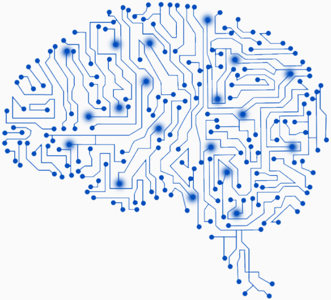
Master 2 Statistique & Data Science, Ingénierie Mathématique

Réseaux de neurones profonds pour l'apprentissage
Deep neural networks for machine learning

Course V – Introduction to Artificial Neural Networks: Generative Models

Bruno Galerne

2024-2025



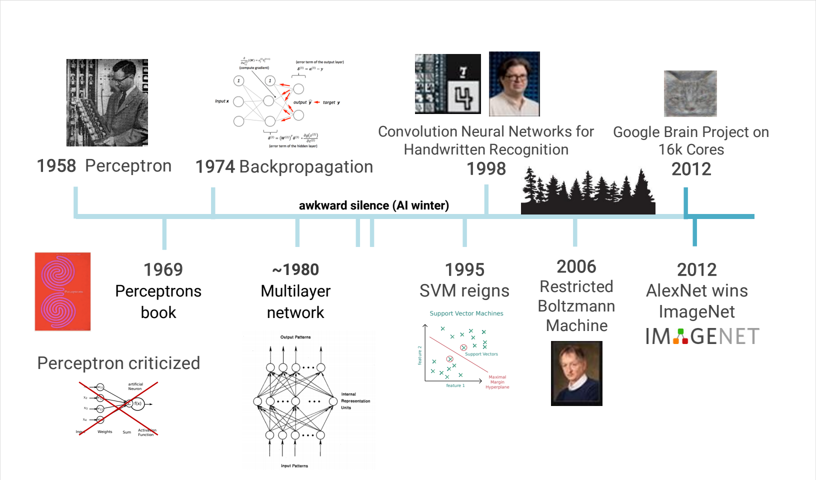
Credits

Most of the slides from **Charles Deledalle's** course "UCSD ECE285 Machine learning for image processing" (30 $\times$ 50 minutes course)

www.charles-deledalle.fr/

https://www.charles-deledalle.fr/pages/teaching.php#learning

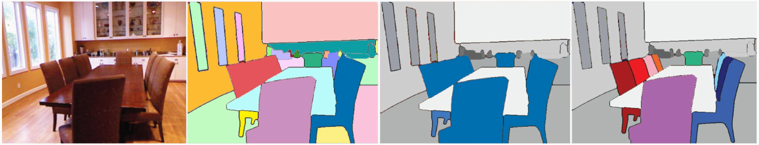
Timeline of (deep) learning



(Source: Lucas Masuch & Vincent Lepetit)

Segmentation

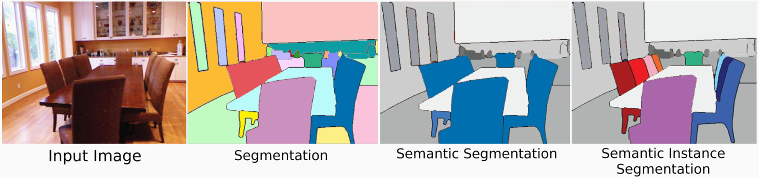
Segmentation – Terminology



Input Image | Segmentation | Semantic Segmentation | Semantic Instance Segmentation

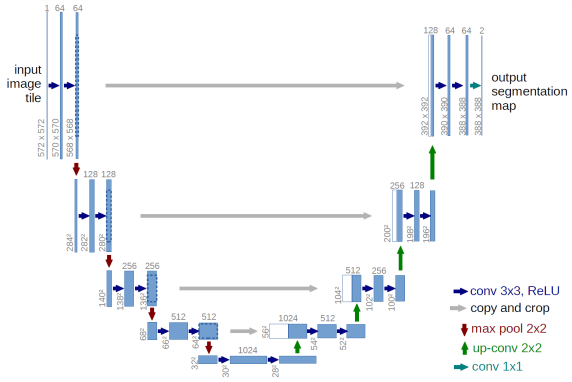
- **Segmentation:**
  - Partition of an image into several "coherent" parts/segments,
  - Without any attempt at understanding what these parts represent,
  - Typically based on color, textures, smoothness of boundaries,
  - Also referred to as super-pixel segmentation.

Segmentation – Terminology



- **Semantic segmentation:**
    - Each segment corresponds to a class label (objects + background),
    - Also referred to as scene parsing or scene labeling.
- **Instance segmentation:**
    - Find object boundaries between objects, including delineations between instances of the same object.
- **Semantic instance segmentation:** find object boundaries + labels.

U-net for image segmentation



(source: From [Ronneberger et al., 2015])

- First proposed in [Ronneberger et al., 2015].
- Idea: Classify each pixel
- Condense spatial information as for image classification.
- Re-affine spatially the classification step by step with mirror upsampling steps (transpose of conv2D with padding) and concatenation.

U-net for image segmentation

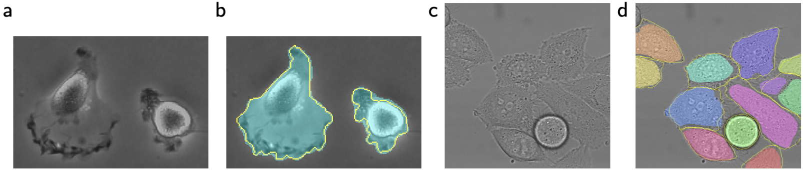
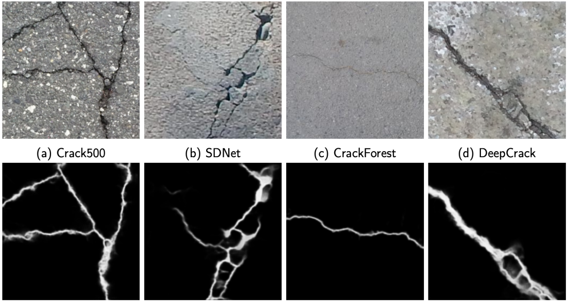


Fig. 4. Result on the ISBI cell tracking challenge. (**a**) part of an input image of the "PhC-U373" data set. (**b**) Segmentation result (cyan mask) with manual ground truth (yellow border) (**c**) input image of the "DIC-HeLa" data set. (**d**) Segmentation result (random colored masks) with manual ground truth (yellow border).

(source: From [Ronneberger et al., 2015])

- Improved state-of-the-art in cell-tracking.
- Can be extended to very different contexts provided enough labeled data.

U-net for image segmentation



(a) Crack500 (b) SDNet (c) CrackForest (d) DeepCrack

(e) Prediction on (a) (f) Prediction on (b) (g) Prediction on (c) (h) Prediction on (d)

(source: From [Drouyer, 2020])

- Example usage: Crack detection
- The network outputs the probability that each pixel belongs to a crack.

