

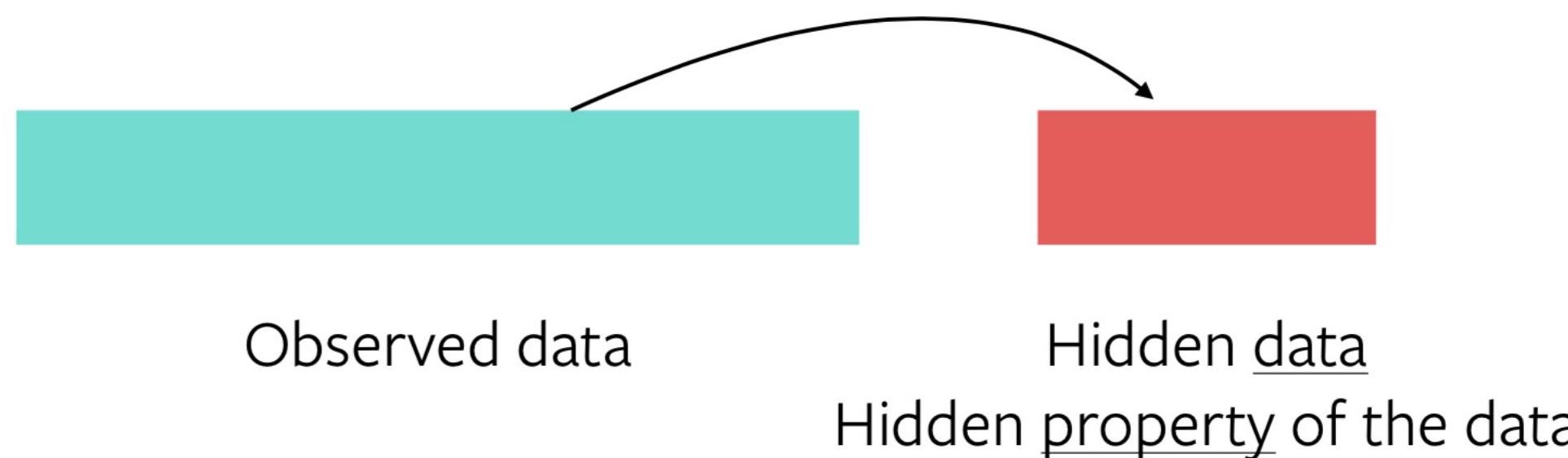
# Overview of Unsupervised Learning & Generative Adversarial Networks

Lecture 11

Slides from: Emily Denton, Ian Goodfellow, Soumith Chintala

# Modeling Data

- Recall “self-supervised” learning
- Predict some part of the data from another part



- Modeling  $p(x_{\text{hidden}} | x_{\text{observed}})$
- Use learned representations for down-stream task (e.g. classification)
- Focus today on **generating novel examples** from underlying distribution  $p(x)$

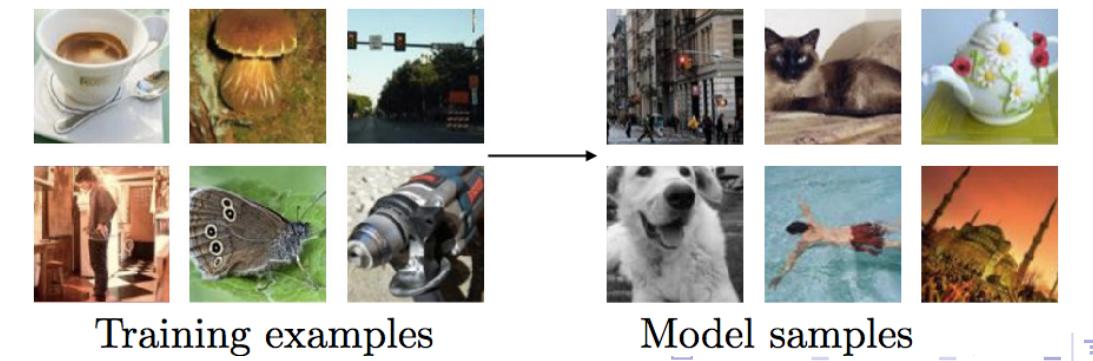
# Density Modeling

- Have access to  $x \sim p_{data}(x)$  through training set
- Want to learn a model  $x \sim p_{model}(x)$
- Want  $p_{model}$  to be similar to  $p_{data}$ :

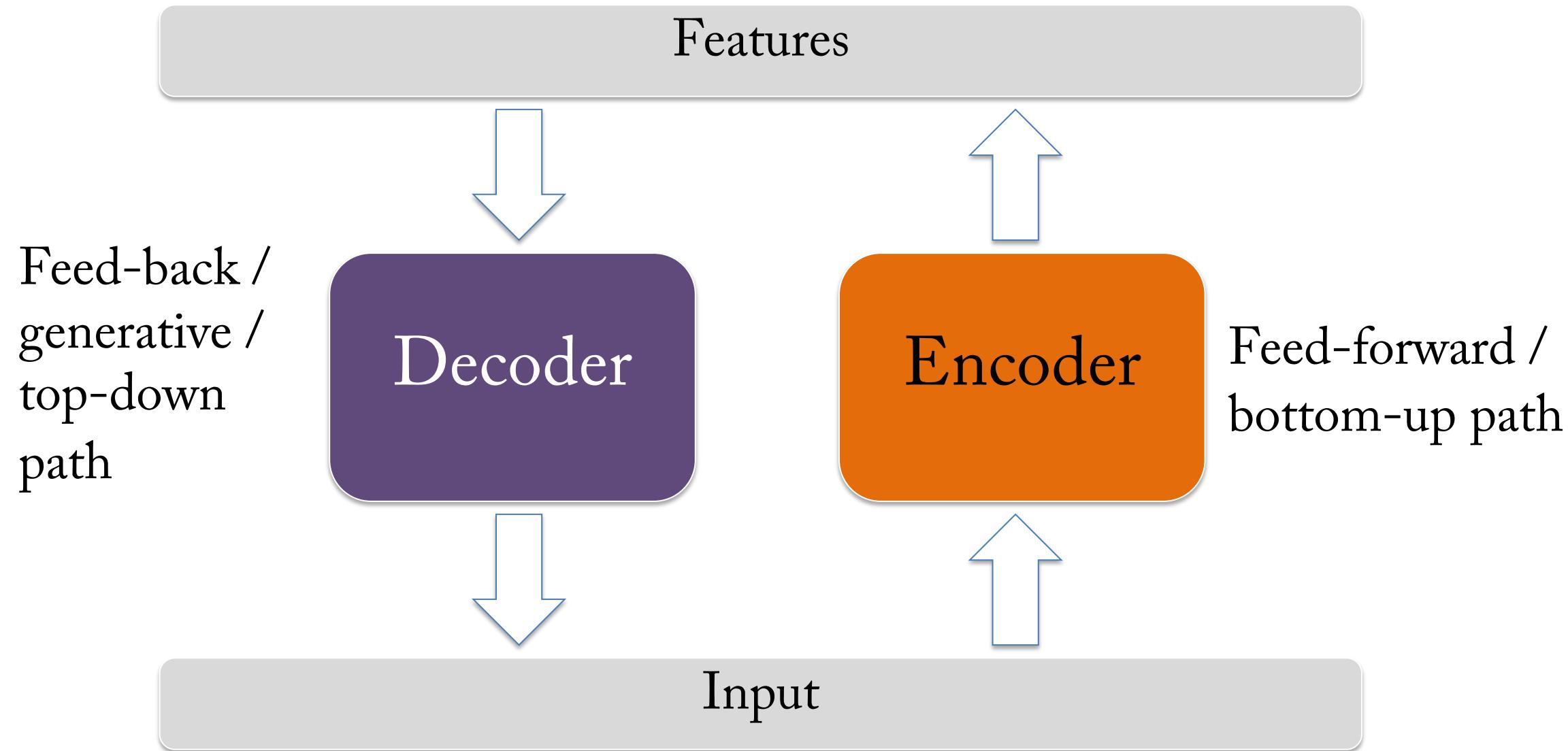
Samples from true data distribution have high likelihood under  $p_{model}$



Samples drawn from  $p_{model}$  reflect structure of  $p_{data}$



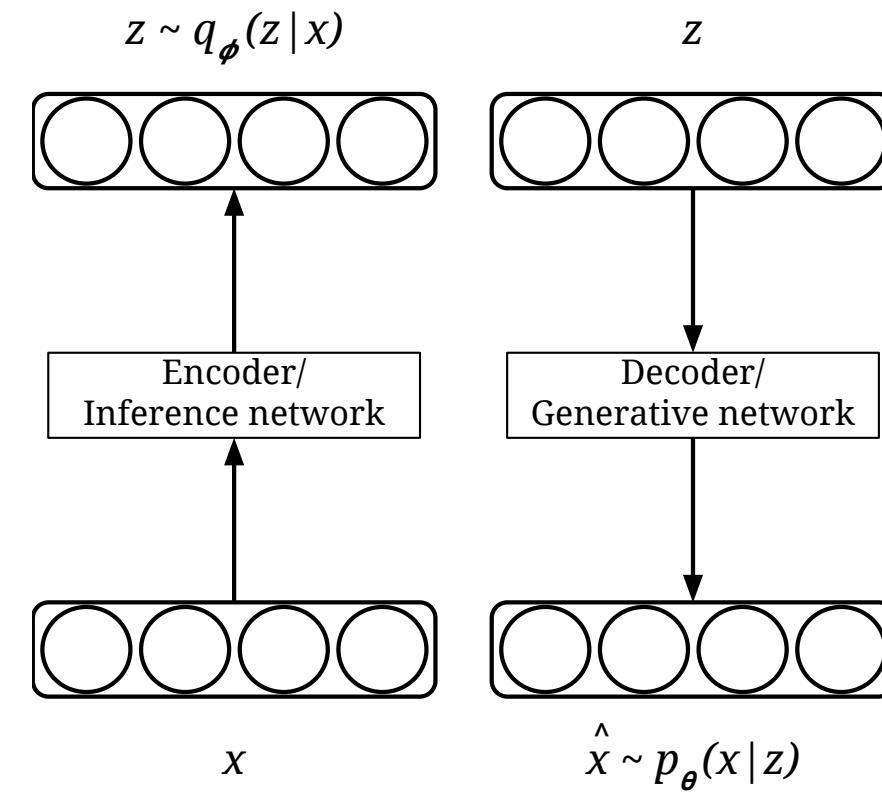
# Auto-Encoder



- Encoder/Decoder will be deep network
- Slightly different architectures for decoder (needs to output image)
- Architecture depends on application

# Variational autoencoder

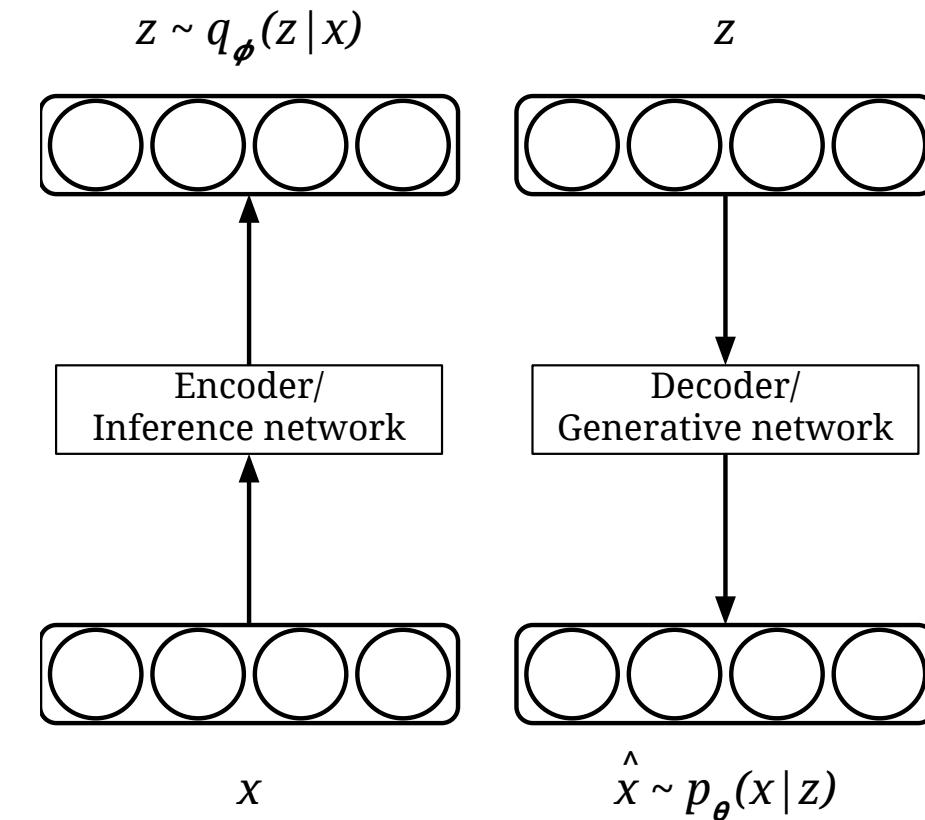
- *Encoder* network maps from image space to latent space
  - Outputs parameters of  $q_\phi(z|x)$
- *Decoder* maps from latent space back into image space
  - Outputs parameters of  $p_\theta(x|z)$



[Kingma & Welling (2013)]

## Example

- *Encoder* network outputs mean and variance of Normal distribution
  - $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$
- *Decoder* network outputs mean (and optionally variance) of Normal distribution
  - $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \mathbf{I})$



[Kingma & Welling (2013)]

## Bounding the marginal likelihood

Recall Jenson's inequality: When  $f$  is concave,  $f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$

$$\begin{aligned}\log p(x) &= \log \int_z p(x, z) \\ &= \log \int_z q(z) \frac{p(x, z)}{q(z)} \\ &\geq \int_z q(z) \log \frac{p(x, z)}{q(z)} = L(x; \theta, \phi) \quad (\text{by Jenseon's inequality})\end{aligned}$$

Evidence Lower  
BOund (ELBO)

Bound is tight when variational approximation matches true posterior:

$$\log p(x) - L(x; \theta, \phi) = \log p(x) - \int_z q(z) \log \frac{p(x, z)}{q(z)}$$

Evidence Lower

$$\begin{aligned} \text{BOund (ELBO)} &= \int_z q(z) \log p(x) - \int_z q(z) \log \frac{p(x, z)}{q(z)} \\ &= \int_z q(z) \log \frac{q(z)p(x)}{p(x, z)} \\ &= \int_z q(z) \log \frac{q(z)}{p(z|x)} \\ &= D_{KL}(q(z; \phi) || p(z|x)) \end{aligned}$$

# Variational autoencoder

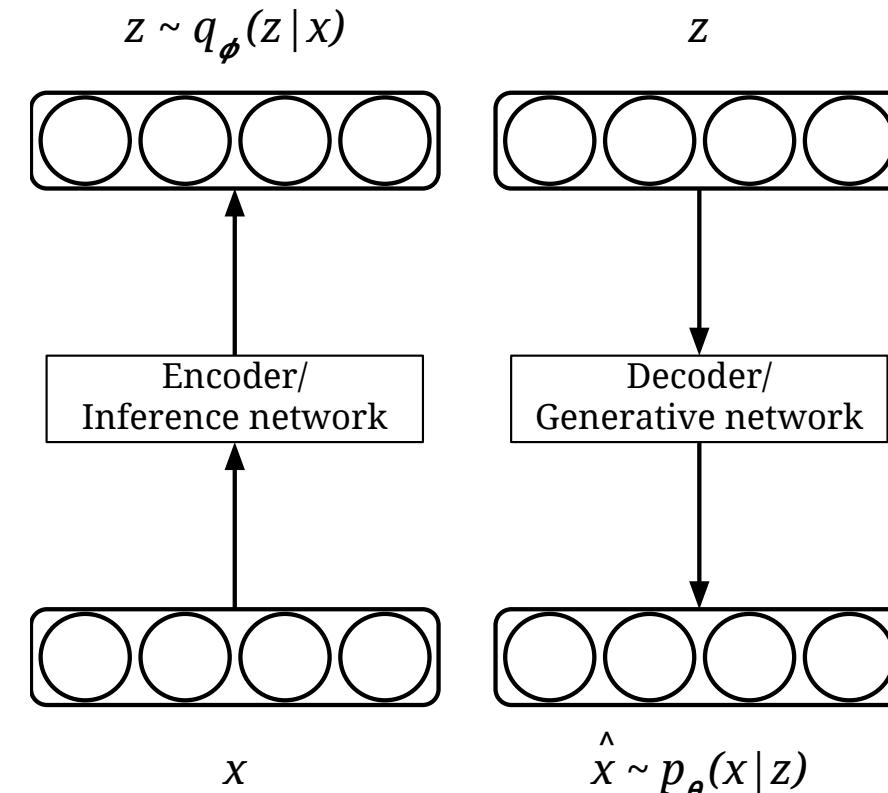
- Rearranging the ELBO:

$$\begin{aligned} L(x; \theta, \phi) &= \int_z q(z|x) \log \frac{p(x, z)}{q(z|x)} \\ &= \int_z q(z|x) \log \frac{p(x|z)p(z)}{q(z|x)} \\ &= \int_z q(z|x) \log p(x|z) + \int_z q(z|x) \log \frac{p(z)}{q(z|x)} \\ &= \mathbb{E}_{q(z|x)} \log p(x|z) - \mathbb{E}_{q(z|x)} \log \frac{q(z|x)}{p(z)} \\ &= \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}} \end{aligned}$$

# Variational autoencoder

- Inference network outputs parameters of  $q_\phi(z|x)$
- Generative network outputs parameters of  $p_\theta(x|z)$
- Optimize  $\theta$  and  $\phi$  jointly by maximizing ELBO:

$$L(x; \theta, \phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$



# Stochastic gradient variation bayes (SGVB) estimator

- Reparameterization trick : re-parameterize  $z \sim q_\phi(z|x)$  as

$$z = g_\phi(x, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

- For example, with a Gaussian can write  $z \sim \mathcal{N}(\mu, \sigma^2)$  as

$$z = \mu + \epsilon\sigma^2 \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

[Kingma & Welling (2013); Rezende *et al.* (2014)]

## Stochastic gradient variation bayes (SGVB) estimator

$$L(x; \theta, \phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

- Using reparameterization trick we form Monte Carlo estimate of reconstruction term:

$$\begin{aligned} \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) &= \mathbb{E}_{p(\epsilon)} \log p_\theta(x|g_\phi(x, \epsilon)) \\ &\simeq \frac{1}{L} \sum_{i=1}^L \log p_\theta(x|g_\phi(x, \epsilon)) \quad \text{where } \epsilon \sim p(\epsilon) \end{aligned}$$

- KL divergence term can often be computed analytically  
(eg. Gaussian)

## VAE learned manifold



6666660000000000000000  
44422222200000000000002  
442222222285555000000002  
44222222223333333555550000002  
4442222223333333333555555533  
4499992222333333333355555533  
44999993333333333355555533  
4499999933333333333355555533  
7999999933333333333388888887  
7999999988888888888888888887  
7999999988888888888888888887  
7999999988888888888866666557  
79999999988888886666666557  
799999999999999966666666557  
799999999999999966666666557  
79999999999999996666666666661  
79999999999999996666666666661  
79999999999999996666666666661  
79999999999999996666666666661

[Kingma & Welling (2013)]

# VAE samples

The figure displays a 10x10 grid of handwritten digits, each generated from a different dimension of a latent space. The digits are arranged in four rows of five, labeled (a) through (d). Each row corresponds to a different latent dimension: (a) 2-D, (b) 5-D, (c) 10-D, and (d) 20-D. The digits transition from being highly noisy and distorted at lower dimensions to appearing increasingly sharp and recognizable at higher dimensions.

|                     |                     |                     |                     |
|---------------------|---------------------|---------------------|---------------------|
| 8 5 / 7 8 / 4 8 2 8 | 5 1 6 5 1 6 7 6 7 + | 1 8 7 1 3 8 5 3 3 8 | 7 2 0 5 9 2 3 9 0 0 |
| 9 6 8 3 9 6 6 3 1 9 | 8 5 8 4 5 8 2 1 6 2 | 2 3 8 2 7 9 2 5 3 8 | 7 5 1 9 1 1 7 1 4 4 |
| 5 3 1 1 3 6 9 1 7 9 | 6 1 5 2 2 8 8 4 3 3 | 3 5 5 9 4 1 9 5 1 1 | 8 9 6 2 8 8 2 9 2 9 |
| 9 9 0 8 6 9 1 9 6 3 | 2 1 6 8 9 1 0 0 4 1 | 1 9 1 8 8 3 3 1 9 2 | 1 9 8 4 3 3 7 0 6 1 |
| 9 2 3 3 3 3 1 3 3 6 | 5 1 9 1 0 1 5 3 5 7 | 4 7 3 6 4 3 0 2 6 3 | 5 4 7 1 1 9 7 9 1 5 |
| 6 9 9 8 6 1 6 6 6 5 | 6 6 6 1 4 9 1 7 5 8 | 5 7 7 0 5 9 2 7 4 5 | 6 9 2 4 3 4 8 2 8 1 |
| 9 5 2 6 6 5 1 8 9 9 | 1 3 4 3 9 1 3 2 7 0 | 6 9 4 3 6 2 8 5 7 2 | 1 5 9 2 5 6 1 3 5 2 |
| 9 9 7 1 3 1 2 8 2 3 | 4 5 8 2 9 7 0 1 5 9 | 8 4 9 0 5 0 7 3 6 6 | 7 9 3 9 2 7 9 3 9 6 |
| 0 4 6 1 2 3 2 0 8 5 | 6 1 9 4 8 7 2 3 9 5 | 7 4 3 6 3 0 3 1 0 1 | 4 5 2 4 3 9 0 1 5 4 |
| 9 7 5 9 9 3 4 8 5 1 | 2 6 4 5 6 0 9 7 7 8 | 2 1 2 0 4 7 1 9 5 0 | 2 8 7 2 5 1 6 2 3 1 |

(a) 2-D latent space

(b) 5-D latent space

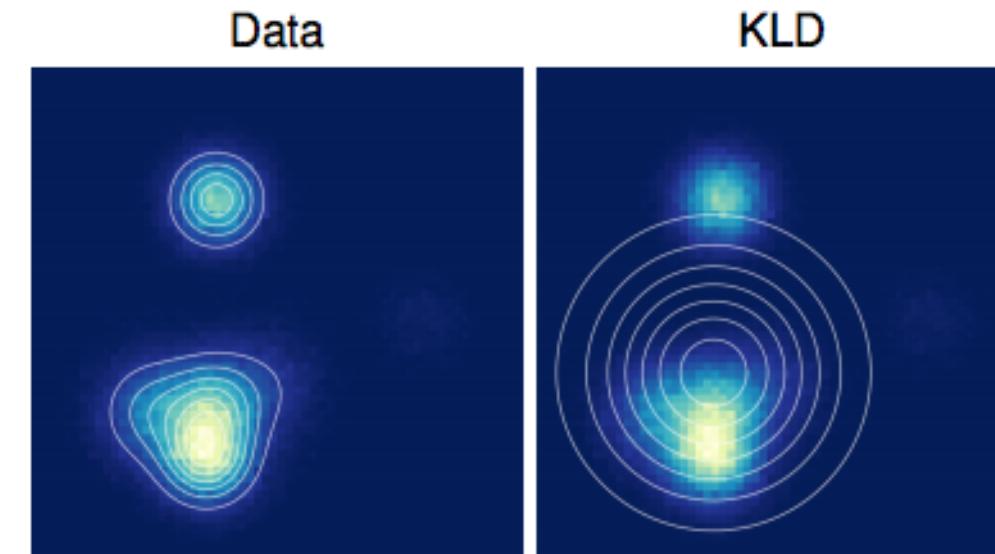
(c) 10-D latent space

(d) 20-D latent space

[Kingma & Welling (2013)]

# VAE tradeoffs

- Pros:
  - Theoretically pleasing
  - Optimizes bound on likelihood
  - Easy to implement
- Cons:
  - Samples tend to be blurry
    - Maximum likelihood minimizes  $D_{KL}(p_{data} || p_{model})$



[Theis *et al.* (2016)]

# Many Other Approaches

- VAE Variants
  - VQ-VAE
  - Beta-VAE
  - Etc.
- Other variants of Autoencoder
  - Restricted / Deep Boltzmann Machines
  - Denoising autoencoders
  - Predictive sparse decomposition
- Decoder-only
  - Sparse coding & hierarchical variants

# VQ-VAE

VQ = Vector quantized

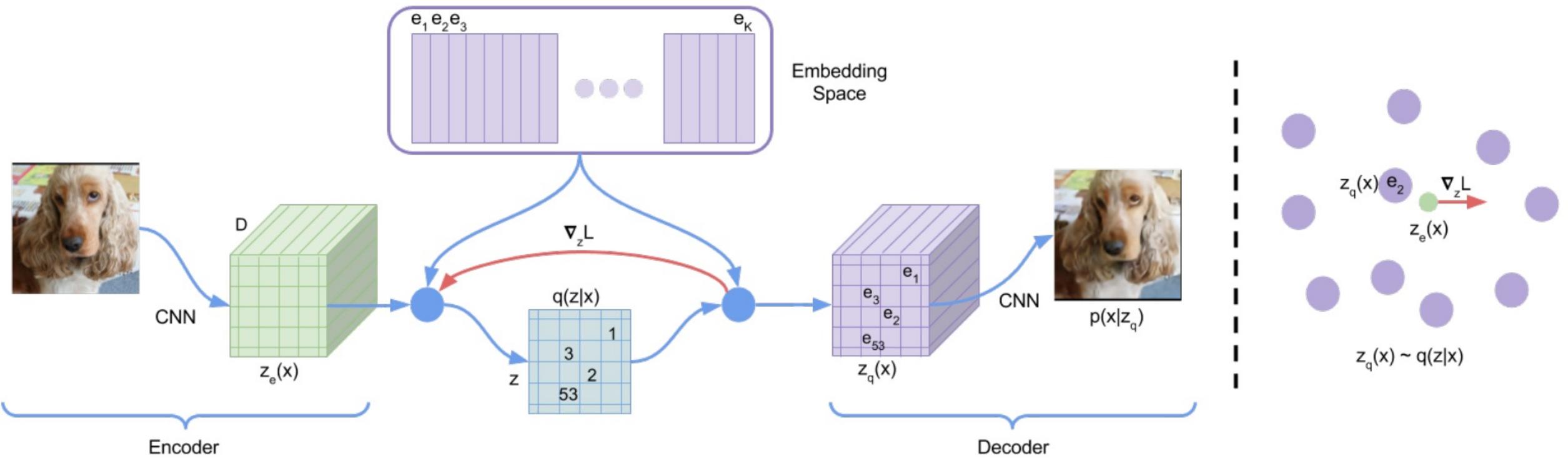
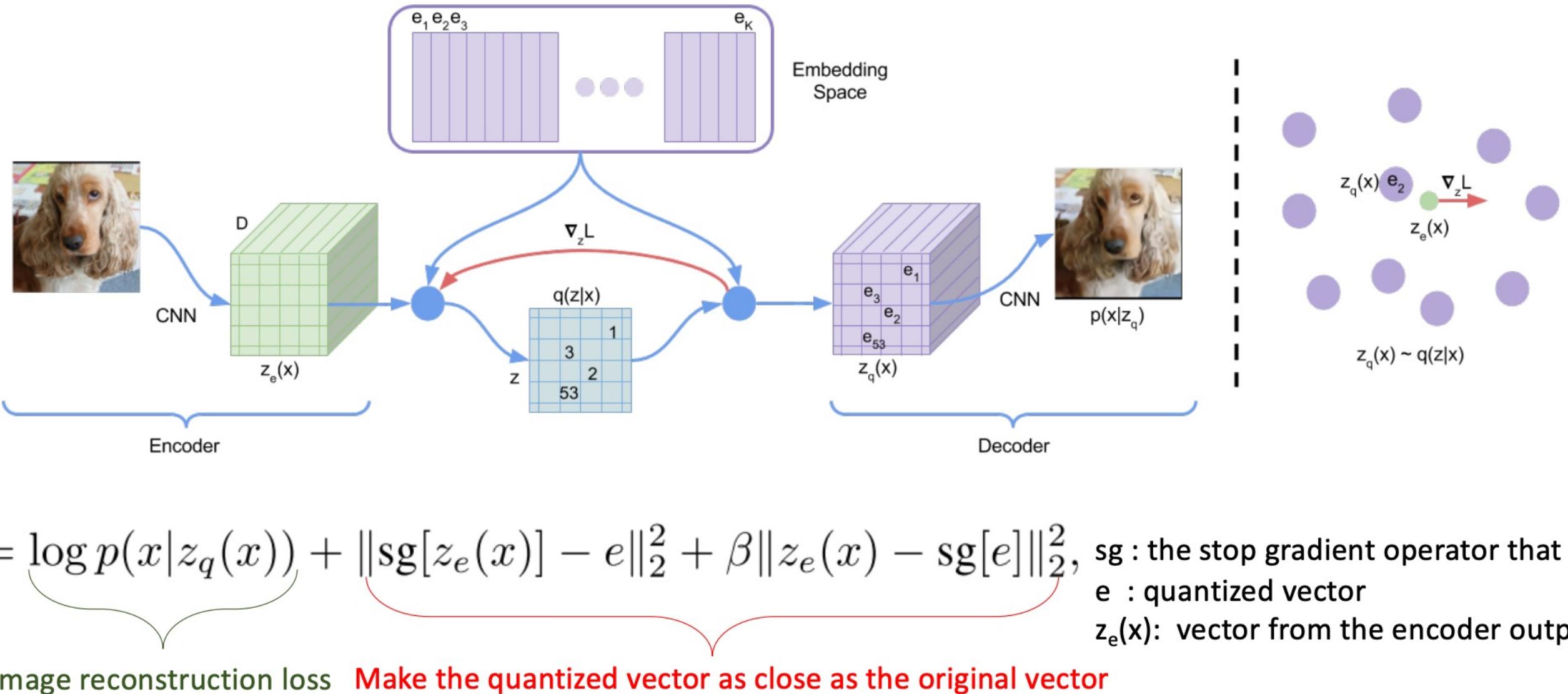


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder  $z(x)$  is mapped to the nearest point  $e_2$ . The gradient  $\nabla_z L$  (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

VQ-VAE: Neural discrete representation learning. Van Den Oord,  
Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

# VQ-VAE



VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

# VQ-VAE Reconstructions



Figure 2: Left: ImageNet 128x128x3 images, right: reconstructions from a VQ-VAE with a 32x32x1 latent space, with K=512.

VQ-VAE: Neural discrete representation learning. Van Den Oord, Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

# VQ-VAE Samples

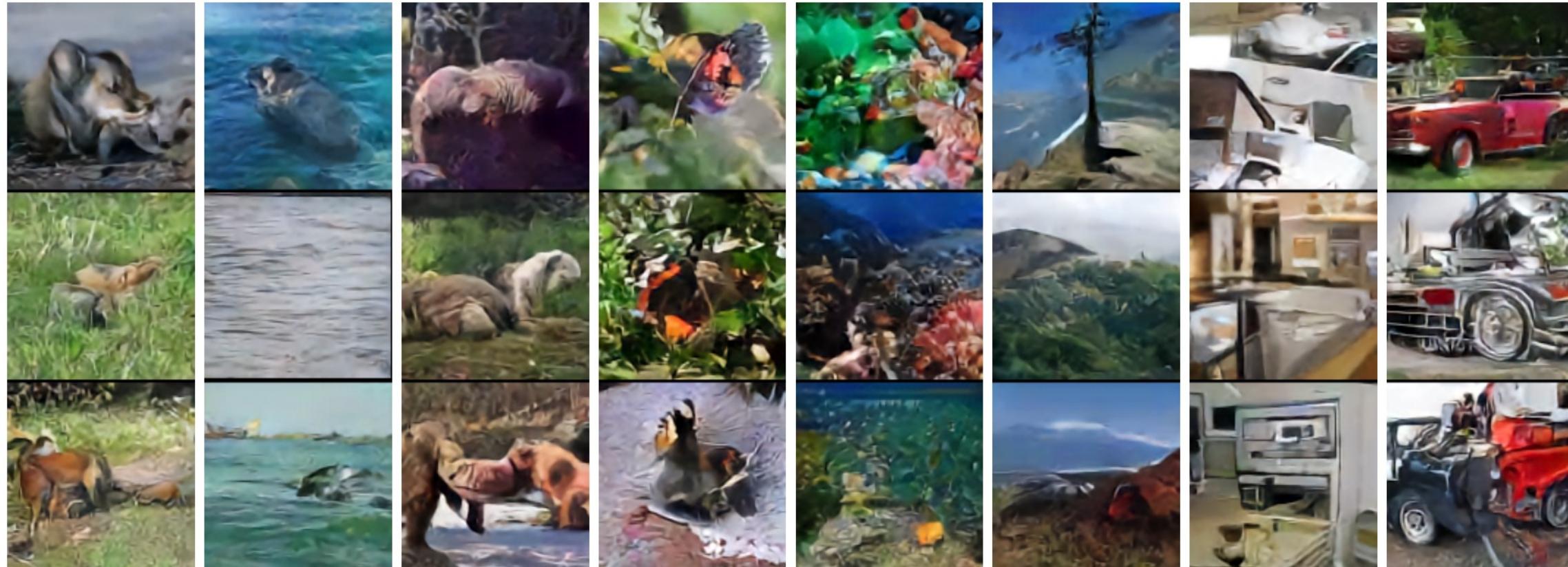


Figure 3: Samples (128x128) from a VQ-VAE with a PixelCNN prior trained on ImageNet images. From left to right: kit fox, gray whale, brown bear, admiral (butterfly), coral reef, alp, microwave, pickup.

VQ-VAE: Neural discrete representation learning. Van Den Oord,  
Aaron. Vinyals, Oriol. Kavukcuoglu, Koray. NeurIPS 2017

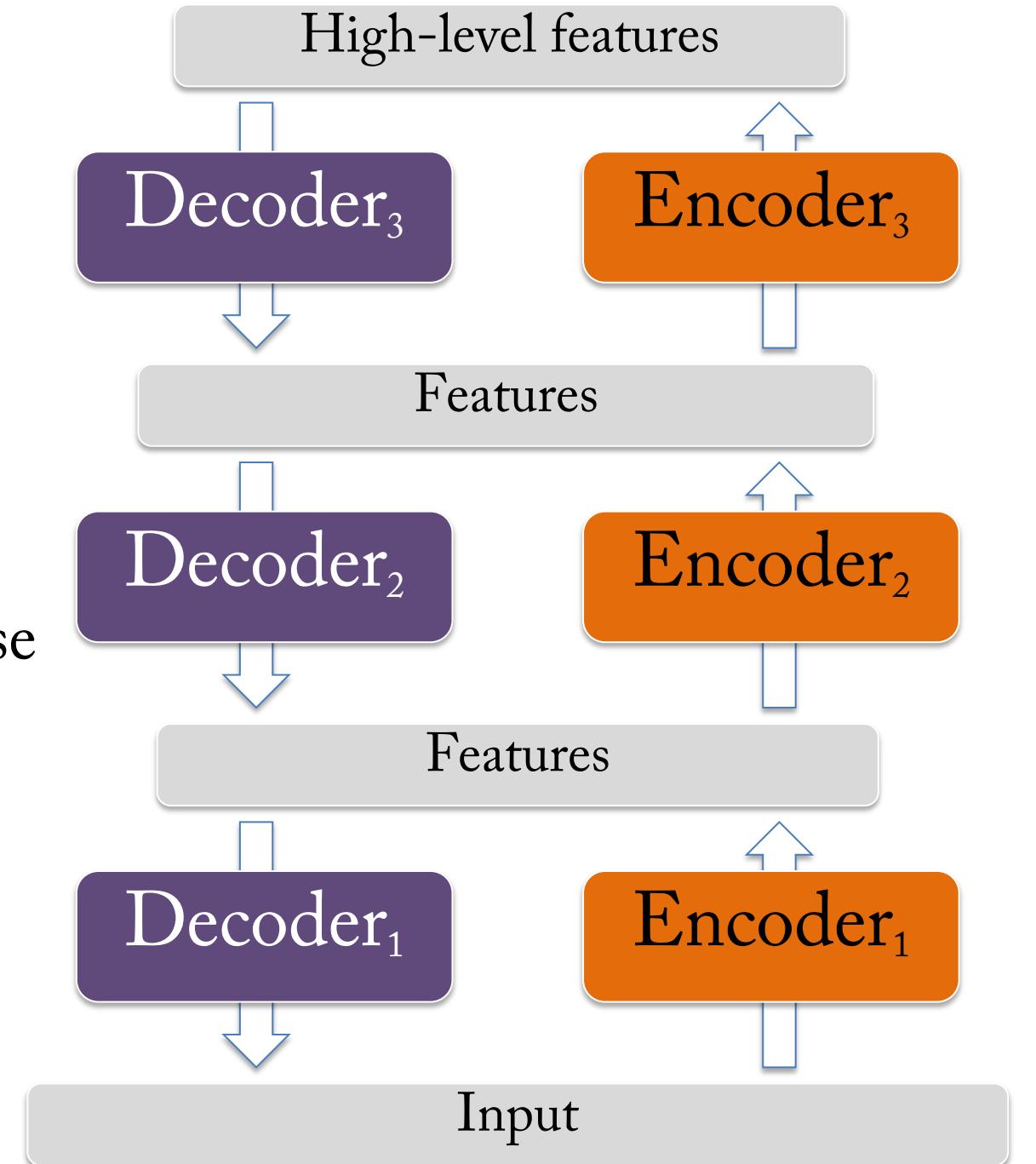
# Many Other Approaches

- VAE Variants
  - VQ-VAE
  - Beta-VAE
  - Etc.
- Other variants of Autoencoder
  - Restricted / Deep Boltzmann Machines
  - Denoising autoencoders
  - Predictive sparse decomposition
- Decoder-only
  - Sparse coding & hierarchical variants

# Stacked Auto-Encoders

- Ladder Networks  
[Rasmus et al. 2015]
  - Reconstruction constraint at each layer
  - Trained end-to-end
- Can be trained layer-wise
  - Stacked RBMs

[Hinton & Salakhutdinov 2006]



# Autoregressive models

- Tractably model a joint distribution of the pixels in the image
- Learn to predict the next pixel given all the previously generated pixels
- Joint distribution of all pixels just product of conditionals:

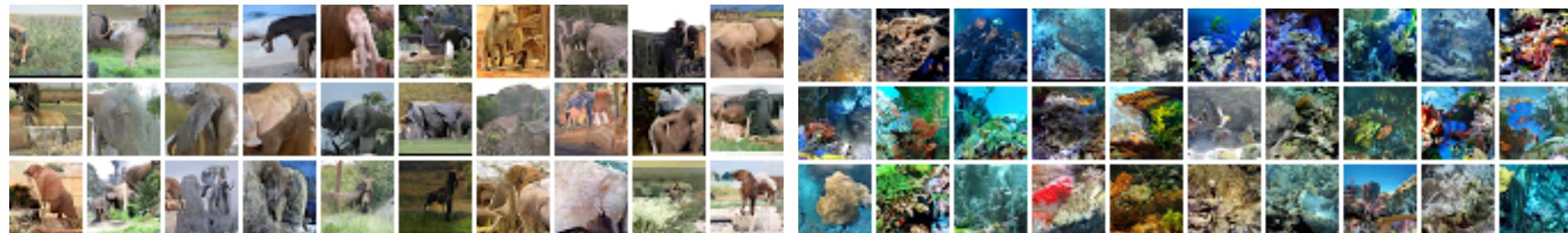
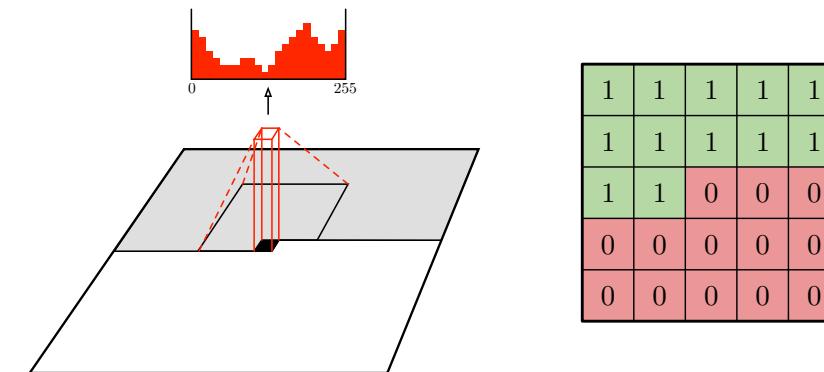
$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1})$$

# Pixel-CNN

[van den Oord et al., arXiv 1606.05328, 2016]

- Conditional generative model of images
- Generate each pixel, in raster-scan order
- Just predict distribution over a single pixel (can be multi-modal)
- See also Video Pixel Networks [Kalchbrenner et al., 2016],
- NADE [Larochelle & Murray 2011] & RIDE [Theis and Bethge, NIPS 2015].

