Visual AI

CPSC 533R

Lecture 6. GANs and Unpaired Image Translation

Helge Rhodin

Assignment 3

- Rendering
- Learning shape spaces
- Interpolating in shape spaces

- Work with your teammate only, don't cheat!
  - disciplinary measures will be reported on your transcripts
    - your future applications may be rejected because of this

Assignment 3: Neural Rendering and Shape Processing

CPSC 532R/533R Visual AI
by Helge Rhodin and Yuchi Zhang

This assignment is on neural rendering and shape processing—computer graphics. We provide you with a dataset of 2D icons and corresponding vector graphics as shown in Figure 1. It stems from a line of work on translating low-resolution icons to visually appealing vector forms and was kindly provided by Sheffer et al. [1] for the purpose of this assignment.

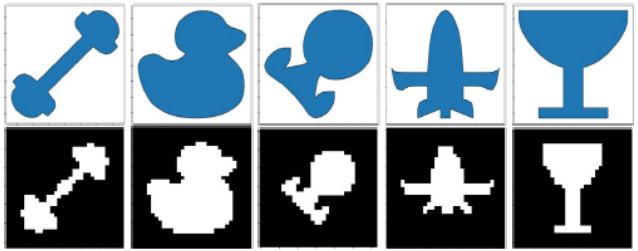


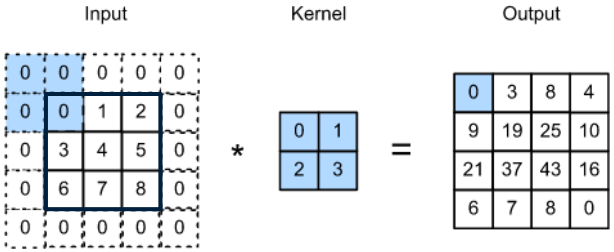
Figure 1: Icon vector graphics and their bitmap representation.

The overall goal of this assignment is to find transformation between icons. We provide the `ImagerIcon` dataset as an HDF5 file. As usual, the `Assignment3_TaskI.ipynb` notebook provides dataloading, training and validation splits, as well as display and training functionality. Compatibility of the developed neural networks with color images is ensured by storing the contained 32 x 32 icon bitmaps as $3 \times W \times H$ tensors. Vector graphics are represented as polygons with $N = 96$ vertices and are stored as $2 \times N$ tensors, with neighboring points stored sequentially. The polygon representation with a fixed number of vertices was attained by subsampling the originally curved vector graphics.

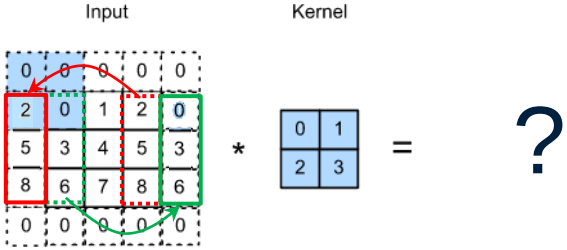
Assignment 3: Convolution with cyclic padding, theory

Convolution

- instead of inserting zeroes, copy values from the opposing side of the image



zero padding



cyclic padding

Cyclic padding, broken in early PyTorch versions

Old, broken versions:

- Setting padding equal to zero or one seems to have the same effect
  - in both cases the output resolution is reduced, as without padding
- Experienced instability when using cyclic padding
- Only tested with 1D convolutions

- Use the newest version!

Auto Encoder (AE)

General case

$$\mathbf{h} = \text{encoder}_\theta(\mathbf{x})$$
$$\mathbf{x}' = \text{decoder}_\theta(\mathbf{h})$$

Simple non-linear case

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{x}' = \sigma(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

Linear case

$$\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\mathbf{x}' = \mathbf{W}'\mathbf{h} + \mathbf{b}'$$

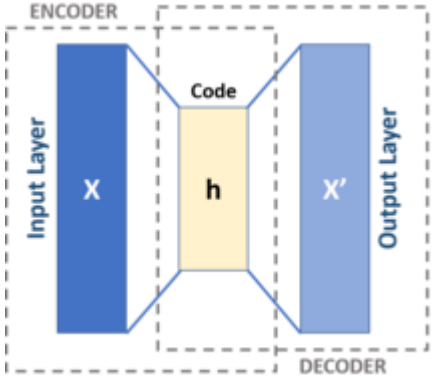
General reconstruction objective

$$\text{loss}(\mathbf{x}, \mathbf{x}')$$

- e.g., MSE loss

A two-layer fully-connected neural network

Similar to PCA when using squared loss (W neither forms an ordered nor orthogonal basis)



https://en.wikipedia.org/wiki/Autoencoder

Linear autoencoder objective

$$\arg\min_{\mathbf{W}} \sum_i \|\mathbf{x} - \mathbf{x}'\|^2$$
$$= \arg\min_{\mathbf{W}} \sum_i \|\mathbf{x}_{(i)} - \mathbf{W}'\mathbf{W}\mathbf{x}_{(i)}\|^2$$

PCA objective

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w}\right)^2 \right\}$$
$$= \arg\max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^T \mathbf{X}^\mathbf{T} \mathbf{X} \mathbf{w} \right\}$$

[From Principal Subspaces to Principal Components with Linear Autoencoders]

Autoencoder variants

Bottleneck autoencoder:

- hidden dimension smaller than input dimension
  - leads to compressed representations
    - like dimensionality reduction with PCA

Sparse autoencoder:

- hidden dimension larger than input dimension
- hidden activation enforced to be sparse
  (= few activations non-zero)

Denoising autoencoder:

- corrupt the input values, e.g. by additive noise

$$\mathbf{h} = \text{encoder}_\theta(\text{noise}(\mathbf{x}))$$
$$\mathbf{x}' = \text{decoder}_\theta(\mathbf{h})$$

Variational Auto Encoder (VAE)

- a probabilistic model
  - *'adding noise on the hidden variables'*
  - more in lecture 8!

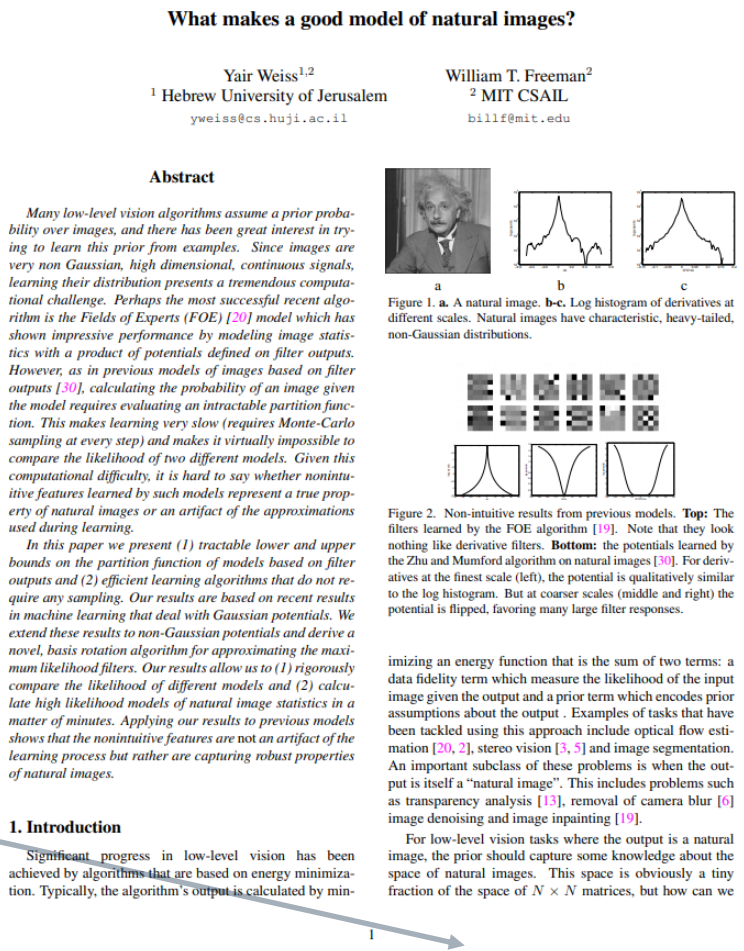
Generative Adversarial Networks (GAN)

What is a natural image?

- a collection of pixels
- natural textures
- local structure
- spatial consistency
- temporal consistency (video)

How can we model all that?

The year I started my BSc



What makes a good model of natural images?

