Visual AI

CPSC 532R/533R – 2019/2020 Term 2

Lecture 10. GANs and Unpaired Image Translation

Helge Rhodin

Assignment 3

- Rendering
- Learning shape spaces
- Interpolating in shape spaces

- Work independently, 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.

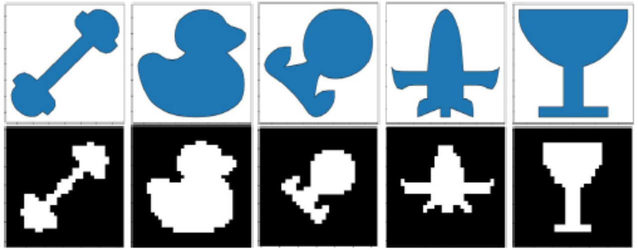
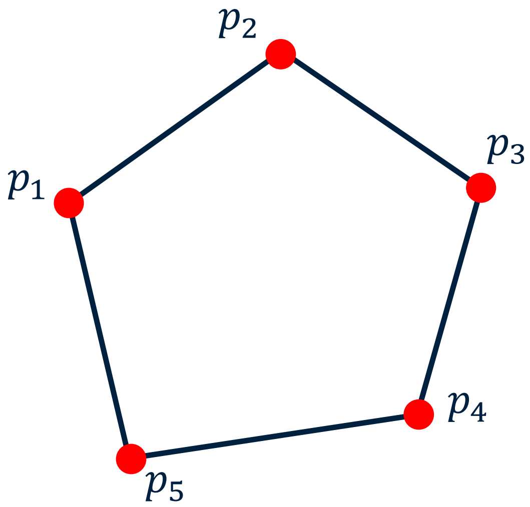
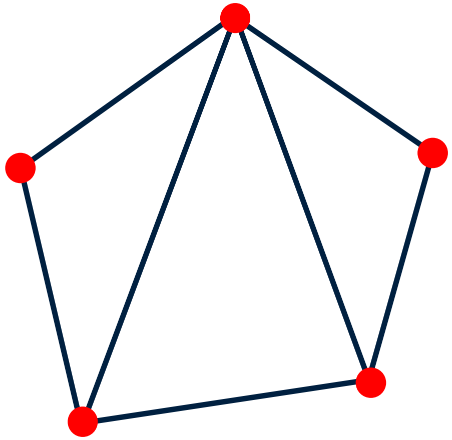


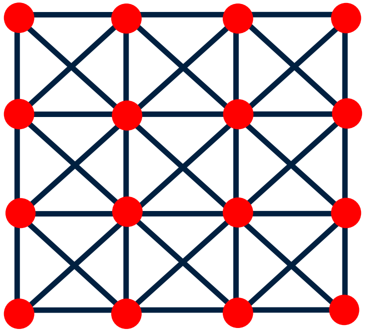
Figure 1: Icon vector graphics and their bitmap representation.

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.

Polygon vs. mesh vs. image







Polygon
- two neighbors per vertex
- suited for 1D convolution

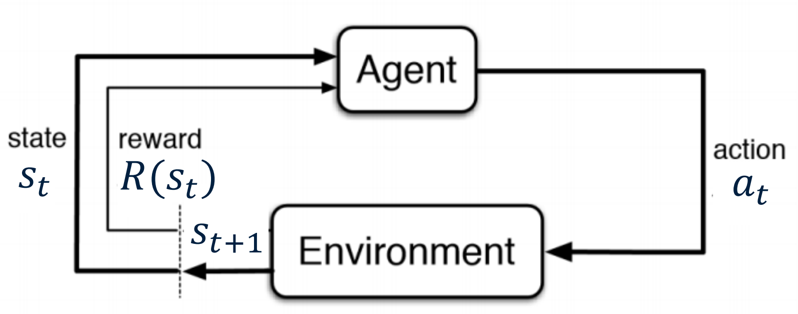
Mesh (e.g., triangles)
- different #neighbors per vertex
- requires graph convolution

Image (regular grid)
- eight neighbors per vertex
- suited for 2D convolution

The order of vertices is not important for defining shapes

Translating the image left/right has no effect on convolution

Recap: Reinforcement learning basics



Definitions:

- s_t, the current state of the agent/environment
- $R(s_t)$, the reward/objective at time t
  - might be zero for almost all t
- $R = \sum_{t=0}^{T} R(s_t)$, the return as sum over all rewards
- a, the action, such as moving right or left
- $a_t = \pi(s_t)$, the policy of which action a_t to perform when in state s_t
- $s_{t+1} = env(s_t, a_t)$, the environment reacting to the agent's action

Goal: finding a good policy π such that R is maximized when executing action $a_t = \pi(s_t)$

Update loop:

- decide on a new action $a_t = \pi(s_t)$
- update the environment state $s_{t+1} = env(s_t, a_t)$
- pay out reward $R(s_t)$

Recap: Binary decisions

Computing expectations

Continuous:

Definition

$$\mathrm{E}_{x\sim p}\, f(x) = \int_{\Omega} f(x)p(x)\, dx$$

Discrete set of C classes:

Definition

$$\mathrm{E}_{x\sim p}\, f(x) = \sum_{i=1}^{C} f(x_i)p(x_i)$$

Estimators

Empirical estimate

$$\mathrm{E}_{x\sim p} \approx \frac{C}{N}\sum_{i=1}^{N} f(x_i) \text{ with } x_i \sim p$$