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, \dots, x_{i-1}).$$



African elephant

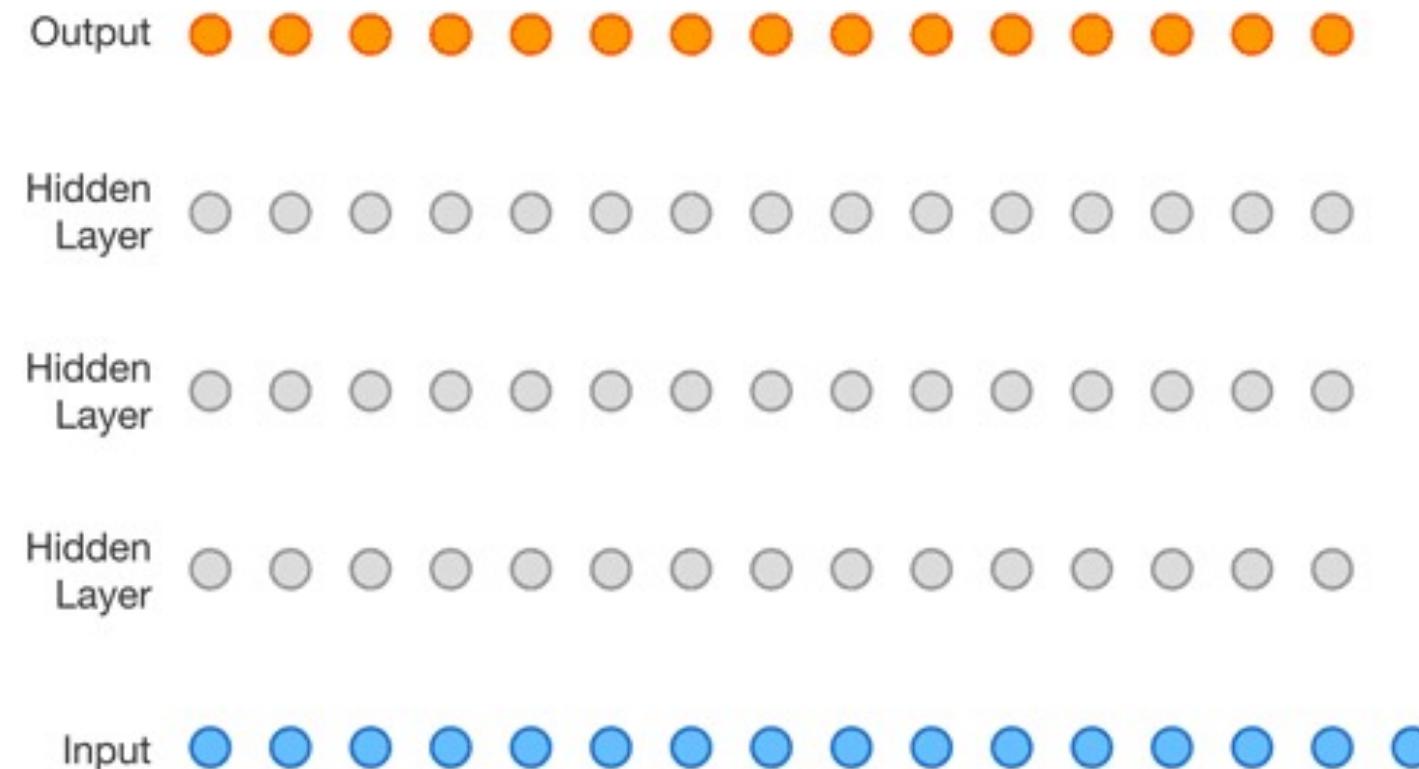
Coral Reef

# Wavenet

[van den Oord et al., arXiv 1609.03499, 2016]

- Generative model of raw speech waveform
- Condition on previous parts of waveform
- Dilated causal convolution layers
- Discrete output distribution (use softmax)

$$p(\mathbf{x}) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1})$$



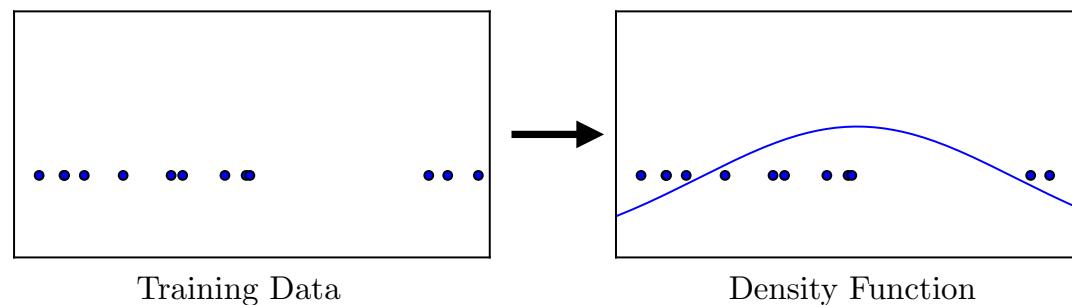
# Generative Adversarial Networks

Slides from: Emily Denton, Ian Goodfellow, Soumith Chintala

# Generative Adversarial Networks

- [Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, NIPS 2014]
- Focus on sample generation

Generative Modeling: Density Estimation



Generative Modeling:  
Sample Generation



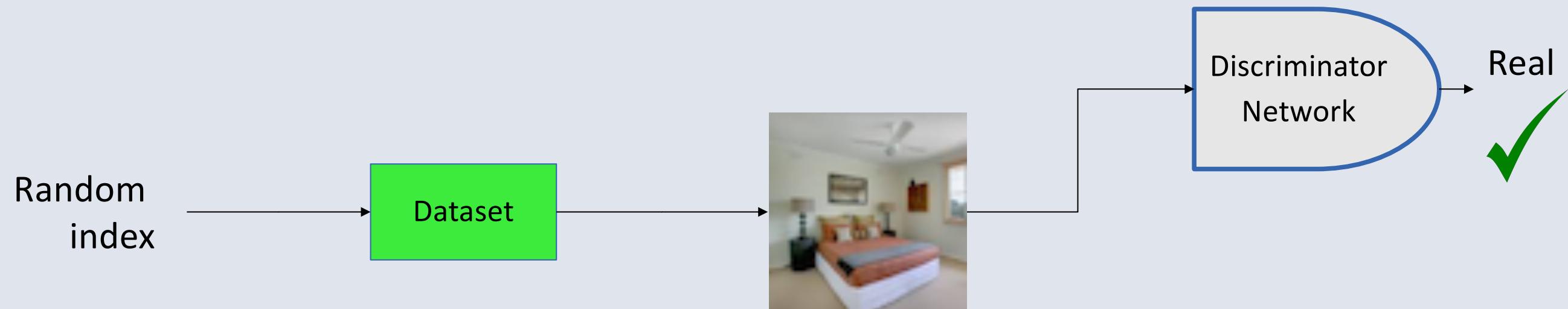
(Goodfellow 2018)

(Goodfellow 2018)

# Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

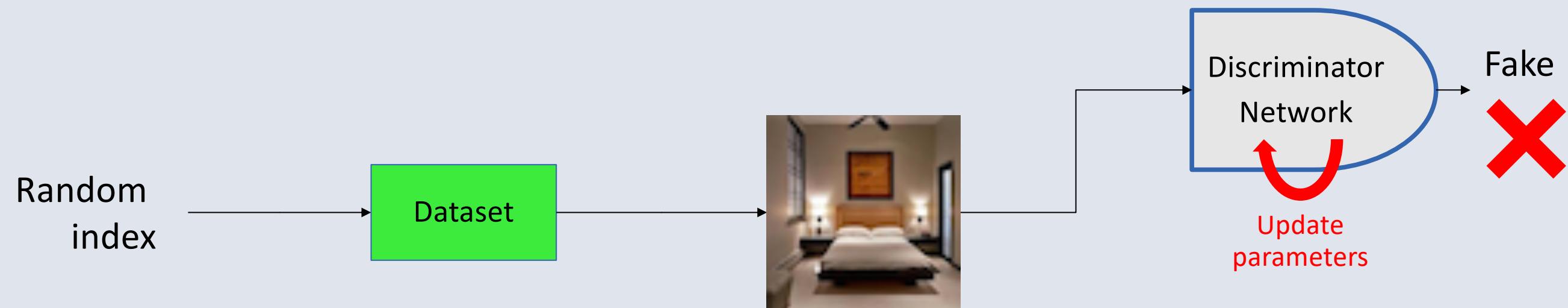
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)



# Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

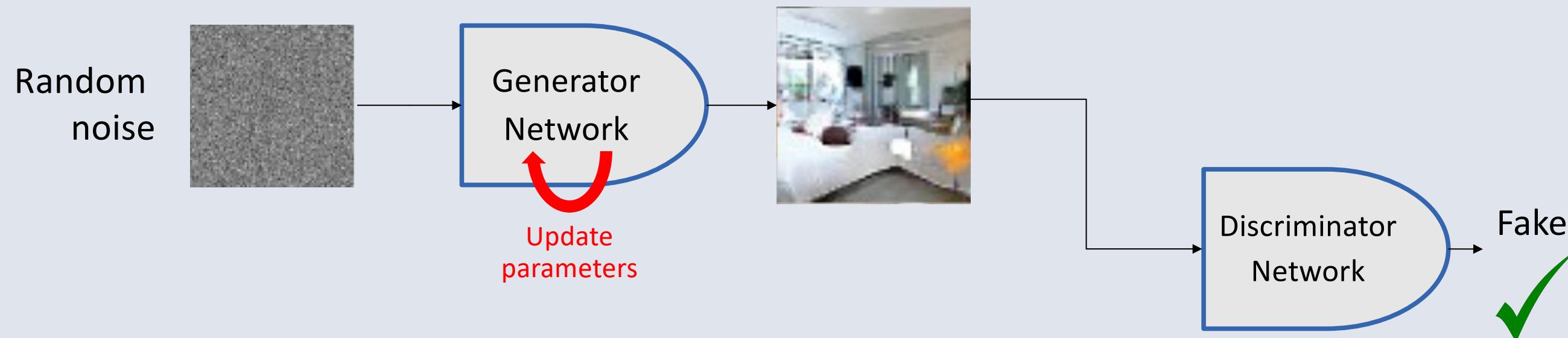
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)



# Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

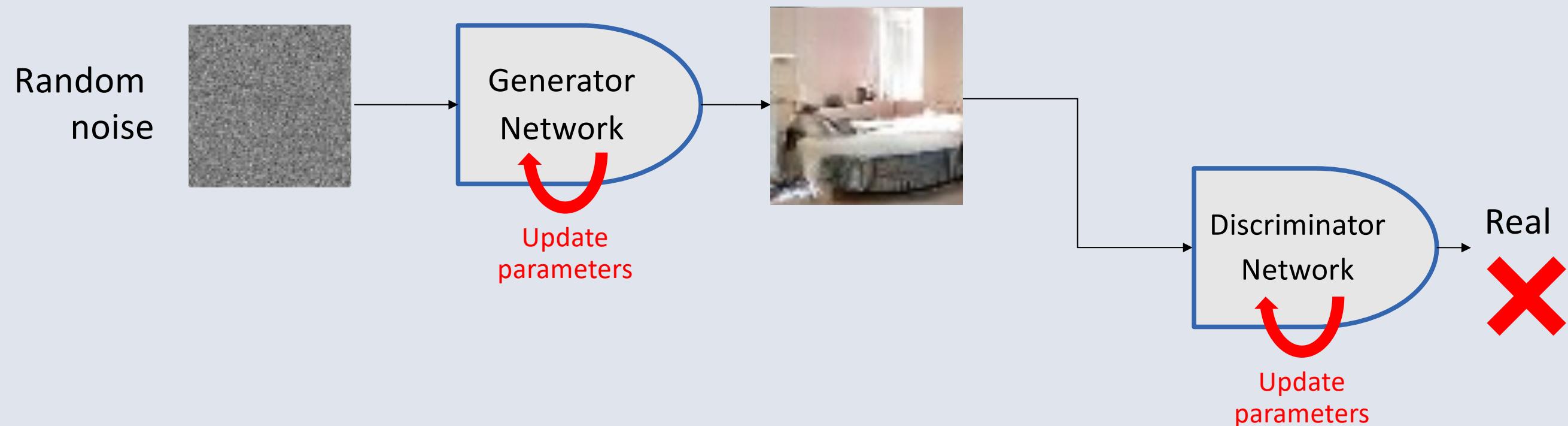
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse Discriminator



# Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

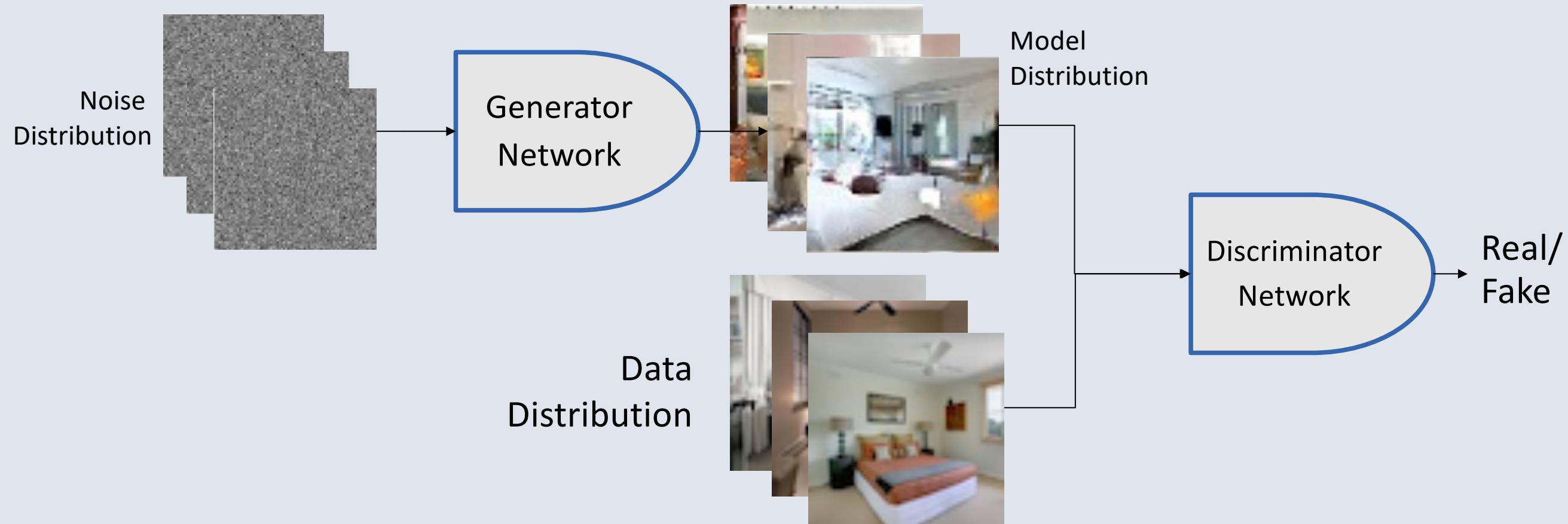
- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse discriminator



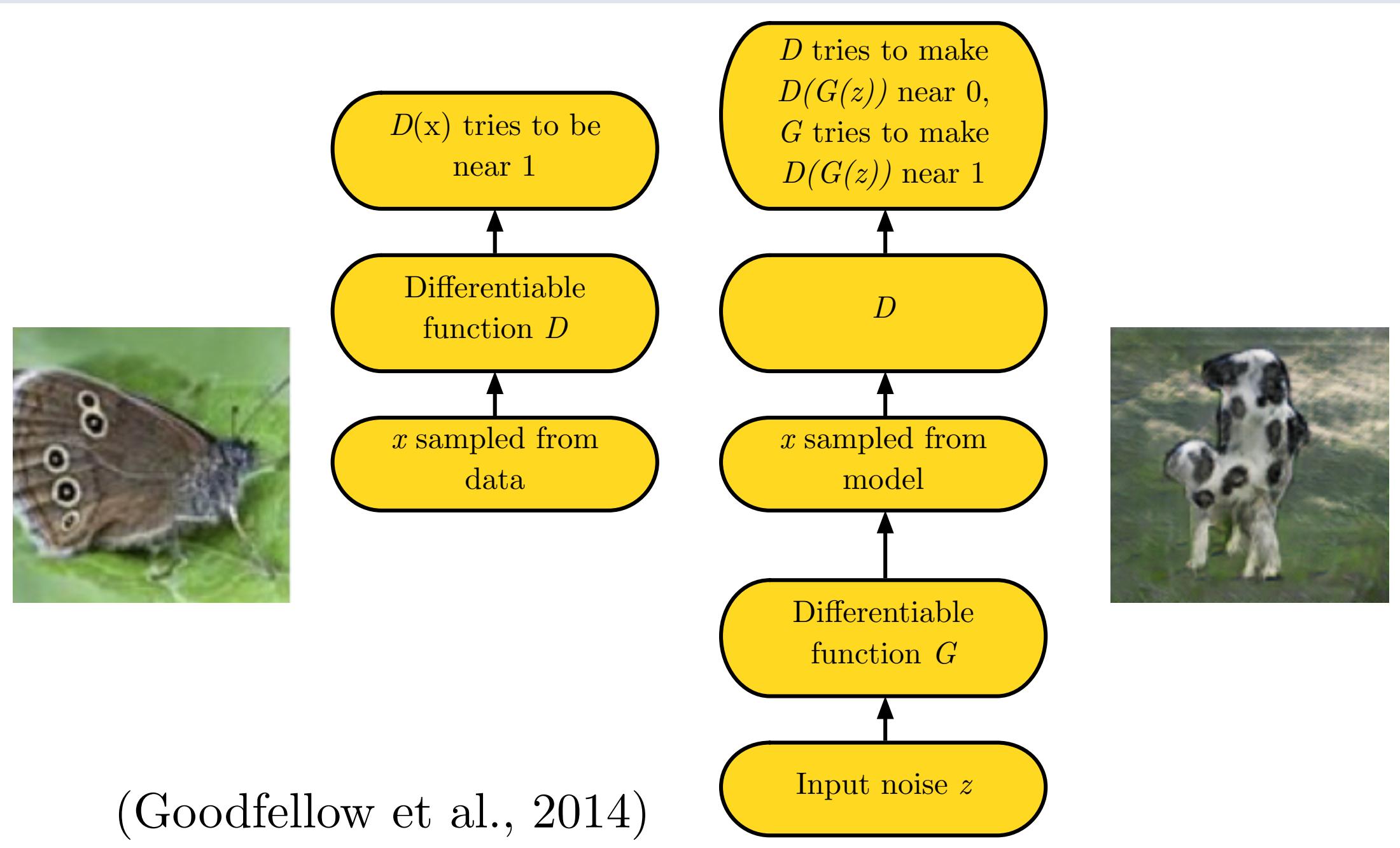
# Generative Adversarial Network

[Goodfellow et al. NIPS 2014]

- Initial application to still images
- Way to train generative model to match **distribution** of data
- Discriminator network predicts if input image is from data (real) or model (fake)
- Generator network tries to confuse Discriminator



# Generative Adversarial Networks



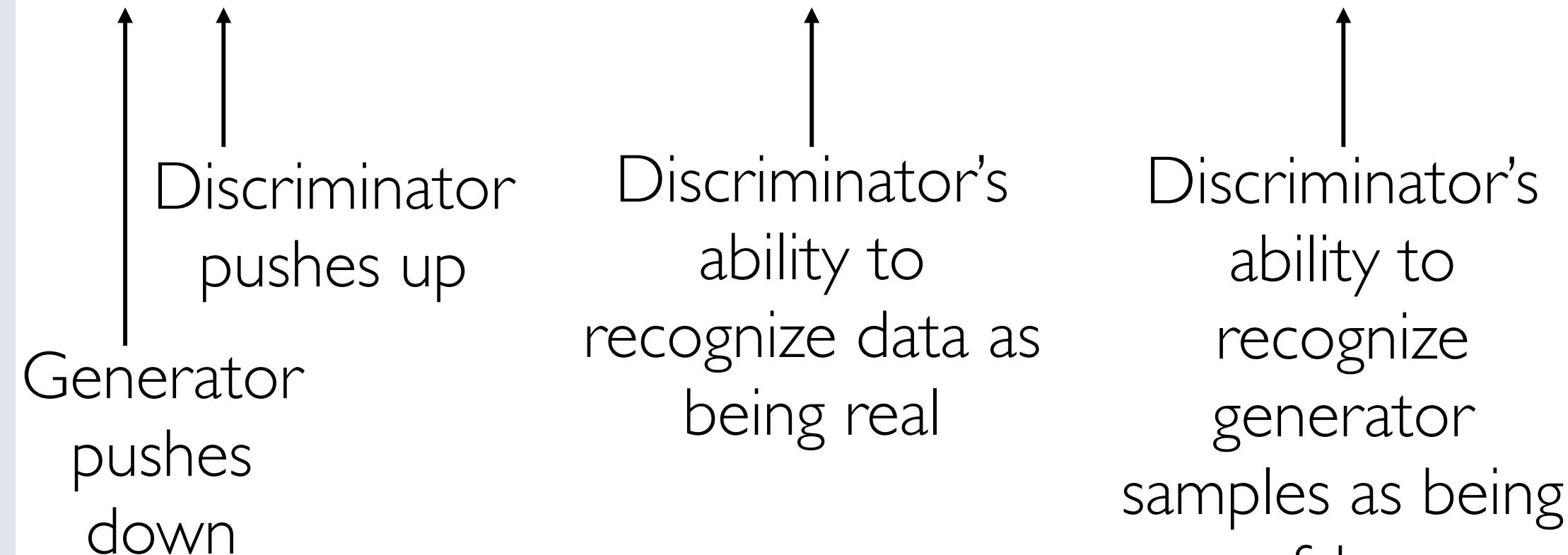
# Generative Adversarial Networks

---

[Generative Adversarial Nets, Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, NIPS 2014]

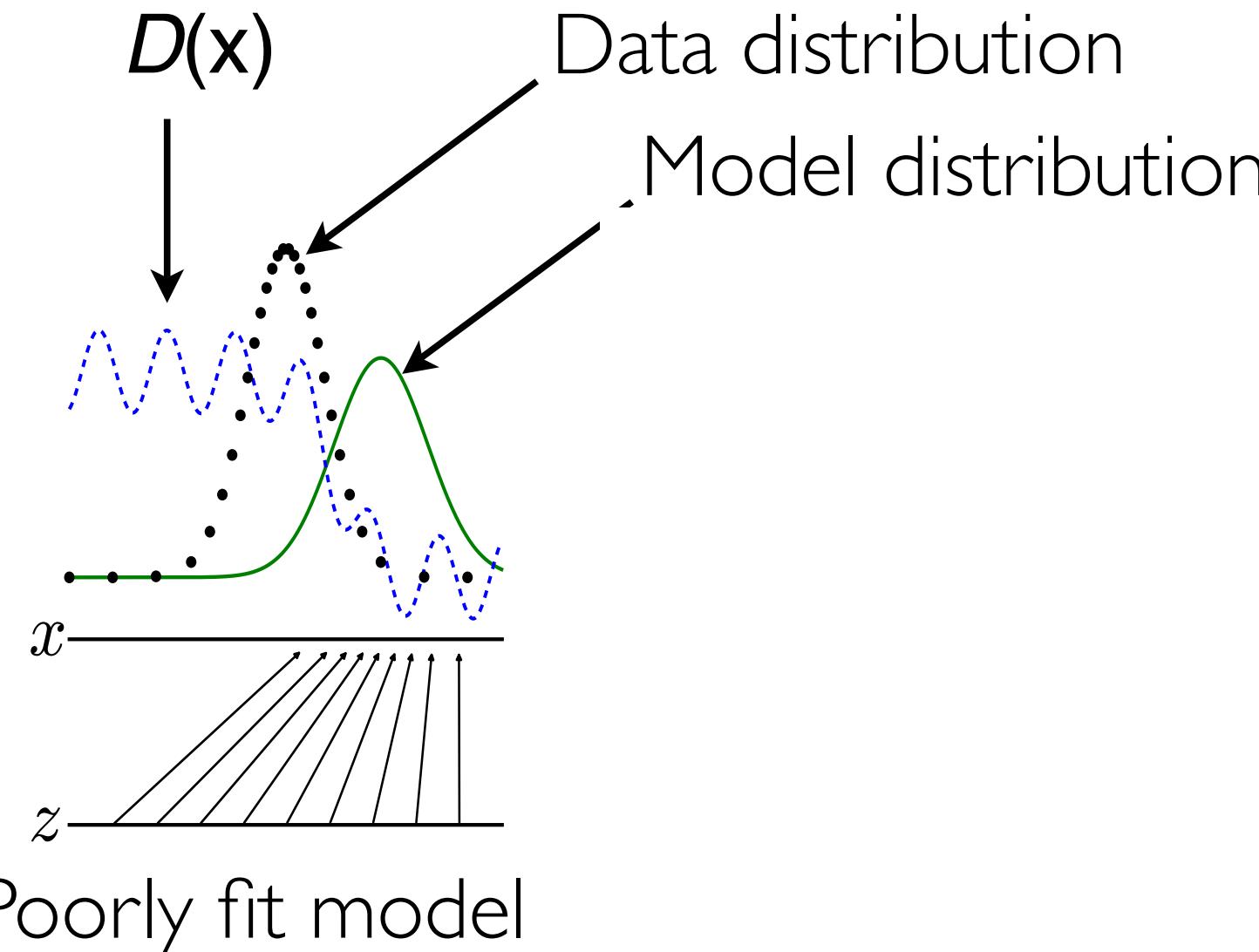
- Minimax value function:

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



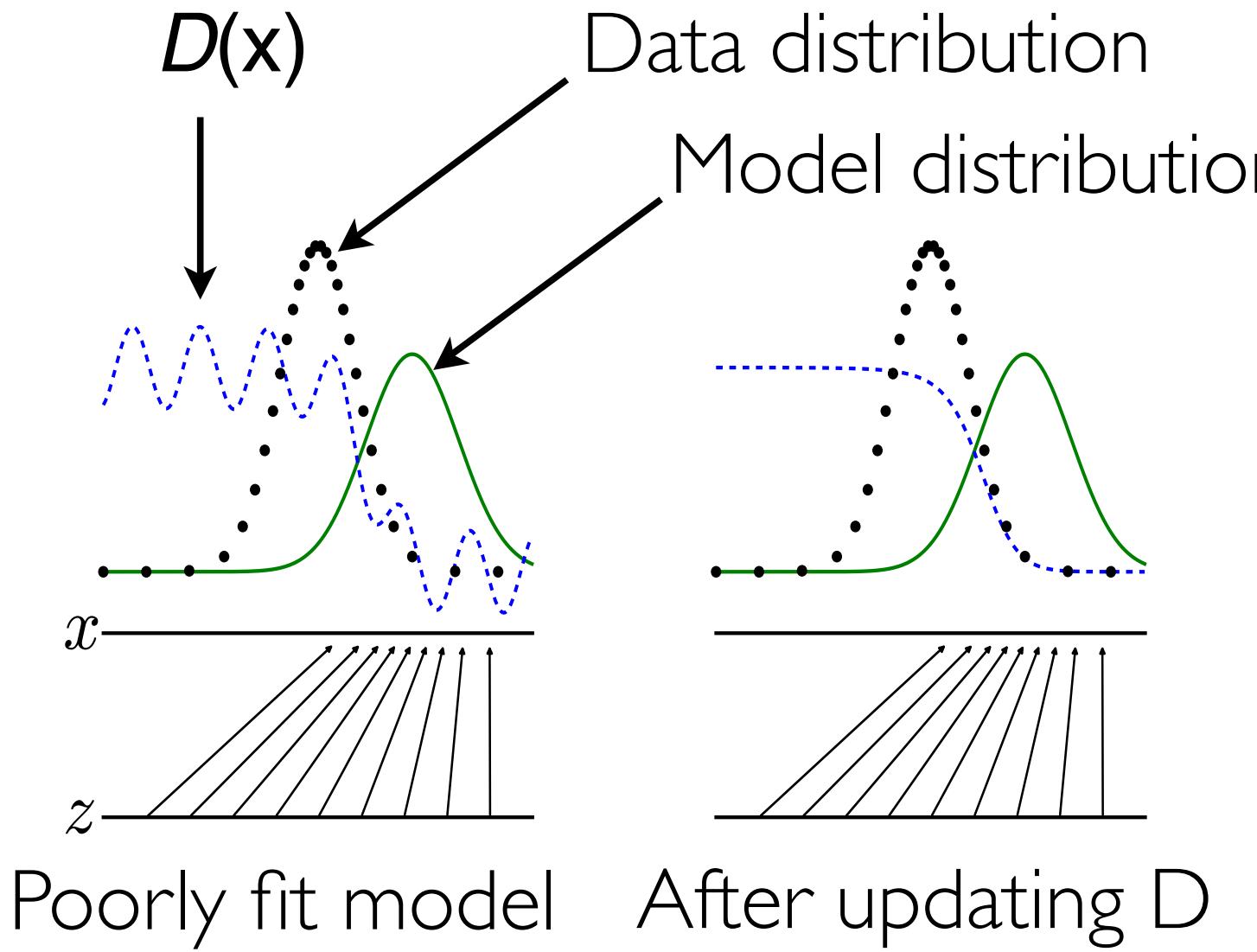
# Generative Adversarial Networks

---



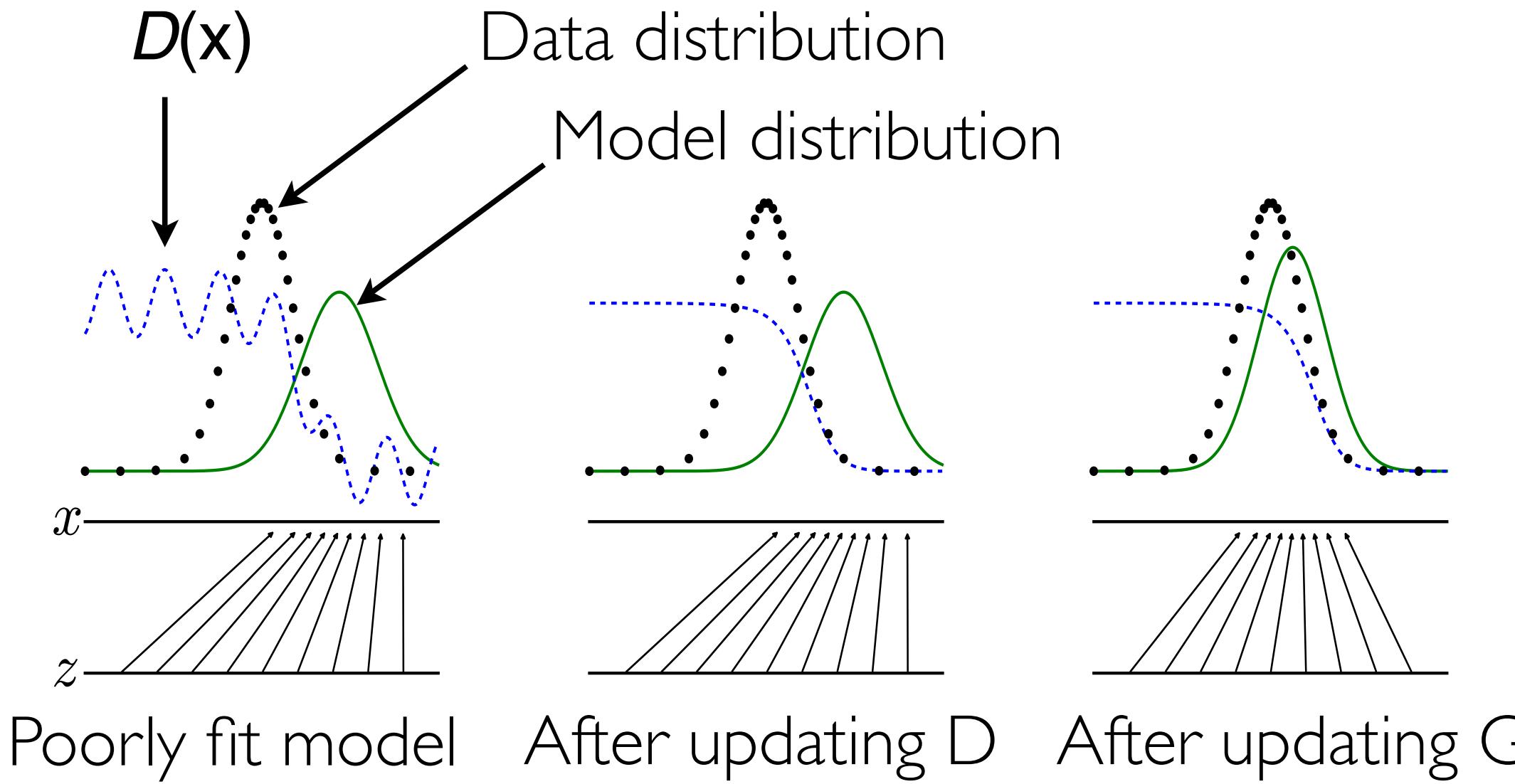
# Generative Adversarial Networks

---

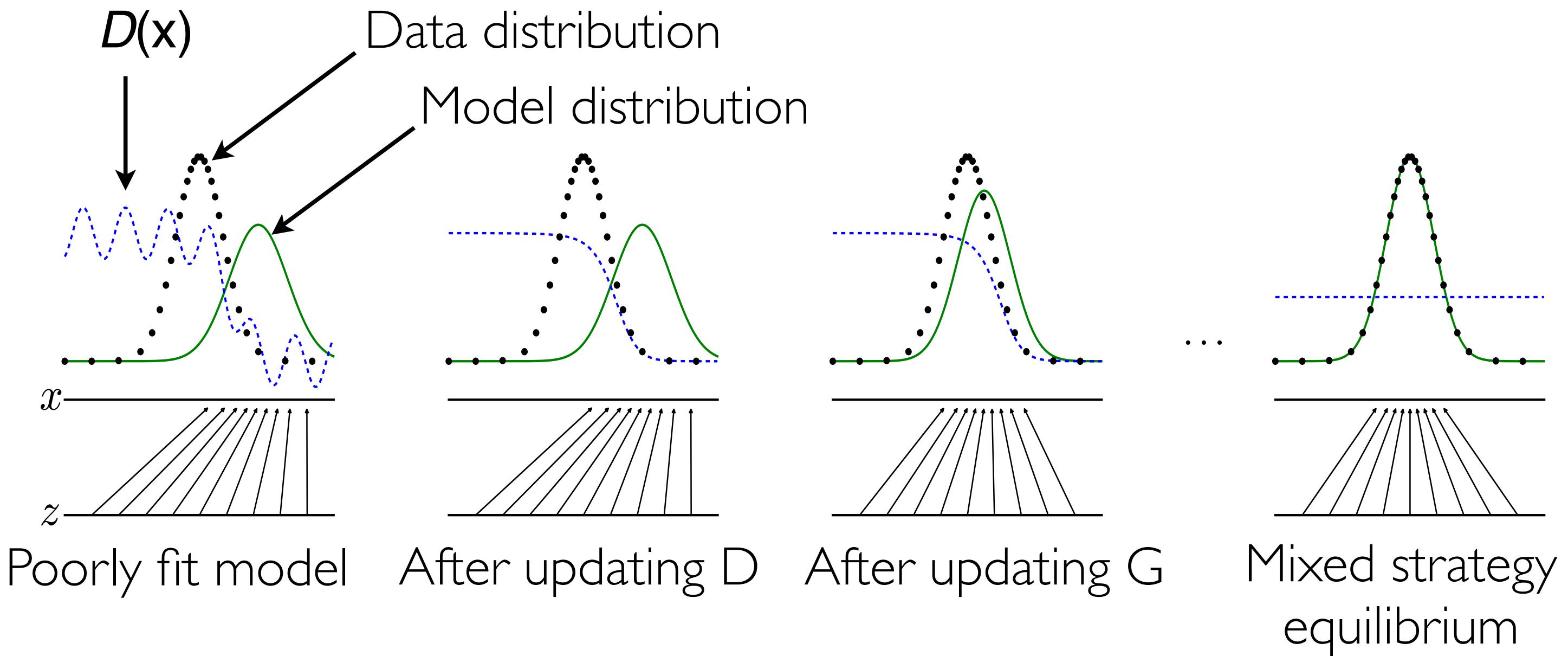


# Generative Adversarial Networks

---

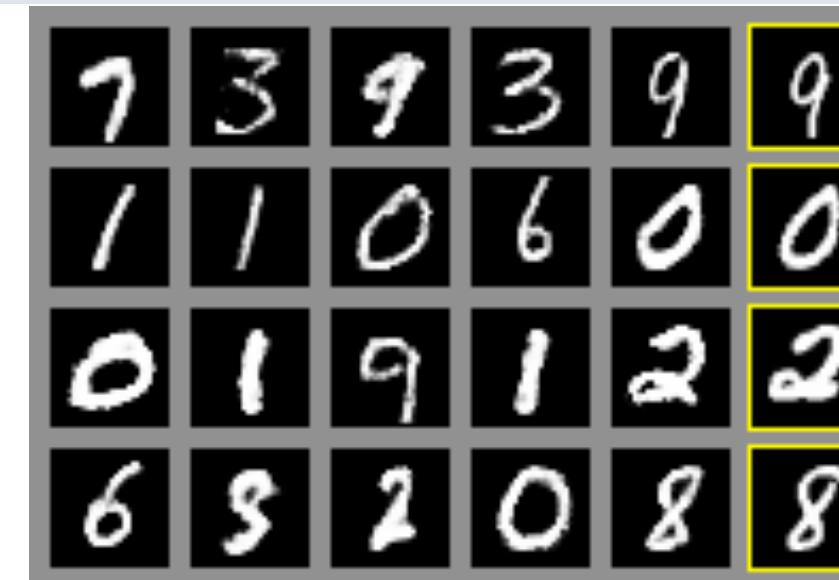


# Generative Adversarial Networks



# Adversarial Network Samples

---



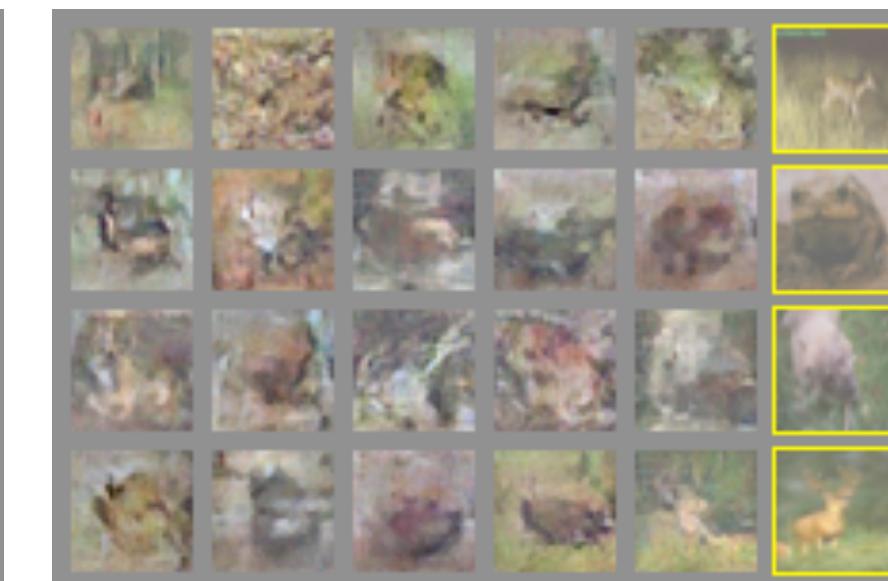
MNIST



TFD



CIFAR-10 (fully connected)



