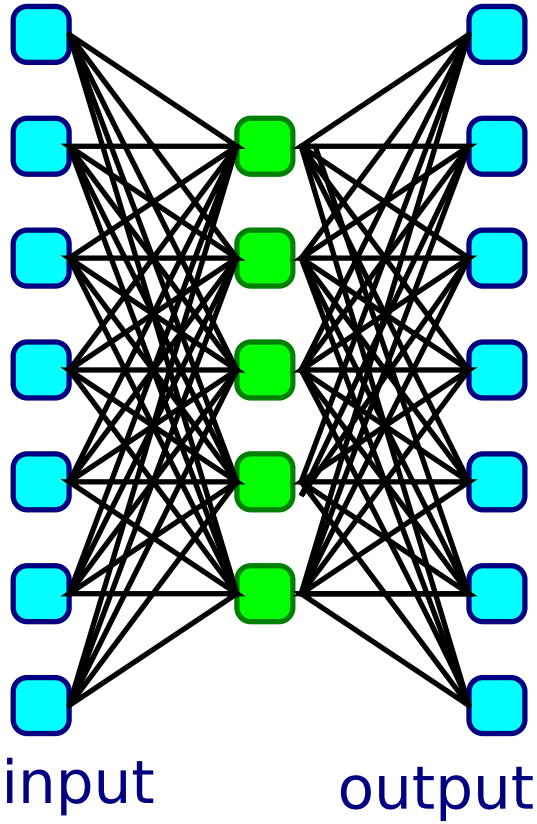


Vanilla autoencoder in keras

```
from keras.layers import Dense
from keras.models import Input, Model
from keras.datasets import fashion_mnist
import numpy as np

(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

input_image = Input(shape=(784,))
encoded = Dense(32, activation='relu')(input_image)
decoded = Dense(784, activation='sigmoid')(encoded)
autoencoder = Model(input_image, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
autoencoder.fit(x_train, x_train,epochs=100,batch_size=256,\\
  shuffle=True,validation_data=(x_test, x_test))
decoded_images = autoencoder.predict(x_test)
score = autoencoder.evaluate(x_test, x_test, batch_size=128)
print("score = ", score)
```

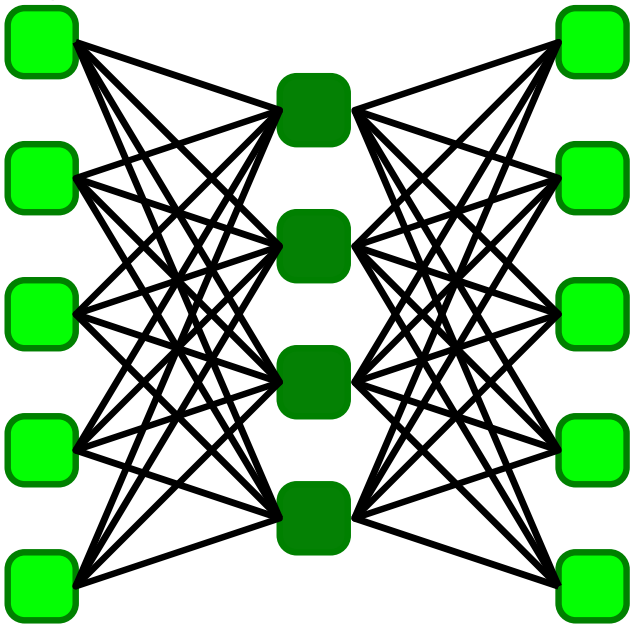
Multilayer autoencoder (1)

- We start to train only one hidden layer using images dataset :



Multilayer autoencoder (2)

- Using compressed representation of images using the first hidden layer, we train a second hidden layer.



