Deep learning for medical imaging school 2021

Hands-on session

2021

Autoencoders

By

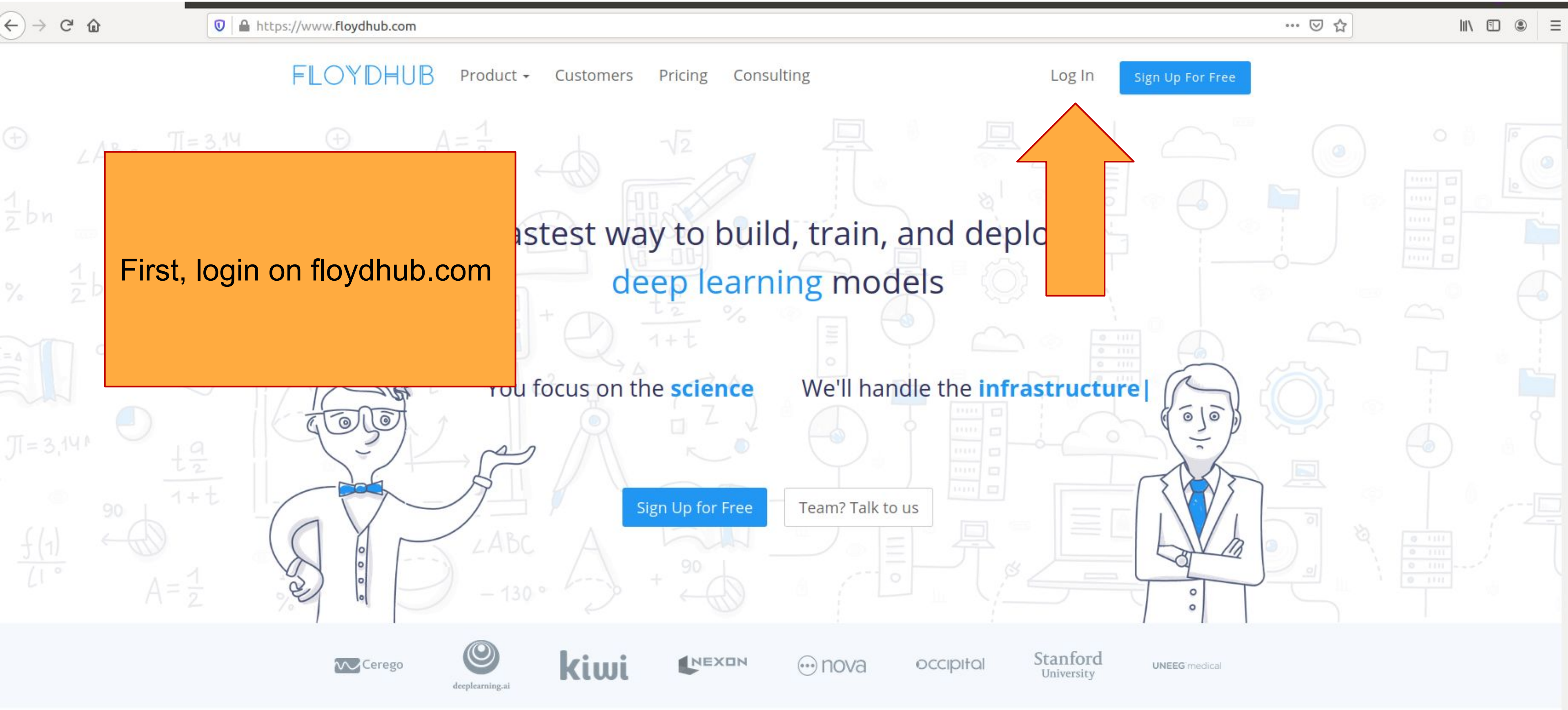
Pierre-Marc Jodoin, Nathan Painchaud

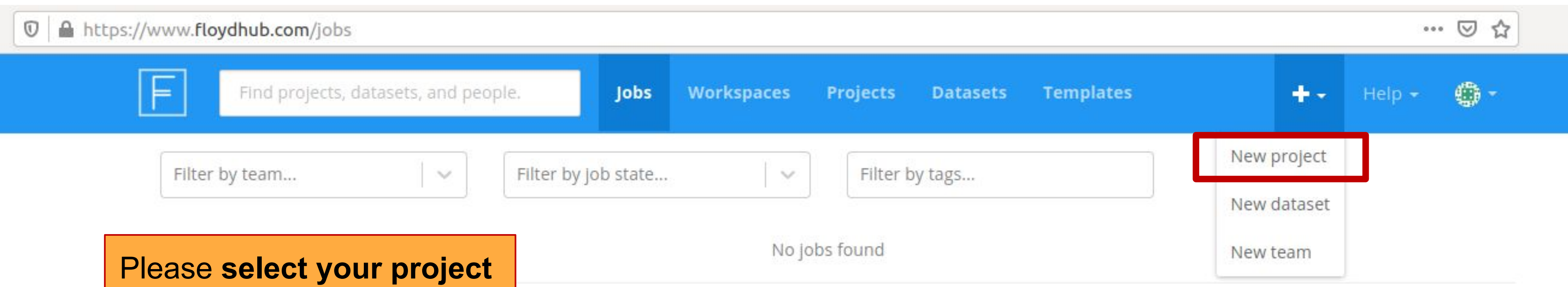
With the support of

Thomas Grenier, Olivier Bernard, Marco A. Armenta, Thierry Judge.

Setup your Floyd hub environment

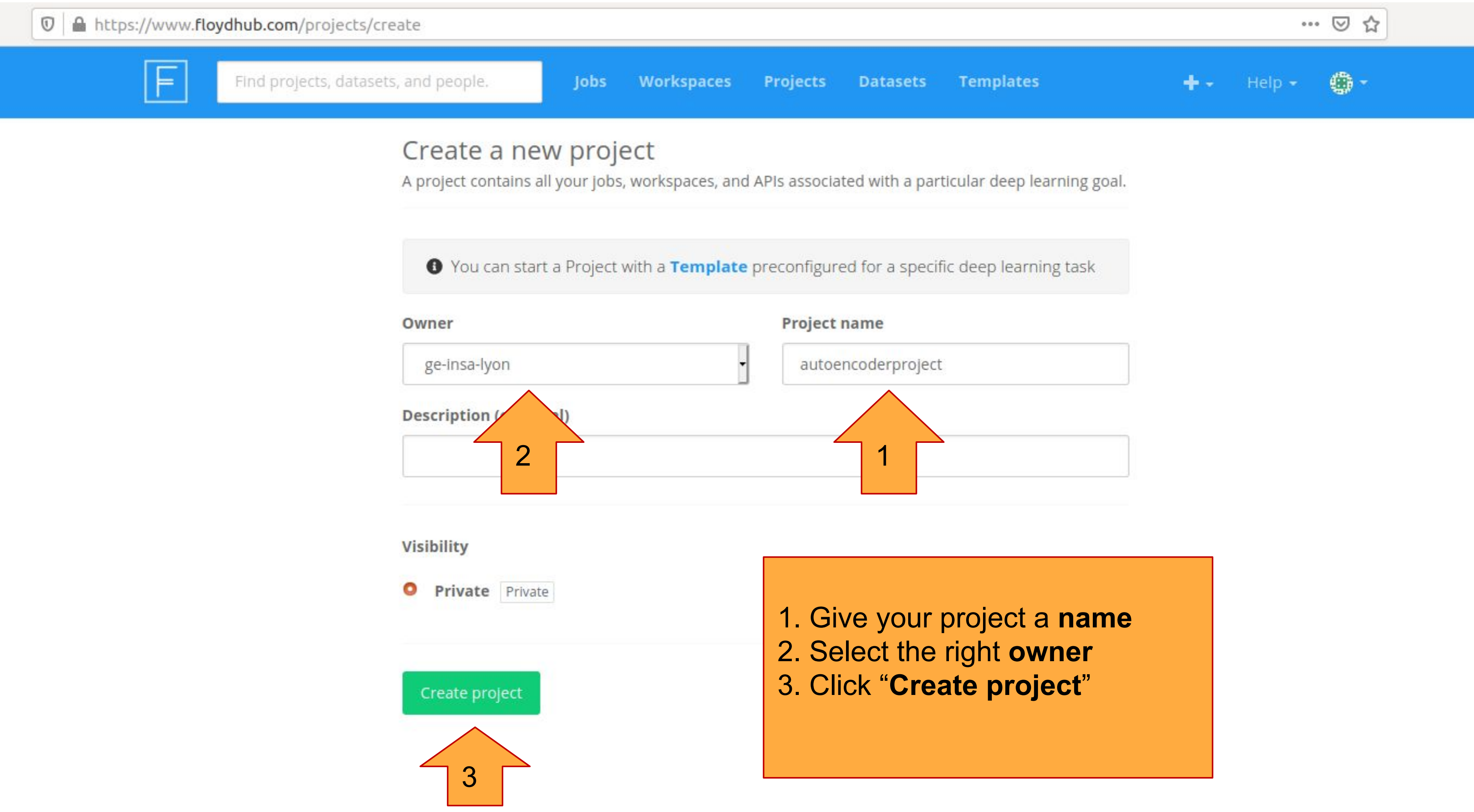
First, login on floydhub.com



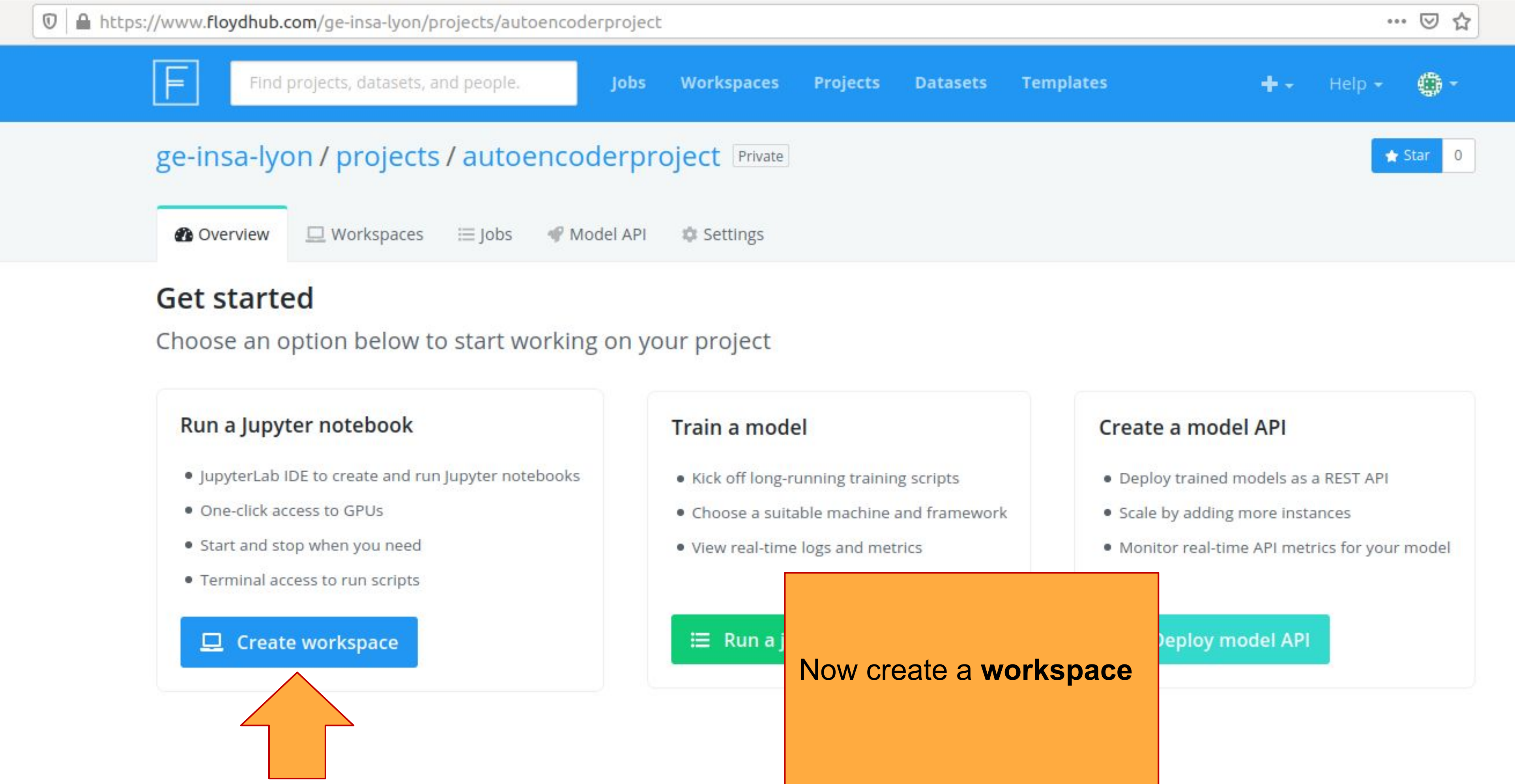


Please **select your project (you should normally have one).** You may skip the next page.

If you do not already have a project, **please create one.**



1. Give your project a **name**
2. Select the right **owner**
3. Click “**Create project**”



Now create a **workspace**

https://www.floydhub.com/ge-insa-lyon/projects/autoencoderproject/workspaces

Find projects, datasets, and people.

Jobs Workspaces Projects Datasets Templates Help

ge-insa-lyon / projects / autoencoderproject Private

Star 0

Overview Workspaces Jobs Model API Settings

0 workspaces

Create Workspace

No workspaces found.

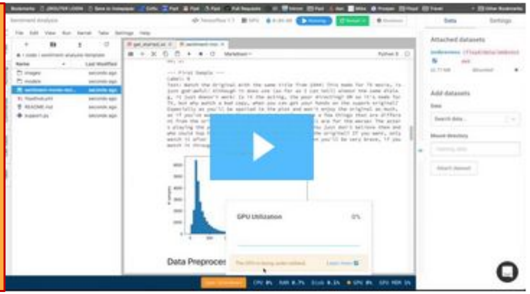
Build models with Workspaces

Workspaces are configurable, interactive development environments built for deep learning and machine learning.

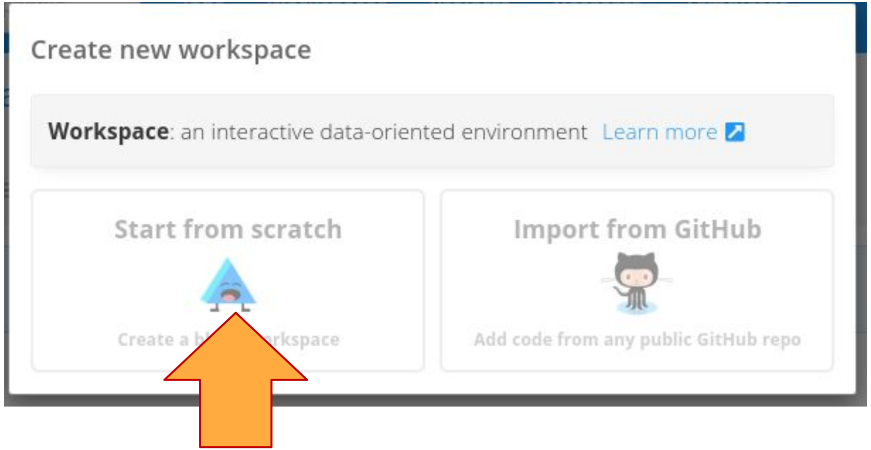
- Create and run Jupyter notebooks
- Toggle between GPU and CPU-powered machines
- Attach datasets to your workspace
- Terminal access to run scripts
- Start and stop when you need

Explore more features of Workspaces

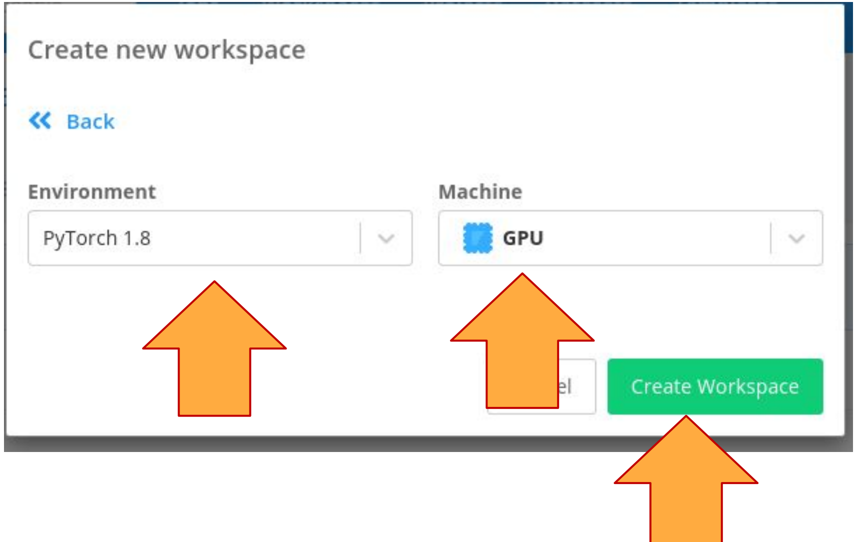
Click on “**Create workspace**”