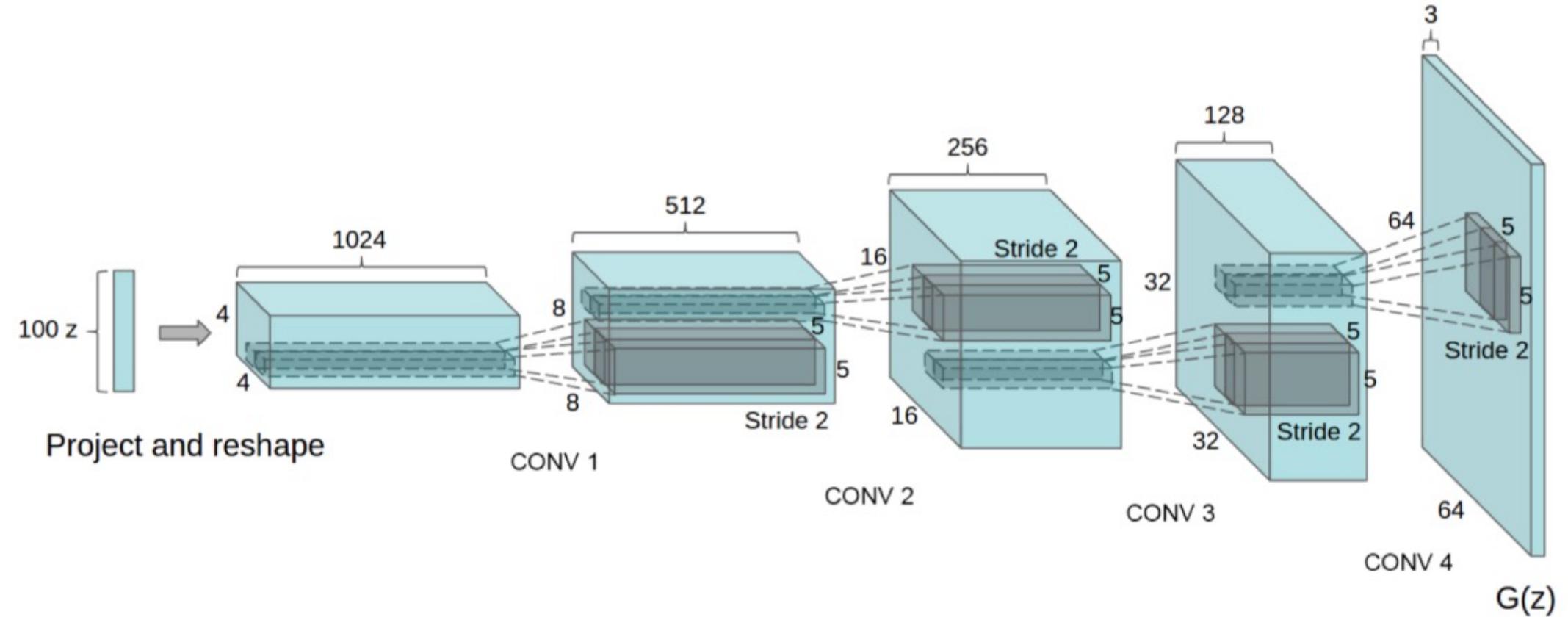
CIFAR-10 (convolutional)

# DCGAN

UNSUPERVISED REPRESENTATION LEARNING  
WITH DEEP CONVOLUTIONAL  
GENERATIVE ADVERSARIAL NETWORKS

- First to generate plausible results at 64x64.
- Improved architectures for generator/discriminator
  - Use strided convolution – no pooling
- Most GAN architectures similar

- Generator architecture



- Good default GAN architecture

Alec Radford & Luke Metz  
indico Research  
Boston, MA  
[{alec,luke}@indico.io](mailto:{alec,luke}@indico.io)

Soumith Chintala  
Facebook AI Research  
New York, NY  
[soumith@fb.com](mailto:soumith@fb.com)

ICLR 2016

# DCGAN

UNSUPERVISED REPRESENTATION LEARNING  
WITH DEEP CONVOLUTIONAL  
GENERATIVE ADVERSARIAL NETWORKS

- First to generate plausible results at 64x64.
- Improved architectures for generator/discriminator
- Most GAN architectures used now are similar
- Lots of tricks to get GANs to train well

Alec Radford & Luke Metz  
indico Research  
Boston, MA  
[{alec,luke}@indico.io](mailto:{alec,luke}@indico.io)

Soumith Chintala  
Facebook AI Research  
New York, NY  
[soumith@fb.com](mailto:soumith@fb.com)

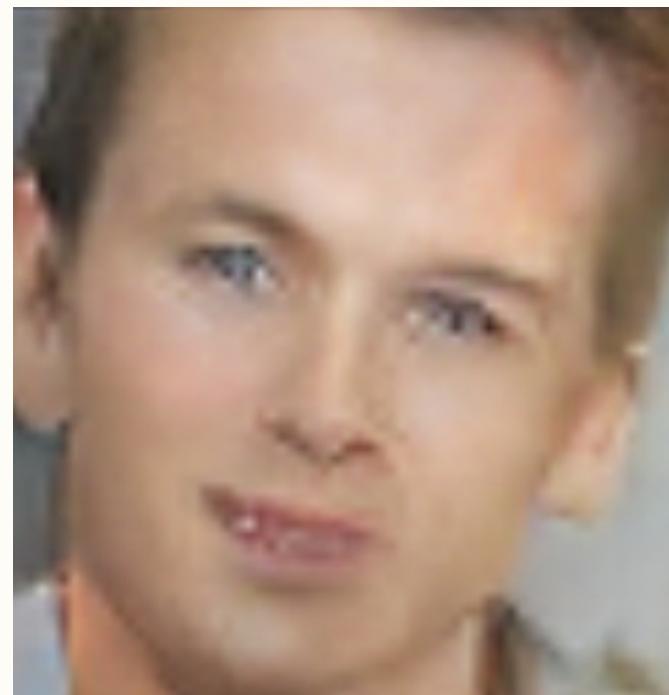
ICLR 2016



# 3.5 Years of Progress on Faces



2014



2015



2016

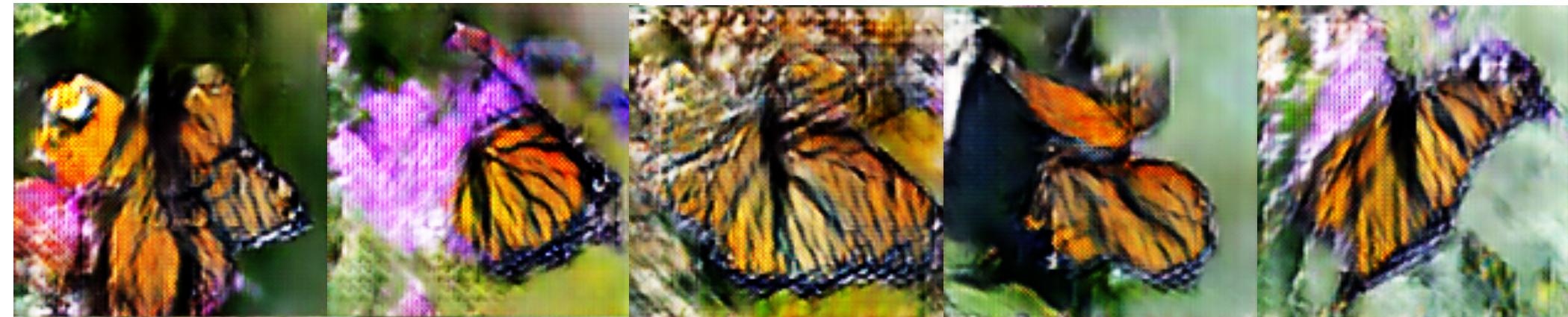


2017

(Brundage et al, 2018)

# <2 Years of Progress on ImageNet

Odena et al  
2016



Miyato et al  
2017



Zhang et al  
2018



(Goodfellow 2018)

# Evaluation of GANs

- Short answer: hard; look at quality of samples
- Computing log-likelihood not directly possible
  - LLH is problematic.  
See [[A note on the evaluation of generative models](#), Lucas Theis, Aäron van den Oord, Matthias Bethge, ICLR 2016]
- Inception score
- User-study: can humans tell fake from real?

# Inception Score

Proposed in 2016

---

## Improved Techniques for Training GANs

---

**Tim Salimans**

tim@openai.com

**Ian Goodfellow**

ian@openai.com

**Wojciech Zaremba**

woj@openai.com

**Vicki Cheung**

vicki@openai.com

**Alec Radford**

alec.radford@gmail.com

**Xi Chen**

peter@openai.com

### Abstract

# Inception Score

- Send generated image through Inception model (trained on Imagenet)

generated image to get the conditional label distribution  $p(y|x)$ . Images that contain meaningful objects should have a conditional label distribution  $p(y|x)$  with low entropy. Moreover, we expect the model to generate varied images, so the marginal  $\int p(y|x = G(z))dz$  should have high entropy. Combining these two requirements, the metric that we propose is:  $\exp(\mathbb{E}_x \text{KL}(p(y|x) || p(y)))$ , where

# Inception Score

- Send generated image through Inception model (trained on Imagenet)

generated image to get the conditional label distribution  $p(y|x)$ . Images that contain meaningful objects should have a conditional label distribution  $p(y|x)$  with low entropy. Moreover, we expect the model to generate varied images, so the marginal  $\int p(y|x = G(z))dz$  should have high entropy. Combining these two requirements, the metric that we propose is:  $\exp(\mathbb{E}_x \text{KL}(p(y|x) || p(y)))$ , where

# How to Train a GAN

**Emily Denton, Martin Arjovsky, Michael Mathieu**

New York University

**Ian Goodfellow**

Google

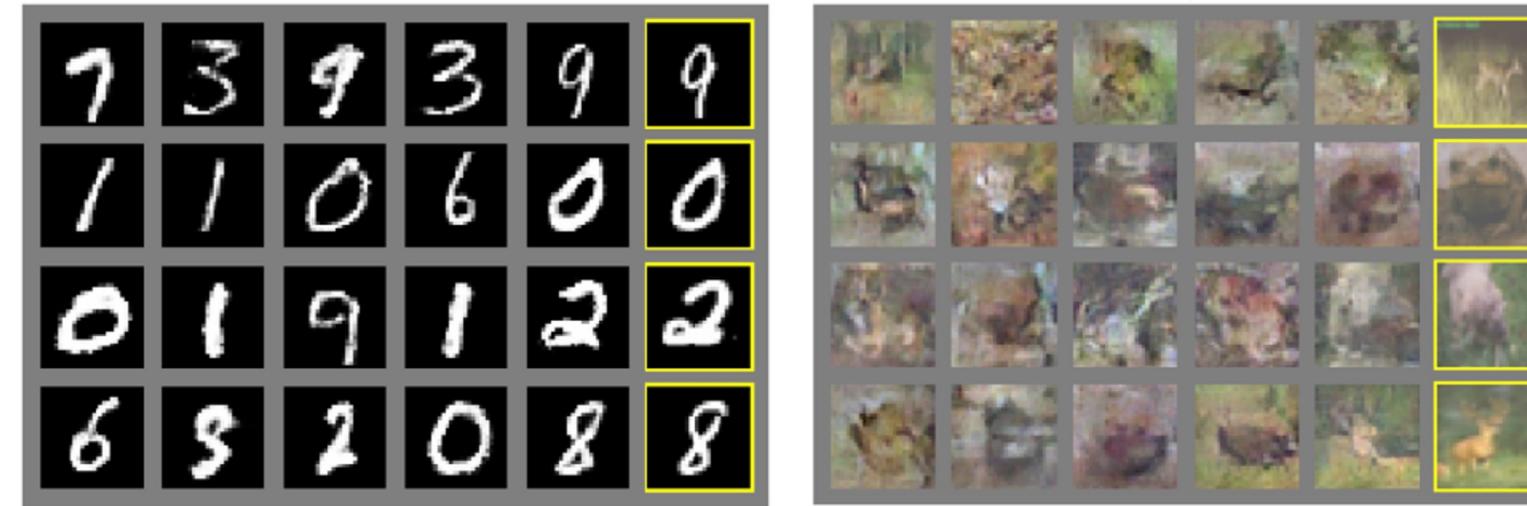
**Soumith Chintala**

Facebook AI Research

# The stability of GANs



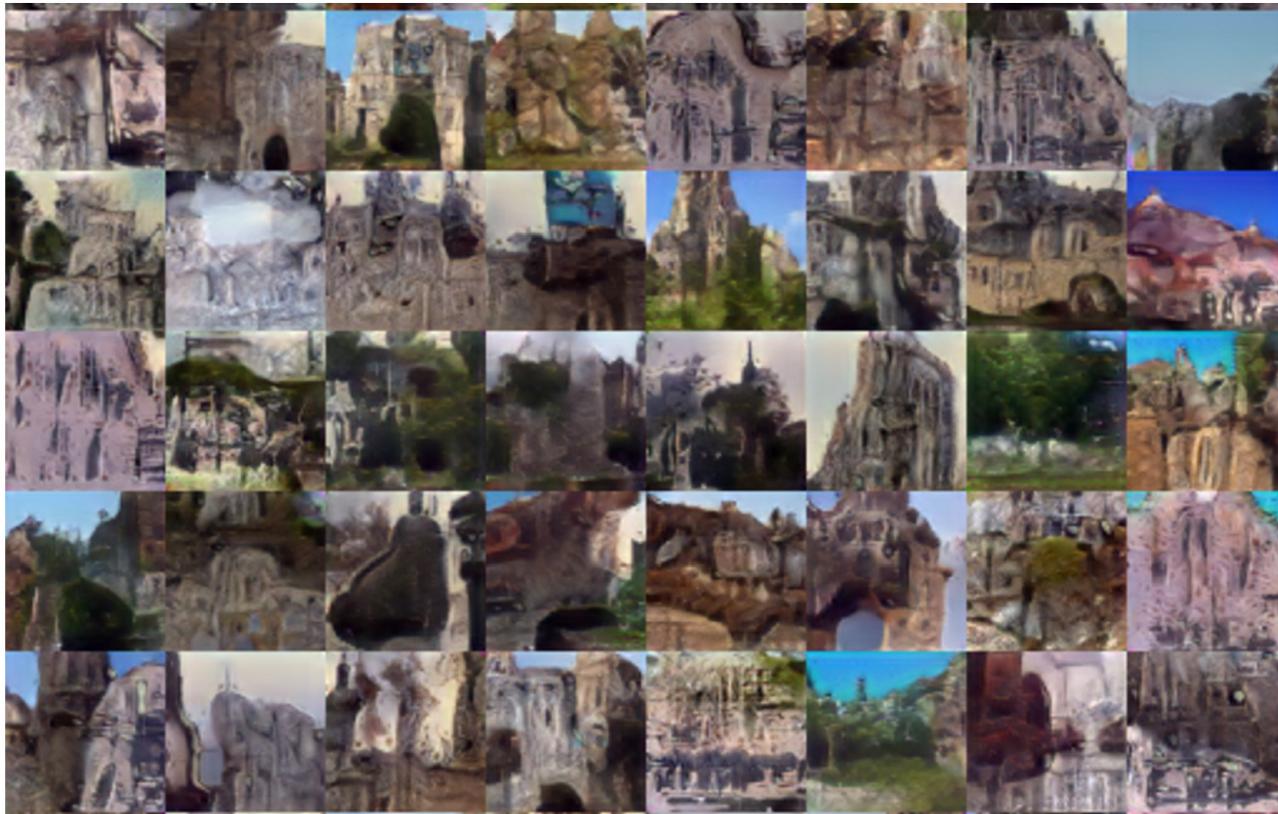
# Timeline - the stability of GANs



Goodfellow et. al. “Generative Adversarial Networks”

2014

# Timeline - the stability of GANs

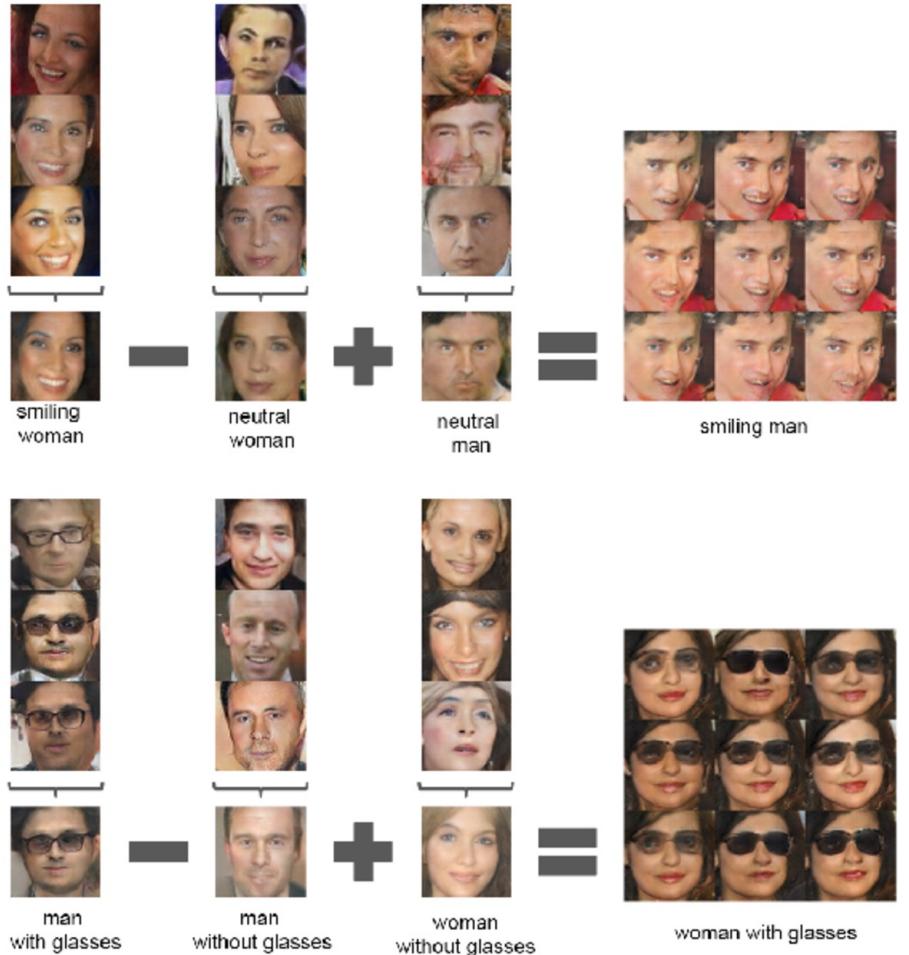


model architecture generator  
visual inspection  
countless failed stability hacks

Denton et. al. “Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks”

2015

# Timeline - the stability of GANs



countless hours finding stable models

stable upto 64x64

mode dropping

underfitting

Radford et. al. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”

2015

# Timeline - the stability of GANs



more heuristics  
more stability

Salimans et. al. “Improved Techniques for Training GANs”

---

2015

# Timeline - the stability of GANs

gradient norm regularization

Least-Squares  
Boundary Equilibrium

Gulrajani et. al. “Improved Training of Wasserstein GANs”

Xudong et al. “Least squares generative adversarial networks.”

Berthelot et. al. “Began: Boundary equilibrium generative adversarial networks.”



2016-2017

# Timeline - the stability of GANs

<https://github.com/khanrc/tf.gans-comparison>  
by Junbum Cha

## GANs comparison without cherry-picking

---

Implementations of some theoretical generative adversarial nets: DCGAN, EBGAN, LSGAN, WGAN, WGAN-GP, BEGAN, DRAGAN and CoulombGAN.

I implemented the structure of model equal to the structure in paper and compared it on the CelebA dataset and LSUN dataset without cherry-picking.

# Comparison to Classification ConvNets

- Throw things at the wall and see what sticks
- Intuition is poorer
- Theoretical work is somewhat improving but still far away
- Objective validation metrics are not there yet

# #1: Normalize the inputs

- normalize the images between -1 and 1
- Tanh as the last layer of the generator output
  - or some kind of bounds normalization

# #2: Modified loss function (classic GAN)

- In papers people write  $\min(\log 1-D)$ , but in practice folks practically use  $\max \log D$ 
  - because the first formulation has vanishing gradients early on
  - Goodfellow et. al (2014)
- In practice:
  - Flip labels when training generator: real = fake, fake = real

# #2: Modified loss function (classic GAN)

- In papers people write  $\min(\log 1-D)$ , but in practice folks practically use  $\max \log D$ 
  - because the first formulation has vanishing gradients early on
  - Goodfellow et. al (2014)
- In practice:
  - Flip labels when training generator: real = fake, fake = real

**LOT OF NEW LOSS FORMULATIONS**

# #2: Modified loss function (classic GAN)

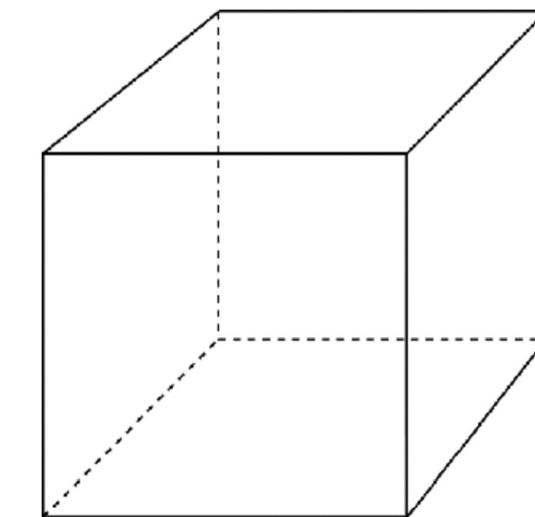
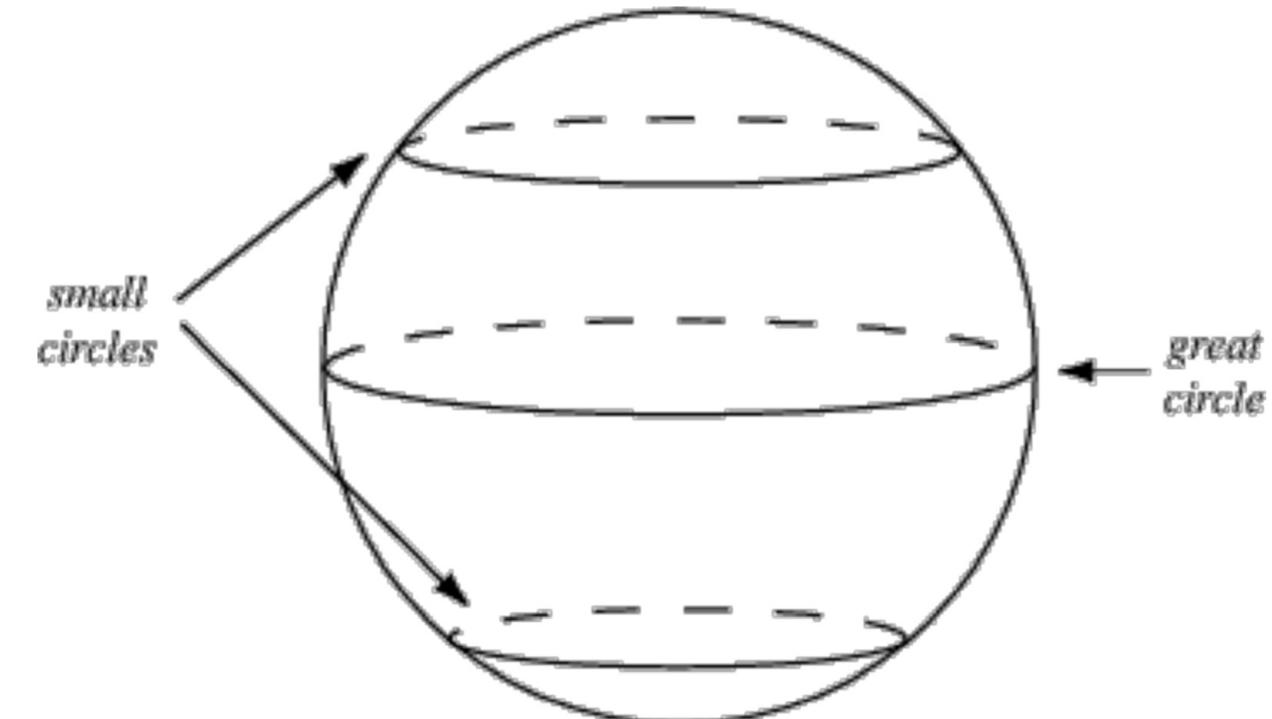
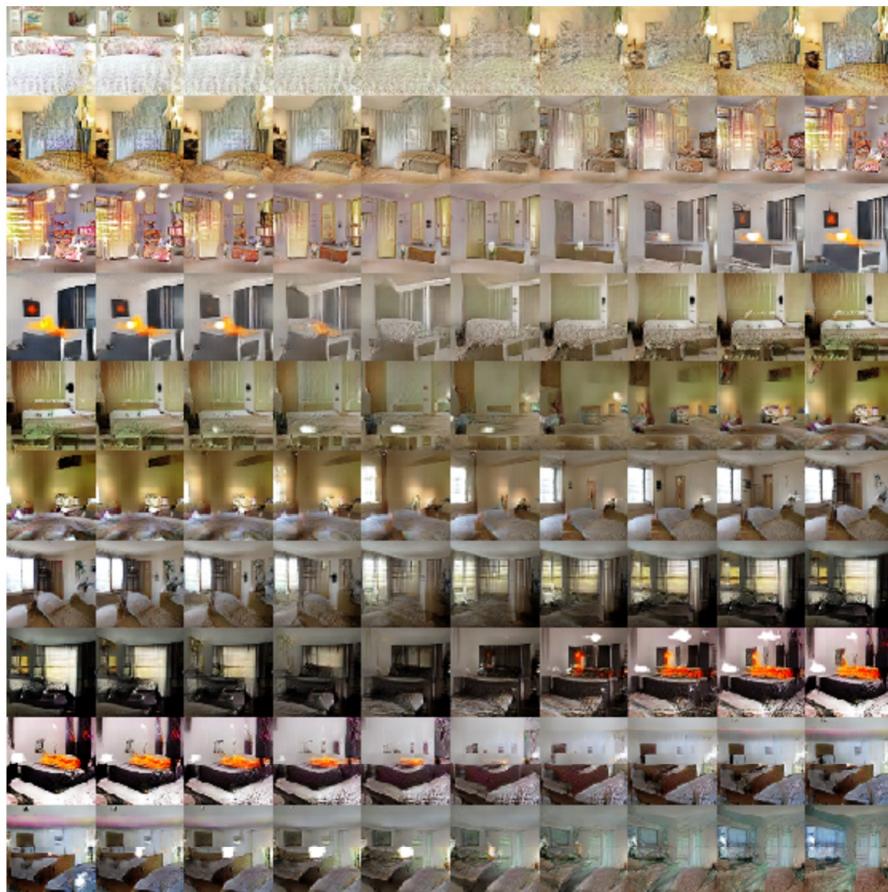
<https://github.com/hwalsuklee/tensorflow-generative-model-collections>

<https://github.com/znxlwm/pytorch-generative-model-collections>

| Name    | Paper Link            | Value Function   |
|---------|-----------------------|--|
| GAN     | <a href="#">Arxiv</a> | $L_D^{GAN} = E[\log(D(x))] + E[\log(1 - D(G(z)))]$<br>$L_G^{GAN} = E[\log(D(G(z)))]$   |
| LSGAN   | <a href="#">Arxiv</a> | $L_D^{LSGAN} = E[(D(x) - 1)^2] + E[D(G(z))^2]$<br>$L_G^{LSGAN} = E[(D(G(z)) - 1)^2]$   |
| WGAN    | <a href="#">Arxiv</a> | $L_D^{WGAN} = E[D(x)] - E[D(G(z))]$<br>$L_G^{WGAN} = E[D(G(z))]$<br>$W_D \leftarrow clip\_by\_value(W_D, -0.01, 0.01)$                     |
| WGAN-GP | <a href="#">Arxiv</a> | $L_D^{WGAN\_GP} = L_D^{WGAN} + \lambda E[ \nabla D(\alpha x - (1 - \alpha G(z)))  - 1]^2$<br>$L_G^{WGAN\_GP} = L_G^{WGAN}$                 |
| DRAGAN  | <a href="#">Arxiv</a> | $L_D^{DRAGAN} = L_D^{GAN} + \lambda E[ \nabla D(\alpha x - (1 - \alpha x_p))  - 1]^2$<br>$L_G^{DRAGAN} = L_G^{GAN}$                        |
| CGAN    | <a href="#">Arxiv</a> | $L_D^{CGAN} = E[\log(D(x, c))] + E[\log(1 - D(G(z), c))]$<br>$L_G^{CGAN} = E[\log(D(G(z), c))]$  |
| infoGAN | <a href="#">Arxiv</a> | $L_{D,Q}^{InfoGAN} = L_D^{GAN} - \lambda L_I(c, c')$<br>$L_G^{InfoGAN} = L_G^{GAN} - \lambda L_I(c, c')$                                   |
| ACGAN   | <a href="#">Arxiv</a> | $L_{D,Q}^{ACGAN} = L_D^{GAN} + E[P(class = c x)] + E[P(class = c G(z))]$<br>$L_G^{ACGAN} = L_G^{GAN} + E[P(class = c G(z))]$               |
| EBGAN   | <a href="#">Arxiv</a> | $L_D^{EBGAN} = D_{AE}(x) + \max(0, m - D_{AE}(G(z)))$<br>$L_G^{EBGAN} = D_{AE}(G(z)) + \lambda \cdot PT$                                   |
| BEGAN   | <a href="#">Arxiv</a> | $L_D^{BEGAN} = D_{AE}(x) - k_t D_{AE}(G(z))$<br>$L_G^{BEGAN} = D_{AE}(G(z))$<br>$k_{t+1} = k_t + \lambda(\gamma D_{AE}(x) - D_{AE}(G(z)))$ |

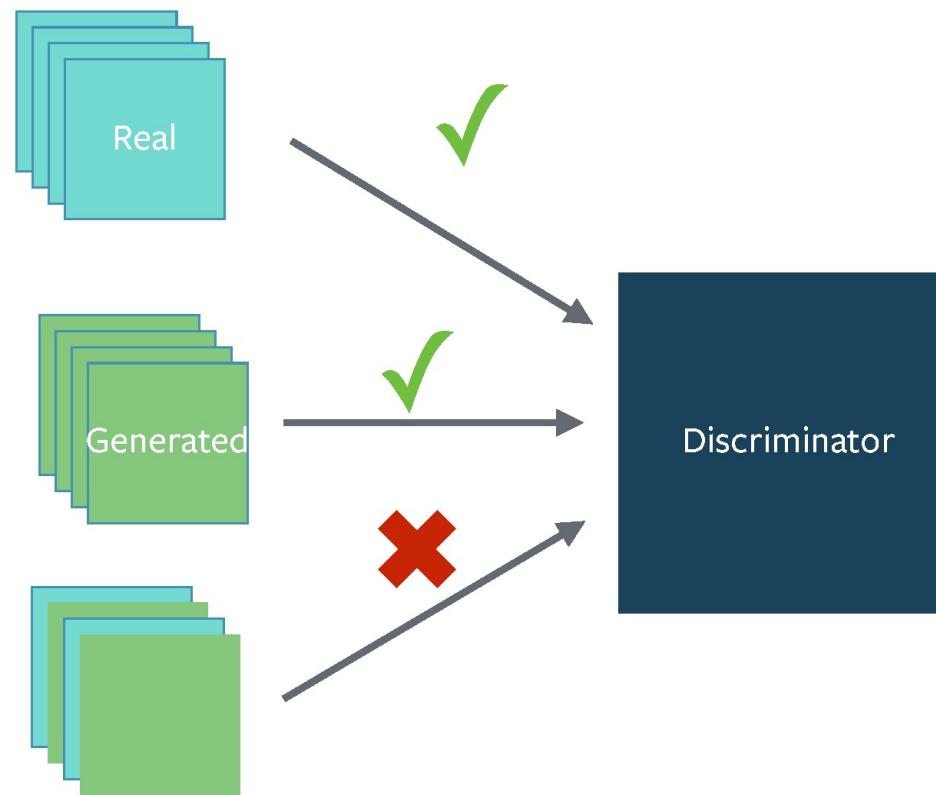
# #3: Use spherical z

- interpolation via great circle
- Tom White “Sampling Generative Networks”
  - <https://arxiv.org/abs/1609.04468>



# #4: BatchNorm

- different batches for real and fake
- when batchnorm is not an option use instance norm



# #5: Avoid Sparse Gradients: ReLU, MaxPool

- the stability of the GAN game suffers
- LeakyReLU (both G and D)
- Downsampling: Average Pooling, Conv2d + stride
- Upsampling: PixelShuffle, ConvTranspose2d + stride
  - PixelShuffle: <https://arxiv.org/abs/1609.05158>

# #6: Soft and Noisy Labels

- Label Smoothing
- making the labels noisy a bit for the discriminator, sometimes

-Salimans et. al. 2016

# #7: Architectures: DCGANs / Hybrids

- DCGAN when you can
- if you can't use DCGANs and no model is stable,
- use a hybrid model: KL + GAN or VAE + GAN
- ResNets from WGAN-gp also work pretty well (but are very slow)
  - [https://github.com/igul222/improved\\_wgan\\_training](https://github.com/igul222/improved_wgan_training)
- Width matters more than Depth

# #8: Stability tricks from RL

- Experience replay
- Things that work for deep deterministic policy gradients
- See Pfau & Vinyals (2016)

# #9: Optimizer: ADAM

- optim.Adam rules!
  - See Radford et. al. 2015
- [MMathieu] Use SGD for discriminator and ADAM for generator

# #10: Use Gradient Penalty

- Regularize the norm of the gradients
  - multiple theories on why this is useful (WGAN-GP, DRAGAN, Stabilizing GANs by Regularization etc.)

# #11: Dont balance via loss statistics (classic GAN)

- Dont try to find a (number of G / number of D) schedule to uncollapse training
- while  $\text{lossD} > X$ :
  - train D
- while  $\text{lossG} > X$ :
  - train G

# #12: If you have labels, use them

- if you have labels available, training the discriminator to also classify the samples: auxillary GANs

# #13: Add noise to inputs, decay over time

- Add some artificial noise to inputs to D (Arjovsky et. al., Huszar, 2016)
  - <http://www.inference.vc/instance-noise-a-trick-for-stabilising-gan-training/>
  - [https://openreview.net/forum?id=Hk4\\_qw5xe](https://openreview.net/forum?id=Hk4_qw5xe)
- adding gaussian noise to every layer of generator (Zhao et. al. EBGAN)
- Improved GANs: OpenAI code also has it (commented out)

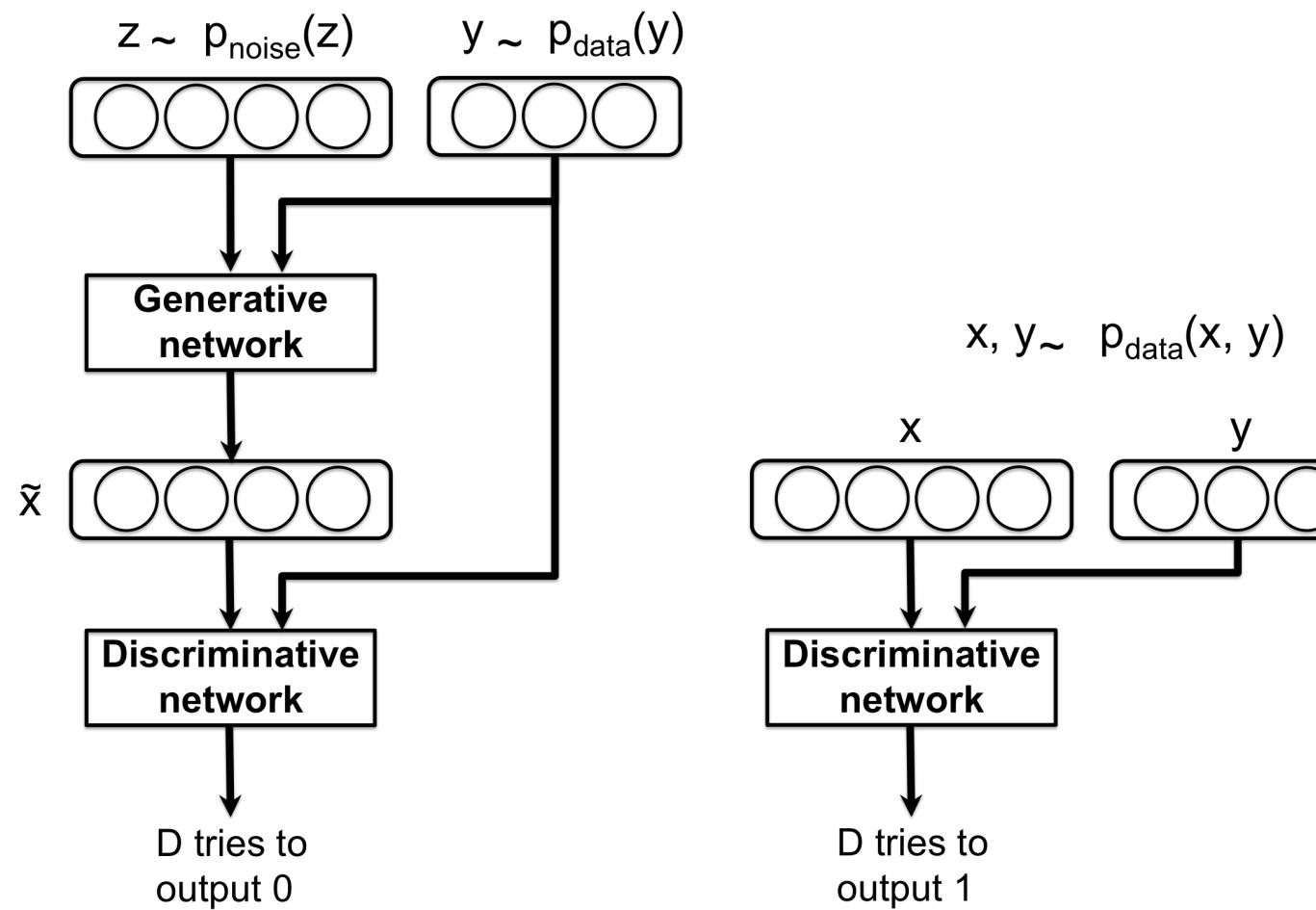
# #14: Train discriminator more

- especially when you have noise
- hard to find a schedule of number of D iterations vs G iterations
- WGAN/WGAN-gp papers suggest 5x D iterations per G iteration

# Conditional GANs

# Conditional generative adversarial networks (CGAN)

- Condition generation on additional info  $y$  (e.g. class label, another image)
- $D$  has to determine if samples are realistic given  $y$



[Mirza and Osindero (2014); Gauthier (2014)]

# #16: Discrete Variables

- Use an Embedding layer
- Add as additional channels to images
- Keep embedding dimensionality low and upsample to match image channel size