More generally a U-net can be trained to produce an image aligned with the input image.

- Segmentation [Ronneberger et al., 2015]
- Denoising (see e.g. DRUNet [Zhang et al., 2022])
- Image to image translation (Pix2pix [Isola et al., 2017])
- Inverse problems: trained to remove artefacts from a crude solution:

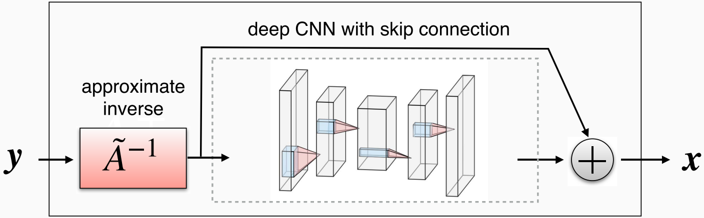
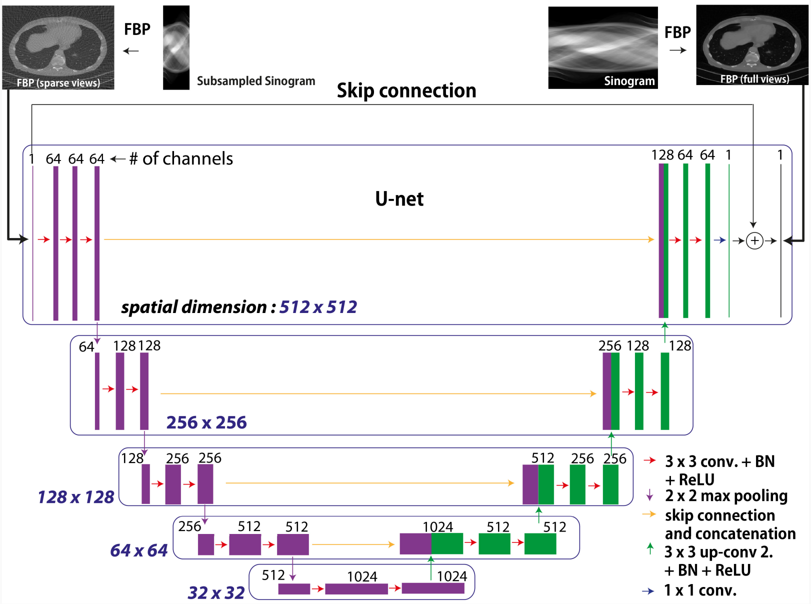


Fig. 7. When an approximate inverse $\tilde{A}^{-1}$ of the forward model is known, a common approach in the supervised setting is to train a deep CNN to remove noise and artifacts from an initial reconstruction obtained by applying $\tilde{A}^{-1}$ to the measurements.

(source: [Ongie et al., 2020])

U-net for inverse problems

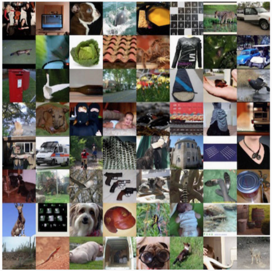


(source: [Jin et al., 2017])

Image generation

Motivations – Image generation

- **Goal:** Generate images that look like the ones of your training set.



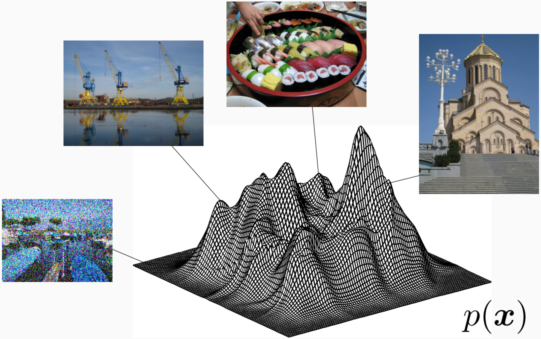
Real images (ImageNet)
(Training set)

Generated images
(Results)

- **What?** Unsupervised learning.
- **Why?** Different reasons and applications:
  - Can be used for simulation, *e.g.*, to generate labeled datasets,
  - Must capture all subtle patterns → provide good features,
  - Can be used for other tasks: super-resolution, style transfer, . . .

Image generation – Explicit density

❶ Learn the distribution of images $p(\boldsymbol{x})$ on a training set.



❷ Generate samples from this distribution.

Image generation – Gaussian model

- Consider a Gaussian model for the distribution of images $\boldsymbol{x}$ with n pixels:

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$p(\boldsymbol{x}) = \frac{1}{\sqrt{2\pi}^{n} |\boldsymbol{\Sigma}|^{1/2}} \exp\left[(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})\right]$$

  - $\boldsymbol{\mu}$: mean image,
  - $\boldsymbol{\Sigma}$: covariance matrix of images.

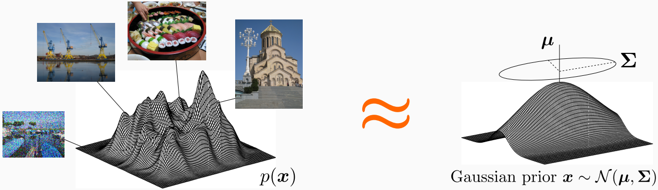


Image generation – Gaussian model

- Take a training dataset $\mathcal{T}$ of images:

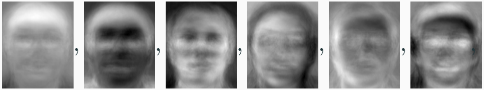
$$\mathcal{T} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$$

$= \Big\{$  $, \ldots \Big\}_{\times N}$

- Estimate the mean

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_i \boldsymbol{x}_i =$$

- Estimate the covariance matrix: $\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_i (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^T = \hat{\boldsymbol{E}} \hat{\boldsymbol{\Lambda}} \hat{\boldsymbol{E}}^T$

$\hat{\boldsymbol{E}} = \Big\{$  $, \ldots \Big\}_{\times N}$

eigenvectors of $\hat{\boldsymbol{\Sigma}}$, *i.e.*, main variation axis

Image generation – Gaussian model

You now have learned a generative model:

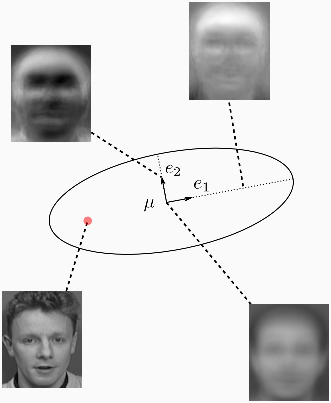


Image generation – Gaussian model

How to generate samples from $\mathcal{N}(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$?