Yair Weiss[1,2]
[1] Hebrew University of Jerusalem
yweiss@cs.huji.ac.il

William T. Freeman[2]
[2] MIT CSAIL
billf@mit.edu

Abstract

Many low-level vision algorithms assume a prior probability over images, and there has been great interest in trying to learn this prior from examples. Since images are very non Gaussian, high dimensional, continuous signals, learning their distribution presents a tremendous computational challenge. Perhaps the most successful recent algorithm is the Fields of Experts (FOE) [20] model which has shown impressive performance by modeling image statistics with a product of potentials defined on filter outputs. However, as in previous models of images based on filter outputs [30], calculating the probability of an image given the model requires evaluating an intractable partition function. This makes learning very slow (requires Monte-Carlo sampling at every step) and makes it virtually impossible to compare the likelihood of two different models. Given this computational difficulty, it is hard to say whether nonintuitive features learned by such models represent a true property of natural images or an artifact of the approximations used during learning.

In this paper we present (1) tractable lower and upper bounds on the partition function of models based on filter outputs and (2) efficient learning algorithms that do not require any sampling. Our results are based on recent results in machine learning that deal with Gaussian potentials. We extend these results to non-Gaussian potentials and derive a novel, basis rotation algorithm for approximating the maximum likelihood filters. Our results allow us to (1) rigorously compare the likelihood of different models and (2) calculate high likelihood models of natural image statistics in a matter of minutes. Applying our results to previous models shows that the nonintuitive features are not an artifact of the learning process but rather are capturing robust properties of natural images.

1. Introduction

Significant progress in low-level vision has been achieved by algorithms that are based on energy minimization. Typically, the algorithm's output is calculated by minimizing an energy function that is the sum of two terms: a data fidelity term which measure the likelihood of the input image given the output and a prior term which encodes prior assumptions about the output . Examples of tasks that have been tackled using this approach include optical flow estimation [20, 2], stereo vision [3, 5] and image segmentation. An important subclass of these problems is when the output is itself a "natural image". This includes problems such as transparency analysis [13], removal of camera blur [6] image denoising and image inpainting [19].

For low-level vision tasks where the output is a natural image, the prior should capture some knowledge about the space of natural images. This space is obviously a tiny fraction of the space of $N \times N$ matrices, but how can we

Figure 1. **a.** A natural image. **b-c.** Log histogram of derivatives at different scales. Natural images have characteristic, heavy-tailed, non-Gaussian distributions.

Figure 2. Non-intuitive results from previous models. **Top:** The filters learned by the FOE algorithm [19]. Note that they look nothing like derivative filters. **Bottom:** the potentials learned by the Zhu and Mumford algorithm on natural images [30]. For derivatives at the finest scale (left), the potential is qualitatively similar to the log histogram. But at coarser scales (middle and right) the potential is flipped, favoring many large filter responses.

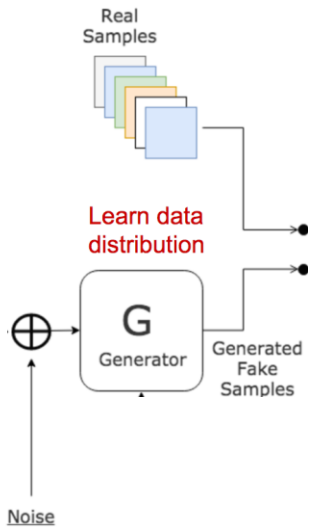
1

[Weiss et al., CVPR 2007]

GAN concept

Goal: Train a generator, G, that produces naturally looking images

Idea: Train a discriminator, D, that distinguishes between real and fake images. Use this generator to train G

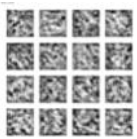


https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html

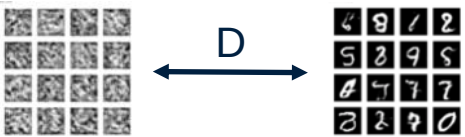
Dissecting GANs

Trying to hand-craft a GAN:

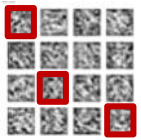
1. Start with a set of random images



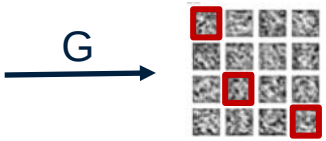
2. Train a discriminator D to classify real/fake images



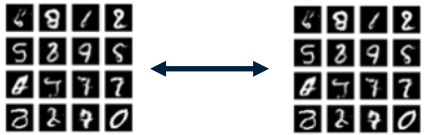
3. Select those fake examples that D is most unsure about



4. Train a generator G that creates more of the good fakes from noise



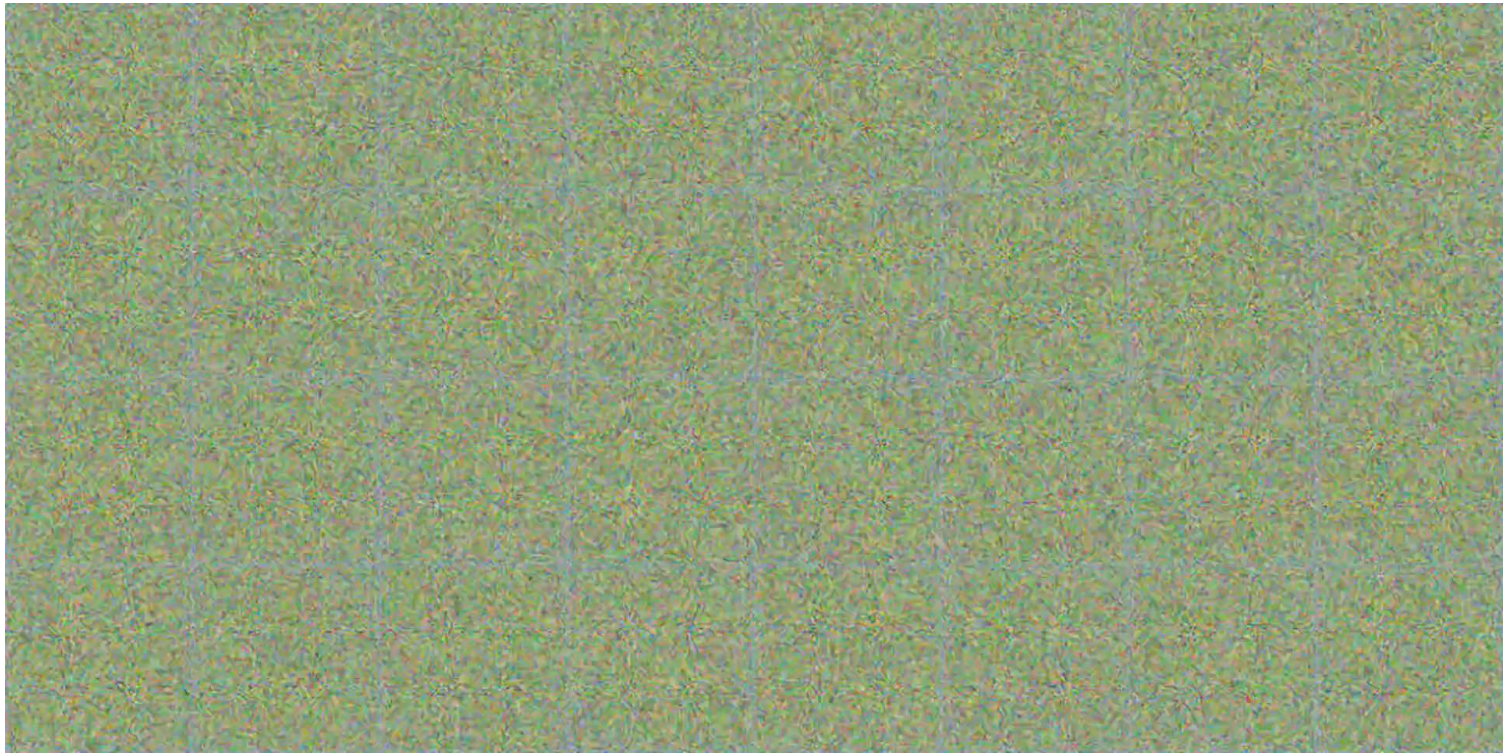
5. Generate new fake images with G. Continue with 2. until fakes are indistinguishable.



This is not how it works, just a mental image!

GAN training examples

Training output, showing training iterations over time



https://www.youtube.com/watch?v=4kY8UlzZEkM

GAN training examples

- Starts with noise
- Loss is not decreasing
  - hard to interpret
- Generator and discriminator learn together
- Hard to maintain balance!