# Conclusion

- Model stability is improving
- Theory is improving
- Hacks are a stop-gap

# PROGRESSIVE GROWING OF GANS FOR IMPROVED QUALITY, STABILITY, AND VARIATION

**Tero Karras**

NVIDIA

{tkarras, taila, slaine, jlehtinen}@nvidia.com

**Timo Aila**

NVIDIA

**Samuli Laine**

NVIDIA

**Jaakko Lehtinen**

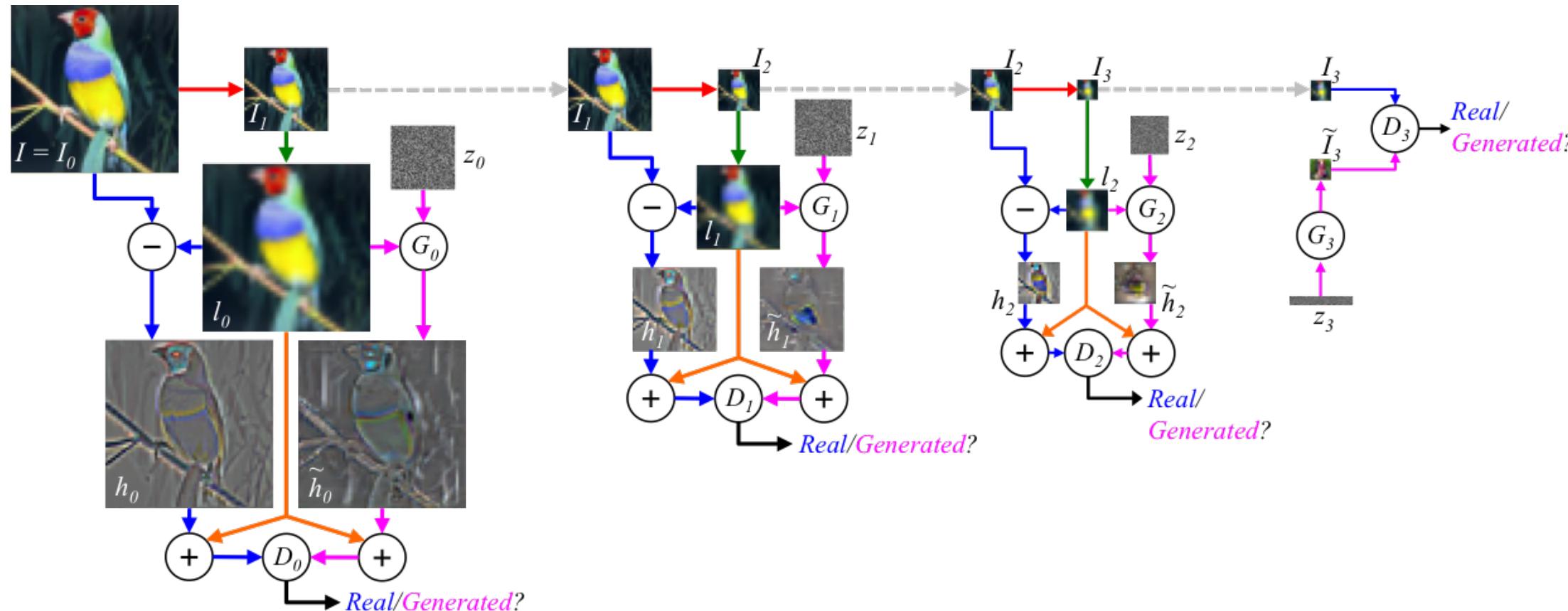
NVIDIA and Aalto University

ICLR 2018

# Prior Work

## Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Emily Denton<sup>1\*</sup>, Soumith Chintala<sup>2\*</sup>,  
Arthur Szlam<sup>2</sup>, Rob Fergus<sup>2</sup>  
NeurIPS 2015



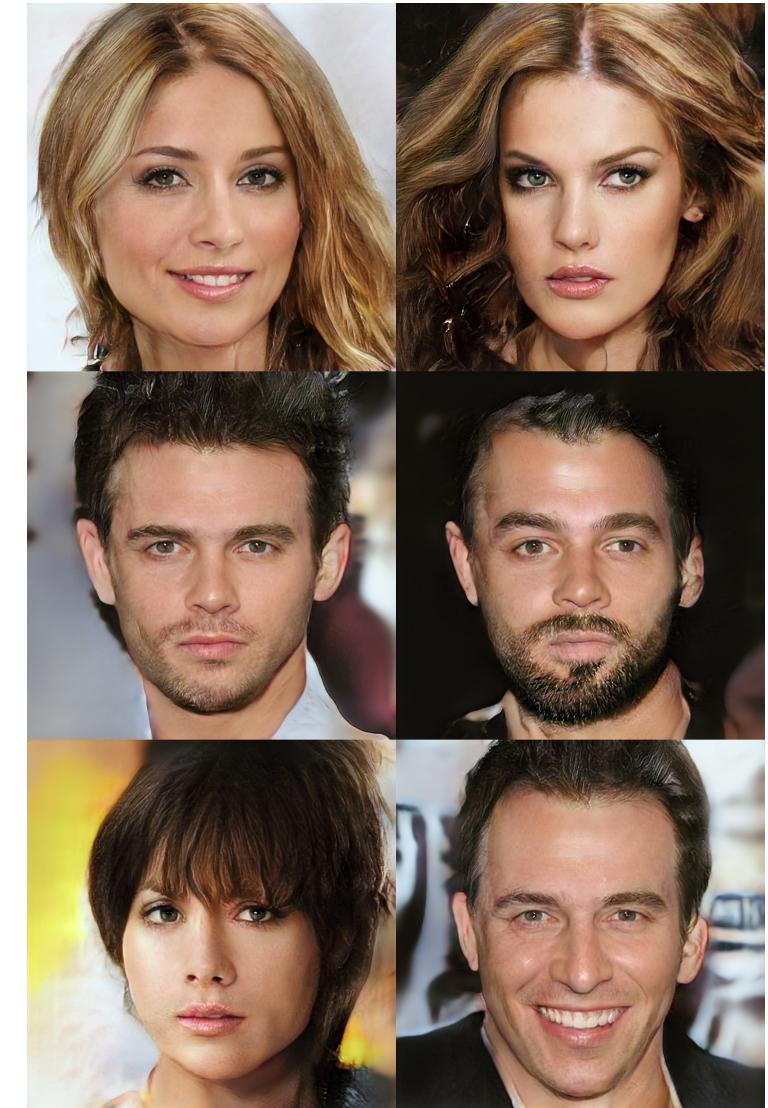
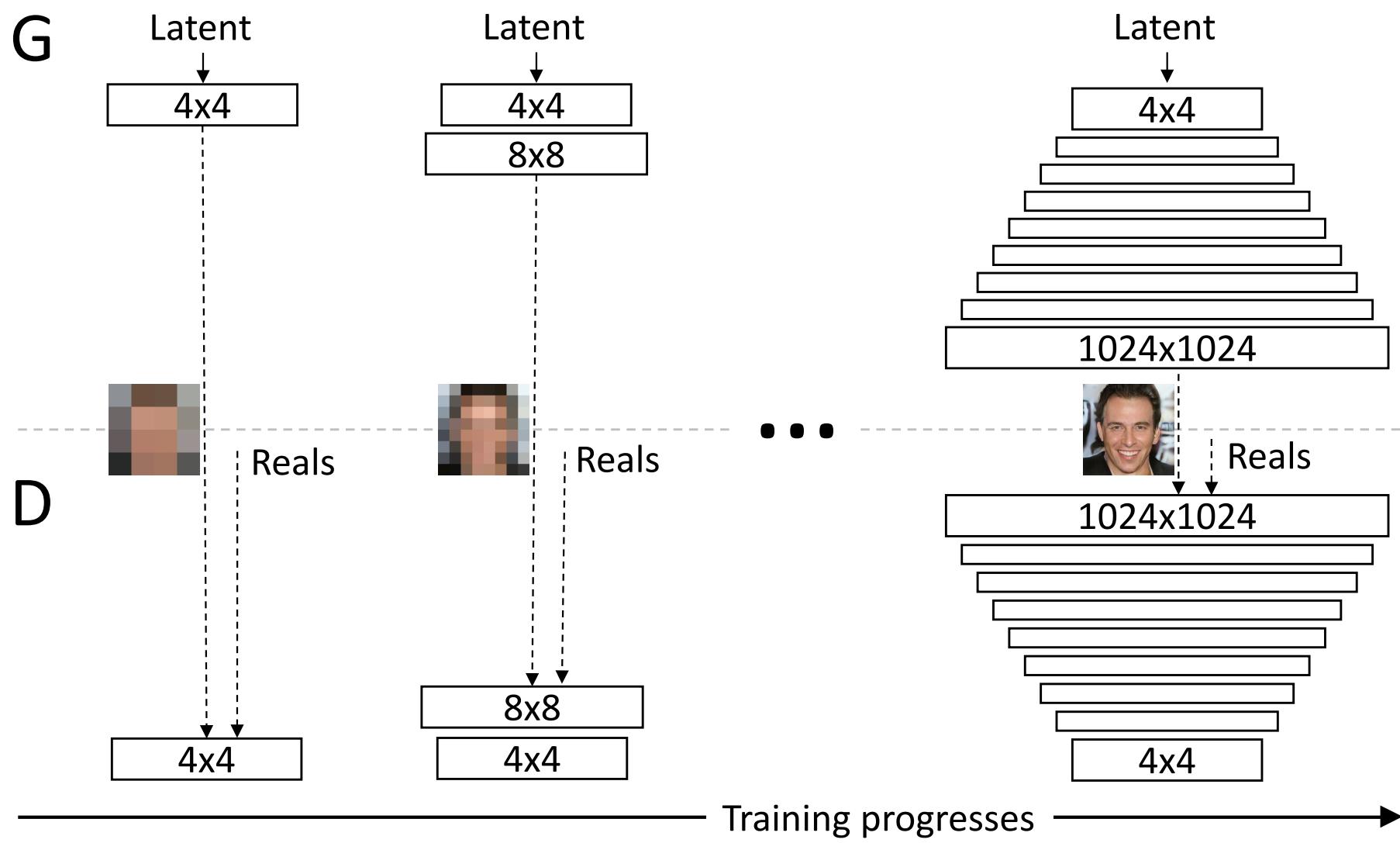
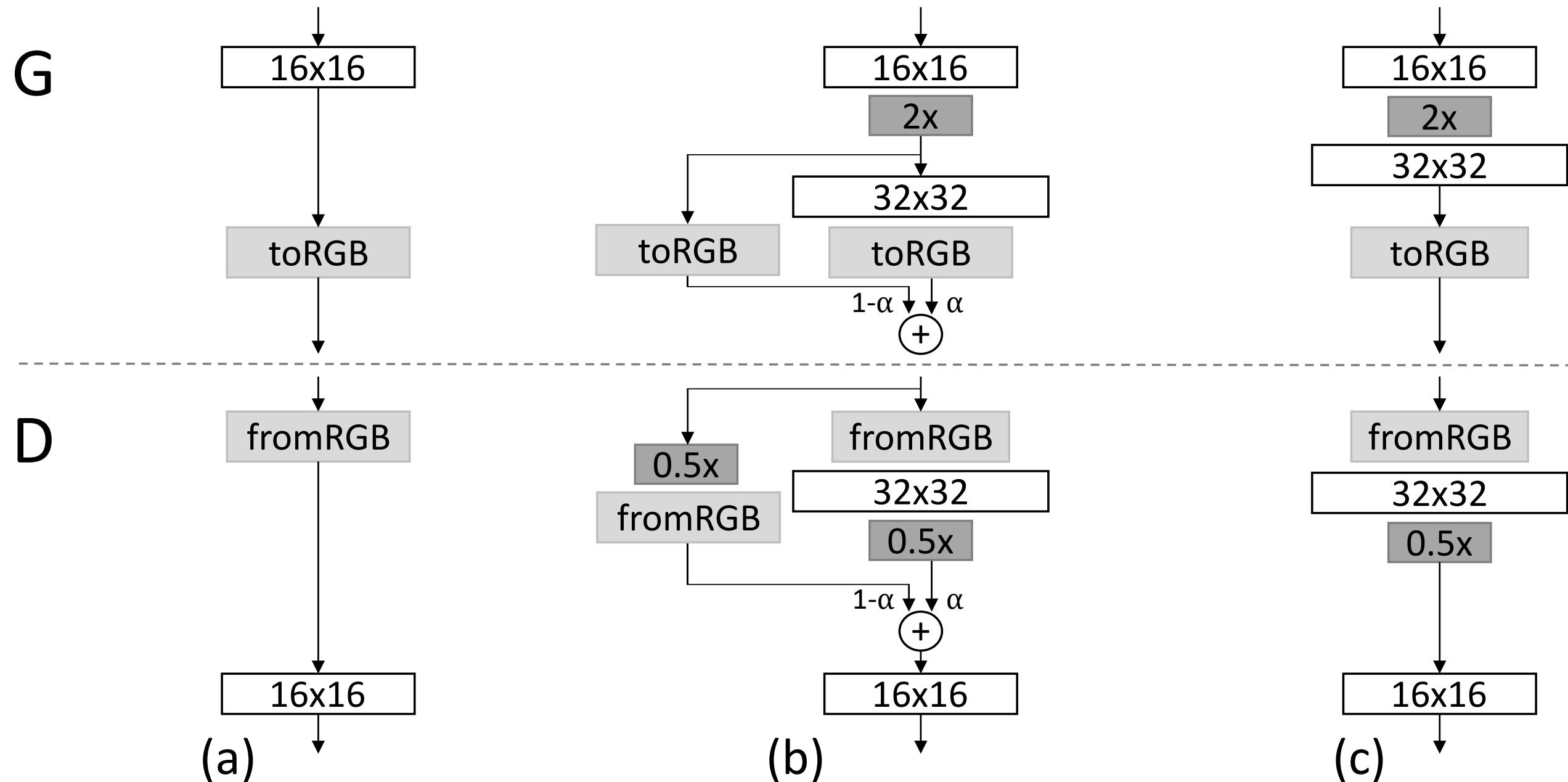


Figure 1: Our training starts with both the generator (**G**) and discriminator (**D**) having a low spatial resolution of  $4 \times 4$  pixels. As the training advances, we incrementally add layers to **G** and **D**, thus increasing the spatial resolution of the generated images. All existing layers 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. On the right we show six example images generated using progressive growing at  $1024 \times 1024$ .

# Smooth blending with scale increase



| Training configuration            | CELEBA                                    |             |             |             |             | LSUN BEDROOM  |   |             |             |             |             |               |
|-----------------------------------|---|-------------|-------------|-------------|-------------|---------------|---|-------------|-------------|-------------|-------------|---------------|
|                                   | Sliced Wasserstein distance $\times 10^3$ |             |             |             |             | MS-SSIM       | Sliced Wasserstein distance $\times 10^3$ |             |             |             |             | MS-SSIM       |
|                                   | 128                                       | 64          | 32          | 16          | Avg         |               | 128                                       | 64          | 32          | 16          | Avg         |               |
| (a) Gulrajani et al. (2017)       | 12.99                                     | 7.79        | 7.62        | 8.73        | 9.28        | 0.2854        | 11.97                                     | 10.51       | 8.03        | 14.48       | 11.25       | <b>0.0587</b> |
| (b) + Progressive growing         | 4.62                                      | <b>2.64</b> | 3.78        | 6.06        | 4.28        | <b>0.2838</b> | 7.09                                      | 6.27        | 7.40        | 9.64        | 7.60        | 0.0615        |
| (c) + Small minibatch             | 75.42                                     | 41.33       | 41.62       | 26.57       | 46.23       | 0.4065        | 72.73                                     | 40.16       | 42.75       | 42.46       | 49.52       | 0.1061        |
| (d) + Revised training parameters | 9.20                                      | 6.53        | 4.71        | 11.84       | 8.07        | 0.3027        | 7.39                                      | 5.51        | 3.65        | 9.63        | 6.54        | 0.0662        |
| (e*) + Minibatch discrimination   | 10.76                                     | 6.28        | 6.04        | 16.29       | 9.84        | 0.3057        | 10.29                                     | 6.22        | 5.32        | 11.88       | 8.43        | 0.0648        |
| (e) Minibatch stddev              | 13.94                                     | 5.67        | 2.82        | 5.71        | 7.04        | 0.2950        | 7.77                                      | 5.23        | 3.27        | 9.64        | 6.48        | 0.0671        |
| (f) + Equalized learning rate     | 4.42                                      | 3.28        | 2.32        | 7.52        | 4.39        | 0.2902        | <b>3.61</b>                               | 3.32        | <b>2.71</b> | 6.44        | 4.02        | 0.0668        |
| (g) + Pixelwise normalization     | <b>4.06</b>                               | 3.04        | <b>2.02</b> | <b>5.13</b> | <b>3.56</b> | 0.2845        | 3.89                                      | <b>3.05</b> | 3.24        | <b>5.87</b> | <b>4.01</b> | 0.0640        |
| (h) Converged                     | 2.42                                      | 2.17        | 2.24        | 4.99        | 2.96        | 0.2828        | 3.47                                      | 2.60        | 2.30        | 4.87        | 3.31        | 0.0636        |

Table 1: Sliced Wasserstein distance (SWD) between the generated and training images (Section 5) and multi-scale structural similarity (MS-SSIM) among the generated images for several training setups at  $128 \times 128$ . For SWD, each column represents one level of the Laplacian pyramid, and the last one gives an average of the four distances.



Figure 3: (a) – (g) CELEBA examples corresponding to rows in Table 1. These are intentionally non-converged. (h) Our converged result. Notice that some images show aliasing and some are not sharp – this is a flaw of the dataset, which the model learns to replicate faithfully.



Figure 5:  $1024 \times 1024$  images generated using the CELEBA-HQ dataset. See Appendix F for a larger version of each image, which includes a color bar for the generated images.



Mao et al. (2016b) ( $128 \times 128$ )

Gulrajani et al. (2017) ( $128 \times 128$ )

Our ( $256 \times 256$ )



POTTEDPLANT

HORSE

SOFA

BUS

CHURCHOUTDOOR

BICYCLE

TVMONITOR

# Nearest-Neighbor Sanity Check



# A Style-Based Generator Architecture for Generative Adversarial Networks

Tero Karras  
NVIDIA

[tkarras@nvidia.com](mailto:tkarras@nvidia.com)

Samuli Laine  
NVIDIA

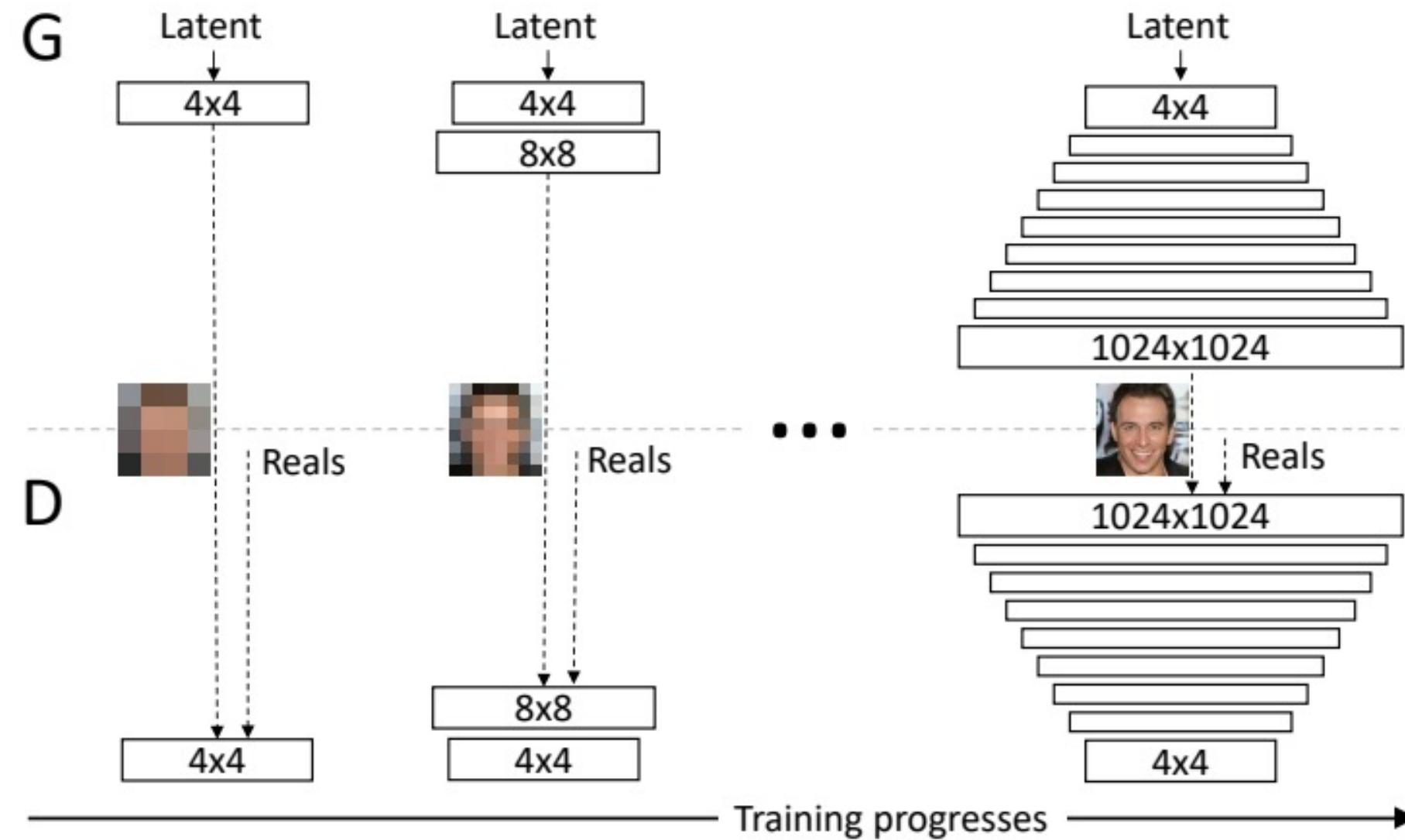
[slaine@nvidia.com](mailto:slaine@nvidia.com)

Timo Aila  
NVIDIA

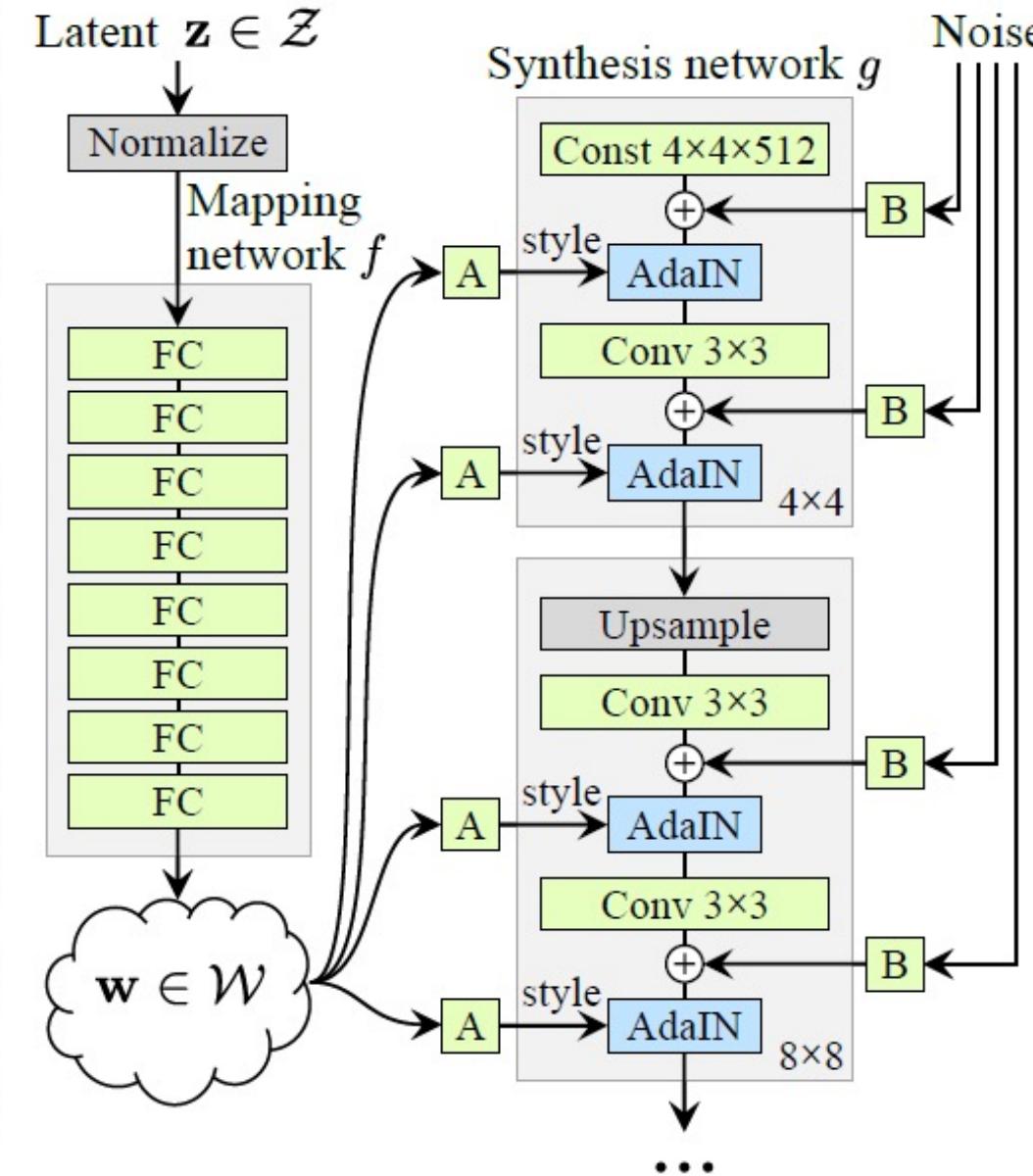
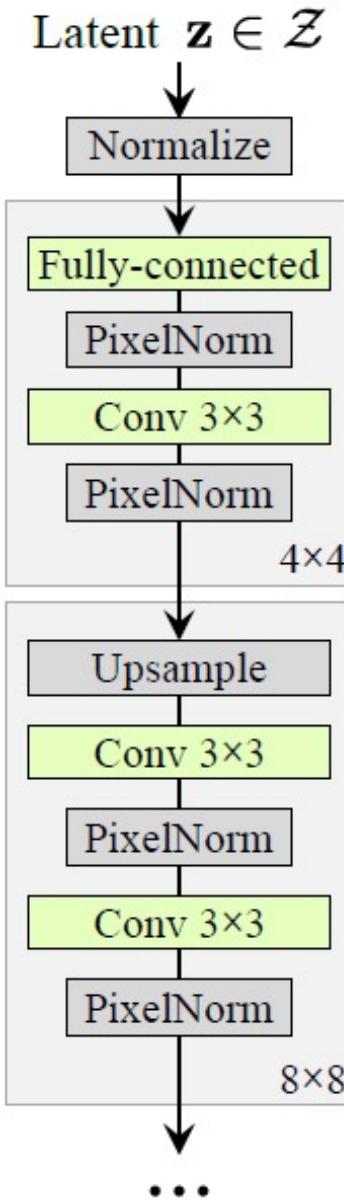
[taila@nvidia.com](mailto:taila@nvidia.com)

<https://arxiv.org/pdf/1812.04948.pdf>  
CVPR 2019

# Baseline Progressive GAN



# Style-based generator

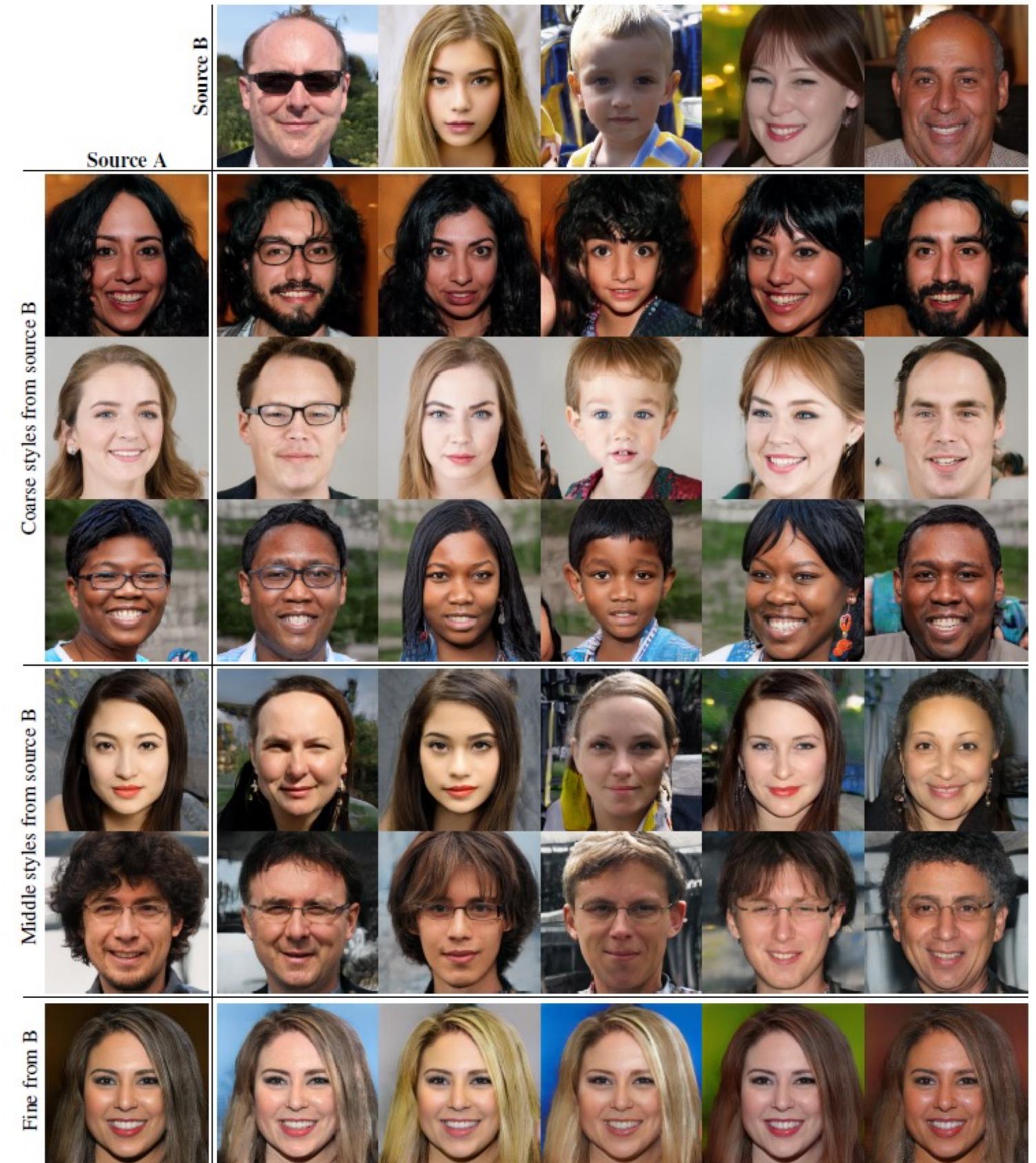


AdaIN: adaptive instance normalization

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

A: a learned affine transform

B: learned per-channel scaling factors



# Stochastic variation

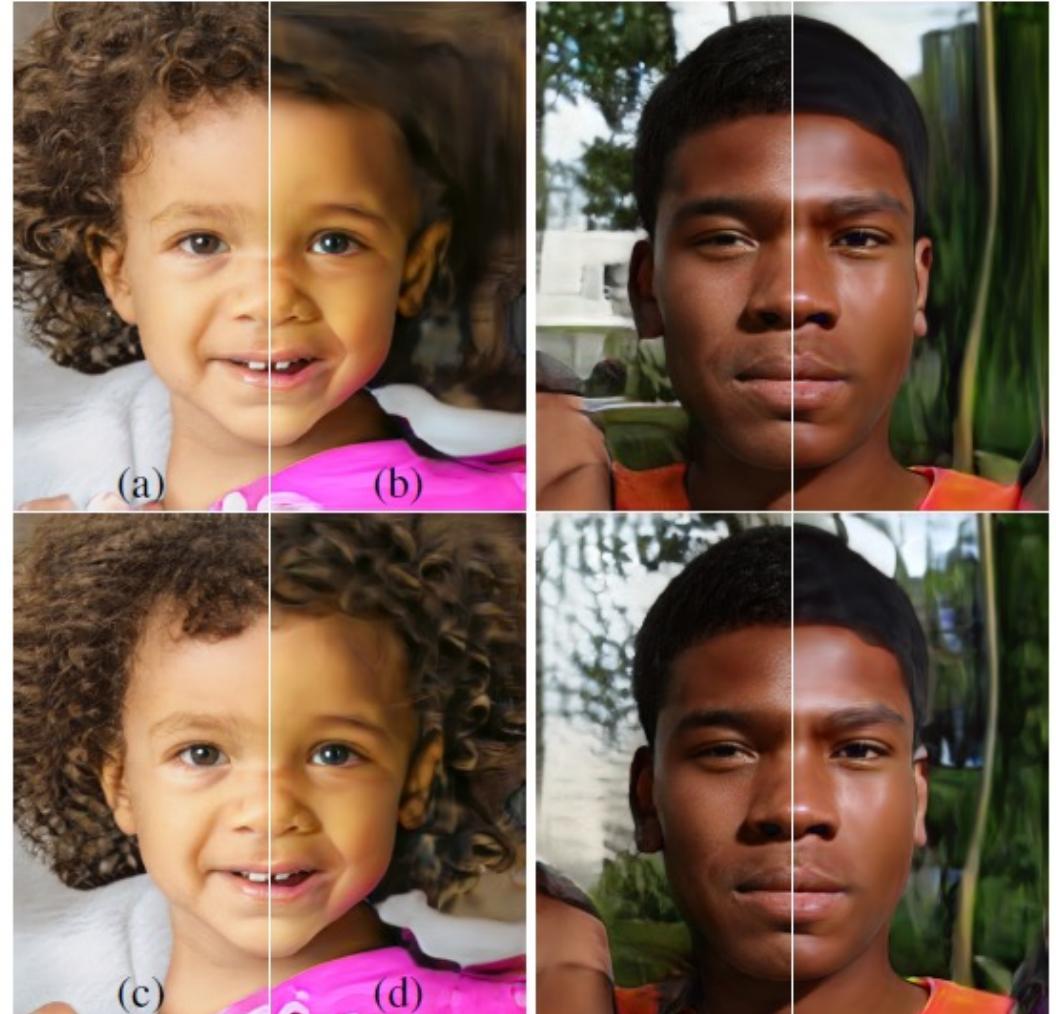
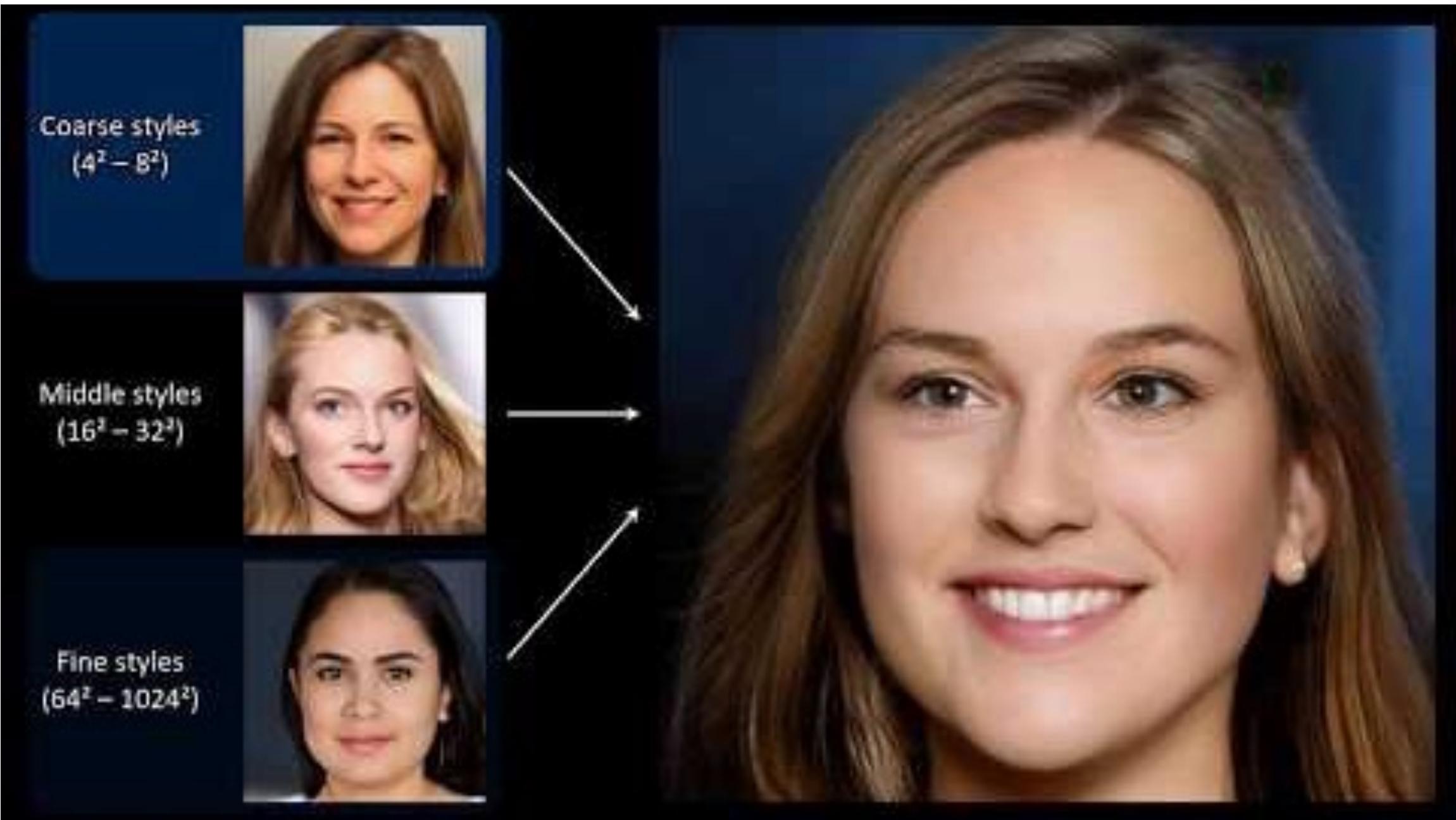


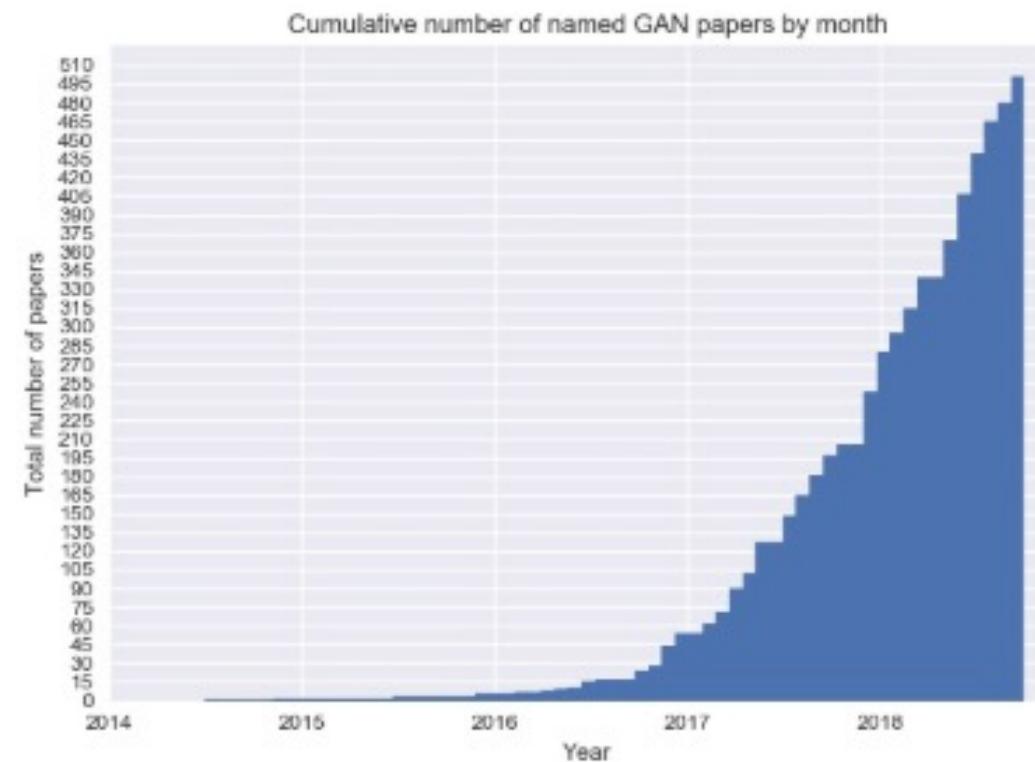
Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only ( $64^2 - 1024^2$ ). (d) Noise in coarse layers only ( $4^2 - 32^2$ ). We can see that the artificial omission of noise leads to featureless “painterly” look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

# StyleGAN



# The GAN Zoo

<https://github.com/hindupuravinash/the-gan-zoo>



- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling ([github](#))
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction ([github](#))
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning ([github](#))
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generative Adversarial Networks ([github](#))
- ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- ACGAN - Coverless Information Hiding Based on Generative adversarial networks
- acGAN - On-line Adaptative Curriculum Learning for GANs
- ActuAL - ACTuAL: Actor-Critic Under Adversarial Learning
- AdaGAN - AdaGAN: Boosting Generative Models
- Adaptive GAN - Customizing an Adversarial Example Generator with Class-Conditional GANs
- AdvEntuRe - AdvEntuRe: Adversarial Training for Textual Entailment with Knowledge-Guided Examples
- AdvGAN - Generating adversarial examples with adversarial networks
- AE-GAN - AE-GAN: adversarial eliminating with GAN
- AE-OT - Latent Space Optimal Transport for Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AF-DCGAN - AF-DCGAN: Amplitude Feature Deep Convolutional GAN for Fingerprint Construction in Indoor Localization System
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AIM - Generating Informative and Diverse Conversational Responses via Adversarial Information Maximization
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference ([github](#))