$$\begin{cases} \boldsymbol{z} & \sim \mathcal{N}(0, \mathrm{Id}_n) \quad \leftarrow \text{Generate random latent variable} \\ \boldsymbol{x} & = \hat{\boldsymbol{\mu}} + \hat{\boldsymbol{E}}\hat{\boldsymbol{\Lambda}}^{1/2}\boldsymbol{z} \end{cases}$$

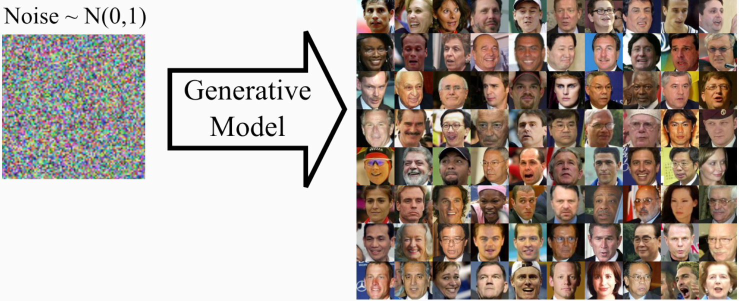
The model does not generate realistic faces.

The Gaussian distribution assumption is too simplistic.

Each generated image is just a linear random combination of the eigenvectors.

The generator corresponds to a linear neural network (without non-linearities).

Image generation – Beyond Gaussian models



But the concept is interesting: can we find a transformation such that each random code can be mapped to a photo-realistic image?

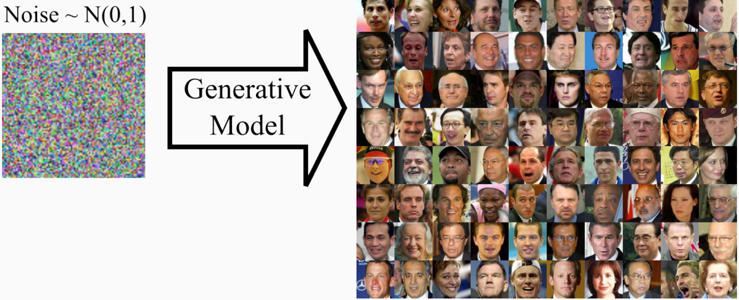
We need to find a way to assess if an image is photo-realistic.

Main references:

1. Original paper: [Goodfellow et al., 2014]
2. NIPS 2016 tutorial: [Goodfellow, 2017]

Generative Adversarial Networks (GAN)

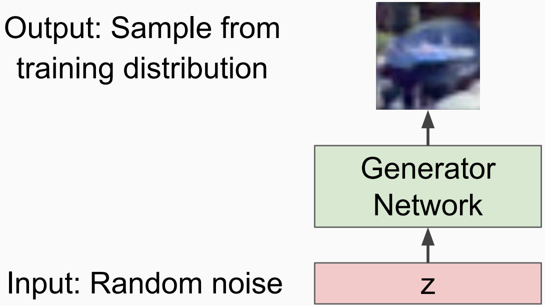
Image generation – Beyond Gaussian models



Generative Adversarial Networks (GAN)

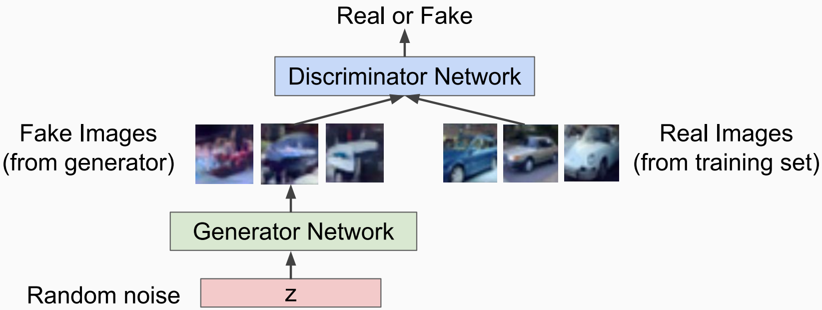
- **Goal:** design a complex model with high capacity able to map latent random noise vectors $\boldsymbol{z} \in \mathbb{R}^k$ to a realistic image $\boldsymbol{x} \in \mathbb{R}^d$.

- **Idea:** Take a deep neural network



- **What about the loss?** Measure if the generated image is **photo-realistic**.

Define a loss measuring how much you can fool a classifier that has learned to distinguish between real and fake images.



- **Discriminator network:** Try to distinguish between real and fake images.
- **Generator network:** Fool the discriminator by generating realistic images.

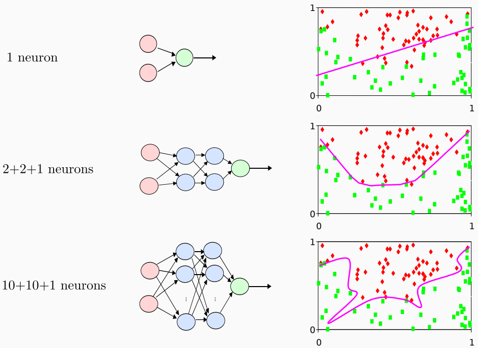
Recap on binary classification

- Given a labeled dataset

$$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)}),\ i = 1, \dots, N\} \subset \mathbb{R}^d \times \{0, 1\}$$

  with **binary labels** $y^{(i)} \in \{0, 1\}$ that corresponds to two classes C_0 and C_1.
- A **parametric classifier** $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to [0, 1]$ outputs a probability such that

$$\boldsymbol{p} = f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \in C_1) \quad \text{and} \quad 1 - \boldsymbol{p} = 1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x} \in C_0)$$



Estimated decision regions: $\hat{C}_1 = \{\boldsymbol{x} \in \mathbb{R}^d,\ f_{\boldsymbol{\theta}}(\boldsymbol{x}) \geqslant \frac{1}{2}\}$ and $\hat{C}_0 = \mathbb{R}^d \setminus \hat{C}_1$.

Complexity/capacity of the network

$\Rightarrow$

Trade-off between generalization and overfitting.