Uniform Monte Carlo sampling

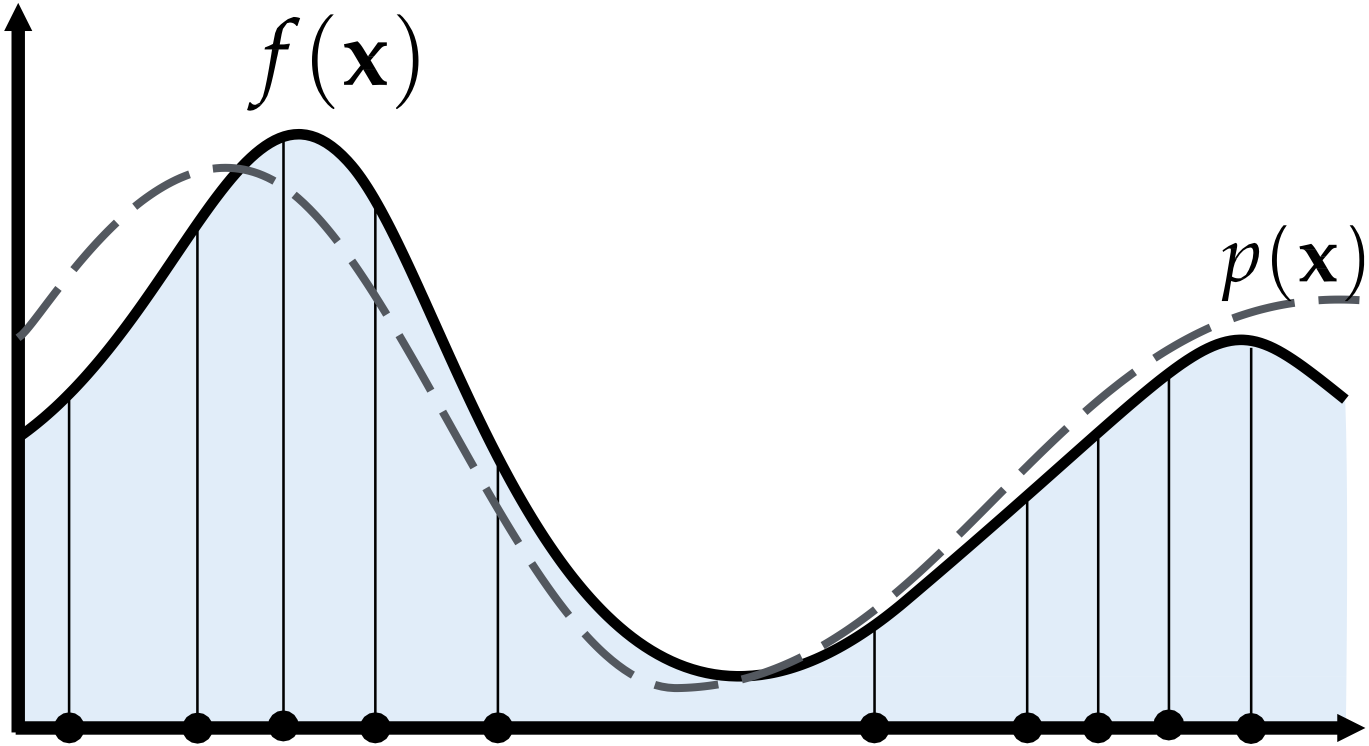
$$\mathrm{E}_{x\sim p} \approx \frac{C}{N}\sum_{i=1}^{N} f(x_i)p(x_i)$$

with N samples x_i drawn uniformly at random

Importance sampling

$$\mathrm{E}_{x\sim p} \approx \frac{C}{N}\sum_{i=1}^{N} \frac{p(x_i)}{q(x_i)} f(x_i) \text{ with } x_i \sim q$$

Recap: Importance sampling



Derivative of discrete random variables II

1. Start from uniform MC sampling of f
   (note, not yet of the gradient)

$$\mathrm{E}[f] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)p_\theta(x_i) \qquad \text{with } x_i \sim \text{Uniform}$$

2. Importance sample with distribution q

$$\mathrm{E}[f] \approx \frac{1}{N}\sum_{i=1}^{N} \frac{p_\theta(x_i)}{q(x_i)} f(x_i) \qquad \text{with } x_i \sim q$$

3. Compute gradient
   (before this was the first step)

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \sum_{i=1}^{N} \frac{\frac{\partial p_\theta(x_i)}{\partial \theta}}{q} f(x_i)$$

4. Assume q=p and express as logarithm
   - the same as log trick!
   - but now it makes sense
     - importance sampling with the current policy
     - we don’t change the samples, hence, no gradient flow through q

$$\frac{\partial \mathrm{E}[f(X)]}{\partial \theta} \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \frac{\partial \log\left(p_\theta(x_i)\right)}{\partial \theta}$$

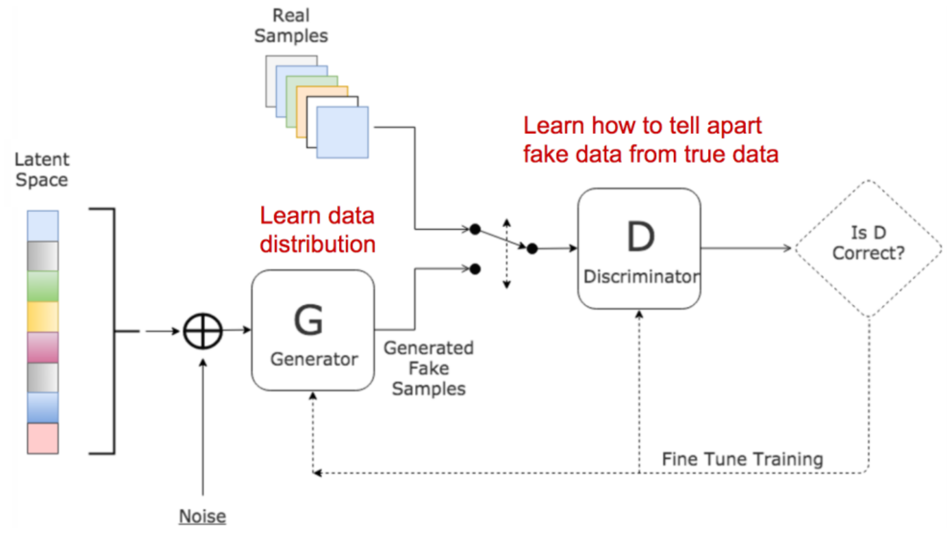
- Advantages: We can sample from q != p, i.e. to encourage exploitation or reduce variance. Easier to implement and understand. A large literature on how to improve importance sampling!

Generative Adversarial Networks (GAN)

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

Recap: GANs

A min max game (related to game theory)

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

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

D should be **low** for fake examples
(from perspective of D)

D should be **high** for fake examples
(from perspective of G)

- Effects:
  - learning a loss function
  - like a VAE, we sample from a Gaussian distribution (some form of a prior assumption)

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:

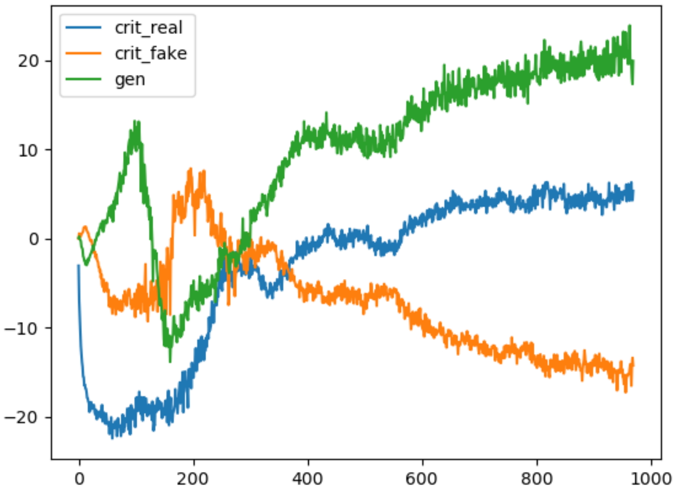
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Blue: outer loop on generator (gradient descent)