# Unpaired Image-to-Image Translation with CycleGAN

Jun-Yan Zhu and Taesung Park

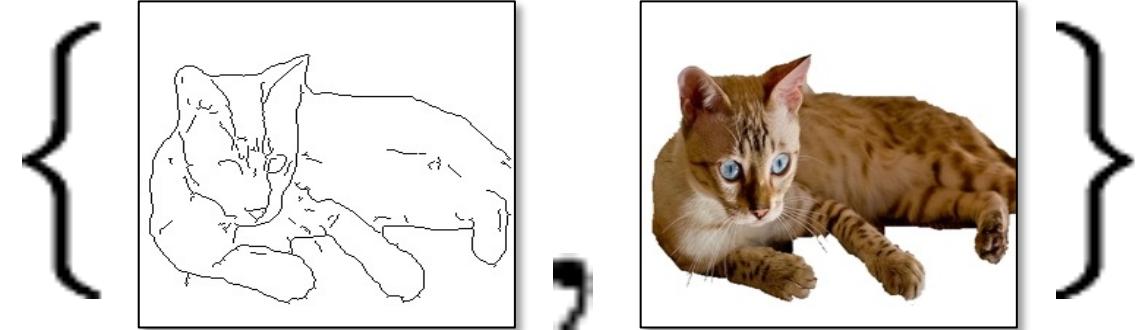
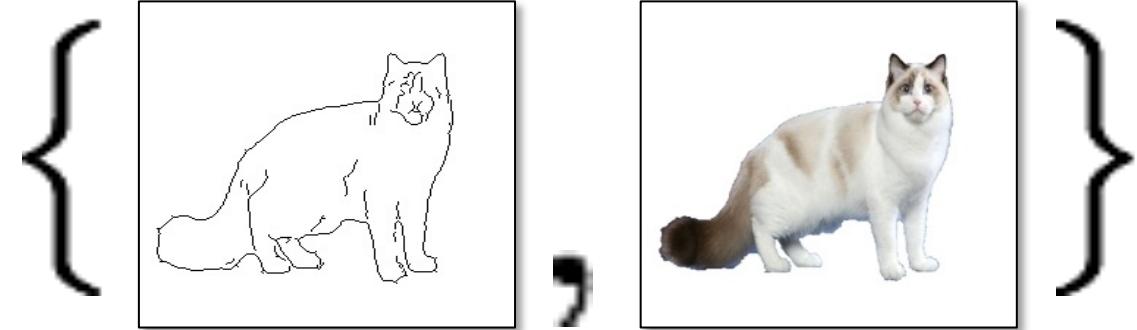
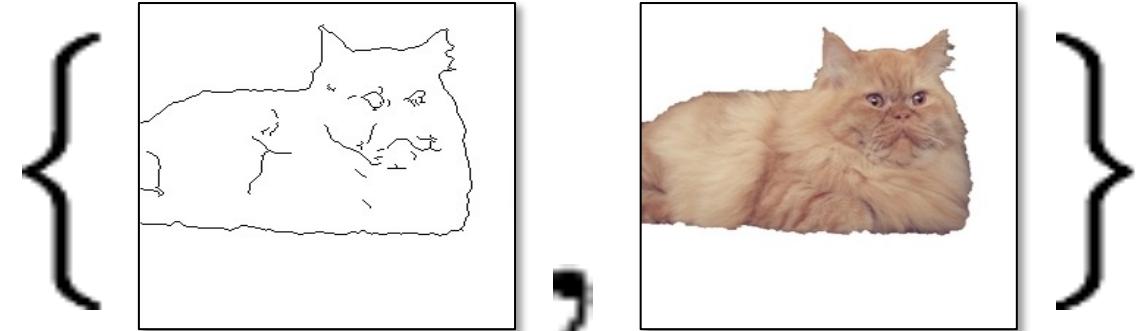
Joint work with Phillip Isola and Alexei A. Efros



# Paired

$x_i$

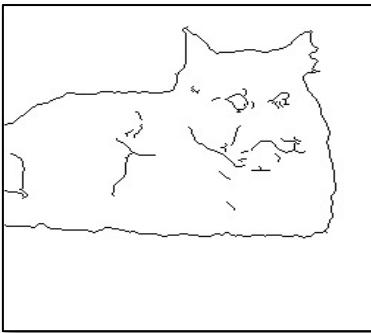
$y_i$



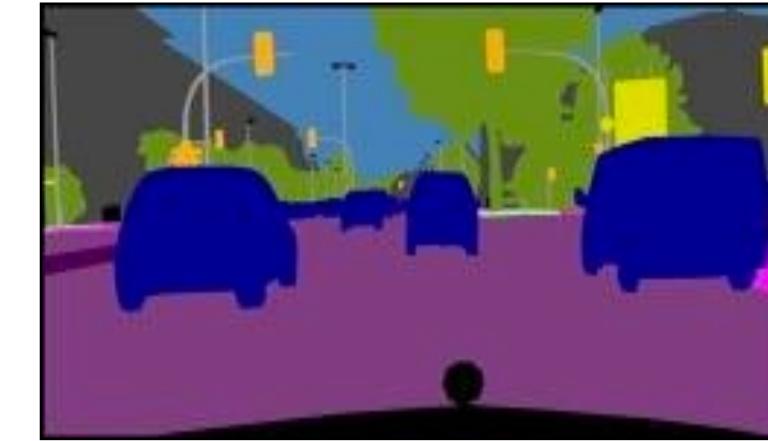
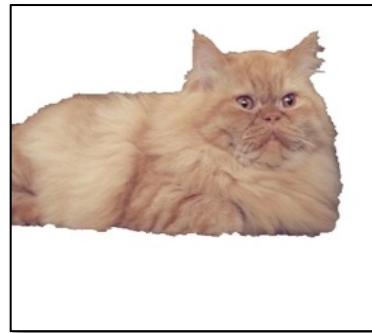
⋮

# Paired

$x_i$



$y_i$



Label  $\leftrightarrow$  photo: per-pixel labeling



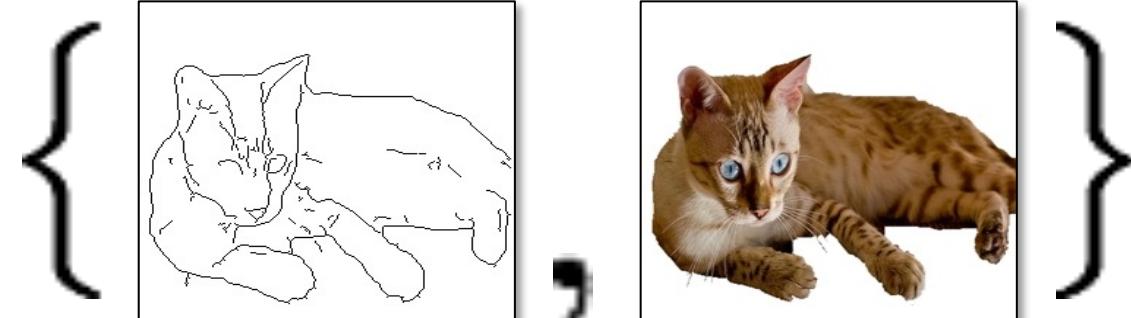
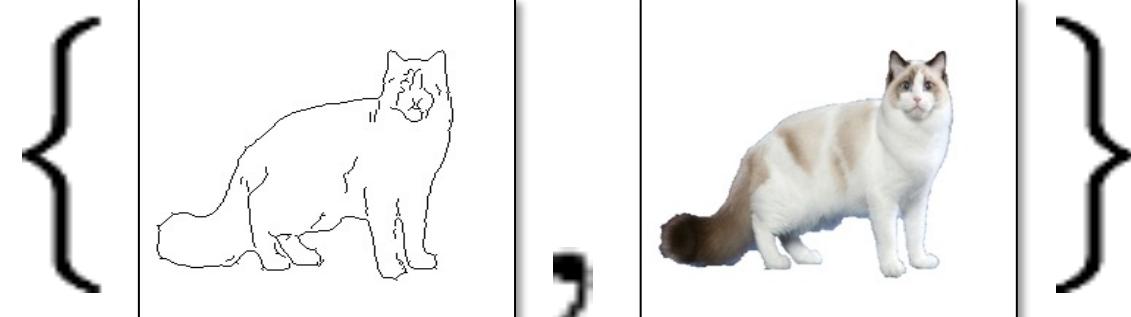
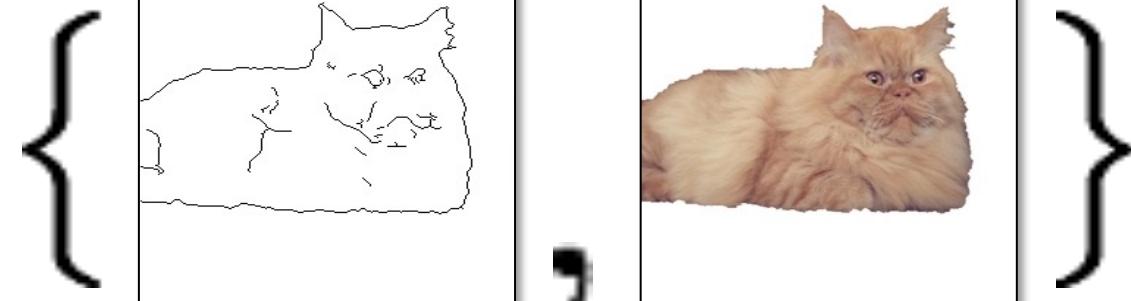
Horse  $\leftrightarrow$  zebra: how to get zebras?

- Expensive to collect pairs.
- Impossible in many scenarios.

# Paired

$x_i$

$y_i$

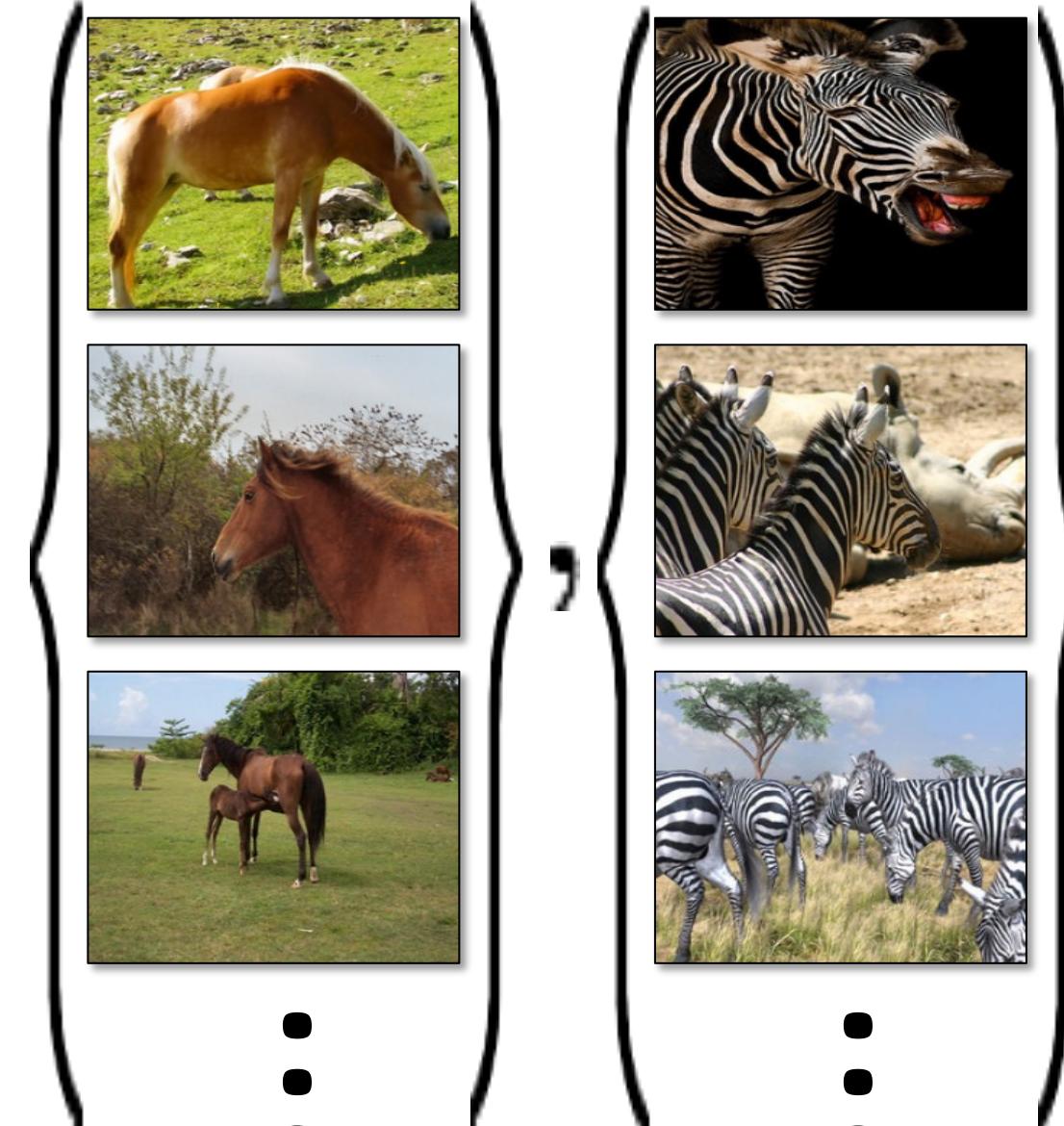


⋮

# Unpaired

$X$

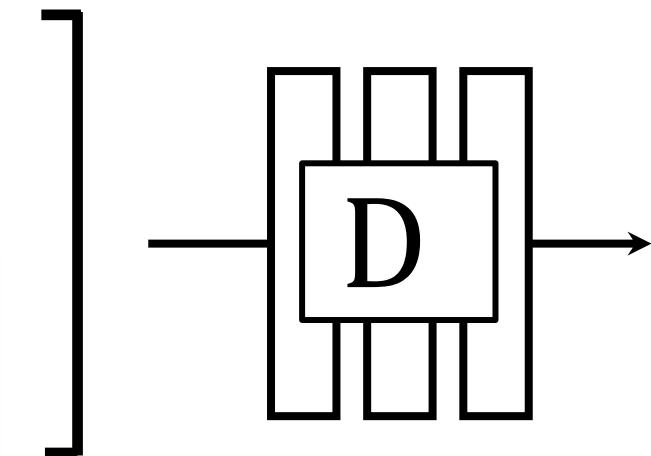
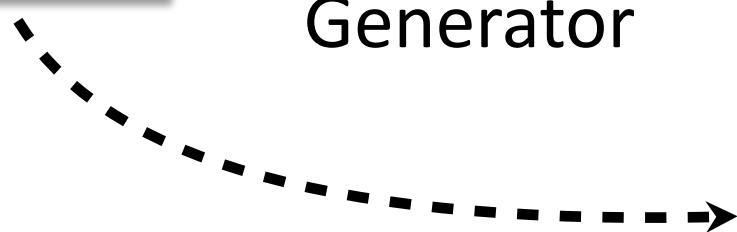
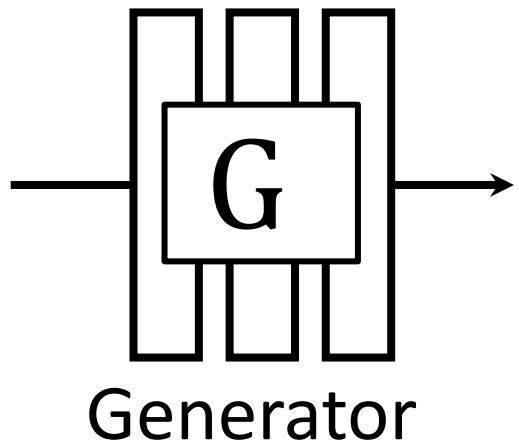
$Y$



$X$



$G(x)$

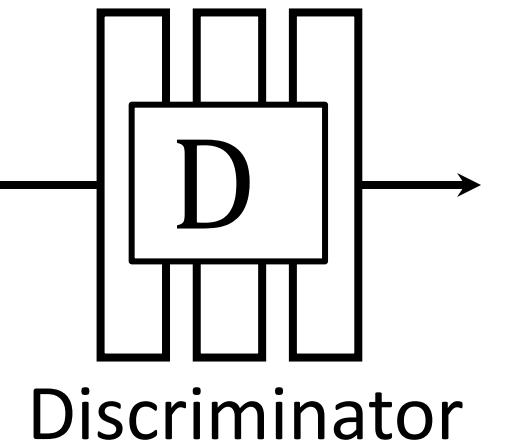
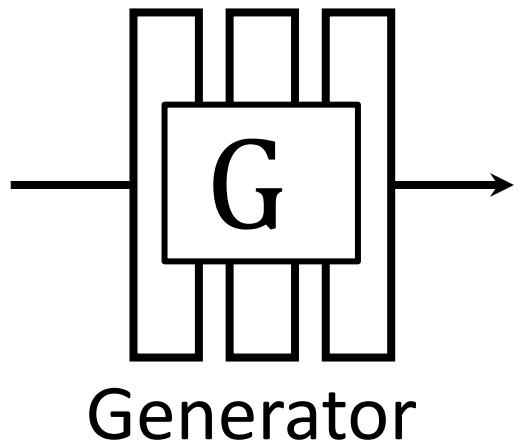


No input-output pairs!

X



$G(x)$

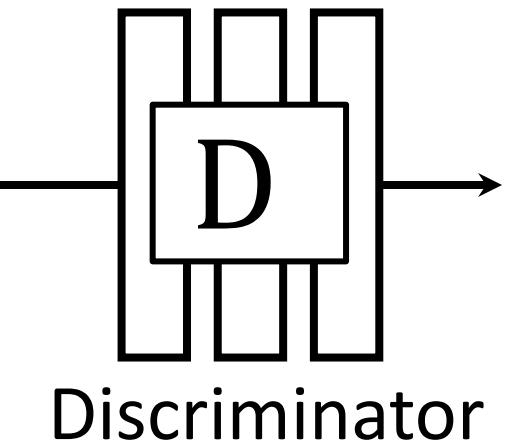
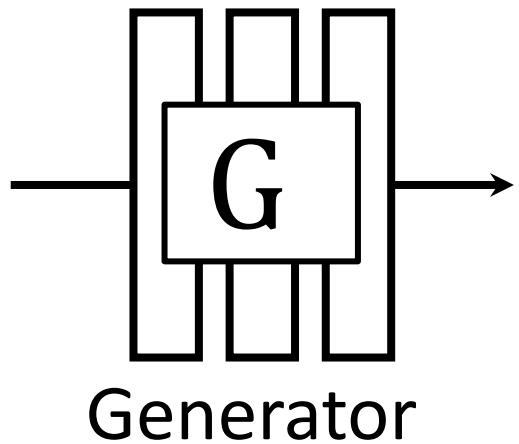


Real!

$x$

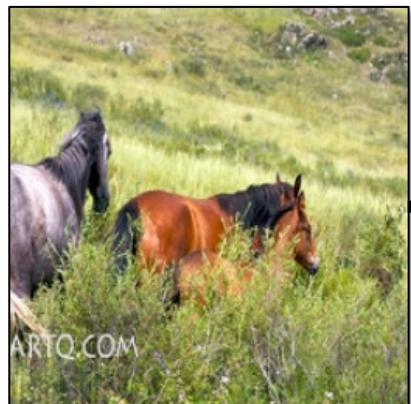


$G(x)$



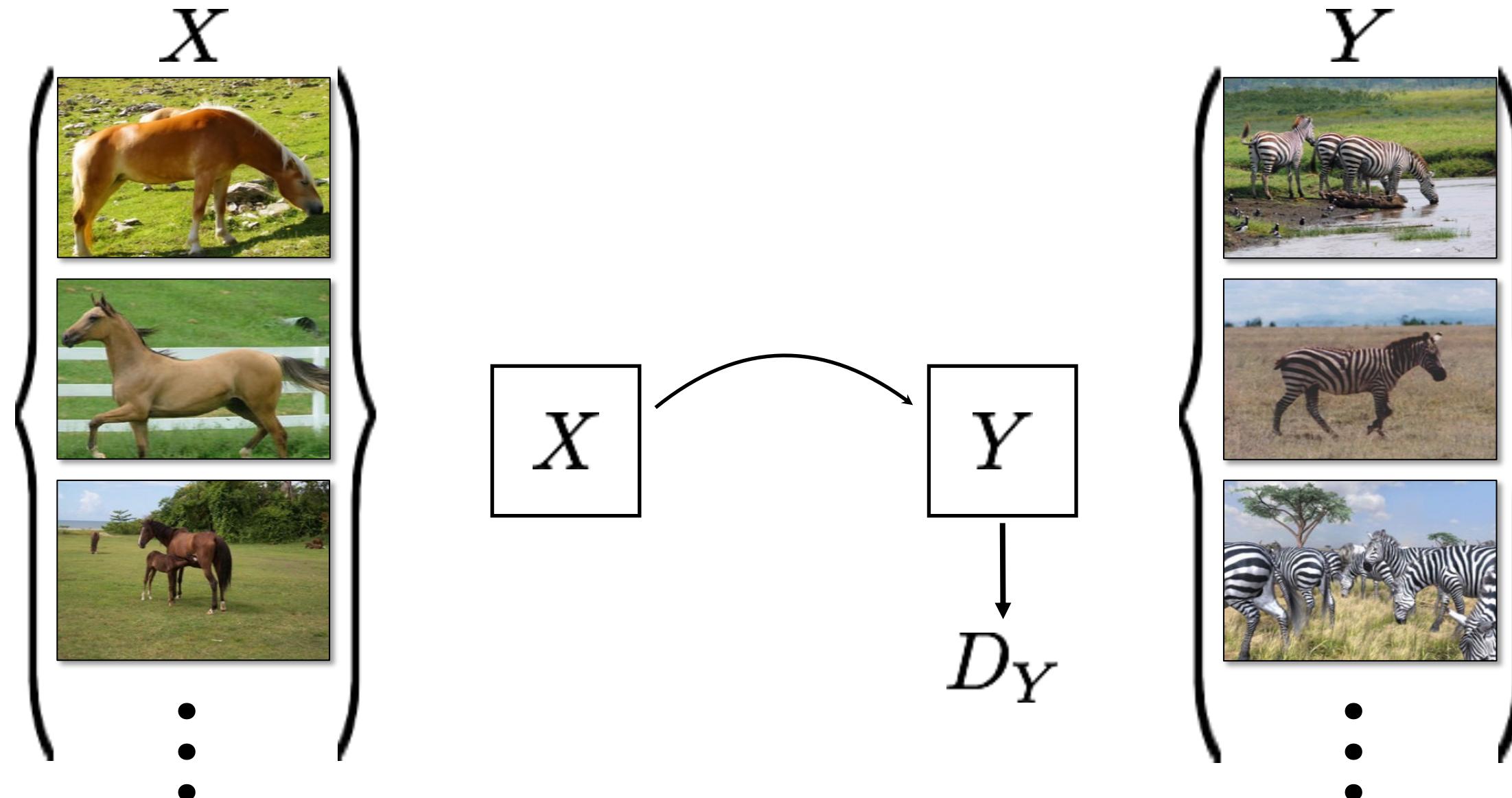
Real too!

GANs do not force output to  
correspond to input



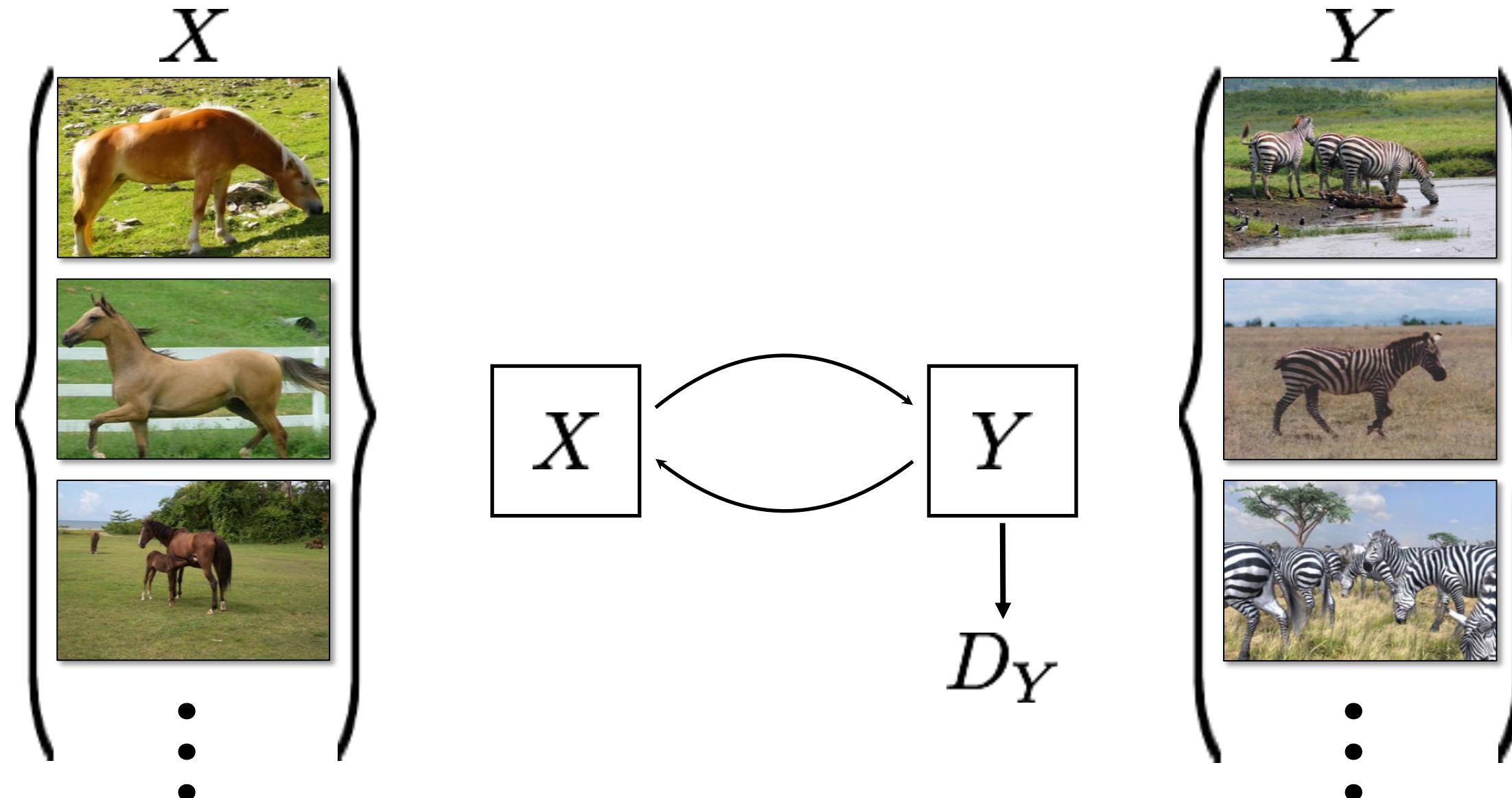
mode collapse!

# Cycle-Consistent Adversarial Networks



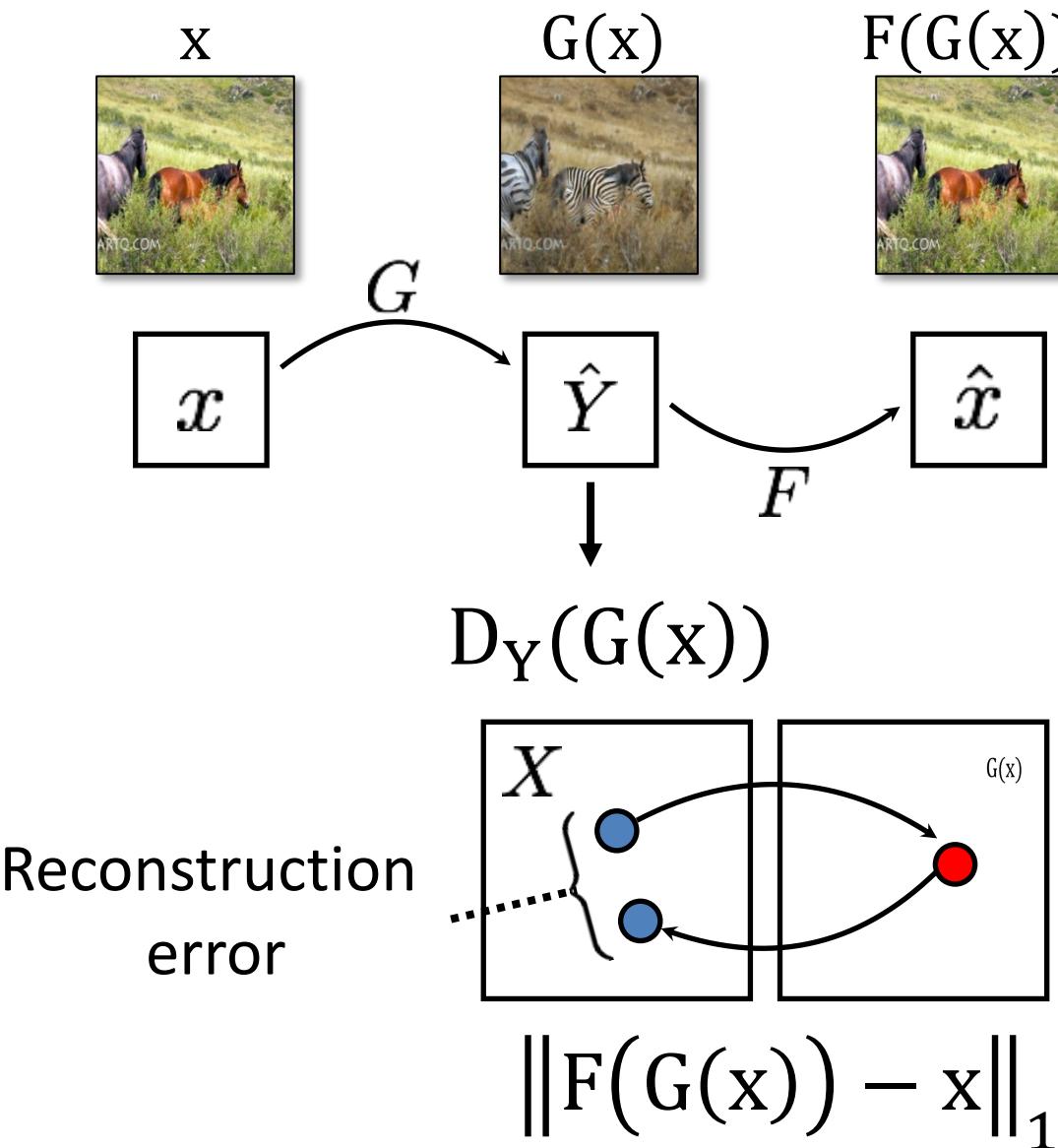
[Zhu\*, Park\*, Isola, and Efros, ICCV 2017]

# Cycle-Consistent Adversarial Networks



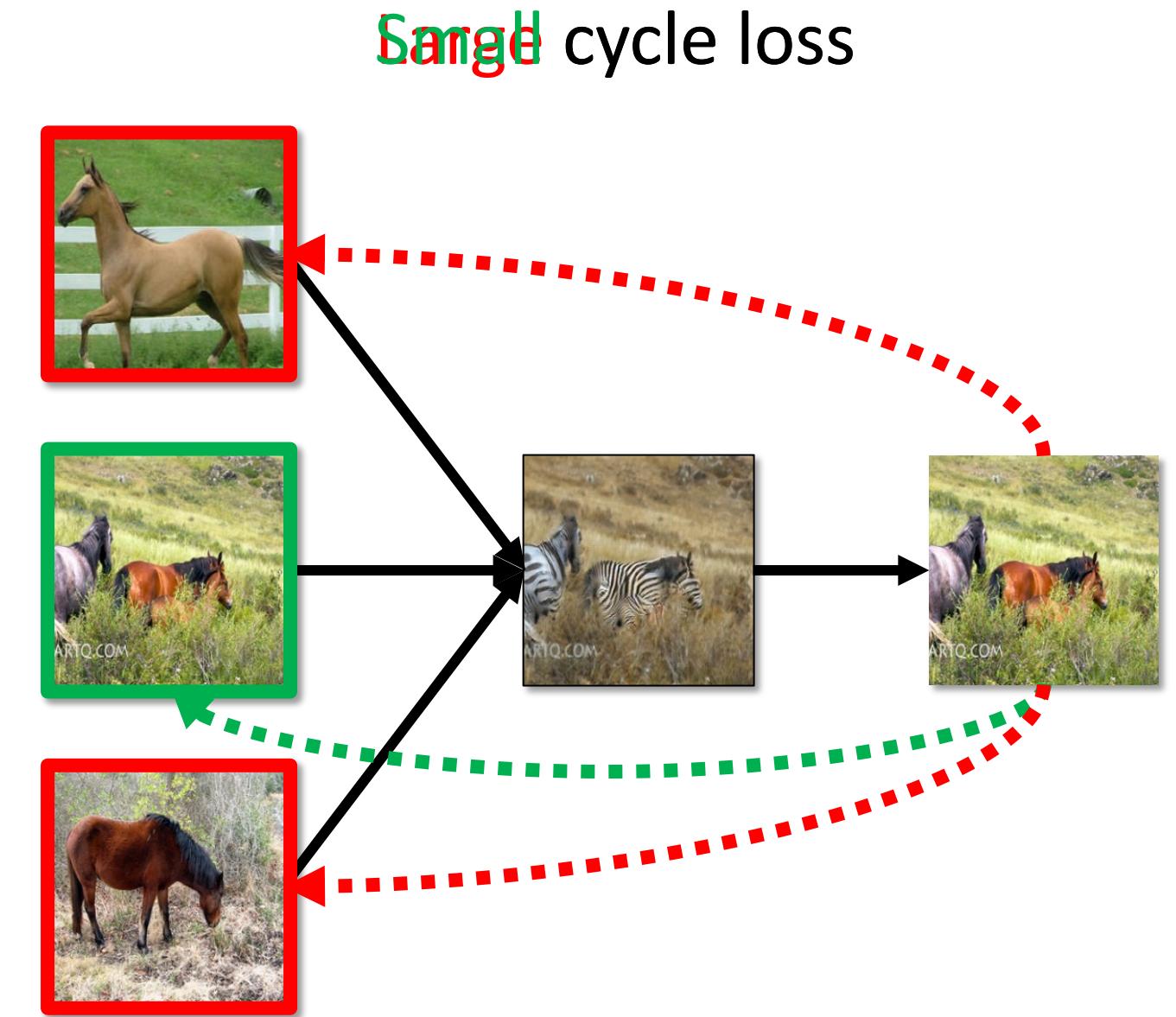
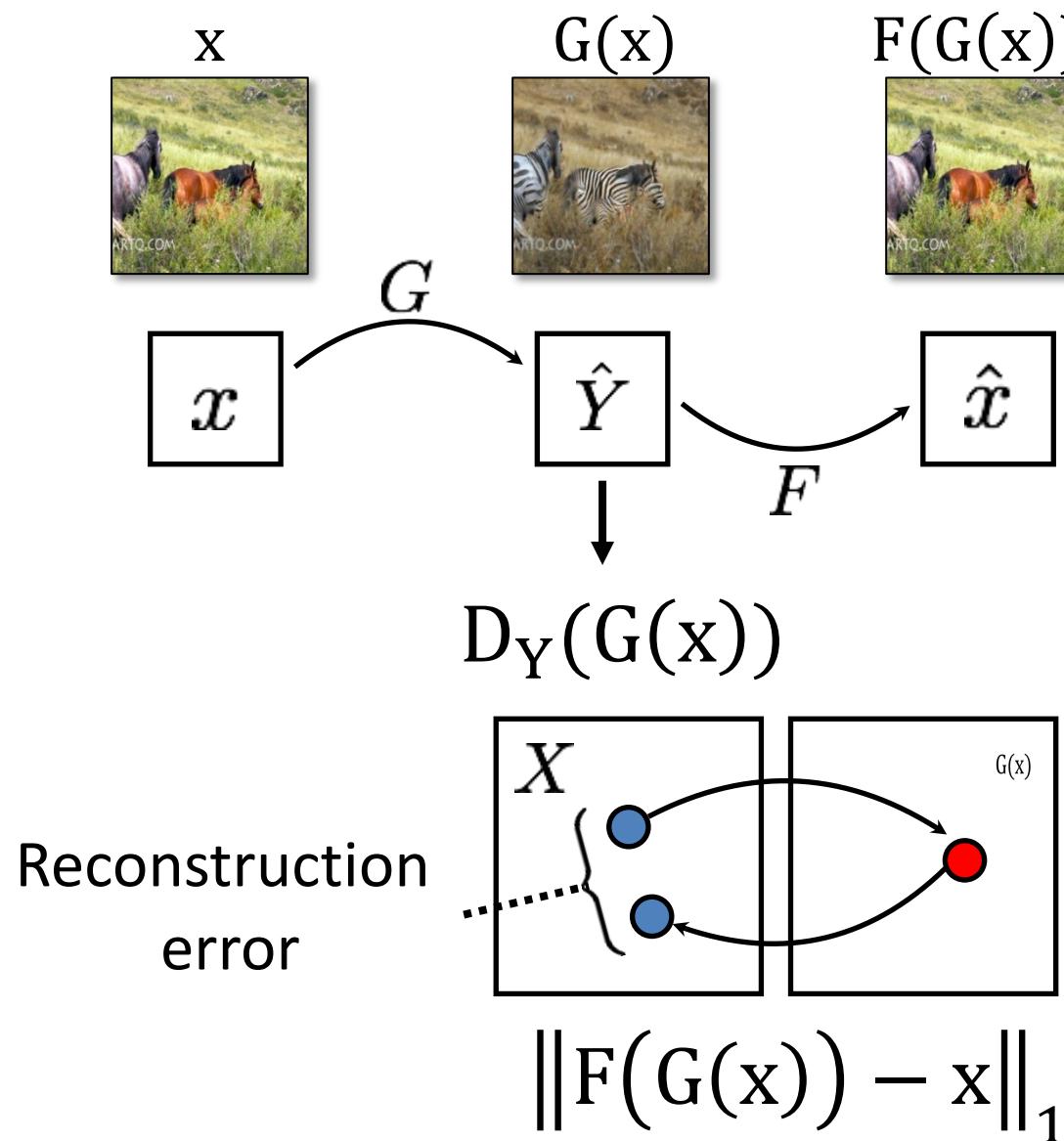
[Zhu\*, Park\*, Isola, and Efros, ICCV 2017]