Recap on binary classification

- **Training:** Logistic regression for binary classification: Maximum likelihood of the dataset (opposite of binary crosss-entropy : `BCELoss` in PyTorch):

$$\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} \left[y^{(i)} \log f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + (1 - y^{(i)}) \log \left(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) \right]$$

- For neural networks, the probability $f_{\boldsymbol{\theta}}$ is obtained using the **sigmoid function** $\sigma(t) = \frac{e^t}{1 + e^t} = \frac{1}{1 + e^{-t}}$ as the activation function of the last layer.
- Beware that $y^{(i)} = 0$ or 1 so only one term is non-zero.
- One could instead regroup the terms of the sum according to the label values:

$$\max_{\boldsymbol{\theta}} \sum_{\substack{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \text{s.t. } y^{(i)} = 1}}^{N} \log f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) + \sum_{\substack{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D} \\ \text{s.t. } y^{(i)} = 0}}^{N} \log \left(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right)$$

- **Discriminator network:** Consider two sets
  - $\mathcal{D}_{\text{real}}$: a dataset of n real images **(real = labeled with $y^{(i)} = 1$)**,
  - $\mathcal{D}_{\text{fake}}$: a dataset of m fake images $\boldsymbol{x} = G_{\theta_g}(\boldsymbol{z})$ **(fake = labeled with $y^{(i)} = 0$)**.

- **Goal:** Find the parameters θ_d of a binary classification network $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ meant to classify real and fake images.
  Minimize the binary cross-entropy, or maximize its negation

$$\max_{\theta_d} \underbrace{\sum_{\boldsymbol{x}_{\text{real}} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}_{\text{real}})}_{\substack{\text{force predicted labels to be 1} \\ \text{for real images}}} + \underbrace{\sum_{\boldsymbol{x}_{\text{fake}} \in \mathcal{D}_{\text{fake}}} \log(1 - D_{\theta_d}(\boldsymbol{x}_{\text{fake}}))}_{\substack{\text{force predicted labels to be 0} \\ \text{for fake images}}}$$

- **How:** Use gradient ascent (Adam).

Generative Adversarial Networks (GAN)

- **Generator network:** Consider a given discriminative model $\boldsymbol{x} \mapsto D_{\theta_d}(\boldsymbol{x})$ and consider $\mathcal{D}_{\text{rand}}$ a set of m random latent vectors.
- **Goal:** Find the parameters θ_g of a network $\boldsymbol{z} \mapsto G_{\theta_g}(\boldsymbol{z})$ generating images from random vectors $\boldsymbol{z}$ such that it fools the discriminator

$$\min_{\theta_g} \quad \underbrace{\sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(G_{\theta_g}(\boldsymbol{z})))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are not fake (away from 0)}}} \tag{1}$$

or alternatively (works better in practice)

$$\max_{\theta_g} \quad \underbrace{\sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log D_{\theta_d}(G_{\theta_g}(\boldsymbol{z})))}_{\substack{\text{force the discriminator to think that} \\ \text{our generated fake images are real (close to 1)}}} \tag{2}$$

- **How:** Gradient descent for (1) or gradient ascent for (2) (Adam)

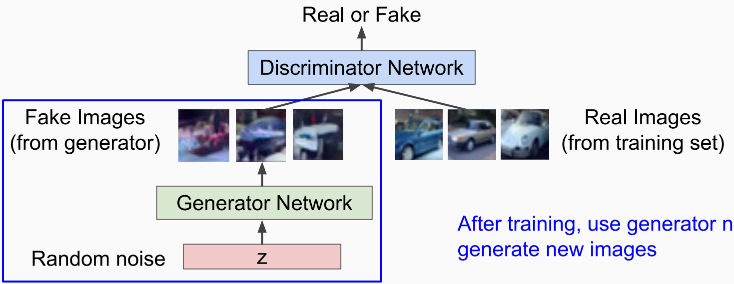
Generative Adversarial Networks (GAN)

- **Train both networks jointly**.
- Minimax loss in a two player game (each player is a network):

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

- **Training algorithm:** Repeat until convergence
  1. Fix θ_g, update θ_d with one step of gradient ascent,
  2. Fix θ_d, update θ_g with one step of gradient descent for (1),
     (or one step of gradient ascent for (2).)

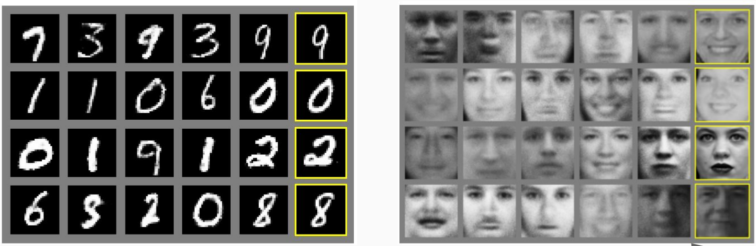
Generative Adversarial Networks (GAN)



After training, use generator network to generate new images

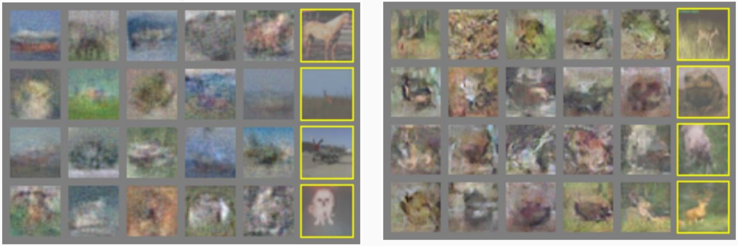
Generative Adversarial Networks (GAN)

Generated samples



Nearest neighbor from training set

Generated samples (CIFAR-10)

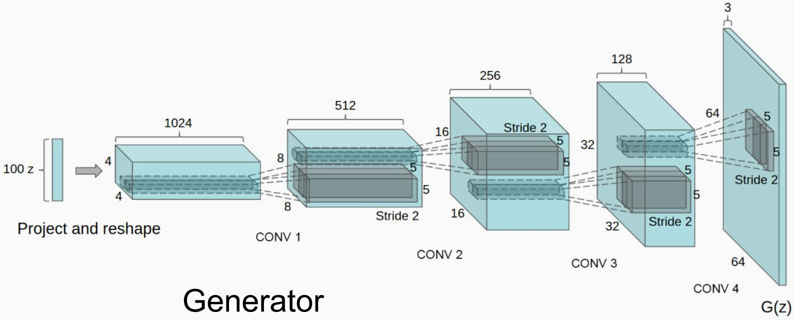


Nearest neighbor from training set

Convolutional GAN

[Radford et al., 2016]

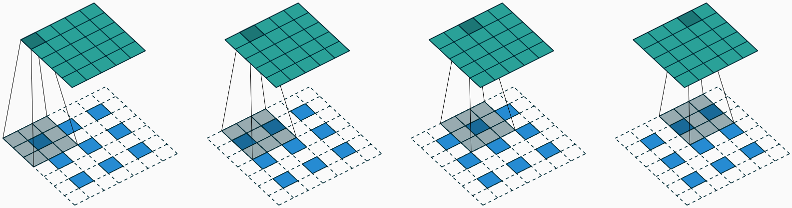
- **Generator:** upsampling network with **fractionally strided convolutions** (i.e. the transpose operator of convolution+subsampling , called `ConvTranspose2d` in PyTorch),
- **Discriminator:** convolutional network with strided convolutions.