Green: inner loop on discriminator (gradient ascent)



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

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

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

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

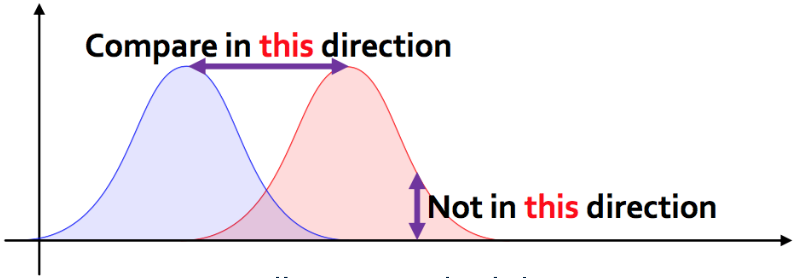
- The *Jensen-Shannon* (JS) divergence

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

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

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

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



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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ 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 $t = 0, \dots, n_{\text{critic}}$ **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

GAN derivation (self-study)

The GAN objective has the form

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

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

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

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

The optimal (extremum) is $y^* = \dfrac{a}{a+b}$,

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

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

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

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

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

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

Expected value

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

Fixed Generator

Assuming known generator image distribution p_g

GAN derivation (self-study)

From the optimum $y^* = \frac{a}{a+b}$ of 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 (self-study)

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

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

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

Jensen–Shannon divergence

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

with

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

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

Comparison: VAE and GAN

	VAE	GAN
Objective	$\min_{\theta,\phi} -\mathbf{E}_{\mathbf{h}\sim q_\phi(\mathbf{h}\|\mathbf{x})}\big(\log p_\theta(\mathbf{x}\|\mathbf{h})\big) + D_{\mathrm{KL}}(q_\phi(\mathbf{h}\|\mathbf{x})\\|p(\mathbf{h}))$	$\min_G \max_D [E_{x\sim p_r}[\log D(x)] + E_{z\sim p_z}[\log(1-D(G(z)))]]$