[https://towardsdatascience.com/graduating-in-gans-going-from-understanding-generative-adversarial-networks-to-running-your-own-39804c283399]

GANs

A min max game (related to game theory)

Take the average over dataset/minibatch

Random noise

$$\min_G \max_D V(D,G) = \min_G \max_D [E_{x\sim p_r}[\log D(x)] + E_{z\sim p_z}[\log(1 - D(G(z)))]]$$

D should be high/max for real examples (from perspective of D, not influenced by G)

D should be **low** for fake examples (from perspective of D, maximize 1-D)

D should be **high** for fake examples (from perspective of G, minimize 1-D)

- Effects:
  - learning a loss function
  - learning the distribution of images
    - the generator can generate samples from the distribution of natural images

GAN training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

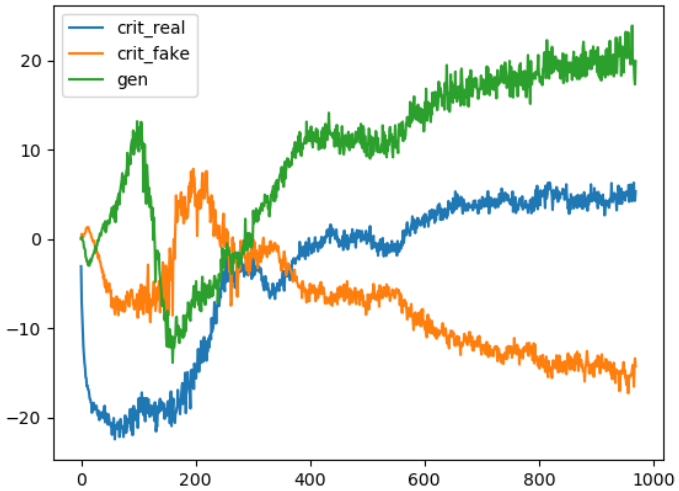
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.



Chaotic GAN loss behavior
(e.g., generator loss going up not down)

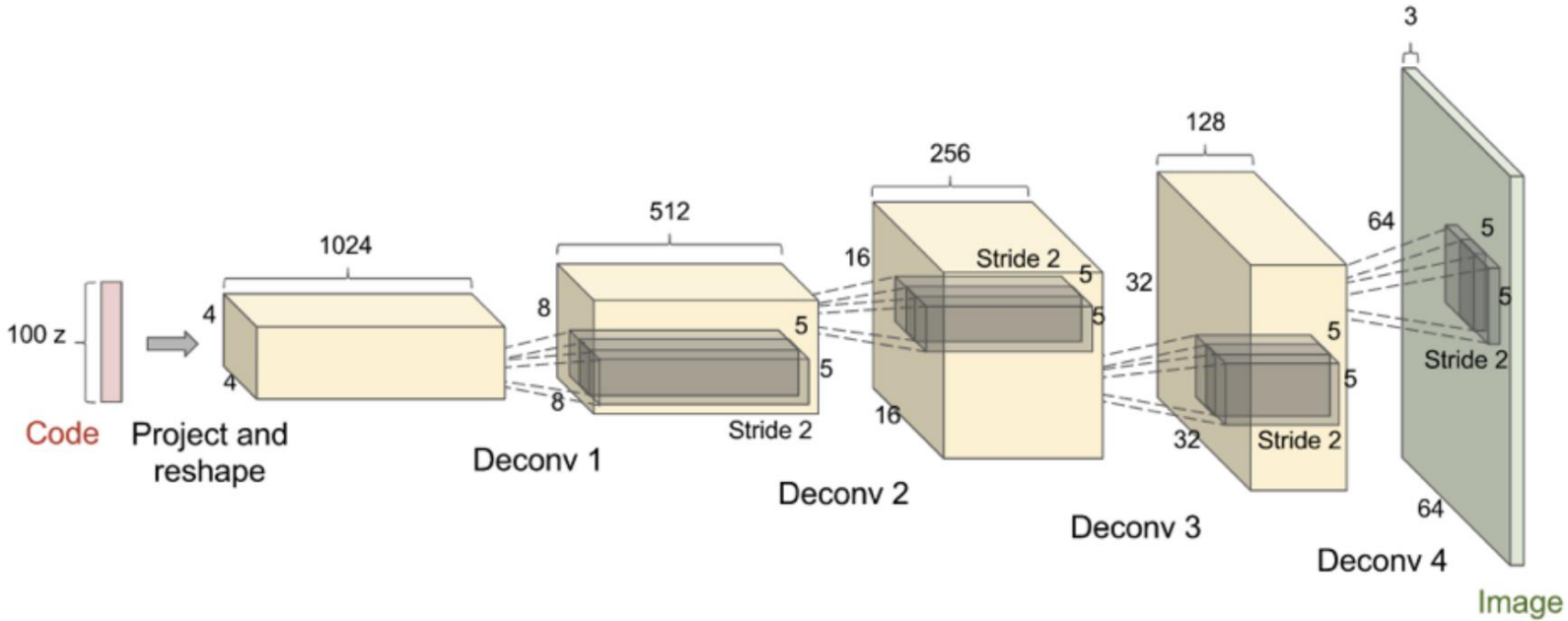
Green: outer loop on generator (gradient descent)

Orange: inner loop on discriminator (gradient ascent)

[Goodfellow et al., Generative Adversarial Networks. 2014]

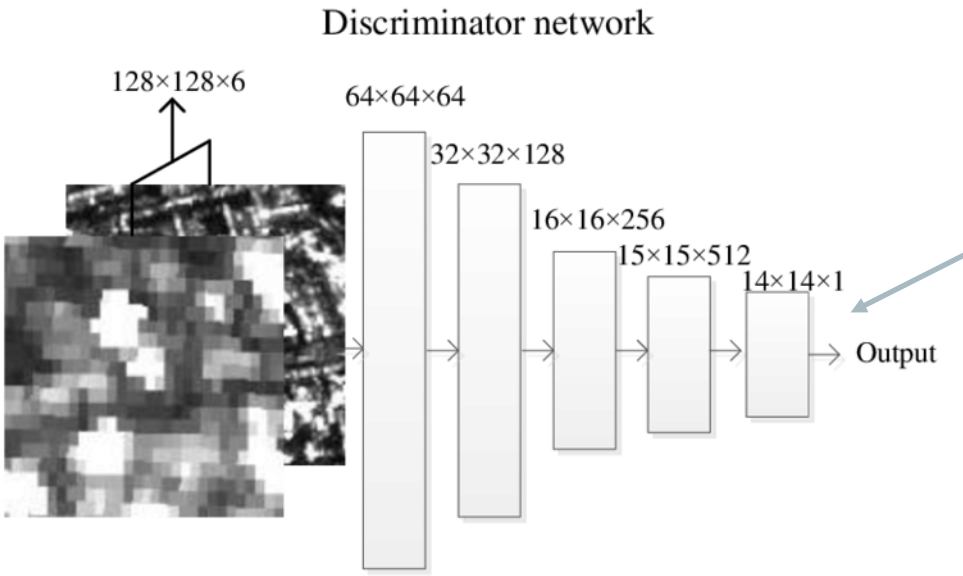
DCGAN

Convolutional generator architecture



PatchGAN

Patch-wise classification into real or fake (instead of globally)



A feature map of fake/real probabilities

[Li and Wandt, Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks]

GAN derivation (self-study)

Theorem: GAN training minimizes the JS divergence between real and generated/fake distributions

- The *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

$$\text{where} \quad \mathbb{P}_m = (\mathbb{P}_r + \mathbb{P}_g)/2$$

Proof: The GAN objective has the form

$$\min_G \max_D [E_{x \sim p_r}[\log D(x)] + E_{z \sim p_z}[\log(1 - D(G(z)))]]$$

$$= \min_G \max_D \int_x p_r(x) \log D(x) + \int_z p_z(z) \log(1 - D(G(z)))$$

$$= \min_G \max_D \int_x p_r(x) \log D(x) + \int_x p_g(x) \log(1 - D(x))$$

$$= \min_G \max_D a \log(y) + b \log(1 - y)$$

Expected value

$$\mathrm{E}_{x \sim q} f(x) = \int q(x) f(x) \, dx$$

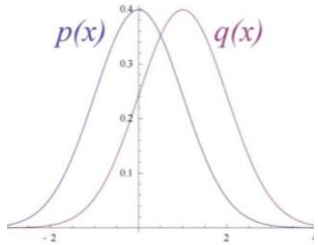
Fixed Generator

Assuming known generator image distribution p_g

Kullback–Leibler divergence and entropy (self-study)

Definition

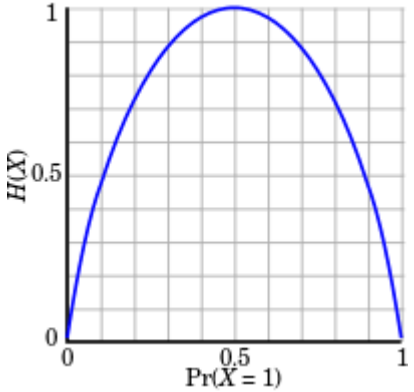
$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$



- a distance between two distributions

Interpretation

- information gain achieved if Q is used instead of P
- relative entropy
  - Entropy: $H(p) = -\sum_{i=1}^{n} p(x_i) \log p(x_i).$
- the expected number of extra bits required to code samples from P using a code optimized for Q rather than the code optimized for P



GAN derivation cont. (self-study)

The optimal (extremum) of a function of form $a\log(y) + b\log(1-y)$ is $y^* = \frac{a}{a+b}$

$a\log(y) + b\log(1-y)$

$y' = \frac{a}{y} - \frac{b}{1-y}$

$\frac{a}{y^*} = \frac{b}{1-y^*}$ Find optimal y^* by setting $y' = 0$.

