GANs

CS535

Daniel G. Aliaga

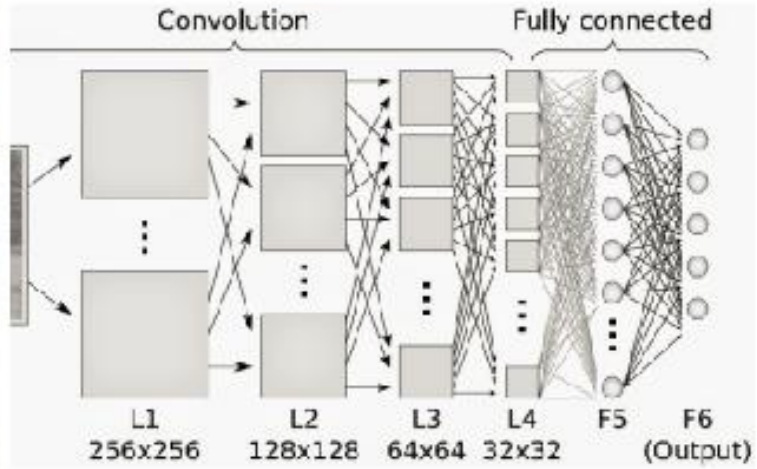
Generative Adversarial Nets

Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozairy, Courville, Bengio

- A two player min-max game to generate data hopefully indistinguishable from real data

what people think he's referring to

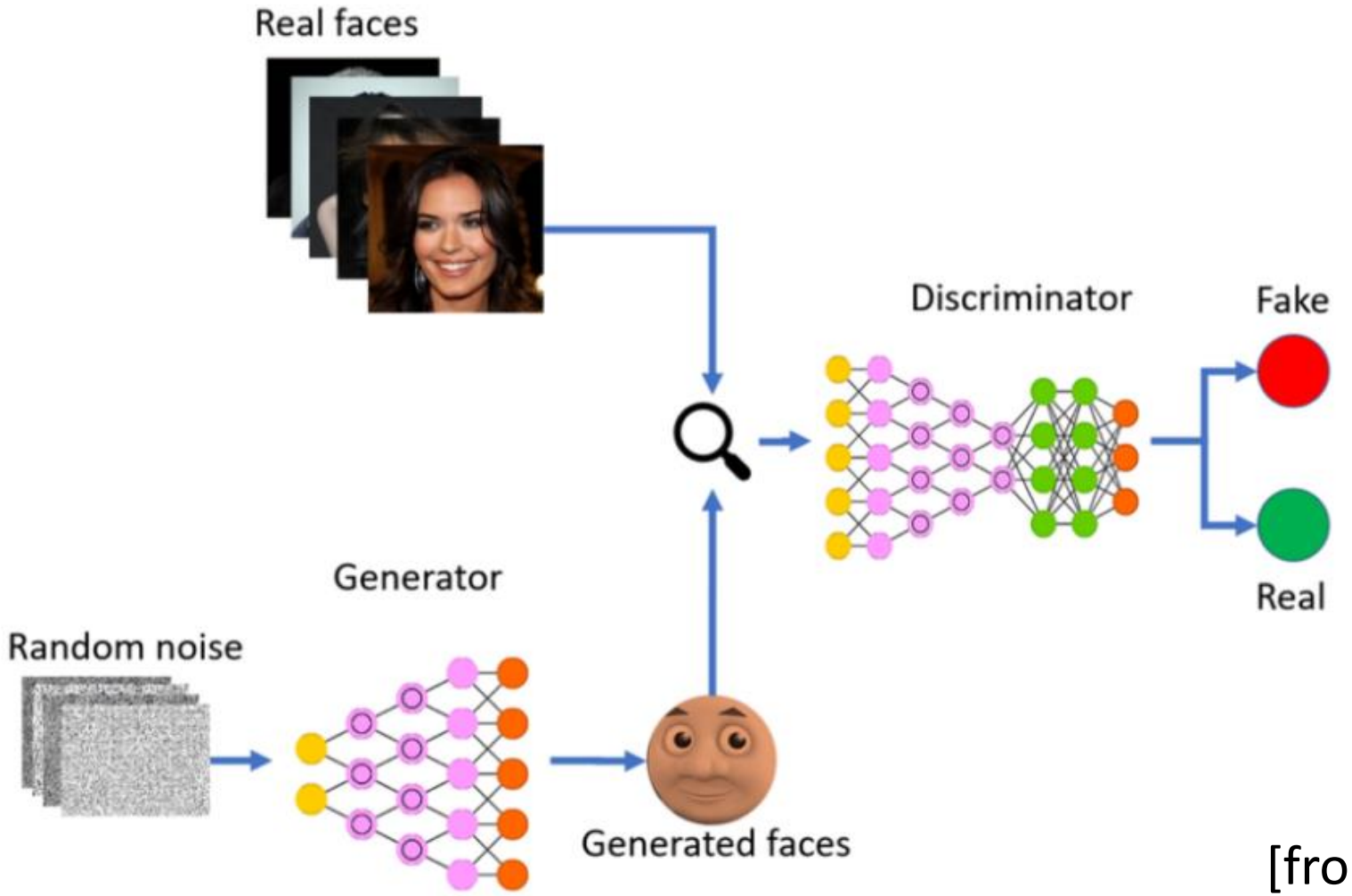
what he's actually referring to



Generative Adversarial Nets

- “We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake.”

Generative Adversarial Nets



[from medium.com]

Generative Adversarial Nets

- G(z): generator, where is z is from p(z)
- D(x): discriminator, where x is from p(x)
- Solve

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

by performing k steps of improving D and then 1 step of improving G

1. Why k steps for D and one for G?
2. What about at beginning? (when G and D are untrained)

Generative Adversarial Nets

1. Why k steps for D and one for G?

Recall that training data is provided so its distribution in a sense is defined. By keeping D nearer to its optimal (which is easier than making a G near its optimal), it helps steer creating G assuming G learns at least slowly

Converse: if D was completely unknown, then the system might start converging to some undesired solution space…

Generative Adversarial Nets

2. What about at beginning? (when G and D are untrained)

Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, log(1 - D(G(z))) saturates.

Rather than training G to minimize log(1 - D(G(z))) we can train G to maximize log D(G(z)).