Transposed convolution arithmetic

Fractionally strided convolutions:

- This is the transpose operator of convolution+subsampling (convolution with stride).
- Called `ConvTranspose2d` in PyTorch



The transpose of convolving a 3×3 kernel over a 5×5 input padded with a 1×1 border of zeros using 2×2 strides (i.e., $i = 5$, $k = 3$, $s = 2$ and $p = 1$). It is equivalent to convolving a 3×3 kernel over a 3×3 input (with 1 zero inserted between inputs) padded with a 1×1 border of zeros using unit strides (i.e., $i' = 3$, $\tilde{i}' = 5$, $k' = k$, $s' = 1$ and $p' = 1$).

(source: From [Dumoulin and Visin, 2016])

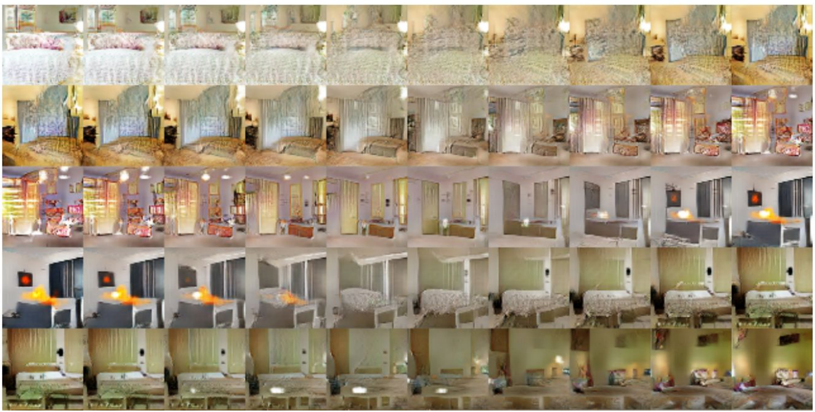
Convolutional GAN

[Radford et al., 2016]

Generations of realistic bedrooms pictures, from randomly generated latent variables.

Convolutional GAN

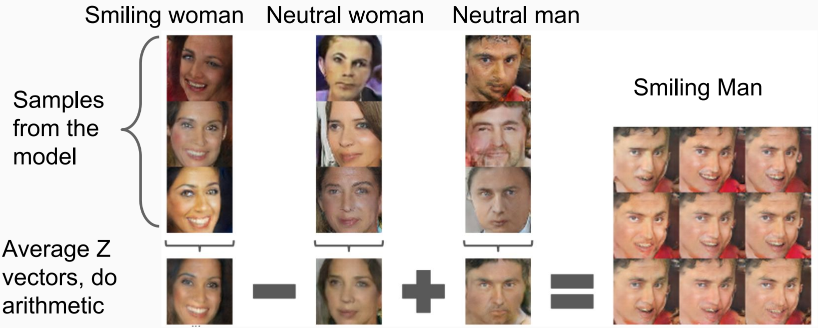
[Radford et al., 2016]



Interpolation in between points in latent space.

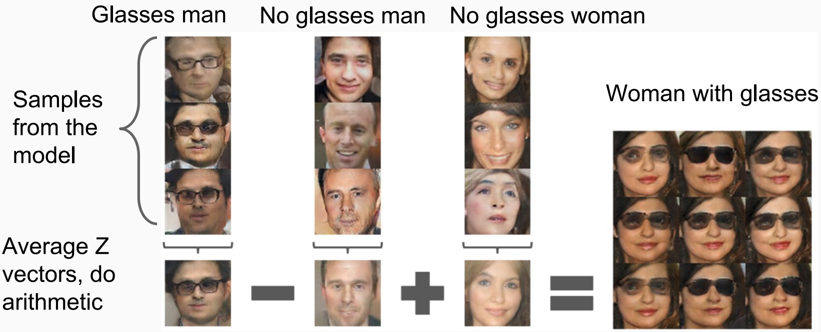
Convolutional GAN – Arithmetic

[Radford et al., 2016]



Convolutional GAN – Arithmetic

[Radford et al., 2016]



Generative Aversarial Networks: Style GAN [Karras et al., 2019]

Image size:
1024× 1024 px
(source: Karras et al.)

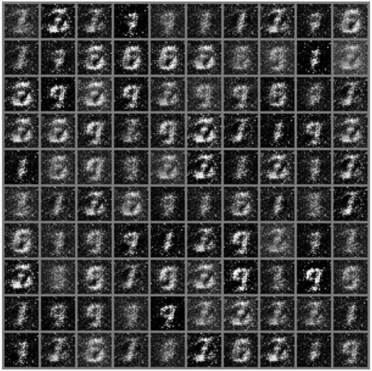
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 1:

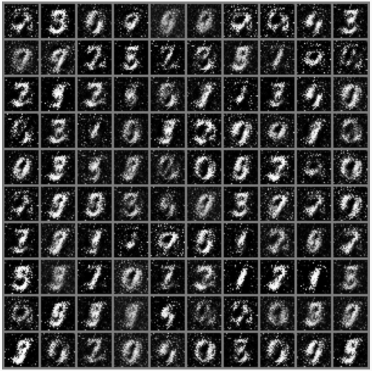


Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 2:



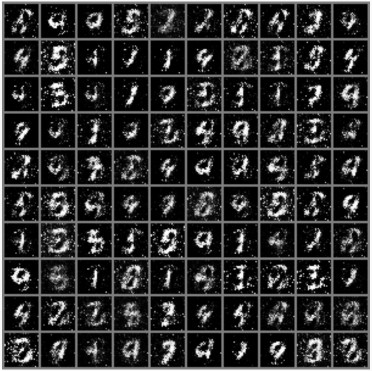
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 3:



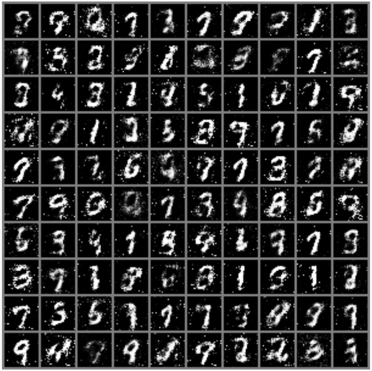
GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 10:



GAN Training

Training with MNIST (60 000 images)

- Adam optimizer
- Learning rate 0.0002 for both the discriminator and the generator

Real images:

Fake images, epoch 100:

GAN Training

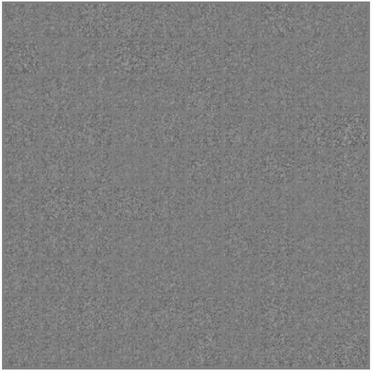
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 1:



