Deep Learning & Engineering Applications

14. Generative Adversarial Networks (GAN)

JIDONG J. YANG

COLLEGE OF ENGINEERING

UNIVERSITY OF GEORGIA

Taxonomy of Generative Models

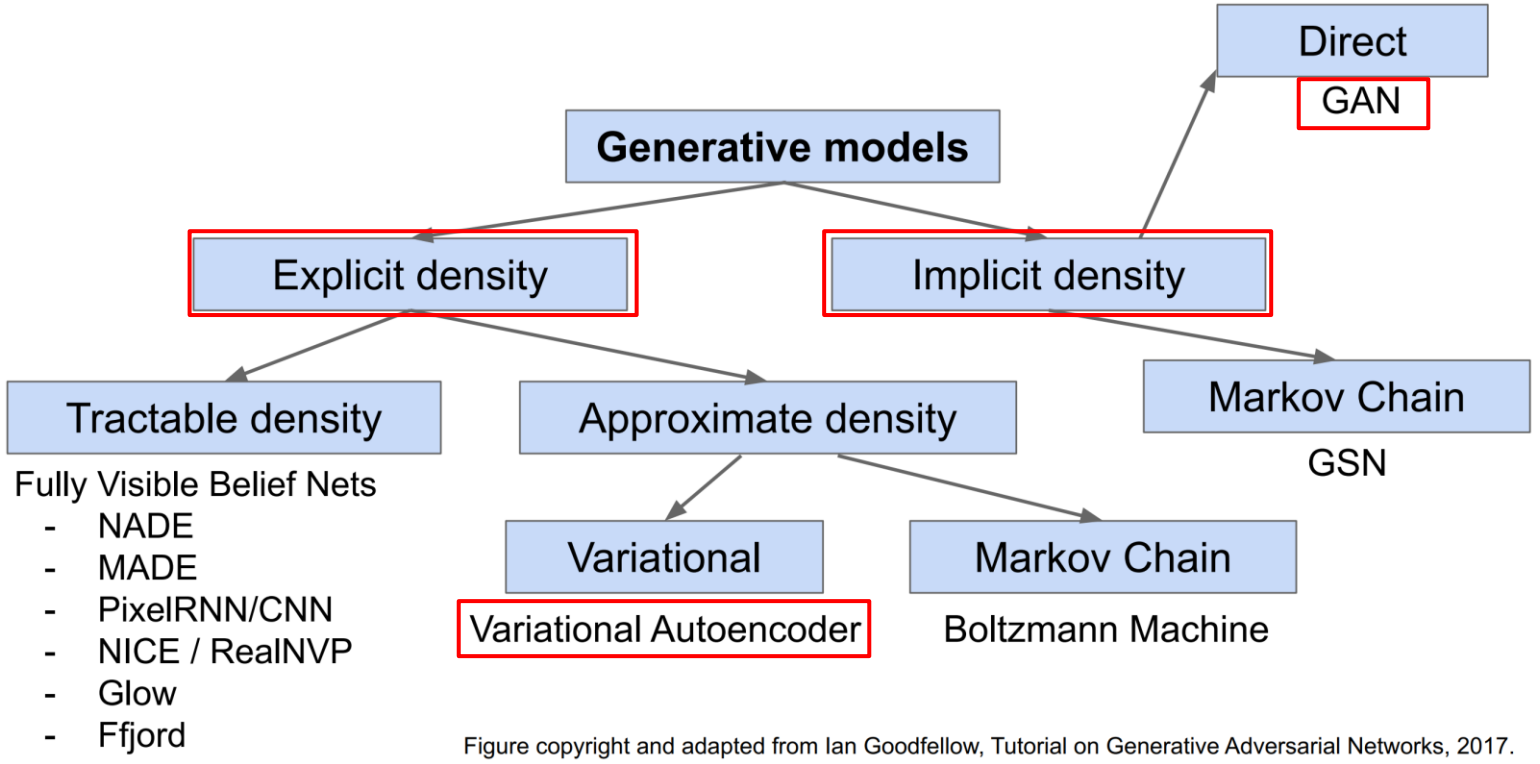


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Image credit: Fei-Fei Li, Ranjay Krishna, Danfei Xu, Lecture 12: Generative Models, May 11, 2021.
The Original Figure is from Ian Goodfellow, NIPS 2016 Tutorial: Generative Adversarial Networks https://arxiv.org/pdf/1701.00160.pdf

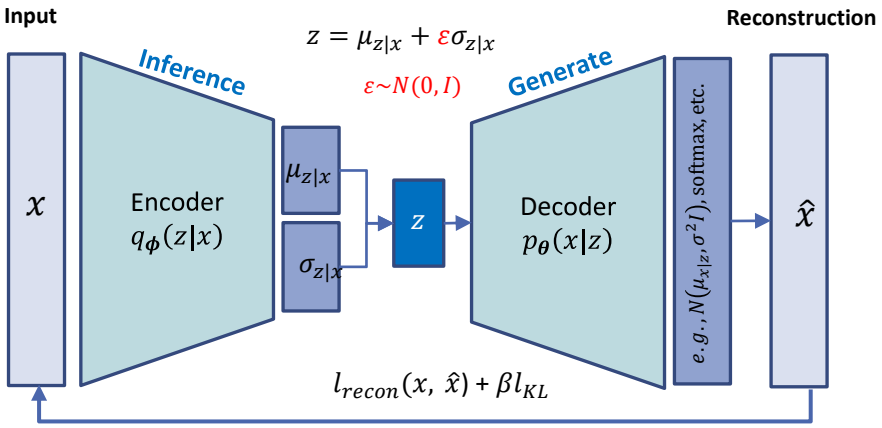
GAN vs. VAE

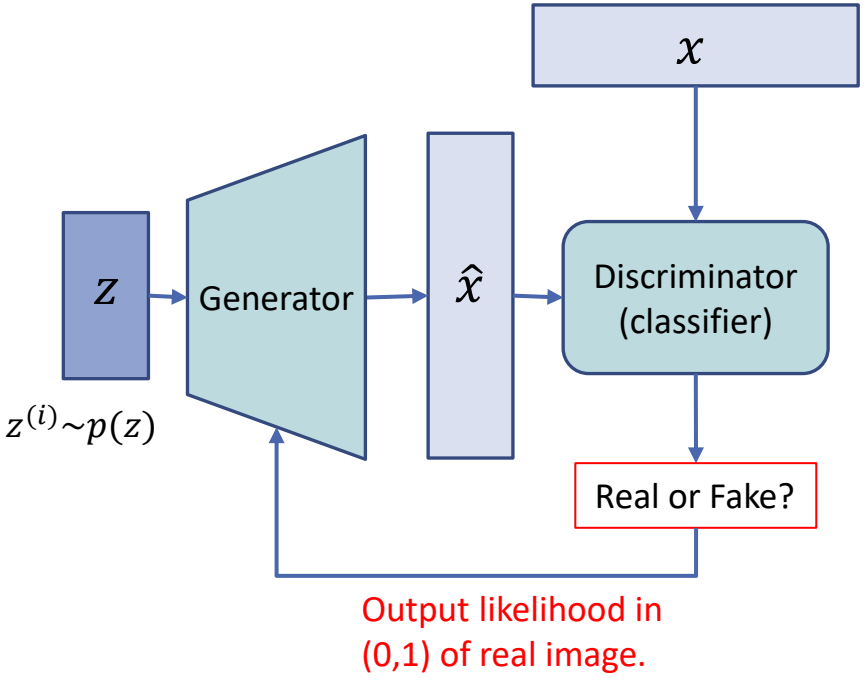
VAE

- Explicitly define a density with latent variables **z** to compute data likelihood (intractable)
- Approximate the data likelihood with a tractable lower bound (VLB or ELBO)

GAN

- Just sampling without explicitly modeling density.
- Use a discriminator to generate learning signal.
- Learning (training) an implicit model through a two-player game.
  - Generator try to fool discriminator by generating real-looking $\hat{x}$ (e.g., images)
  - Discriminator try to distinguish between real x and fake $\hat{x}$.





Two perspectives of GAN

Two-player game

Minimization of a “distance” or “loss function” learned by the discriminator

Training GAN – Two-player Game

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Log-prob that D correctly predict real data x are real.

Log-prob that D correctly predict generated data G(z) are generated.

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

GAN Algorithm

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

[Goodfellow et al., 2014] https://arxiv.org/abs/1406.2661

Generative models as distance minimization

The objective of generative models is often to minimize a distance or divergence.

Maximum likelihood is equivalent to minimizing KL divergence between data distribution: $p(x)$ and model-generated data distribution: $q_\theta(x)$.

$$KL(p(x)||q_\theta(x)) = \int p(x) \log \frac{p(x)}{q_\theta(x)} dx$$

$$KL(p(x)||q_\theta(x)) = 0 \longrightarrow q_\theta(x) = p(x)$$

Effects of different KL divergences?