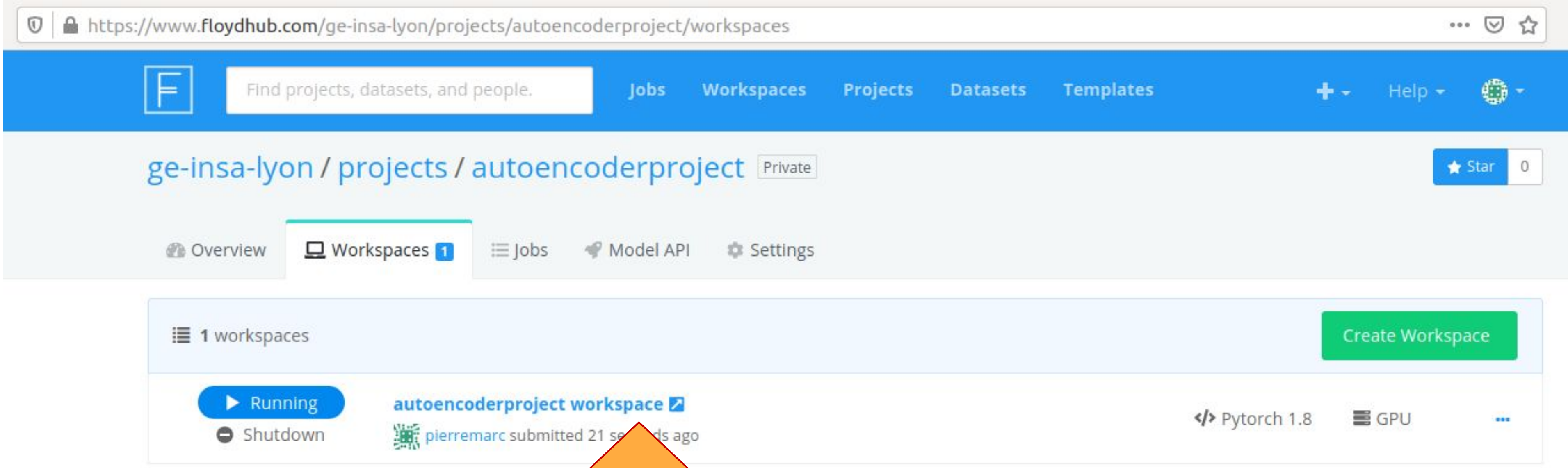


Select “**Start from scratch**”

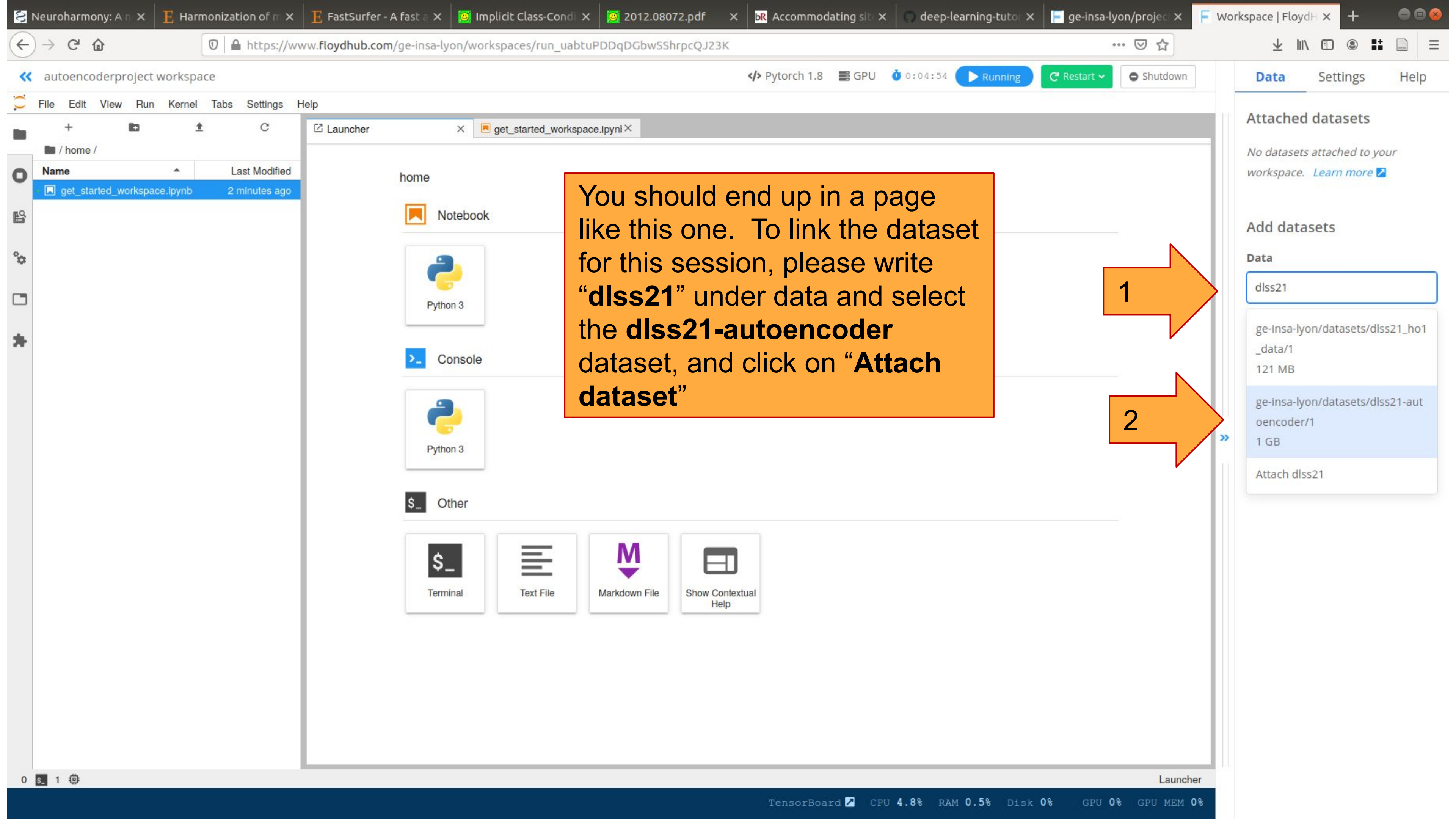


Then “**PyTorch 1.8**”, “**GPU**” and click “**Create Workspace**”





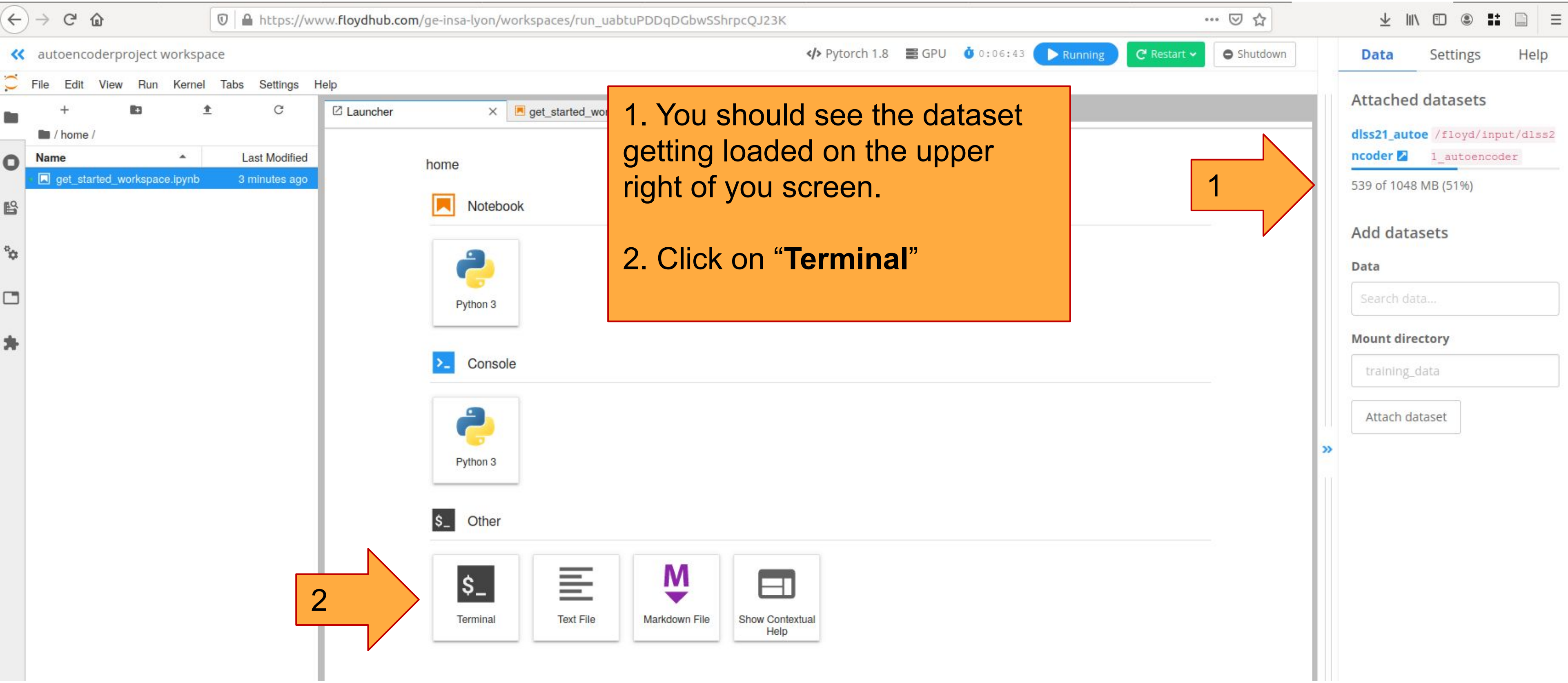
You should see your new workspace. So **click on it**.

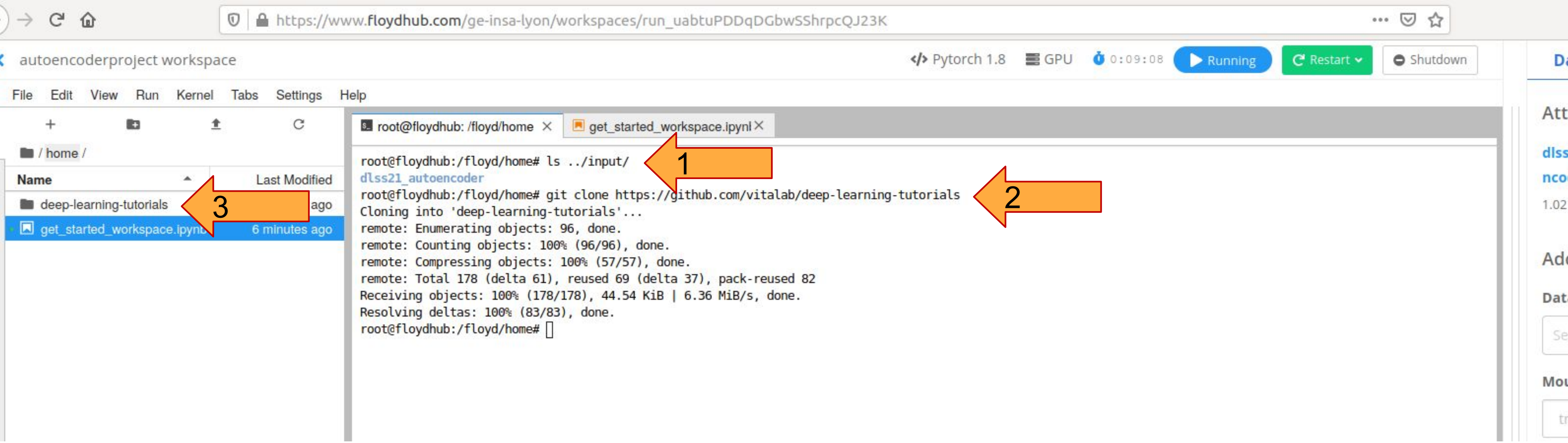


You should end up in a page like this one. To link the dataset for this session, please write “**dlss21**” under data and select the **dlss21-autoencoder** dataset, and click on “**Attach dataset**”
1
2
autoencoderproject workspace
Attached datasets
No datasets attached to your workspace. Learn more
Add datasets
Data
dlss21
ge-insa-lyon/datasets/dlss21_ho1_data/1
121 MB
ge-insa-lyon/datasets/dlss21-autoencoder/1
1 GB
Attach dlss21

1. You should see the dataset getting loaded on the upper right of you screen.

2. Click on “**Terminal**”

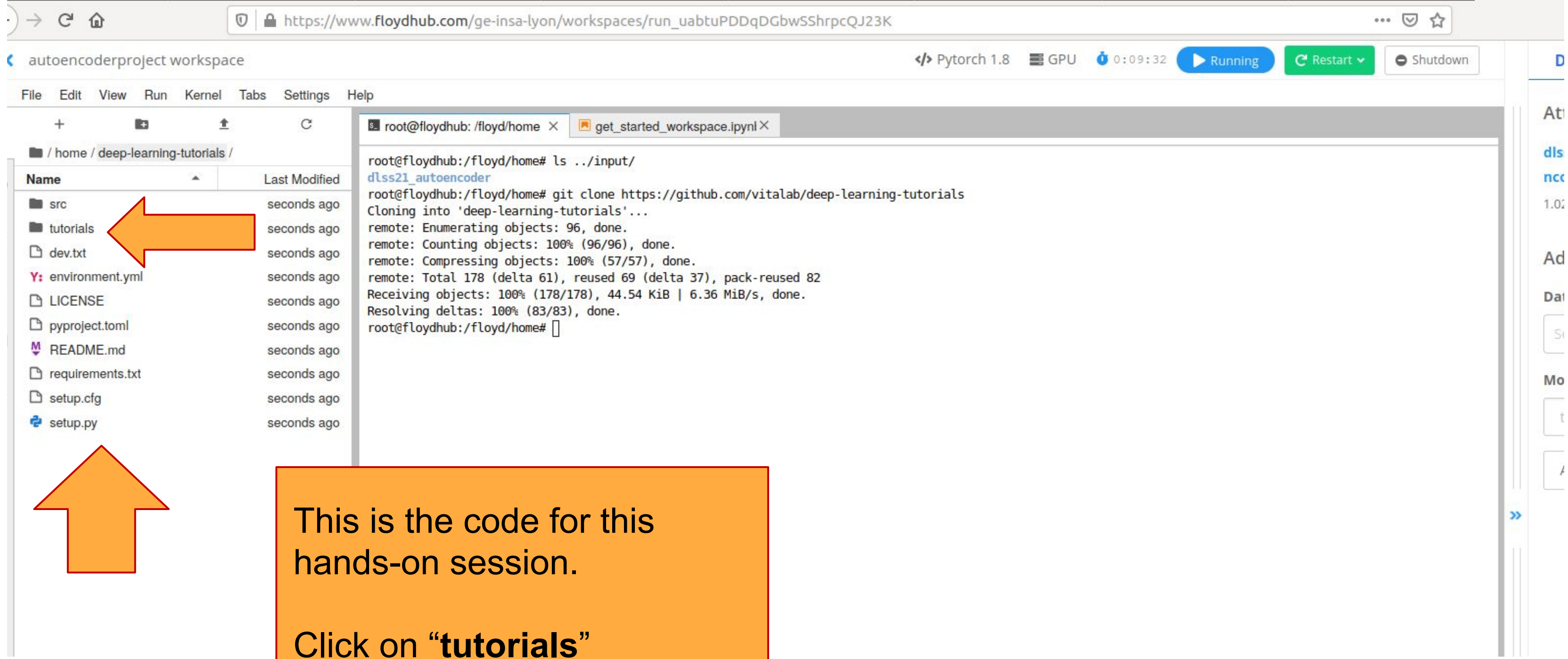




1. Type “`ls ../input/`” in the terminal. You should see the autoencoder dataset.

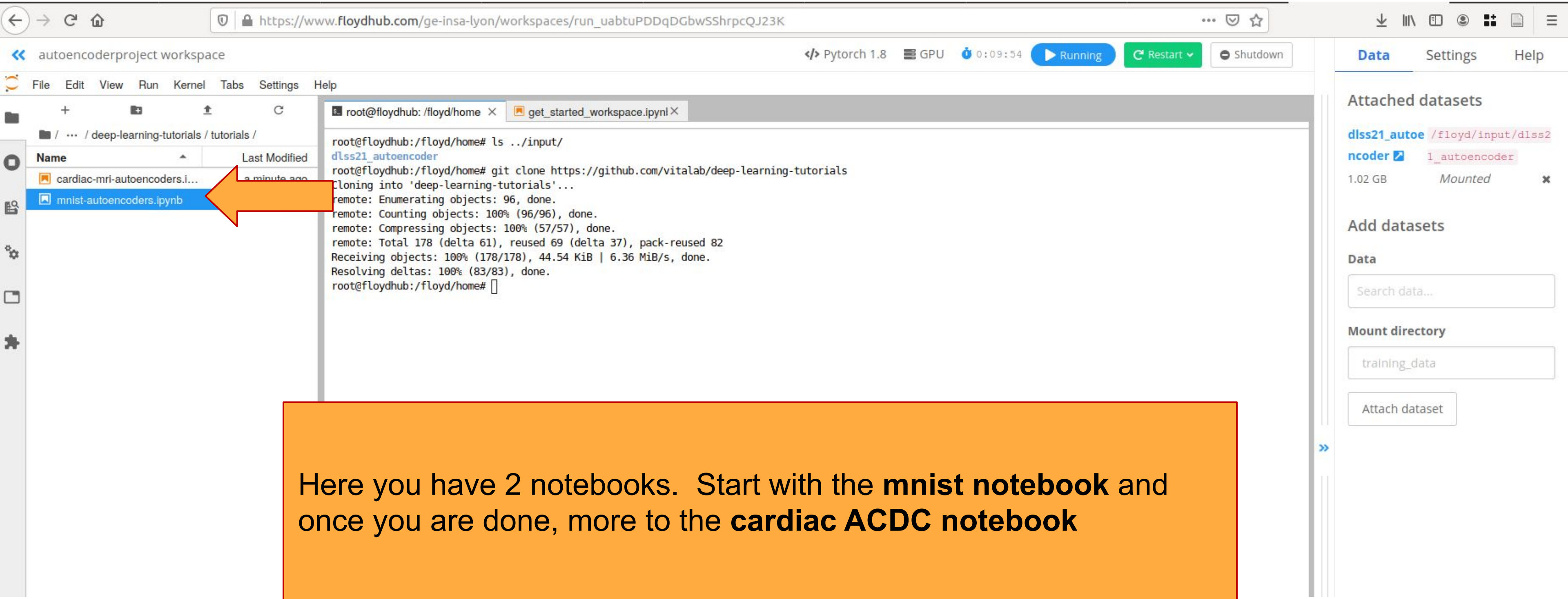
2. Type “`git clone https://github.com/vitalab/deep-learning-tutorials`” to download the code

3. A new **`deep-learning-tutorials`** folder should appear. Click on that folder



This is the code for this hands-on session.