Sampling a ‘natural’ image

VAE

- Draw a random sample from a Gaussian

$$\mathbf{h} \sim \mathcal{N}(0,1)$$

- Apply the decoder on h

GAN

- Draw a random sample from a Gaussian

$$z \sim \mathcal{N}(0,1)$$

- Apply the generator on z

Computing the probability of a given image x

VAE

- Apply the encoder on x

$$\mathbf{h} = e_\theta(\mathbf{x})$$

- Evaluate the prior on h

$$\mathcal{N}(\mathbf{h}|0,1)$$

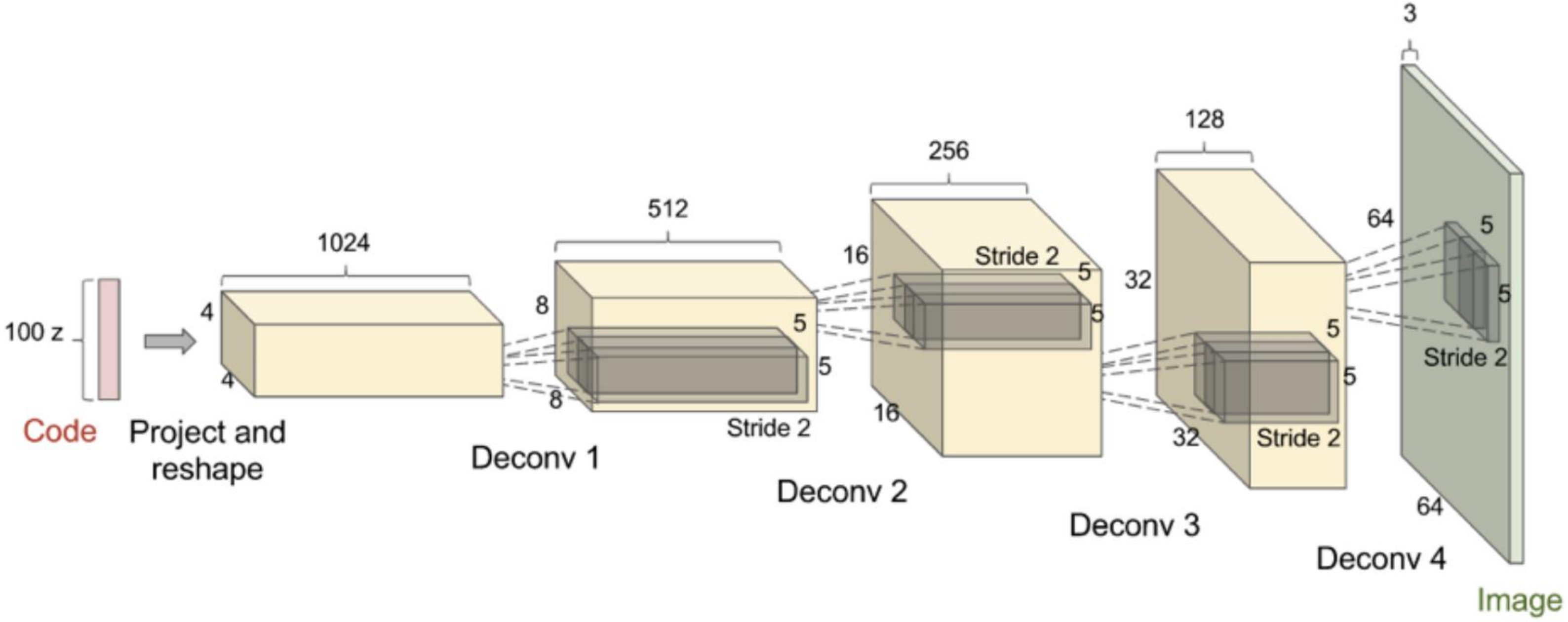
- thanks to explicit density model

GAN

- Not applicable!
  - it models an implicit density

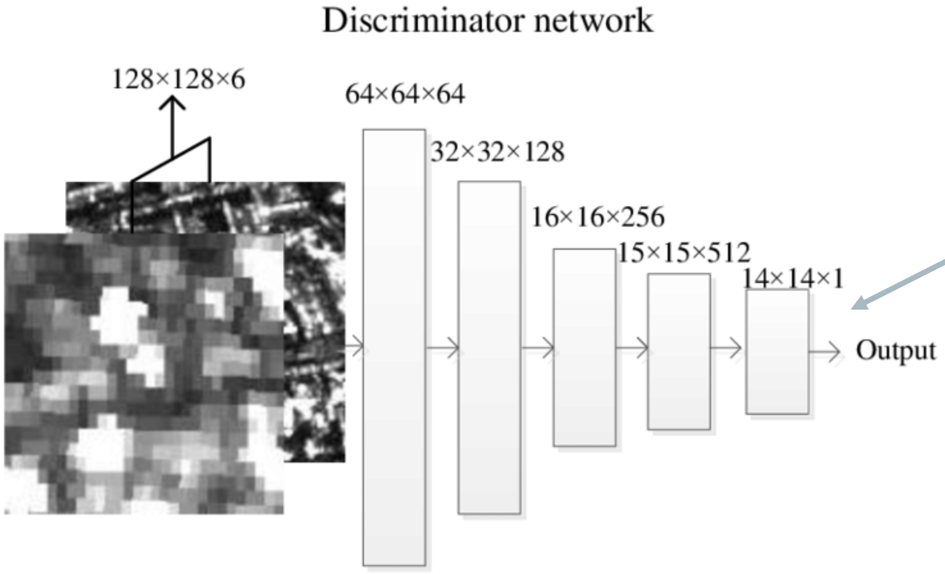
DCGAN

Convolutional generator architecture



PatchGAN

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



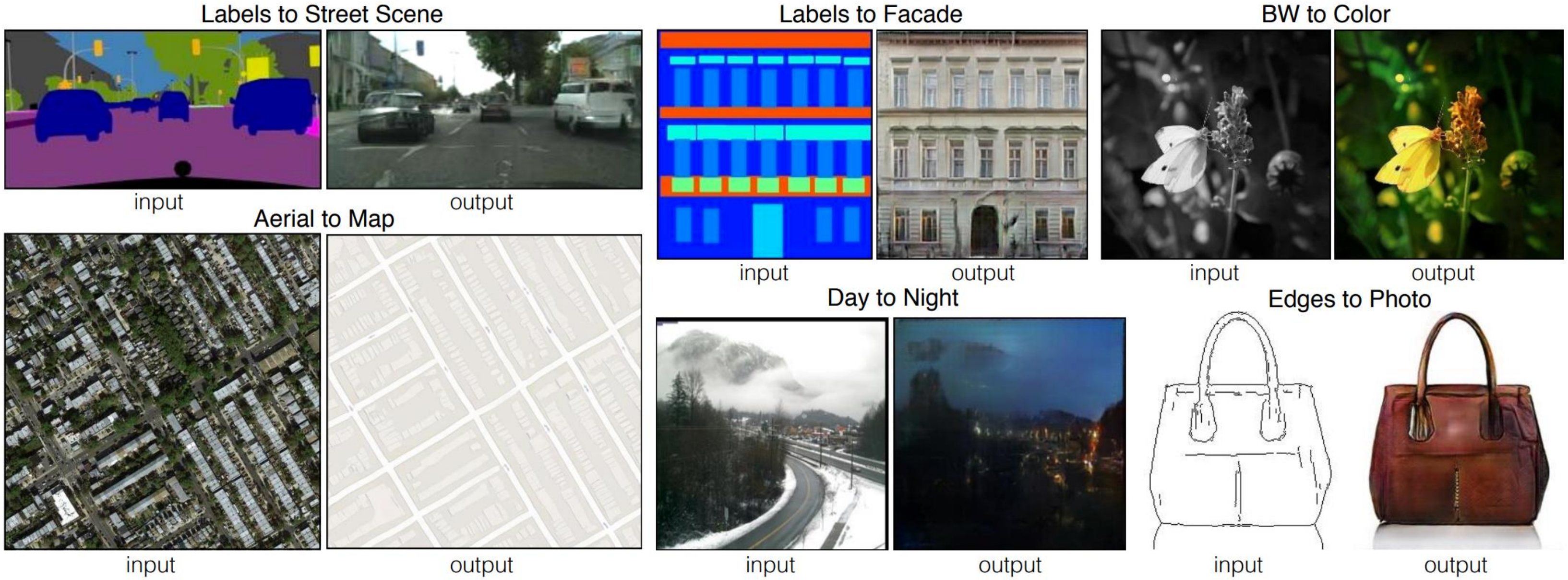
A feature map of
fake/real probabilities

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

Image translation

Image translation

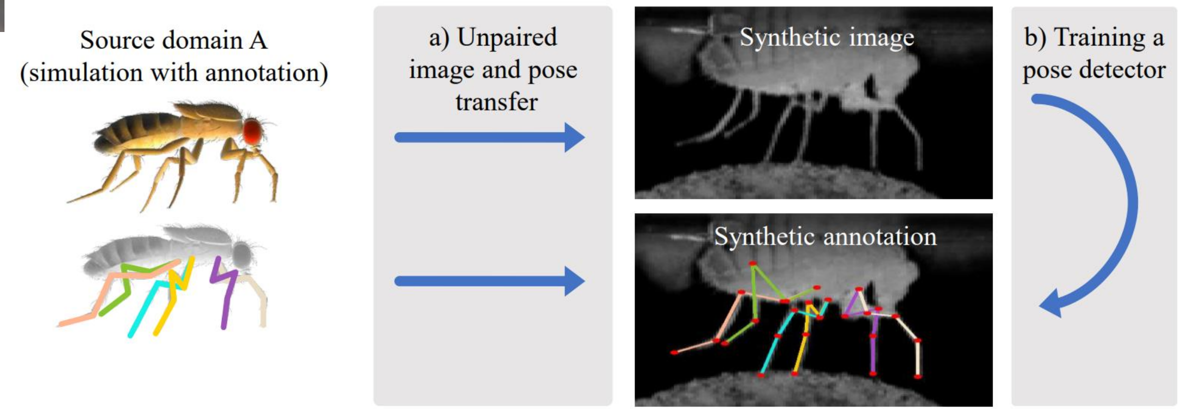
First week of paper reading..