Forward KL: $KL(p||q_\theta) = \int p(x) \log \frac{p(x)}{q_\theta(x)} dx$

$$\min_\theta KL(p||q_\theta) = \min_\theta \int p(x)[\log p(x) - \log q_\theta(x)]\, dx$$

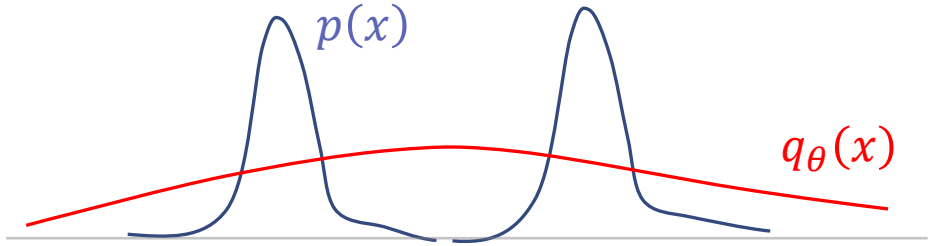
$$= \max_\theta \int p(x) \log q_\theta(x)\, dx$$

Reverse KL: $KL(q_\theta||p) = \int q_\theta(x) \log \frac{q_\theta(x)}{p(x)} dx$

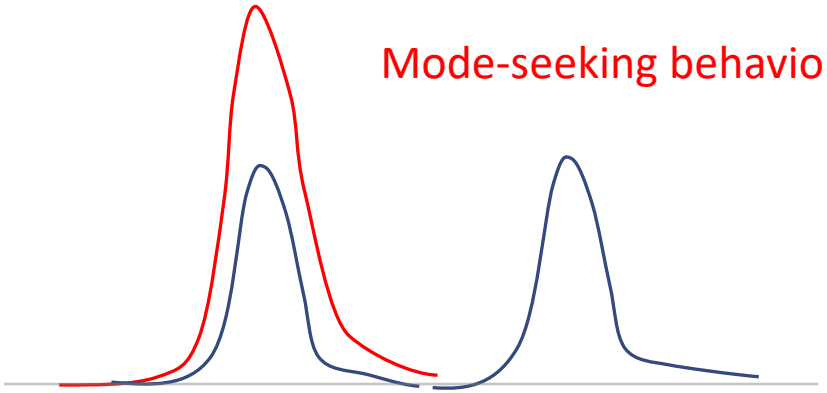
$$\min_\theta KL(q_\theta||p) = \min_\theta \int q_\theta(x)[\log q_\theta(x) - \log p(x)]\, dx$$

$$= \max_\theta \int [q_\theta(x) \log p(x) - q_\theta(x) \log q_\theta(x)]\, dx$$

Maximum Likelihood;
Mean-seeking behavior



Mode-seeking behavior



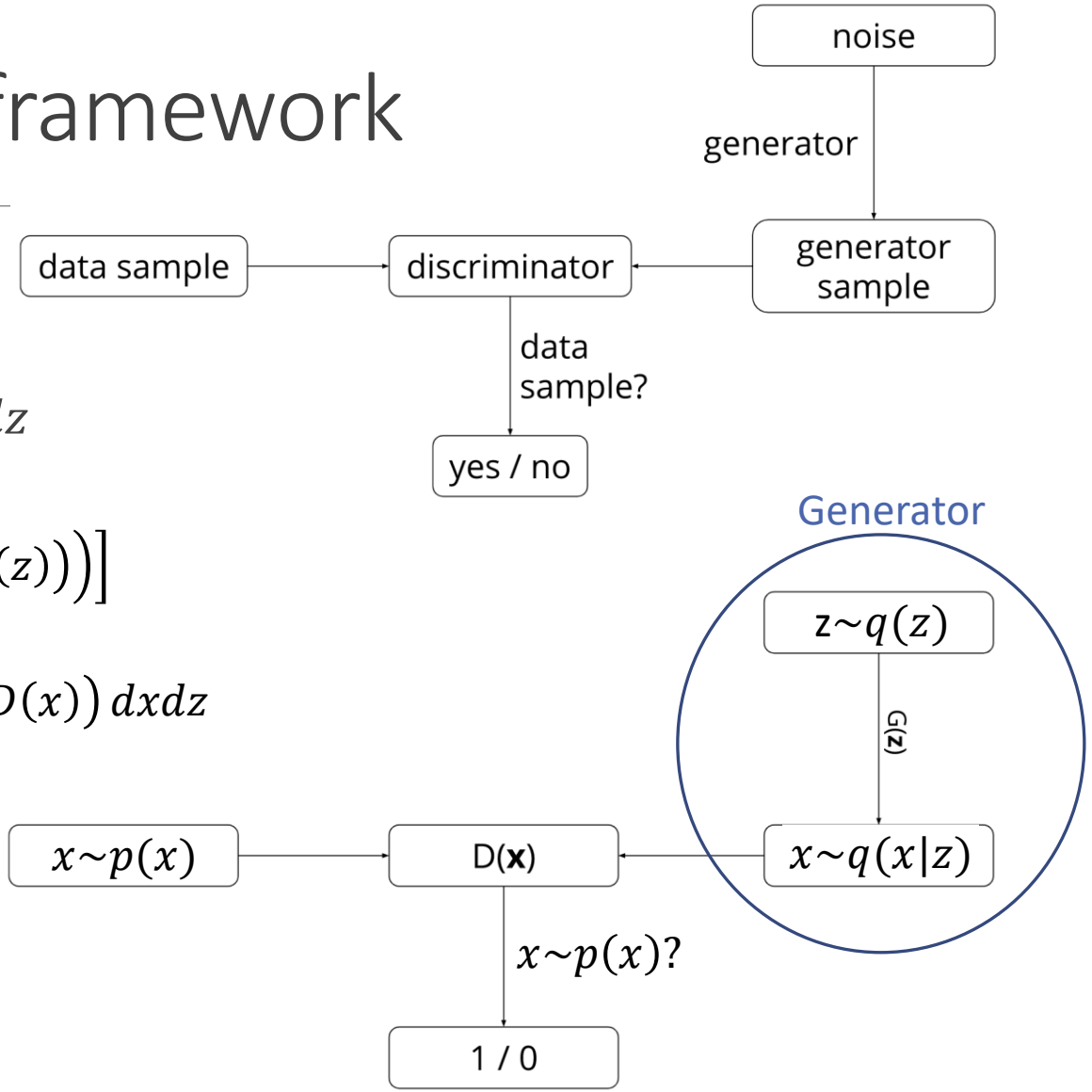
View GAN as a probabilistic framework

Two marginal distributions:

- Data marginal, $p(x)$
- Model (generator) marginal, $q(x) = \int q(z)q(x|z)dz$

$$\min_G \max_D V(D,G) = E_{p(x)}[\log D(x)] + E_{q(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

$$= \int p(x)\log\big(D(x)\big)\,dx + \iint q(z)q(x|z)\log\big(1 - D(x)\big)\,dxdz$$



Are GANs doing divergence minimization?

If the discriminator D is optimal, the generator minimizes the Jensen Shannon Divergence (JSD) between the true and generated distributions.

JSD is a symmetric and bounded metric.

$$\begin{aligned} \mathrm{JS}(p,q) &:= \frac{1}{2}\left(\mathrm{KL}\left(p:\frac{p+q}{2}\right)+\mathrm{KL}\left(q:\frac{p+q}{2}\right)\right) \\ &= \frac{1}{2}\int\left(p\log\frac{2p}{p+q}+q\log\frac{2q}{p+q}\right)\mathrm{d}\mu = \mathrm{JS}(q,p) \end{aligned}$$

In comparison

$$\mathrm{KL}(p:q) := \int p\log\frac{p}{q}\mathrm{d}\mu$$

KL is asymmetric and may diverge to infinity.

[Nielsen 2019] https://arxiv.org/pdf/1912.00610.pdf

Are GANs doing divergence minimization?

It can be shown that for a fixed generator, the optimal discriminator is:

$$D^* = \frac{p(x)}{p(x) + q(x)}$$

Given the optimal discriminator, minimizing the value function with respect to the generator parameters is equivalent to minimizing the Jensen-Shannon divergence between p(x) and q(x).

This implies that as training progresses, the generator produces synthetic samples that look more and more like the training data.

Proof

$$D^* = \frac{p(x)}{p(x)+q(x)}$$

$$\begin{aligned}\mathrm{JS}(p,q) &:= \frac{1}{2}\left(\mathrm{KL}\left(p:\frac{p+q}{2}\right)+\mathrm{KL}\left(q:\frac{p+q}{2}\right)\right)\\ &= \frac{1}{2}\int\left(p\log\frac{2p}{p+q}+q\log\frac{2q}{p+q}\right)\mathrm{d}\mu = \mathrm{JS}(q,p)\end{aligned}$$

$$V(D,G) = \int p(x)\log\big(D(x)\big)\,dx + \iint q(z)q(x|z)\log\big(1-D(x)\big)\,dxdz$$

$$= \int p(x)\log\big(D(x)\big)\,dx + \int q(x)\log\big(1-D(x)\big)dx$$

$$= \int\{p(x)\log\big(D(x)\big)+q(x)\log\big(1-D(x)\big)\}dx \qquad (1)$$

$\max_D V(D,G) = ?$ $\qquad \frac{\partial V(D,G)}{\partial D} = \frac{p(x)}{D(x)} - \frac{q(x)}{1-D(x)} = 0 \qquad D^* = \frac{p(x)}{p(x)+q(x)} \qquad (2)$

Plug (2) in (1): $\quad V(D^*,G) = \int\left\{p(x)\log\left(\frac{p(x)}{p(x)+q(x)}\right)+q(x)\log\left(\frac{q(x)}{p(x)+q(x)}\right)\right\}dx$

$$\min_G V(D^*,G) = \min_G \int\left\{p(x)\log\left(\frac{p(x)}{p(x)+q(x)}\right)+q(x)\log\left(\frac{q(x)}{p(x)+q(x)}\right)\right\}dx$$

$$= \min_G \int\left\{p(x)\log\left(\frac{2p(x)}{p(x)+q(x)}\right)-p(x)log2+q(x)\log\left(\frac{2q(x)}{p(x)+q(x)}\right)-q(x)log2\right\}dx$$

$$= \min_G \int\left\{p(x)\log\left(\frac{2p(x)}{p(x)+q(x)}\right)+q(x)\log\left(\frac{2q(x)}{p(x)+q(x)}\right)\right\}dx - \int p(x)log2\,dx - \int q(x)log2\,dx$$