GAN Training

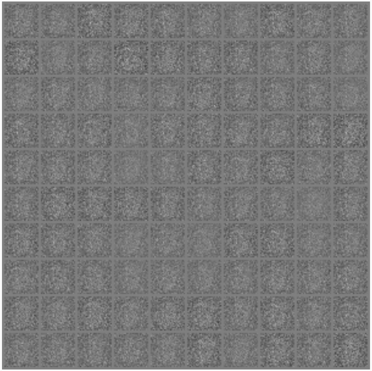
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 2:



GAN Training

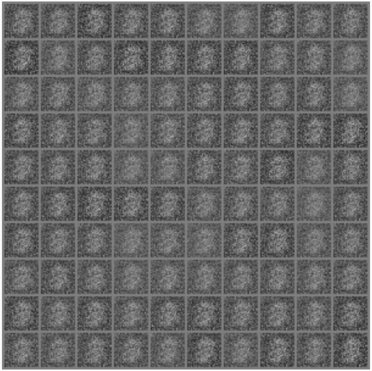
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 3:



GAN Training

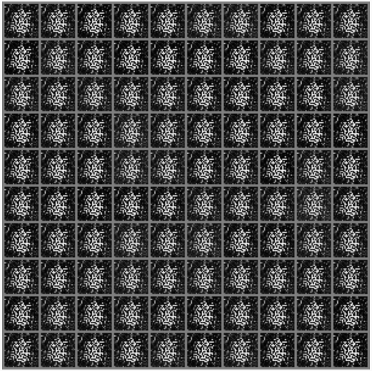
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 10:



GAN Training

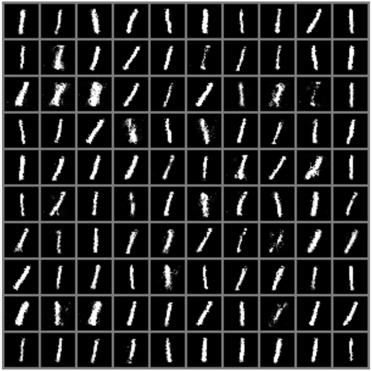
Training GANs is quite unstable!

The generator can suffer *mode collapse*: It always produces the same image (one mode only).

Same as before **but with SGD instead of Adam**.

Real images:

Fake images, epoch 100:



Conditional GANs

- Conditional GANs: Train the generator and the discriminator by passing some information about the images.

Example: Class conditional generator and discriminator

- **Generator:** Generate a fake "3".
- **Discriminator:** Is it a real or a fake "3"?

Unconditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

Class conditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{(\boldsymbol{x}, c) \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}, c) + \sum_{(\boldsymbol{z}, c) \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z}, c)}_{\text{fake}}, c))$$

Conditional GANs

- Conditional GANs: Train the generator and the discriminator by passing some information about the images.

Example: Class conditional generator and discriminator

- **Generator:** Generate a fake "3".
- **Discriminator:** Is it a real or a fake "3"?

Unconditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

Class conditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{(\boldsymbol{x}, c) \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}, c) + \sum_{(\boldsymbol{z}, c) \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z}, c)}_{\text{fake}}, c))$$

- **Discussion:** How to do this in practice?

Conditional GANs

- Conditional GANs: Train the generator and the discriminator by passing some information about the images.

Example: Class conditional generator and discriminator

- **Generator:** Generate a fake "3".
- **Discriminator:** Is it a real or a fake "3"?

Unconditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{\boldsymbol{x} \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}) + \sum_{\boldsymbol{z} \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z})}_{\text{fake}}))$$

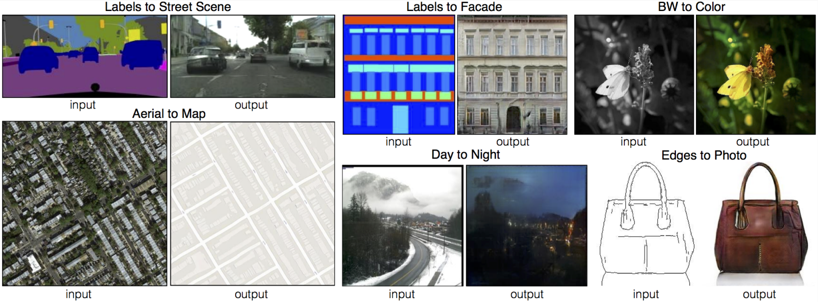
Class conditional training:

$$\min_{\theta_g} \max_{\theta_d} \sum_{(\boldsymbol{x}, c) \in \mathcal{D}_{\text{real}}} \log D_{\theta_d}(\boldsymbol{x}, c) + \sum_{(\boldsymbol{z}, c) \in \mathcal{D}_{\text{rand}}} \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{z}, c)}_{\text{fake}}, c))$$

- **Discussion:** How to do this in practice? Use `torch.nn.Embedding`.

Conditional GANs: image-to-image translation

Pix2pix: Image-to-Image Translation with Conditional Adversarial Nets [Isola et al., 2017]



(source: From [Isola et al., 2017])

- Training using a set of image pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
- GAN conditioned on input image $\boldsymbol{x}$ to produce $\boldsymbol{y} = G(\boldsymbol{x})$.
- Opens the way for new creative tools

Pix2pix: Image-to-Image Translation with Conditional Adversarial Nets [Isola et al., 2017]

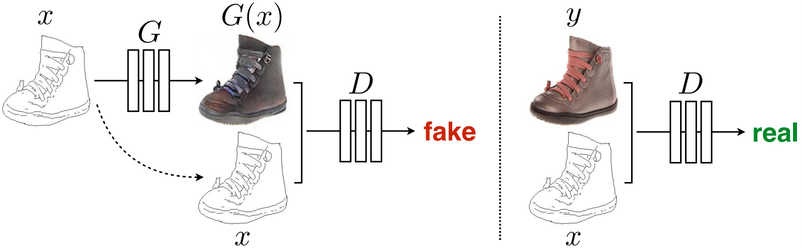
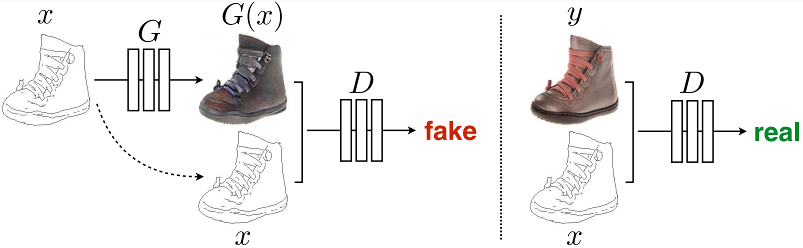