Later on, revert log(1 - D(G(z))) …

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:

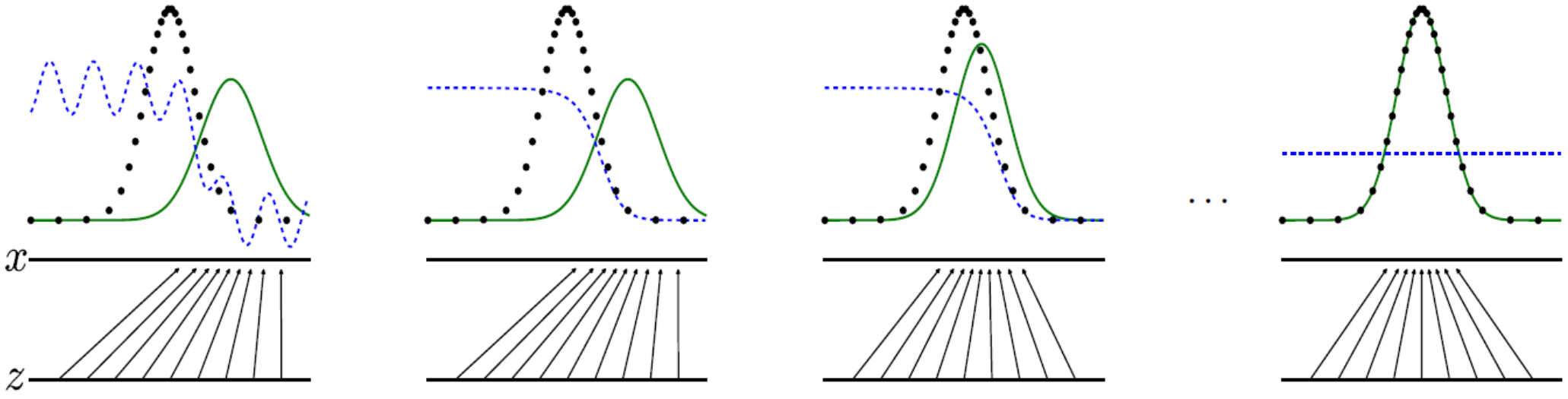
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Generative Adversarial Nets

- Training…
- D(blue), p_{data} (black), p_g (green)



"25%" training…

"50%" training…

Trained!

What is optimal D?

- When real cannot be distinguished from fake
- Value?
  - 0.5
- Why?
- $D = \frac{p_{data}(x)}{p_{data(x)} + p_{data(x)}} = 0.5$

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

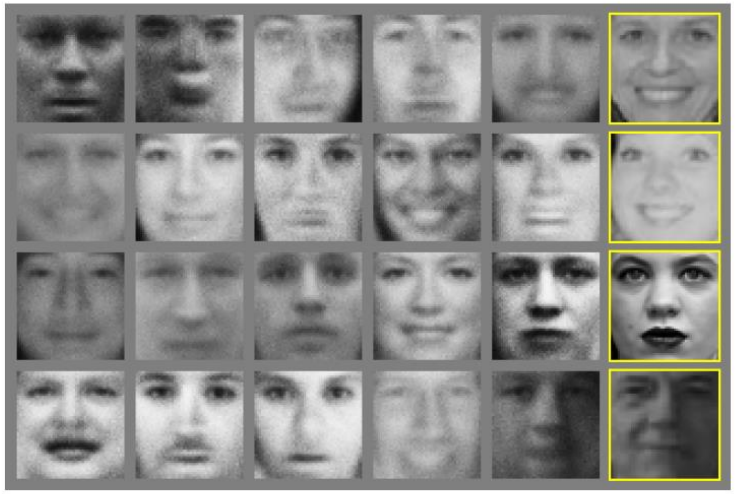
What is optimal value of the minimax?

- C(G) $= \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right] + \mathbb{E}_{\boldsymbol{x} \sim p_g} \left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})} \right]$

- C(G) = log 1/2 + log 1/2 = - log 4

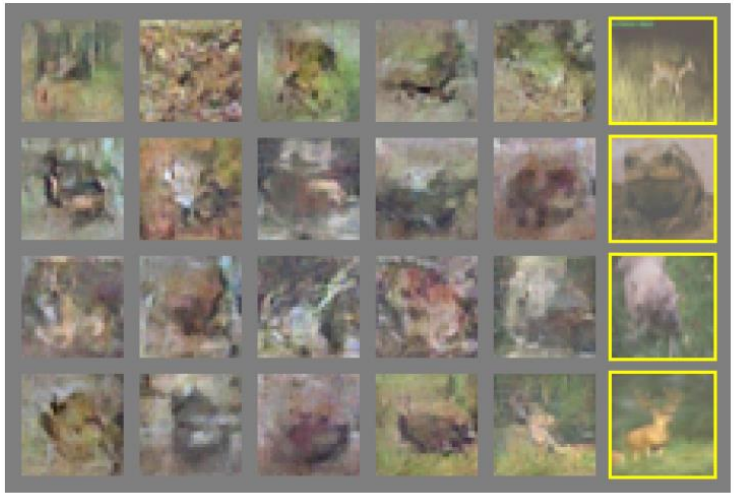
Examples (2014)

a)



b)

c)



d)

Wasserstein GAN

Arjovsky, Chintala, Bottou

- Minimizes an approximation of the Earth-Mover's distance (EM) rather than the Jensen-Shannon divergence as in the original GAN formulation

- It leads to more stable training than original GANs with less evidence of mode collapse

How do you condition/control the generation based on an input?

- Random noise (input noise vector)?
  - Hard to predict output…
- Another option?
  - Condition based on paired data during training…

