Neuroengineering (I)
7. Generative adversarial networks (GANs)

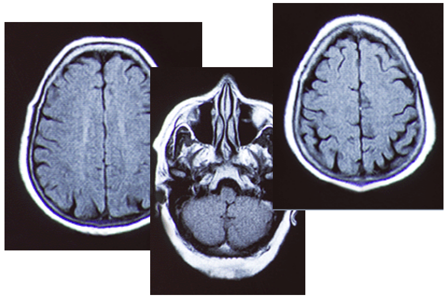
- **Scuola di Ingegneria Industriale e dell'Informazione** – Politecnico di Milano
- Prof. Pietro Cerveri

Generative models

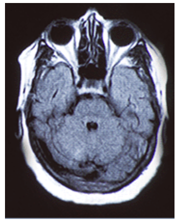
Unsupervised method to build a model explaining data

Given training data, generate new instances extracted from same distribution

Distribution → density estimation p



Training data ~ $p_{data}(x)$



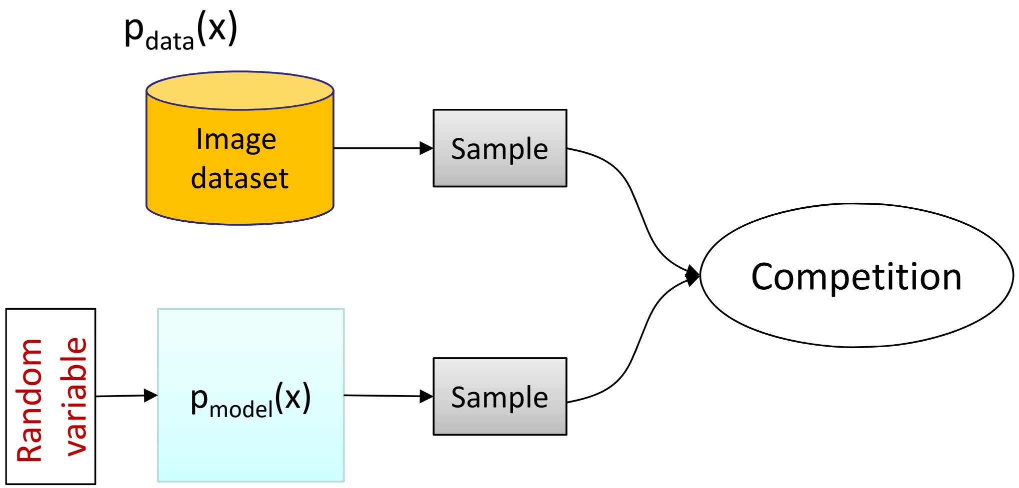
Generated new instance ~ $p_{model}(x)$

Density estimation can be explicit: p_{model} is defined
Density estimation is unknown and is to be learnt from data (GANs approach)

GAN principle

Model distribution is learnt from training distribution to generate data through competitive two-player game

Approach: sample from a simple distribution (e-g. random noise)

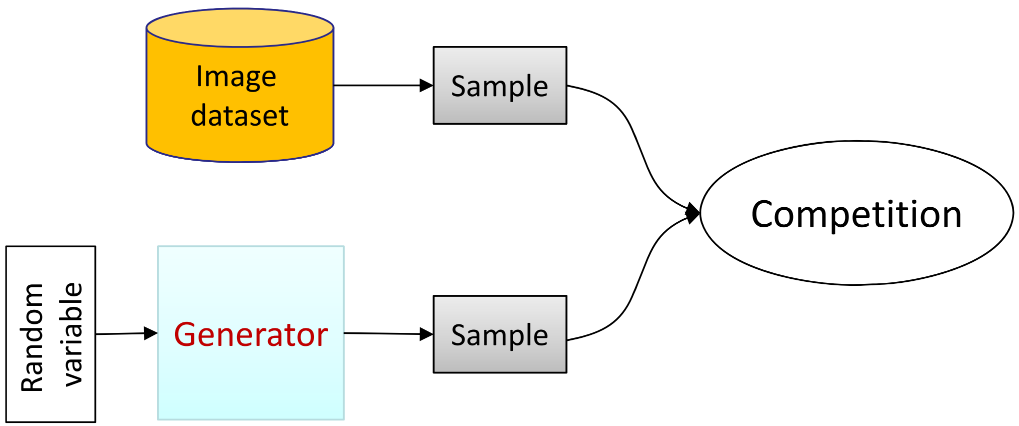


GAN principle

Model distribution is learnt from training distribution to generate data through competitive two-player game

Approach: sample from a simple distribution (e-g. random noise)

Learn the transformation from random noise to the data by means of a Generator network