# Cycle-Consistent Adversarial Networks



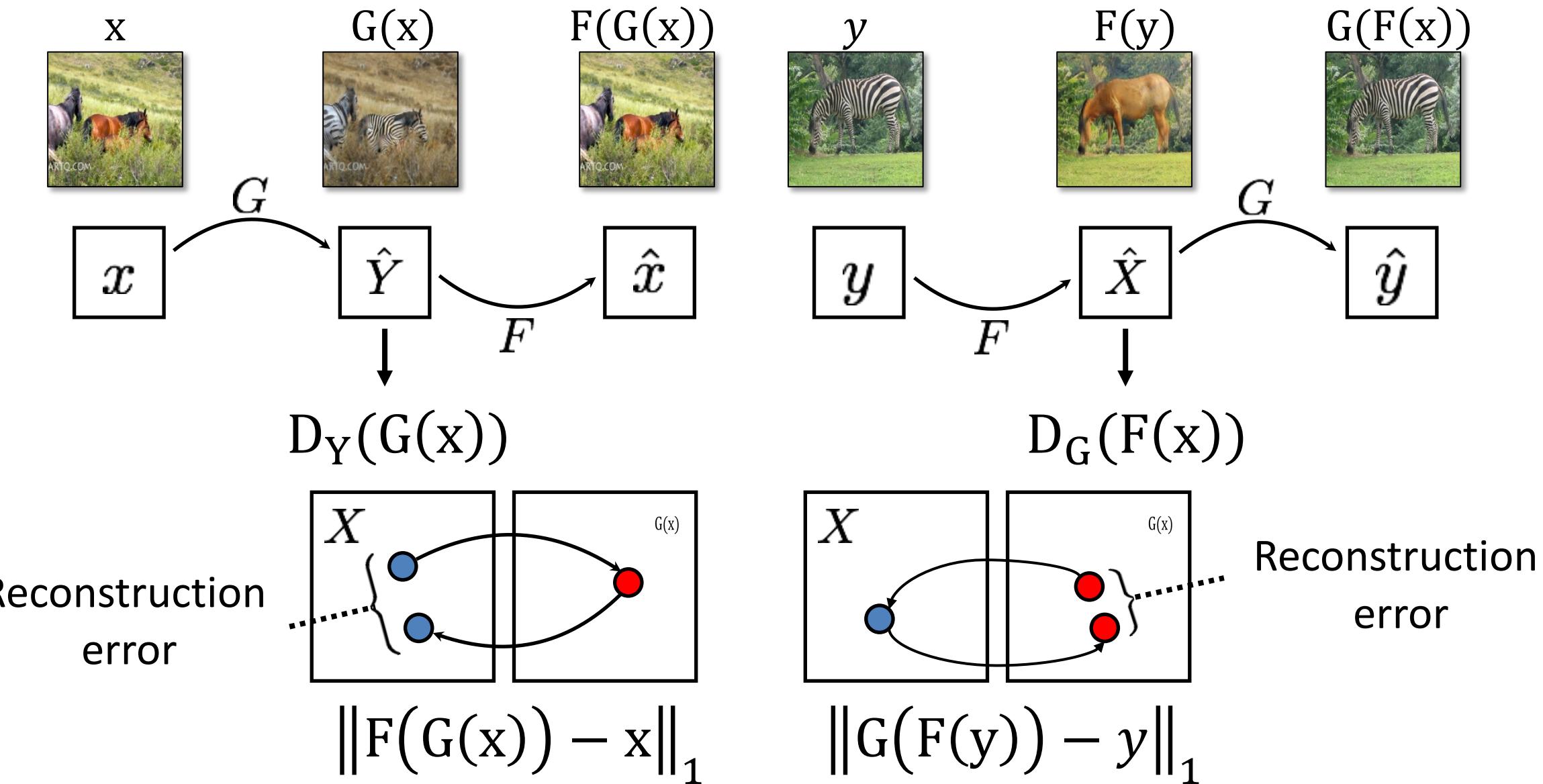
[Zhu\*, Park\*, Isola, and Efros, ICCV 2017]

# Cycle Consistency Loss



[Zhu\*, Park\*, Isola, and Efros, ICCV 2017]

# Cycle Consistency Loss



See similar formulations [Yi et al. 2017], [Kim et al. 2017] [Zhu\*, Park\*, Isola, and Efros, ICCV 2017]

# Results





# Collection Style Transfer



Photograph  
@ Alexei Efros



Monet



Cezanne



Van Gogh



Ukiyo-e

Input



Monet



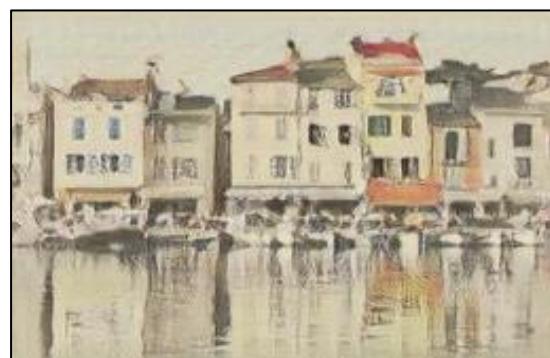
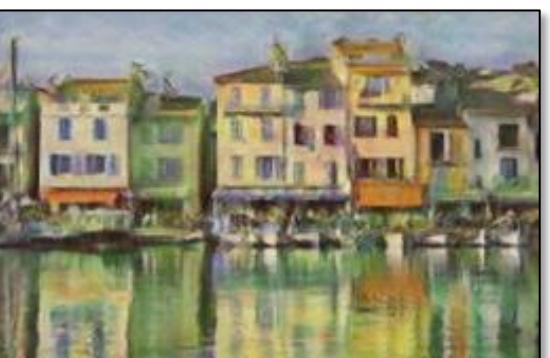
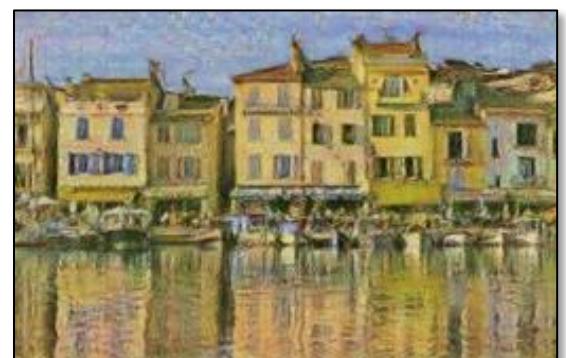
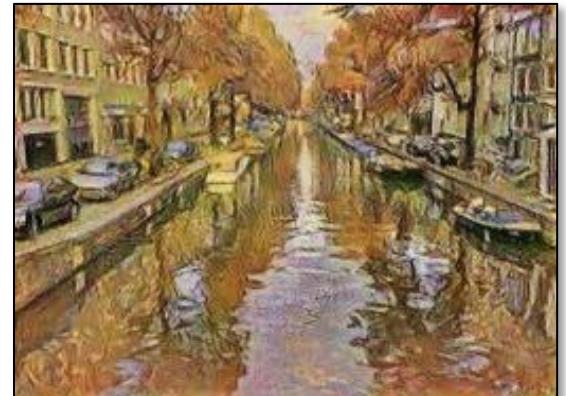
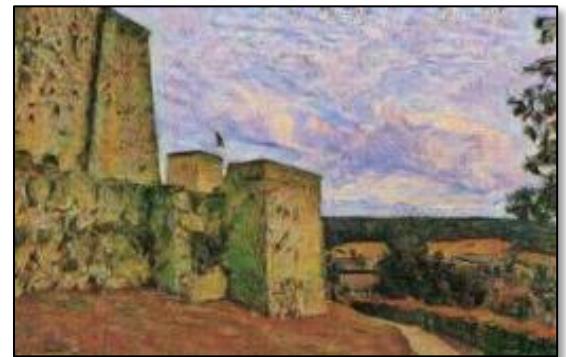
Van Gogh



Cezanne



Ukiyo-e

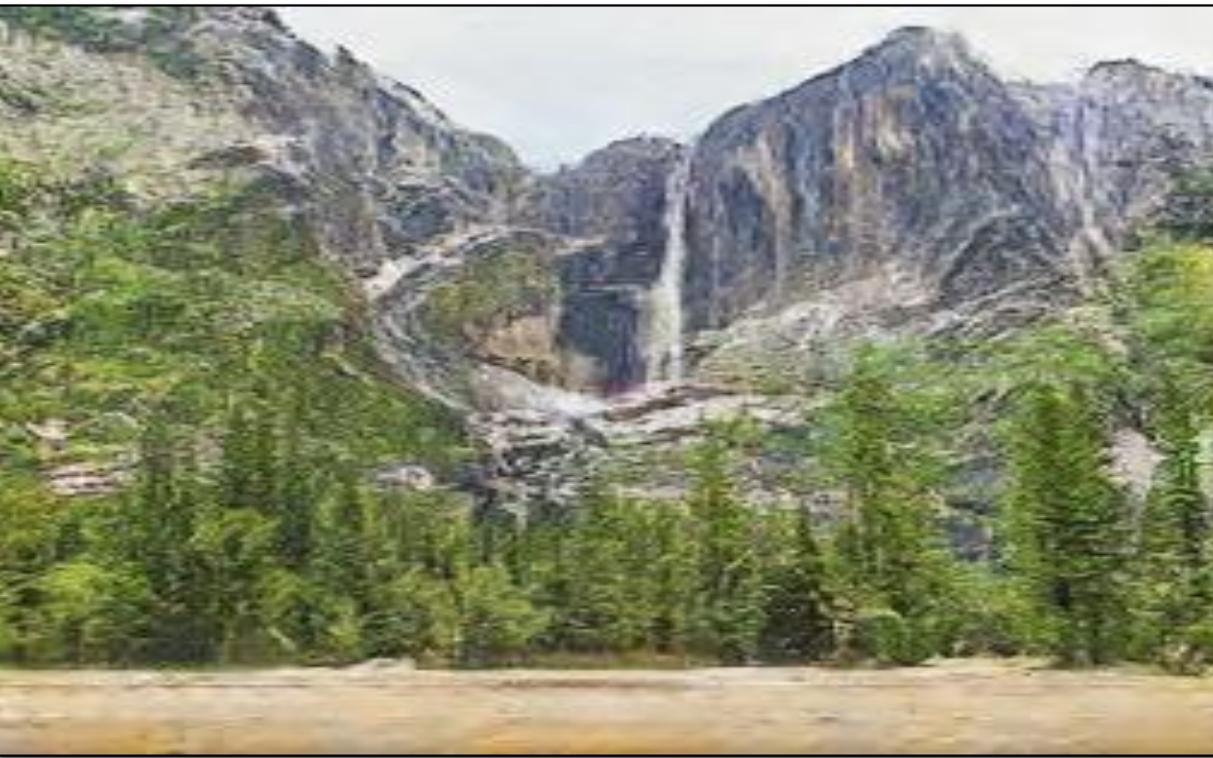


# Monet's paintings → photos



# Monet's paintings → photos





| Loss               | Map → Photo                   | Photo → Map                   |
|--------------------|-------------------------------|-------------------------------|
|                    | % Turkers labeled <i>real</i> | % Turkers labeled <i>real</i> |
| CoGAN [30]         | 0.6% ± 0.5%                   | 0.9% ± 0.5%                   |
| BiGAN/ALI [8, 6]   | 2.1% ± 1.0%                   | 1.9% ± 0.9%                   |
| SimGAN [45]        | 0.7% ± 0.5%                   | 2.6% ± 1.1%                   |
| Feature loss + GAN | 1.2% ± 0.6%                   | 0.3% ± 0.2%                   |
| CycleGAN (ours)    | <b>26.8% ± 2.8%</b>           | <b>23.2% ± 3.4%</b>           |

AMT ‘real vs fake’ test on maps ↔ aerial

| Loss               | Per-pixel acc. | Per-class acc. | Class IOU   |
|--------------------|----------------|----------------|-------------|
| CoGAN [30]         | 0.40           | 0.10           | 0.06        |
| BiGAN/ALI [8, 6]   | 0.19           | 0.06           | 0.02        |
| SimGAN [45]        | 0.20           | 0.10           | 0.04        |
| Feature loss + GAN | 0.06           | 0.04           | 0.01        |
| CycleGAN (ours)    | <b>0.52</b>    | <b>0.17</b>    | <b>0.11</b> |

FCN scores on cityscapes labels→ photos

| Loss               | Per-pixel acc. | Per-class acc. | Class IOU   |
|--------------------|----------------|----------------|-------------|
| CoGAN [30]         | 0.45           | 0.11           | 0.08        |
| BiGAN/ALI [8, 6]   | 0.41           | 0.13           | 0.07        |
| SimGAN [45]        | 0.47           | 0.11           | 0.07        |
| Feature loss + GAN | 0.50           | 0.10           | 0.06        |
| CycleGAN (ours)    | <b>0.58</b>    | <b>0.22</b>    | <b>0.16</b> |

Classification performance of photo→labels

# CycleGAN implementations

PyTorch

## [pytorch-CycleGAN-and-pix2pix](#)

Image-to-image translation in PyTorch (e.g., horse2zebra, edges2cats, and more)

● Python    ★ 4.3k    ⚙ 970

Torch

## [CycleGAN](#)

Software that can generate photos from paintings, turn horses into zebras, perform style transfer, and more.

● Lua    ★ 6.5k    ⚙ 940

20+ implementations by researchers/developers:

- Tensorflow, Chainer, mxnet, Lasagne, Keras...

# DALL-E model

<https://openai.com/blog/dall-e/>

- 12B parameter Transformer - version of GPT-3 trained to generate images from text descriptions.
- 1. Pre-train VAE model to represent 8x8 natural image patches as discrete code ( $K=8192$ ).
  - So 256x256 image  $\rightarrow$  32x32 grid of discrete codes.
- 2. Concatenate 1024 image codes with text embedding of description. Pass to auto-regressive Transformer. Model joint distribution of text, images.
- Train on Conceptual Captions (3.3M text/image pairs) + JFT-300M.
- Sample by conditioning on user-specified caption.
- Reranking procedure using pre-trained model (CLIP).

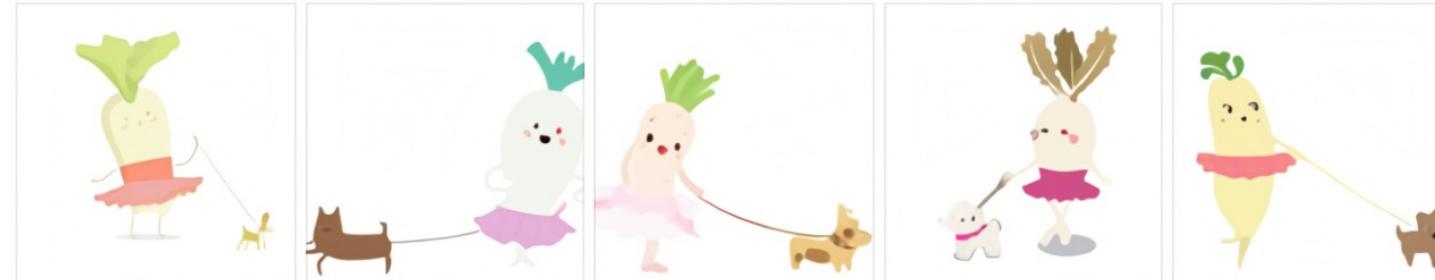
# DALL-E model

<https://openai.com/blog/dall-e/>

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED  
IMAGES



Edit prompt or view more images↓

TEXT PROMPT

an armchair in the shape of an avocado....

AI-GENERATED  
IMAGES



Edit prompt or view more images↓

# DALL-E model [OpenAI]

<https://openai.com/blog/dall-e/>

TEXT PROMPT

a store front that has the word 'openai' written on it. . . .

AI-GENERATED  
IMAGES



Edit prompt or view more images↓

TEXT & IMAGE  
PROMPT

the exact same cat on the top as a sketch on the bottom

AI-GENERATED  
IMAGES



Edit prompt or view more images↓

# DALL-E model

| TEXT PROMPT         | a pentagonal green clock. a green clock in the shape of a pentagon.                 |  |   |   |   |
|---------------------|---|--|---|---|---|
| AI-GENERATED IMAGES |    |    |    |    |    |
|                     |    |    |    |    |    |
|                     |   |   |   |   |   |
|                     |  |  |  |  |  |
|                     |  |  |  |  |  |

We find that DALL-E can render familiar objects in polygonal shapes that are sometimes unlikely to occur in the real world. For some objects, such as “picture frame” and “plate,” DALL-E can reliably draw the object in any of the polygonal shapes except heptagon. For other objects, such as “manhole cover” and “stop sign,” DALL-E’s success rate for more unusual shapes, such as “pentagon,” is considerably lower.

For several of the visuals in this post, we find that repeating the caption, sometimes with alternative phrasings, improves the consistency of the results.

# **Disentangling Content and Pose with an Adversarial loss**

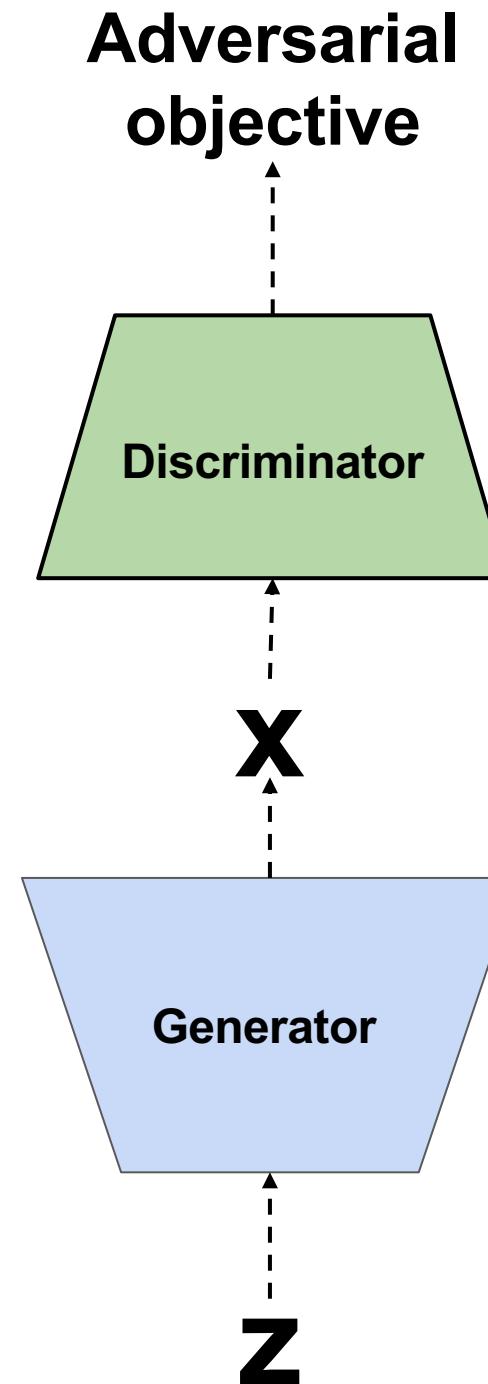
Emily Denton

CVPR GAN Tutorial  
June 2018

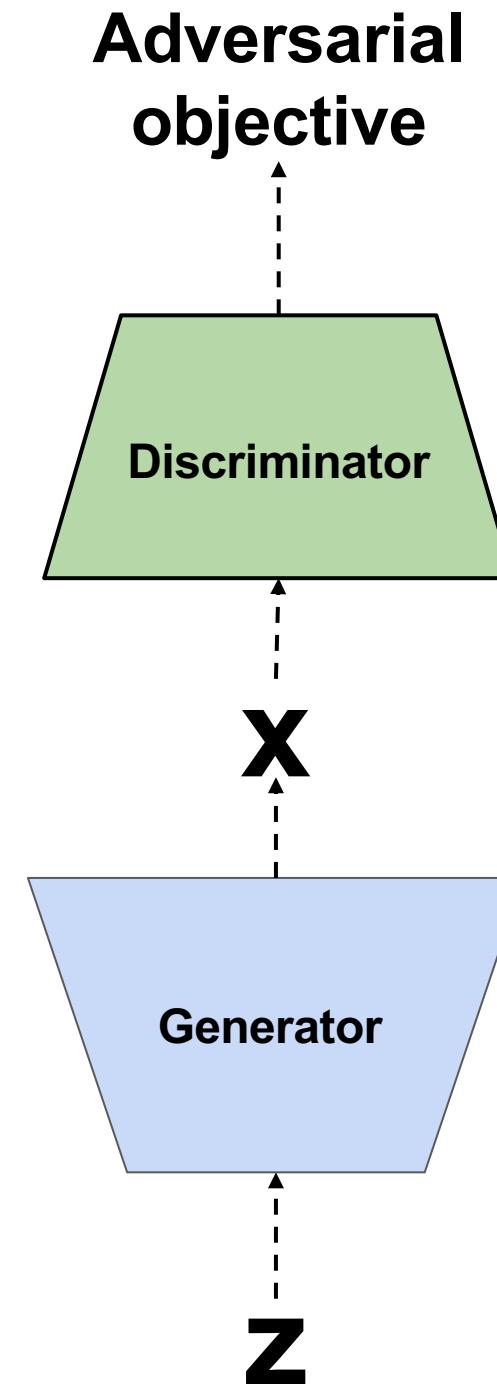


**NYU**

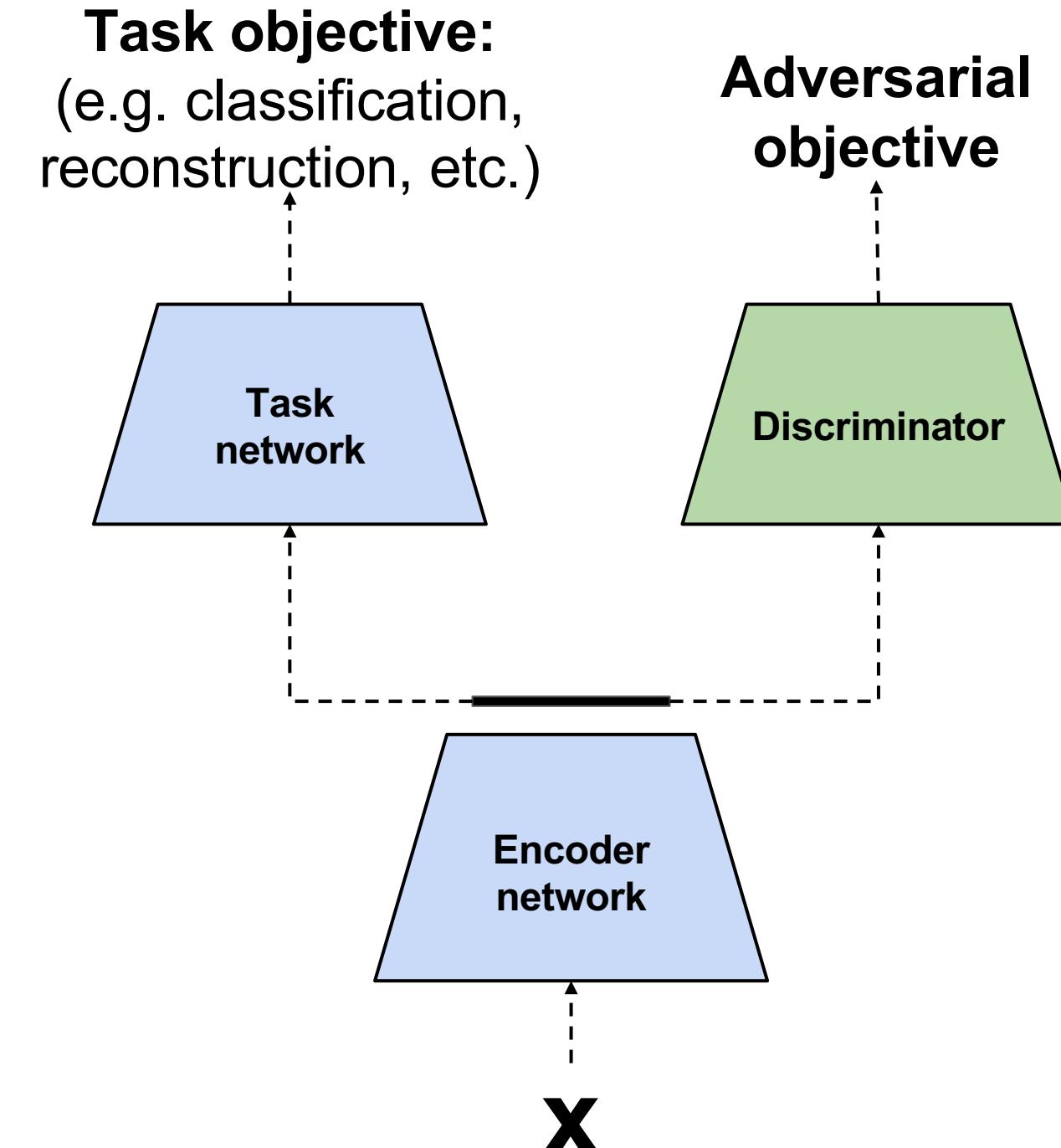
## Generative adversarial network framework:



## Generative adversarial network framework:

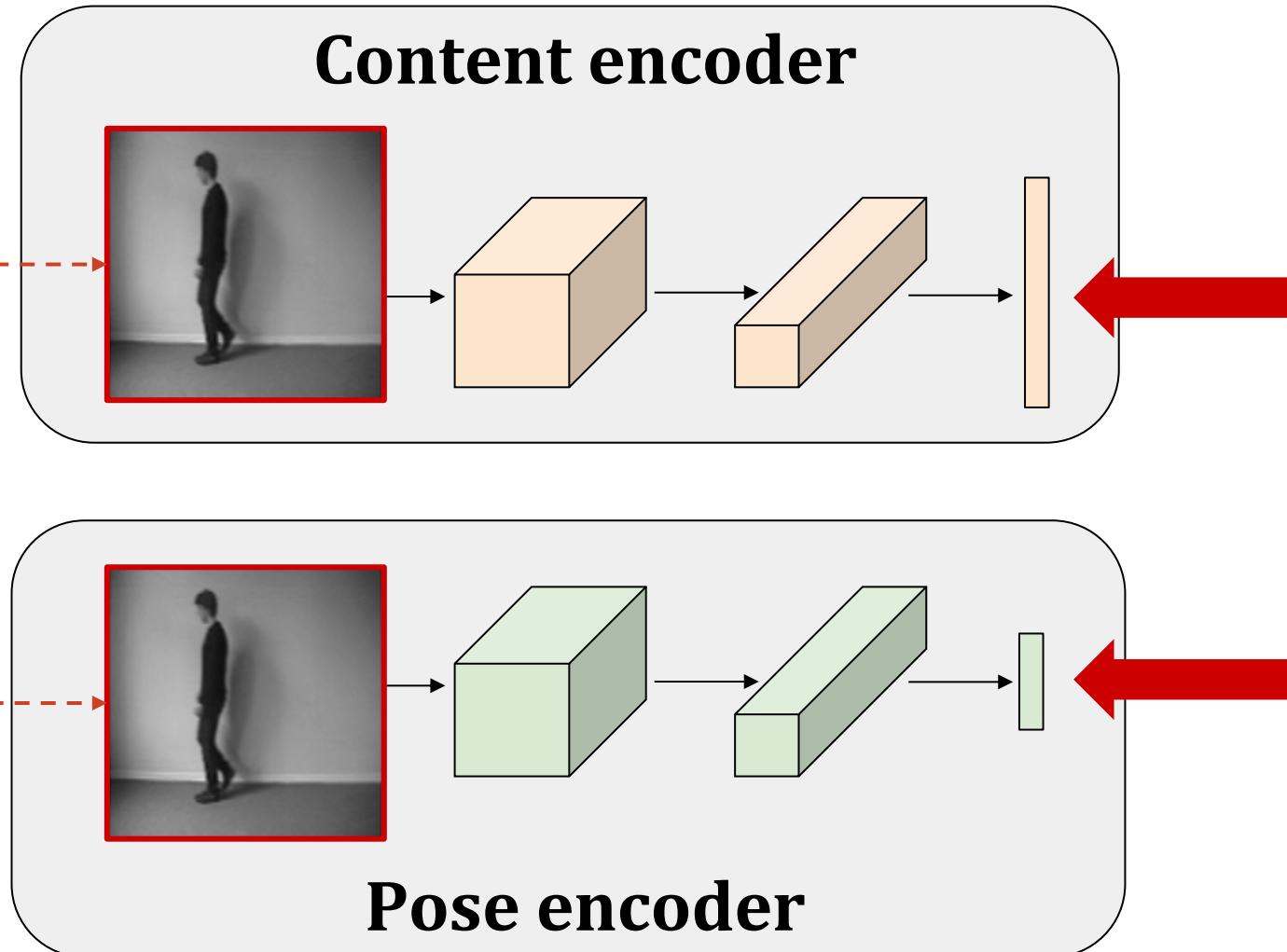


## Adversarial losses to shape representations:



# DrNet: two separate encoders

Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017

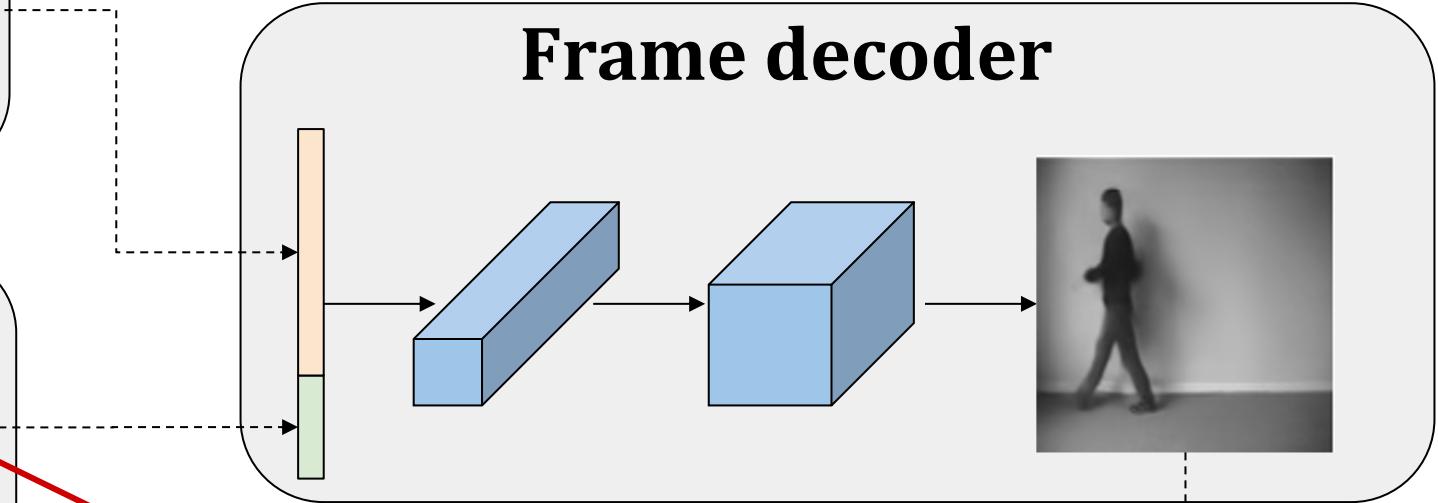
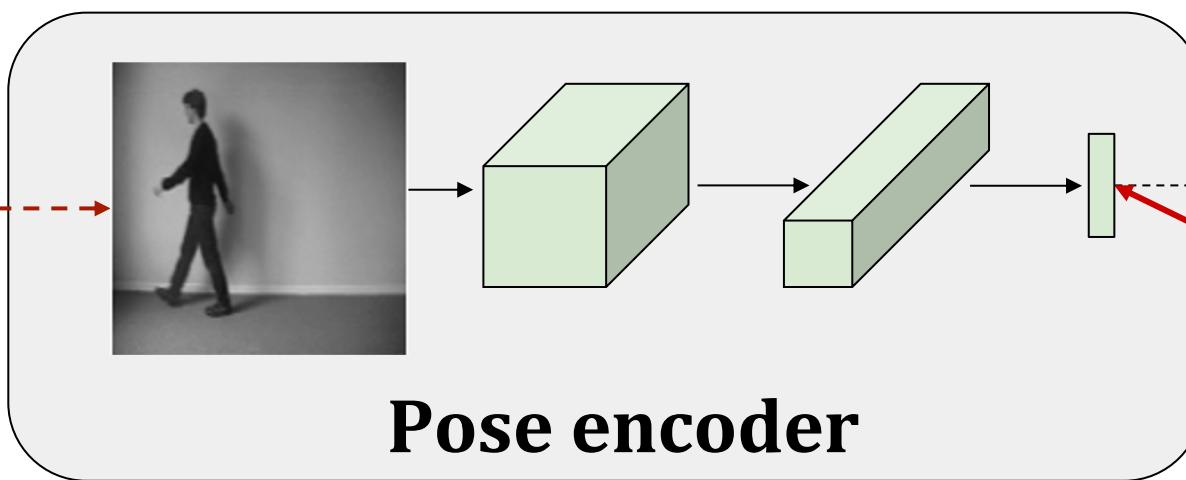
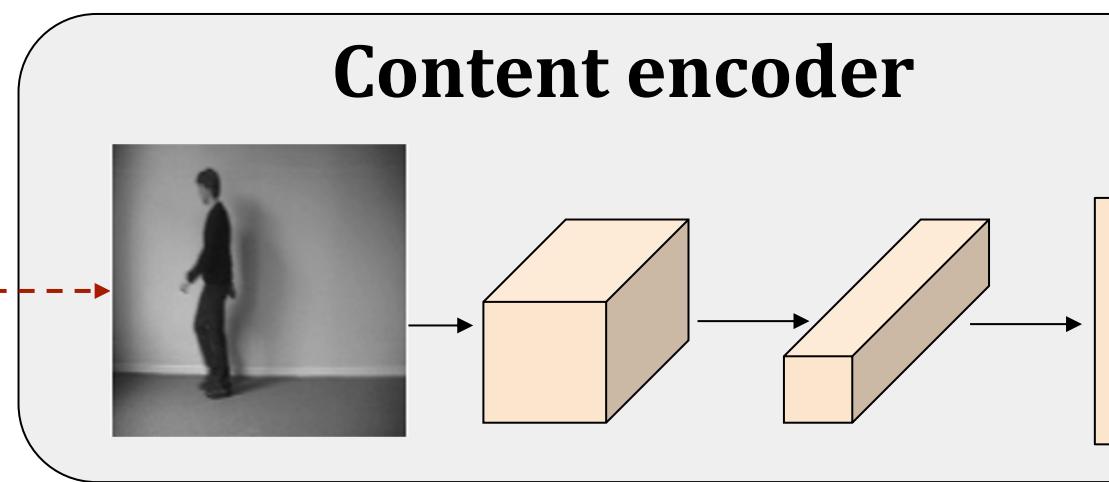


**Time invariant information:**  
Lighting/Background  
Identity/clothing

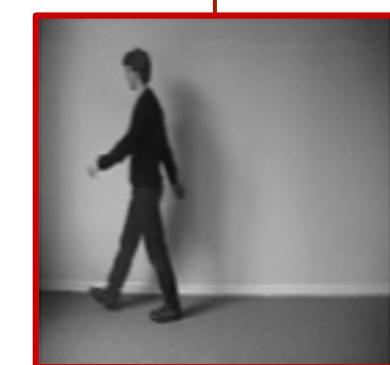
**Time varying information:**  
Pose of body



**Content vector should contain anything predictable from past frame**



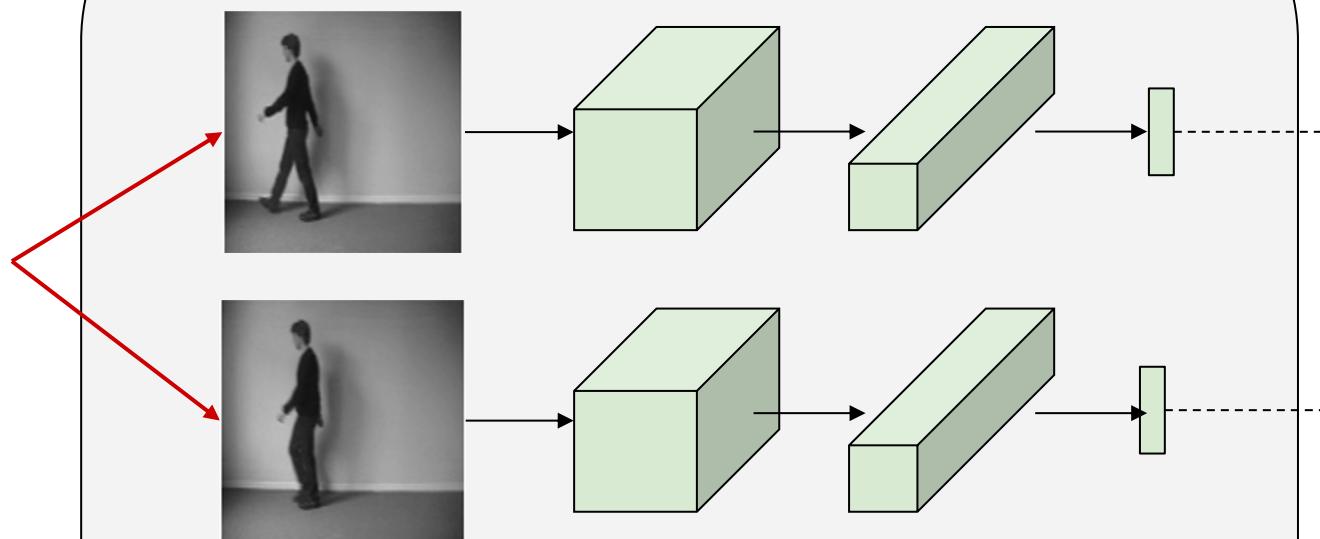
***Don't want pose vector encoding anything constant across time***



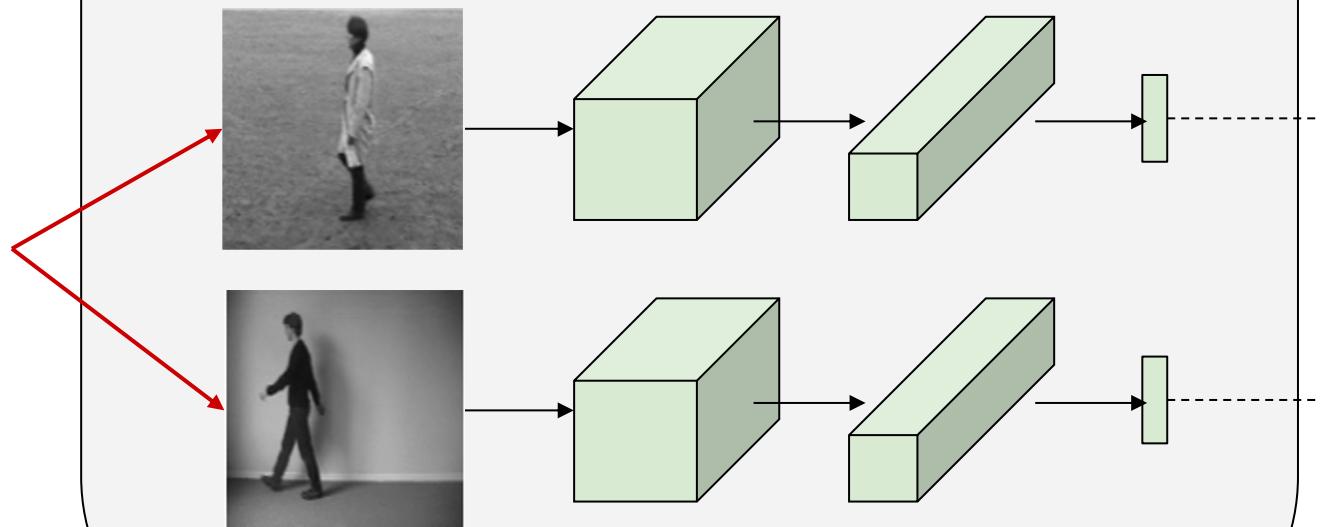
**$L_{\text{reconstruction}}$**

## Pose encoder:

Same video

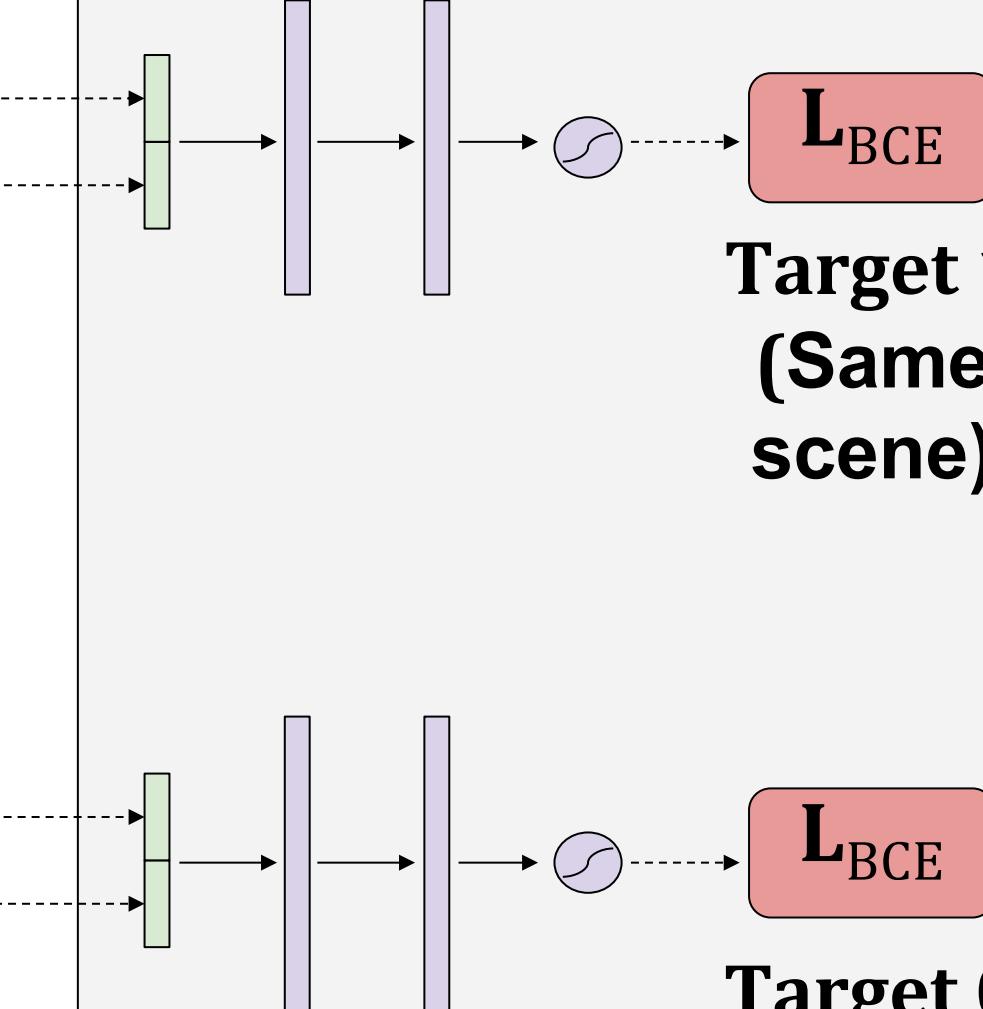


Different video



## Scene discriminator:

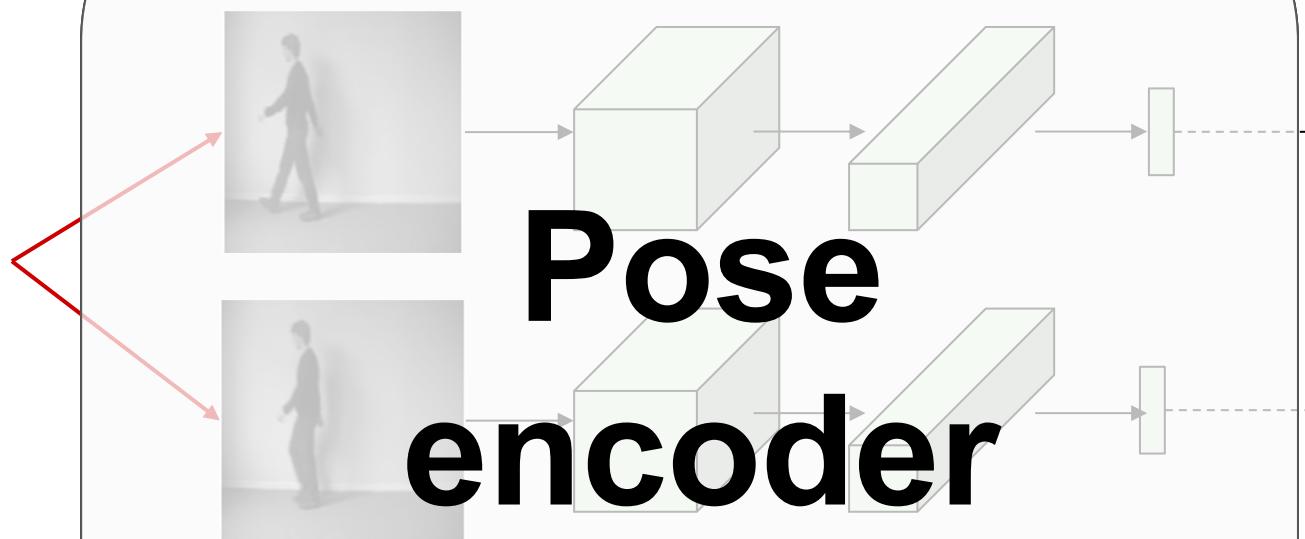
$L_{BCE}$   
Target 1  
(Same scene)