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

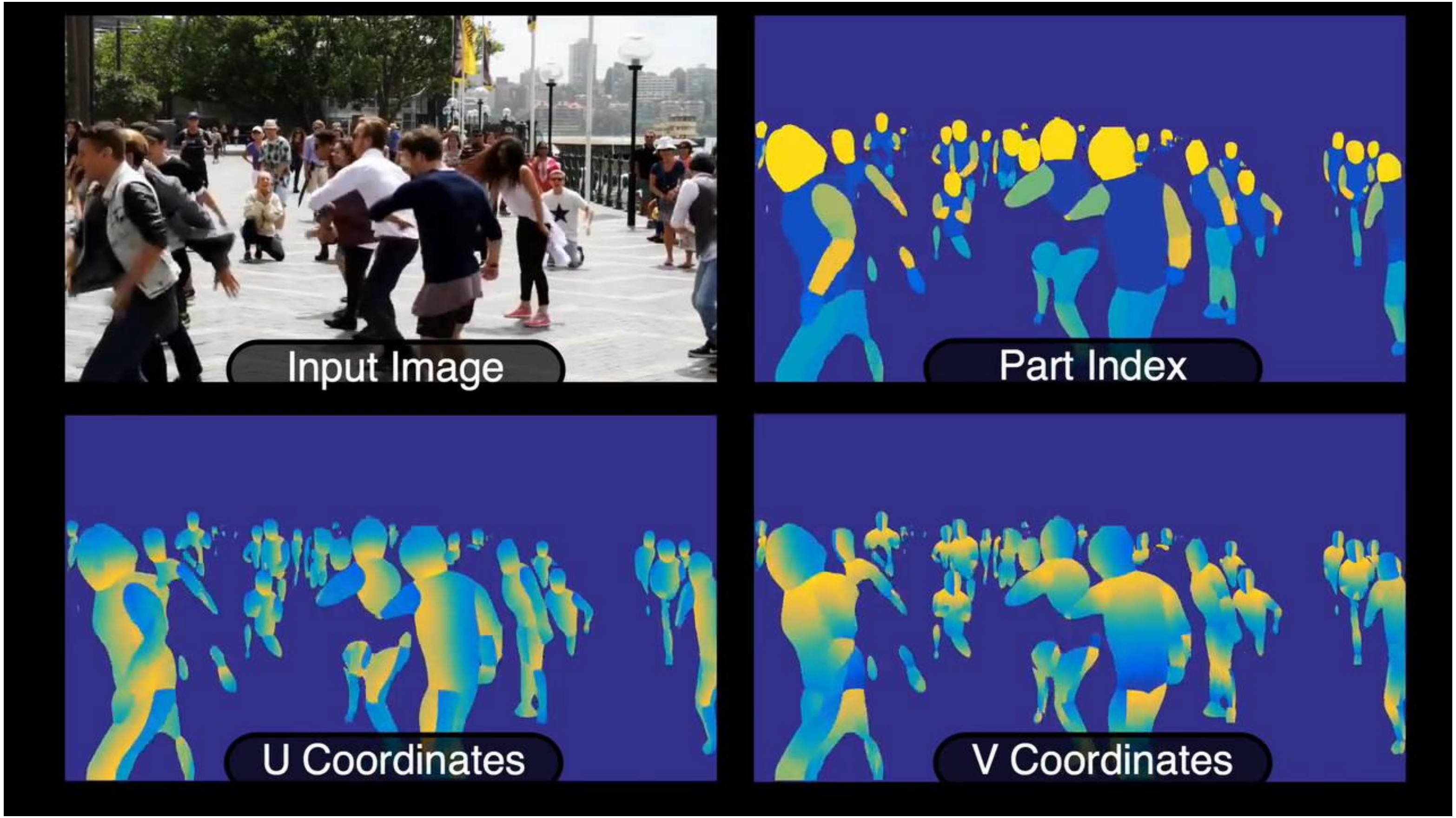
Further image to image translation examples

[Everybody dance now]



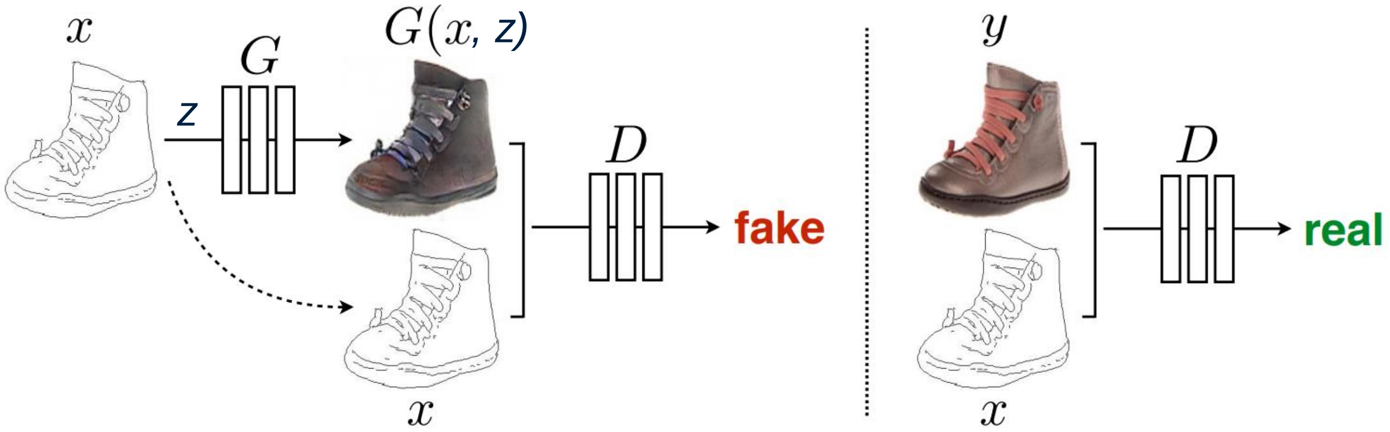
[Deformation-aware Unpaired Image Translation for Pose Estimation on Laboratory Animals]

Even more image to image translation examples



[Dense pose]