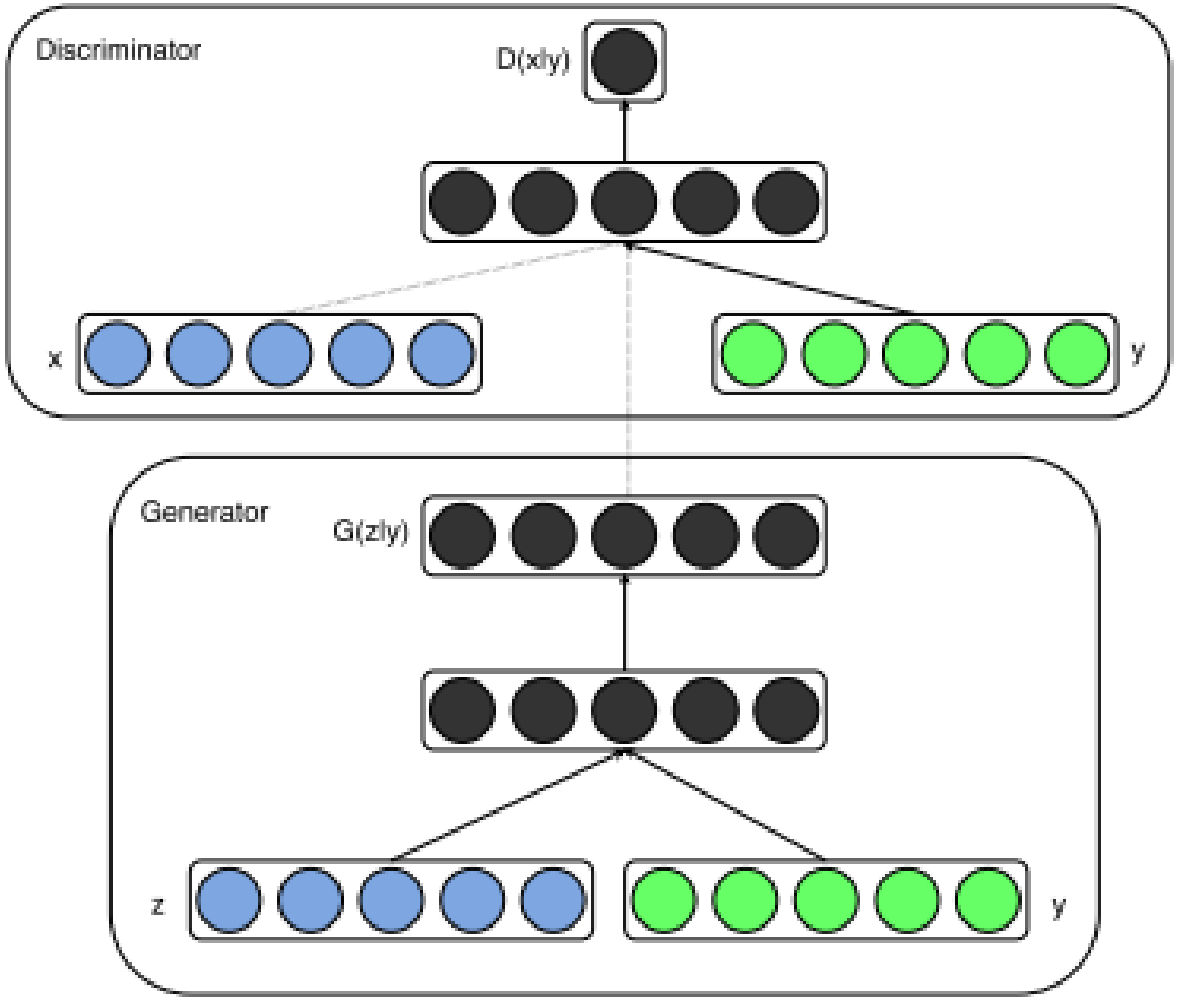
Conditional GANs

- Condition is “y”

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_y[\log D(y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x, z))].$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x, z)\|_1].$$

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G).$$



pix2pix: https://affinelayer.com/pixsrv

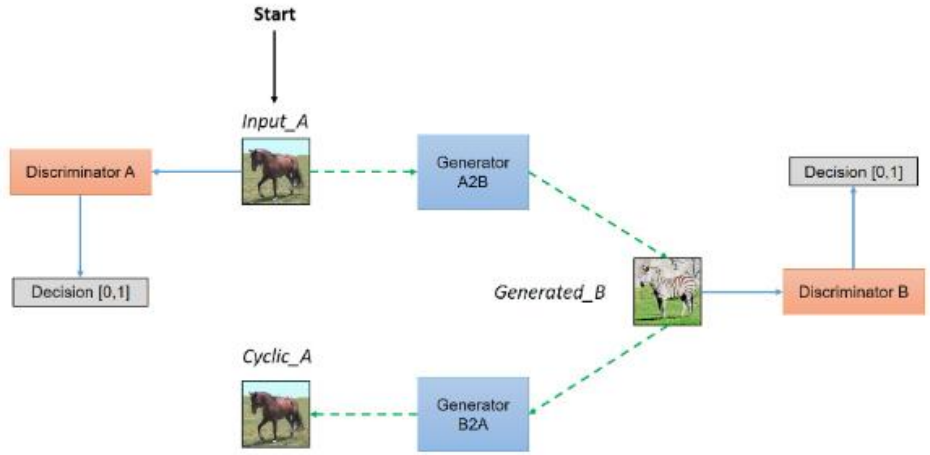
- *Semantic labels↔photo*, trained on the Cityscapes dataset [12].
- *Architectural labels→photo*, trained on CMP Facades [45].
- *Map↔aerial photo*, trained on data scraped from Google Maps.
- *BW→color photos*, trained on [51].
- *Edges→photo*, trained on data from [65] and [60]; binary edges generated using the HED edge detector [58] plus postprocessing.
- *Sketch→photo*: tests edges→photo models on human-drawn sketches from [19].
- *Day→night*, trained on [33].
- *Thermal→color photos*, trained on data from [27].
- *Photo with missing pixels→inpainted photo*, trained on Paris StreetView from [14].

How to control without paired data?

- We want to take an image from an input domain D_i and then transform it into an image of target domain D_t without necessarily having a one-to-one mapping between images from input to target domain in the training set

How to control without paired data?

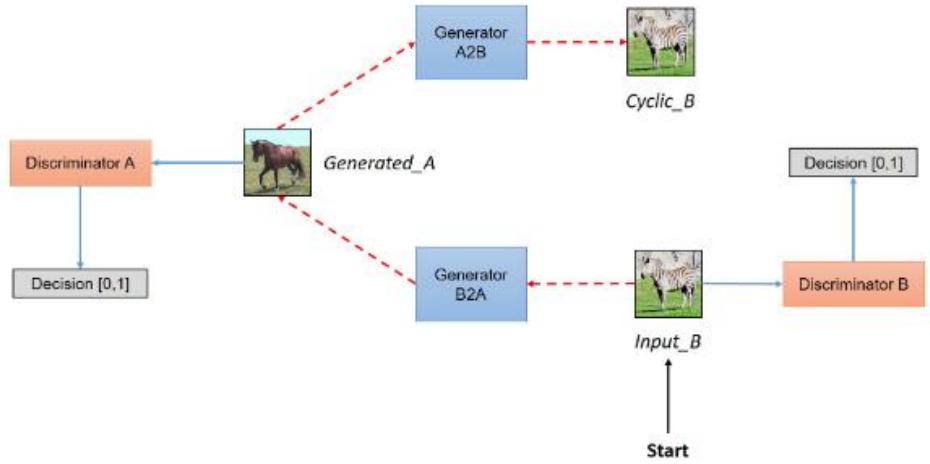
- CycleGAN: Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks



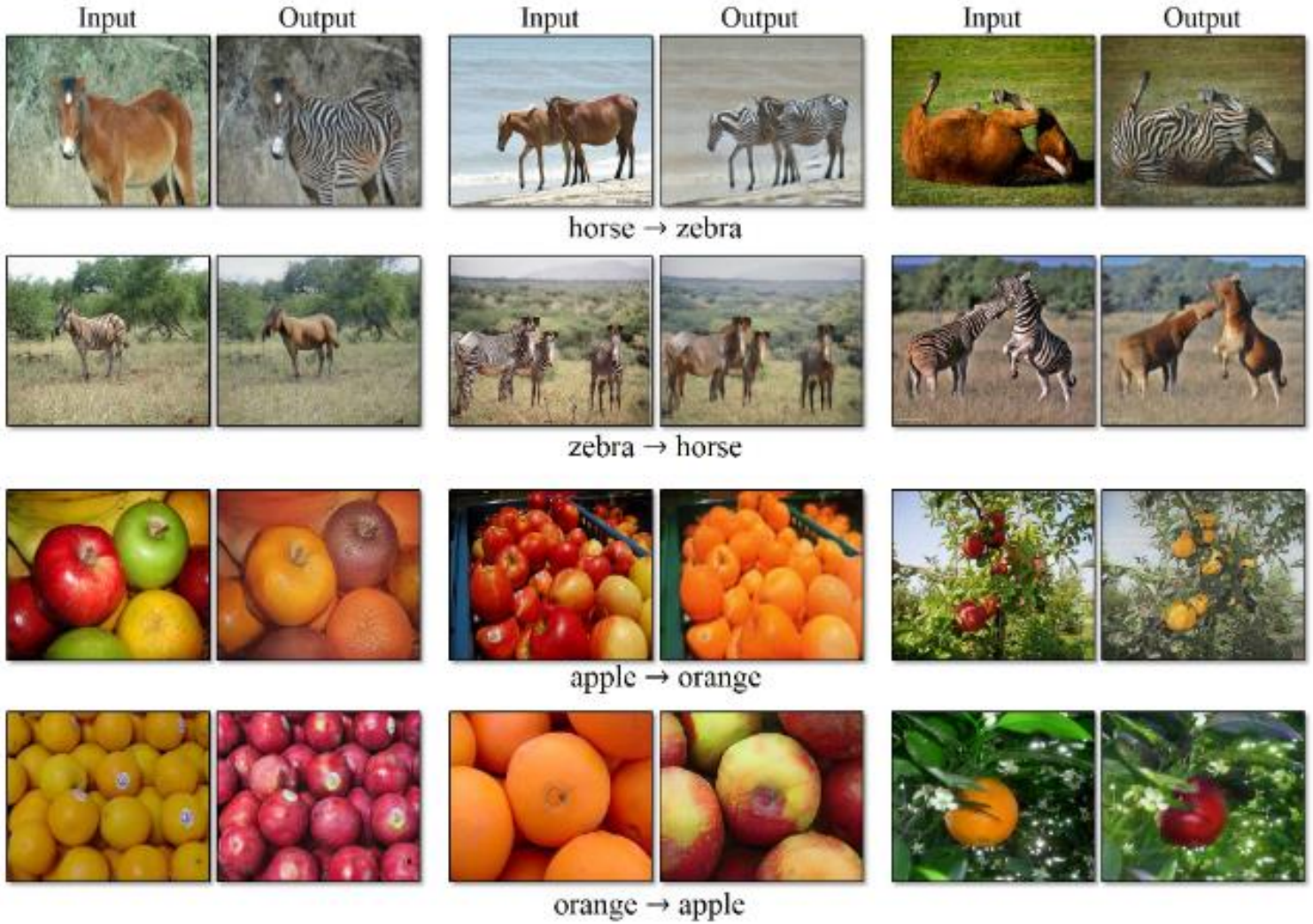
$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