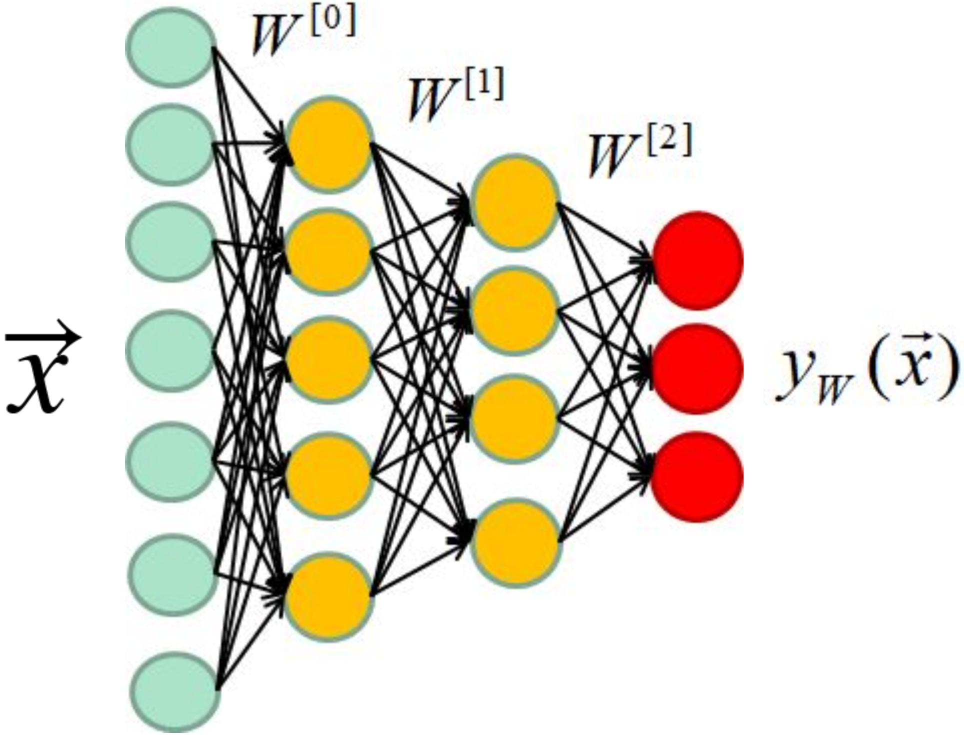
Click on “**tutorials**”



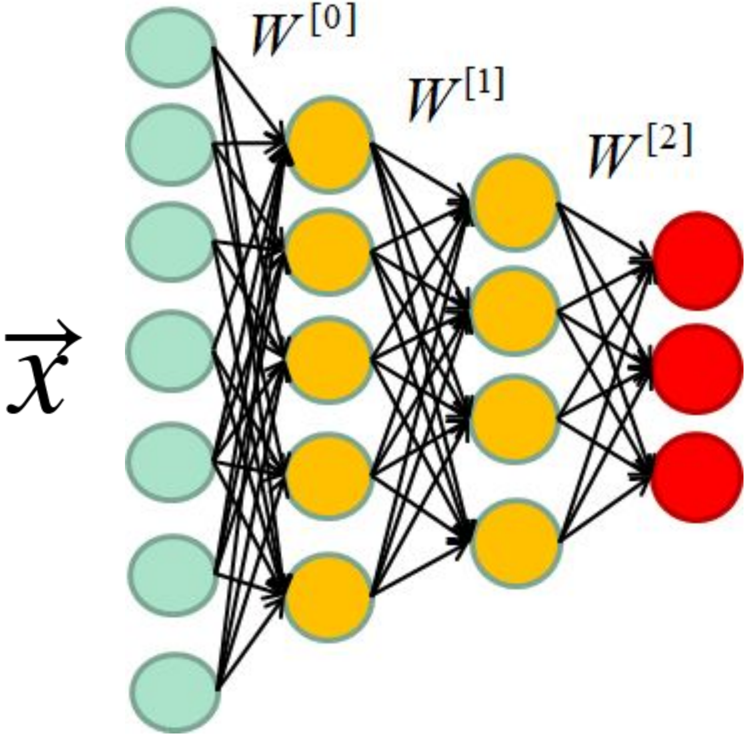
Here you have 2 notebooks. Start with the **mnist notebook** and once you are done, more to the **cardiac ACDC notebook**

Overview of the Autoencoder hands-on

(very) quick recap


$\vec{x}$
$W^{[0]}$
$W^{[1]}$
$W^{[2]}$
$y_W(\vec{x})$

Annotated dataset: $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

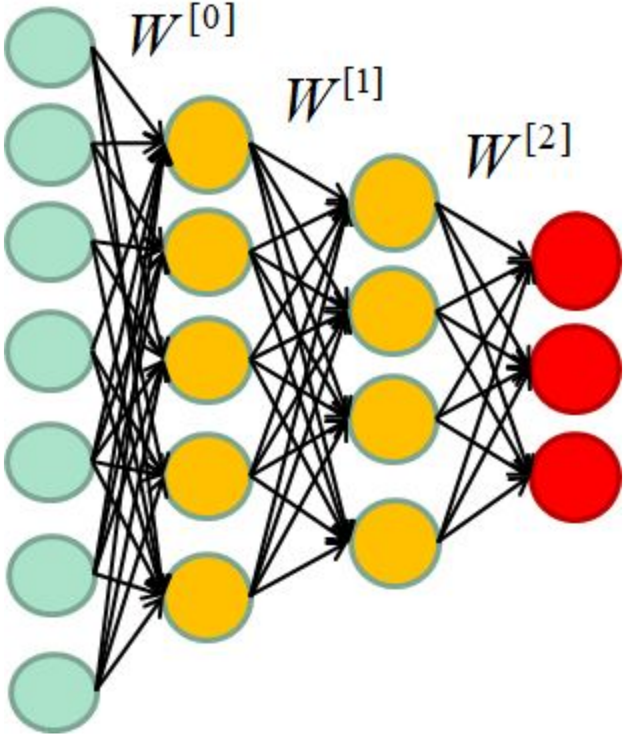


$y_W(\vec{x}) \dashrightarrow L(y_W(\vec{x}), D)$

Forward pass

Annotated dataset: $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

$\vec{x}$

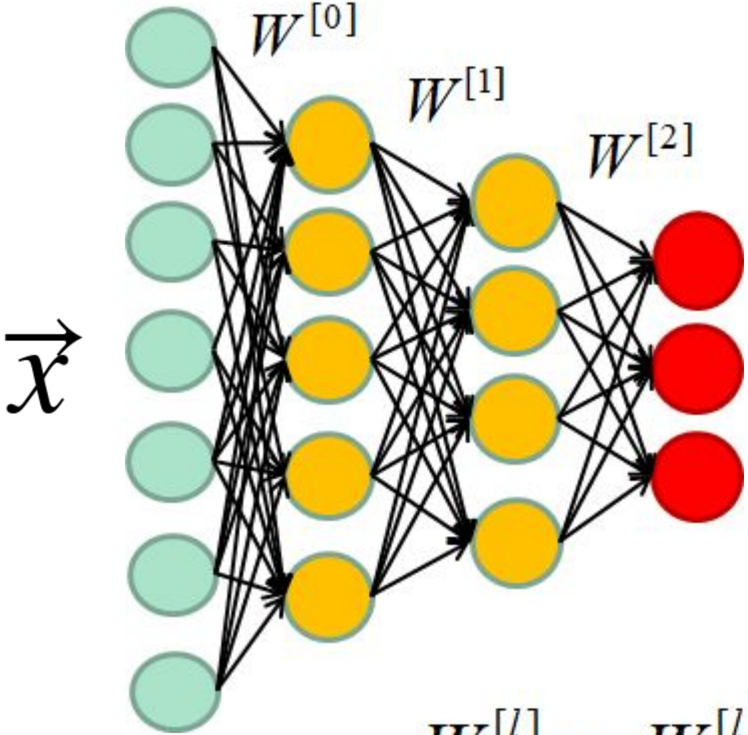


$y_W(\vec{x}) \dashrightarrow L(y_W(\vec{x}), D)$

$$\frac{\partial(L(y_W(\vec{x})))}{\partial W_j^{[l]}}$$

Backward pass

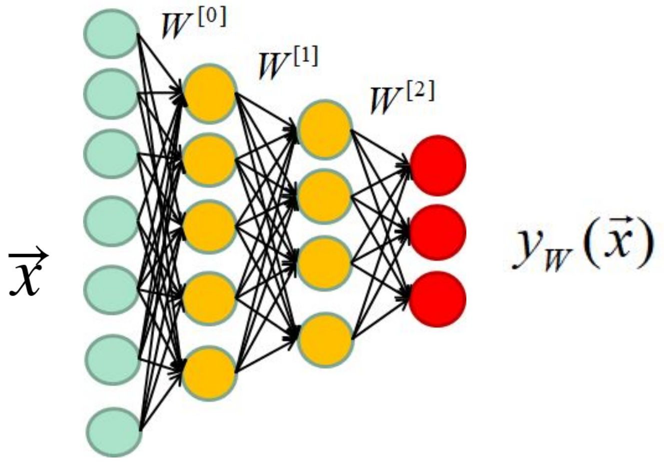
Annotated dataset: $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \ldots, (\vec{x}_N, t_N)\}$

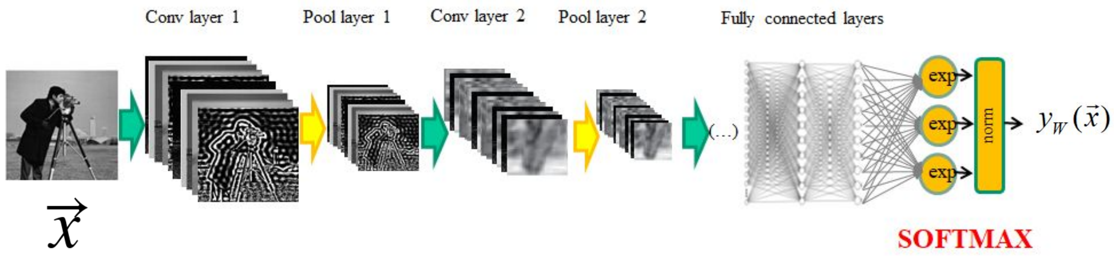


$y_W(\vec{x}) \dashrightarrow L(y_W(\vec{x}), D)$

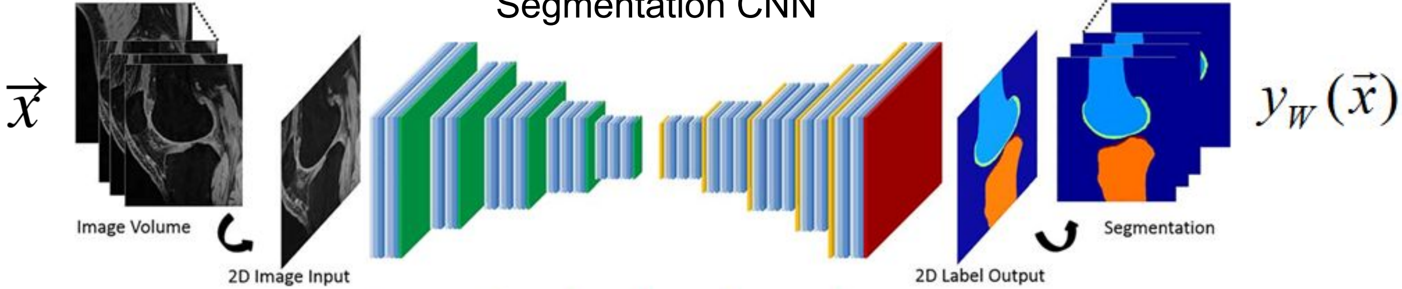
Gradient descent

$$W_j^{[l]} \leftarrow W_j^{[l]} - \eta \frac{\partial \left(L(y_W(\vec{x})) \right)}{\partial W_j^{[l]}}$$





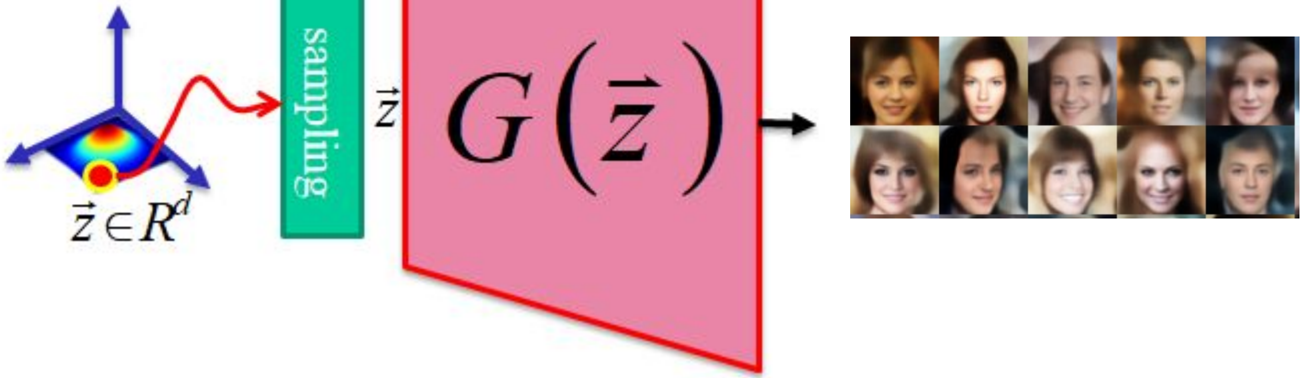
Segmentation CNN



Fang Liu, etal., Deep convolutional neural network and 3D deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. in Magnetic resonance in medicine 2018 DOI:10.1002/mrm.26841

GAN

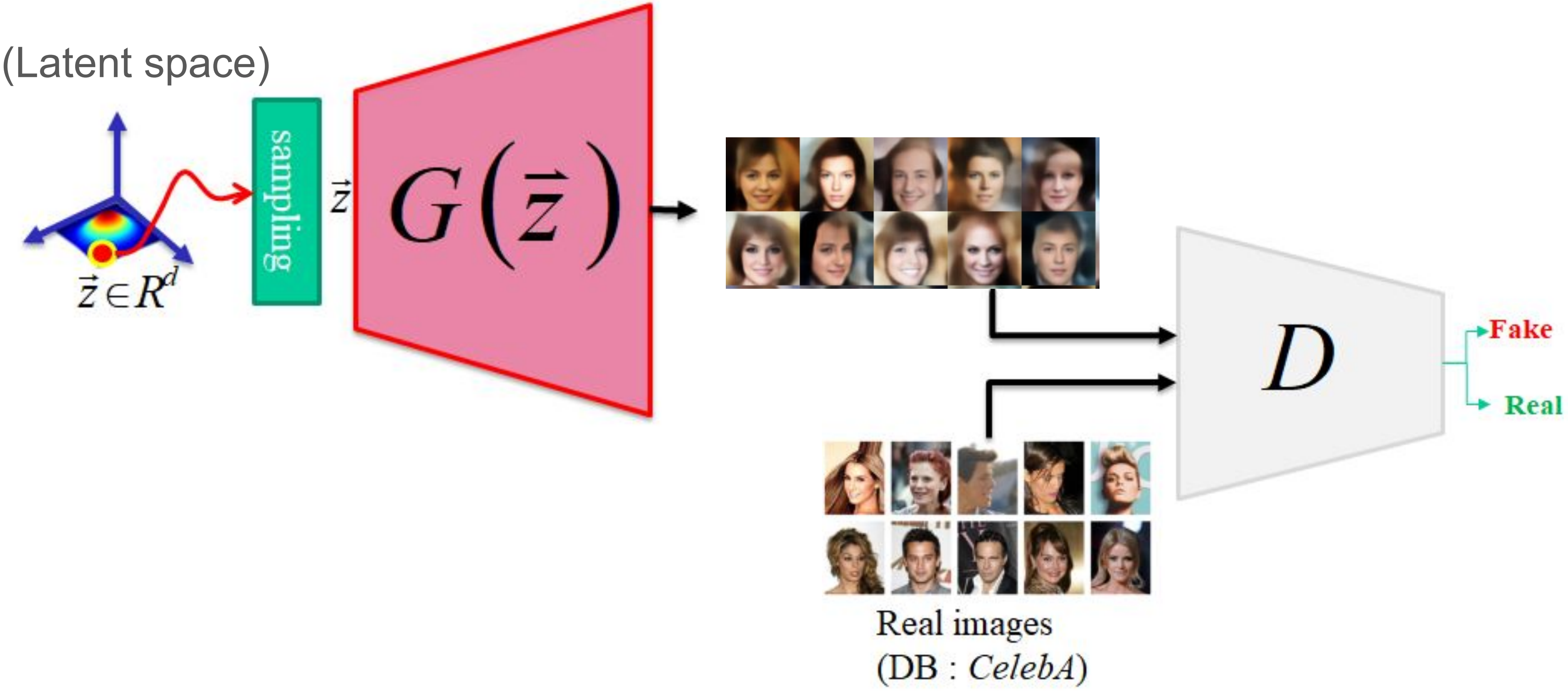
(Latent space)