$\frac{1-y^*}{y^*} = \frac{b}{a}$

$\frac{1}{y^*} = \frac{a+b}{a}$

$y^* = \frac{a}{a+b}$

From the general form, it follows that the maximum is reached for the discriminator D*

$$p_r(x)\log D(x) + p_g(x)\log(1-D(x)) \qquad \Longrightarrow \quad D^*(x) = \frac{p_r(x)}{p_r(x)+p_g(x)}$$

We assumed that the generator, G, is fixed and we have a way to evaluate p_g (generated image distr.)

- in practice, we can not estimate p_g (opposed to a VAE)
  - we can only sample from p_g by sampling from p_z and applying G
- but for the mathematical derivation we can make this assumption

GAN derivation cont. (self-study)

Using the optimal value of D, we reach a form that is equal to the JS-divergence

$$\min_G V(D^*, G) = \int_x \Big(p_r(x) \log D^*(x) + p_g(x) \log(1 - D^*(x)) \Big) dx$$
$$= \int_x \left(p_r(x) \log \frac{p_r(x)}{p_r(x) + p_g(x)} + p_g(x) \log \frac{p_g(x)}{p_r(x) + p_g(x)} \right) dx$$

$$D_{JS}(p_r \| p_g) = \frac{1}{2} D_{KL}(p_r || \frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g || \frac{p_r + p_g}{2})$$
$$= \frac{1}{2} \left(\int_x p_r(x) \log \frac{2p_r(x)}{p_r(x) + p_g(x)} dx \right) + \frac{1}{2} \left(\int_x p_g(x) \log \frac{2p_g(x)}{p_r(x) + p_g(x)} dx \right)$$
$$= \frac{1}{2} \left(\log 2 + \int_x p_r(x) \log \frac{p_r(x)}{p_r(x) + p_g(x)} dx \right) +$$
$$\frac{1}{2} \left(\log 2 + \int_x p_g(x) \log \frac{p_g(x)}{p_r(x) + p_g(x)} dx \right)$$
$$= \frac{1}{2} \Big(\log 4 + \min_G V(D^*, G) \Big)$$

Jensen–Shannon divergence

$$\mathrm{D}_{\mathrm{JS}}(P \parallel Q) = \frac{1}{2} D_{KL}(P \parallel M) + \frac{1}{2} D_{KL}(Q \parallel M)$$

with $M = \frac{1}{2}(P + Q)$

Kullback–Leibler divergence (relative entropy)

- dissimilarity measure between distributions
  - not symmetric, KL(p,q) != KL(q,p)
- Definition for continuous distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \left(\frac{p(x)}{q(x)} \right) dx$$

probability density of P and Q, the distribution of real and fake images

[https://medium.com/@jonathan_hui/proof-gan-optimal-point-658116a236fb]

From classical (JS) to Wasserstein GAN

Diverse measures exist to compare probability distributions (here generated and real image distribution)

- The *Total Variation* (TV) distance

$$\delta(\mathbb{P}_r, \mathbb{P}_g) = \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_g(A)| \ .$$

- The *Kullback-Leibler* (KL) divergence

$$KL(\mathbb{P}_r \| \mathbb{P}_g) = \int \log \left(\frac{P_r(x)}{P_g(x)} \right) P_r(x) d\mu(x) \ ,$$

- The *Jensen-Shannon* (JS) divergence

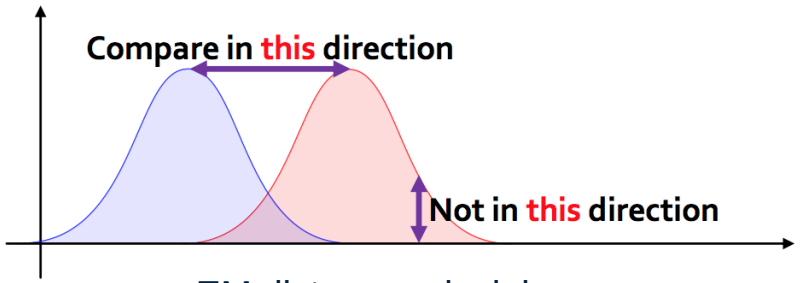
$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r \| \mathbb{P}_m) + KL(\mathbb{P}_g \| \mathbb{P}_m) \ ,$$

$$\text{where} \quad \mathbb{P}_m = (\mathbb{P}_r + \mathbb{P}_g)/2$$

JS is what the classical GAN optimizes

- The *Earth-Mover* (EM) distance or Wasserstein-1

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right] \ ,$$



EM distance principle

[Arjovsky et al., Wasserstein GAN. 2017]

GAN vs. WGAN

Wasserstein distance is even simpler!

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

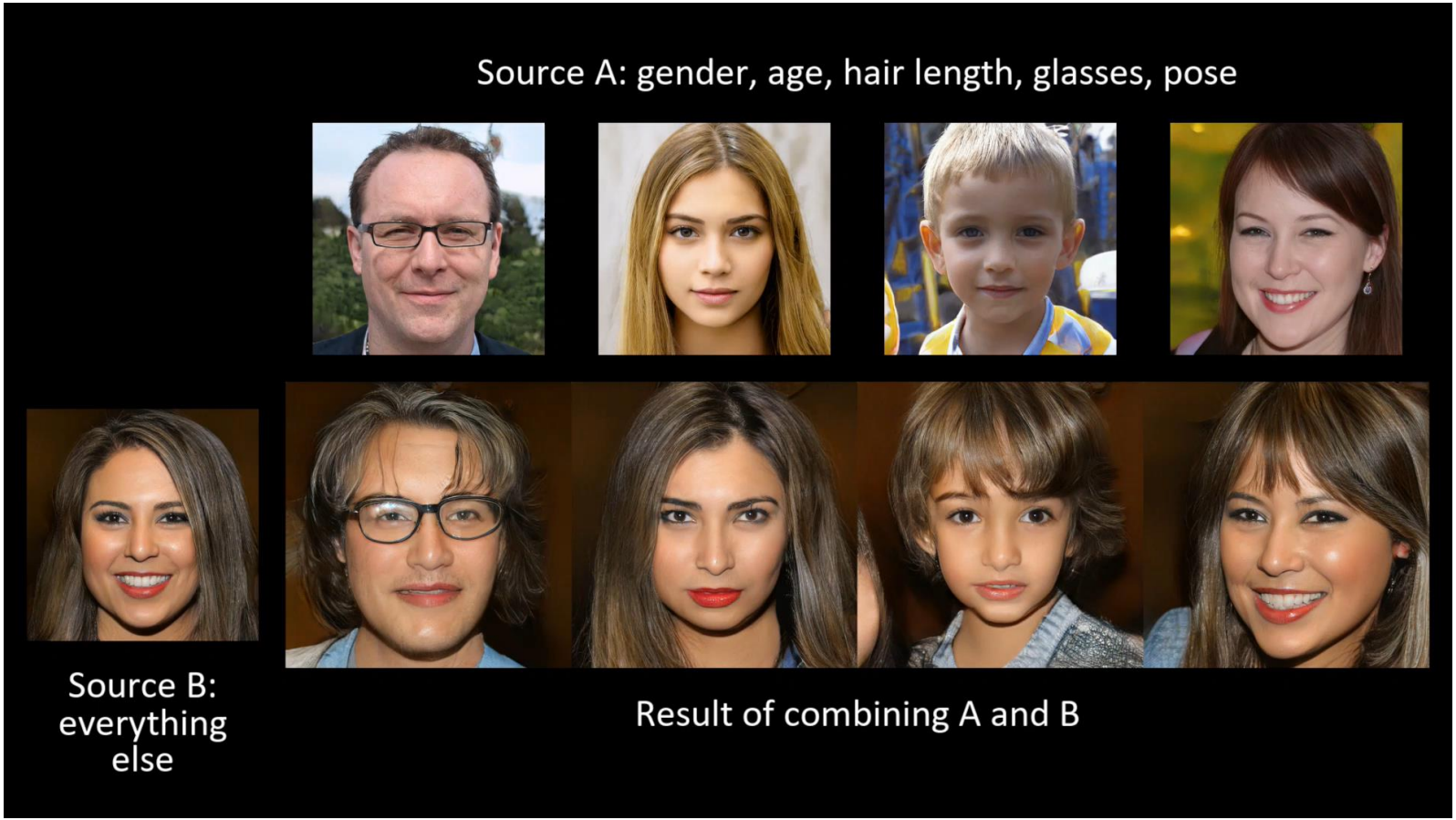
Require: : α, the learning rate. c, the clipping parameter. m, the batch size.
n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