where

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

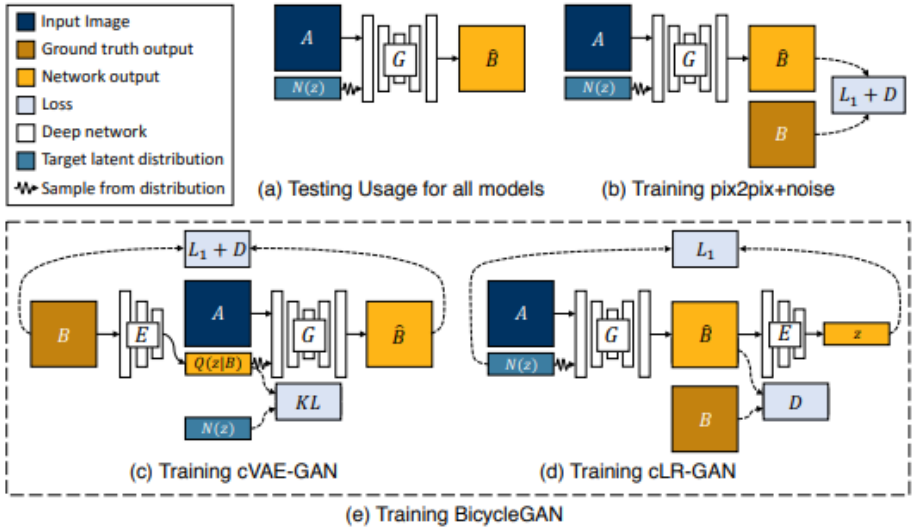


CycleGAN

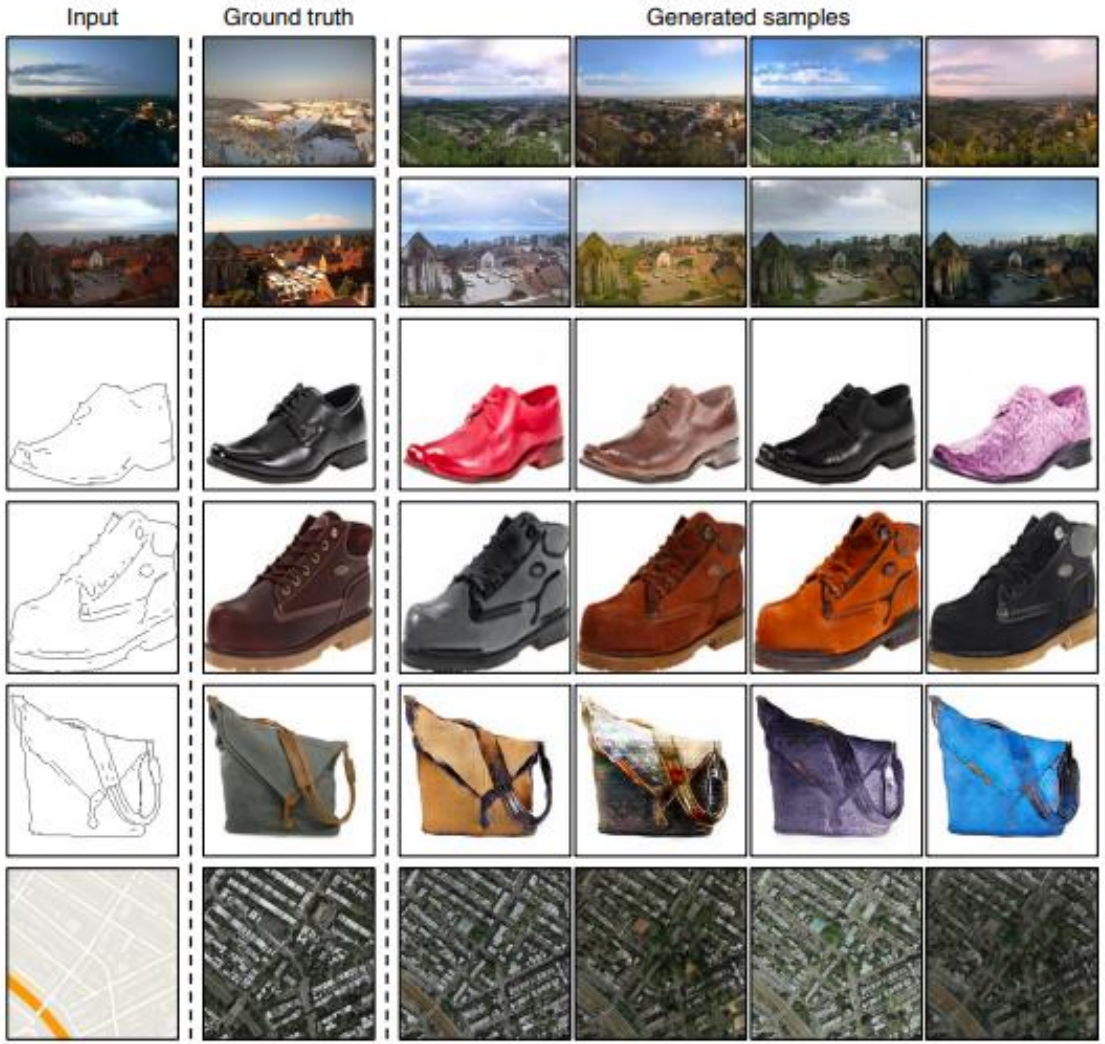

Input
Output
Input
Output
Input
Output
horse → zebra
zebra → horse
apple → orange
orange → apple

BicycleGAN: Toward Multimodal Image-to-Image Translation

- Like cGAN (i.e., pix2pix) but seek to generate realistic random variations for a single provided condition
  - Still uses paired data…
- Baseline : pix2pix condition + noise vector
  - Does not work too well
- BicycleGAN:



BicycleGAN: Toward Multimodal Image-to-Image Translation

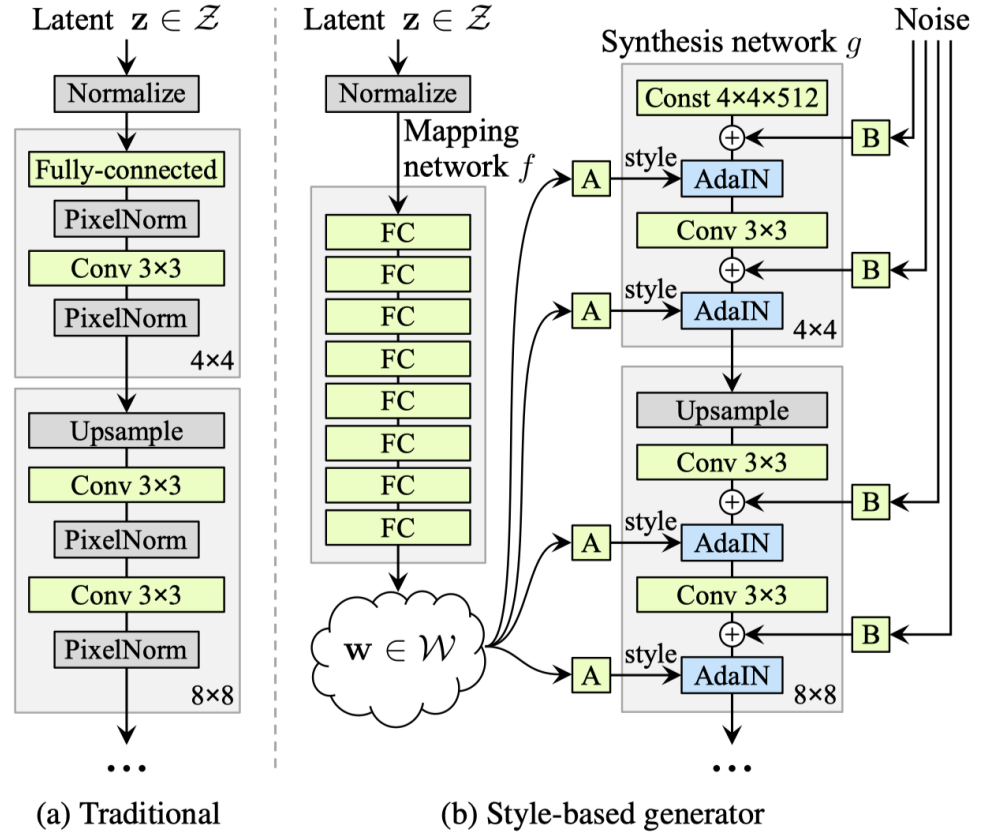


StyleGANs: What are they?

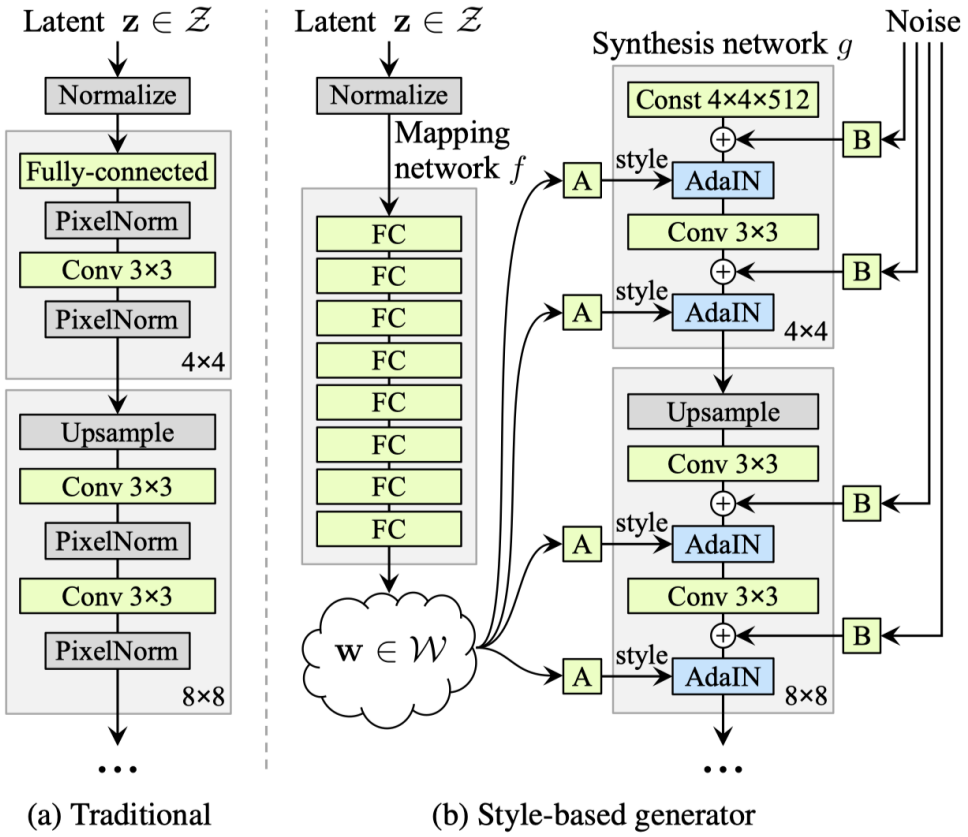
- Demo:
  - artbreeder.com
- Extension to the GAN architecture with proposed large changes including
  - use of a mapping network to map points in latent space to an intermediate latent space
  - use of the intermediate latent space to control style at each point in the generator model
  - introduction to noise as a source of variation at each point in the generator model
- The discriminator or the loss function is not modified in the original implementation of StyleGan so the same discussion about GAN loss functions, regularization, and hyperparameters can be applied here as well