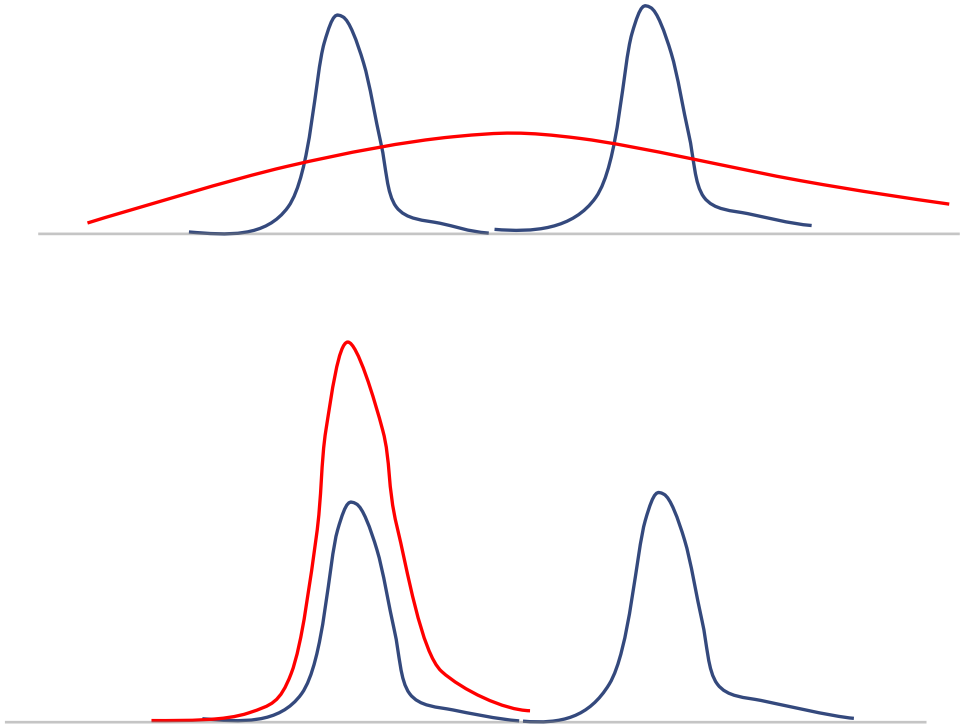
$$= \min_G 2\,JS(p,q) - log4 \qquad = \min_G JS(p,q)$$

What does it mean by optimizing JSD?

By definition, JSD is a mixture of forward KL and reverse KL. As such, it inherits behaviors from both forward KL (mean seeking) and reverse KL (mode seeking).

$$\begin{aligned}\mathrm{JS}(p,q) &:= \frac{1}{2}\left(\mathrm{KL}\left(p:\frac{p+q}{2}\right)+\mathrm{KL}\left(q:\frac{p+q}{2}\right)\right)\\ &= \frac{1}{2}\int\left(p\log\frac{2p}{p+q}+q\log\frac{2q}{p+q}\right)\mathrm{d}\mu = \mathrm{JS}(q,p)\end{aligned}$$

Which behavior prevails depending on the initialization of the model.



The discriminator will not be optimal in practice

$$D^* = \frac{p(x)}{p(x) + q(x)}$$

Limited computational resources.

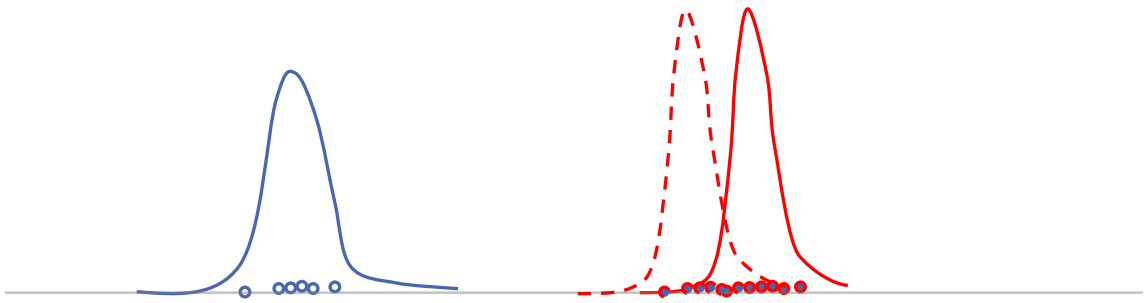
Only have access to samples rather than the true data distribution.

Issues with KL and JSD

No learning signal from KL or JSD divergence if there is non overlapping support between the data distribution $p(x)$ and the model-generated distribution, $q_\theta(x)$.

$$KL(p||q_\theta) = \int p(x) \log \frac{p(x)}{q_\theta(x)} dx \quad = \infty$$

$$JS(p, q_\theta) = \frac{1}{2} \int \left(p(x) \log \frac{2p(x)}{p(x) + q_\theta(x)} + q_\theta(x) \log \frac{2q_\theta(x)}{p(x) + q_\theta(x)} \right) dx \quad = log2$$



Can we use other divergences/distances?

$$\min_G \max_D V(D,G)$$

Wasserstein Distance: $W(p, q_\theta) = \sup_{\|f\|_L \leq 1} E_{p(x)} f(x) - E_{q_\theta(x)} f(x)$

f : 1-Lipschitz functions.

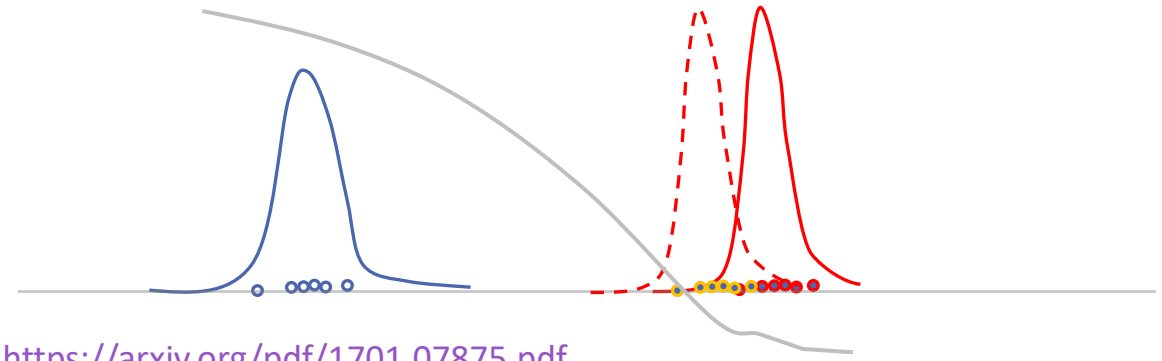
$|f(x) - f(y)| \leq |x - y|$

$$\min_G W(p, q_\theta) = \min_G \sup_{\|f\|_L \leq 1} E_{p(x)} f(x) - E_{q_\theta(x)} f(x)$$

Computationally Intractable

Wasserstein GAN: $\min_G W(p, q_\theta) = \min_G \max_{\|D\|_L \leq 1} E_{p(x)} D(x) - E_{q_\theta(z)} D(G(z))$

Learned distance



[Arjovsky et al., 2017] https://arxiv.org/pdf/1701.07875.pdf

Other divergences:
maximum mean discrepancy (MMD), f-divergence, etc.

Learned loss function

In practice, GAN do not do divergence minimization.

The discriminator D is smooth approximation of the decision boundary of the underlying divergence, which can be seen as learning a "loss function" or "distance" between data distribution and model distribution that provide useful gradients to the model.

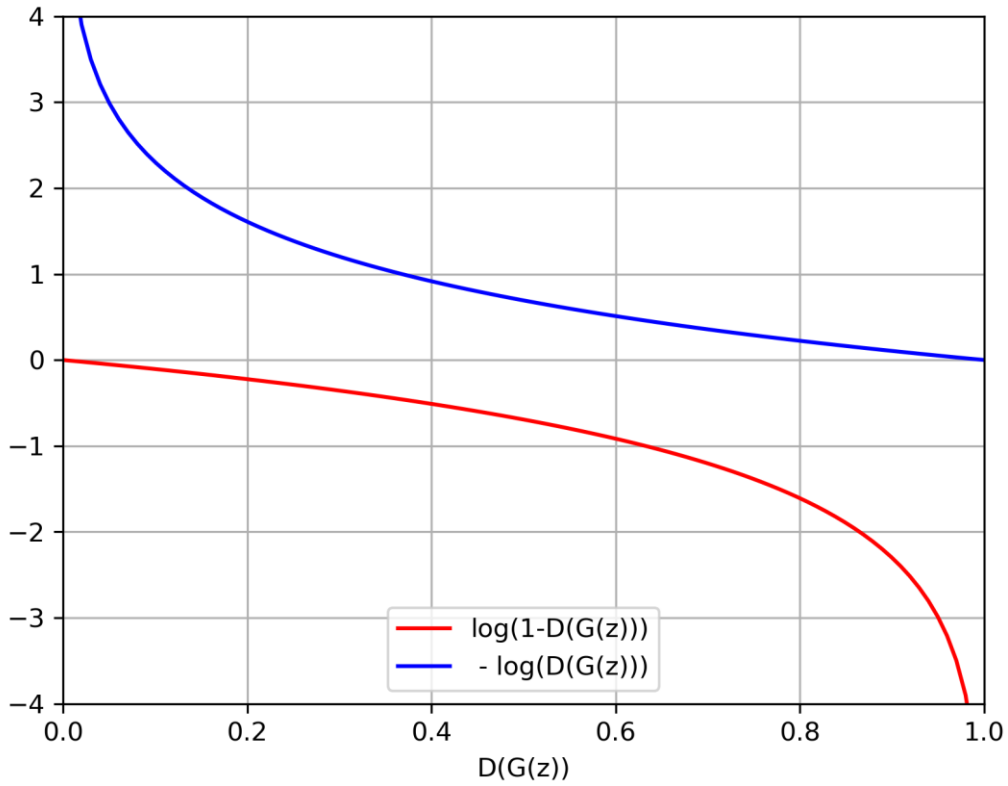
$$\min_G \max_D V(D, G)$$

Practical Trick

Rather than training G to minimize log(1 − D(G(z))), train G to maximize log D(G(z)).