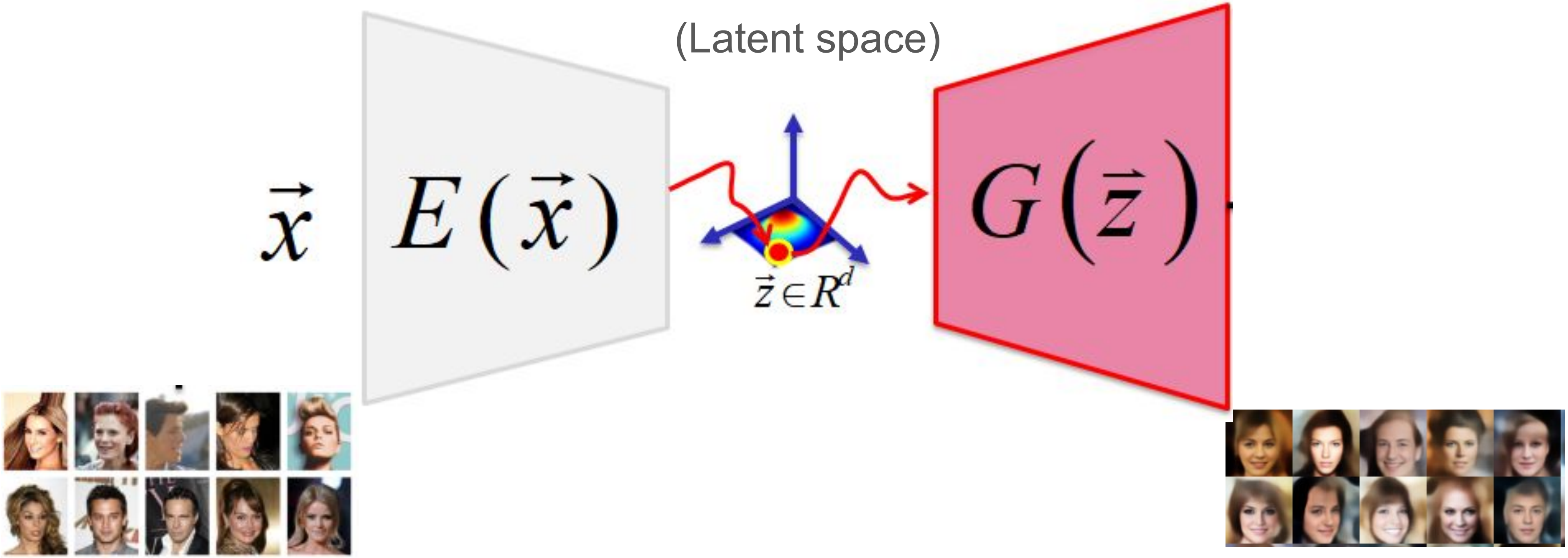
Loss?

Annotated dataset?

GAN

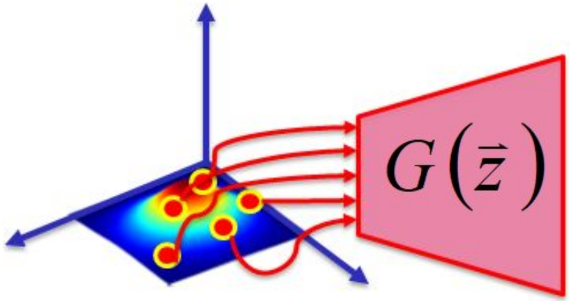

(Latent space)
$\vec{z} \in R^d$
sampling
$\vec{z}$
$G(\vec{z})$
D
Fake
Real
Real images
(DB : *CelebA*)

Autoencoders



Autoencoders (once training is over)

(Latent space)



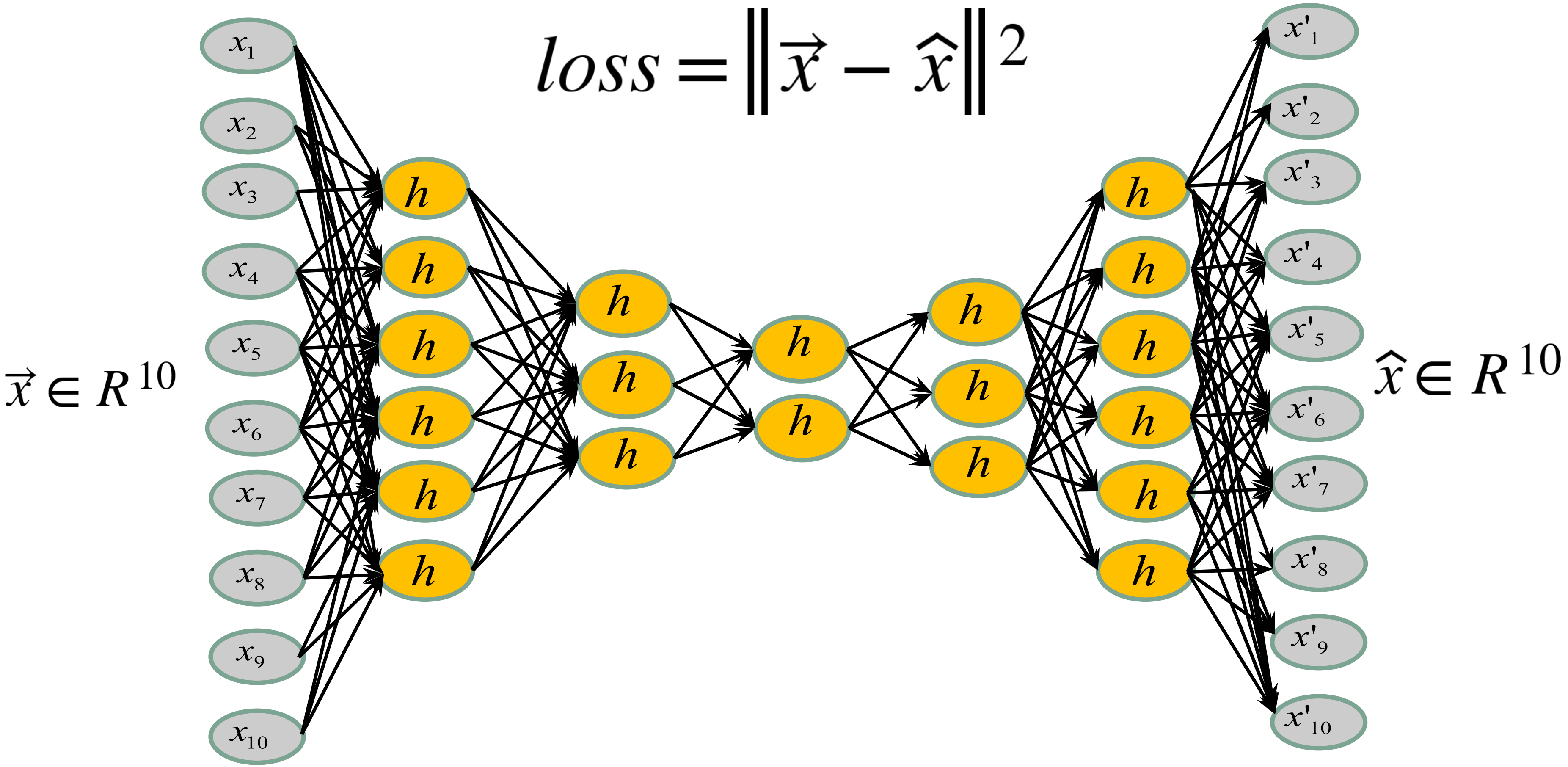
Summary

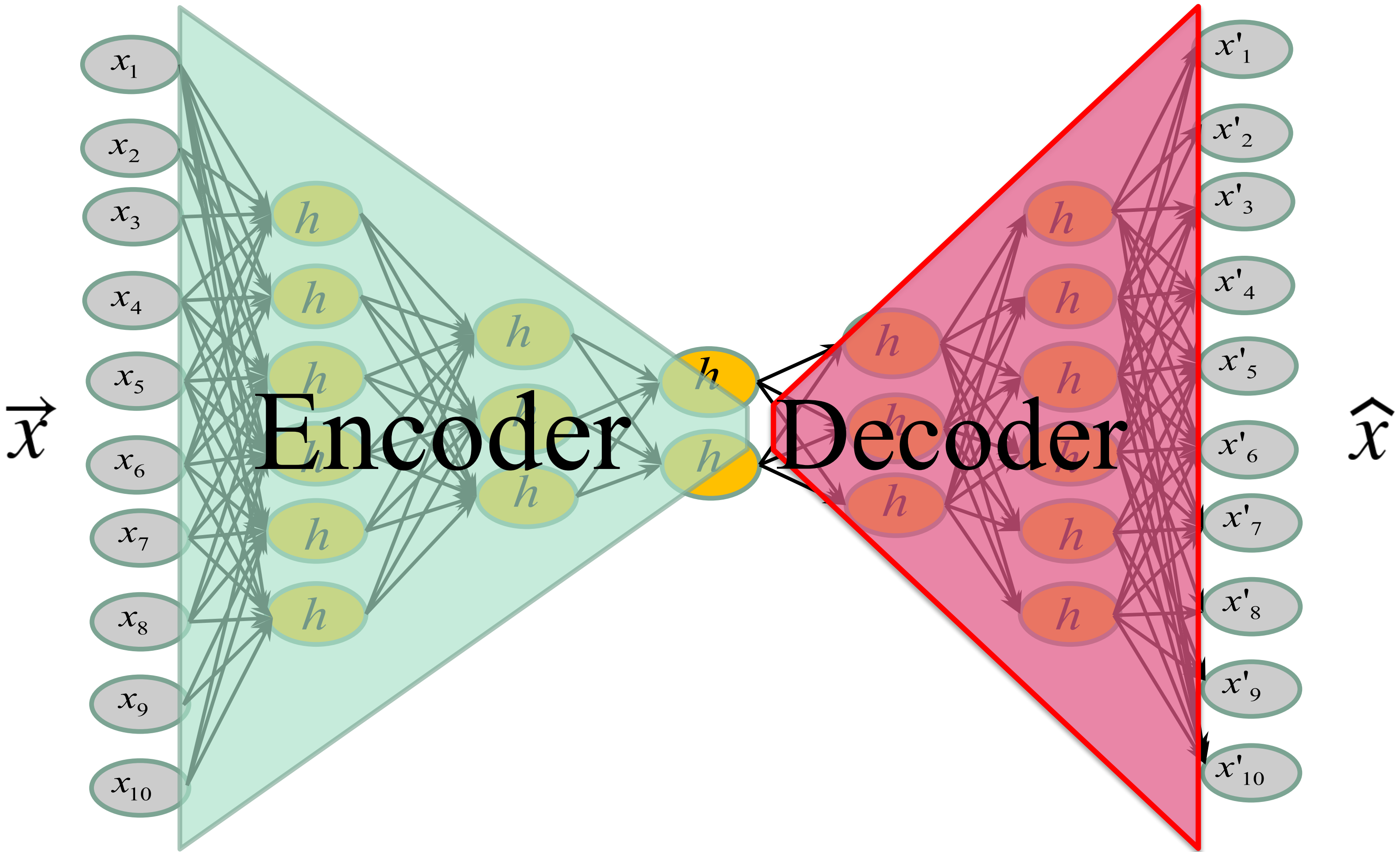
1. What are autoencoders and variational autoencoders?
2. How do they apply to MNIST (grayscale images)?
3. How do they apply to segmentation maps (ACDC cardiac labels)?

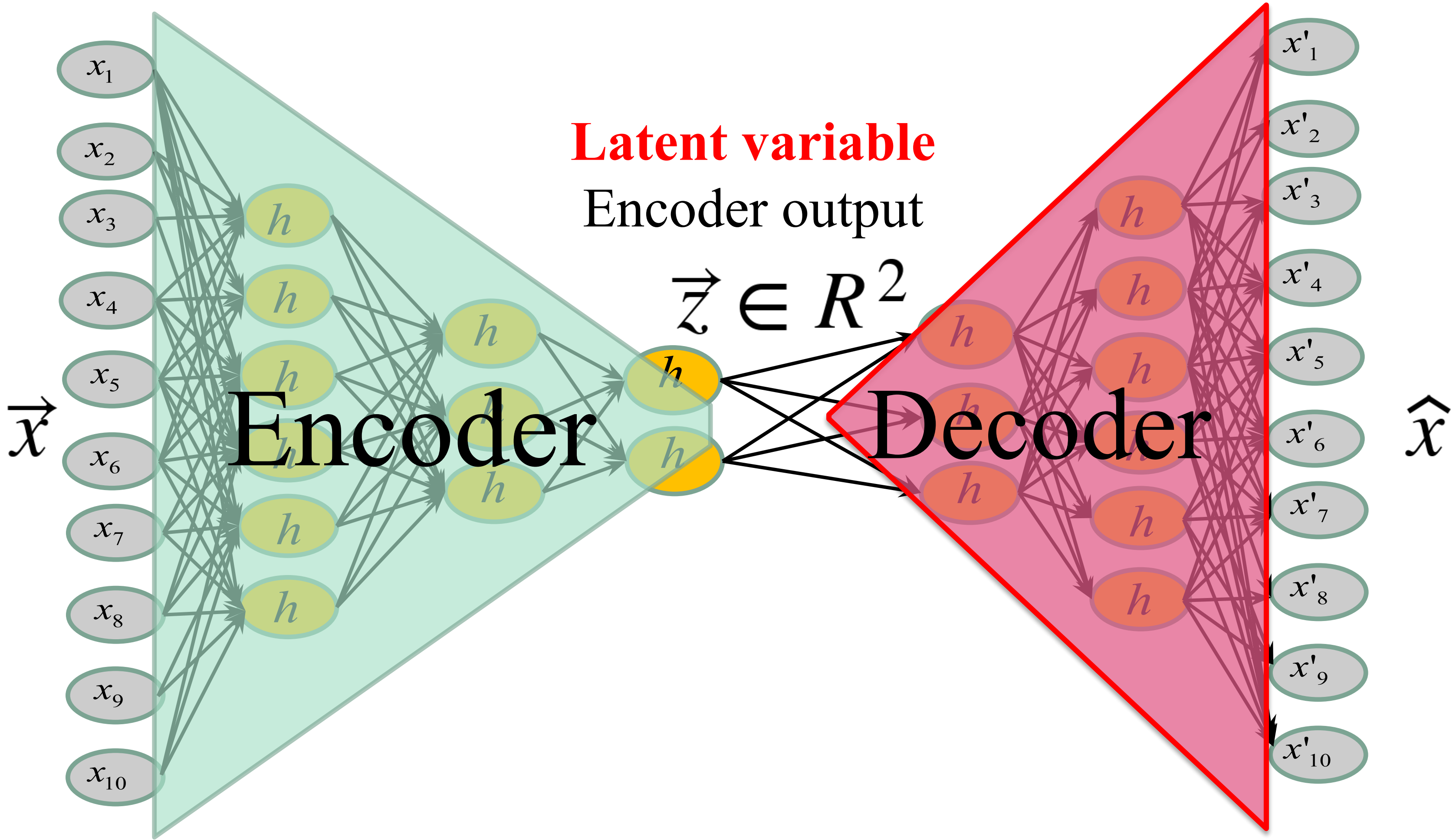
Goal : learn the latent representation of a set of data

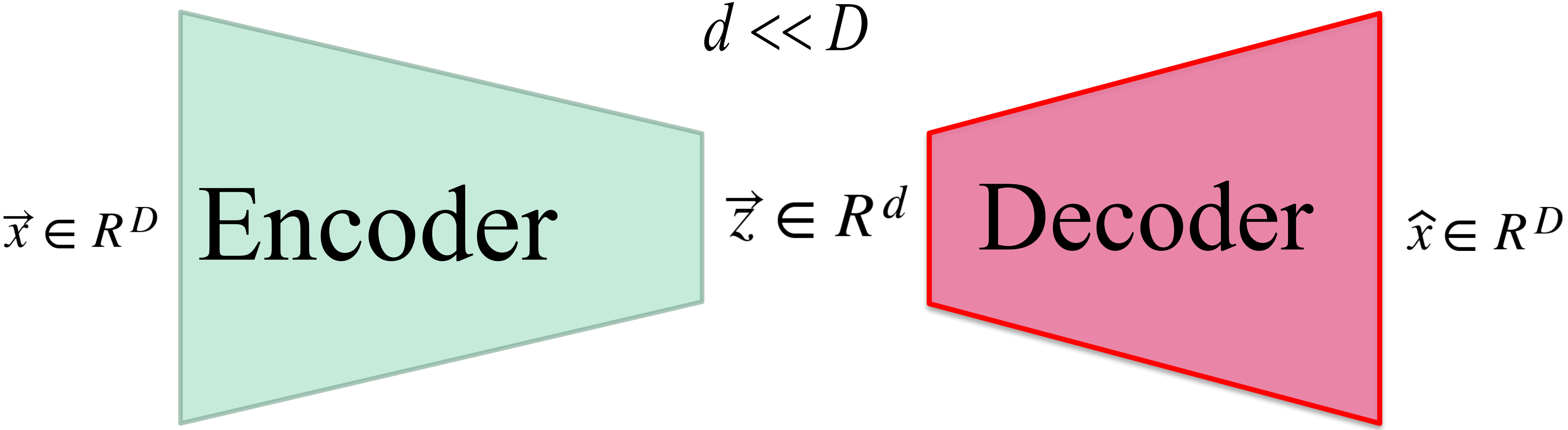
How : by training a Neural Net to output its own … input!