Style-based generator

- Typically latent code for a generator is given as a Input layer (first layer of feed-forward network)
- Instead we switch to a learned constant
  - Latent code: $z \in Z$
  - Non-linear mapping: $f: Z \to W$ produces $w \in W$ – implemented using a 8-layer MLP
  - Dimensionality of both spaces to 512
- Learned affine transformations specialize w to styles $y = (y_s, y_b)$ which is used to control adaptive instance normalization (AdaIN) operations after each convolution layer of the synthesis network g
  - $AdaIN(x_i, y) = y_{s,i} \frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$
  - AdaIN operation takes a feature map x_i, which is normalized separately, and then scaled and biased using corresponding scalar components from style y
  - That means dimensionality of y is twice the number of feature maps on that layer



[Karras et. al., 2019]



(a) Traditional (b) Style-based generator

- "A" stands for a learned affine transform
- "B" applies learned per-channel scaling factors to the noise input.
- The mapping network f consists of 8 layers
- Synthesis network g consists of 18 layers — two for each resolution ($4^2 - 1024^2$).
- The output of the last layer is converted to RGB using a separate 1 × 1 convolution
- generator has a total of 26.2M trainable parameters, compared to 23.1M in the traditional generator
- W controls the generator through AdaIN
- Gaussian noise is added after each convolution

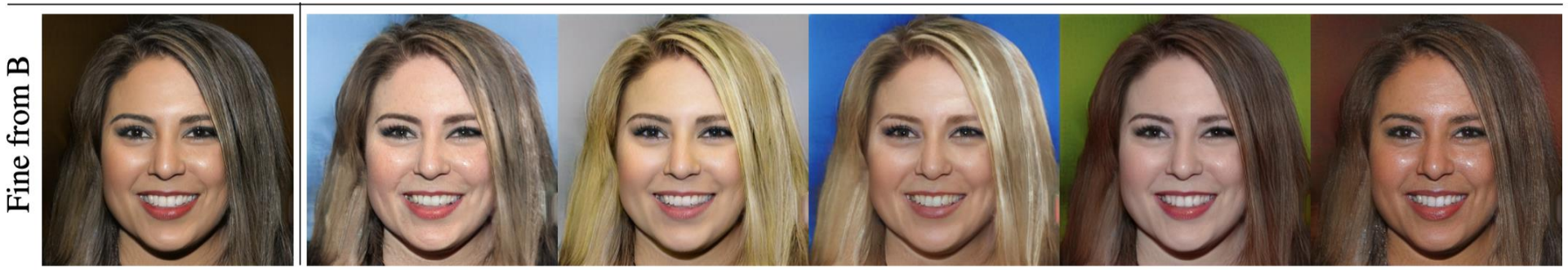
Style-based generator *(cont.)*

- The generator is provided a a way to generate stochastic details by introducing explicit noise inputs
- These are single-channel images that are of uncorrelated Gaussian noise
- The noise is broadcasted to all feature maps using learned per-feature scaling factors and then added to output of the corresponding convolution

[Karras et. al., 2019]

Style Mixing

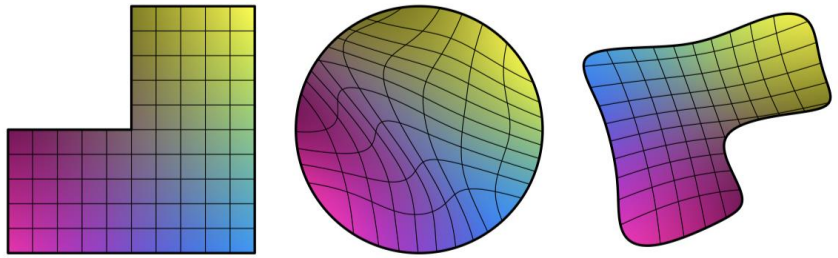
- *Mixing regularization:* encourages style to localize.
  - given percentage of images are generated using two random latent codes instead of one during training
  - generating such an image simply requires from one latent code to another
- Two latent codes z_1, z_2 runs though the mapping network and have corresponding w_1, w_2 control the style so that w_1 applies before the crossover point and w_2 after it.
- This technique prevents the network from assuming that adjacent styles are correlated



[Karras et. al., 2019]

Stochastic Variation

- In human portraits, the following can be considered stochastic
  - Exact placement of hairs
  - Stubble
  - Freckles
  - Skin pores
- Noise tends to only affect the stochastic aspects of the generation process.



(a) Distribution of features in training set

(b) Mapping from $\mathcal{Z}$ to features

(c) Mapping from $\mathcal{W}$ to features

(a) An example training set where some combination (e.g., long haired males) is missing. (b) This forces the mapping from Z to image features to become curved so that the forbidden combination disappears in Z to prevent the sampling of invalid combinations. (c) The learned mapping from Z to W is able to “undo” much of the warping.

[Karras et. al., 2019]

(a) Generated image (b) Stochastic variation (c) Standard deviation

Examples of stochastic variation.

(a) Two generated images.

(b) Zoom-in with different realizations of input noise. While the overall appearance is almost identical, individual hairs are placed very differently

(c) Standard deviation of each pixel over 100 different realizations, highlighting which parts of the images are affected by the noise. The main areas are the hair, silhouettes, and parts of background, but there is also interesting stochastic variation in the eye reflections.