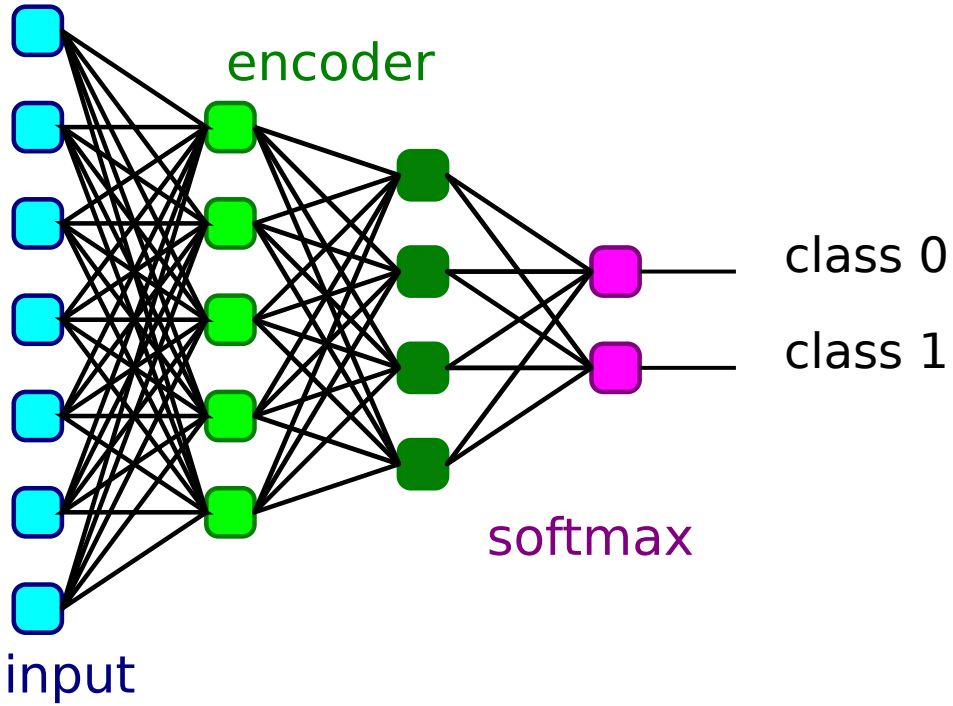
- We then repeat this process until the targeted depth is reached.

Multilayer autoencoder (3)

- Learning layer after layer is easy and don't need huge hardware
  - However, the weights are forced to be symetric
- Learning the whole autoencoder at once allows weights to be different
  - Even is the autoencoder architecture is still symetric

Classification using an autoencoder representation

- All layers except the output layer are trained using non labeled data.
- Output layer is trained using labels (softmax layer).



- The whole network is fine tuned.

Multilayer autoencoder in keras

```
encoded = Dense(128, activation='relu')(input_image)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
encoded = Dense(10, activation='relu')(encoded)

decoded = Dense(32, activation='relu')(encoded)
decoded = Dense(64, activation='relu')(decoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)
```

Multilayer encoder - decoder in keras

- encoder

```
encoderModel = Model(inputImage, encoded)
# encode the whole training dataset
xTrainEncoded = encoderModel.predict(xTrain)
# or encode only one data
latentData = encoderModel.predict(np.array([data,]))
```