Note : if you are familiar with AE and VAE, you may skip the next couple of slides.


$loss = \|\vec{x} - \hat{x}\|^2$
$\vec{x} \in R^{10}$
$\hat{x} \in R^{10}$
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
h
x'_1 x'_2 x'_3 x'_4 x'_5 x'_6 x'_7 x'_8 x'_9 x'_{10}

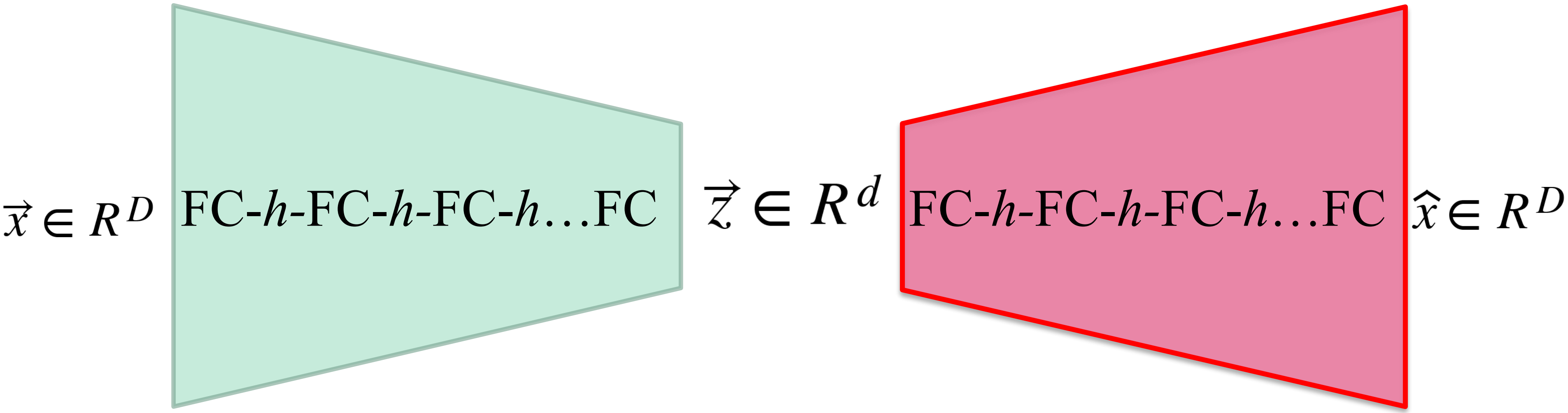

$\vec{x}$
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
Encoder
Decoder
x'_1
x'_2
x'_3
x'_4
x'_5
x'_6
x'_7
x'_8
x'_9
x'_{10}
$\hat{x}$


Latent variable
Encoder output
$\vec{z} \in R^2$
Encoder
Decoder
$\vec{x}$
$\hat{x}$
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
x_9
x_{10}
x'_1
x'_2
x'_3
x'_4
x'_5
x'_6
x'_7
x'_8
x'_9
x'_{10}
h

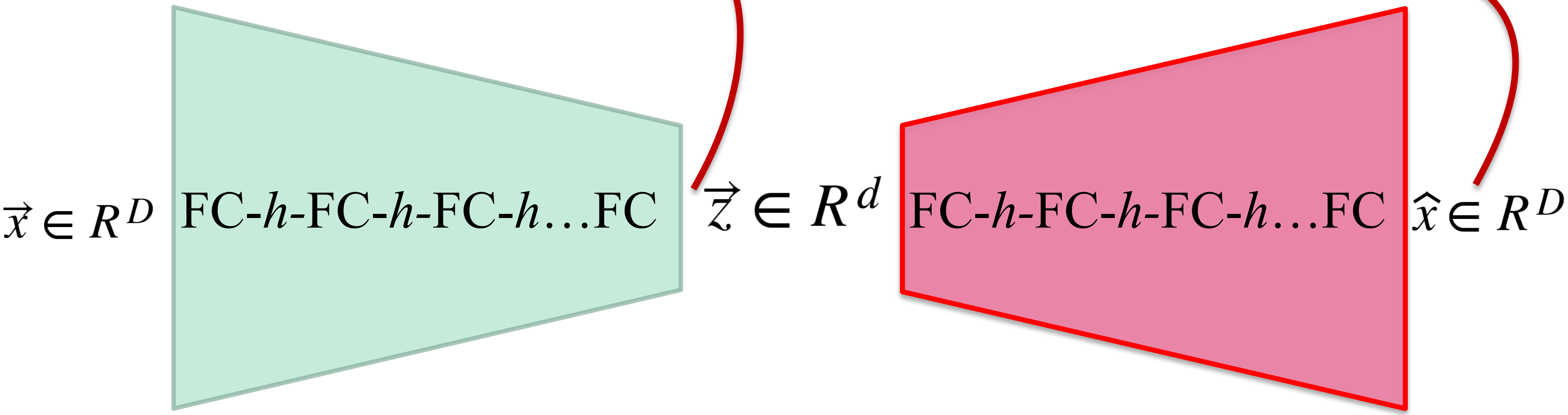

$d << D$
$\vec{x} \in R^D$
Encoder
$\vec{z} \in R^d$
Decoder
$\hat{x} \in R^D$

Fully-Connected Layers

h : activation function

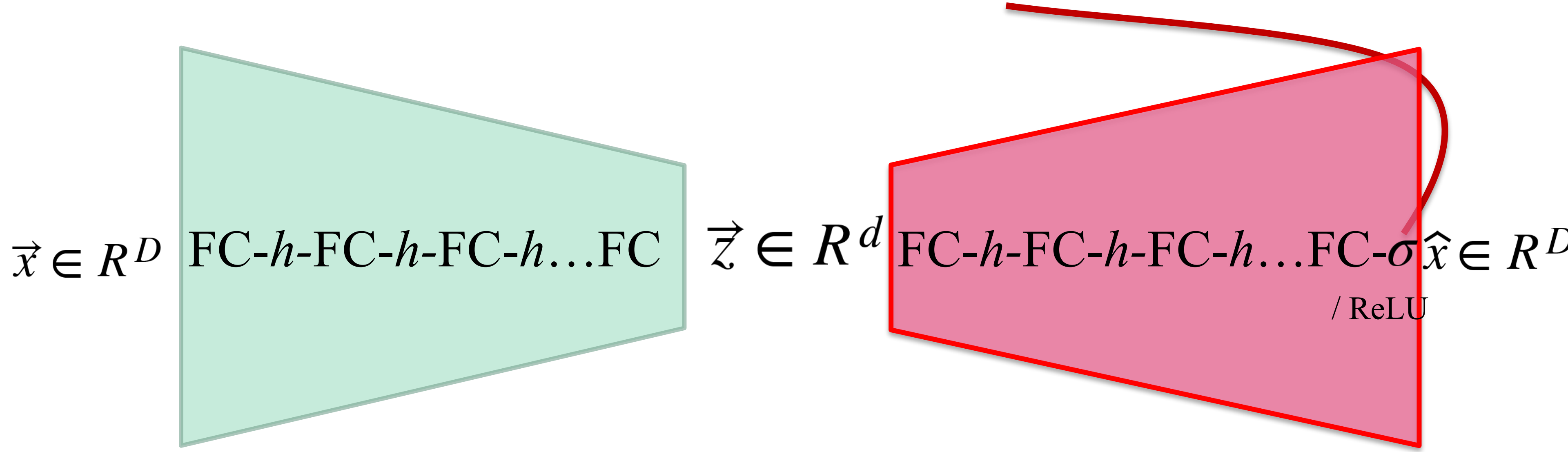


Very often...
No activation function at the output
of the encoder and the decoder



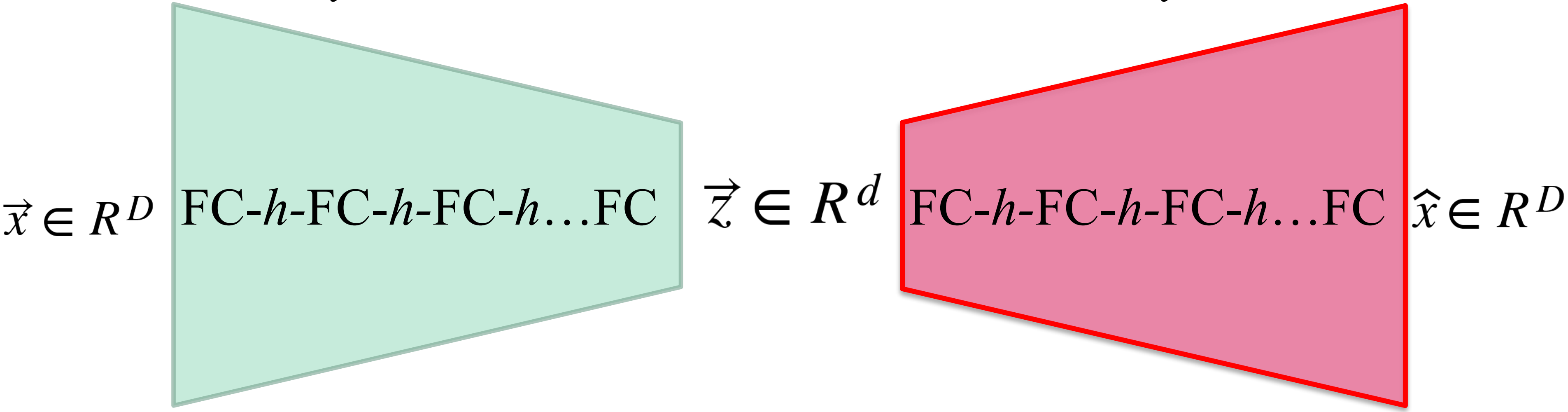
See **why?**

Sometimes **sigmoid** to predict pixel values between 0 and 1 or **ReLU** when the pixel values can be large but never negative.



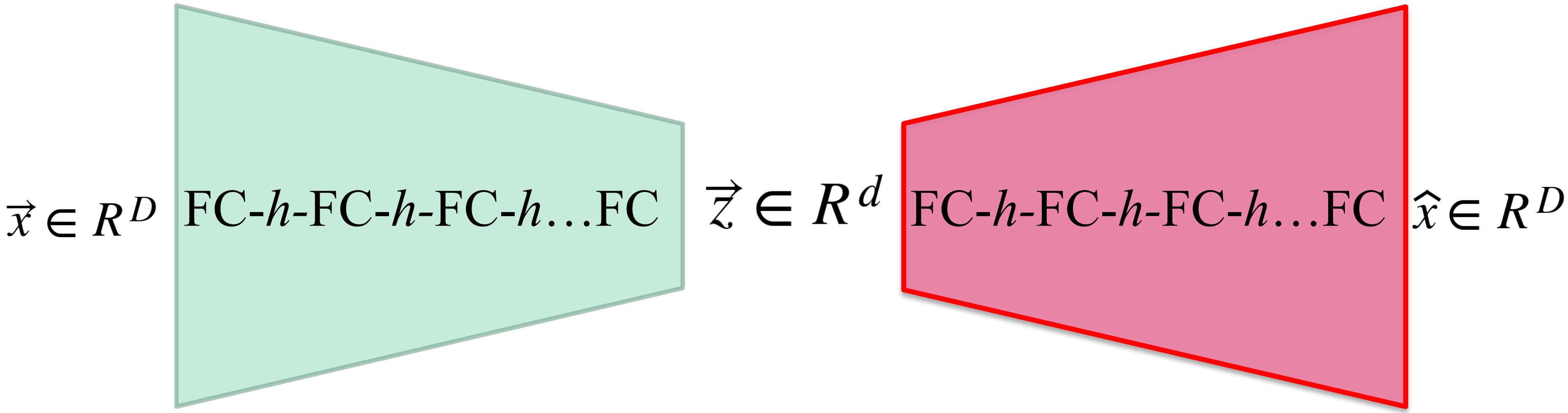
The number of neurons
Decrease (or stay the same)
from a layer to another

The number of neurons
Increase (or stay the same)
from a layer to another



Very often...

The structure of the encoder is the **dual** of that of the decoder



Simple MNIST Autoencoder

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12, 2))
        self.decoder = nn.Sequential(
            nn.Linear(2, 12), nn.ReLU(True),
            nn.Linear(12, 64), nn.ReLU(True),
            nn.Linear(64, 128), nn.ReLU(True),
            nn.Linear(128, 28 * 28))

    def forward(self, x):
        z = self.encoder(x)
        x_prime = self.decoder(z)
        return x_prime
```

Latent space 2D

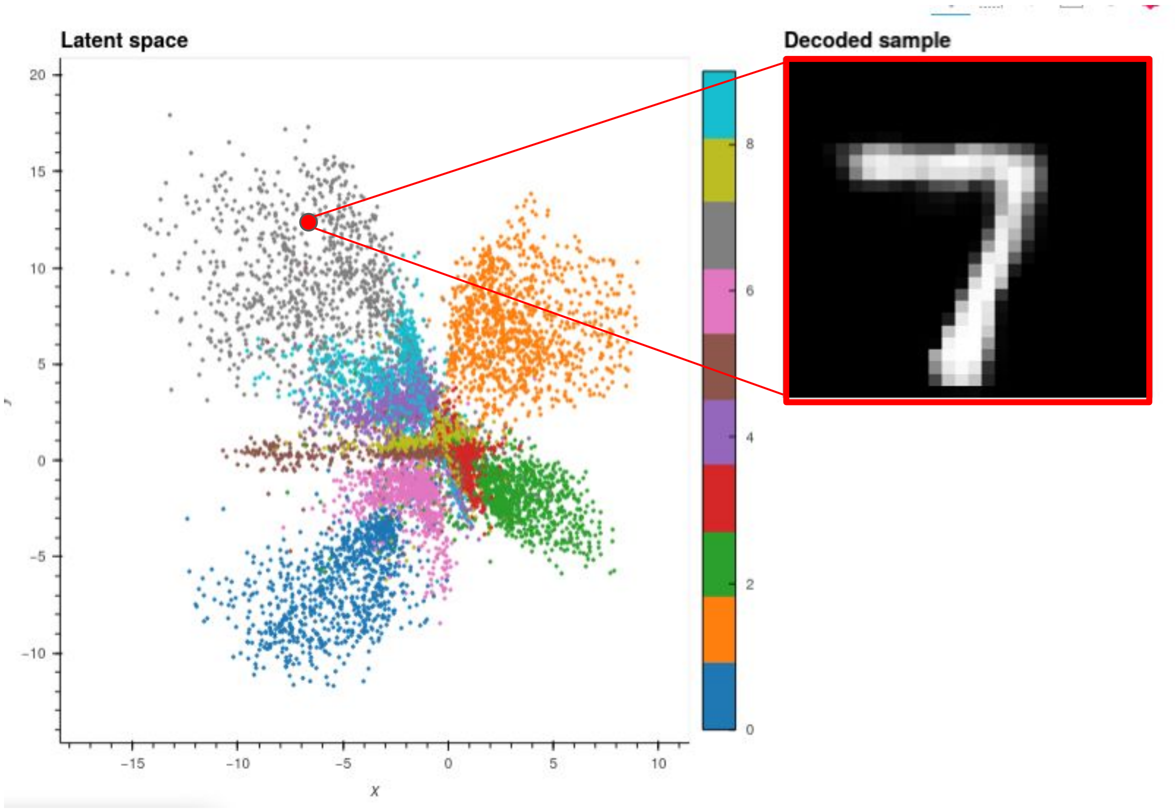
Simple MNIST Autoencoder

symmetry

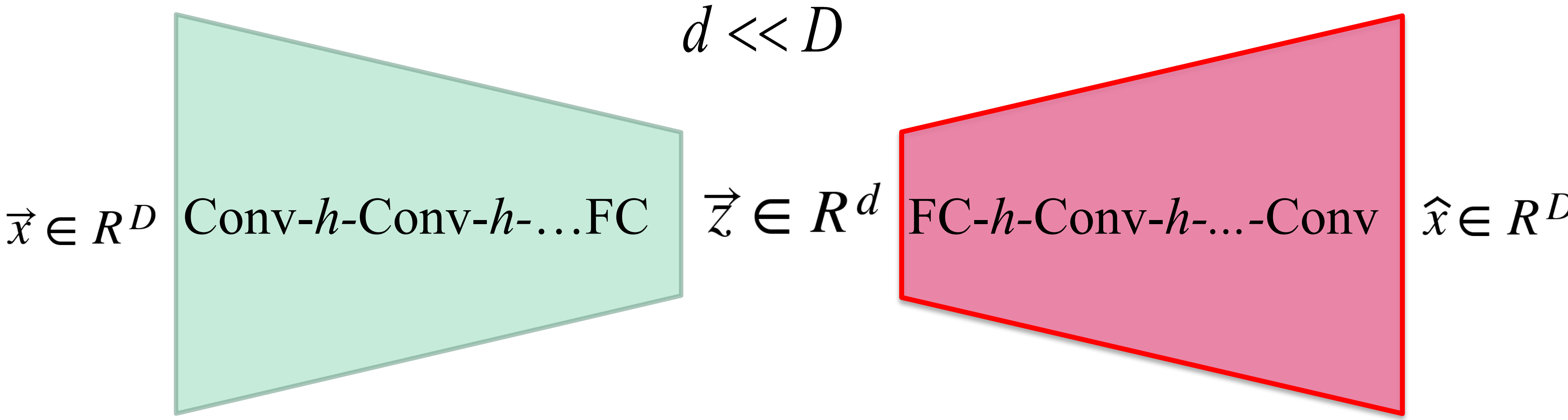
```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128), nn.ReLU(True),
            nn.Linear(128, 64), nn.ReLU(True),
            nn.Linear(64, 12), nn.ReLU(True),
            nn.Linear(12, 2))
        self.decoder = nn.Sequential(
            nn.Linear(2, 12), nn.ReLU(True),
            nn.Linear(12, 64), nn.ReLU(True),
            nn.Linear(64, 128), nn.ReLU(True),
            nn.Linear(128, 28 * 28))

