Large-Scale GAN

Arthur Leclaire

TELECOM Paris

IP PARIS

MVA Generative Modeling
January 27th, 2026

Plan

Popular Image Databases

- MNIST (digits): $60k$ images with 28^2 px (10 classes)
- Fashion-MNIST (clothes): $70k$ images with 28^2 px (10 classes)
- CIFAR-10: $60k$ images with 32^2 px (10 classes)
- ImageNet: $\approx 1430k$ images of various size (1000 classes)
- CelebA: $\approx 200k$ images with 178×278 px
- CelebA-HQ: $\approx 30k$ images with 1024^2 px
- LSUN (Bedroom/Cat/Churches/...): $\approx 100k$ or $1M$ images with 256^2 px
- FFHQ (or FFHQ-U): $70k$ images with 1024^2 px
- LAION-5B: $5.85B$ images of various size

Neural Network architecture

Generator and discriminator networks can have various layers:

- Fully connected (FC) layers
- Upsampling (interpolation) or Subsampling (max/average pooling) layers
- Convolution/Transposed convolution (with stride)
- Activation functions: RELU, leakyRELU, sigmoid, tanh, etc
- BatchNorm
- ...

Input noise Z has often uniform distribution $\mathcal{U}([0,1]^p)$ or Gaussian distribution $\mathcal{N}(0,\mathrm{Id})$.

Convolution

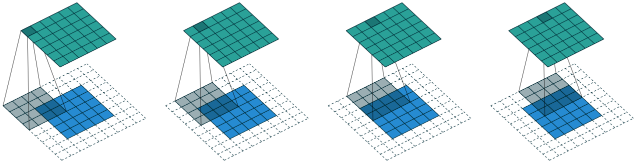
Let $u : \Omega \to \mathbf{R}^C$ be defined on $\Omega = [0 : M-1] \times [0 : N-1]$.

Let $w : \omega \to \mathbf{R}^{C' \times C}$ be defined on a small $\omega \subset \mathbb{Z}^2$. (Often, $\omega = [-k, k]^2$)

Definition

The convolution $w * u$ of the image u with kernel w is defined by

$$w * u(x) = \sum_{y \in \omega} w(y) u(x-y) = \sum_{z \in -\omega} \tilde{w}(z) u(x+z) \quad \text{where} \quad \tilde{w}(z) = w(-z).$$



NB: There are several possible border conditions (restriction, constant padding, periodic, ...)

Convolution and Transposed convolution

Notice that

- The transpose of a convolution with a $k \times k$ kernel is a convolution with a $k \times k$ kernel
- The transpose of a border crop is zero-padding the borders.
- The transpose of a crude subsampling is zero-inserting.

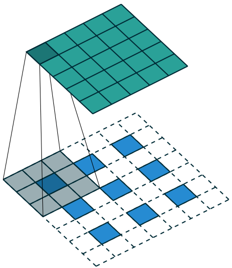
Strided convolutions:

- A “convolution with stride” is a convolution followed by subsampling.
- Called `Conv2d` in PyTorch

Fractionally strided convolutions:

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

One Example from [Dumoulin and Visin, 2016]



"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 $p' = 1$)."

See also `https://madebyollin.github.io/convnet-calculator/`

BatchNorm layer

Principle of BatchNormalization:

- Consider a batch $(x_n)_{1\leq n\leq N}$ of N responses to a neural layer with C features.
- For each n, $x_{n,i} \in \mathbf{R}^{W\times H}$ is the i-th feature map of the n-th image.
- Batch normalization consists in computing for any n, i

$$y_{n,i} = \gamma_i z_{n,i} + \beta_i \quad \text{with} \quad z_{n,i} = \frac{x_{n,i} - m_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$

  where m_i, σ_i are the mean and std of the gathered feature maps $(x_{n,i})_{1\leq n\leq N}$.
  (In other words, m_i, σ_i contains averages over N and spatial dimensions H, W.)
- γ_i, β_i are trainable parameters.
- Implemented in `BatchNorm2d` in PyTorch.

At inference: normalization is done with $m_i, \sigma_i, \gamma_i, \beta_i$ learned during training.
Switch to inference mode with `model.eval()`.

Different Kinds of Normalization

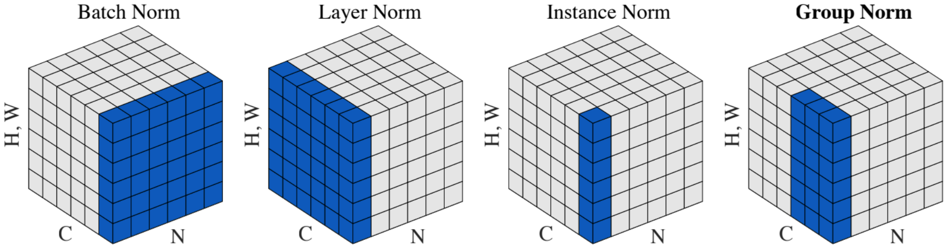


Diagram from [Wu and He, 2018]

- H, W: spatial dimensions
- C: channel dimension
- N: batch dimension

(See the formula for InstanceNorm in [Ulyanov et al., 2017])

Convolutional GAN

[Radford et al., 2016]

Important principles of the construction:

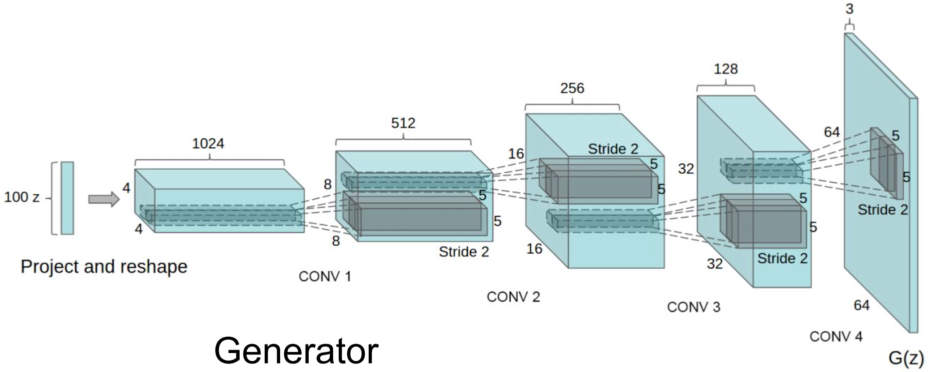
- “All convolutional”: remove max pooling layers, and learn downsampling instead
- Eliminate Fully-Connected Layers
- Batch Normalization to stabilize learning (except on generator output, and discriminator input)
- ReLU activations for the generator
- LeakyReLU activations for the discriminator

Generator: upsampling network with **fractionally strided convolutions**

Discriminator: convolutional network with **strided convolutions**

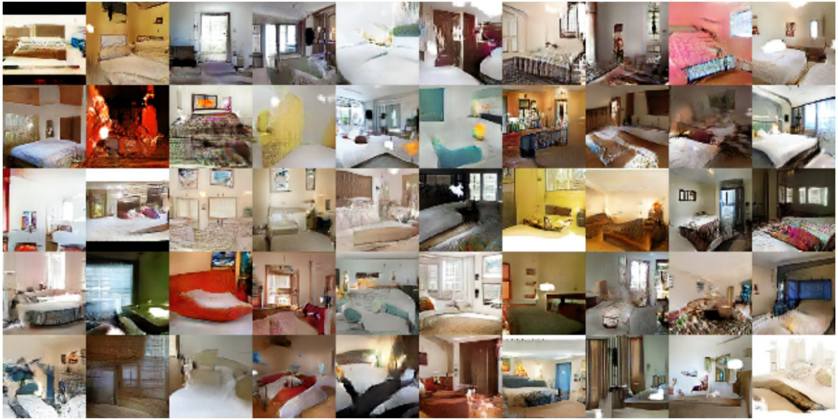
DCGAN Architecture

[Radford et al., 2016]



Generator

Image Generation with DCGAN [Radford et al., 2016]

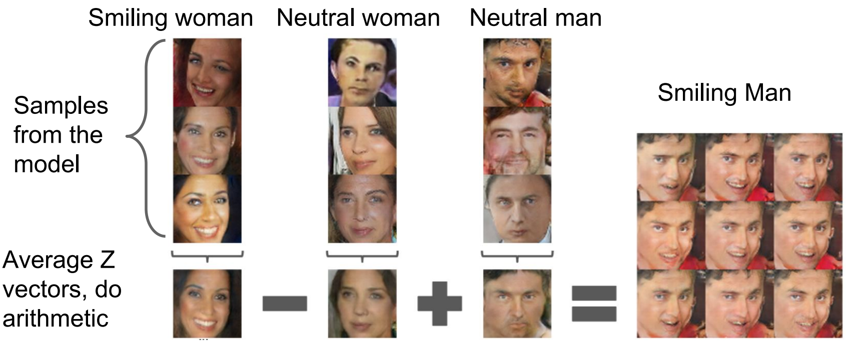


Generations of realistic bedrooms pictures, from randomly generated latent variables.

Image Interpolation with DCGAN [Radford et al., 2016]

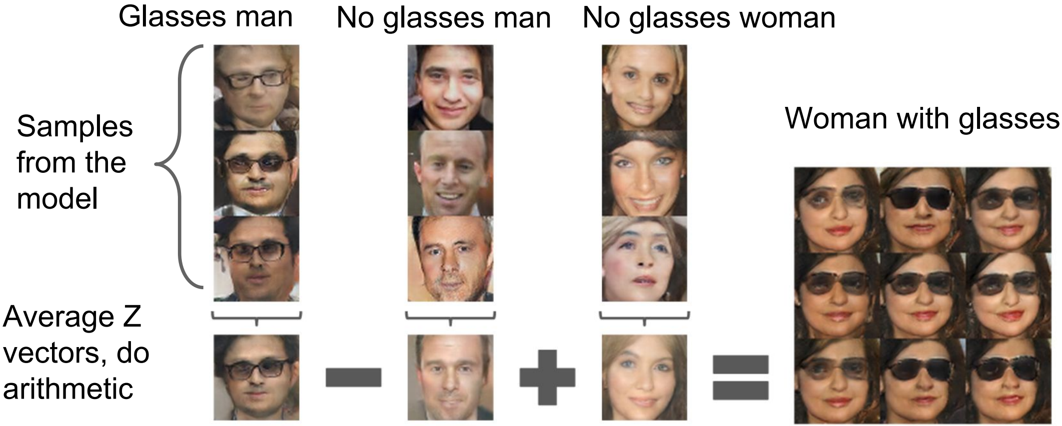
Interpolation in between points in latent space.