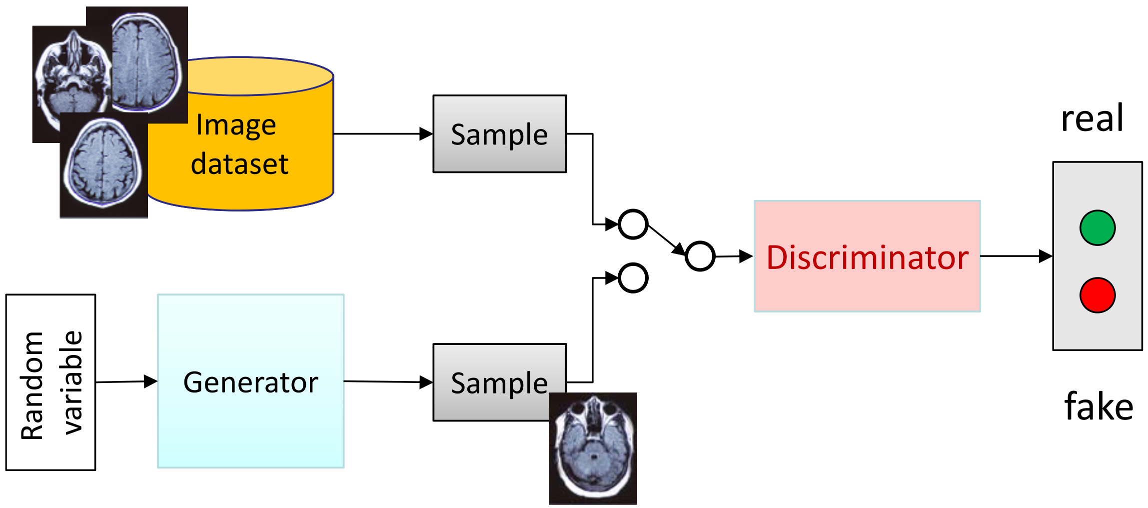
GAN principle

Model distribution is learnt from training distribution to generate data through two-player game

Approach: sample from a simple distribution (e-g. random noise)

Learn the transformation from random noise to the data by means of a Generator network

Use a Discriminator network to classify real and fake data



GAN principle

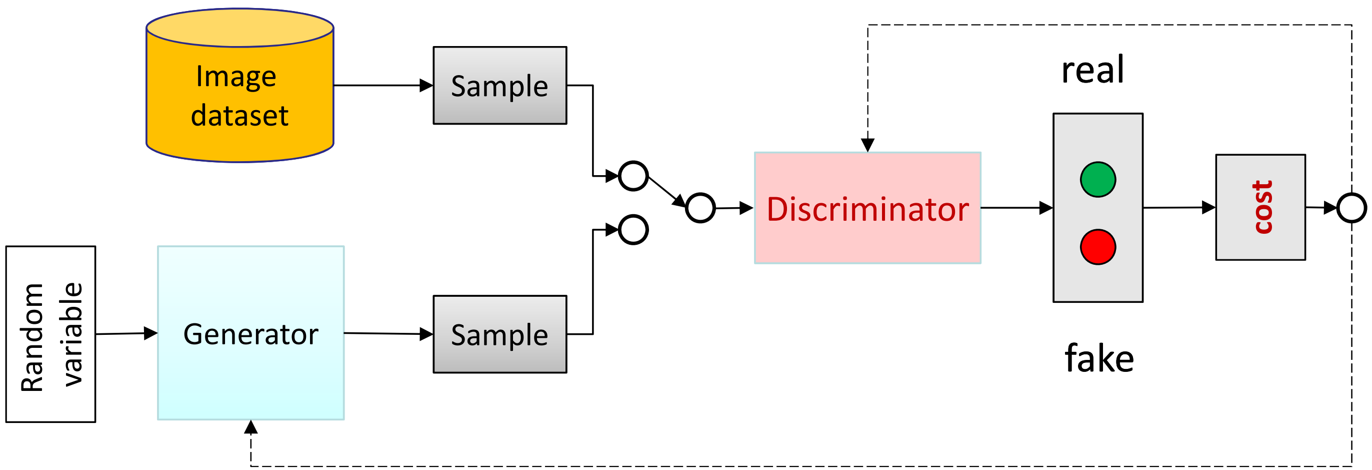
Model distribution is learnt from training distribution to generate data through two-player game

Approach: sample from a simple distribution (e-g. random noise)

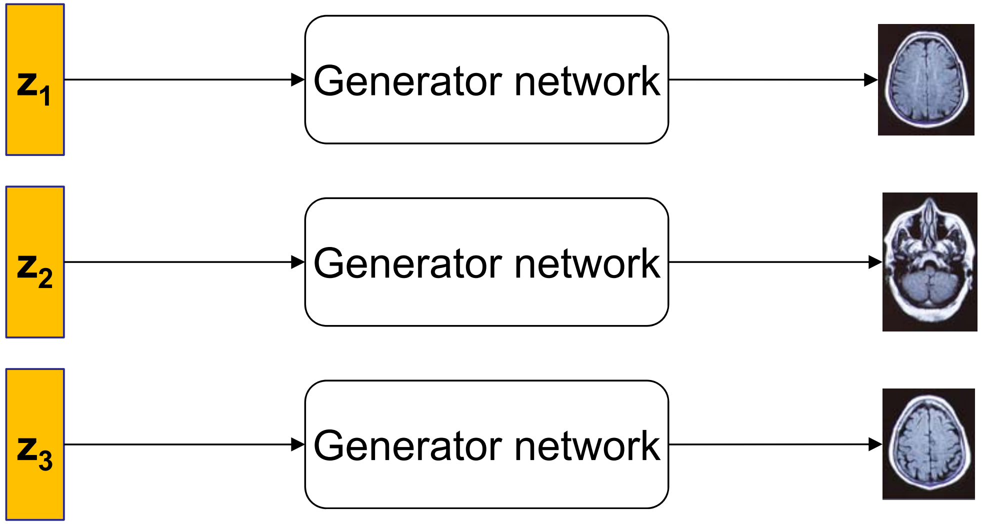
Learn the transformation from random noise to the data by means of a Generator network

Use a Discriminator network to classify real and fake data

Need a cost (loss) function to feedback both to generator and discriminator (training)