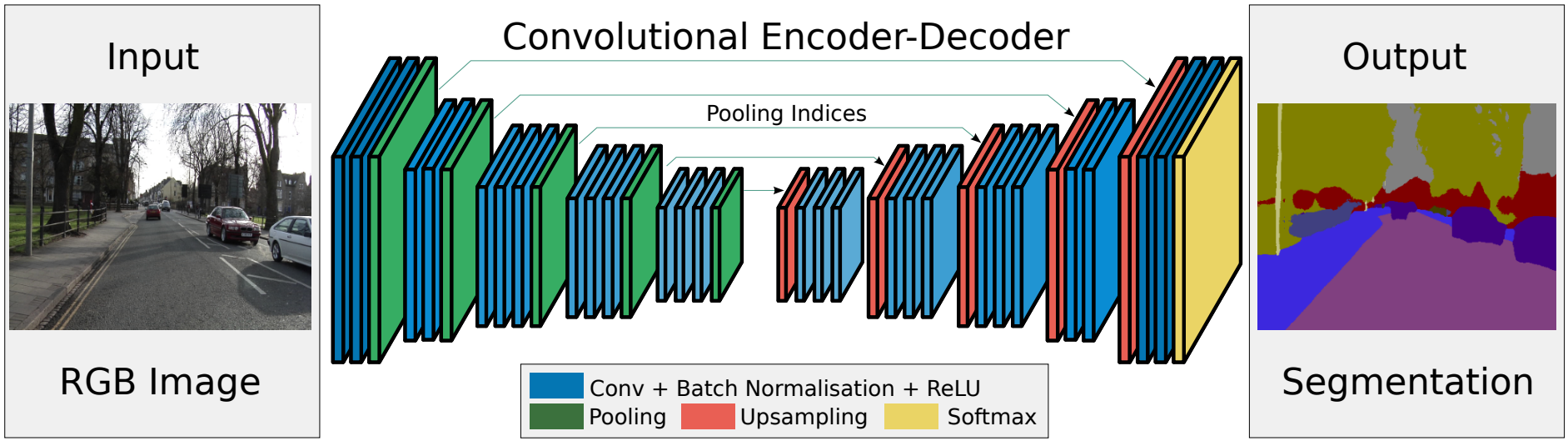
- decoder

```
inputLatent = Input(shape=(10,))
decoder = autoencoder.layers[-4](inputLatent)
decoder = autoencoder.layers[-3](decoder)
decoder = autoencoder.layers[-2](decoder)
decoder = autoencoder.layers[-1](decoder)
decoderModel = Model(inputLatent,decoder)
# decode only one data
decodedData = decoderModel.predict(np.array([latentData,]))
```

Outline

Convolution Autoencoder

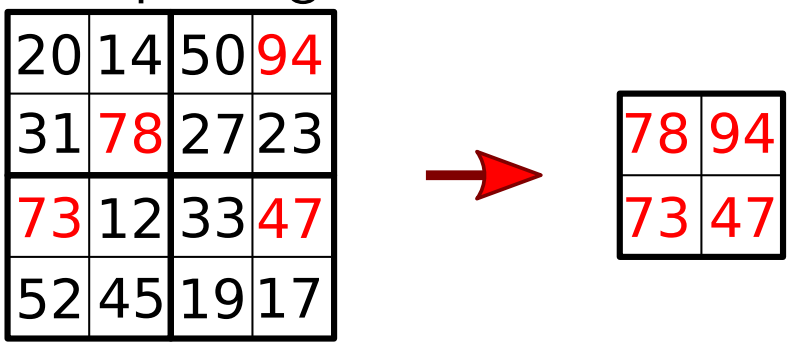
- simply replace dense layers by convolutional layers
  - in keras, replace `Dense` by `Conv2D`
- application to segmentation : SegNet



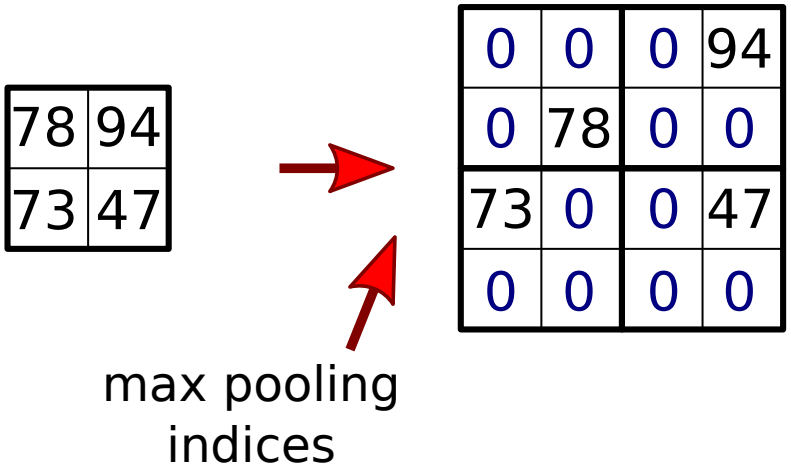
from Badrinarayanan etal, arXiv :1511.00561v3