Arithmetic with DCGAN [Radford et al., 2016]



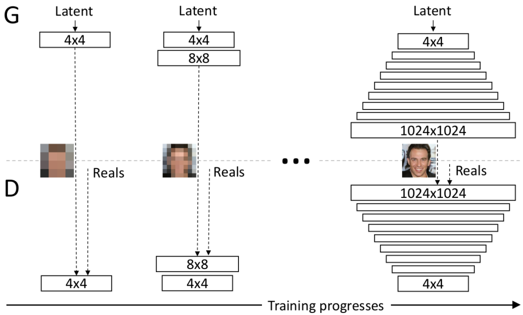
- Average latent vector of several samples
- After arithmetic, add a small random perturbation to get similar samples

Arithmetic with DCGAN [Radford et al., 2016]



Progressive Growing of GANs [Karras et al., 2018]

- Progressive Multiresolution Training
- Mirror architectures for *G* and *D*
- Simple upsampling/downsampling
  nearest neighbor upsampling;
  average pooling downsampling
  - Minibatch statistics layer at the end of *D*
  - Pixelwise feature normalization
- Training with WGAN-GP



StyleGAN [Karras et al., 2019]

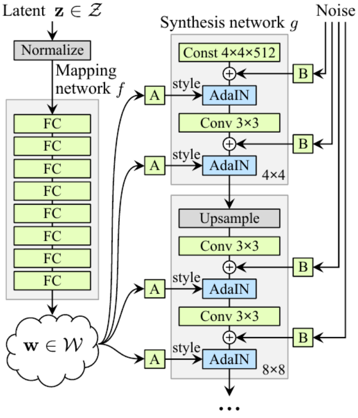
- "Separation of high-level features (pose, identity) from stochastic variation (freckles, hair)"
- Embed latent code z into an intermediate latent space w with a multilayer perceptron (8 FC layers)
- Spatially invariant style vector $y = (y_s, y_b)$ for each feature map, obtained from w
- AdaIN: Adaptive Instance Normalization

$$\text{AdaIN}(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

where the feature map x_i is normalized separately. (No learned parameters γ, β here.)

$$\text{AdaIN}(x_{n,i}, y) = y_{s,i}\frac{x_{n,i} - \mu(x_{n,i})}{\sigma(x_{n,i})} + y_{b,i}$$

- Style mixing (playing with two latent codes w_1, w_2)



(b) Style-based generator

StyleGAN [Karras et al., 2019]

- “Separation of high-level features (pose, identity) from stochastic variation (freckles, hair)”
- Embed latent code z into an intermediate latent space w with a multilayer perceptron (8 FC layers)
- Spatially invariant style vector $y = (y_s, y_b)$ for each feature map, obtained from w
- AdaIN: Adaptive Instance Normalization

$$\mathrm{AdaIN}(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

where the feature map x_i is normalized separately. (No learned parameters γ, β here.)

$$\mathrm{AdaIN}(x_{n,i}, y) = y_{s,i} \frac{x_{n,i} - \mu(x_{n,i})}{\sigma(x_{n,i})} + y_{b,i}$$

- Style mixing (playing with two latent codes w_1, w_2)

StyleGAN [Karras et al., 2019]

StyleGAN allows for style mixing at different scales (by using the corresponding subparts of w).

StyleGAN2 [Karras et al., 2020]

- AdaIN causes droplet artifacts in StyleGAN
  $\rightarrow$ Weight modulation/demodulation instead of AdaIN
- Path length regularization: fixed-norm steps in w results in fixed-norm changes in image space
- Residual connections with downsampling in D
- Skip connections in G
- No progressive growing (which leads to *phase artifacts*)

Face Generation with StyleGAN2 [Karras et al., 2020]

https://thispersondoesnotexist.com/

Face Generation with StyleGAN2 [Karras et al., 2020]

https://thispersondoesnotexist.com/

Face Generation with StyleGAN2 [Karras et al., 2020]

https://thispersondoesnotexist.com/

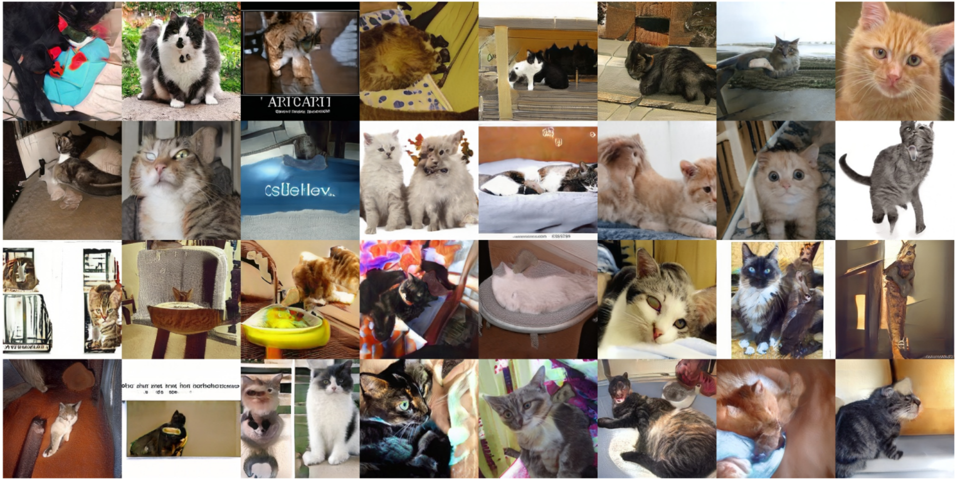
StyleGAN vs StyleGAN2

First row: real images
Second row: samples of StyleGAN after projection on the latent code