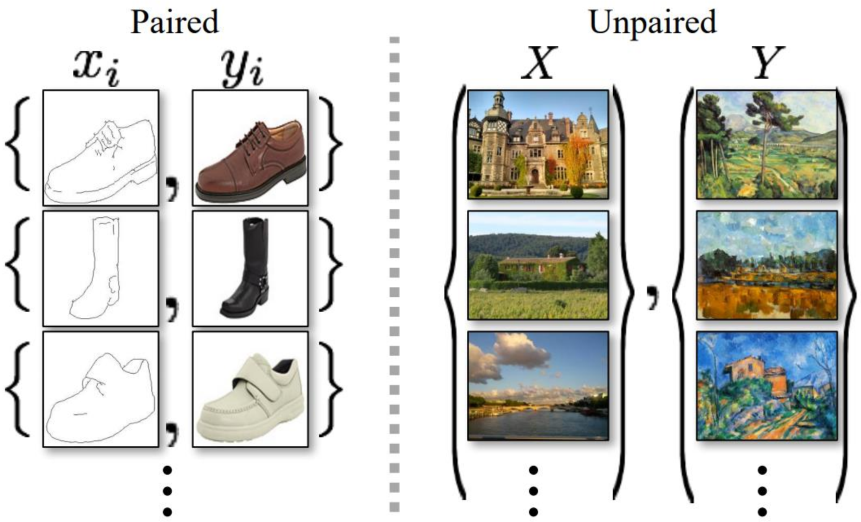
Conditional Generative Adversarial Nets



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



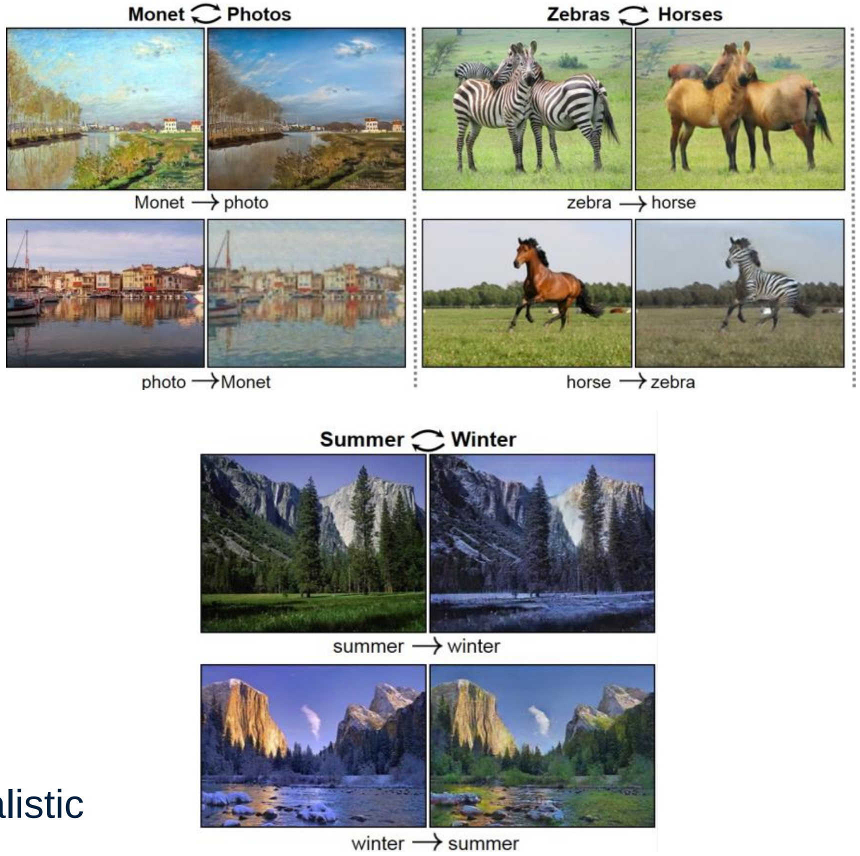
Cycle GAN

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

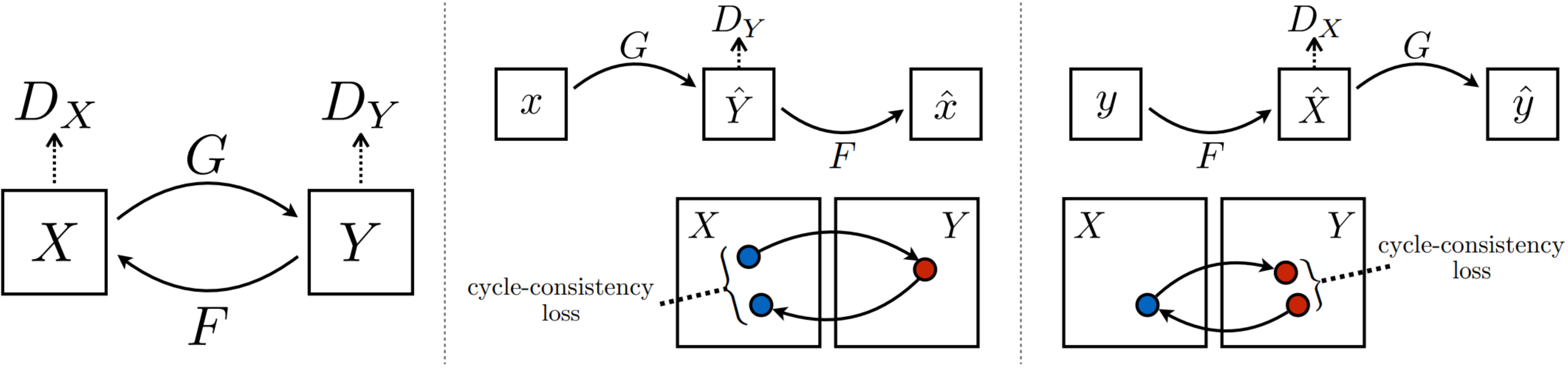
How can we learn a mapping?