Result in the same fixed point of the dynamics of G and D.

Provide much stronger gradients early in learning.



GAN Algorithm (with the practical trick)

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:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

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 ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Evaluate generative models

Inception Score

- Use a pretrained ImageNet classifier to compare (via KL divergence) the distribution of labels from data and the distribution of labels from samples.
- Measures:
  - Sample quality
  - Dropping classes (e.g., mode collapse)
  - Correlates with human evaluation
  - Does not measure differences beyond class labels
  - Requires pretrained classifier

Evaluate generative models

Frechet Inception Distance

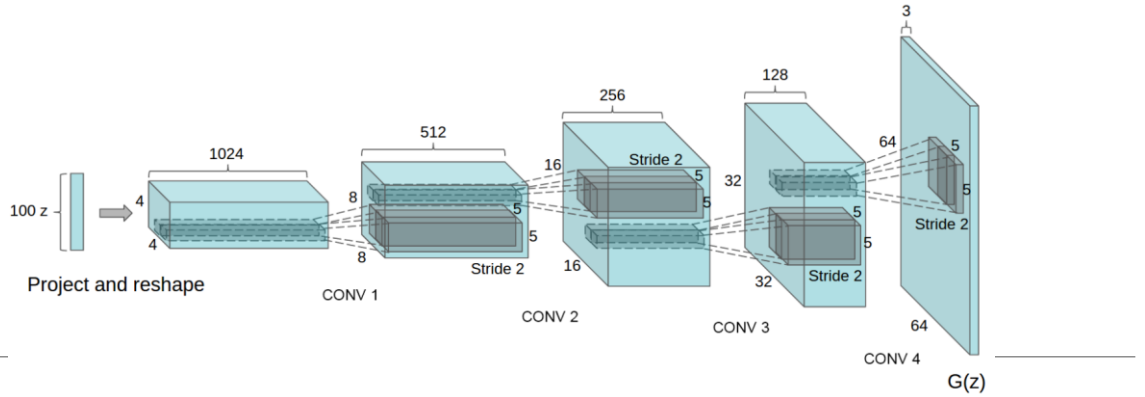
- Use a pretrained ImageNet classifier to compare (via Frechet distance) the distribution of layer features from data and the distribution of layer features from samples.
- Measures:
  - Sample quality
  - Dropping classes
  - Captures feature level statistics
  - Correlates with human evaluation
  - Requires pretrained classifier
  - Biased for a small number of samples (new measure for fix: Kernel Inception Distance (KID), see [Binkowski et al., 2018] https://arxiv.org/abs/1801.01401)

Check nearest neighbors for overfitting

Take a generated image and check the most similar images (similarity measured in the feature space of a pretrained ImageNet classifier) in the dataset.

The image generated should be similar to those in the dataset (i.e., meaningful interpolation) rather than memorized “copies” of images in the dataset.

Deep Convolutional Generative Adversarial Networks (DCGAN)



Downsample with strided convolutions.

Non-linearity: ReLU for generator, Leaky-ReLU (0.2) for discriminator

Output non-linearity: tanh for Generator, sigmoid for Discriminator

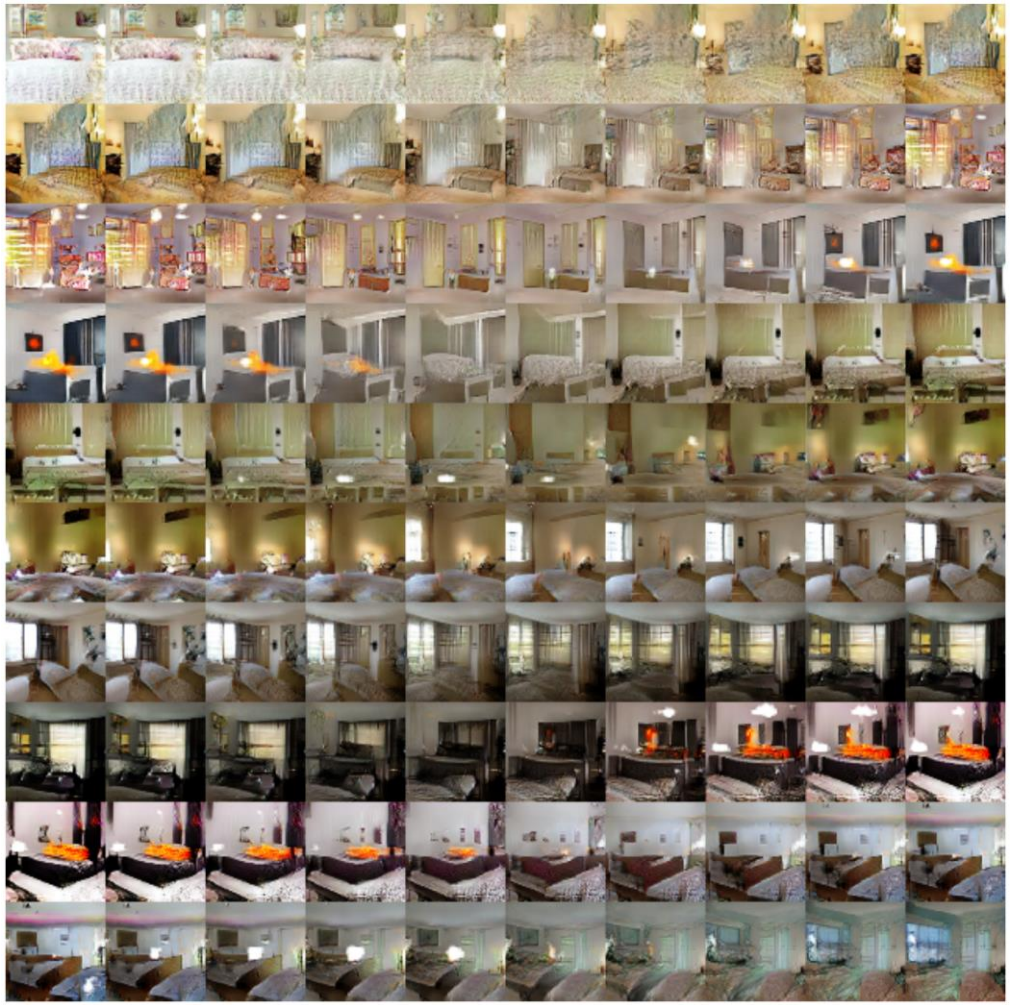
BatchNorm used to prevent mode collapse

```python
class Discriminator(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is (nc) x 64 x 64
            nn.Conv2d(nc, ndf, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*2) x 16 x 16
            nn.Conv2d(ndf * 2, ndf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*4) x 8 x 8
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            # state size. (ndf*8) x 4 x 4
            nn.Conv2d(ndf * 8, 1, 4, 1, 0, bias=False),
            nn.Sigmoid()
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)

        return output.view(-1, 1).squeeze(1)
```

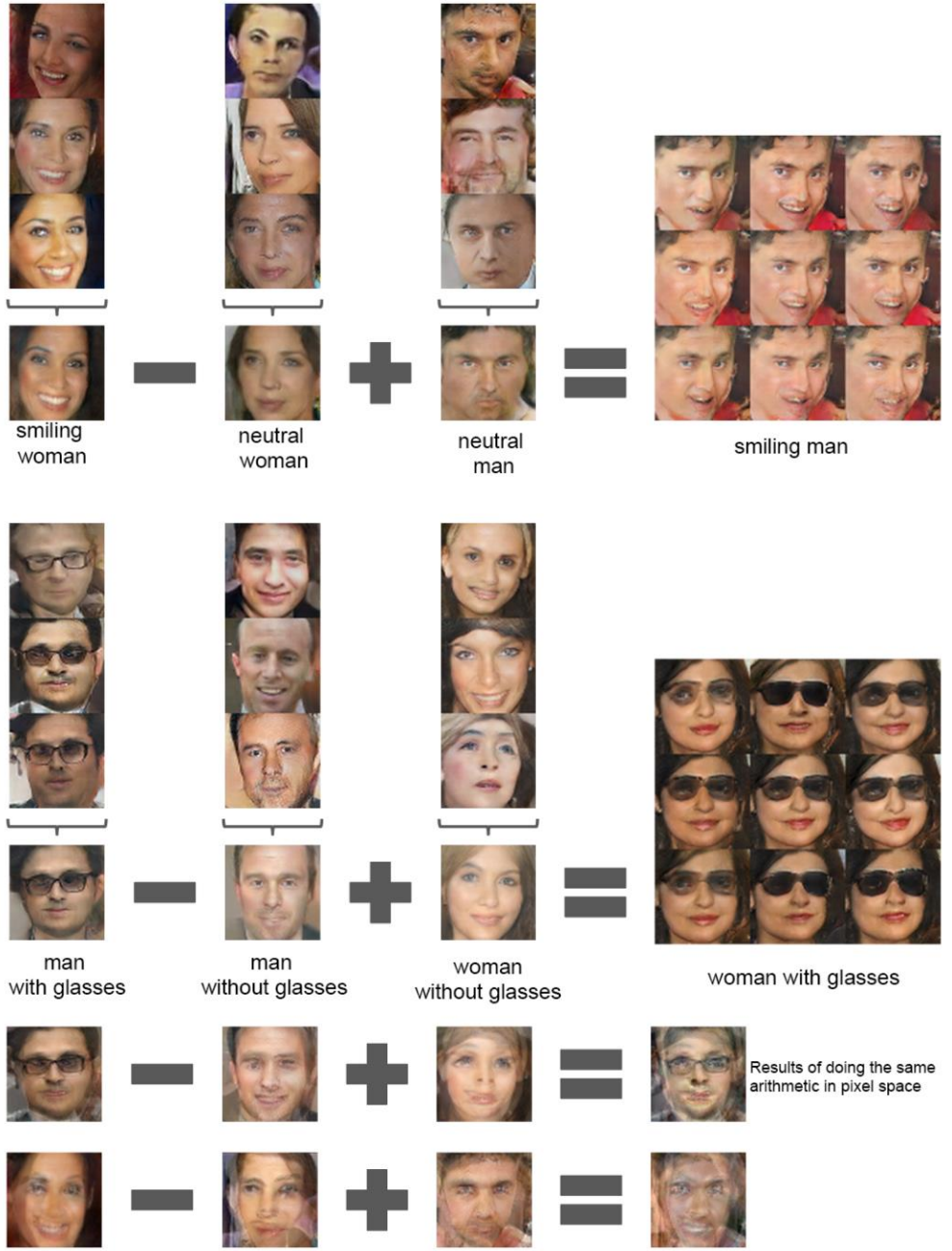
```python
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d(     nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d(ngf * 4, ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d(ngf * 2,     ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d(    ngf,      nc, 4, 2, 1, bias=False),
            nn.Tanh()
            # state size. (nc) x 64 x 64
        )

    def forward(self, input):
        if input.is_cuda and self.ngpu > 1:
            output = nn.parallel.data_parallel(self.main, input, range(self.ngpu))
        else:
            output = self.main(input)
        return output
```