StyleGAN vs StyleGAN2

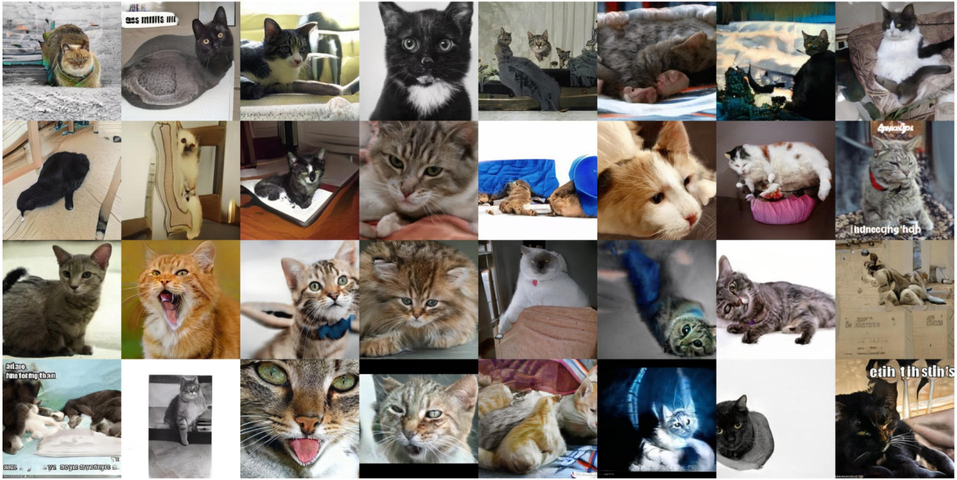
First row: real images
Second row: samples of StyleGAN2 after projection on the latent code

The Cat Challenge...



Samples of StyleGAN2-Model1 trained on LSUN Cat

The Cat Challenge...

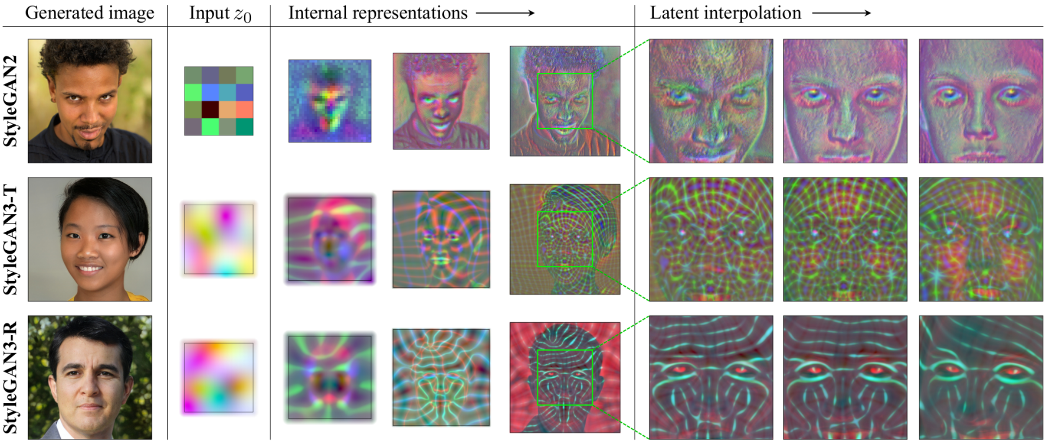


Samples of StyleGAN2-Model2 trained on LSUN Cat

StyleGAN3 aka Alias-free GANs

- Aliasing artifacts present in some GANs results due to:
  · non-ideal upsampling
  · pointwise activations
- Enforce continuous equivariance to sub-pixel translation (Shannon is back...)
- Also, ensure that no aliasing appears through the network:
  · use band-limited filters
  · use low-pass filters when needed

StyleGAN3 aka Alias-free GANs



Conditional GANs

Conditional GANs: Train the generator and the discriminator by passing a class information:

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

Class conditional training:

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

where

- $\mathcal{D}_{\text{real}}$ is a collection of real labeled data.
- $\mathcal{D}_{\text{rand}}$ is a collection of synthetic latent code and labels.

This requires to choose a distribution on c to generate the synthetic image $G_{\theta_G}(z, c)$.

Plan

Large-Scale GAN Training

Quality Metrics for Generative Models

Generative Texture Models

Quality of a Generative Model

- **Question:** How to measure that the generator covers well the training data?
- **Main idea:** Comparing image distributions is hard...
  but comparing measurements from it is easier.
- Classification neural networks provide a set of deep non-linear features.
  For example, VGG19 [Simonyan and Zisserman, 2015], or Inception Networks [Szegedy et al., 2016].
- **Measure quality of the generative model by looking at how deep statistics are preserved**
  Somehow, this ensures that the database is well-covered.
- Keep in mind that
  · The network used to get the features must be relevant w.r.t. the generative task at play.
  · Quantitative results highly depend on the network and implementation details.

Inception Score ↑ [Salimans et al., 2016]

- The inception score measures if μ generates a diverse collection of meaningful pictures
- For an image x, `Inception-v3` gives a label distribution $p(y|x)$ (discrete on $N = 1000$ labels)
- Images containing meaningful objects have $p(y|x)$ with low entropy
- In order to generate various images, $p(y) = \int p(y|x)\mu(dx)$ should have high entropy

The Inception Score then writes as

$$\mathrm{IS}(\mu) = \exp\left(\int \mathrm{KL}\left(p(y|x)|p(y)\right)\mu(dx)\right) \in [1, N]$$

It is 1 iff for a.e. x, $p(\cdot|x) = p(\cdot)$ (label distribution does not depend on x)
It is N iff for a.e. x, $p(\cdot|x)$ is concentrated on one label, and $\forall y$, $\int p(y|x)\mu(dx) = \frac{1}{N}$

How to compute it in practice:

- Compute an estimate $\hat{p}(y)$ of $p(y) = \int p(y|x)\mu(dx)$ by drawing samples of μ
- Estimate $\int \mathrm{KL}(p(y|x)|\hat{p}(y))\mu(dx)$ by drawing samples of μ

Fréchet Inception Distance (FID) $\downarrow$ [Heusel et al., 2017]

The FID measures how close are two image distributions μ, ν in terms of features distributions.
It is based on the response of `Inception-v3` [Szegedy et al., 2016] before last pooling layer:

$$f : \mathbf{R}^d \to \mathbf{R}^m$$

that extracts $m = 2048$ features (as a generic image summary)
NB: Images may have to be resized/normalized to fit into this network.