WGAN

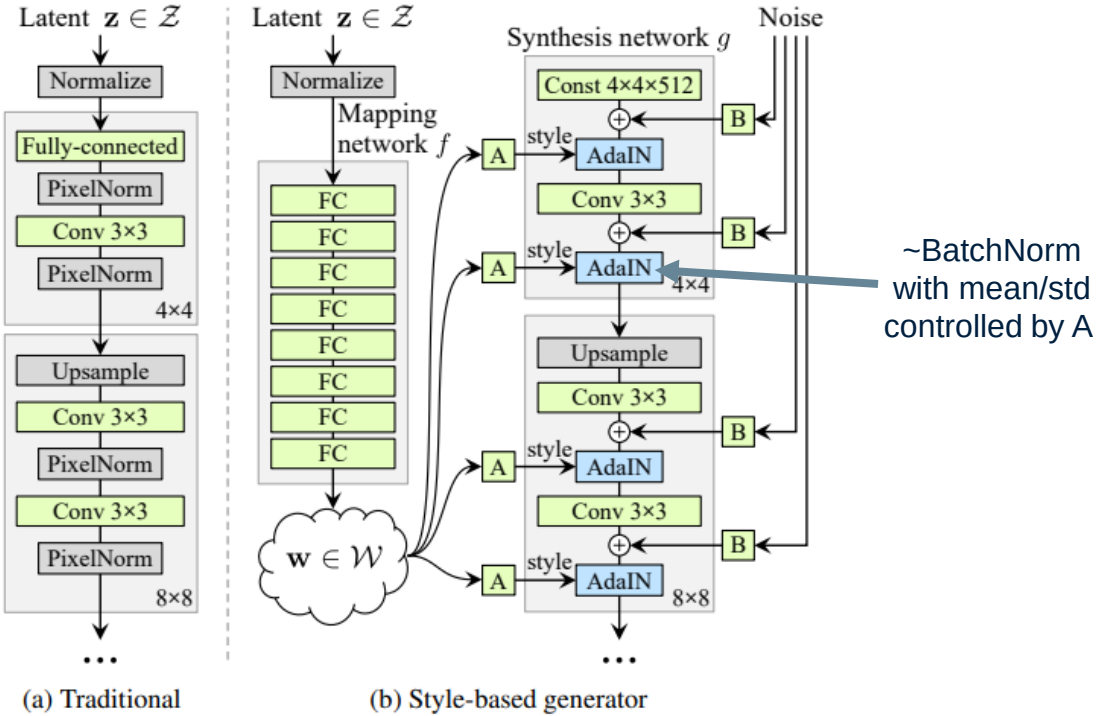
the only difference is the log
(besides different notations)

Style GAN results

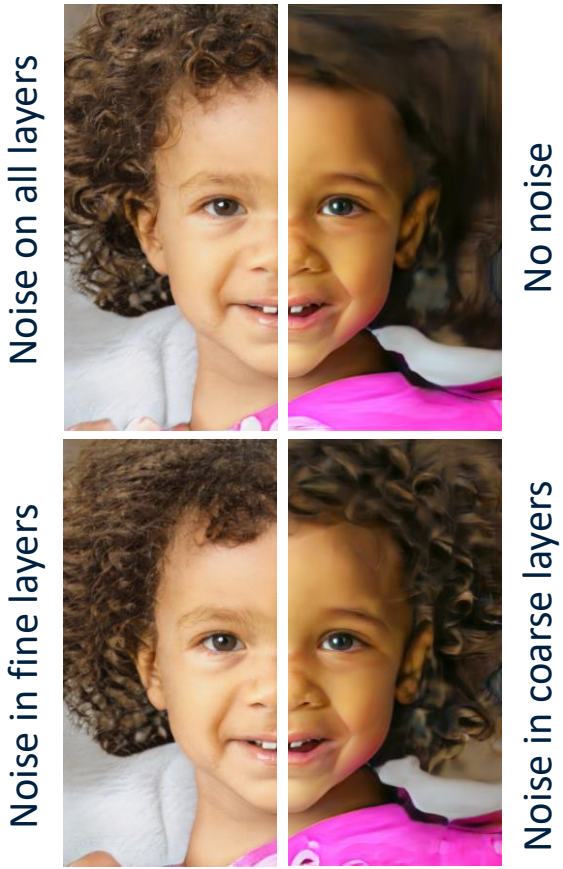


Style GAN internals

- Compute style description given noise (form of non-Gaussian noise)
- Apply style and add noise at all layers (of ProgGAN generator)



~BatchNorm with mean/std controlled by A



Adaptive Instance Normalization (AdaIN)

Instance Normalization:

- like batch norm, but normalizing across the spatial dimensions (instead of elements in the batch)

$$\mathrm{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta$$

Conditional Instance Normalization

- make the offset and scaling (gamma and beta) dependent on a style *s*
  - *e.g., extracted with pre-trained network*

$$\mathrm{CIN}(x; s) = \gamma^s \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta^s$$

Adaptive Instance Normalization

- normalize the mean and std of the target with the one of the source

$$\mathrm{AdaIN}(x, y) = \sigma(y) \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

[Huang and Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization]

(Unpaired) Image translation

Paired vs. unpaired image translation

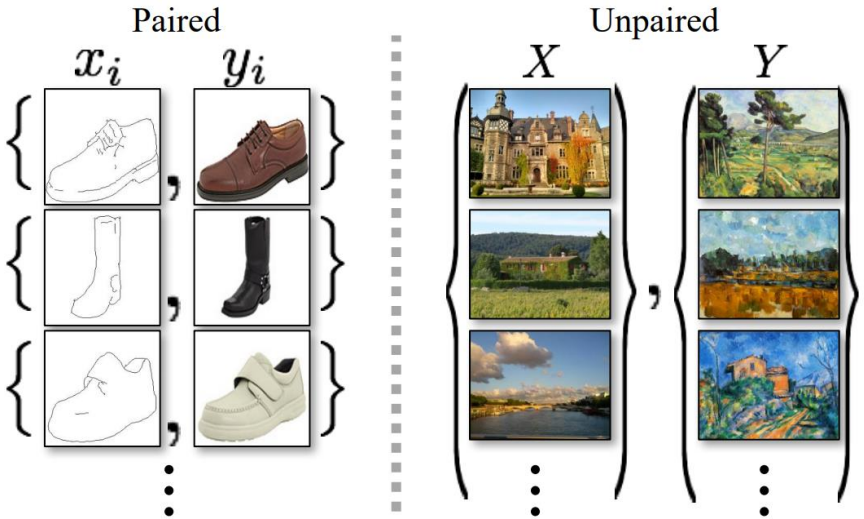
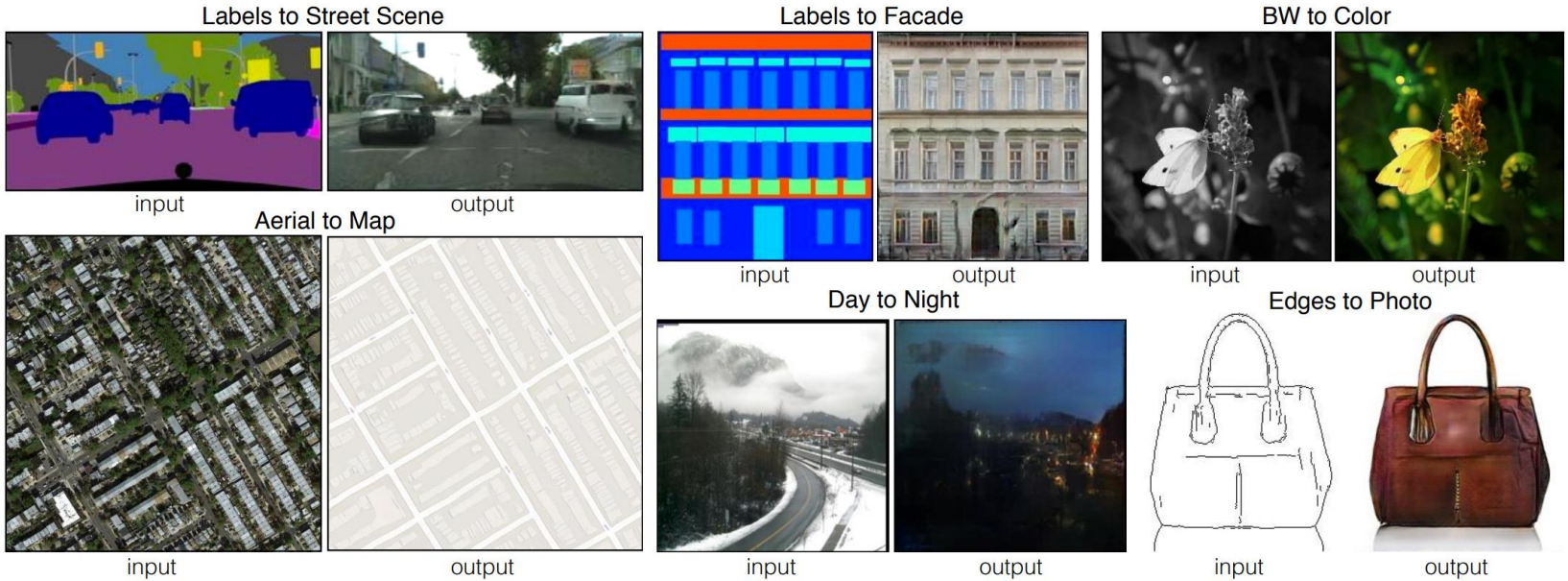