[Radford et al. 2015] https://arxiv.org/pdf/1511.06434.pdf

DCGAN



[Radford et al. 2015] https://arxiv.org/pdf/1511.06434.pdf



BigGANs

Make GANS big

- Big batches (e.g., 2048)
- Big models
- Big datasets
- Big (high resolution) images

Trained on ImageNet (1.2M images) and JFT (300M images)

Large empirical study

- Hinge loss in D, spectral norm, self-attention, projection discriminator, orthogonal regularization, skip connection from noise, class label embedding shared across layers, truncation trick, etc.

Class condition samples generated

[Brock et al., 2019] https://arxiv.org/pdf/1809.11096.pdf

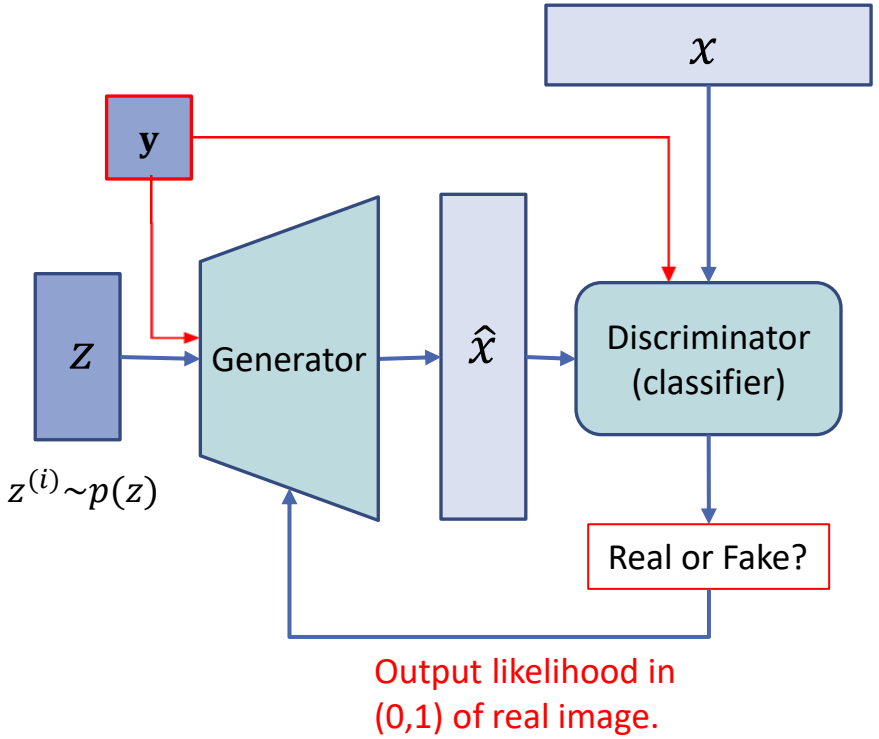
Unconditional vs Conditional

Unconditional

- Generate a sample with no control over what kind of sample to generate.

Conditional

- Specify what kind of samples to generate.



GANs for Representation Learning

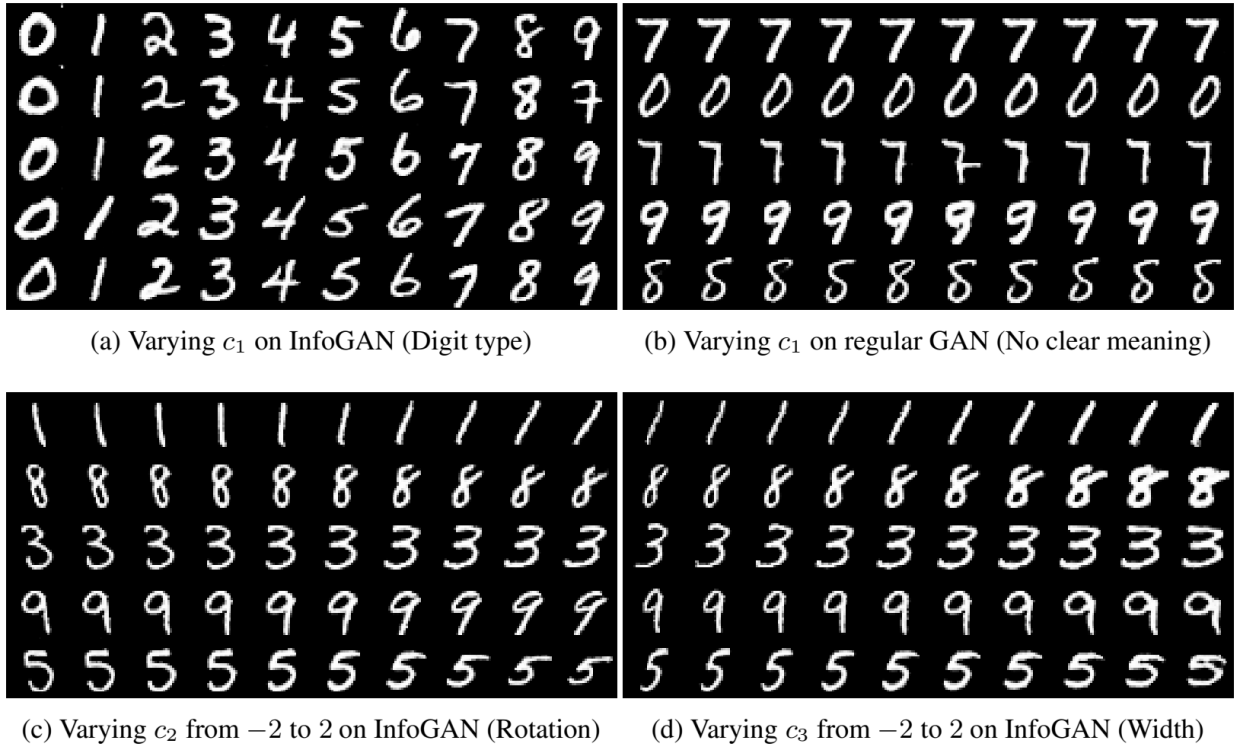
InfoGAN

There should be high mutual information between latent codes c and generator distribution G(z, c).

$$\min_{G,Q} \max_{D} V_{\text{InfoGAN}}(D, G, Q) = V(D, G) - \lambda L_I(G, Q)$$

$$\begin{aligned}
I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\
&= \mathbb{E}_{x \sim G(z,c)}[\mathbb{E}_{c' \sim P(c|x)}[\log P(c'|x)]] + H(c) \\
&= \mathbb{E}_{x \sim G(z,c)}[\underbrace{D_{\text{KL}}(P(\cdot|x) \parallel Q(\cdot|x))}_{\geq 0} + \mathbb{E}_{c' \sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&\geq \mathbb{E}_{x \sim G(z,c)}[\mathbb{E}_{c' \sim P(c|x)}[\log Q(c'|x)]] + H(c) \\
&= E_{c \sim P(c), x \sim G(z,c)}[\log Q(c|x)] + H(c) \\
&= L_I(G, Q)
\end{aligned}$$

- Generator learns to associate a discrete 10-way categorical latent variable (c_1) with class labels.
- Two continuous variables (c_2, c_3) captured rotation and width.



(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)

(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

[Chen et al., 2016] https://arxiv.org/pdf/1606.03657.pdf

InfoGAN

(a) Rotation

(b) Width

[Chen et al., 2016] https://arxiv.org/pdf/1606.03657.pdf

Adversarially Learned Inference (ALI)/Bidirectional GAN (BiGAN)

Augments GAN's generator with **an additional network**. This network receives a data sample as input and produces a synthetic "z" as output.