Algorithm to compute the FID score:

1. Draw samples (x_i) and (y_j) of $X \sim \mu$ and $Y \sim \nu$ and compute the features $(f(x_i))$, $(f(y_j))$
2. Fit Gaussian distributions $\mathcal{N}(m_X, \Sigma_X)$ and $\mathcal{N}(m_Y, \Sigma_Y)$ to $(f(x_i)), (f(y_j))$ (in $\mathbf{R}^{2048}$)
3. Return the 2-Wasserstein distance between the Gaussian distributions, i.e. the Fréchet distance: [Dowson and Landau, 1982]

$$W_2^2\Big(\mathcal{N}(m_X, \Sigma_X), \mathcal{N}(m_Y, \Sigma_Y)\Big) = \|m_X - m_Y\|_2^2 + \mathrm{Tr}\Big(\Sigma_X + \Sigma_Y - 2(\Sigma_X \Sigma_Y)^{\frac{1}{2}}\Big)$$

NB: FID can be adapted to the “single-image” case: **SiFID** [Shaham et al., 2019]
SiFID compares distributions of features obtained after a convolution layer (spatially averaged)

Comments on Generative Quality

- Inception Score does not depend on the target distribution ν.
- Need to distinguish "precision/recall" for evaluating quality [Lucic et al., 2018].
  "Precision" is the probability that a fake image falls within the distribution of real images.
  "Recall" is the probability that a real image falls within the distribution of fake sample.
  IS mainly captures precision. FID captures both precision and recall.
- The IS and FID are not enough to measure the fact that samples are photo-realistic.
  [Barratt and Sharma, 2018]
- Other measures have been proposed better correlated with Human prediction of quality.
  [Kolchinski et al., 2019]

Are GANs created equal?

[Lucic et al., 2018]

Many variants of GAN training exist, with various architectures and more or less stable training.

- Regarding quality of generated images, may GAN variants perform similarly.
  [Lucic et al., 2018] proposed a large comparison framework, with a budget for hyperparameter tuning, and by averaging over several random seeds.
- “WGANs work because they fail” [Stanczuk et al., 2021], [Mallasto et al., 2019]
  The dual training in WGAN-GP does not approximate the Wasserstein distance correctly.
  But estimating it more precisely (e.g. semi-discrete WGAN) often leads to blurrier samples.
  $\rightarrow$ The quality of a generative network relies on good features learned by the discriminator.

Plan

Exemplar-based Texture Synthesis

- Examplar texture :

$$u_0 : \Omega \to \mathbf{R}^d$$

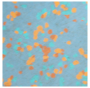
defined on a discrete rectangle $\Omega \subset \mathbb{Z}^2$.

- Texture model: stationary random field

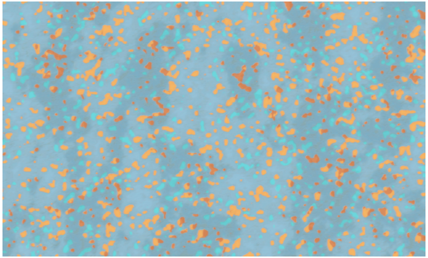
$$V : \mathbb{Z}^2 \to \mathbf{R}^d$$

The problem can be split into

- Estimate a model V
- Draw one (or several) samples of V



Exemplar u_0



Synthesis v

What do we want to preserve ?

- Covariance, Fourier spectrum
  [Lewis, 1984], [Van Wijk, 1991], [Galerne et al., 2011], [Gilet et al., 2014]
- Wavelet statistics
  [Heeger & Bergen, 1995], [Zhu et al., 1998], [Portilla & Simoncelli, 2000], [Tartavel et al., 2014], [Zhang & Mallat, 2017], [Bruna & Mallat, 2019]
- Local Aspect, Patch statistics
  [Efros & Leung, 1999], [Kwatra et al., 2005], [Lefebvre & Hoppe, 2005]
- Neural Statistics
  [Gatys et al., 2015], [Lu et al., 2015], [Ulyanov et al., 2016]

What do we want to preserve ?

- Covariance, Fourier spectrum
  [Lewis, 1984], [Van Wijk, 1991], [Galerne et al., 2011], [Gilet et al., 2014]
- Wavelet statistics
  [Heeger & Bergen, 1995], [Zhu et al., 1998], [Portilla & Simoncelli, 2000], [Tartavel et al., 2014], [Zhang & Mallat, 2017], [Bruna & Mallat, 2019]
- **Local Aspect, Patch statistics**
  [Efros & Leung, 1999], [Kwatra et al., 2005], [Lefebvre & Hoppe, 2005]
- **Neural Statistics**
  [Gatys et al., 2015], [Lu et al., 2015], [Ulyanov et al., 2016]

Texture Synthesis with Patch Optimal Transport

[Galerne et al., 2018]

QUESTION : **How to prescribe the patch distribution at several resolutions ?**

PRINCIPLE OF THE "TEXTO" MODEL:

- Initialize with a Gaussian field at coarse resolution
- At each resolution, apply a patch transport map to reimpose the exemplar patch distribution ν_s
- Upsample cleverly to go from one scale to the next