Global aspects such as identity and pose are unaffected by stochastic variation.

[Karras et. al., 2019]

Can you disentangle the latent vector?

- How?

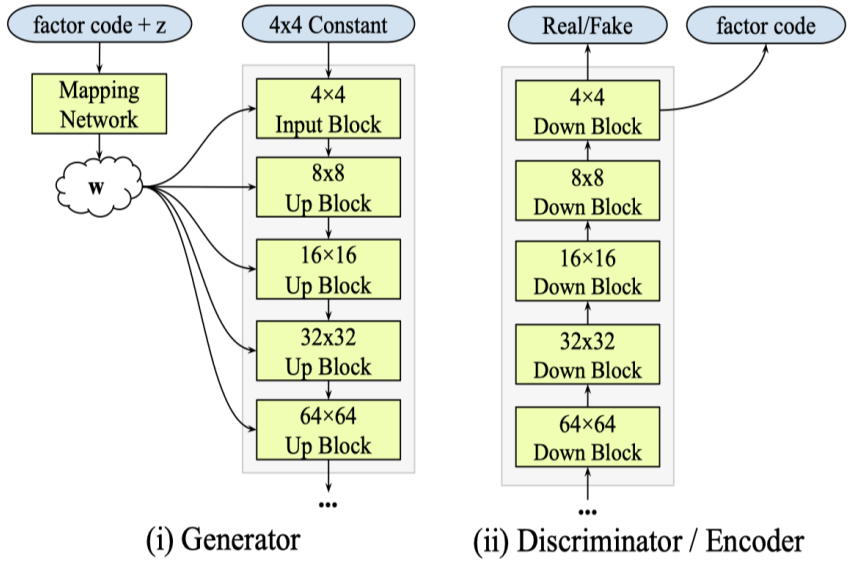
Semi-Supervised StyleGan

- Disentanglement learning is crucial for controllable generation, where the control or style factors specified to the generator need to be disentangled for faithful representation
- Problem with current disentangled methods
  - Non-identifiability
    - Hard acquiring large number of fully annotated samples
    - Multiple runs will not observe the same latent representations
  - Human feedback is required to discern
    - factors the model has learnt
    - Semantic meaning different values of discovered factor code represent
- We need a reliable way to control generation for practical use
- Adding a small amount of labeled data may resolve non-identifiability and lead to interpretable factors.

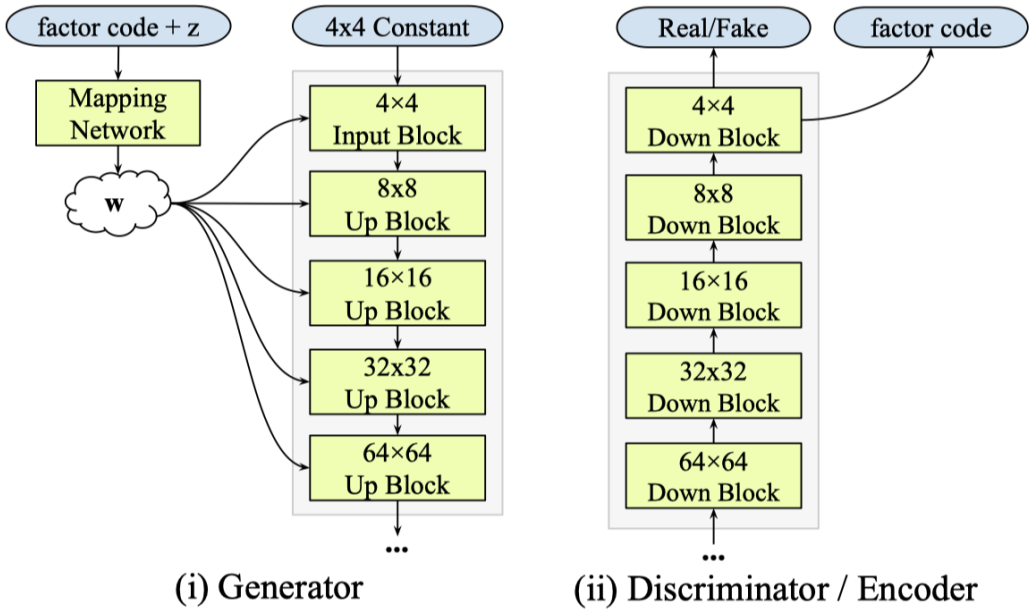
[Nie et. al., 2020]

Unsupervised Disentanglement Learnin

- Using Info-StyleGan, enabling StyleGAN with mutual information loss, provides a stronger prior for disentanglement compared to regularization used in Variational Autoencoders or GANs
- Mapping network in the generator of Info-Style GAN now conditions on a *factor code,* a vector representing each factor of variation in each dimension, by simply concatenating it with the latent code z.
- The output of the mapping network is called *conditional styles.* This will modulate each block in the synthesis network using AdaIN.



[Nie et. al., 2020]



Mapping network in the generator conditions on the factor code and the encoder, which shares all layers in the discriminator except for the last layer, predicts its value.

Unsupervised code reconstruction loss

- *unsupervised code reconstruction* loss - the mutual information loss of InfoGAN can approximately be an
  - $L_{unsup} = \sum_{c\sim C, z\sim p_z} \| E\big(G(c,z)\big) - c \|_2$

  Where C denotes the set of all factor codes and p_z denotes the prior distribution of latent code z. G represents the generator, E the encoder, D the discriminator, and γ is a hyperparameter that controls tradeoff between image realism and disentanglement quality

  - $L^G = L_{GAN} + \gamma \quad L_{unsup}$
  - $L^{(D,E)} = -L_{GAN} + \gamma \quad L_{unsup}$

  L_{GAN} is the original GAN loss function

[Nie et. al., 2020]

Semi-StyleGAN

- How do we make the disentanglement model… semi-supervised?
- Supervised code reconstruction! – for the small amount of labeled data
  - $L_{sup} = \sum_{(x,c)\sim\tau} \| E(x) - c \|_2$
    where τ represents set of label real image and factor code. The semi-supervised loss functions become

$$L^G = L_{GAN} + \gamma_G L_{unsup}$$
$$L^{(D,E)} = -L_{GAN} + \gamma_E L_{unsup} + \beta L_{\text{sup}}$$

Where β is weight of the supervised term

[Nie et. al., 2020]