Defines two joint distributions:

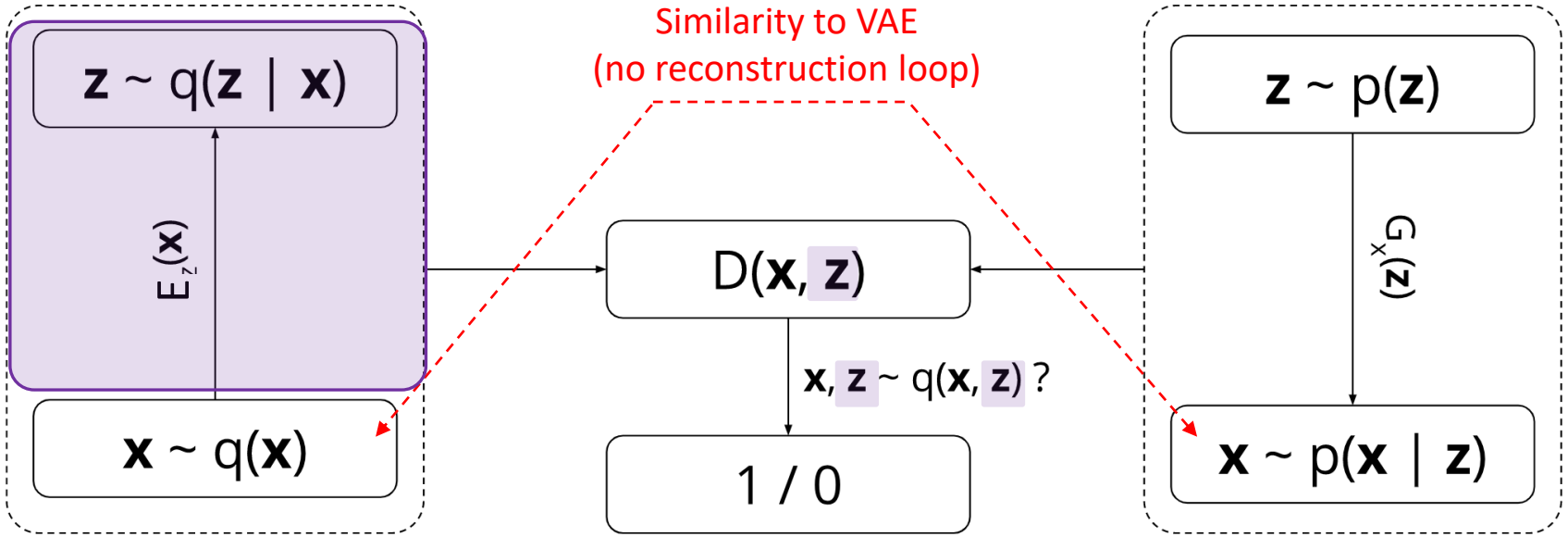
- the ***Encoder (E)*** joint $q(x,z)=q(x)q(z|x)$
- the ***Decoder (G)*** joint $p(x,z)=p(z)p(x|z)$

ALI & BiGAN Models

Deep directed generative models that jointly learns a generation network and an inference network using an adversarial process.

Unlike other approaches to learning inference in deep directed generative models (e.g., VAE), the objective function involves no explicit reconstruction loop. Instead of focusing on achieving a pixel-perfect reconstruction, they tend to produce believable reconstructions with interesting variations.

ALI Probabilistic View



$$\min_G \max_D V(D,G) = \mathbb{E}_{q(\mathbf{x})}[\log(D(\mathbf{x}, E_z(\mathbf{x})))] + \mathbb{E}_{p(\mathbf{z})}[\log(1 - D(G_x(\mathbf{z}), \mathbf{z}))]$$

$$= \iint q(\mathbf{x})q(\mathbf{z} \mid \mathbf{x}) \log(D(\mathbf{x}, \mathbf{z}))d\mathbf{x}d\mathbf{z} + \iint p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}) \log(1 - D(\mathbf{x}, \mathbf{z}))d\mathbf{x}d\mathbf{z}$$

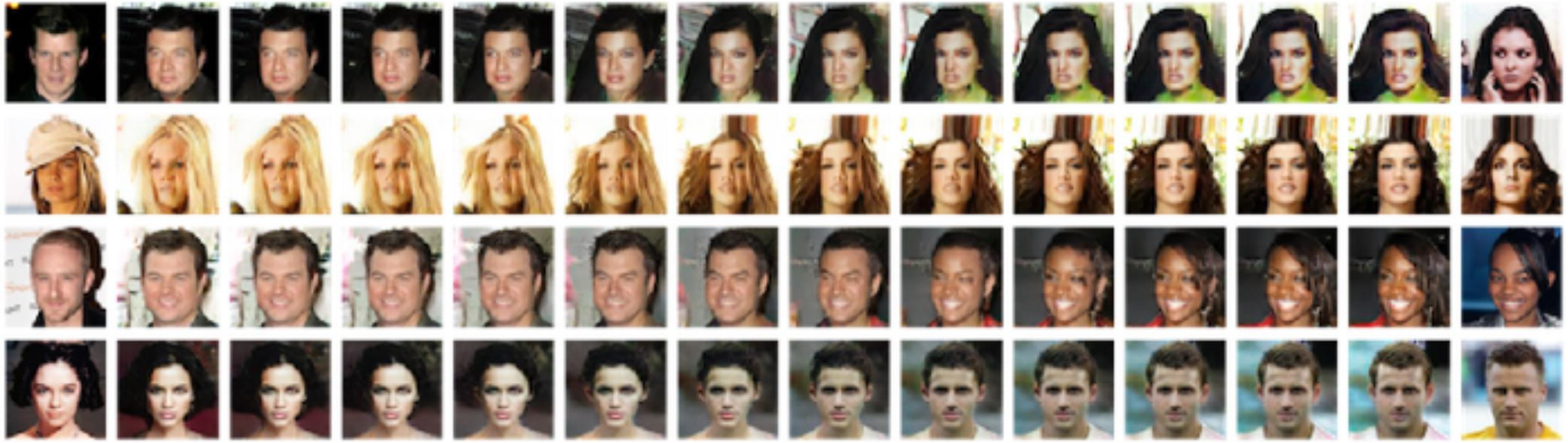
[Dumoulin et al., 2016] https://arxiv.org/abs/1606.00704

Adapted from https://ishmaelbelghazi.github.io/ALI/

Latent space interpolations (ALI)

Sample pairs of validation set examples x1 and x2 and project them into z1 and z2 by sampling from the encoder. Then linearly interpolate between z1 and z2 and pass the intermediary points through the decoder to plot the input-space interpolations.

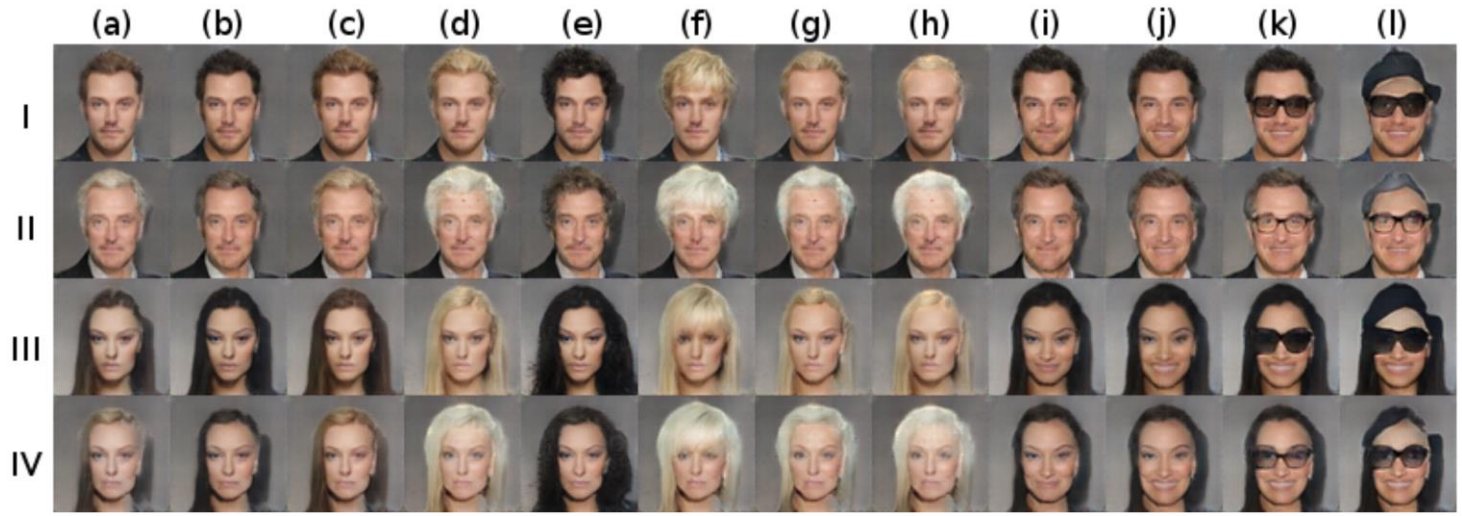
Smooth transitions between pairs of example indicates that ALI is not concentrating its probability mass exclusively around training examples, but rather has learned latent features that generalize well.



[Dumoulin et al., 2016] https://arxiv.org/abs/1606.00704

Conditional Generation (ALI)

- Provide the encoder, decoder and discriminator networks with a conditioning variable y.
- The attributes are linearly embedded in the encoder, decoder and discriminator.
- Apply the conditional version of ALI to CelebA using the dataset's 40 binary attributes.