$L_{BCE}$   
Target 0  
(Different scene)

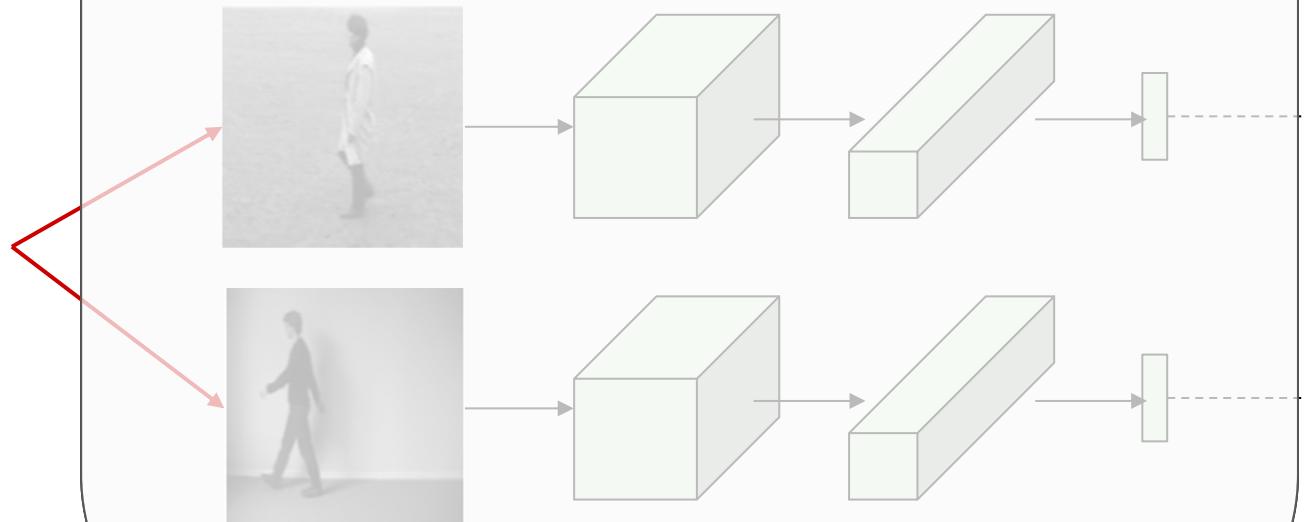
Pose encoder:

Same  
video



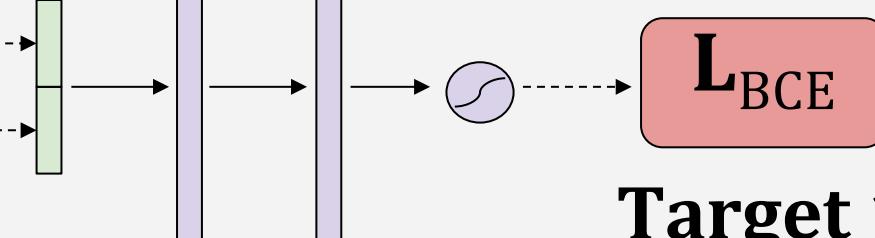
**Pose  
encoder  
held fixed**

Different  
video

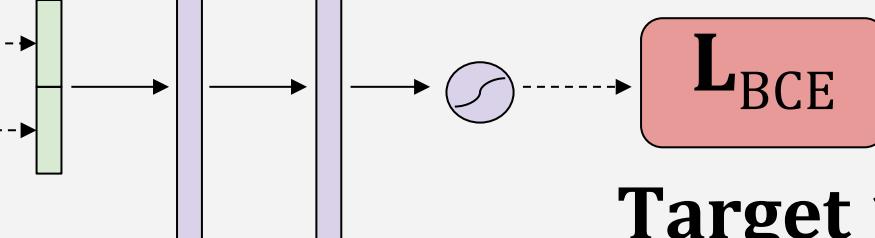


Scene discriminator:

Target 1  
(Same  
scene)

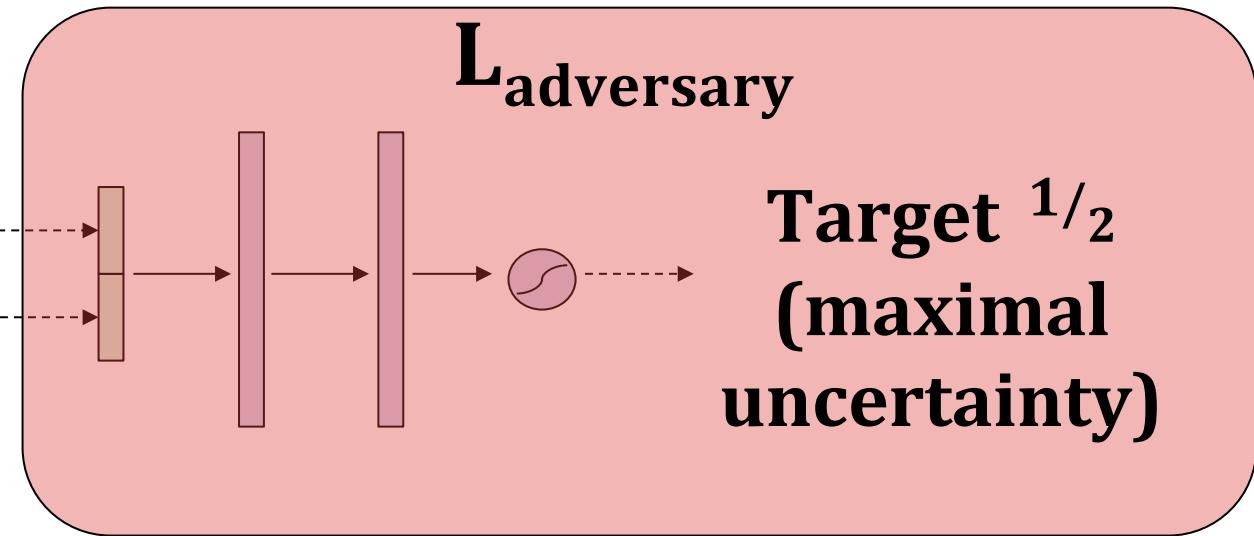
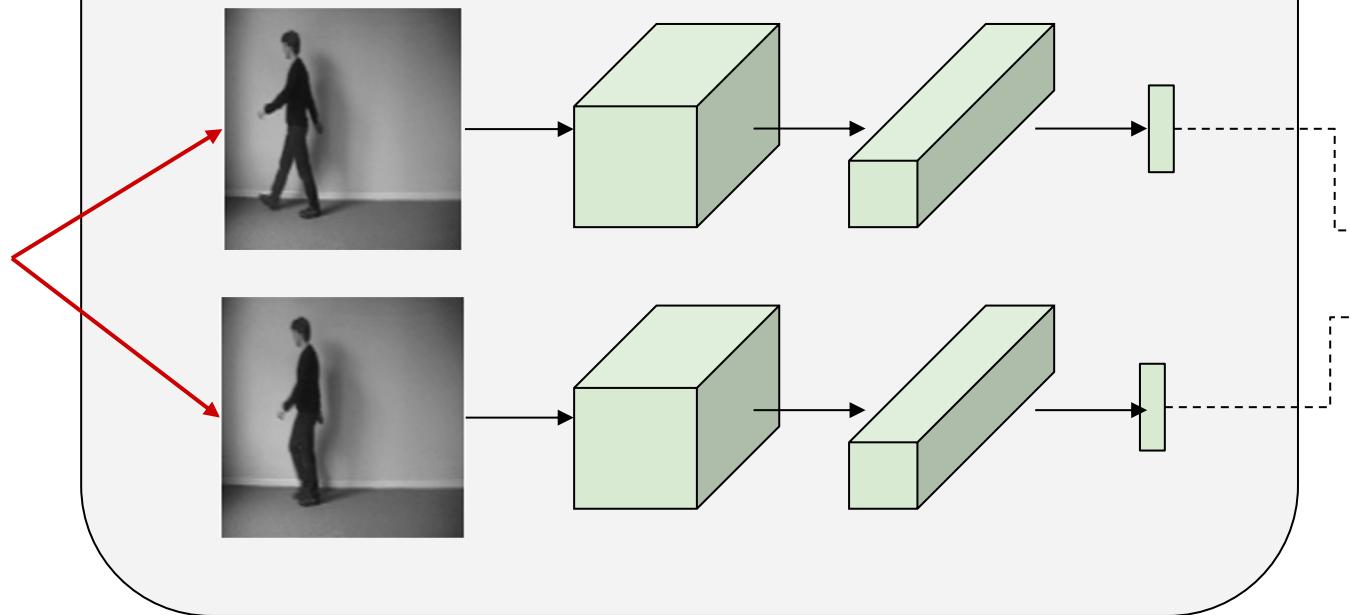


Target 0  
(Different  
scene)



## Pose encoder:

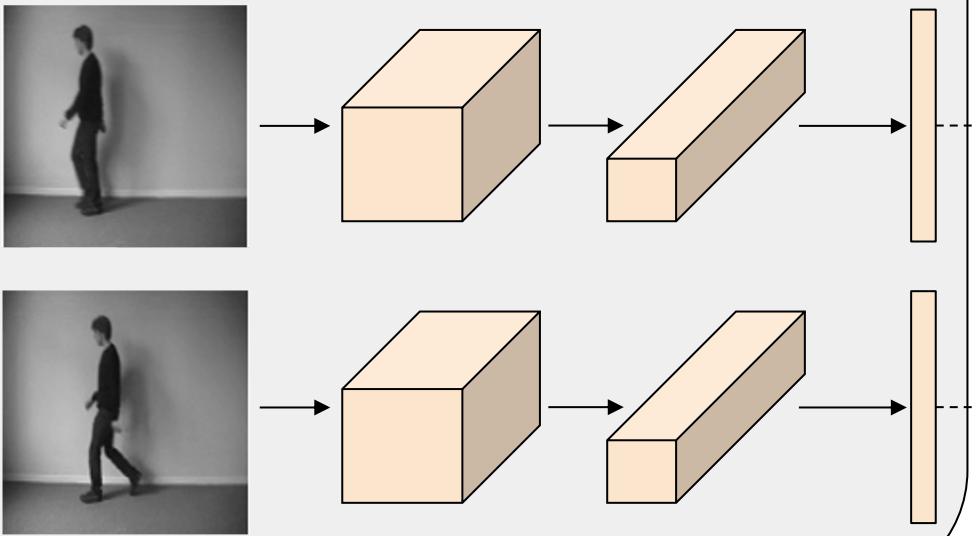
Same video



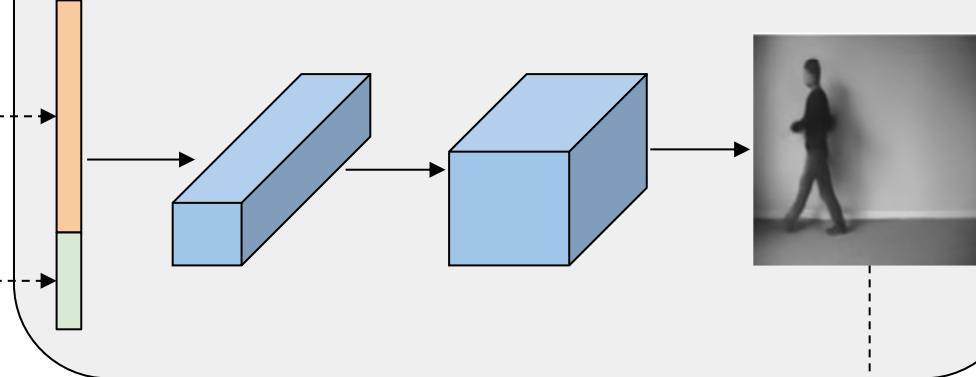
Train pose encoder to produce pose vectors that make the discriminator **maximally uncertain** about the content of the video

Scene discriminator held fixed, only used to compute gradients for pose encoder

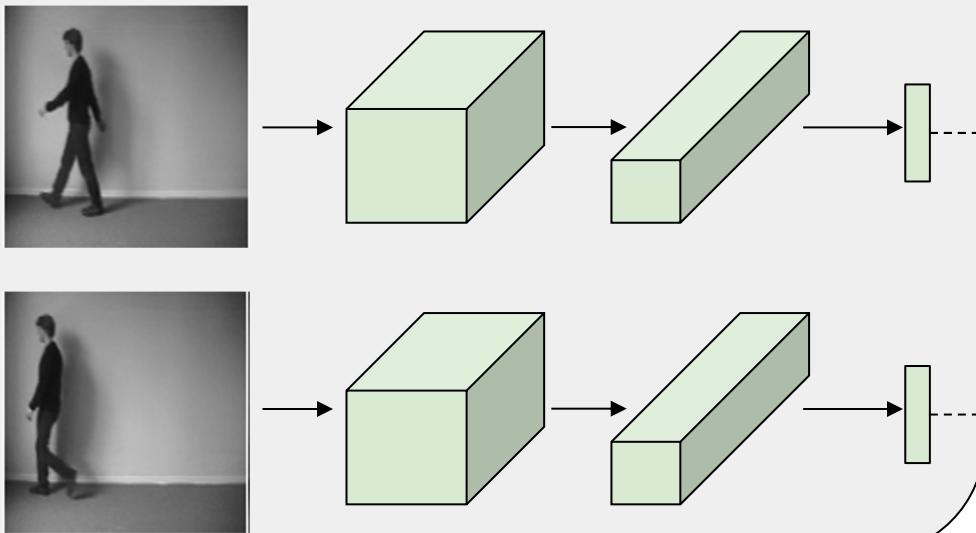
## Content encoder



## Frame decoder:



## Pose encoder



## $L_{adversarial}$

Target =  $\frac{1}{2}$   
(maximal uncertainty)

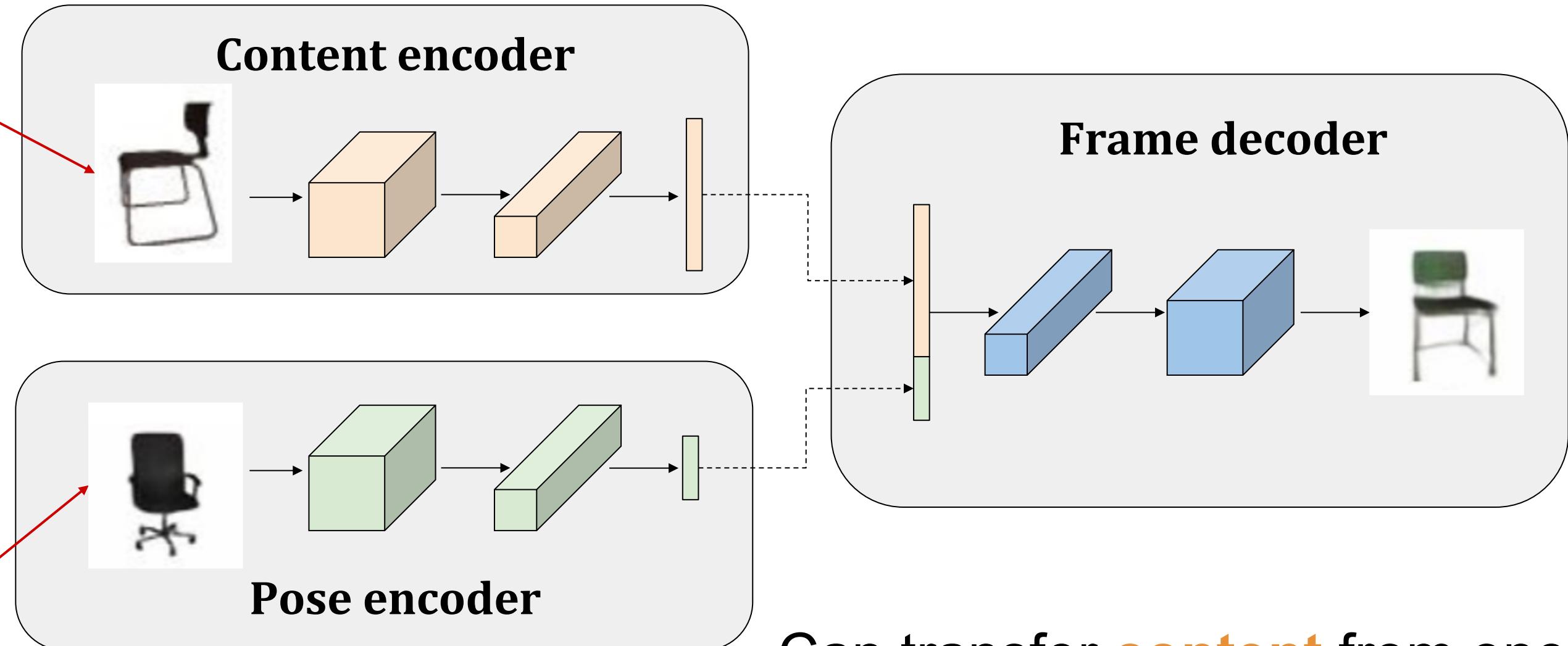
$L_{reconstruction}$

$$\mathcal{L} = \mathcal{L}_{reconstruction}(E_c, E_p, D) + \alpha \mathcal{L}_{similarity}(E_c) + \beta (\mathcal{L}_{adversarial}(E_p) + \mathcal{L}_{adversarial}(C))$$

# Image synthesis by analogy

Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017

Content  
image



Pose  
image

Can transfer **content** from one image and **pose** from another to synthesize a **new image**

# Image synthesis by analogy

Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017

Pose



Content

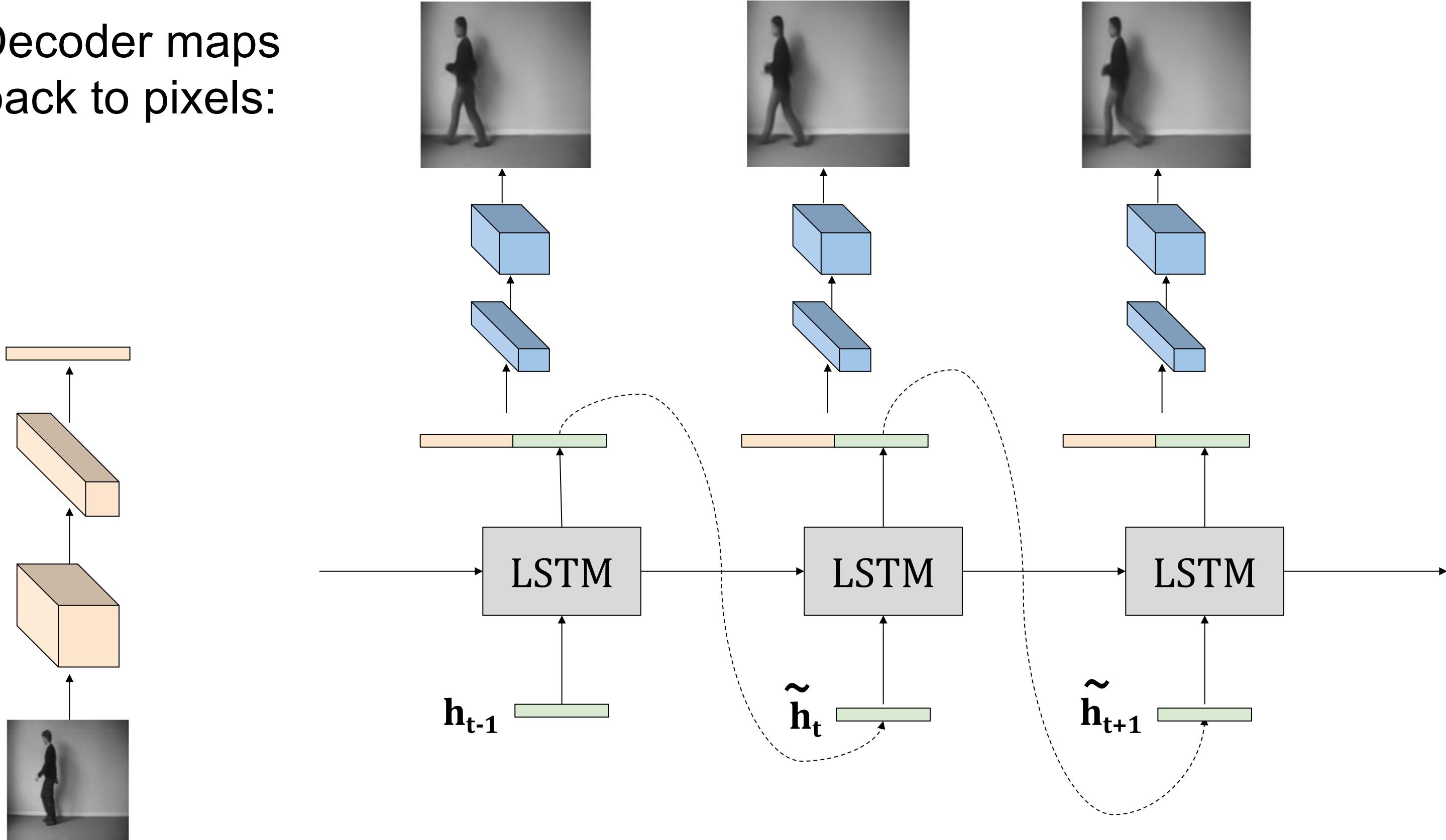


# Interpolation in pose space

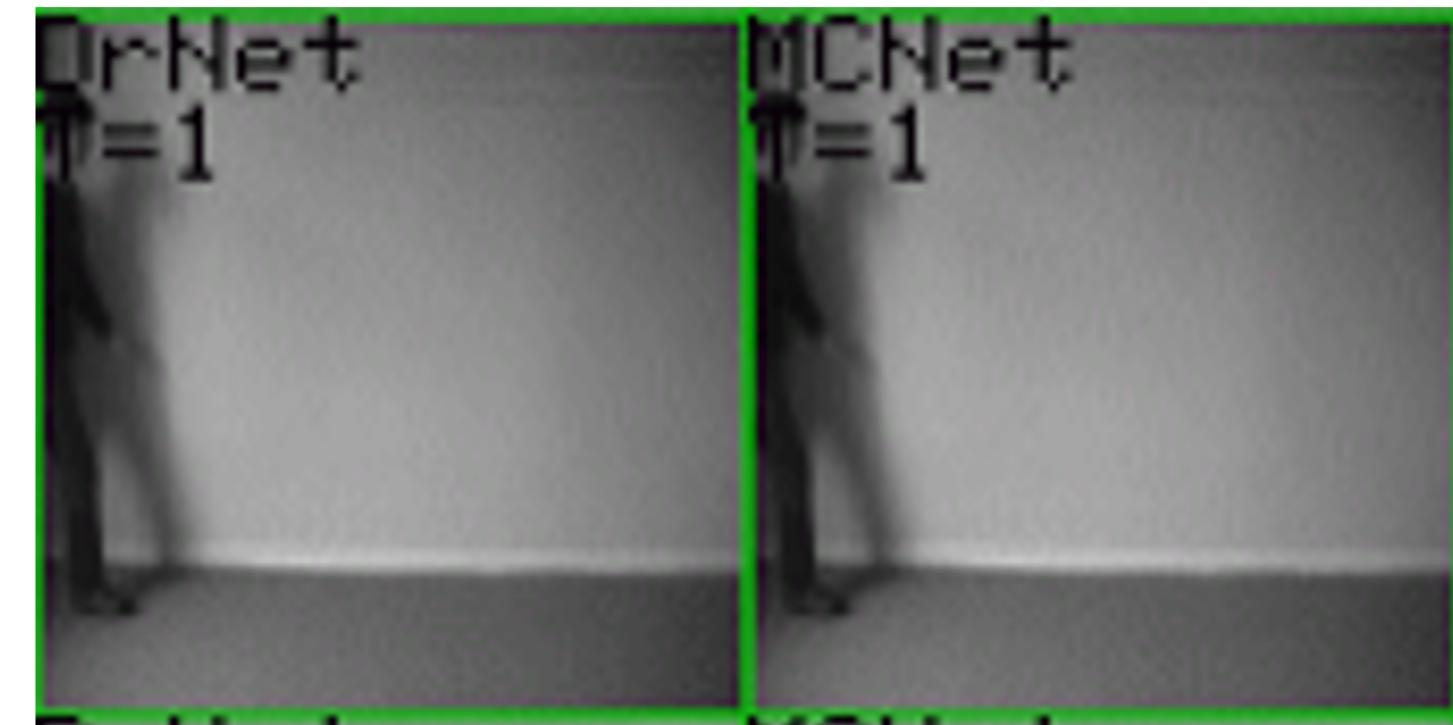
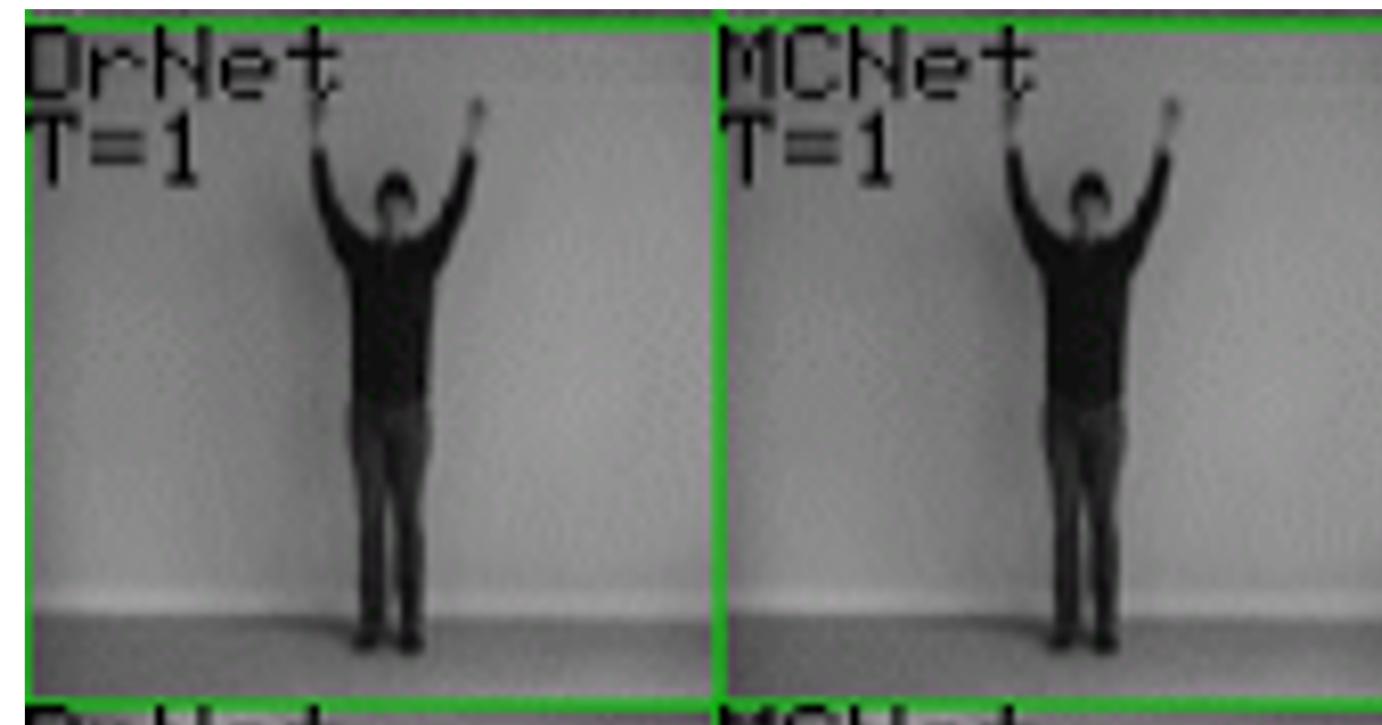
Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017



Decoder maps  
back to pixels:



# KTH long term video generation

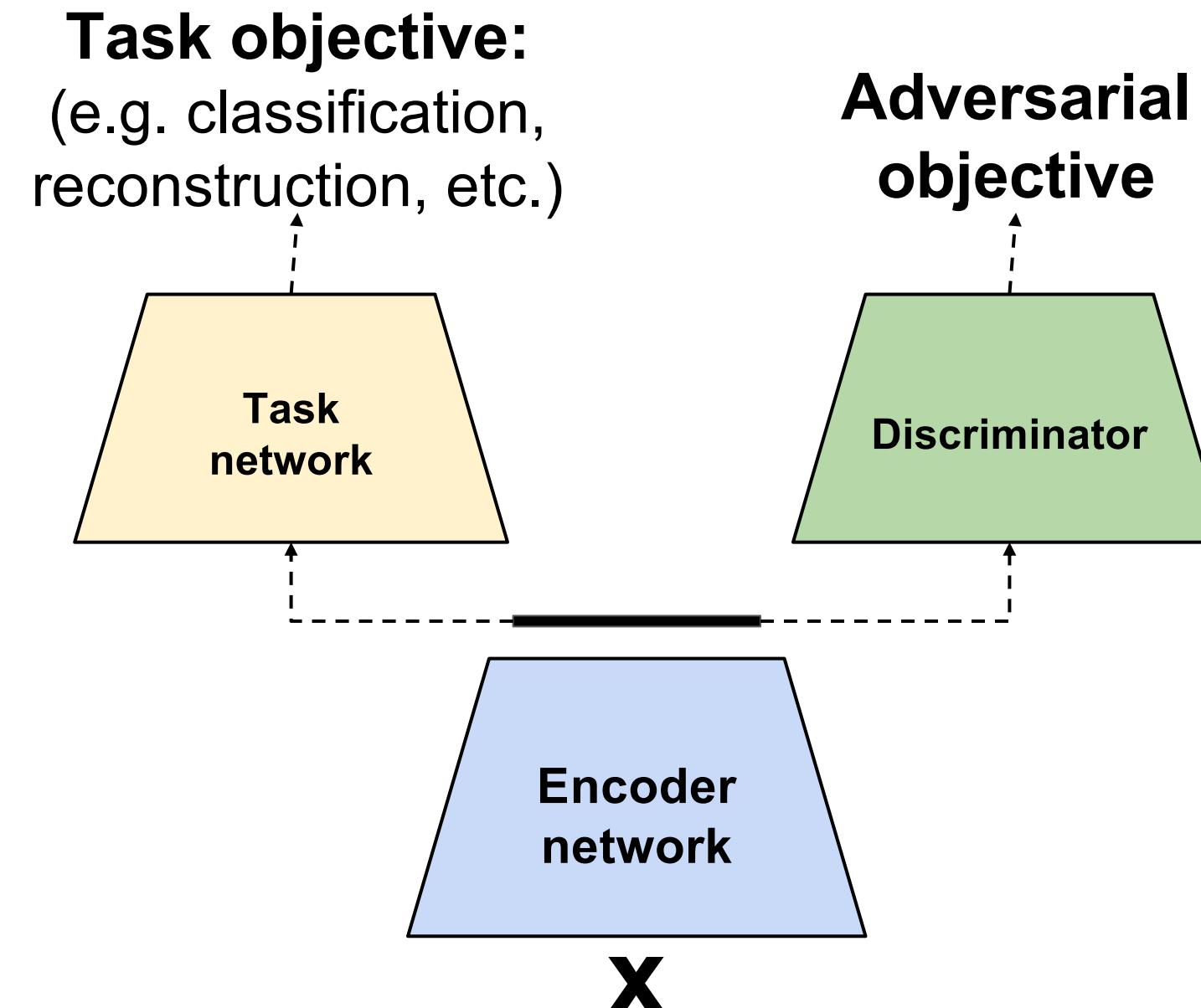


Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017

# Part I: Disentangling content and pose with an adversarial loss

Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017

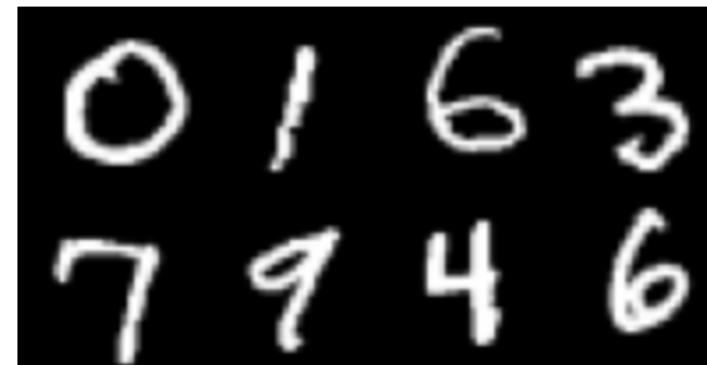
## Part II: Survey of adversarial losses in feature space



# Domain adaptation

Labelled examples from **source domain**,  
few or no labels from **target domain**