    def forward(self, x):
        z = self.encoder(x)
        x_prime = self.decoder(z)
        return x_prime
```

MNIST latent space (for 1000 images)

Each 2D point corresponds to an image

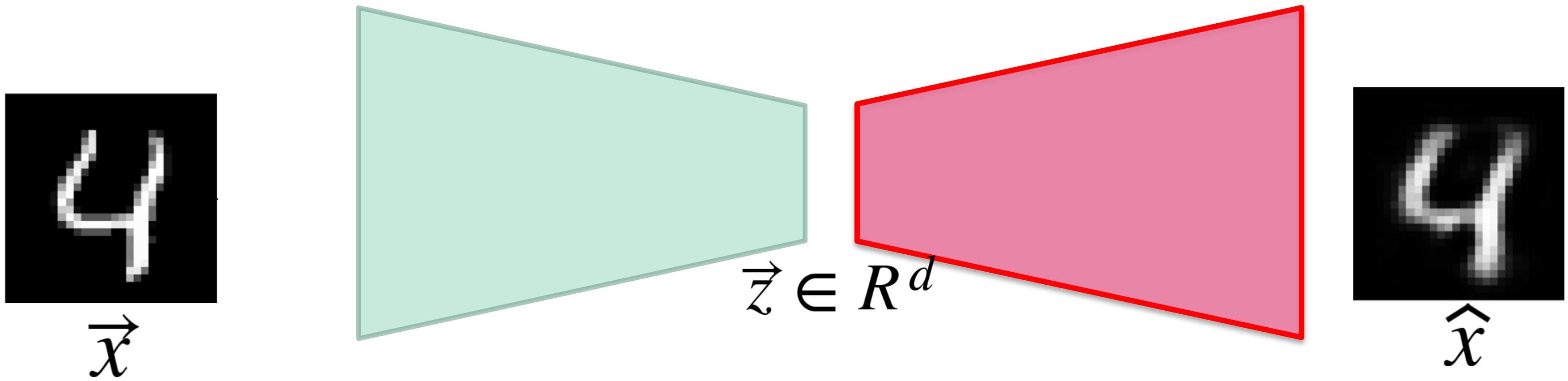


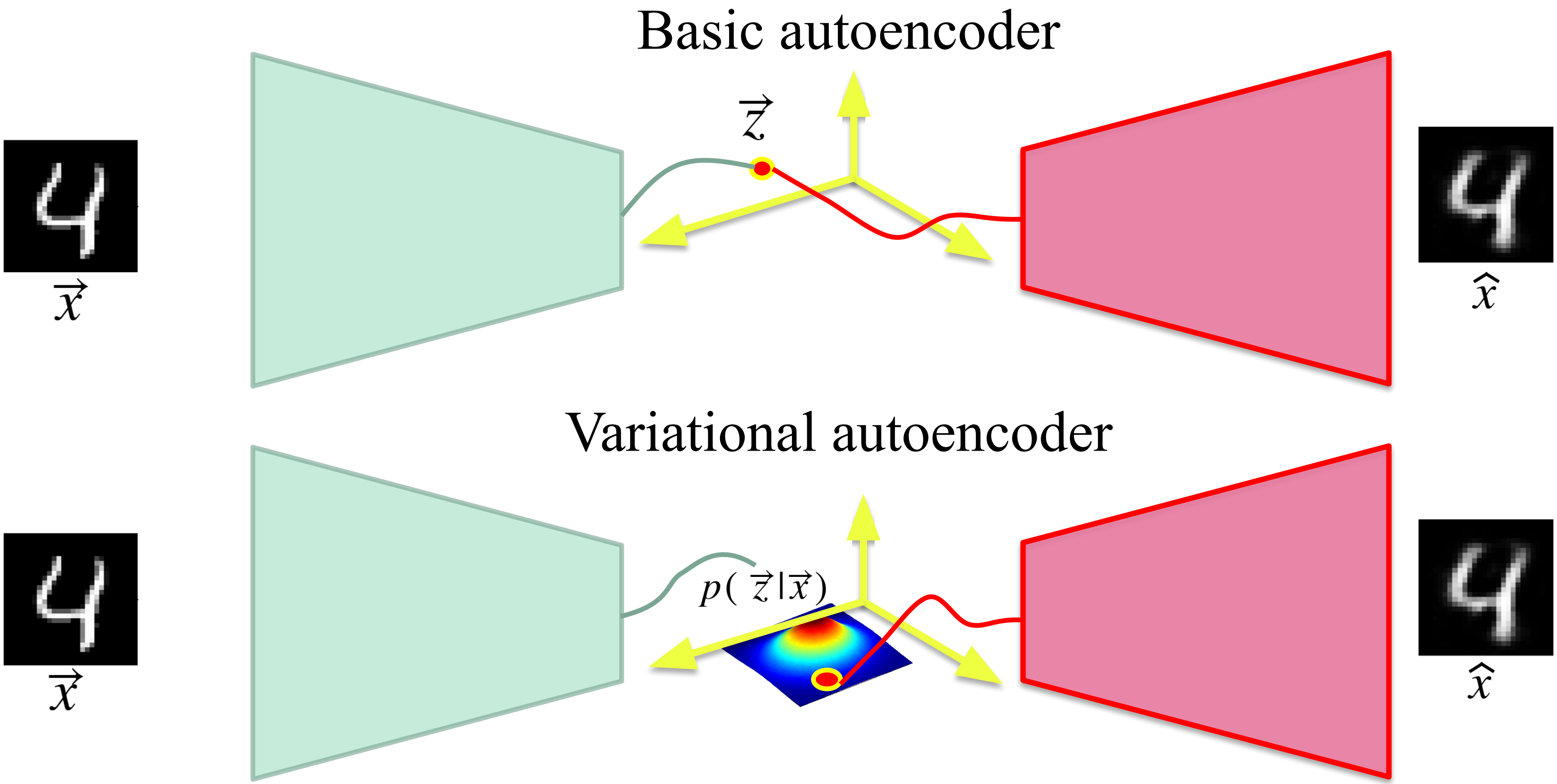
Conv layers



$d << D$
$\vec{x} \in R^D$
Conv-*h*-Conv-*h*-…FC
$\vec{z} \in R^d$
FC-*h*-Conv-*h*-…-Conv
$\hat{x} \in R^D$

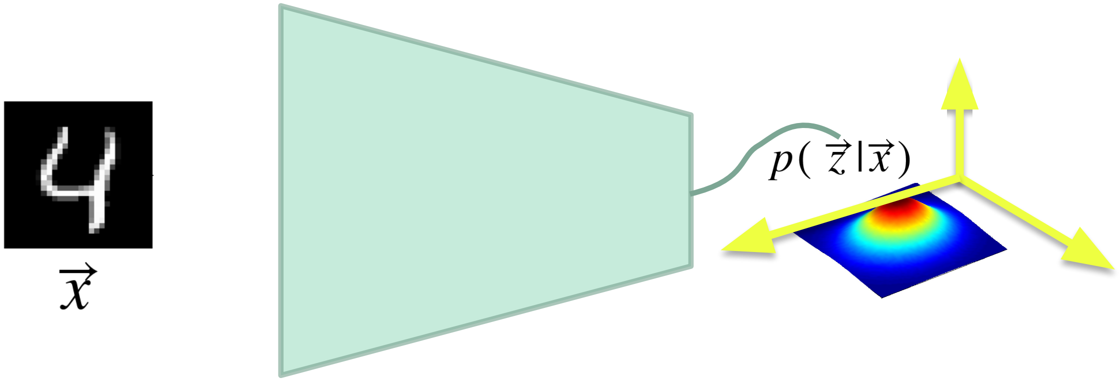
Basic image-based autoencoder





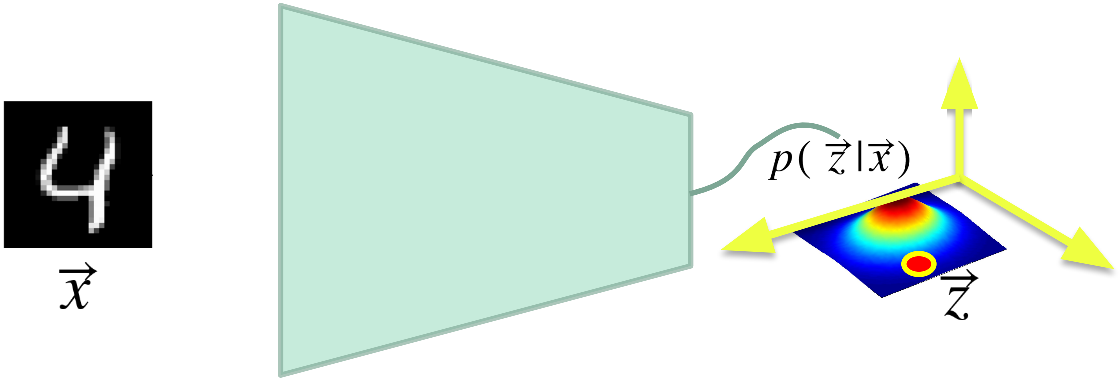
Basic autoencoder
$\vec{z}$
$\vec{x}$
$\hat{x}$
Variational autoencoder
$p(\vec{z}|\vec{x})$
$\vec{x}$
$\hat{x}$

Variational autoencoder



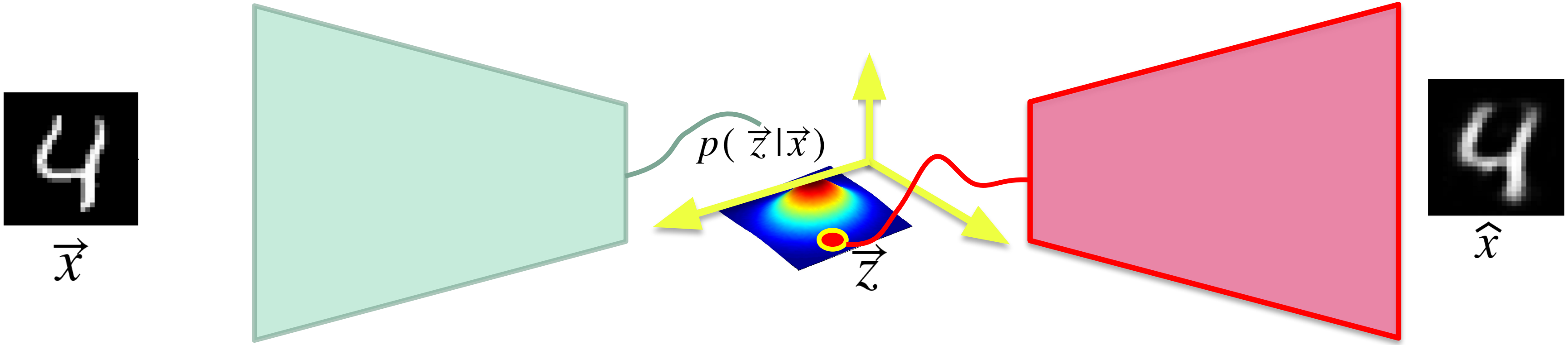
The encoder outputs a **distribution** $p(\vec{z}|\vec{x})$ and not just a **vector** $\vec{z}$

Variational autoencoder



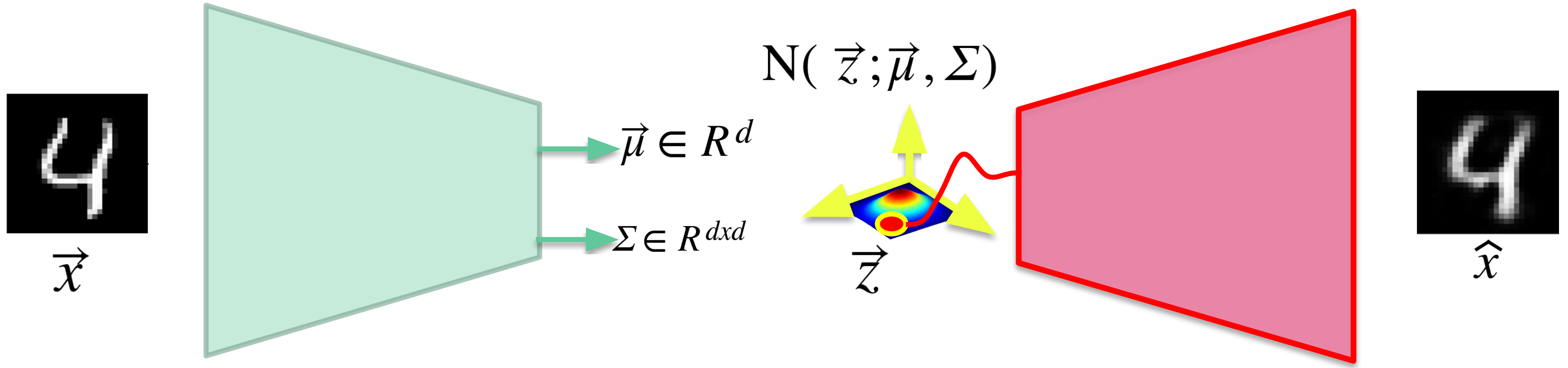
Random sample $\vec{z} \sim P(\vec{z}|\vec{x})$

Variational autoencoder

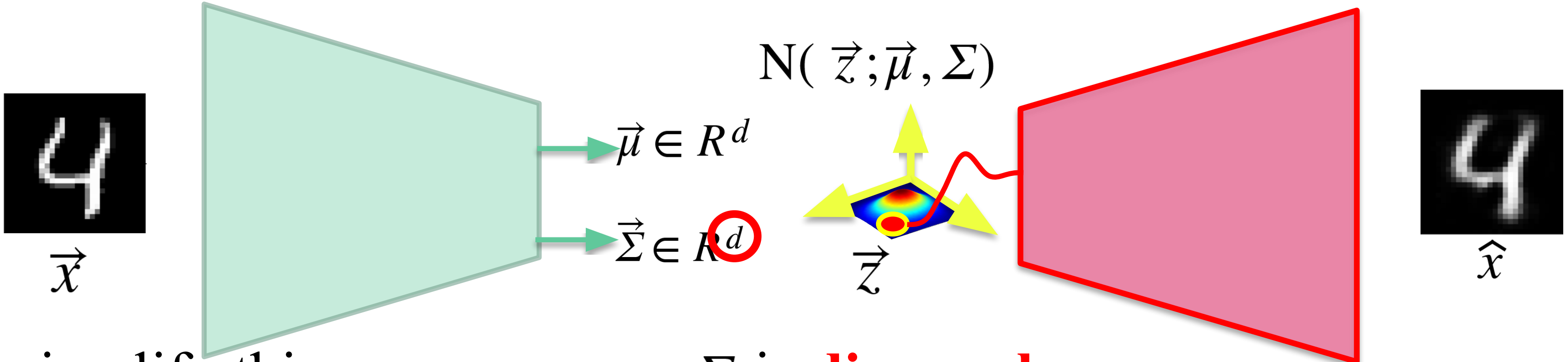


...and rebuild $\hat{x}$

$$p(\vec{z}\,|\,\vec{x}) \sim Gaussian$$



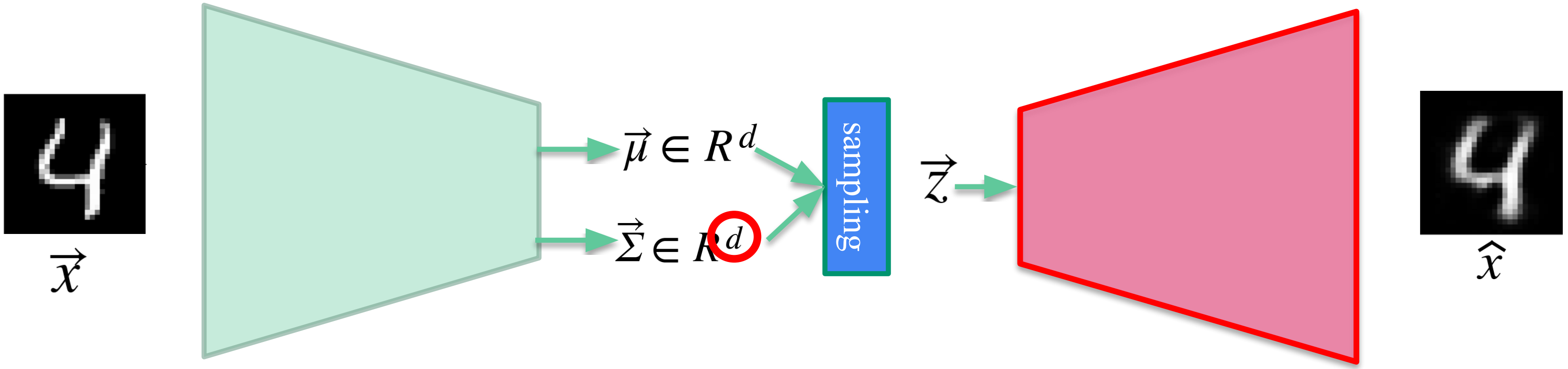
$$p(\vec{z}\,|\,\vec{x}) \sim Gaussian$$



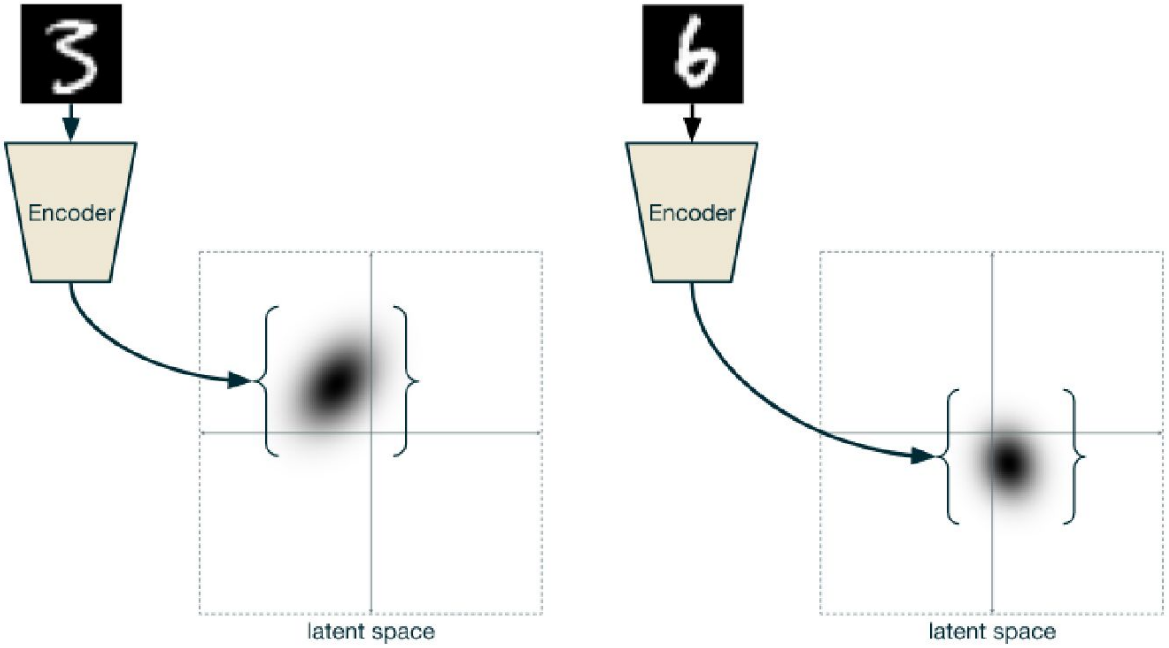
To simplify things, we assume Σ is **diagonal**

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & 0 & 0 \\ 0 & \sigma_1^2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \sigma_d^2 \end{pmatrix}$$

Variational autoencoder

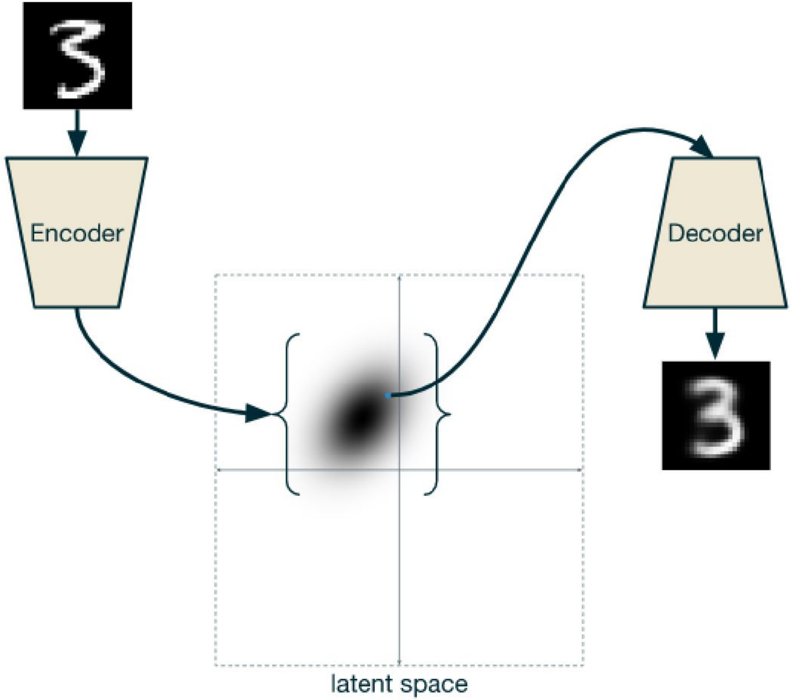


Another way of seeing things…



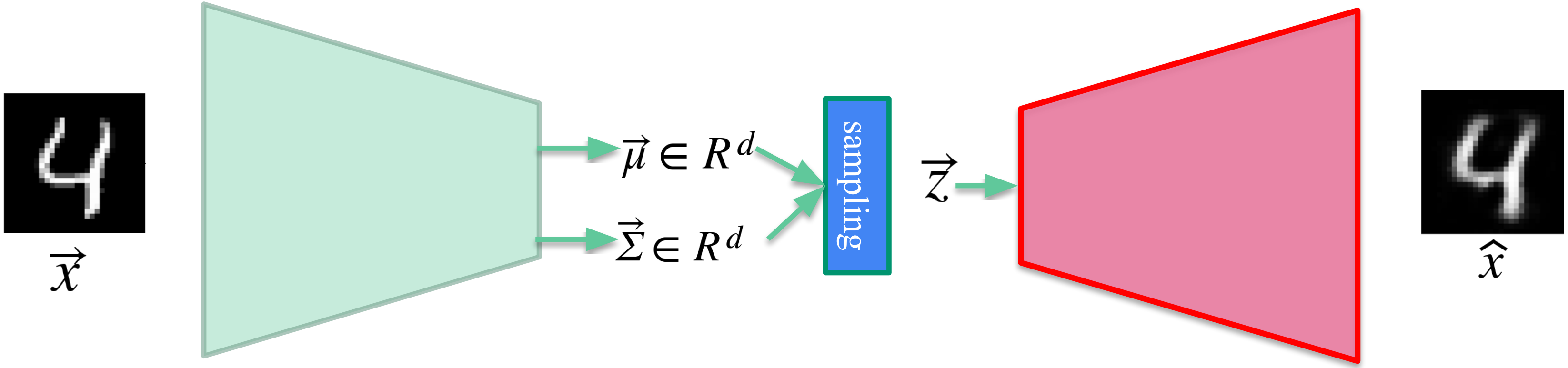
https://ijdykeman.github.io/ml/2016/12/21/cvae.html

Another way of seeing things…



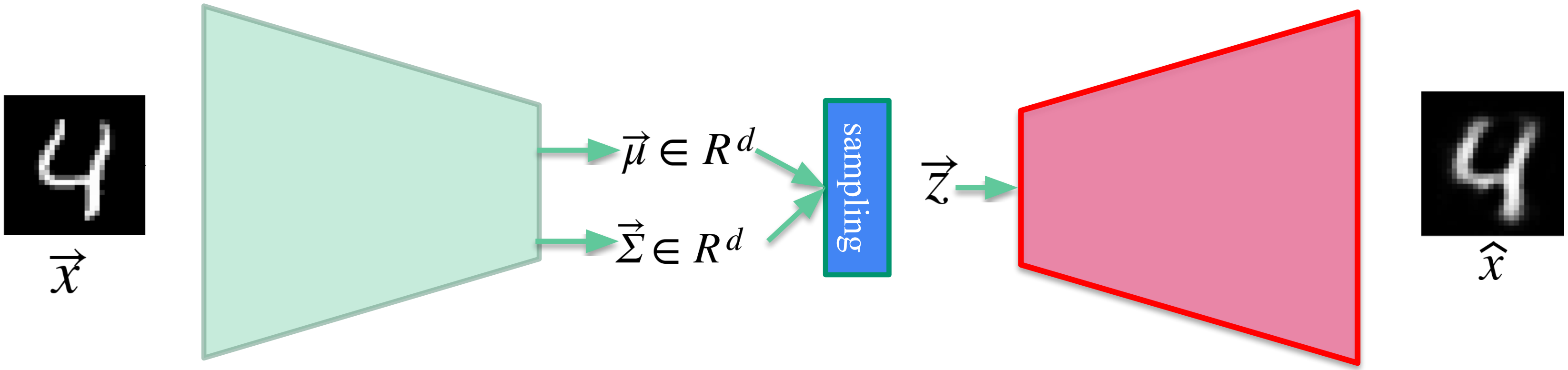