Focus on the generator

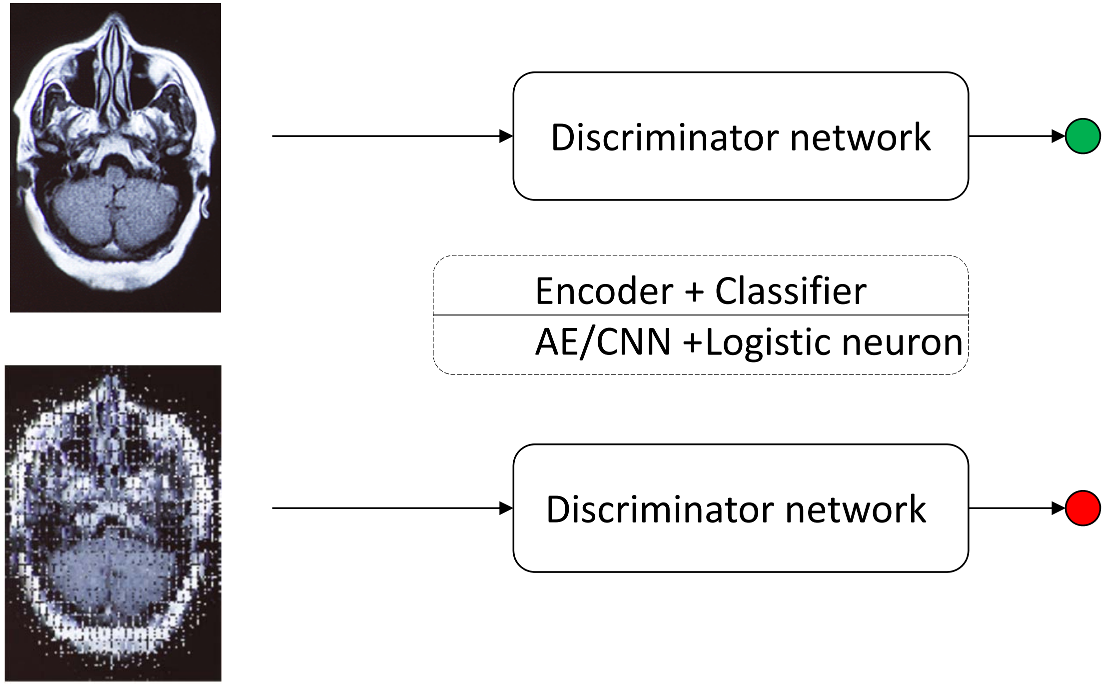


Gaussian pattern $\mathbf{z}_i$ of size H

Decoder

Output image of size M×N mimicking a sample of the training distribution

Focus on the Discriminator



Training GANs: two-player game

Generator network: try to fool the discriminator by producing real-looking images

Discriminator network: try to distinguish between real and fake images

Principle of training

The generator must be able to successfully trick the discriminator so that we are generating images that look like image from the training set.

Training GANs: two-player game

Generator network: try to fool the discriminator by producing real-looking images

Discriminator network: try to distinguish between real and fake images

Principle of training

The generator must be able to successfully trick the discriminator so that we are generating images that look like image from the training set.

Training approach: estimating jointly θ_g and θ_d in the so-called minimax game formulation (Nash equilibrium)

θ_g: parameters of the generator network
θ_d: parameters of the discriminator network

Training GANs: two-player game

Generator network: try to fool the discriminator by producing real-looking images

Discriminator network: try to distinguish between real and fake images

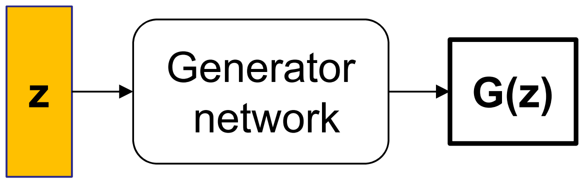
train jointly in minimax game by means of expectation values E

$$min_{\theta_g} max_{\theta_d} \left[E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Discriminator output for real data x

Discriminator output for simulated data G(z)

Discriminator outputs likelihood in (0,1) of real image



θ_g: parameters of the generator network
θ_d: parameters of the discriminator network

Training GANs: two-player game

Generator network: try to fool the discriminator by producing real-looking images

Discriminator network: try to distinguish between real and fake images

train jointly in minimax game by means of expectation values E

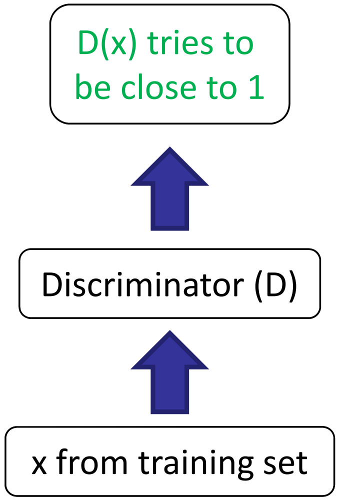
$$min_{\theta_g} max_{\theta_d} \left[E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)\right]$$