Figure 2: Training a conditional GAN to map edges→photo. The discriminator, D, learns to classify between fake (synthesized by the generator) and real {edge, photo} tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

(source: From [Isola et al., 2017])

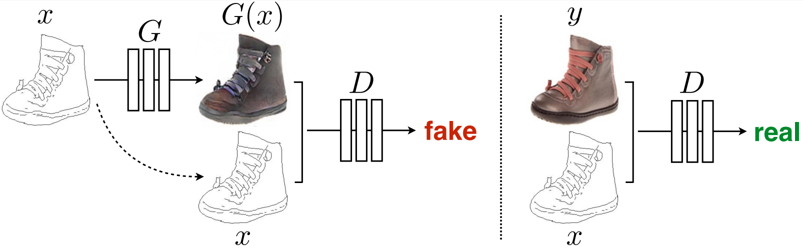


(source: From [Isola et al., 2017])

Architecture details:

- Generator: **U-net architecture** (see recap later)
- Discriminator: Patch discriminator applied to each 70×70 patch, and the score is spatially average.
- Both are fully convolutional so larger images can be used at test time.

Conditional GANs: image-to-image translation



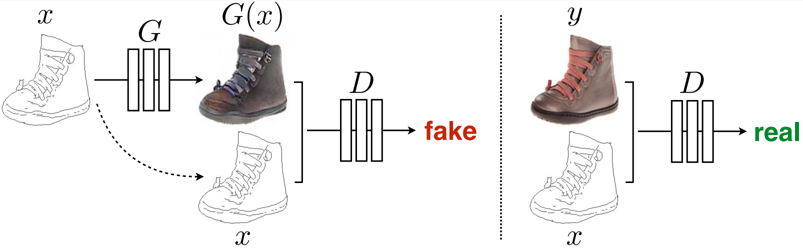
(source: From [Isola et al., 2017])

Architecture details:

- Generator: **U-net architecture** (see recap later)
- Discriminator: Patch discriminator applied to each 70×70 patch, and the score is spatially average.
- Both are fully convolutional so larger images can be used at test time.

Question: What is missing here?

Conditional GANs: image-to-image translation



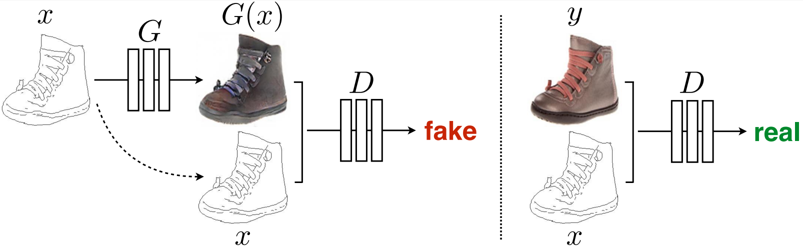
(source: From [Isola et al., 2017])

Architecture details:

- Generator: **U-net architecture** (see recap later)
- Discriminator: Patch discriminator applied to each 70×70 patch, and the score is spatially average.
- Both are fully convolutional so larger images can be used at test time.

Question: What is missing here? No latent code z in the generator, but randomness thanks to dropout in the network. Still stochasticity is limited given an input x.

Conditional GANs: image-to-image translation



(source: From [Isola et al., 2017])

Training loss:

- The training of generator combines a GAN loss and an ℓ_1 loss:

$$\min_{\theta_g} \max_{\theta_d} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log D_{\theta_d}(\boldsymbol{y},\boldsymbol{x}) + \log(1 - D_{\theta_d}(\underbrace{G_{\theta_g}(\boldsymbol{x})}_{\text{fake}},\boldsymbol{x}) + \| \underbrace{G_{\theta_g}(\boldsymbol{x})}_{\text{fake}} - \boldsymbol{y}\|_1$$

- The discriminator looks at generated patches while the ℓ_1 loss is global.
- This is a mixed loss between classical supervised training and unsupervised GAN training.

Conditional GANs: image-to-image translation

Figure 4: Different losses induce different quality of results. Each column shows results trained under a different loss. Please see `https://phillipi.github.io/pix2pix/` for additional examples.

(source: From [Isola et al., 2017])

Adversarial loss for inverse problems

- With the pix2pix example we see that an adversarial loss is used to produce realistic images.
- Using only ℓ_1 or ℓ_2 loss leads to blurry results. This is known as "regression to the mean" issue.
- Adversarial losses helps to improve the visual aspect, but can also hallucinate details. Use with care in scientific context.
- Introducing generative capabilities to networks is especially important for tasks where content has to be inferred with few to no available information, such as **super-resolution**.

Super-resolution

SRGAN [Ledig et al., 2017]

Adversarial loss: Same as GAN but replace the latent code by the LR image

$$\min_{\theta_{\text{SR}}} \max_{\theta_{\text{D}}} \sum \log D_{\theta_D}(\boldsymbol{x}^{\text{HR}}) + \log(1 - D_{\theta_D}(\boldsymbol{x}^{\text{SR}})) + \lambda L_{\text{content}}(\boldsymbol{x}^{\text{SR}}, \boldsymbol{x}^{\text{HR}})$$

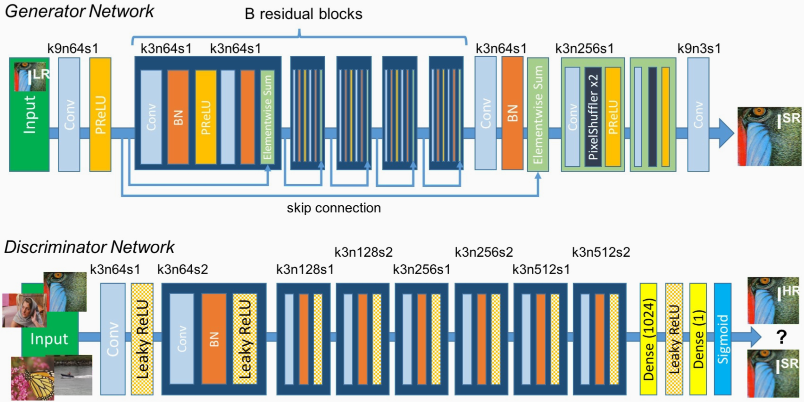
$$\text{where} \quad \boldsymbol{x}^{\text{SR}} = G_{\theta_{\text{SR}}}(\boldsymbol{x}^{\text{LR}}) \quad \text{and} \quad \boldsymbol{x}^{\text{LR}} = \boldsymbol{H}(\boldsymbol{x}^{\text{HR}})$$

Content loss: Euclidean distance between the L^{th} feature tensors obtained with VGG for the SR and HR images, respectively:

$$L_{\text{content}}(\boldsymbol{x}^{\text{SR}}, \boldsymbol{x}^{\text{HR}}) = \|\boldsymbol{h}^{\text{SR}} - \boldsymbol{h}^{\text{HR}}\|_2^2 \quad \text{with} \quad \begin{cases} \boldsymbol{h}^{\text{SR}} = \text{VGG}^L(\boldsymbol{x}^{\text{SR}}) \\ \boldsymbol{h}^{\text{HR}} = \text{VGG}^L(\boldsymbol{x}^{\text{HR}}) \\ \boldsymbol{x}^{\text{SR}} = \text{G}_{\theta_{\text{SR}}}(\boldsymbol{x}^{\text{LR}}) \end{cases}$$

- Force images to have similar high level feature tensors.
- Supposed to be closer to perceptual similarity [Johnson et al., 2016].