Image translation

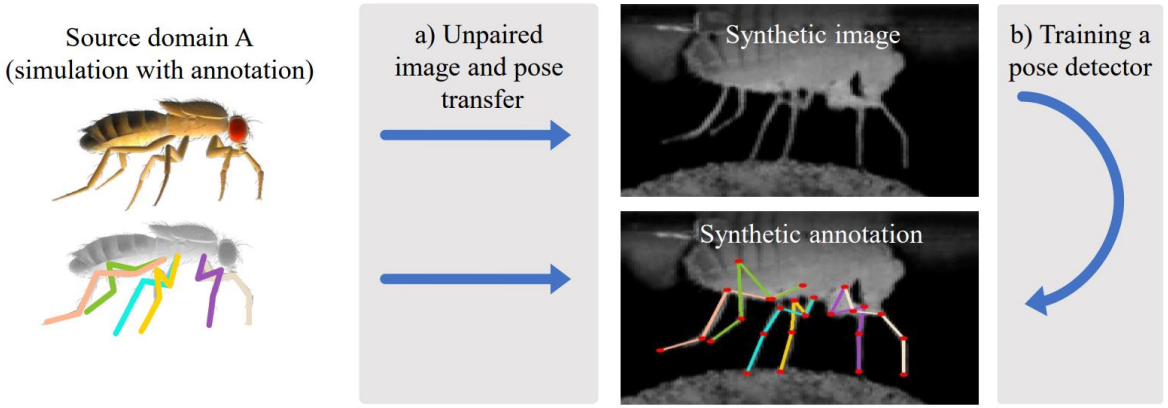
First week of paper reading..



[Isola et al., Image-to-Image Translation with Conditional Adversarial Networks]

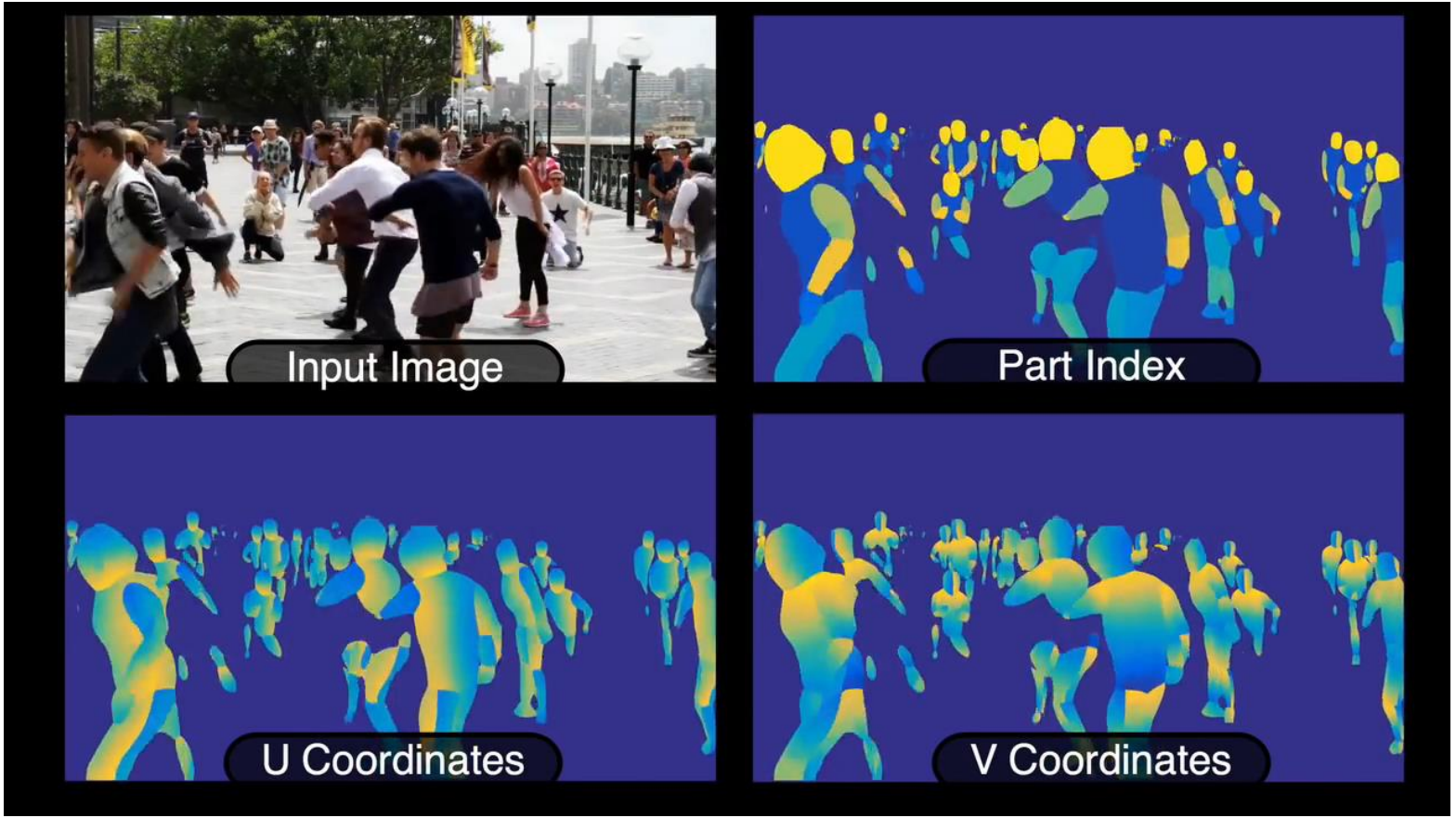
Further image to image translation examples

[Everybody dance now]



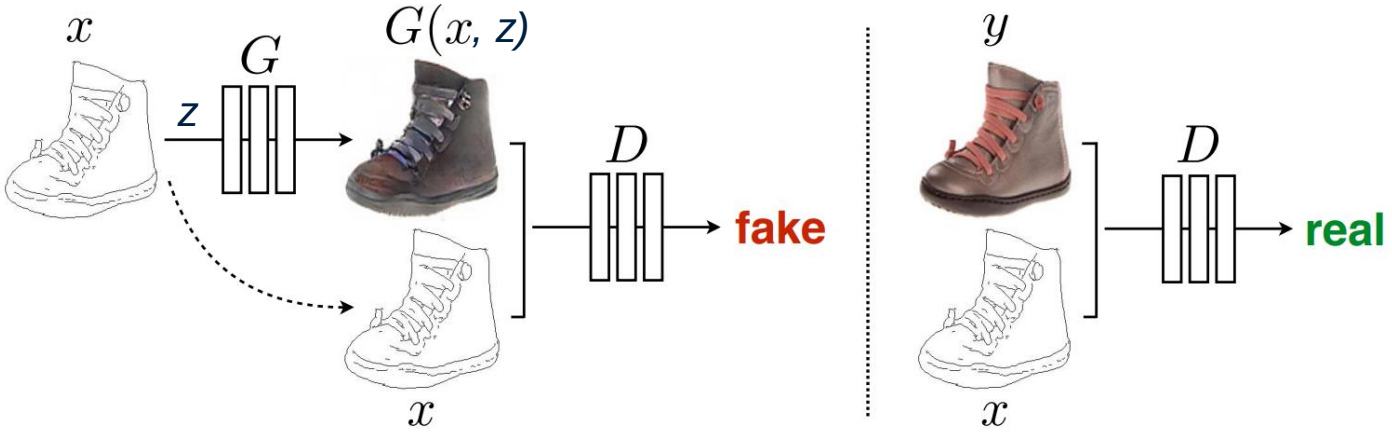
[Deformation-aware Unpaired Image Translation for Pose Estimation on Laboratory Animals]