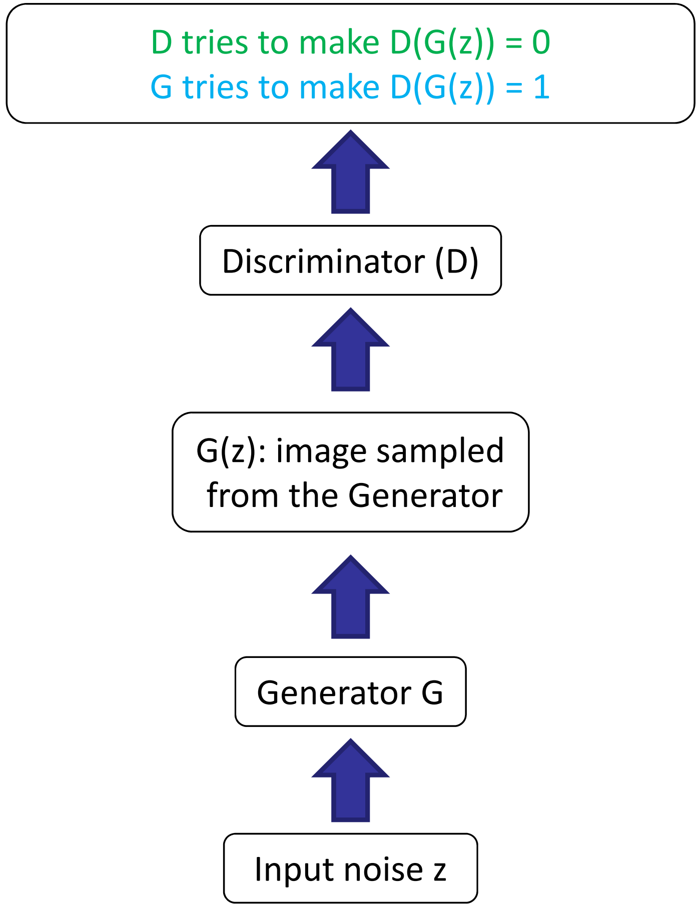
- Discriminator (θ_d) wants to maximize objective such that $D_{\theta_d}(x)$ is close to 1 (real) and $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ is close to 0 (simulated/fake)
- Generator wants to minimize objective such that $D_{\theta_d}\left(G_{\theta_g}(z)\right)$ is close to 1 (discriminator is tricked into thinking generated $G_{\theta_g}(z)$ is real)

Training GANs: two-player game

Generator network: try to fool the discriminator by producing real-looking images

Discriminator network: try to distinguish between real and fake images





Training GANs: two-player game

Minimax objective function to learn θ_g and θ_d parameters

$$min_{\theta_g} max_{\theta_d} \left[E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

Alternate between

1. Gradient ascent on discriminator

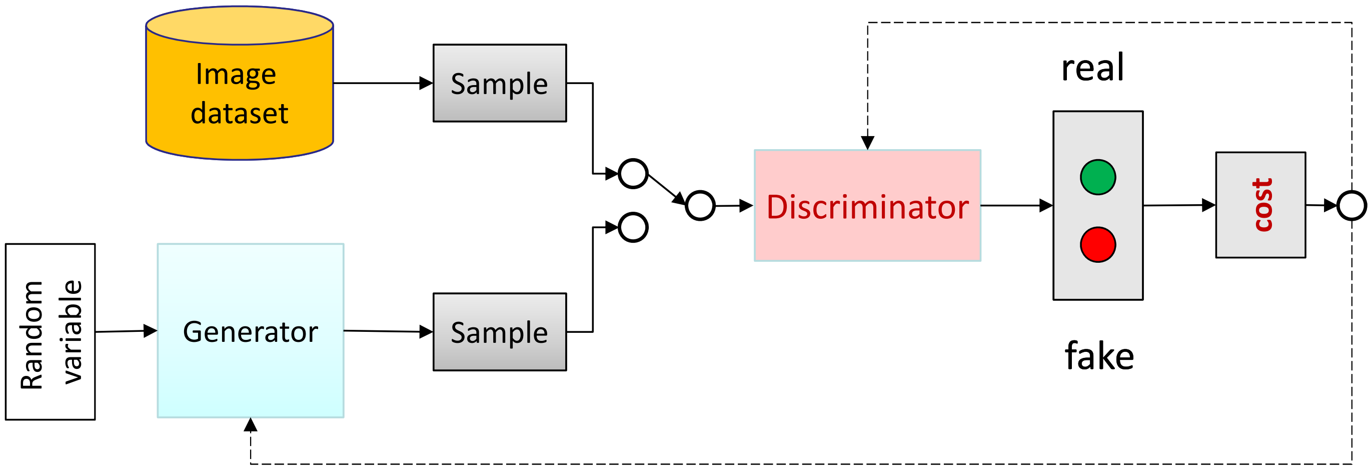
$$max_{\theta_d} \left[E_{x \sim p_{data}} log D_{\theta_d}(x) + E_{z \sim p_z} log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right) \right]$$

2. Gradient descent on generator

$$min_{\theta_g} E_{z \sim p_z} log \left(1 - D_{\theta_d} \left(G_{\theta_g}(z) \right) \right)$$

Training GANs schema

$$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}\left[logD_{\theta_d}\left(x^{(i)}\right) + log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right)\right]$$ (gradient ascent)



$$-\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\left[log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right)\right]$$ (gradient descent)

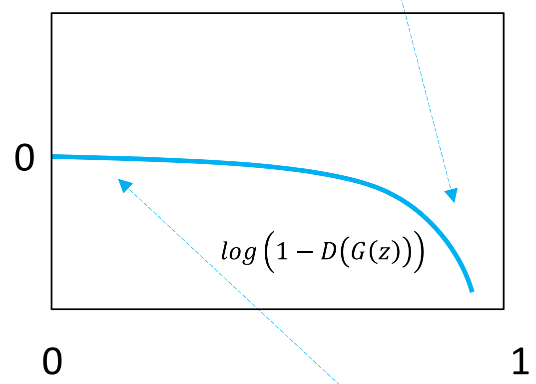
batch of *m* samples

Focus on generator

Gradient descent on generator

$$min_{\theta_g} E_{z\sim p_z} log\left(1 - D_d\left(G_{\theta_g}(z)\right)\right)$$