- 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

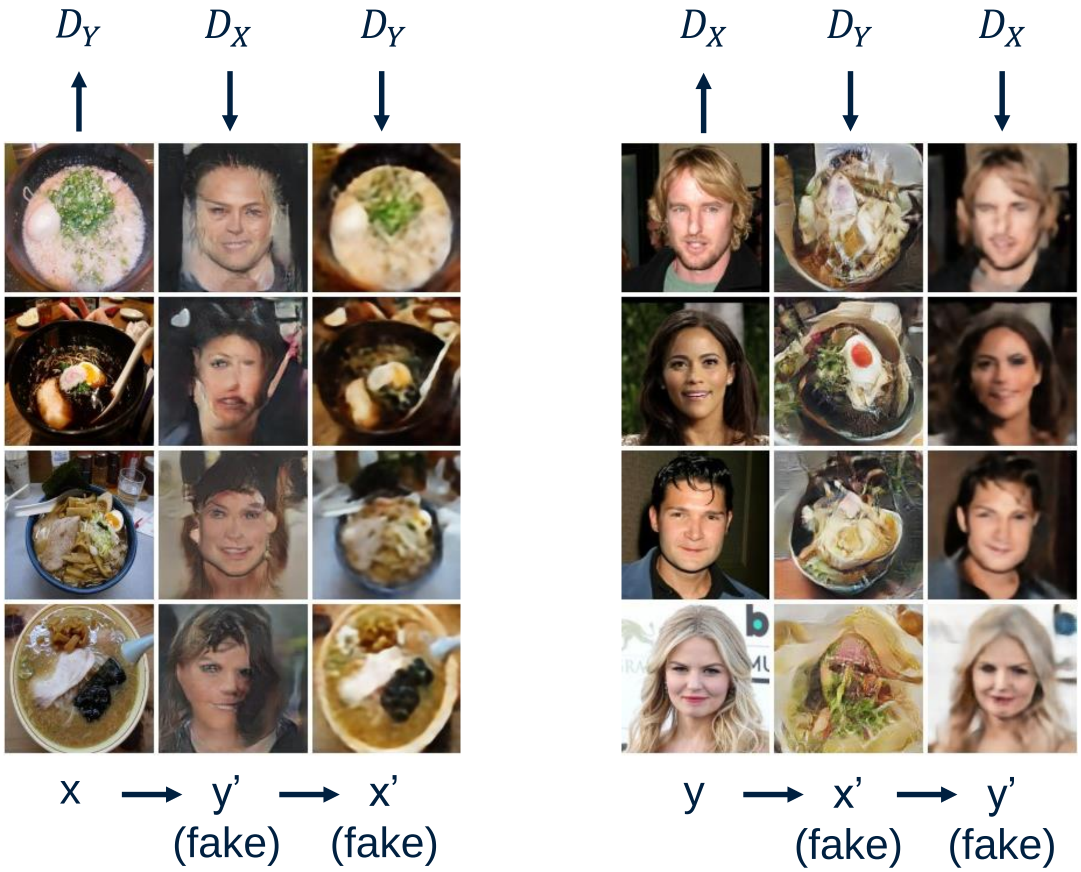


- Jointly learn to
  - map from X to Y and back to X
  - map from Y to X and back to Y

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



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

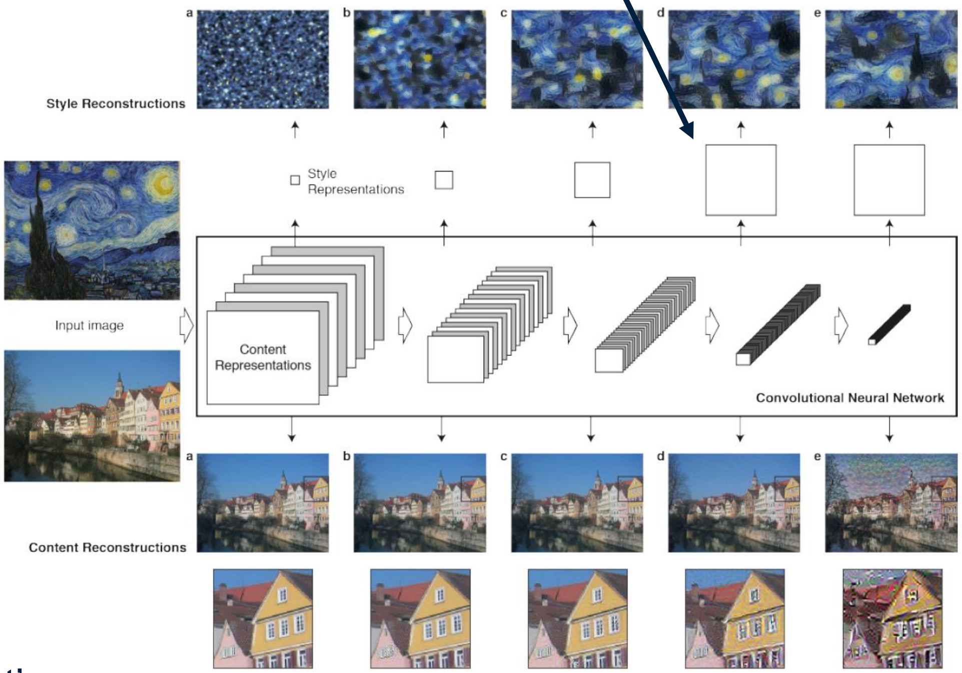
Impressive examples (StyleGAN)

Style transfer

Style influence grows with depth

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]

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

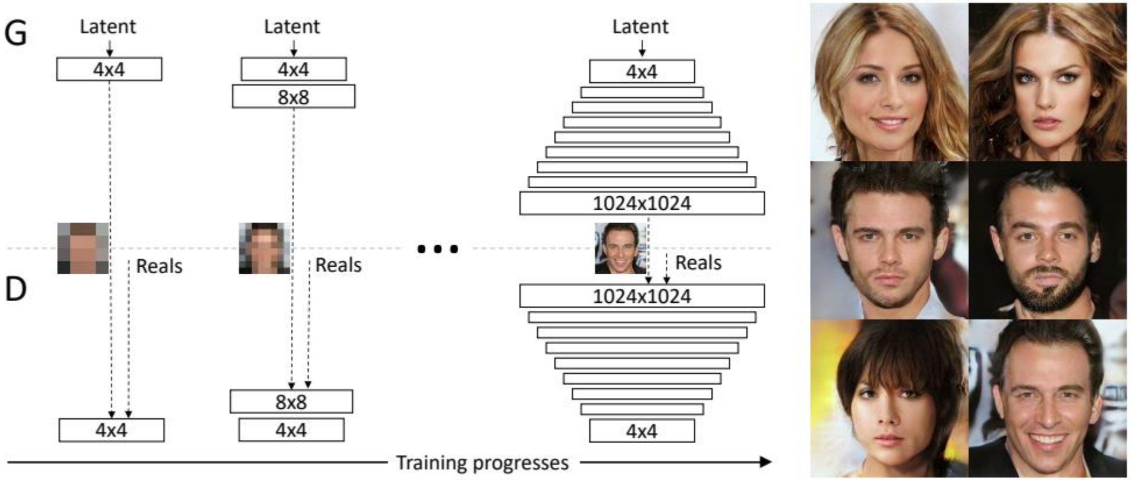
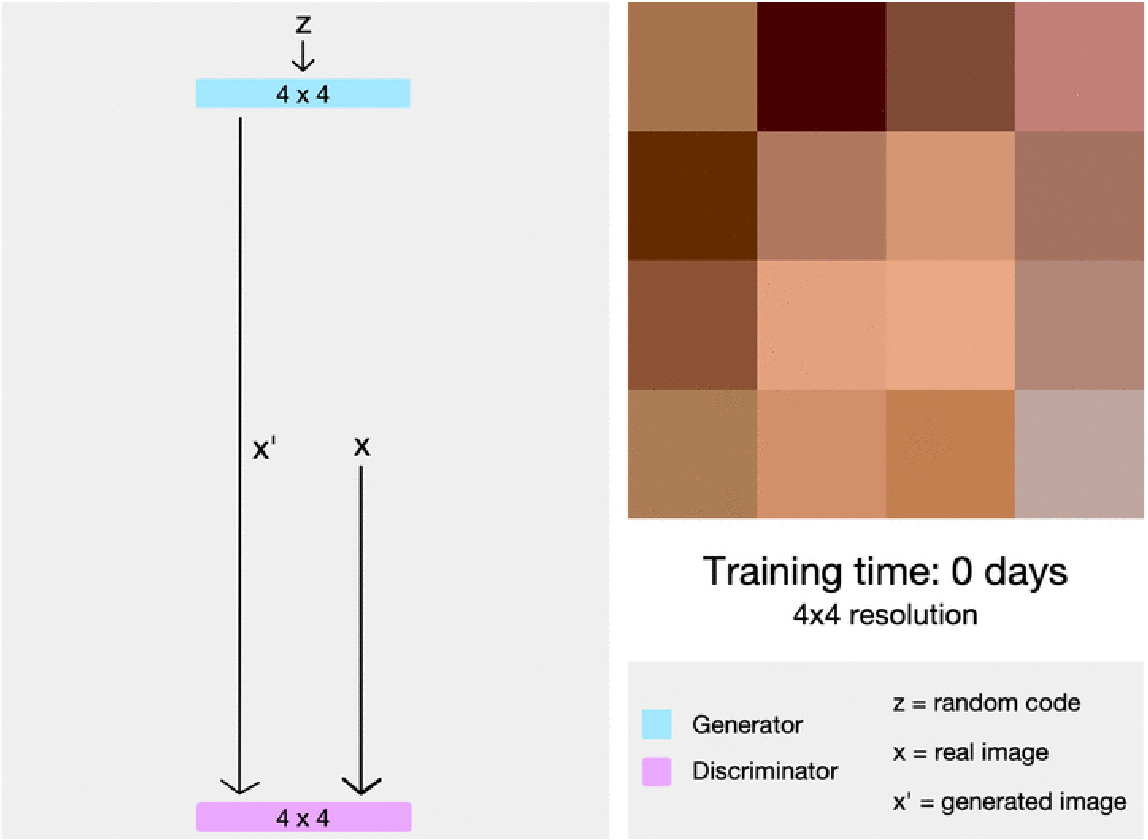