Even more image to image translation examples



[Dense pose]

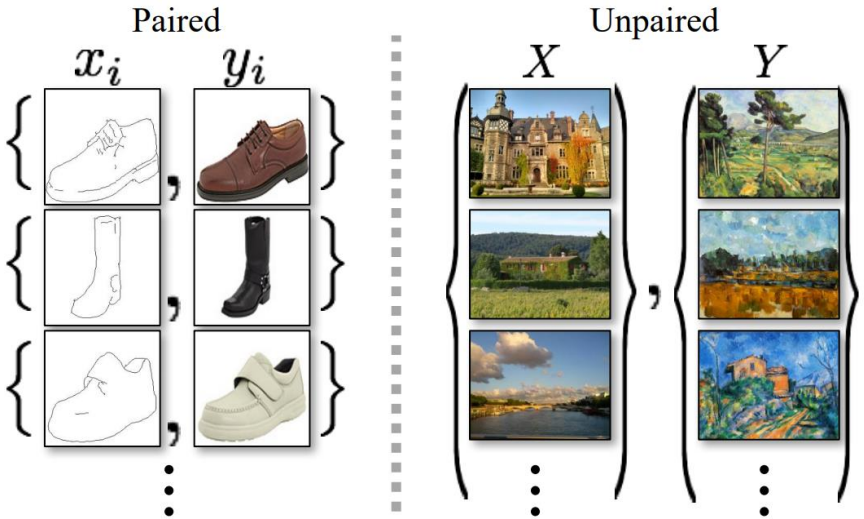
Conditional Generative Adversarial Nets



GAN, but with additional input (here edge map) on top of the noise

- the noise will trigger properties that are hidden in the condition, here color
- both the generator and discriminator receive the condition as input

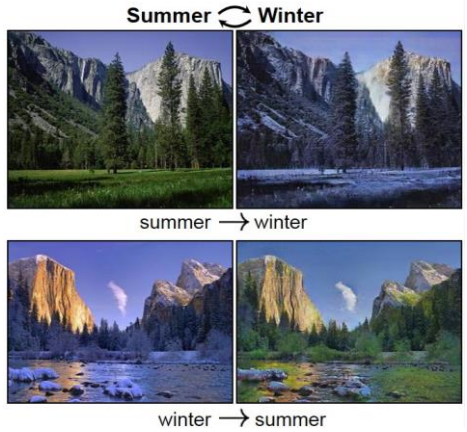
Paired vs. unpaired



Cycle GAN

Unpaired image translation

- a set of images for the source
  (e.g., many paintings)
- a set of images for the target
  (e.g., real photographs)
- no image-to-image spatial correspondence
- no image-to-image color correspondence

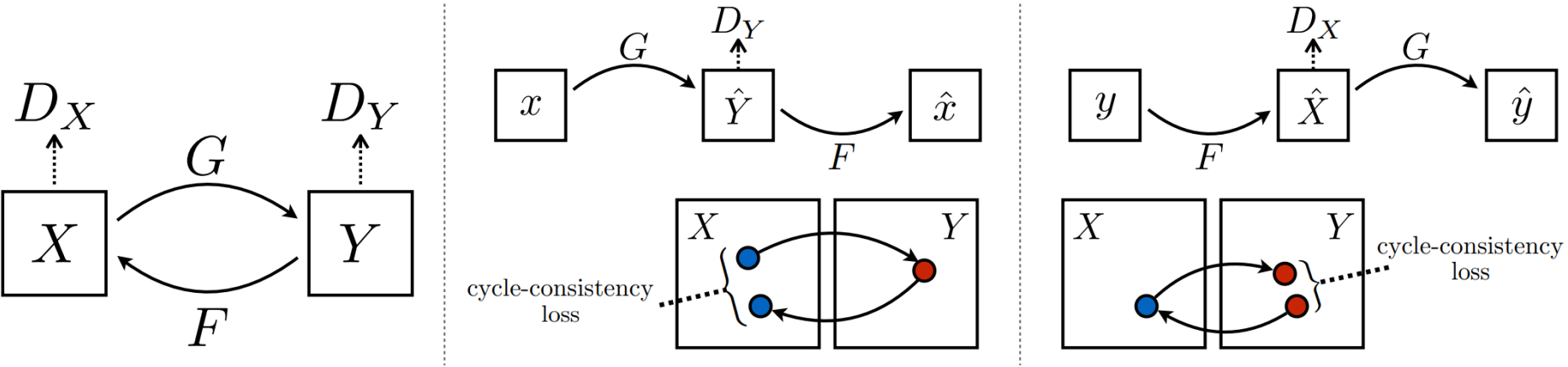


How can we learn a mapping?

- by limiting the capacity of the translator
  - few parameters
  - local operations (convolution)
- ensuring that the generated target images are realistic
  - similar in distribution

Cycle GAN principle

Construct an identity function by chaining two translation networks



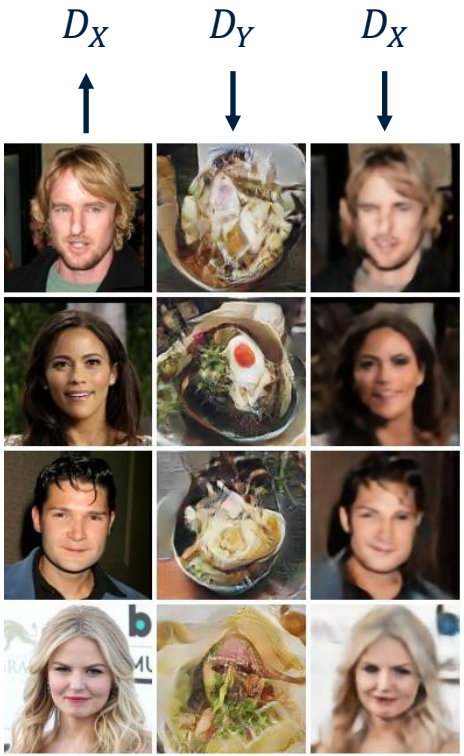
- Jointly learn to
  - map from X to Y and back to X
  - map from Y to X and back to Y

Canonical solutions?

Training examples (face to ramen)

- example images of both classes in one batch
- map between domains
  - one generator per class
- apply discriminator on all generated images
  - one discriminator per class

x → y' → x'
(fake) (fake)



y → x' → y'
(fake) (fake)

Cycle GAN issues

Warning:

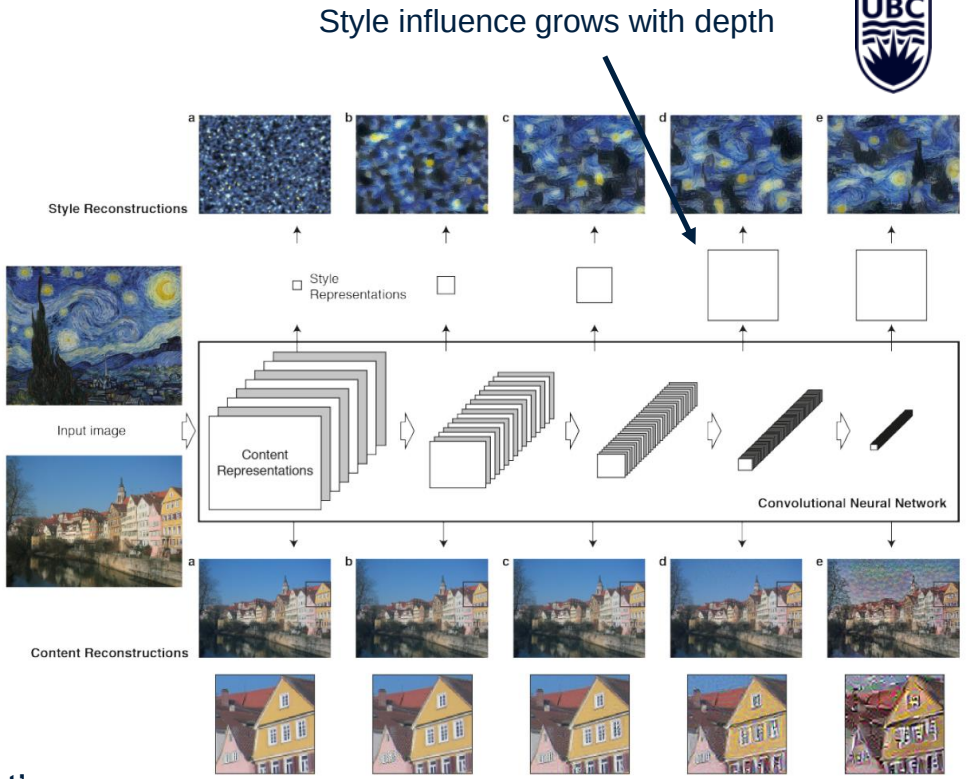
- difficult to train
- requires similar objects in source and target
  - e.g., zebra and horse
- works best on textures, not so well on shape changes
  - good on zebra and horse
  - bad to translate from mouse to elephant