Pooling - Upsampling

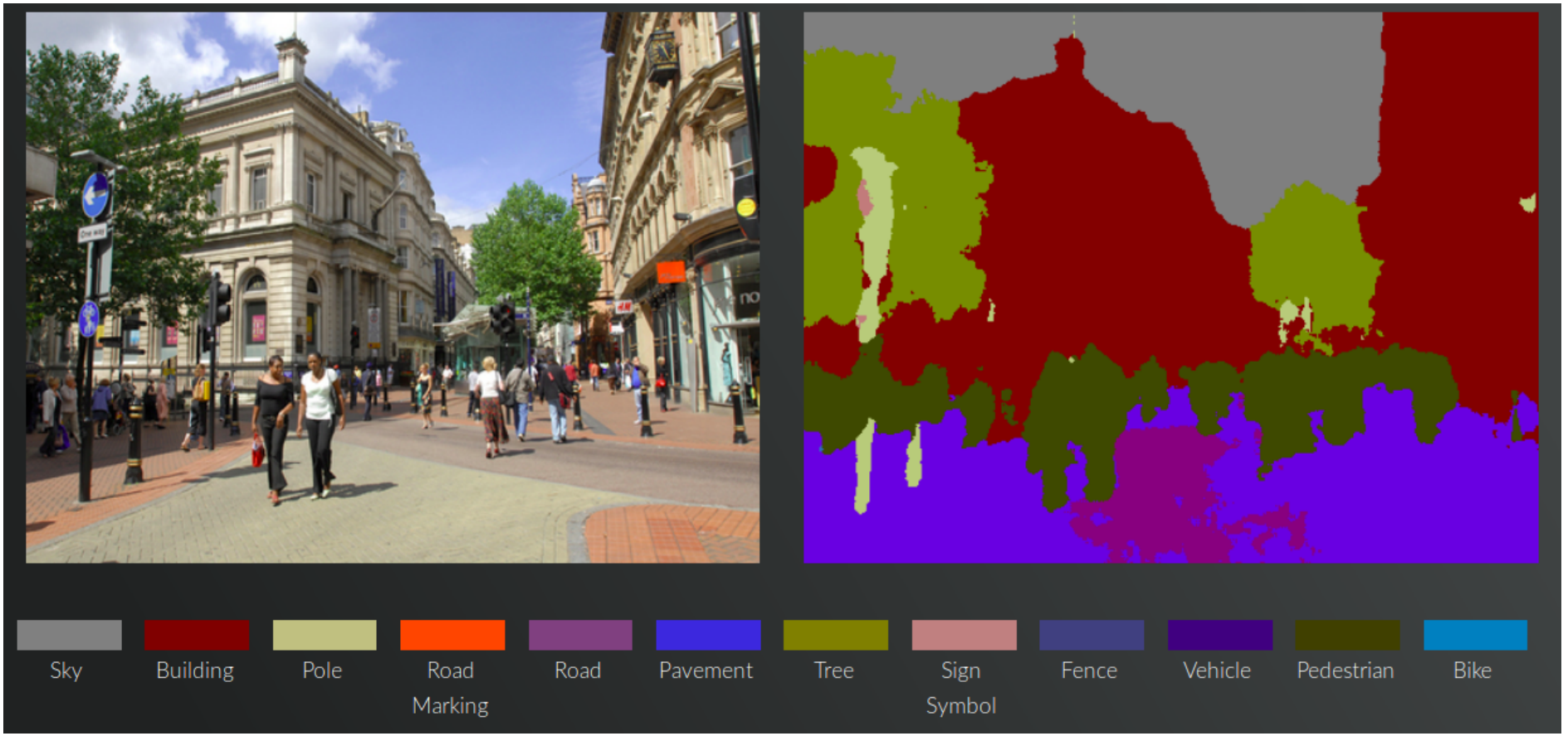
- Max pooling



- Upsampling



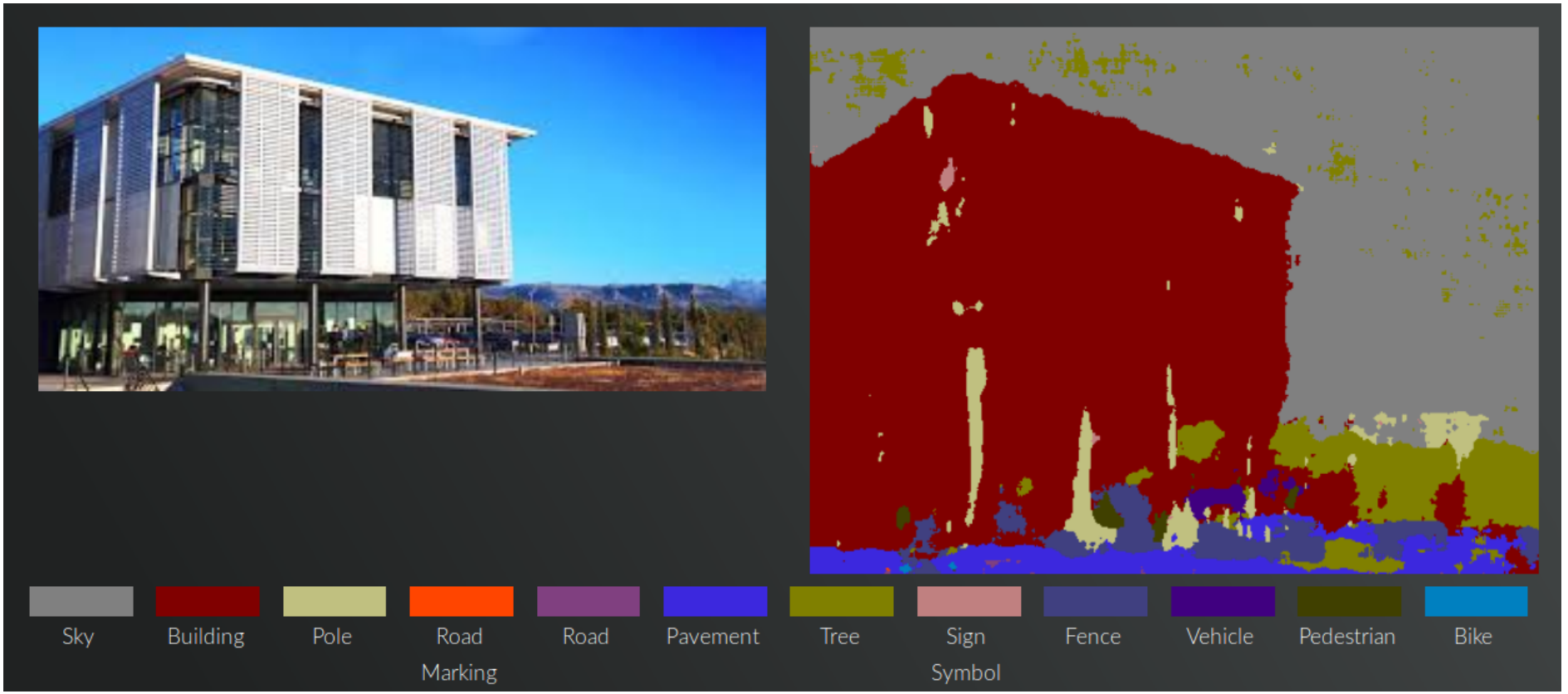
SegNet example I