Gradient signal (high) dominated by region where sample is already good



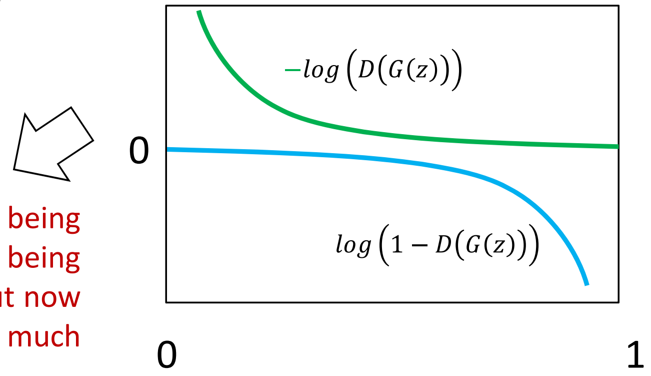
Practically, optimizing this generator objective is weak

When sample is likely fake, you want to learn from it to improve generator. But gradient in this region is relatively flat

Focus on generator

Instead: Gradient ascent on different generator objective

$$max_{\theta_g} E_{z\sim p_z} log\left(D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$



Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. Same objective of fooling discriminator, but now higher gradient signal for bad samples → works much better

Training GANs implementation

Putting it together: GAN training algorithm

for number of training iterations **do**
  for *k* steps **do**

- Sample a set of *m* noise samples $\{\boldsymbol{z}^{(1)}, .., \boldsymbol{z}^{(m)}\}$ from noise source $p_g(\boldsymbol{z})$
- Sample a set of *m* examples $\{\boldsymbol{x}^{(1)}, .., \boldsymbol{x}^{(m)}\}$ from training dataset
- Update the discriminator by ascending its stochastic gradient

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[log D_{\theta_d}(x^{(i)}) + log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z^{(i)})\right)\right)\right]$$

  end for

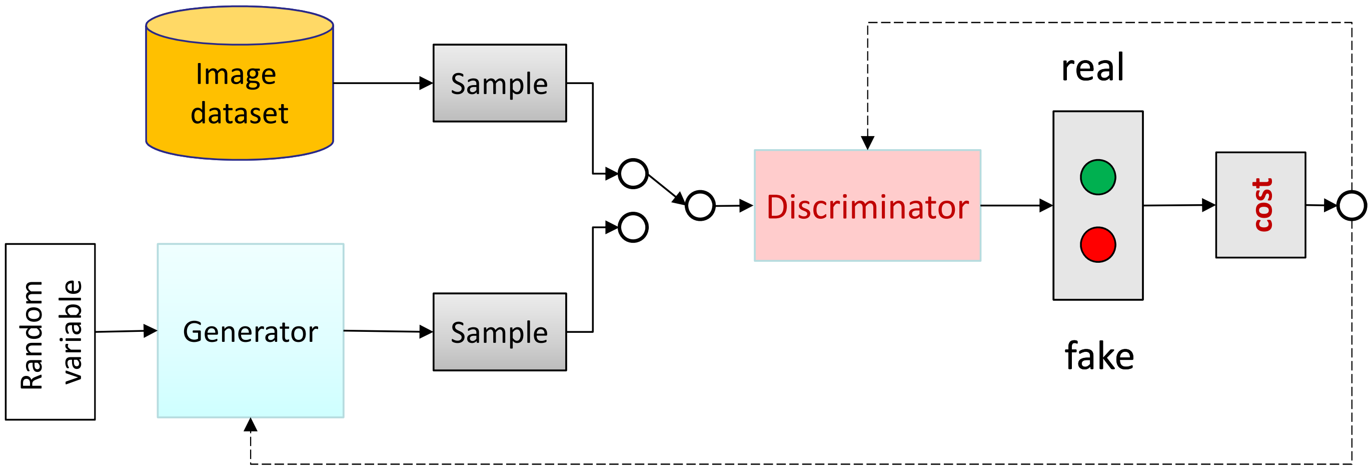
- Sample a set of *m* noise samples $\{\boldsymbol{z}^{(1)}, .., \boldsymbol{z}^{(m)}\}$ from noise source $p_g(\boldsymbol{z})$
- Update the generator by ascending its stochastic gradient

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[log\left(D_{\theta_d}\left(G_{\theta_g}(z^{(i)})\right)\right)\right]$$

end for

Training GANs schema

$$\nabla_{\theta_d} \frac{1}{m}\sum_{i=1}^{m}\left[\log D_{\theta_d}\left(x^{(i)}\right) + \log\left(1 - D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right)\right]$$ (gradient ascent)



$$\nabla_{\theta_g} \frac{1}{m}\sum_{i=1}^{m}\left[\log\left(D_{\theta_d}\left(G_{\theta_g}\left(z^{(i)}\right)\right)\right)\right]$$ (gradient ascent)