A better way of doing it

- Issues with *supervised code reconstruction* things like image rotations and color randomization will cause inconsistency.
- What else can we do?
  - Mixed observation code pairs. Use this to add a smoothness regularization.
  - Given a labeled observation-code pair $(x, c) \sim \tau$ and a generated pair (x′,c′) where $x' = G(z, c')$, we get a set of mixed observation-code pairs $M = \{(\tilde{x}, \tilde{c})\}$ by

$$\lambda \sim Beta(\xi, \xi), \lambda' = \max(\lambda, 1 - \lambda)$$
$$\tilde{x} = \lambda' x + (1 - \lambda') x'$$
$$\tilde{c} = \lambda' c + (1 - \lambda') c'$$

where ξ is a hyperparameter

[Nie et. al., 2020]

Encouraging smoothness in the latent space of GANs

$$L_{sr} = \sum_{(x,c)\sim M} \| E(x) - c \|_2$$

We can use this to update our previous loss terms!

$$L^G = L_{GAN} + \gamma_G L_{unsup} + \alpha L_{sr}$$

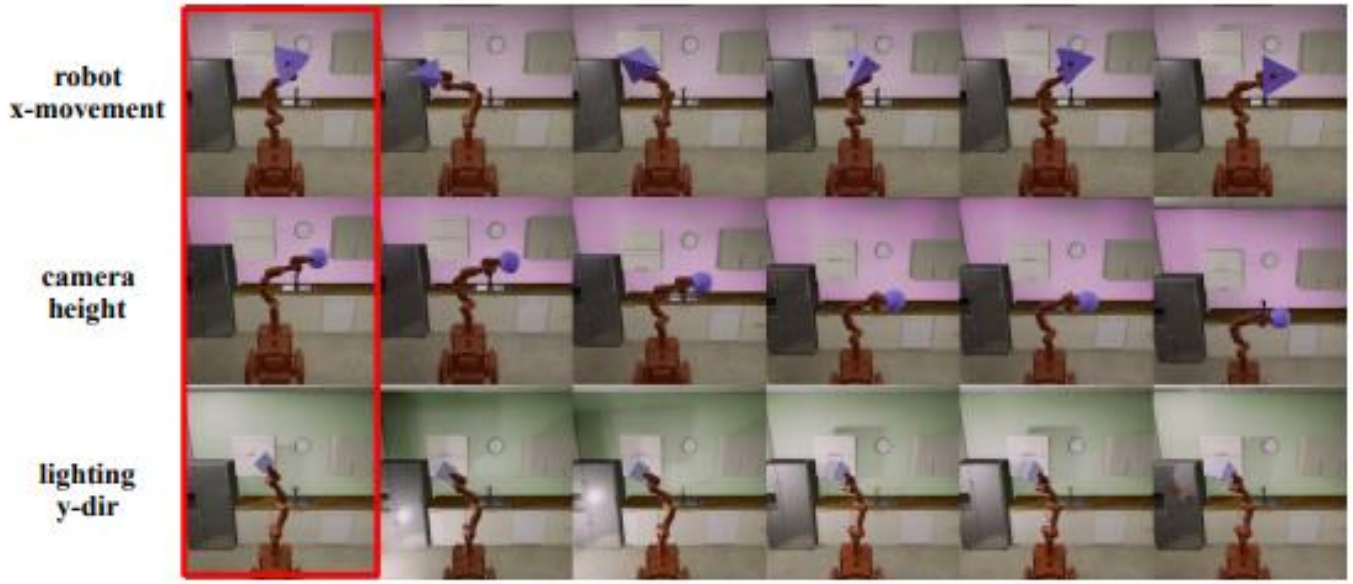
. $$L^{(D,E)} = -L_{GAN} + \gamma_E L_{unsup} + \beta L_{\text{sup}} + \alpha L_{sr}$$

Where α is the weight of the smoothness term

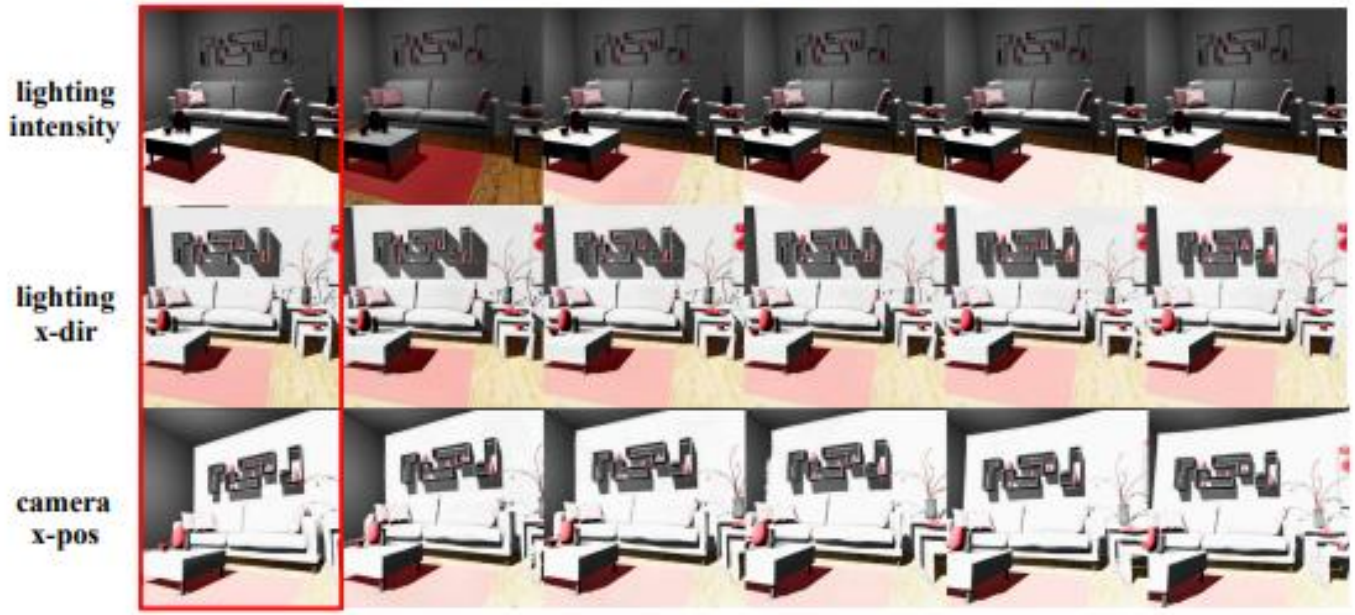
This way, it not only encourages smooth behaviors of both the generator and encoder, but also takes good advantages of enormous fake data for disentanglement.

[Nie et. al., 2020]

Figure 4. Latent traversal of Semi-StyleGAN on CelebA with resolution 256x256 by using 0.5% of the labeled data, where we use $\gamma = 1$ and disentangle all 40 binary attributes. See Appendix C.3 for the results of other attributes.



(a) Semi-StyleGAN on Isaac3D with 0.5% of labeled data



(b) Semi-StyleGAN on Falcor3D with 1% of labeled data

New thoughts: Implicit competitive regularization in GANs