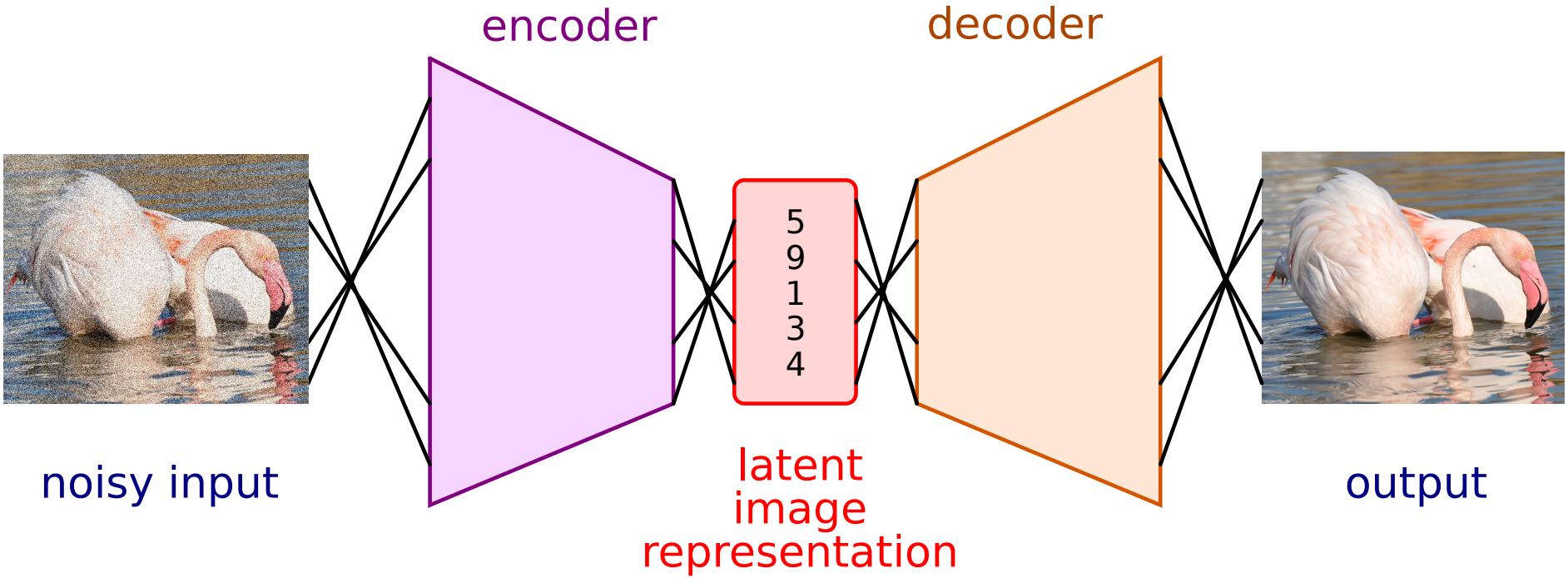
also on youtube : `https://www.youtube.com/watch?v=CxanE_W46ts`

SegNet example II

Demo page no more available