SRGAN [Ledig et al., 2017]



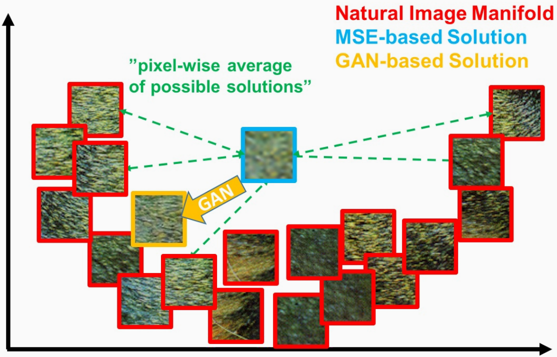
Architecture of Generator and Discriminator Network with corresponding kernel size (k), number of feature maps (n) and stride (s) indicated for each convolutional layer.

(source: From [Ledig et al., 2017])

Both networks are trained by alternating their gradient based updates.

Super-resolution

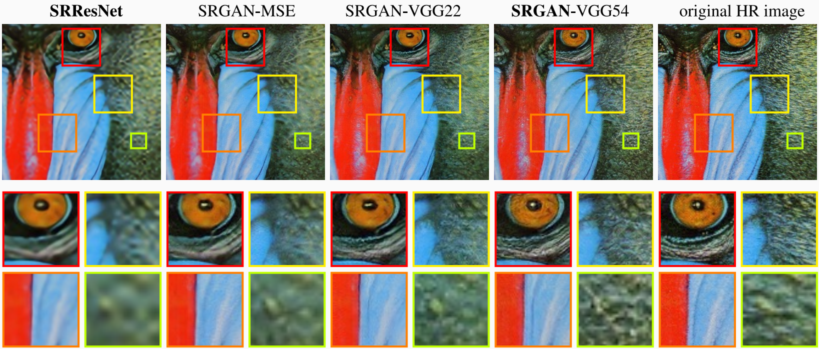
SRGAN [Ledig et al., 2017]



(source: From [Ledig et al., 2017])

- The SR problem is ill-posed → infinite number of solutions,
- MSE promotes a pixel-wise average of them → over-smooth,
- GAN drives reconstruction towards the "natural image manifold".

Super-resolution



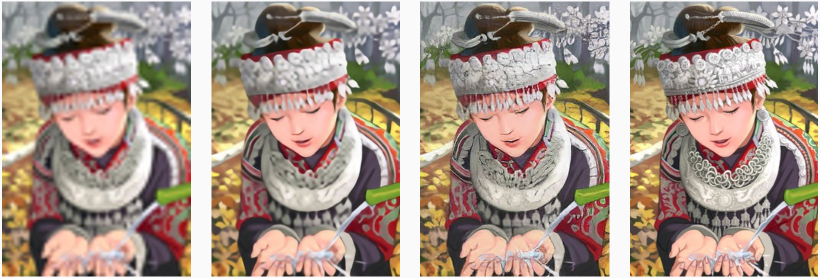
(source: From [Ledig et al., 2017])

×4 upsampling (16× more pixels)

- SRResNet: ResNet SR generator trained with MSE,
- SRGAN-MSE: generator and discriminator with MSE content loss,
- SRGAN-VGG22: generator and discriminator with VGG22 content loss,
- SRGAN-VGG54: generator and discriminator with VGG54 content loss.

Super-resolution

bicubic (21.59dB/0.6423) | SRResNet (23.53dB/0.7832) | SRGAN (21.15dB/0.6868) | original



(source: From [Ledig et al., 2017])

×4 upsampling (16× more pixels)

- Even though some details are lost, they are replaced by “fake” but photo-realistic objects (instead of blurry ones).
- Remark that SRResNet is blurrier but achieves better PSNR.

Drouyer, S. (2020).
An 'All Terrain' Crack Detector Obtained by Deep Learning on Available Databases.
Image Processing On Line, 10:105–123.
https://doi.org/10.5201/ipol.2020.282.

Dumoulin, V. and Visin, F. (2016).
A guide to convolution arithmetic for deep learning.
ArXiv e-prints.

Goodfellow, I. (2017).
Nips 2016 tutorial: Generative adversarial networks.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative adversarial nets.
In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017).
Image-to-image translation with conditional adversarial networks.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jin, K. H., McCann, M. T., Froustey, E., and Unser, M. (2017).
Deep convolutional neural network for inverse problems in imaging.
IEEE Transactions on Image Processing, 26(9):4509–4522.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016).
Perceptual losses for real-time style transfer and super-resolution.
In Leibe, B., Matas, J., Sebe, N., and Welling, M., editors, *Computer Vision – ECCV 2016*, pages 694–711, Cham. Springer International Publishing.

Karras, T., Laine, S., and Aila, T. (2019).
A style-based generator architecture for generative adversarial networks.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and Shi, W. (2017).
Photo-realistic single image super-resolution using a generative adversarial network.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ongie, G., Jalal, A., Metzler, C. A., Baraniuk, R. G., Dimakis, A. G., and Willett, R. (2020).
Deep learning techniques for inverse problems in imaging.
IEEE Journal on Selected Areas in Information Theory, 1(1):39–56.

Radford, A., Metz, L., and Chintala, S. (2016).
Unsupervised representation learning with deep convolutional generative adversarial networks.
In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Ronneberger, O., Fischer, P., and Brox, T. (2015).
U-net: Convolutional networks for biomedical image segmentation.
In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

Zhang, K., Li, Y., Zuo, W., Zhang, L., Van Gool, L., and Timofte, R. (2022).
Plug-and-play image restoration with deep denoiser prior.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 44(10):6360–6376.

Questions?

(Most) Slides from Charles Deledalle

Sources, images courtesy and acknowledgment

K. Chatfield
P. Gallinari,
C. Hazırbaş
A. Horodniceanu
Y. LeCun
V. Lepetit
L. Masuch
A. Ng
M. Ranzato