https://ijdykeman.github.io/ml/2016/12/21/cvae.html

ELBO loss : Evidence Lower Bound loss



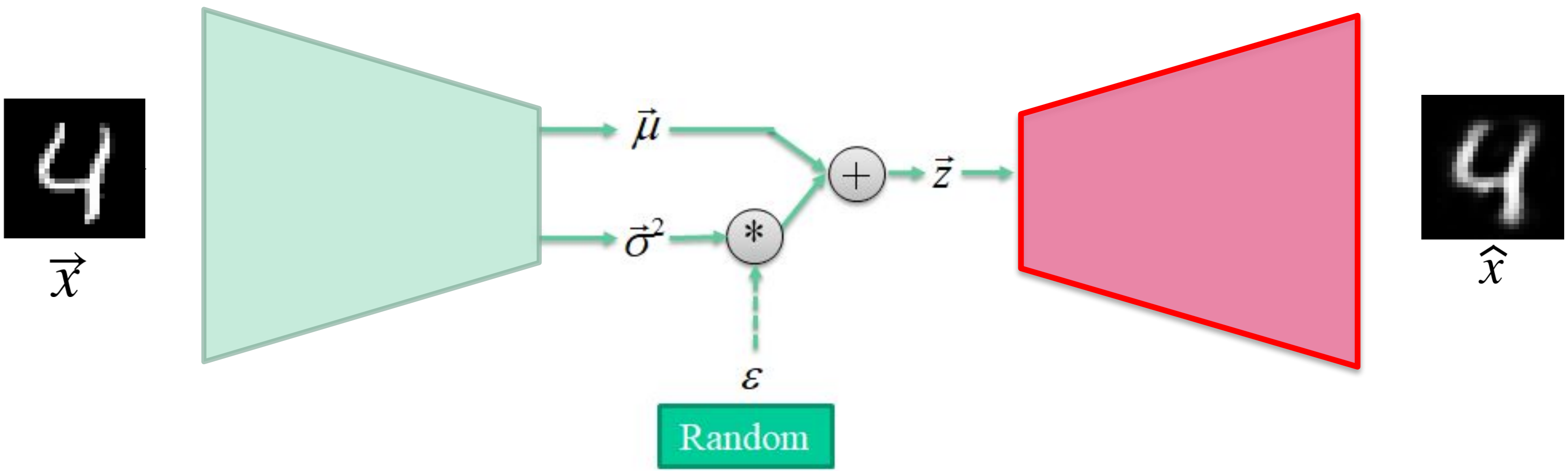
$$Loss = \underbrace{(\vec{x} - \hat{x})^2}_{\text{Loss decoder}} + \underbrace{\lambda KL\left(\mathrm{N}\left(\vec{z}; \vec{0}, \vec{1}\right), \mathrm{N}\left(\vec{z}; \vec{\mu}, \vec{\Sigma}\right)\right)}_{\text{Loss encoder}}$$

Other loss (in case the output is binary)

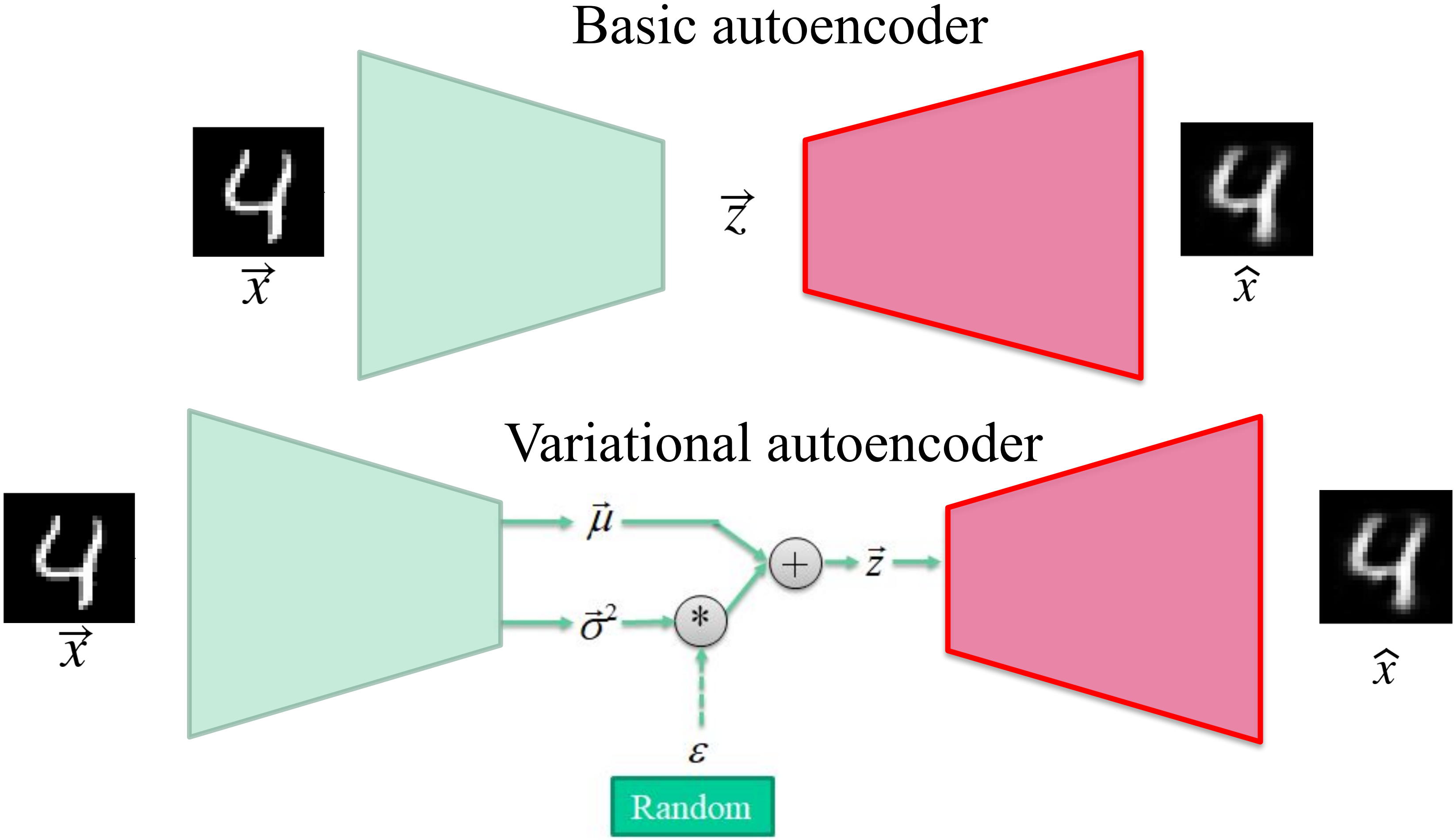


$$Loss = \underbrace{CrossEntropy(\vec{x}, \hat{x})}_{\text{Loss decoder}} + \underbrace{\lambda KL\left(N(\vec{z}; \vec{0}, \vec{1}), N(\vec{z}; \vec{\mu}, \vec{\Sigma})\right)}_{\text{Loss encoder}}$$

Reparametrization trick instead of sampling



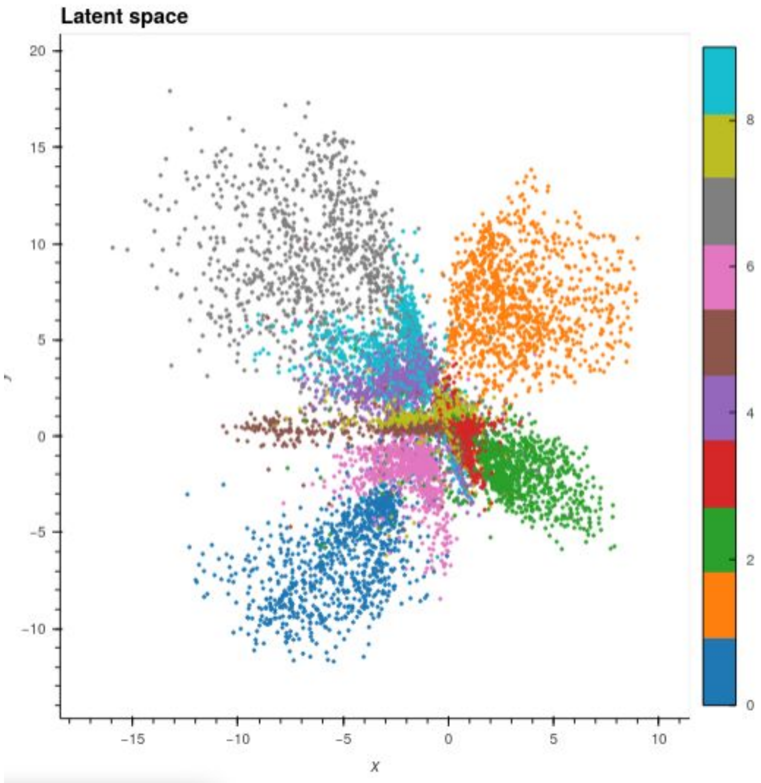
MNIST

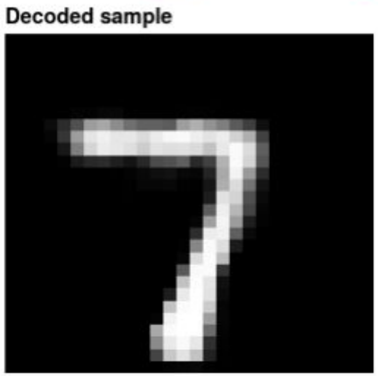


Basic autoencoder
$\vec{x}$
$\vec{z}$
$\hat{x}$
Variational autoencoder
$\vec{x}$
$\vec{\mu}$
$\vec{\sigma}^2$
*
+
$\vec{z}$
ε
Random
$\hat{x}$

MNIST

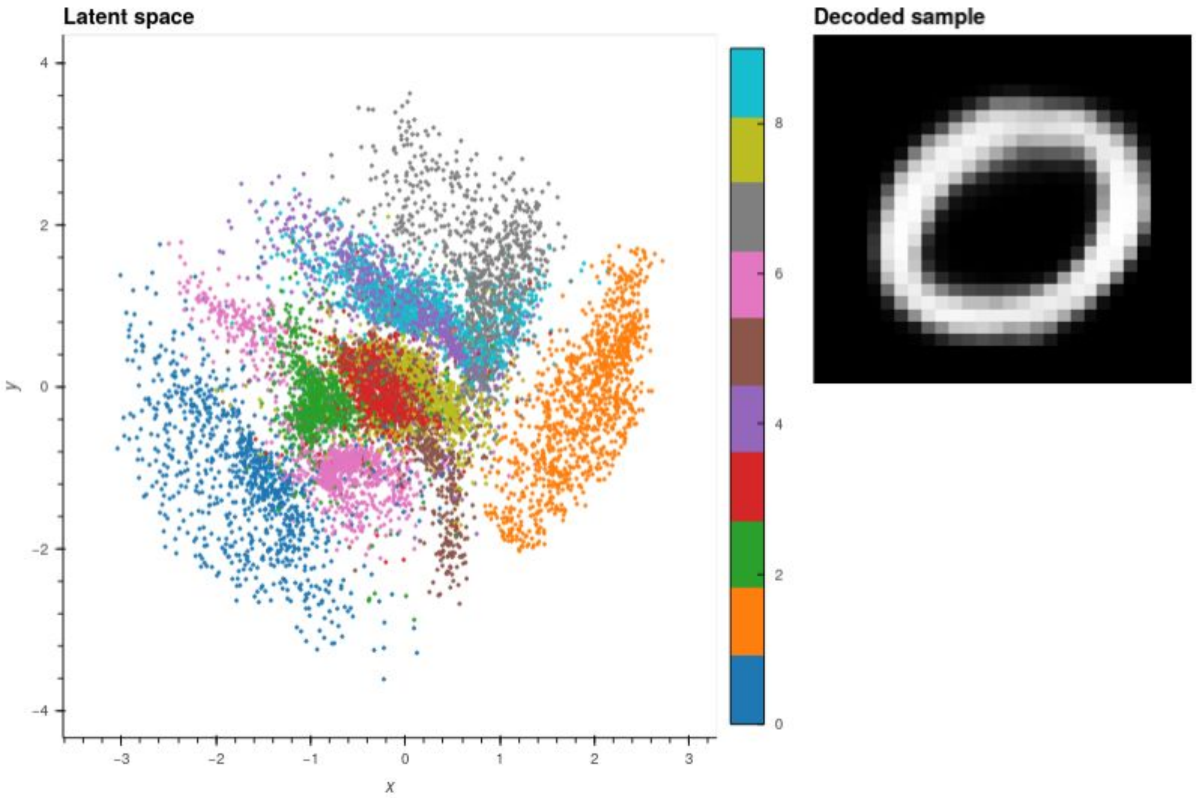
Visualize the latent space (autoencoder)





MNIST

Visualize the latent space (variational autoencoder)



MNIST

Code is in the file:

`mnist-autoencoders.ipynb`

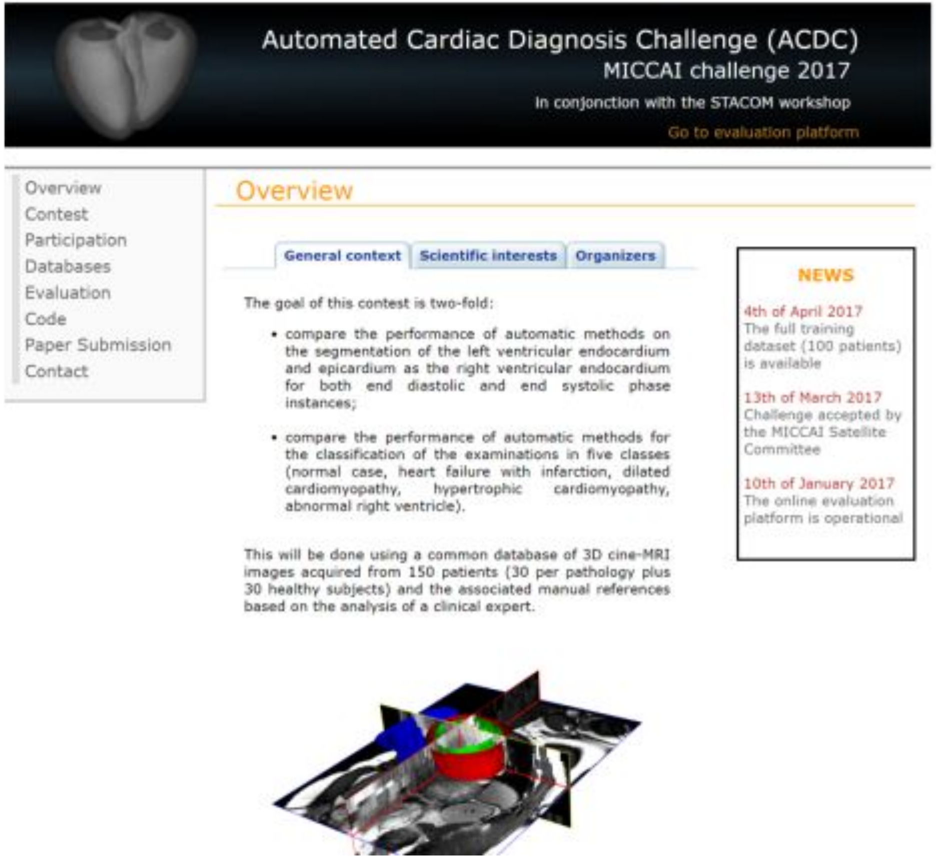
Automated Cardiac Diagnosis Challenge

ACDC

150 exams (all from different patients) divided into 5 groups:

- dilated cardiomyopathy (DCM),
- hypertrophic cardiomyopathy (HCM),
- myocardial infarction (MINF),
- abnormal right ventricle (RV)
- patients without cardiac disease (NOR).

website: ***creatis.insa-lyon.fr/Challenge/acdc***



Automated Cardiac Diagnosis Challenge (ACDC)
MICCAI challenge 2017
In conjonction with the STACOM workshop
Go to evaluation platform

- Overview
- Contest