but code available from many sources.



Denoising an image



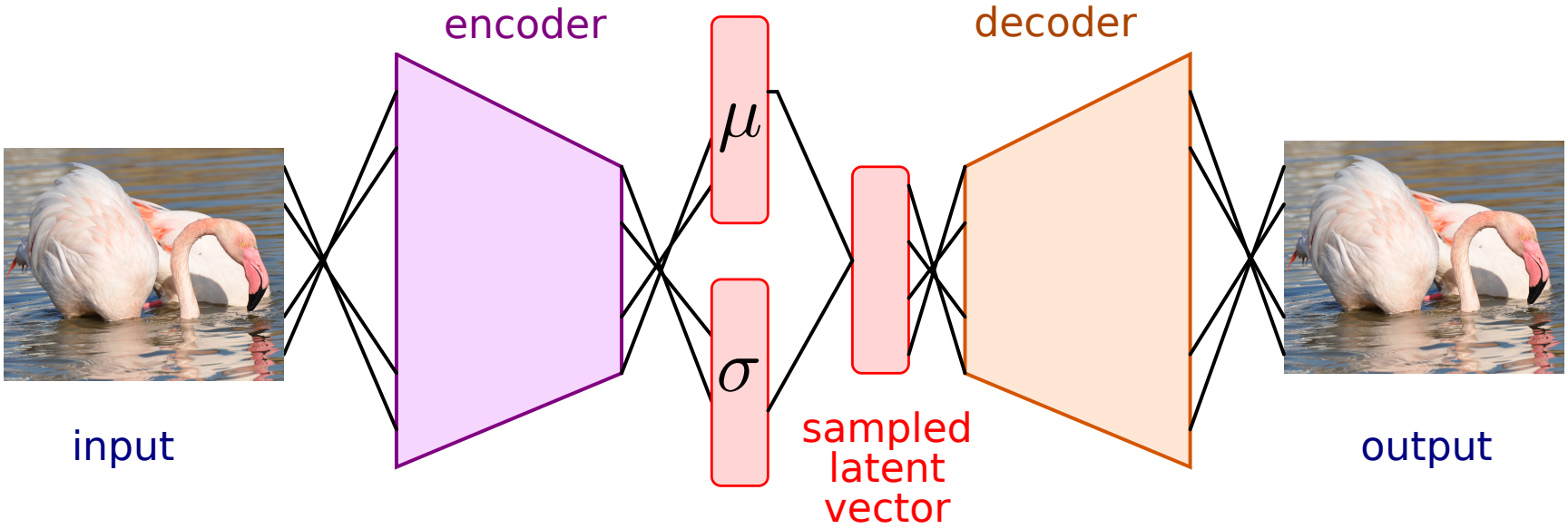
Experiment : representing an image using autoencoder