Figure 1: Our training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Here $\boxed{N \times N}$ 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



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

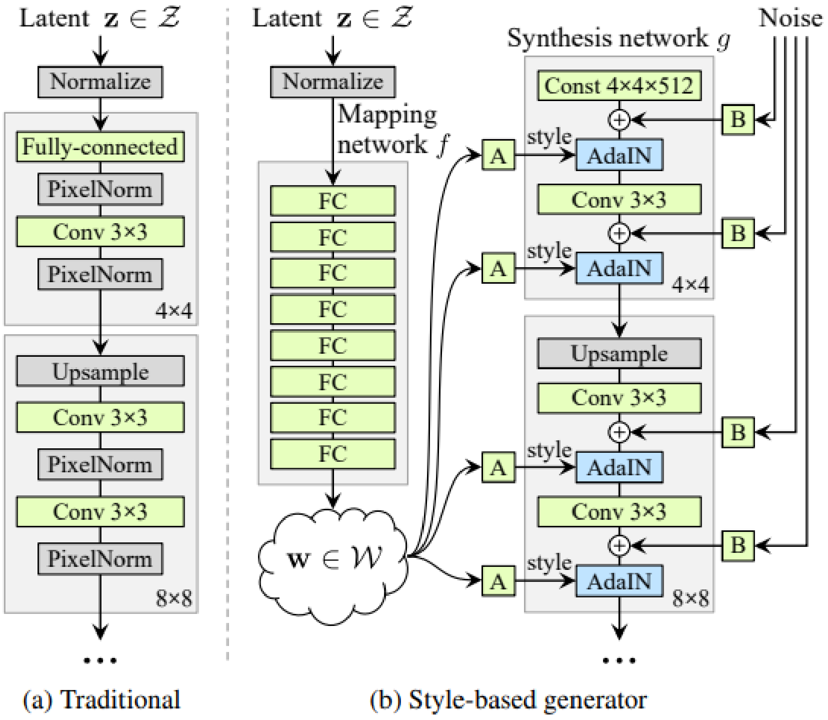
- normalize the mean and std of the target with the one of the source

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

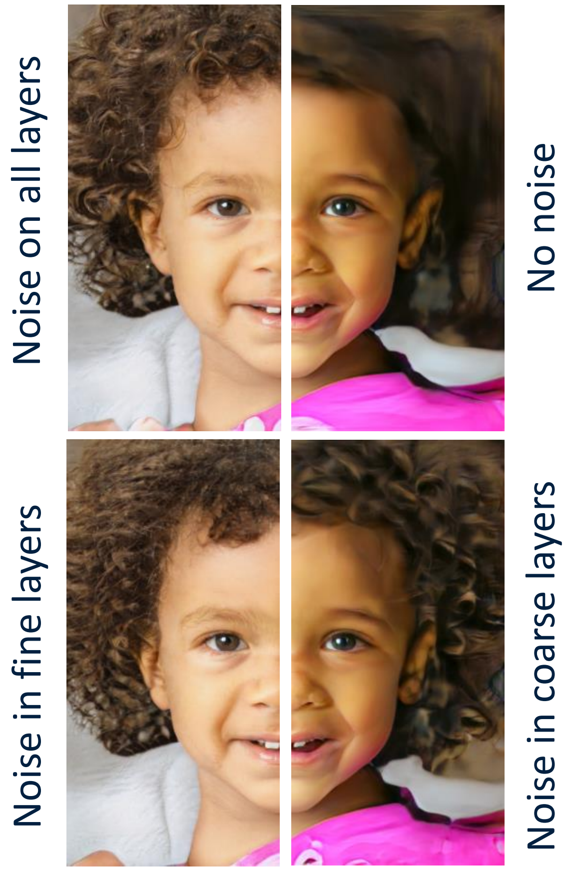
[Huang and Belongie. Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization]

Style GAN internals

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



(a) Traditional (b) Style-based generator



Additional details

Effects of style

- Coarse
  - resolution of up to 82
    - affects pose, general hair style, face shape, etc
- Middle
  - resolution of 162 to 322
    - affects finer facial features, hair style, eyes open/closed, etc.
- Fine
  - resolution of 642 to 1024
    - affects color scheme (eye, hair and skin) and micro features

- Can only generate images and mix generated images/latent codes
  - can not reconstruct (would require encoder)
  - no fine-grained control of individual features

Style GAN results