batch of *m* samples

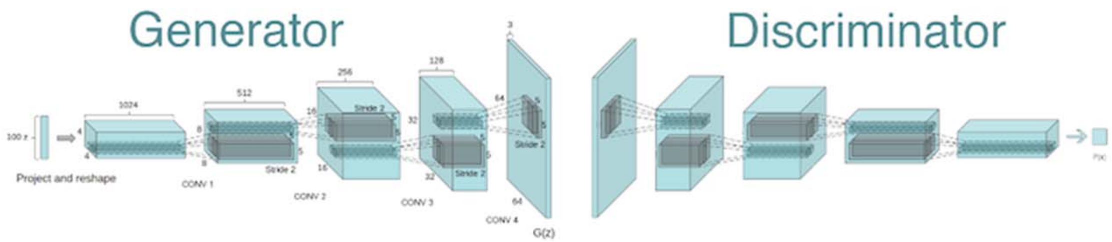
GANs architecture

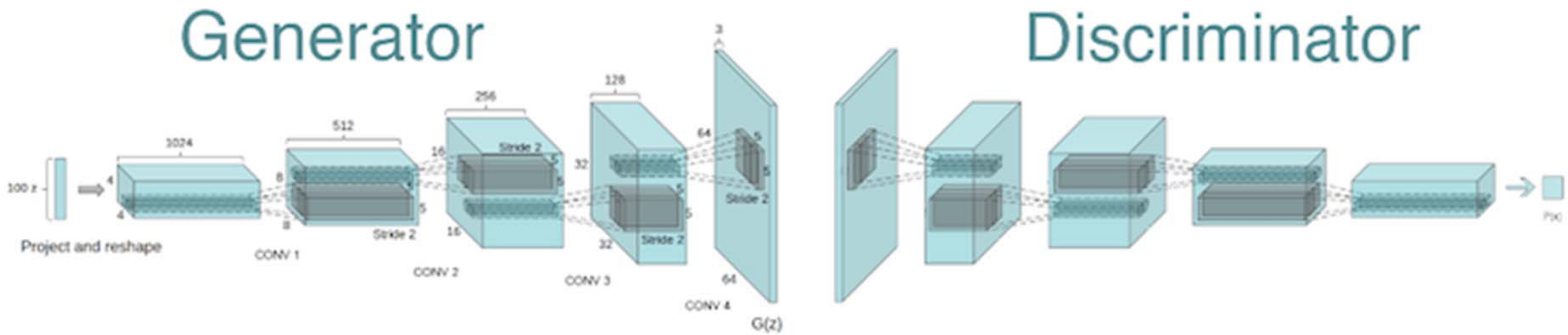
Main guidelines from the literature (Goodfellows et al., 2015, Radford et al., 2016)

Generator is an up-sampling network which implements deep deconvolution
Discriminator is a convolutional network (some other authors have also proposed fully connected FF)

- Replace any pooling layers with strided (stride length>1) convolutions (discriminator) and fractional-strided convolutions (generator)
- No need of fully-connected hidden layers
- Use ReLU (rectifier linear unit) activation in generator for all layers except the output that uses Tanh
- Use LeakyReLU activation (https://keras.io/layers/advanced-activations/) in the discriminator for all layers

f(x) = alpha * x for x < 0, f(x) = x for x >= 0.
It allows a small gradient when the unit is not active

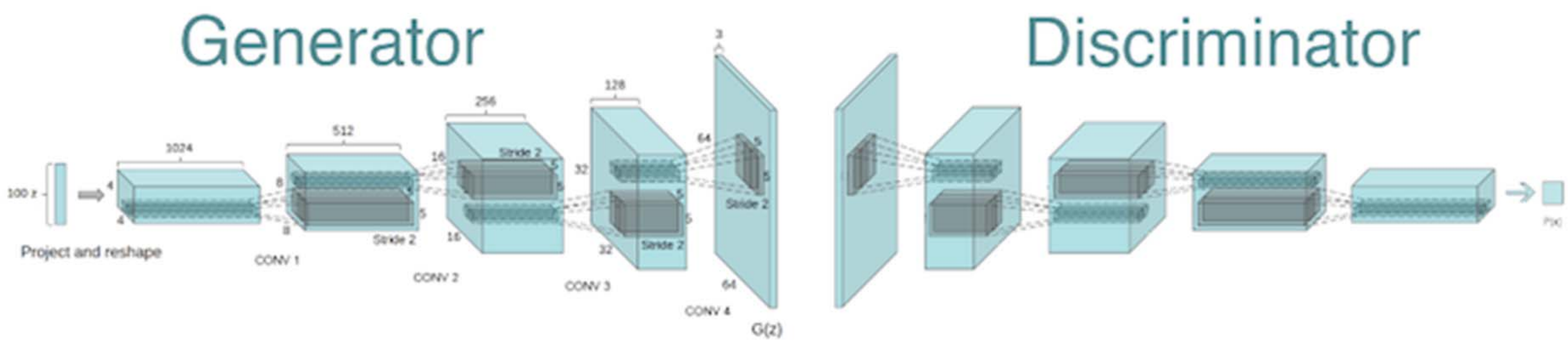




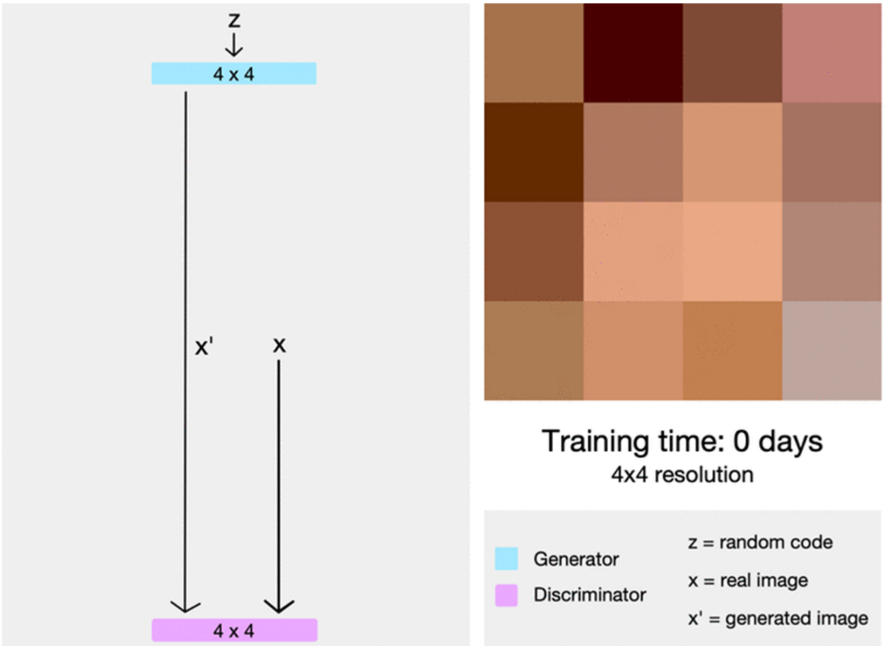
DCGAN, Radford et al., 2016 https://arxiv.org/pdf/1511.06434.pdf