- classification task
- representation of image using autoencoder
- classification with any standard classification algorithm (MLP, SVM, ...)

Outline

Variational Autoencoder (VAE)

- Autocoders learn a latent representation of data
- Variational autoencoders learn the parameter of a probability function modeling the latent data
  - ex : latent parameters μ and σ, assuming the distribution is gaussian
- It is thus a generative model able to generate similar data
  - regularisation of distribution parameters (mean and variance) in order to ensure that :
    - 2 samples similar in the latent space should have similar content when decoded
    - a data sampled from the latent space should generate meaningful content when decoded



Reparameterization trick

- problem with sampling : how to compute the gradient ?
- trick : consider variable z :

$$z = \mu + \sigma \odot \epsilon$$

  where $\epsilon \sim \mathcal{N}(0, 1)$
- learn μ and σ
- z is generated from μ, σ and ϵ
- backpropagation of gradients through μ and σ

VAE in practice : example using MNIST

VAE in practice : encoder

```
# this is an example of architecture.
# Feel free to add other layers
dimLatent = 28
inputImage = Input(shape=(784,))
encoded = Dense(256, activation='relu')(inputImage)
zMu = Dense(dimLatent)(encoded)
zLogSigma = Dense(dimLatent)(encoded)
```

VAE in practice : sample generation

```
from tensorflow.keras import backend as bk

def sampling(args):
    mu, logSigma = args
    epsilon = bk.random_normal(shape=(bk.shape(mu)[0], dimLatent), m
    # z = mu + sigma * epsilon
    return mu + bk.exp(logSigma) * epsilon

# Lambda layer for wrapping the sampling function
z = Lambda(sampling)([zMu, zLogSigma])

#encoder model
encoder = Model(inputImage, [zMu, zLogSigma, z])
```

VAE in practice : decoder and whole model

```
# symmetric to the encoder
latent = Input(shape=(dimLatent,))
decoded = Dense(256, activation='relu')(latent)
decoded = Dense(size1, activation='sigmoid')(decoded)
decoder = Model(latent, decoded)

# whole model to be trained
# z in the element at index 2 from [zMu, zLogSigma, z]
outputImage = decoder(encoder(inputImage)[2])
vae = Model(inputImage, outputImage)
```

VAE in practice : loss function

```
from tensorflow.keras.losses import mean_squared_error
## Loss function is composed of 2 terms:
# 1. reconstruction loss
reconstruction_loss = mean_squared_error(inputImage, outputImage)
reconstruction_loss *= size1
print(reconstruction_loss)
# 2. Kullback-Leibner loss
kl_loss = 1 + zLogSigma - bk.square(zMu) - bk.exp(zLogSigma)
kl_loss = bk.sum(kl_loss, axis=-1)
kl_loss *= -0.5
print(zMu)
print(kl_loss)
# vae loss
vae_loss = bk.mean(reconstruction_loss + kl_loss)
vae.add_loss(vae_loss)
```