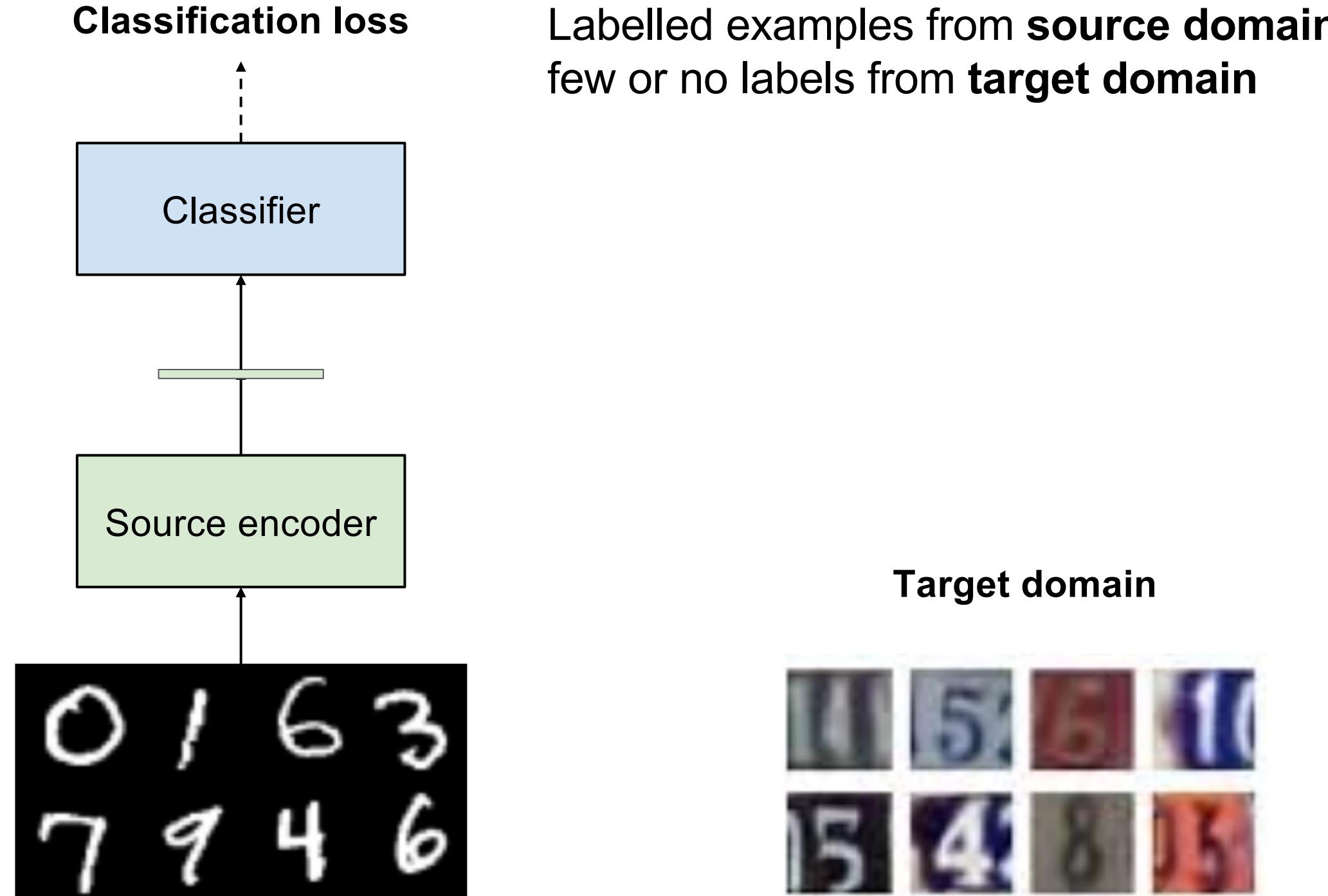
Source domain



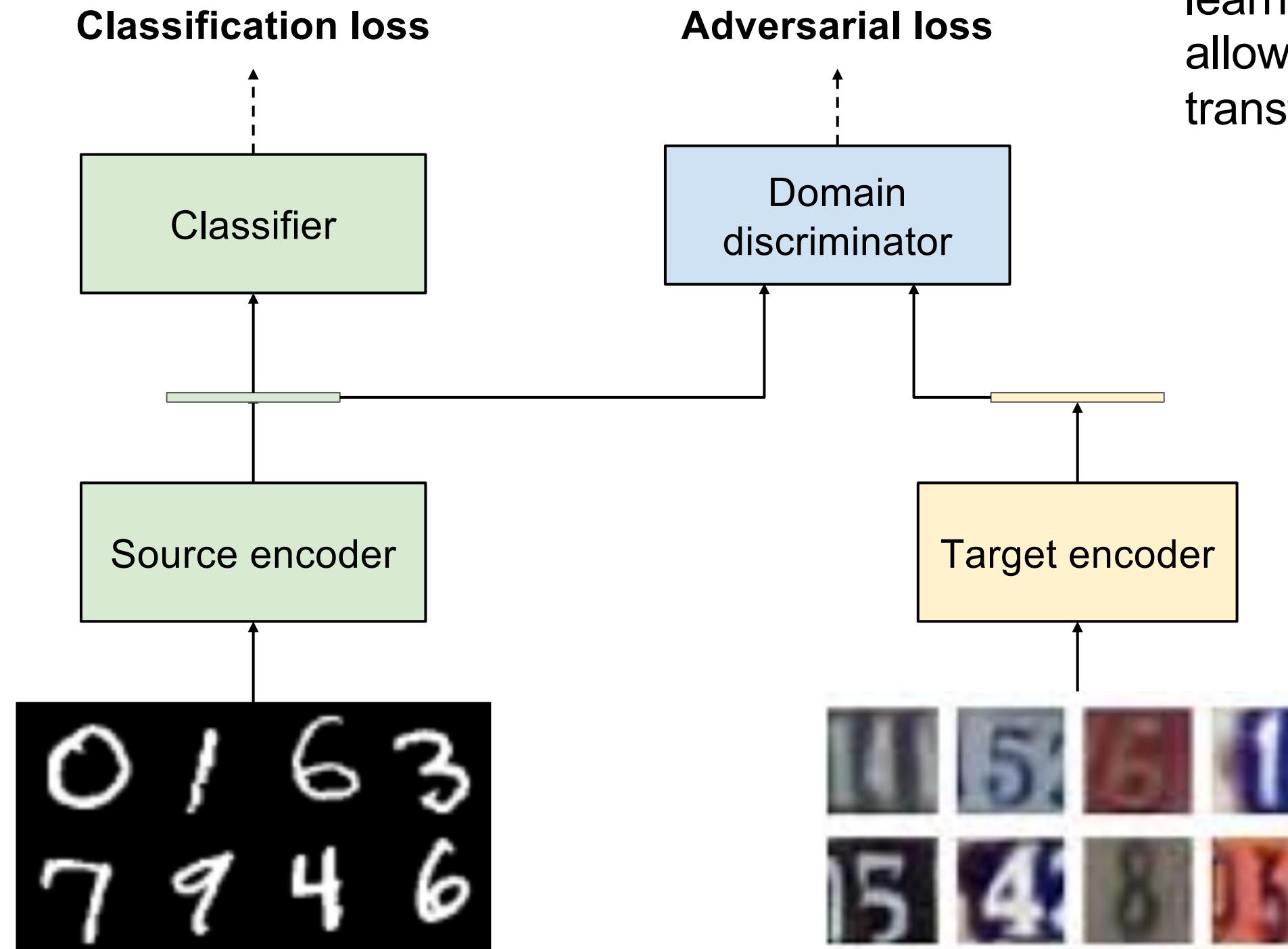
Target domain



# Domain adaptation

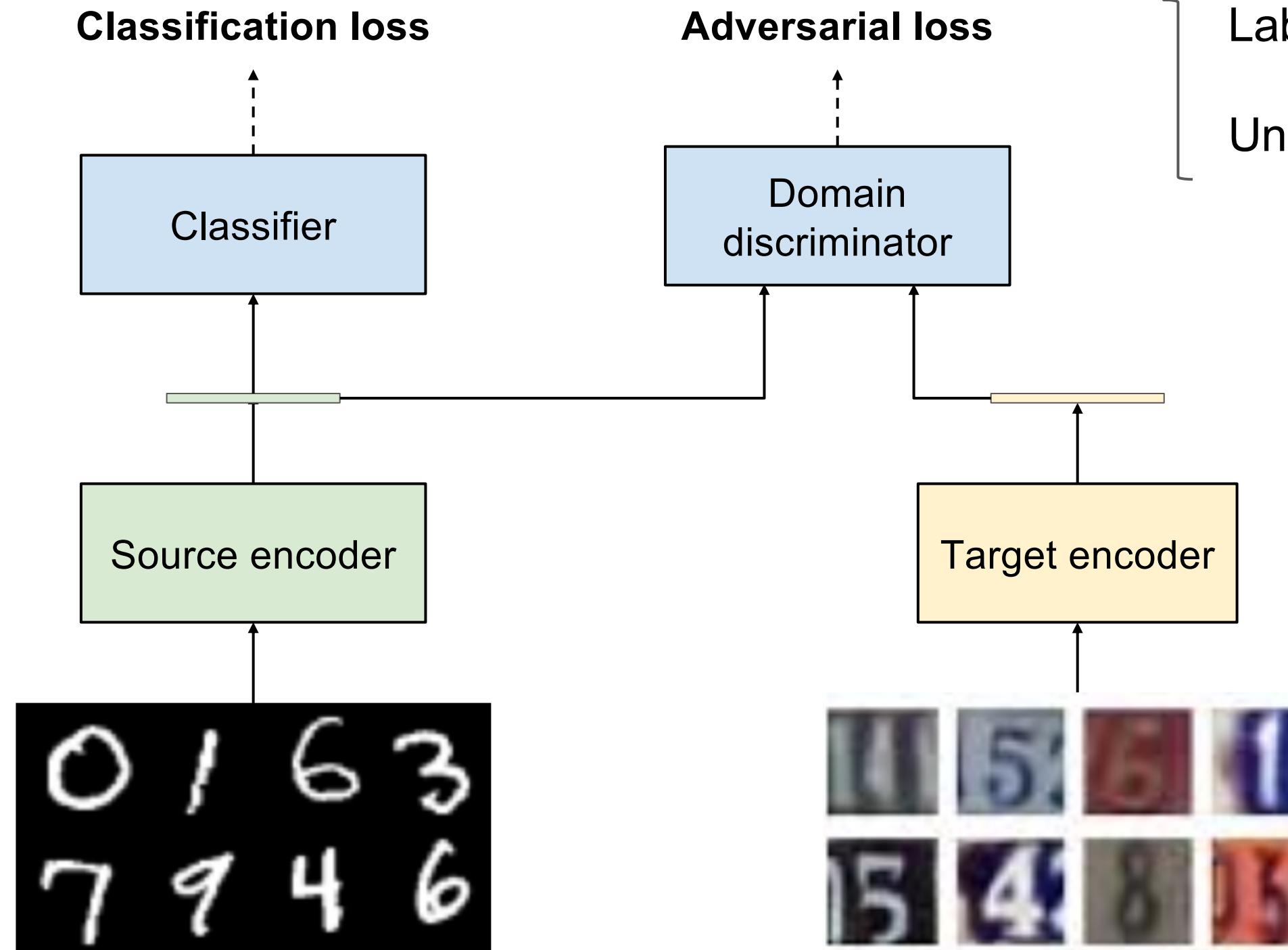


# Domain adaptation



**Adversarial loss** can be used to learn **domain invariant features**, allowing source classifier to transfer to target domain

# Domain adaptation



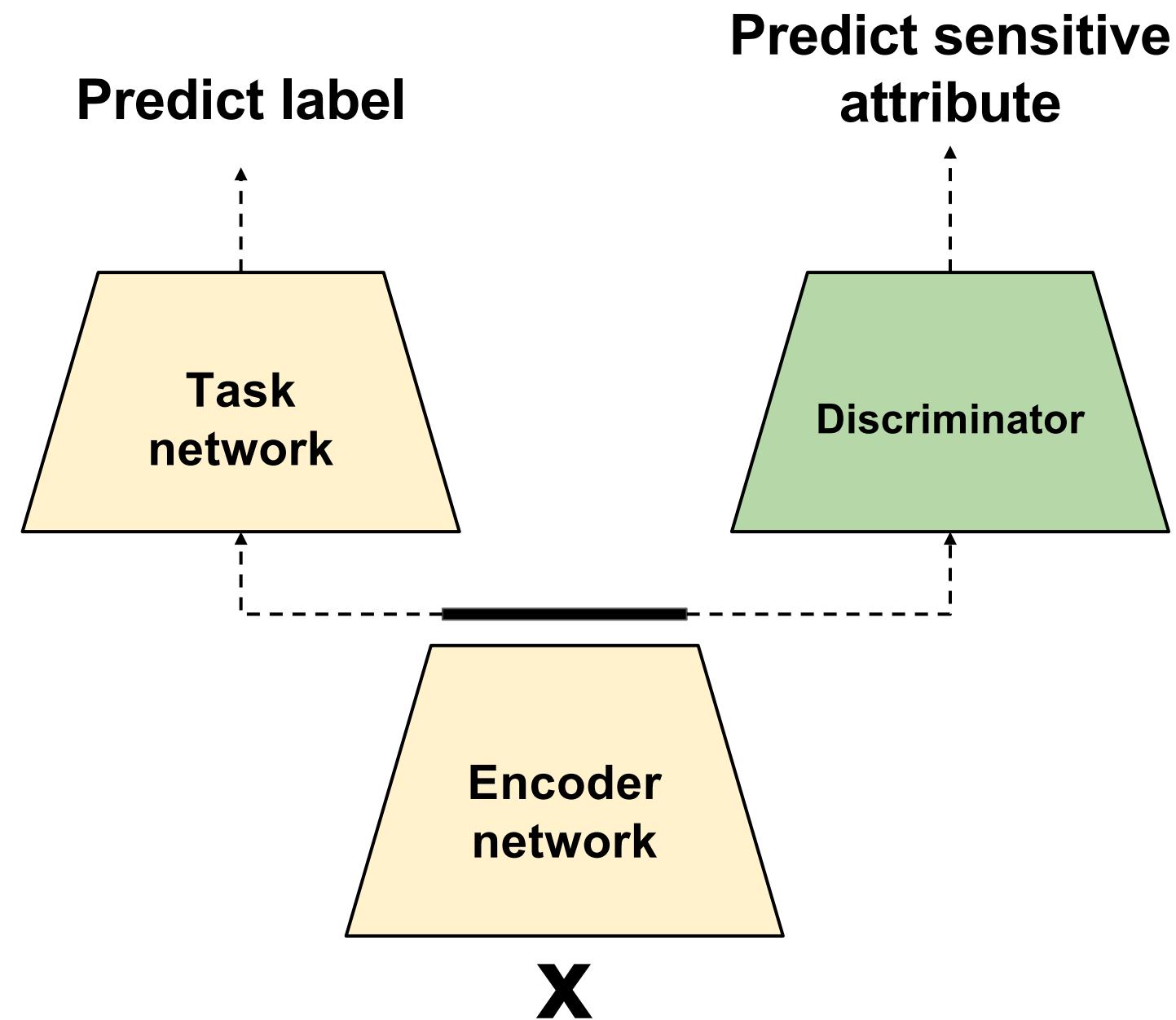
Gradient reversal [*Ganin and Lempitsky, 2015*]

Label flip [*Tzeng et al. 2017*]

Uniform target [*Tzeng et al. 2015*]

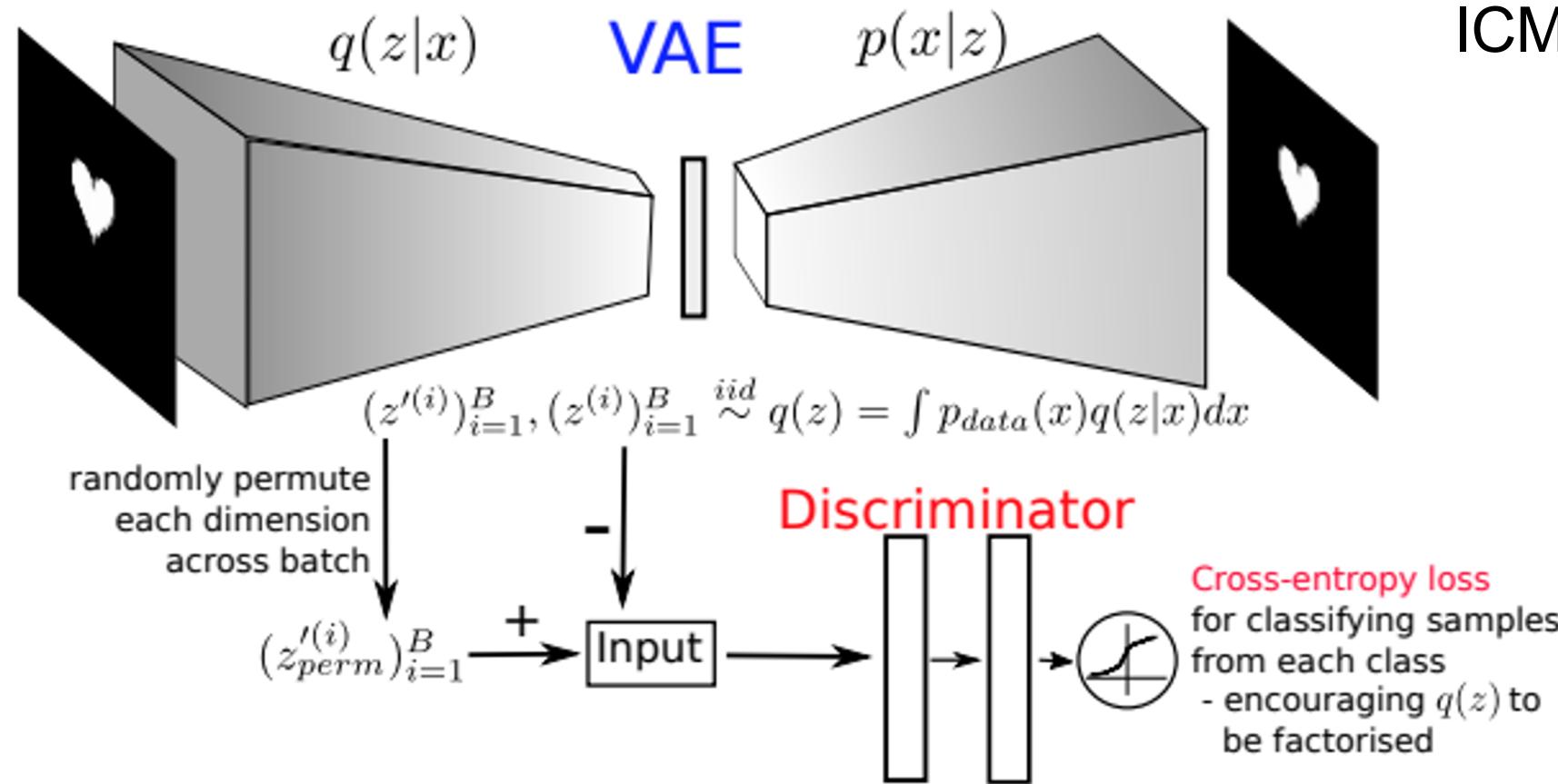
# Learning fair representations

- Closely related to problem of domain adaptation
  - source/transfer domain vs. demographic groups
- Different formulations of adversarial objectives achieve different notions of fairness
  - Edwards & Storkey, 2016
  - Beutel et al. 2017
  - Zhang et al. 2018
  - Madras et al. 2018



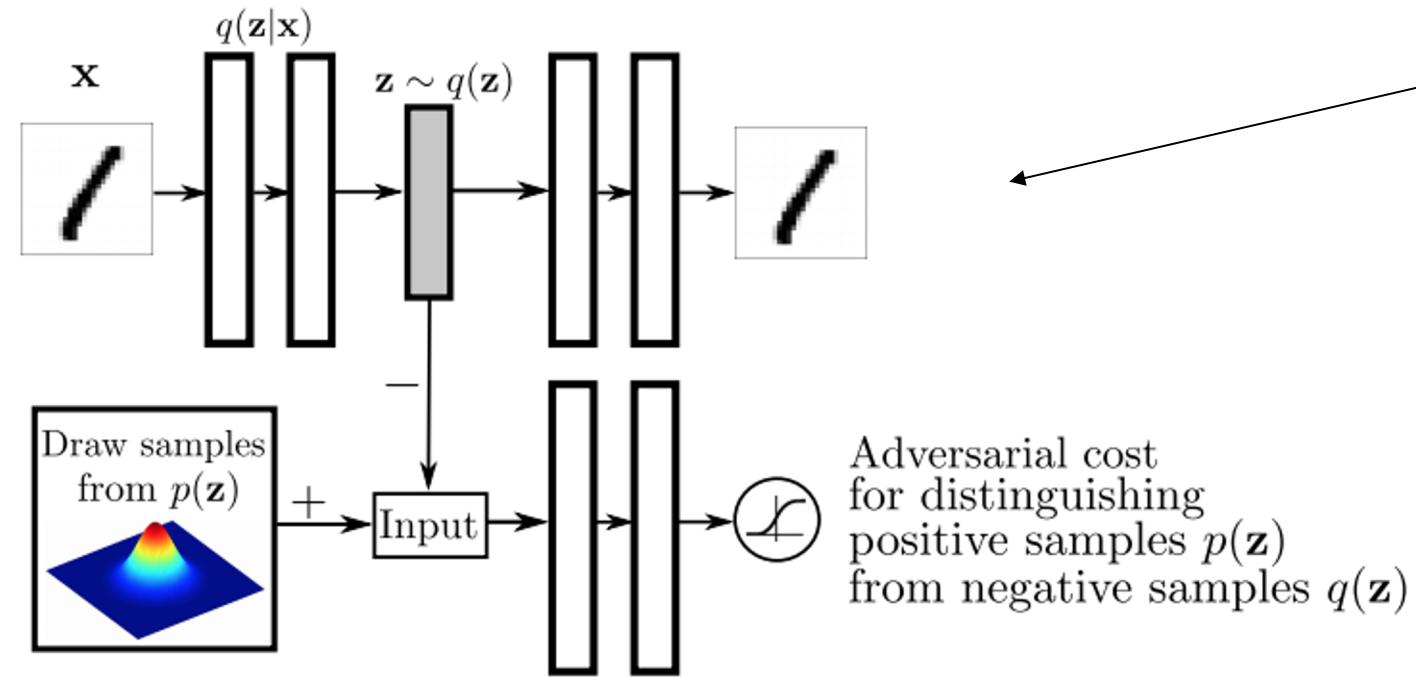
# Independent components

Kim and Mnih. Disentangling by Factorising.  
ICML, 2018



- Discriminate marginal distribution vs. product of marginals:  $q(z_1, \dots, z_n)$  vs.  $\prod q(z_i)$
- Earlier work on discrete code setting by Schmidhuber (1992)

# Prior distributions of generative models



## Adversarial autoencoders:

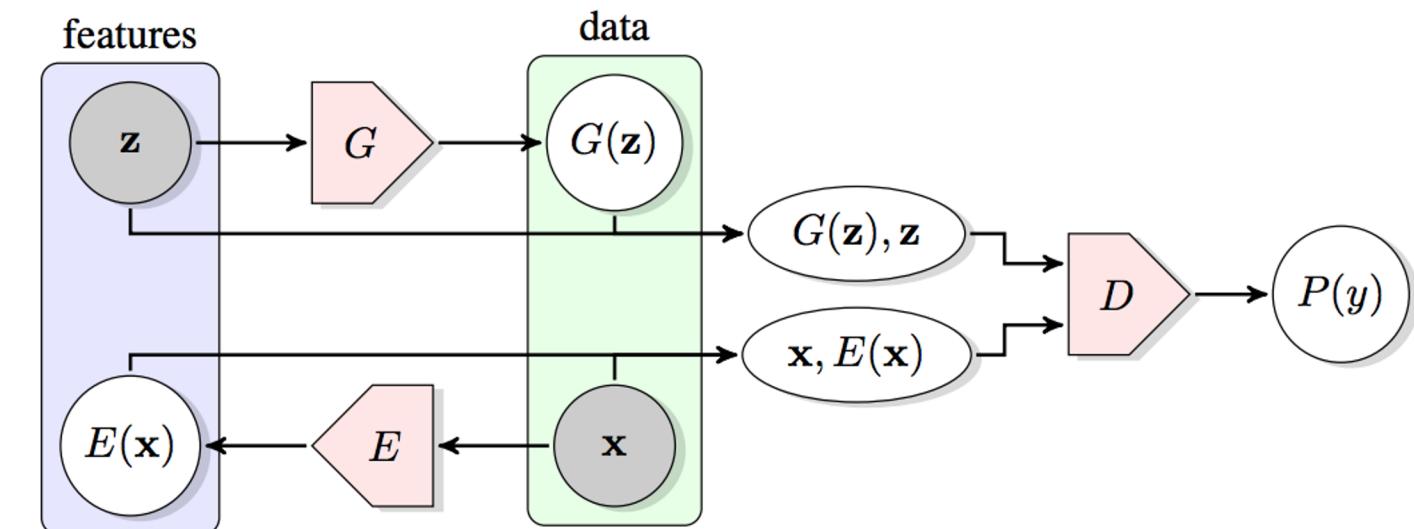
Match aggregate approx posterior  $q(z)$   
[Makhzani et al. 2016]

## Adversarial variational bayes:

Match approx posterior  $q(z|x)$   
[Mescheder et al. 2017]

## Adversarial feature learning:

GAN loss in image space and latent space  
[Dumoulin et al. 2017; Donahue et al. 2017]

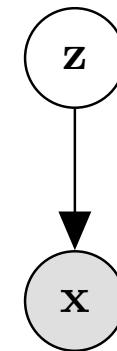


## References

- Beutel et al. *Data decisions and theoretical implications when adversarially learning fair representations*. arXiv:1707.00075, 2017.
- Denton and Birodkar. *Unsupervised Learning of Disentangled Representations from Video*. NIPS, 2017.
- Donahue et al. *Adversarial Feature Learning*. ICLR, 2017.
- Dumoulin et al. *Adversarially Learned Inference*. ICLR, 2017
- Edwards & Storkey. *Censoring Representations with an Adversary*. ICLR, 2016.
- Ganin and Lempitsky. *Unsupervised domain adaptation by backpropagation*. ICML, 2015.
- Kim and Mnih. *Disentangling by Factorising*. ICML, 2018.
- Madras et al. *Learning Adversarially Fair and Transferable Representations*. ICML, 2018.
- Makhzani et al. *Adversarial Autoencoders*. ICLR Workshop, 2016.
- Mescheder et al. *Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks*. ICML, 2017.
- Schmidhuber. *Learning factorial codes by predictability minimization*. Neural Computation, 1992.
- Tzeng et al. *Simultaneous deep transfer across domains and tasks*. ICCV, 2015.
- Tzeng et al. *Adversarial discriminative domain adaptation*. CVPR, 2017.
- Villegas, et al. *Decomposing motion and content for natural video sequence prediction*. In ICLR, 2017.
- Zhang et al. *Mitigating Unwanted Biases with Adversarial Learning*. AIES, 2018.

# More detailed Variational Auto-encoder derivation

# Directed graphical models



- We assume data is generated by:

$$z \sim p(z) \quad x \sim p(x|z)$$

- $z$  is latent/hidden  $x$  is observed (image)
- Use  $\theta$  to denote parameters of the generative model

# Parameter estimation

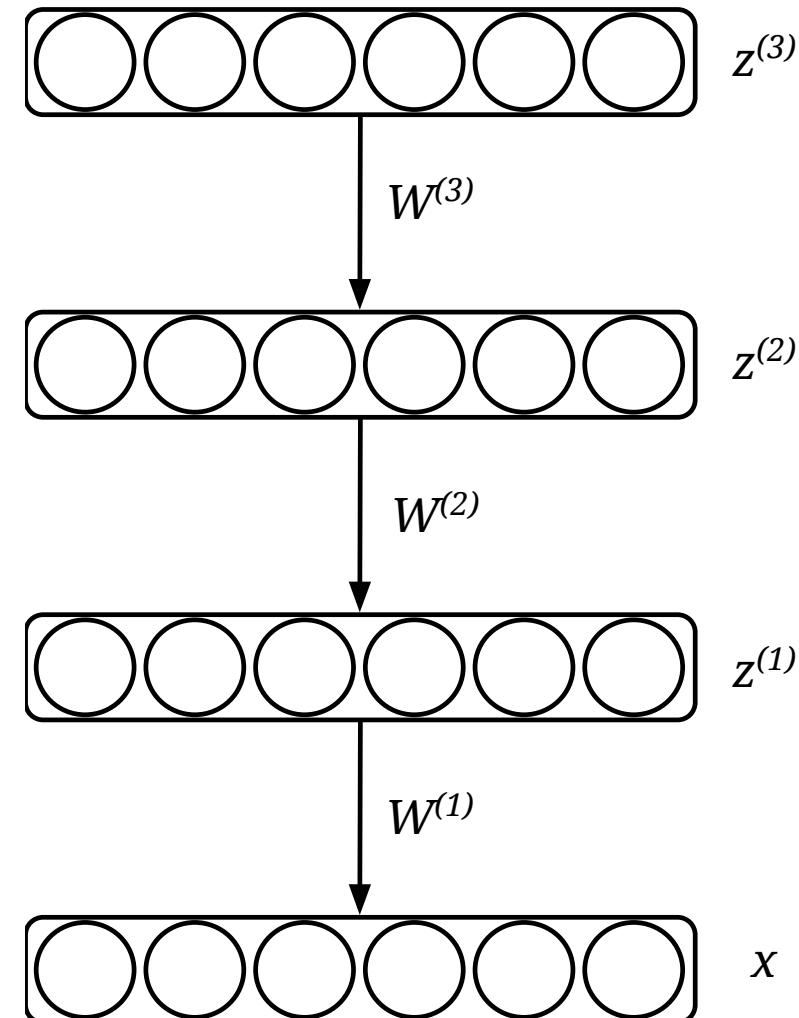
- Given dataset  $\{x_1, \dots, x_n\}$ , maximize likelihood of data under model:

$$\max_{\theta} \sum_{i=1}^n \log p(x_i; \theta) = \max_{\theta} \sum_{i=1}^n \sum_z \log p(x_i, z; \theta)$$

- This quantity often intractable, difficult to optimize directly
- Can be optimized with iterative Expectation Maximization (EM) algorithm
  - Fix parameters and compute log likelihood wrt  $p(z|x; \theta^t)$
  - Fix  $z$  find parameters  $\theta^{(t+1)}$  to maximize log likelihood

# Parameter estimation

- Standard EM requires access to posterior  $p(z|x)$
- For the deep neural net models we care about this is infeasible
- Solution: introduce *variational* approximation  $q(z; \phi)$  to  $p(z|x)$
- Will give bound on log likelihood



## Bounding the marginal likelihood

Recall Jenson's inequality: When  $f$  is concave,  $f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$

$$\begin{aligned}\log p(x) &= \log \int_z p(x, z) \\&= \log \int_z q(z) \frac{p(x, z)}{q(z)} \\&\geq \int_z q(z) \log \frac{p(x, z)}{q(z)} = L(x; \theta, \phi) \quad (\text{by Jenseons inequality}) \\&= \int_z q(z) \log p(x, z) - \int_z q(z) \log q(z) \\&= \underbrace{\mathbb{E}_{q(z)}[\log p(x, z)]}_{\text{Expectation of joint distribution}} + \underbrace{H(q(z))}_{\text{Entropy}}\end{aligned}$$

# Learning directed graphical models

- Maximize bound on likelihood of data:

$$\max_{\theta} \sum_{i=1}^N \log p(x_i; \theta) \geq \max_{\theta, \phi_1, \dots, \phi_N} \sum_{i=1}^N L(x_i; \theta, \phi_i)$$

- Historically, used different  $\phi_i$  for every data point
  - But we'll move away from this soon..
- Can still use EM style algorithm to iteratively optimize
- For more info see Blei *et al.* (2003)

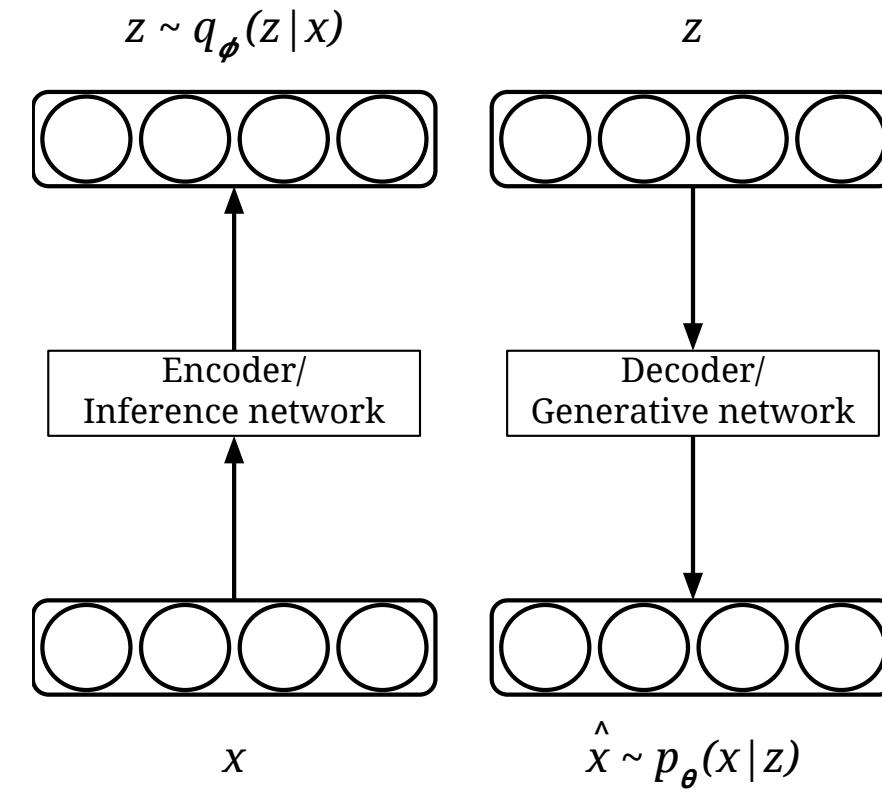
## New method of learning: approximate inference model

- Instead of having different variational parameters for each data point, fit a conditional parametric function
- The output of this function will be the parameters of the variational distribution  $q(z|x)$
- Instead of  $q(z)$  we have  $q_\phi(z|x)$   
Evidence Lower BOund (ELBO)
- ELBO becomes:

$$L(x; \theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z)]}_{\text{Expectation of joint distribution}} + \underbrace{H(q_\phi(z|x))}_{\text{Entropy}}$$

# Variational autoencoder

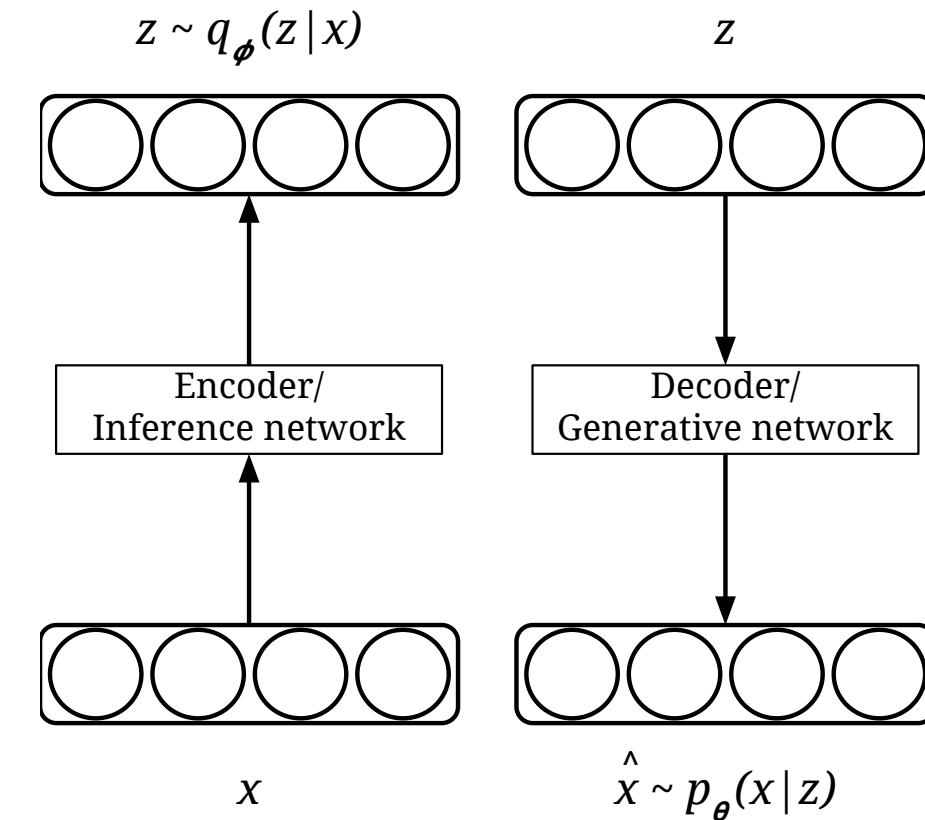
- *Encoder* network maps from image space to latent space
  - Outputs parameters of  $q_\phi(z|x)$
- *Decoder* maps from latent space back into image space
  - Outputs parameters of  $p_\theta(x|z)$



[Kingma & Welling (2013)]

## Example

- *Encoder* network outputs mean and variance of Normal distribution
  - $q_\phi(z|x) = \mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$
- *Decoder* network outputs mean (and optionally variance) of Normal distribution
  - $p_\theta(x|z) = \mathcal{N}(\mu_\theta(z), \mathbf{I})$



[Kingma & Welling (2013)]

# Variational autoencoder

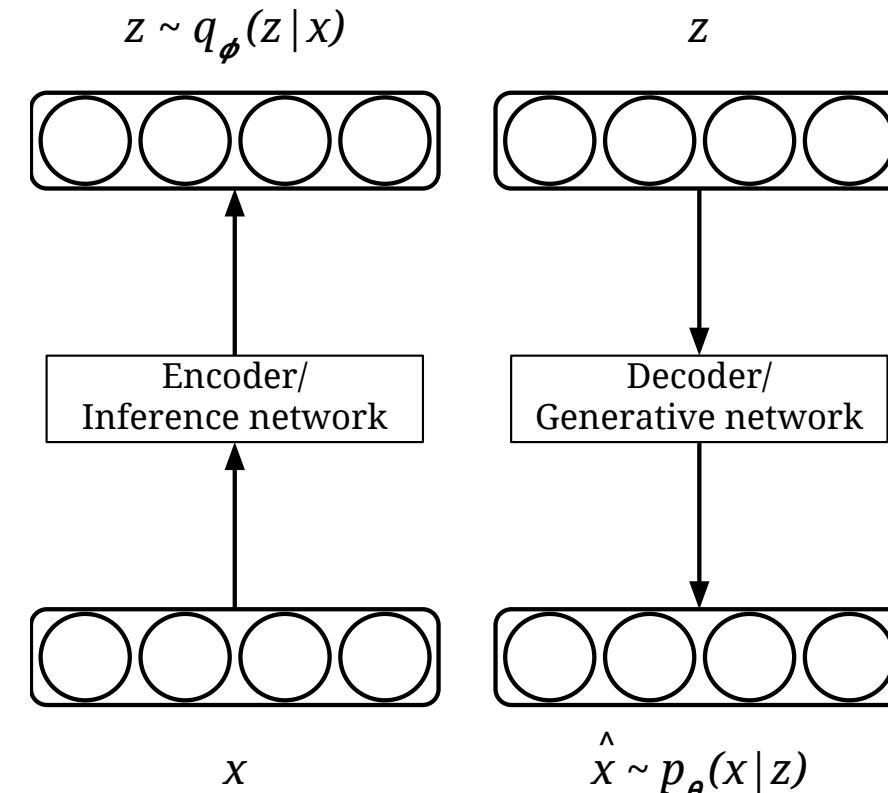
- Rearranging the ELBO:

$$\begin{aligned} L(x; \theta, \phi) &= \int_z q(z|x) \log \frac{p(x, z)}{q(z|x)} \\ &= \int_z q(z|x) \log \frac{p(x|z)p(z)}{q(z|x)} \\ &= \int_z q(z|x) \log p(x|z) + \int_z q(z|x) \log \frac{p(z)}{q(z|x)} \\ &= \mathbb{E}_{q(z|x)} \log p(x|z) - \mathbb{E}_{q(z|x)} \log \frac{q(z|x)}{p(z)} \\ &= \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}} \end{aligned}$$

# Variational autoencoder

- Inference network outputs parameters of  $q_\phi(z|x)$
- Generative network outputs parameters of  $p_\theta(x|z)$
- Optimize  $\theta$  and  $\phi$  jointly by maximizing ELBO:

$$L(x; \theta, \phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$



# Stochastic gradient variation bayes (SGVB) estimator

- Reparameterization trick : re-parameterize  $z \sim q_\phi(z|x)$  as

$$z = g_\phi(x, \epsilon) \text{ with } \epsilon \sim p(\epsilon)$$

- For example, with a Gaussian can write  $z \sim \mathcal{N}(\mu, \sigma^2)$  as

$$z = \mu + \epsilon\sigma^2 \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

[Kingma & Welling (2013); Rezende *et al.* (2014)]

## Stochastic gradient variation bayes (SGVB) estimator

$$L(x; \theta, \phi) = \underbrace{\mathbb{E}_{q(z|x)} \log p(x|z)}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(z|x)||p(z))}_{\text{Prior term}}$$

- Using reparameterization trick we form Monte Carlo estimate of reconstruction term:

$$\begin{aligned} \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) &= \mathbb{E}_{p(\epsilon)} \log p_\theta(x|g_\phi(x, \epsilon)) \\ &\simeq \frac{1}{L} \sum_{i=1}^L \log p_\theta(x|g_\phi(x, \epsilon)) \quad \text{where } \epsilon \sim p(\epsilon) \end{aligned}$$

- KL divergence term can often be computed analytically  
(eg. Gaussian)

## VAE learned manifold



6666660000000000000000  
444222222000000000000002  
442222222285555000000002  
44222222223333333555555533  
444222222333333333355555533  
449933333333333555555533  
449999993333333333355555533  
4499999993333333333355555533  
7999999993333333333388888887  
7999999998888888888888888887  
7999999998888888888888888887  
7999999998888888888866666557  
79999999998888886666666557  
799999999998888866666666557  
7999999999998888666666666557  
79999999999998886666666666557  
799999999999998866666666666557  
7999999999999998666666666666557  
79999999999999998666666666666557  
799999999999999998666666666666557  
7999999999999999998666666666666557  
799999999999999999986666666666666557  
79999999999999999999866666666666666557  
7999999999999999999998666666666666666557

[Kingma & Welling (2013)]

# VAE samples



(a) 2-D latent space

(b) 5-D latent space

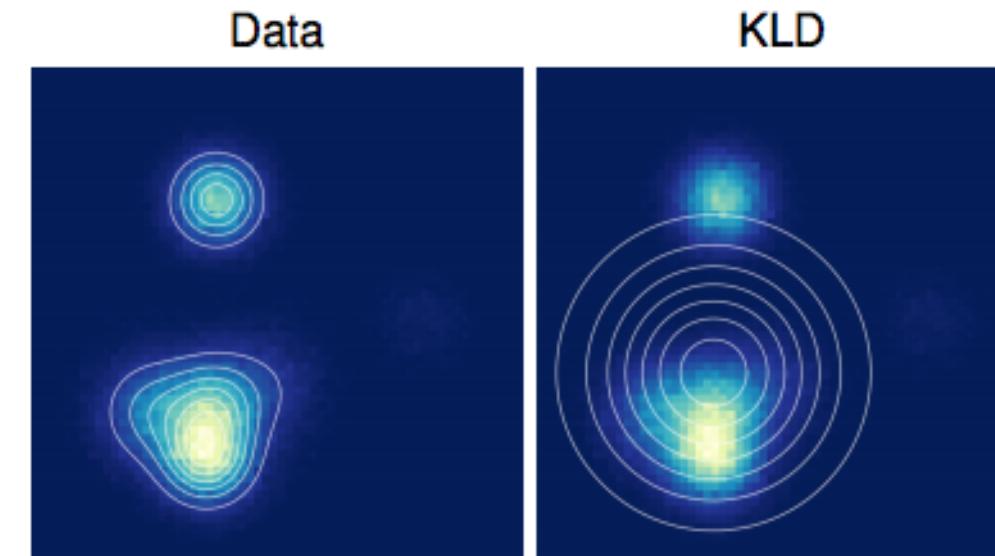
(c) 10-D latent space

(d) 20-D latent space

[Kingma & Welling (2013)]

# VAE tradeoffs

- Pros:
  - Theoretically pleasing
  - Optimizes bound on likelihood
  - Easy to implement
- Cons:
  - Samples tend to be blurry
    - Maximum likelihood minimizes  $D_{KL}(p_{data} || p_{model})$



[Theis *et al.* (2016)]