The row attributes are
- I: male, attractive, young
- II: male, attractive, older
- III: female, attractive, young
- IV: female, attractive, older

The column attributes are
- a: unchanged
- b: black hair
- c: brown hair
- d: blond hair
- e: black hair, wavy hair
- f: blond hair, bangs
- g: blond hair, receding hairline
- h: blond hair, balding
- i: black hair, smiling
- j: black hair, smiling, mouth slightly open
- k: black hair, smiling, mouth slightly open, eyeglasses
- l: black hair, smiling, mouth slightly open, eyeglasses, wearing hat

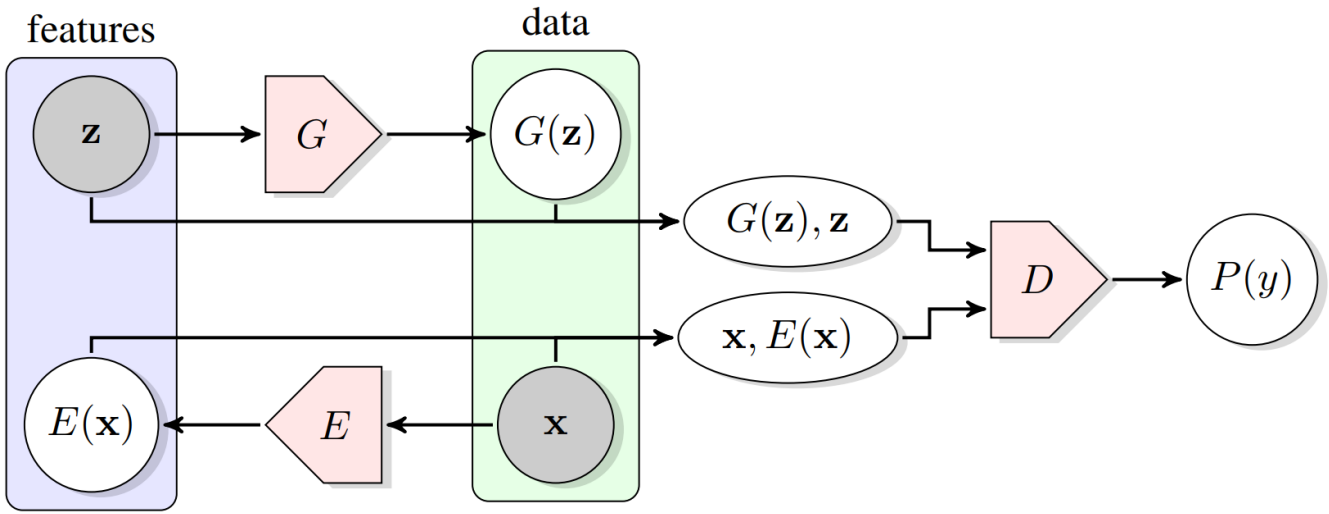
[Dumoulin et al., 2016] https://arxiv.org/abs/1606.00704

Bidirectional Generative Adversarial Networks (BiGANs)

In the global optimum, E and G are inverses, i.e., for all x and z, we have:

- x=G(E(x))
- z=E(G(z))

In practice, this inversion property does not hold perfectly. But the reconstruction still often capture interesting semantics.



$$\min_{G,E} \max_{D} V(D,E,G)$$

$$V(D,E,G) := \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{X}}}\Big[\underbrace{\mathbb{E}_{\mathbf{z}\sim p_E(\cdot|\mathbf{x})}\left[\log D(\mathbf{x},\mathbf{z})\right]}_{\log D(\mathbf{x},E(\mathbf{x}))}\Big] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{Z}}}\Big[\underbrace{\mathbb{E}_{\mathbf{x}\sim p_G(\cdot|\mathbf{z})}\left[\log\left(1-D(\mathbf{x},\mathbf{z})\right)\right]}_{\log(1-D(G(\mathbf{z}),\mathbf{z}))}\Big]$$

[Donahue et al., 2017] https://arxiv.org/pdf/1605.09782v7.pdf

BiGANs

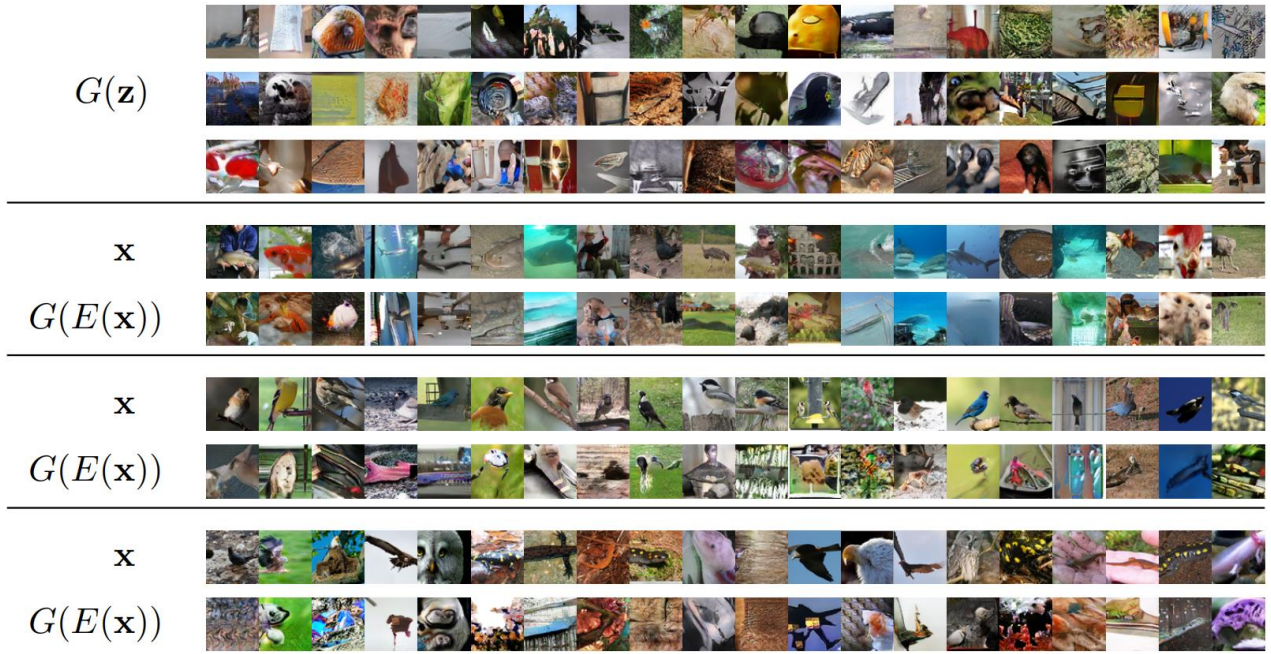


Figure 4: Qualitative results for ImageNet BiGAN training, including generator samples $G(\mathbf{z})$, real data $\mathbf{x}$, and corresponding reconstructions $G(E(\mathbf{x}))$.

[Donahue et al., 2017] https://arxiv.org/pdf/1605.09782v7.pdf

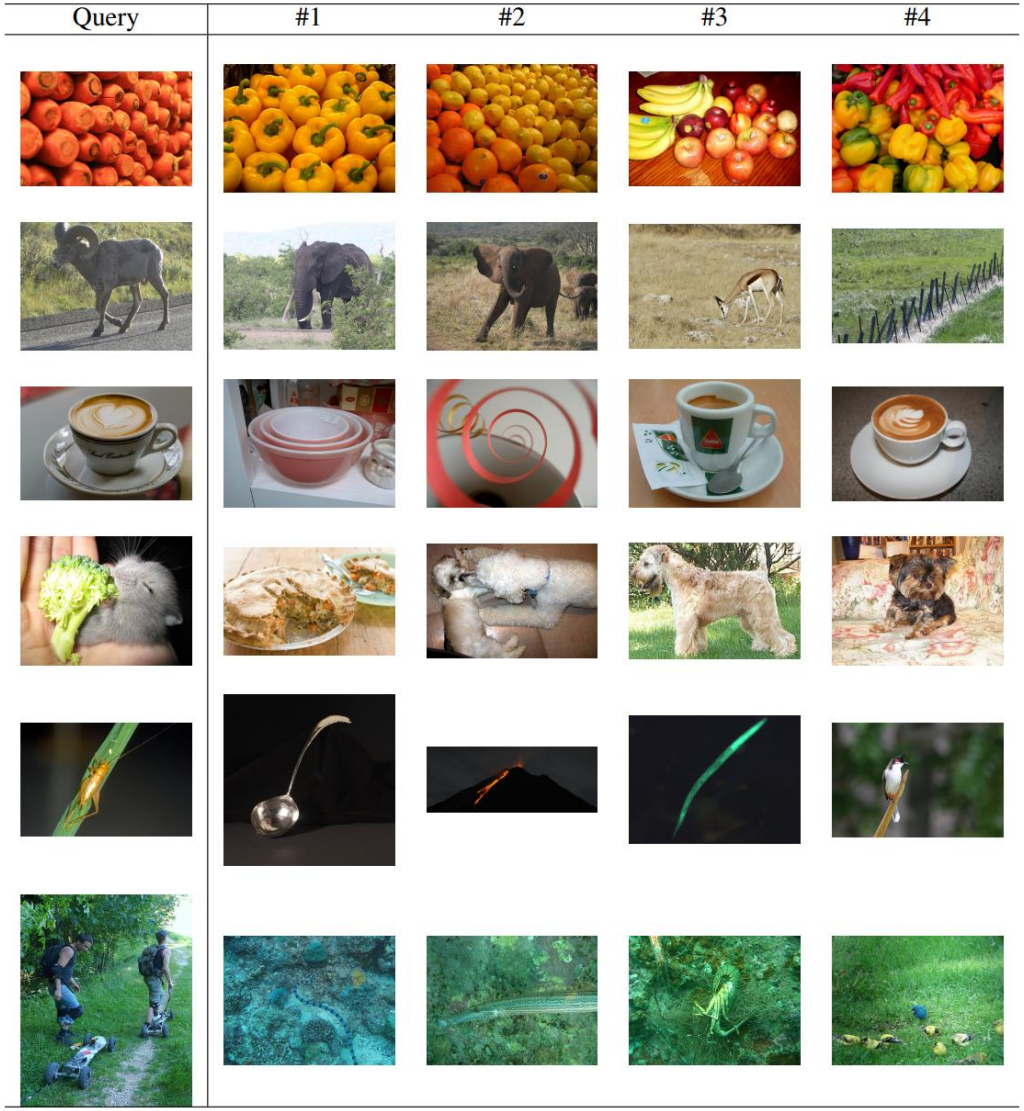


Figure 5: For the query images used in Krähenbühl et al. (2016) (left), nearest neighbors (by minimum cosine distance) from the ImageNet LSVRC (Russakovsky et al., 2015) training set in the *fc6* feature space of the ImageNet-trained BiGAN encoder E. (The *fc6* weights are set randomly; this space is a random projection of the learned *conv5* feature space.)