Examples using MNIST

- original test images and decodings. Latent dimension = 28

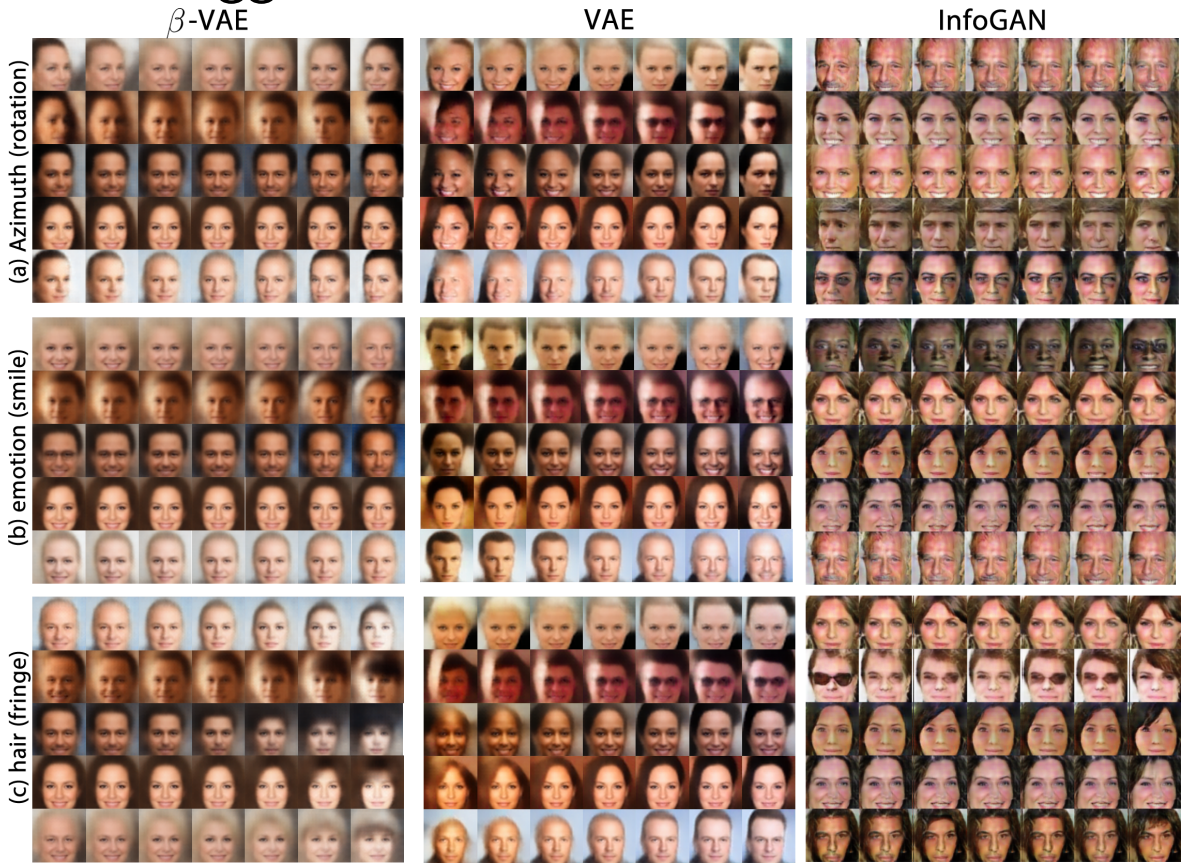
- from a test image, generate several latent representation and decodings

- interpolations between latent representation of 2 digits and decodings

- random latent representations and decodings

Disentangled Variational Autoencoder (β-VAE)

- ensure no correlation between latent variables
- param. β : weight to Kullback-Leibler divergence in the loss fn
- From Higgins etal, ICLR 2017 `https://openreview.net/pdf?id=Sy2fzU9gl`



- implementation :

`https://github.com/Knight13/beta-VAE-disentanglement` or

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