Further reading

Style transfer

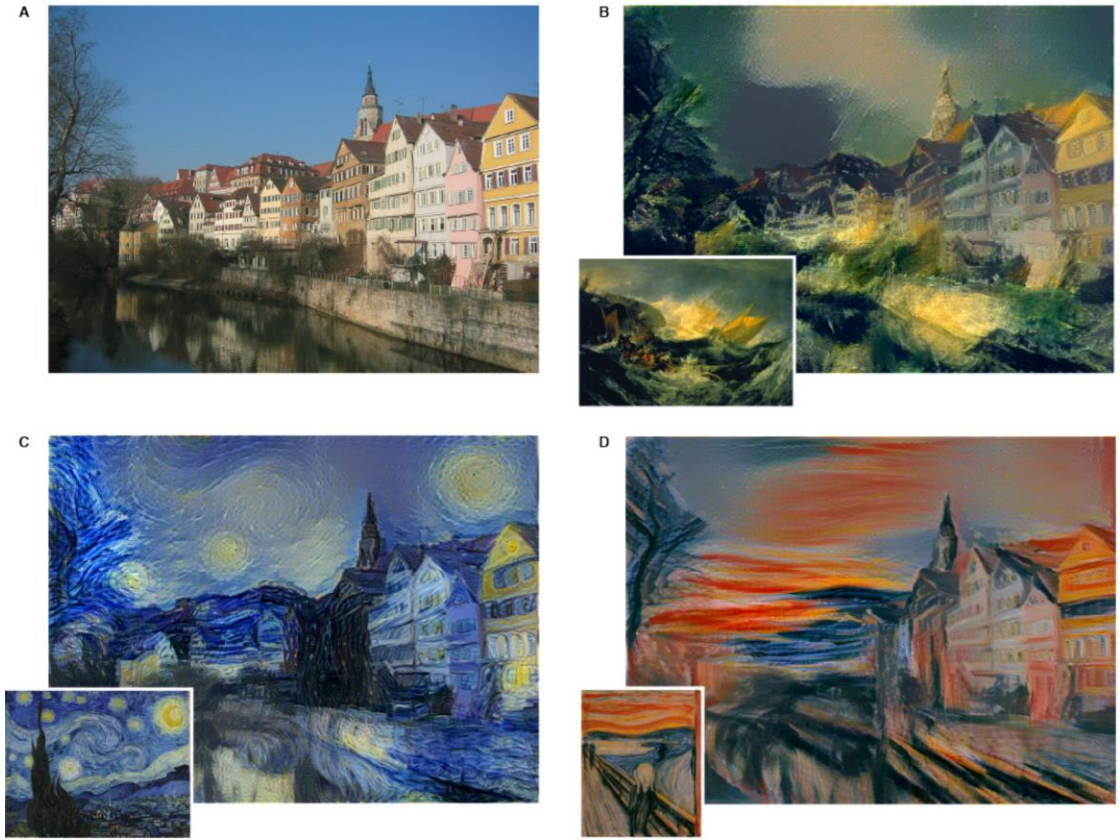
Idea: ‘turn NN training on its head’

- apply gradient descent
  - with respect to the ‘input’ image (instead of NN weights)
  - keep the neural network weights fixed
- find neural network features that
  1. capture style (averaged spatially)
     - correlation between features of a layer
  2. capture content
     - l2 difference between features of a layer
- set the objective function as the distance of ‘input’
  - to style target (painting), in terms of style features
  - to content target (photo), in terms of content features



[Gatys et al., A Neural Algorithm of Artistic Style 2015]

Style transfer results



Progressive GAN (ProgGAN)

Concept:

- build a classical GAN setup
  - generator G
  - discriminator D
- start low res (4x4)
- train for a bit, then add new layers
- optimize all layers
  - new and old

Benefits:

- quick convergence
- scales to high resolution
  - 1024 x 1024

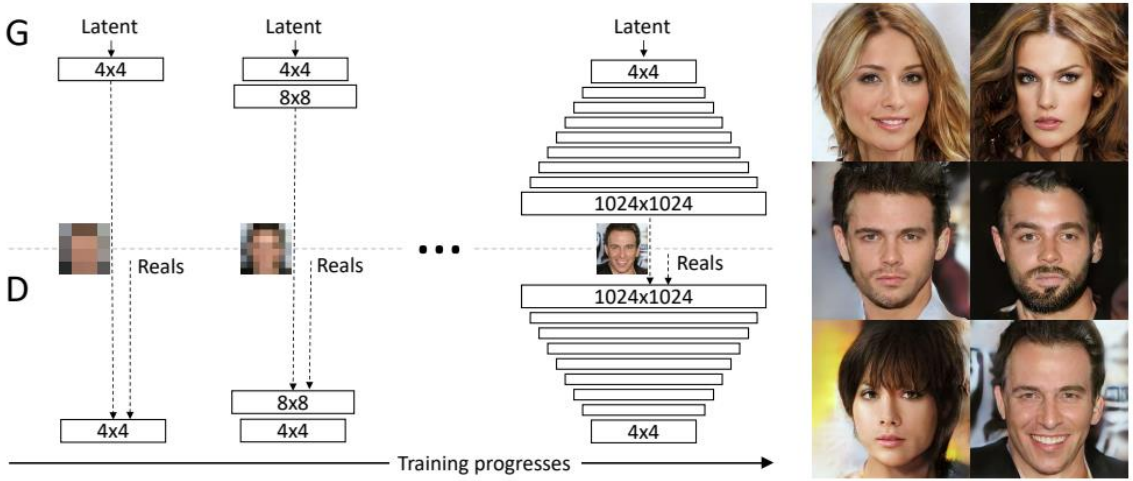
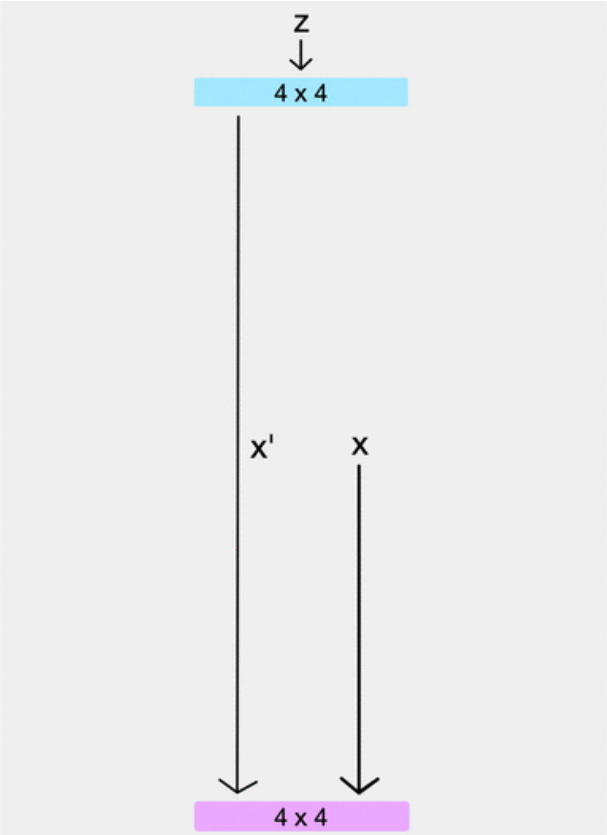
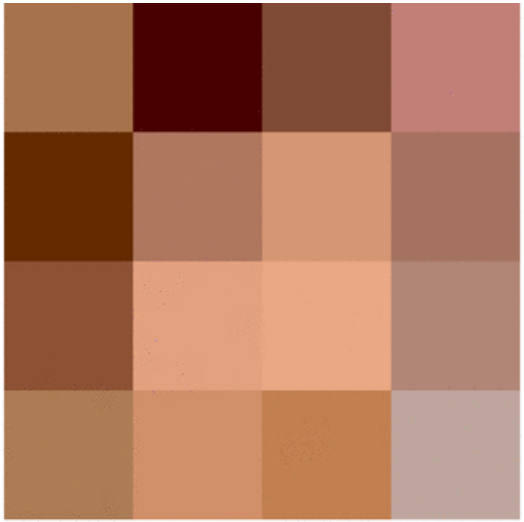


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $N \times N$ refers to convolutional layers operating on $N \times N$ spatial resolution. This allows stable synthesis in high resolutions and also speeds up training considerably. One the right we show six example images generated using progressive growing at 1024×1024.

Progressive GAN example





Training time: 0 days
4x4 resolution

Generator
Discriminator

z = random code
x = real image
x' = generated image