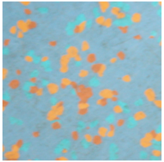


Image u^0
Patch distrib ν^0

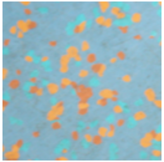


Image u^1
Patch distrib ν^1

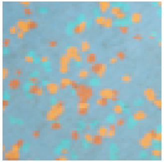


Image u^2
Patch distrib ν^2

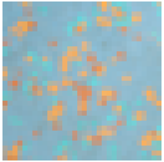


Image u^3
Patch distrib ν^3

The Texto Model

Compute exemplar $u^s : \Omega \cap (2^s\mathbb{Z}^2) \to \mathbf{R}^d$ at different scales $s = 0, \dots, S-1$
and corresponding patch distributions ν^s

Initialize synthesis with Gaussian field U_{S-1} at the coarse scale

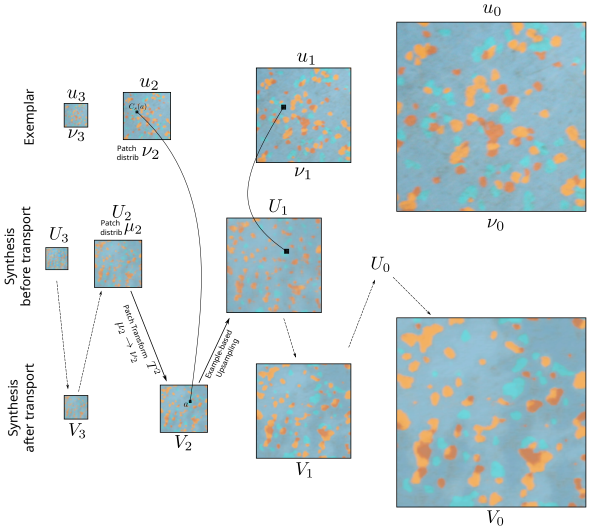
For $s = S-1, \dots, 0$,

- Estimate the patch distribution μ_s of U_s
- Learn a patch semi-discrete OT map T_s such that $T_s \sharp \mu_s \approx \nu_s$
  (Recall that T_s is a biased nearest neighbor assignment!)
- Apply T_s to all patches of U_s and recompose by averaging to an image V_s
- If $s > 0$, upsample V_s to initialize the next scale U_{s-1}
  (For that, use patches at the same positions, but twice larger.)

Output: synthesis at fine scale V_0

Remark: Once the model learnt, one can discard the learning steps ■ to do synthesis on-the-fly

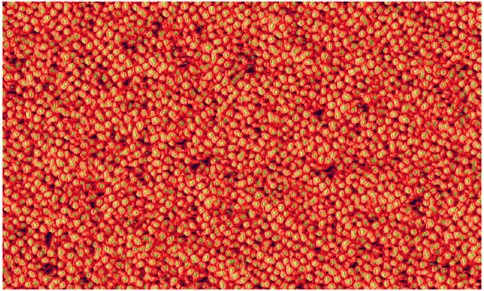
Texto in one diagram



Texto Results

- Long-range independence property
- Patches are transformed independently
  $\rightarrow$ allows for parallel computations
- Patch OT maps can be computed offline.
  $\rightarrow$ allows for very fast synthesis
- Synthesis slightly blurry
  due to patch averaging

Original
128×128



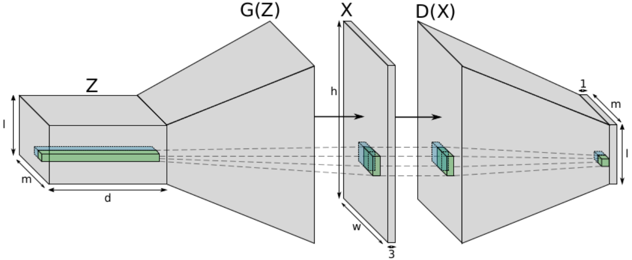
Synthesis
1280×768 (4 scales, 1s)

Spatial GANs [Jetchev et al., 2016], [Bergmann et al., 2017]

- Symmetric Convolutional Networks for G and D (as DCGAN, see later)
- From a $l \times m$ noise Z, $g_\theta(Z)$ generates a $h \times w$ image (in practice $l = m = 4$ and $h = w = 640$)
- Standard GAN loss (binary cross-entropy) but averaged over spatial positions (λ, μ):

$$\sum_{\lambda,\mu} \mathbb{E}[\log(1 - D_{\lambda,\mu}(g_\theta(Z)))] + \mathbb{E}[\log D_{\lambda,\mu}(Y')] \quad \text{where } Y' \text{ is a patch from } u_0$$

- PSGAN works on an augmented noise input Z, with local, global and periodic parts

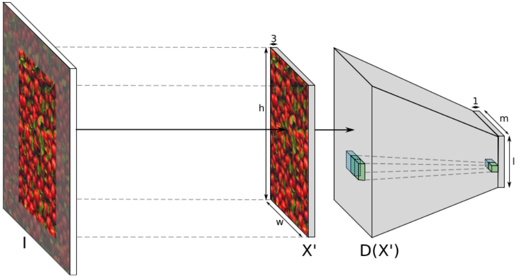


Spatial GANs [Jetchev et al., 2016], [Bergmann et al., 2017]