- Participation
- Databases
- Evaluation
- Code
- Paper Submission
- Contact

Overview

General context | Scientific interests | Organizers

The goal of this contest is two-fold:

- compare the performance of automatic methods on the segmentation of the left ventricular endocardium and epicardium as the right ventricular endocardium for both end diastolic and end systolic phase instances;
- compare the performance of automatic methods for the classification of the examinations in five classes (normal case, heart failure with infarction, dilated cardiomyopathy, hypertrophic cardiomyopathy, abnormal right ventricle).

This will be done using a common database of 3D cine-MRI images acquired from 150 patients (30 per pathology plus 30 healthy subjects) and the associated manual references based on the analysis of a clinical expert.

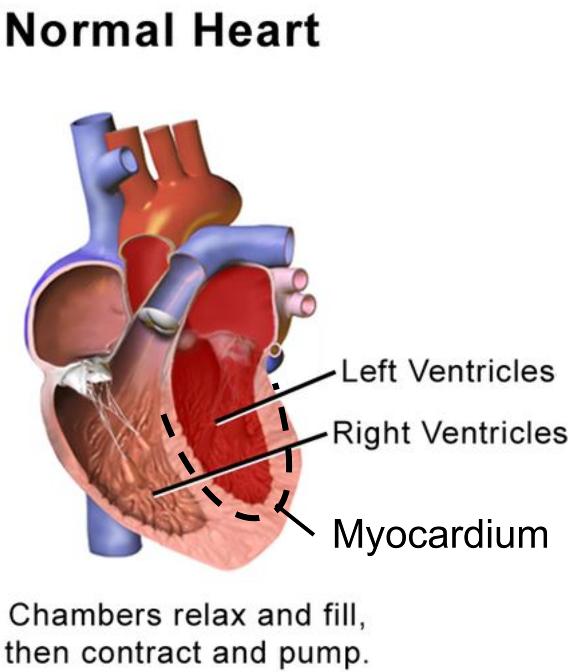
NEWS

4th of April 2017
The full training dataset (100 patients) is available

13th of March 2017
Challenge accepted by the MICCAI Satellite Committee

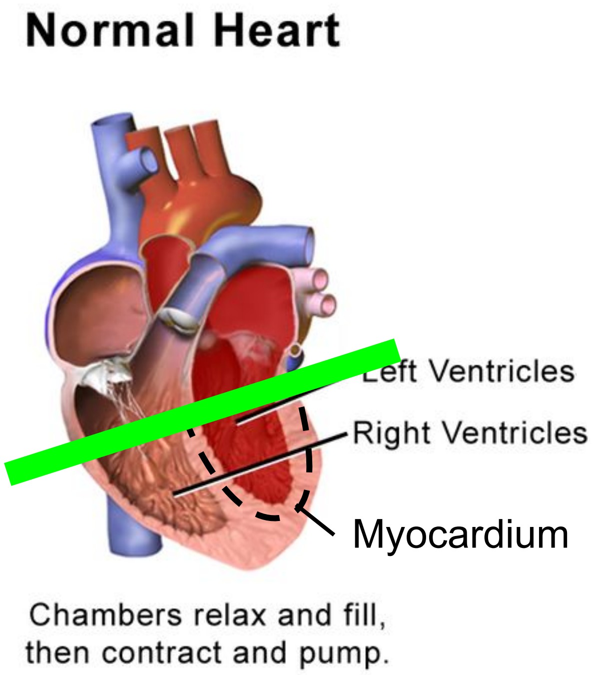
10th of January 2017
The online evaluation platform is operational

Cardiac anatomy

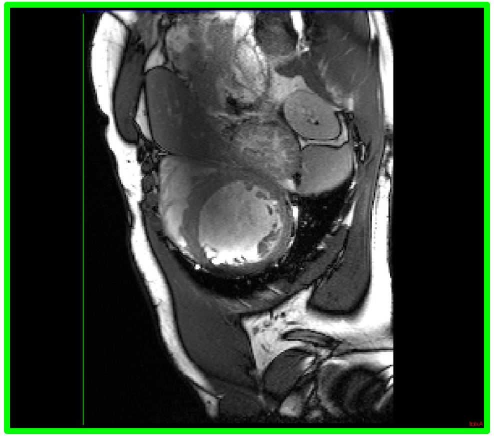


https://commons.wikimedia.org/

Cardiac anatomy

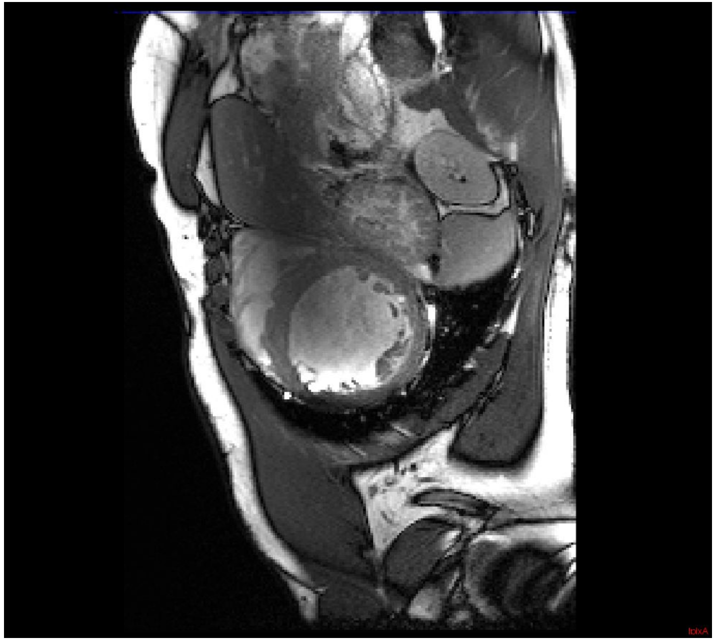


Cross-section : **short axis view**

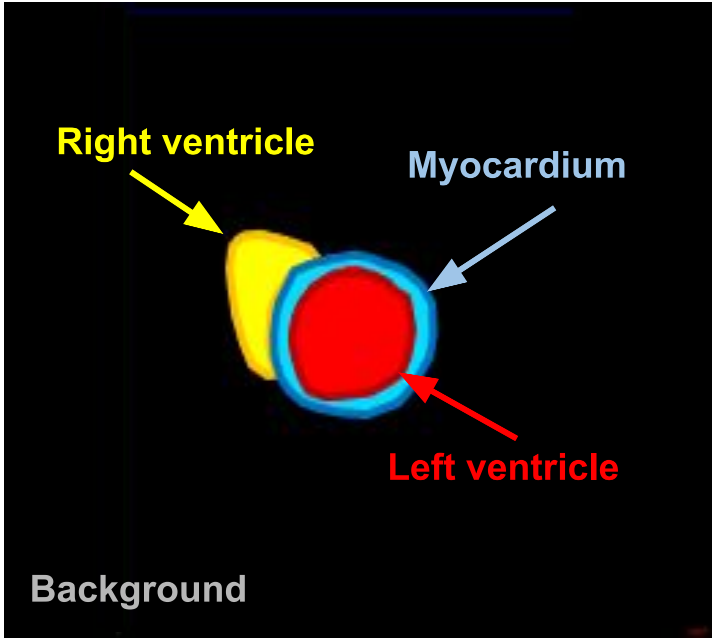


https://commons.wikimedia.org/

Short axis cardiac MRI

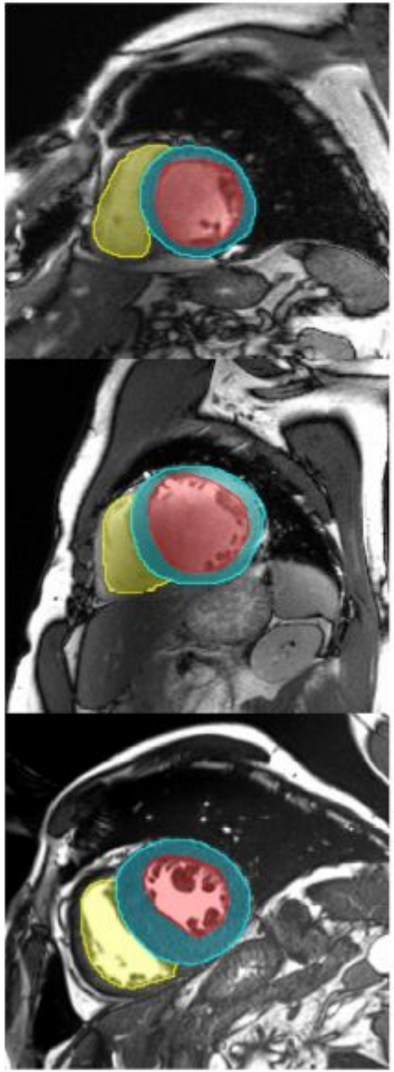


Short axis segmentation map



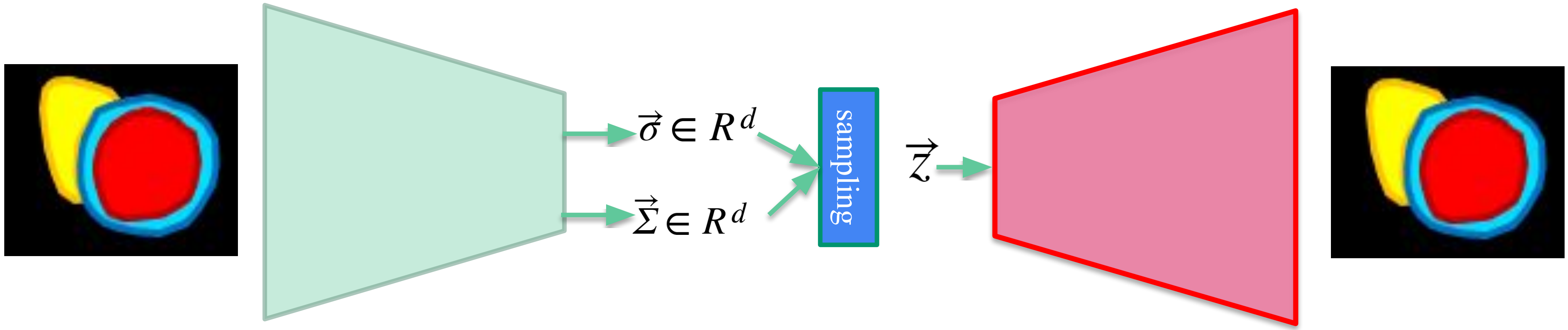
AC DC

Other examples:



ACDC

Convolutional Variational autoencoder



ACDC

Code is in the file:

`cardiac-mri-autoencoders.ipynb`