Generated bed rooms

deep convolutional generative adversarial network

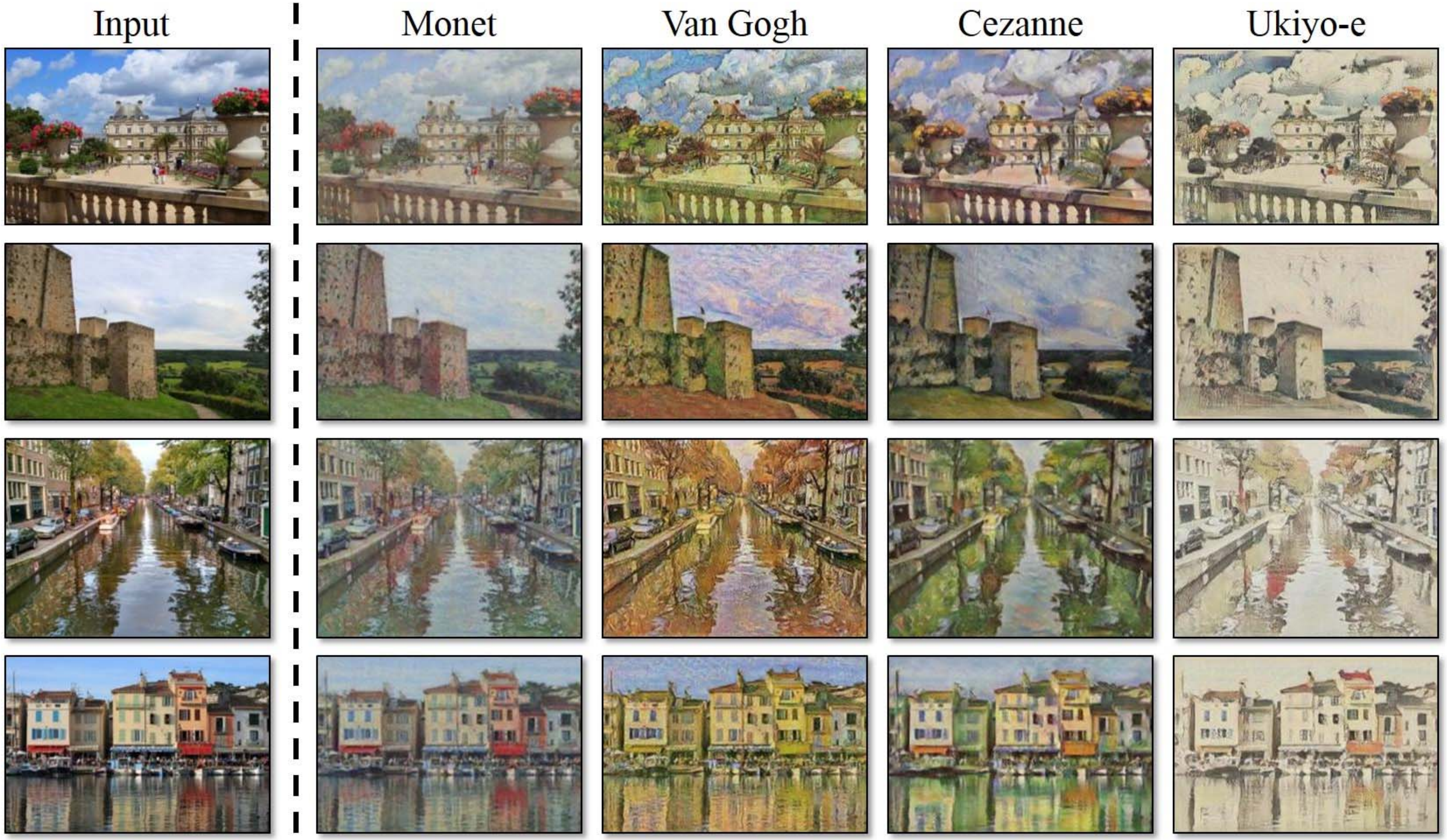


ProGAN:
Karras et al. 2017



Progressive growing of generative adversarial networks

Image-to-image translation – CycleGAN, Zhu et al., 2017



https://junyanz.github.io/CycleGAN/

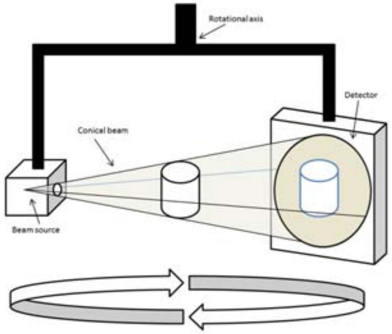
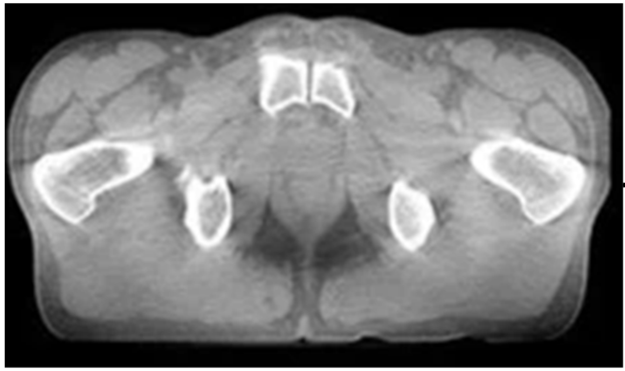


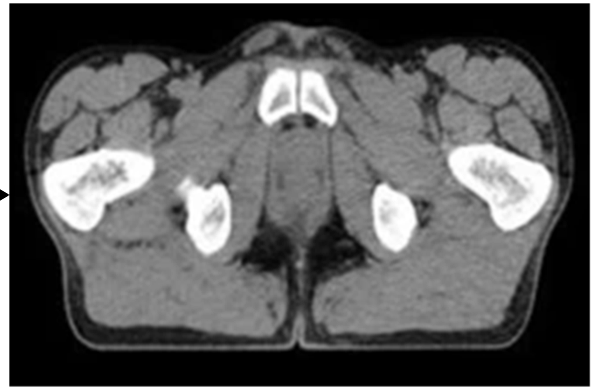
Image to image translation in biomedicine

Objective: to consider images belonging to domain A and transform them into a different domain B by transferring the typical characteristics of domain B



Cone Beam CT

Cyclic
Coherence
GAN



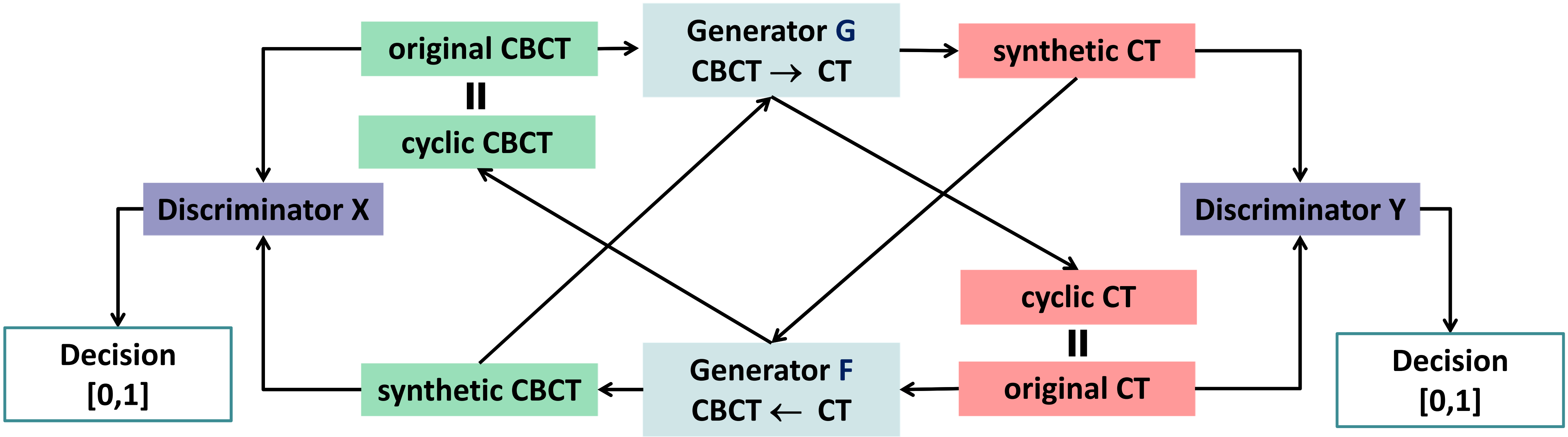
CT

Scientific literature
Kurz et al., Physics in Medicine & Biology, 2019
Liang et al., Physics in Medicine & Biology, 2019
Kida et al., 2019
Yang et al., Scientific Reports, 2020
Tiem et al., Scientific Reports, 2021
Rossi et al., Diagnostics 2021

Applications
- Radiotherapy: re-planning and adaptation during treatment sessions
- Diagnostics: use in medical centres with more limited resources

Cyclic Coherence GAN

$$L_{cyc}(G) = E_{x\sim p_{data}}(F(G(x)) - x)$$



$$L_{cyc}(F) = E_{y\sim p_{data}}(G(F(y)) - y)$$

$$L_X(F, D_X, X, Y) = \left[E_{x\sim p_{data}} log D_X(x) + E_{(y)\sim p_{(y)}} log\left(1 - D_X\big(F(y)\big)\right)\right]$$

x: original CBCT, F(y): synthetic CBCT, y: original CT

$$L_Y(G, D_Y, X, Y) = \left[E_{y\sim p_{data}} log D_Y(y) + E_{(x)\sim p_{(x)}} log\left(1 - D_Y\big(G(x)\big)\right)\right]$$

x: original CBCT, G(x): synthetic CT, y: original CT

GANs synthesis

- It does not need a priori density function of the generator
- It learns how to generate from training data through 2-player game
- It is subject of intense research: evolving architectures

but

- It is more unstable to train wrt traditional FFNN and CNN
- There is active research to improve quality and speed up training