- Symmetric Convolutional Networks for G and D (as DCGAN, see later)
- From a $l \times m$ noise Z, $g_\theta(Z)$ generates a $h \times w$ image (in practice $l = m = 4$ and $h = w = 640$)
- Standard GAN loss (binary cross-entropy) but averaged over spatial positions (λ, μ):

$$\sum_{\lambda,\mu} \mathbb{E}[\log(1 - D_{\lambda,\mu}(g_\theta(Z)))] + \mathbb{E}[\log D_{\lambda,\mu}(Y')] \quad \text{where } Y' \text{ is a patch from } u_0$$

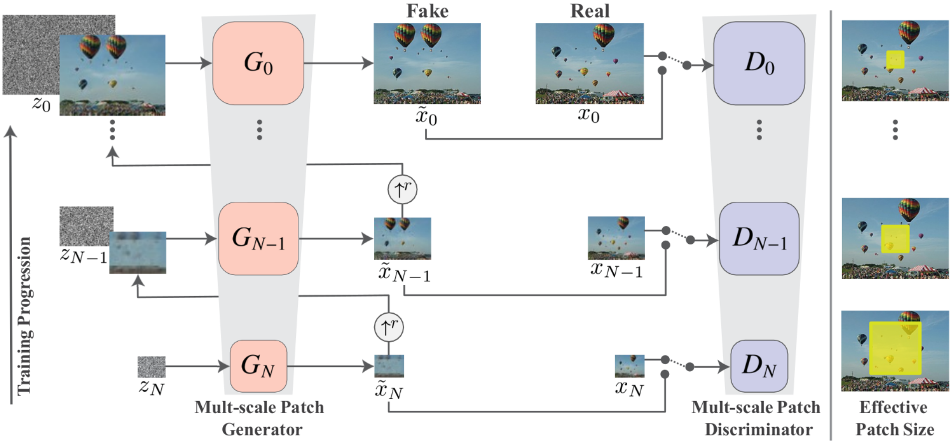
- PSGAN works on an augmented noise input Z, with local, global and periodic parts



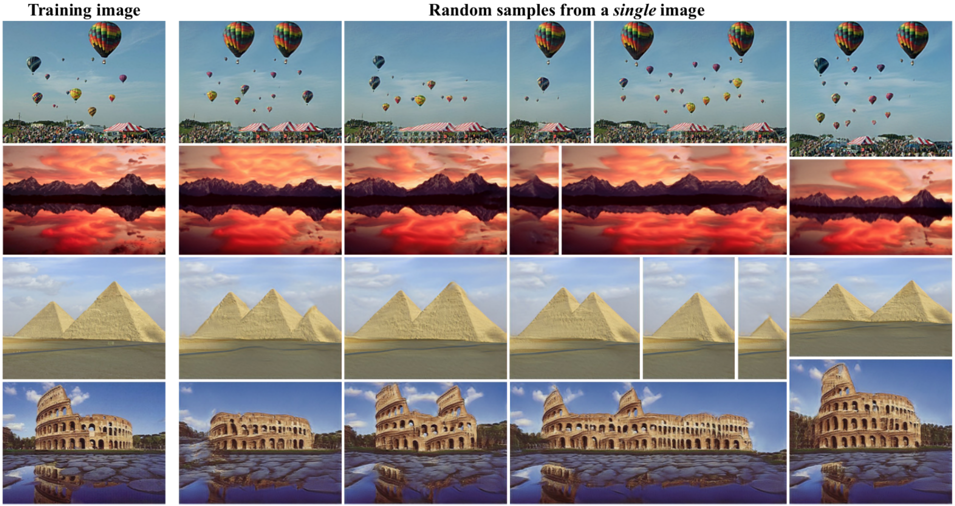
SinGAN: Learning from a Single Image [Shaham et al., 2019]

- Capture the multi-scale patch distributions of an image (possibly non-texture)
- Coarse-to-fine generator
- Patch-based dicriminator learned with WGAN-GP loss, at each scale
- Loss defined over all patches of the image, and not randomly selected patches
  $\rightarrow$ allows the network to learn boundary conditions

SinGAN: Learning from a Single Image [Shaham et al., 2019]



SinGAN: Learning from a Single Image [Shaham et al., 2019]



Generative Networks for Texture Synthesis [Houdard et al., 2023]

IDEA : Build a generative network g_θ that directly constrains features distributions where

$$\mathcal{F}_p(u) : \Omega \to \mathbf{R}^{d_p} \text{ extracts features fo type } p.$$

For each feature type p, let

- $\mu_{\theta p}$: distribution of features $\mathcal{F}_p(g_\theta(Z))$
- ν_p : empirical distribution of features $\mathcal{F}_p(u_0)$

Examples:

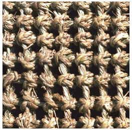
- $\mathcal{F}_p(u) : \Omega \to \mathbf{R}^{s_p \times s_p}$ extracts the $s_p \times s_p$ patches of u
- $\mathcal{F}_p(u) : \Omega_p \to \mathbf{R}^{d_p}$ extracts the response to layer p of a neural network (e.g. VGG)

Learning of GOTEX model

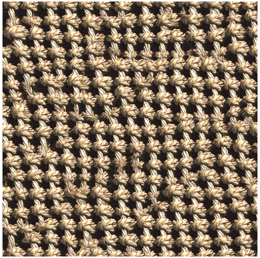
$$\inf_\theta \sum_p W(\mu_{\theta p}, \nu_p)$$

$\longrightarrow$ Alternate optimization with one dual variable ψ_p for each p

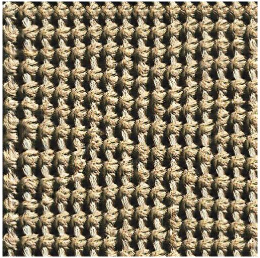
Samples of Texture Networks



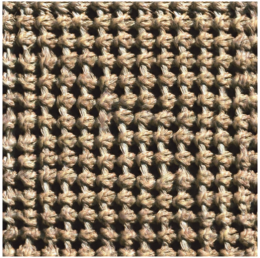
Original



GOTEX
[Houdard et al., 2023]

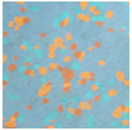


PSGAN
[Bergmann et al., 2017]

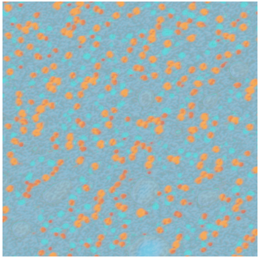


SinGAN
[Shaham et al., 2019]

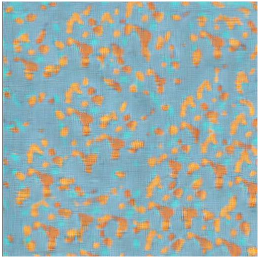
Samples of Texture Networks



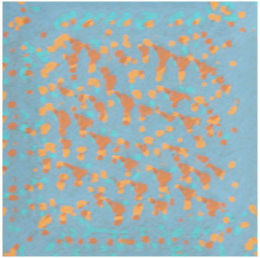
Original



GOTEX
[Houdard et al., 2023]



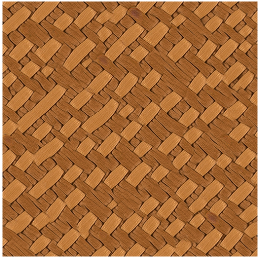
PSGAN
[Bergmann et al., 2017]



SinGAN
[Shaham et al., 2019]

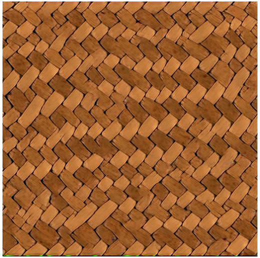
Samples of Texture Networks

Original



GOTEX
[Houdard et al., 2023]

PSGAN
[Bergmann et al., 2017]



SinGAN
[Shaham et al., 2019]

Take-home Messages

- We discussed several architectures for image generation.
- Large-scale synthesis benefits from architectures adapted for multi-resolution synthesis.
- Recent generative models crucially rely on
  · several tricks for training or designing the architecture
  · very long training of models...
  · with a very large number of parameters
  · and a very large dataset.
- FID score gives a reasonable/simple way to measure the quality of a generative model...
  but it does not suffice to judge photo-realism of the samples.

THANK YOU FOR YOUR ATTENTION!

References I

Barratt, S. and Sharma, R. (2018).
A note on the inception score.
arXiv preprint arXiv:1801.01973.

Bergmann, U., Jetchev, N., and Vollgraf, R. (2017).
Learning texture manifolds with the periodic spatial gan.
In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 469–477. JMLR. org.

Dowson, D. C. and Landau, B. V. (1982).
The fréchet distance between multivariate normal distributions.
Journal of Multivariate Analysis, 12(3):450–455.

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

Galerne, B., Leclaire, A., and Rabin, J. (2018).
A texture synthesis model based on semi-discrete optimal transport in patch space.
SIAM Journal on Imaging Sciences, 11(4):2456–2493.

References II

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017).
Gans trained by a two time-scale update rule converge to a local Nash equilibrium.
In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Houdard, A., Leclaire, A., Papadakis, N., and Rabin, J. (2023).
A generative model for texture synthesis based on optimal transport between feature distributions.
Journal of Mathematical Imaging and Vision, 65(1):4–28.

Jetchev, N., Bergmann, U., and Vollgraf, R. (2016).
Texture synthesis with spatial generative adversarial networks.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018).
Progressive Growing of GANs for Improved Quality, Stability, and Variation.
In *Proceedings of International Conference on Learning Representations*.

References III

Karras, T., Laine, S., and Aila, T. (2019).
A style-based generator architecture for generative adversarial networks.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020).
Analyzing and improving the image quality of StyleGAN.
In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119.

Kolchinski, Y. A., Zhou, S., Zhao, S., Gordon, M. L., and Ermon, S. (2019).
Approximating human judgment of generated image quality.
CoRR, abs/1912.12121.

Lucic, M., Kurach, K., Michalski, M., Gelly, S., and Bousquet, O. (2018).
Are GANs created equal? A large-scale study.
In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

References IV

Mallasto, A., Montúfar, G., and Gerolin, A. (2019).
How well do WGANs estimate the Wasserstein metric?
arXiv preprint arXiv:1910.03875.

Radford, A., Metz, L., and Chintala, S. (2016).
Unsupervised representation learning with deep convolutional generative adversarial networks.
In Bengio, Y. and LeCun, Y., editors, *Proceedings of ICLR*.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. (2016).
Improved techniques for training gans.
Advances in neural information processing systems, 29.

Shaham, T. R., Dekel, T., and Michaeli, T. (2019).
SinGAN: Learning a Generative Model from a Single Natural Image.
In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580.

Simonyan, K. and Zisserman, A. (2015).
Very deep convolutional networks for large-scale image recognition.
In Bengio, Y. and LeCun, Y., editors, *Proceedings of the International Conference on Learning Representations*.

References V

Stanczuk, J., Etmann, C., Kreusser, L. M., and Schönlieb, C.-B. (2021).
Wasserstein GANs work because they fail (to approximate the Wasserstein distance).
arXiv preprint arXiv:2103.01678.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016).
Rethinking the inception architecture for computer vision.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2017).
Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6924–6932.

Wu, Y. and He, K. (2018).
Group normalization.
In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.