BigBiGANs

BiGANs trained using the BigGAN G and D architecture.

ResNet-style Encoders E

Figure 2: Selected reconstructions from an unsupervised BigBiGAN model (Section 3.3). Top row images are real data $\mathbf{x} \sim P_{\mathbf{x}}$; bottom row images are generated reconstructions of the above image $\mathbf{x}$ computed by $\mathcal{G}(\mathcal{E}(\mathbf{x}))$. Unlike most explicit reconstruction costs (e.g., pixel-wise), the reconstruction cost implicitly minimized by a (Big)BiGAN [7, 10] tends to emphasize more semantic, high-level details. Additional reconstructions are presented in Appendix B.

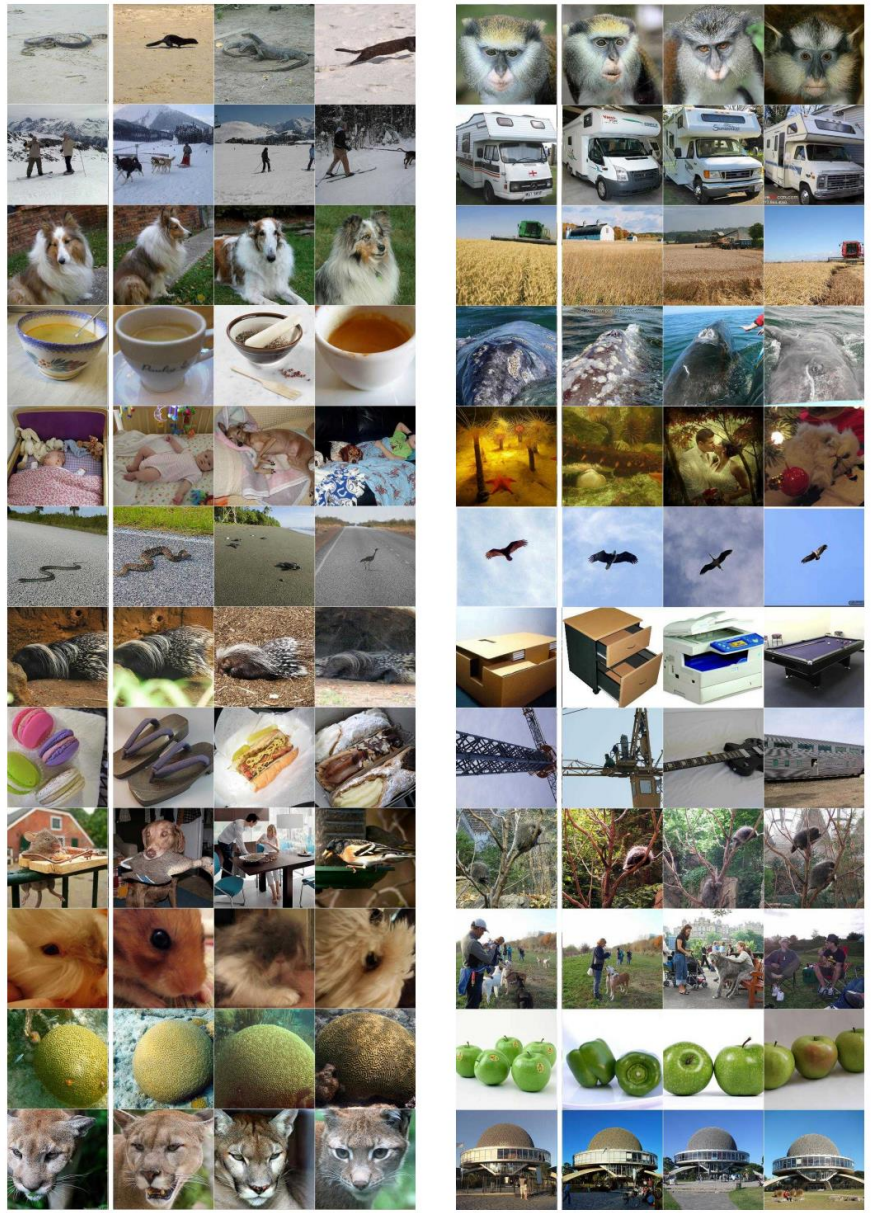


Figure 13: Nearest neighbors in BigBiGAN $\mathcal{E}$ feature space, from our best performing model (*RevNet* $\times 4$, $\uparrow \mathcal{E}$ *LR*). In each row, the first (left) column is a query image, and the remaining columns are its three nearest neighbors from the training set (the leftmost being the nearest, next being the second nearest, etc.). The query images above are the first 24 images in the ImageNet validation set.

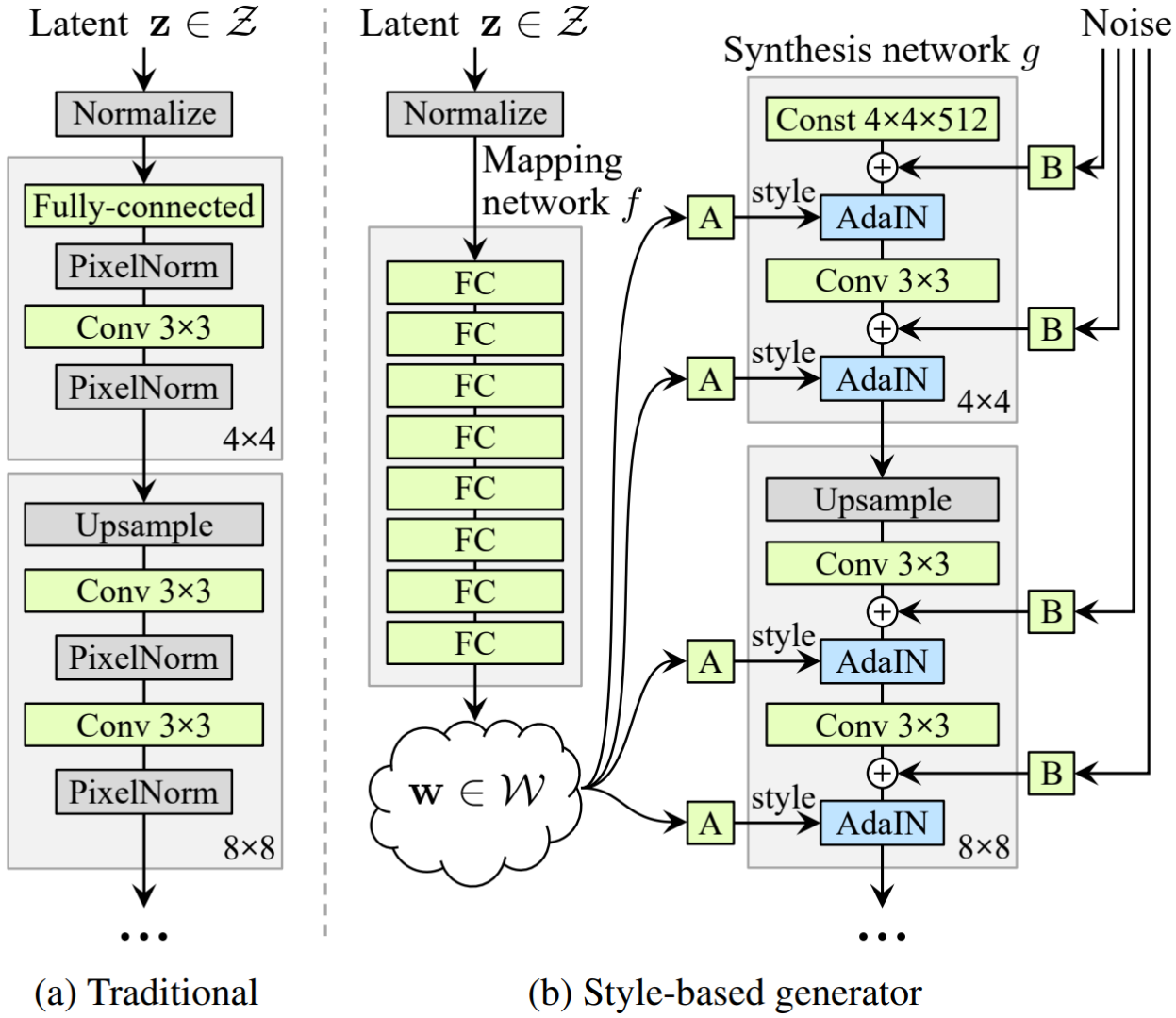
[Donahue & Simonyan, 2019] https://arxiv.org/abs/1907.02544v2

StyleGAN

- The **mapping network** and affine transformations can be viewed as a way to draw samples for each style from a learned distribution.
- The **synthesis network** can be viewed as a way to generate a novel image based on a collection of styles.
- The effects of each style are localized in the network, i.e., modifying a specific subset of the styles can be expected to affect only certain aspects of the image.

Adaptive Instance Normalization:

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



"A" stands for a learned affine transform, and "B" applies learned per-channel scaling factors to the noise input.

[Karras et al., 2019] https://arxiv.org/pdf/1812.04948.pdf

StyleGAN

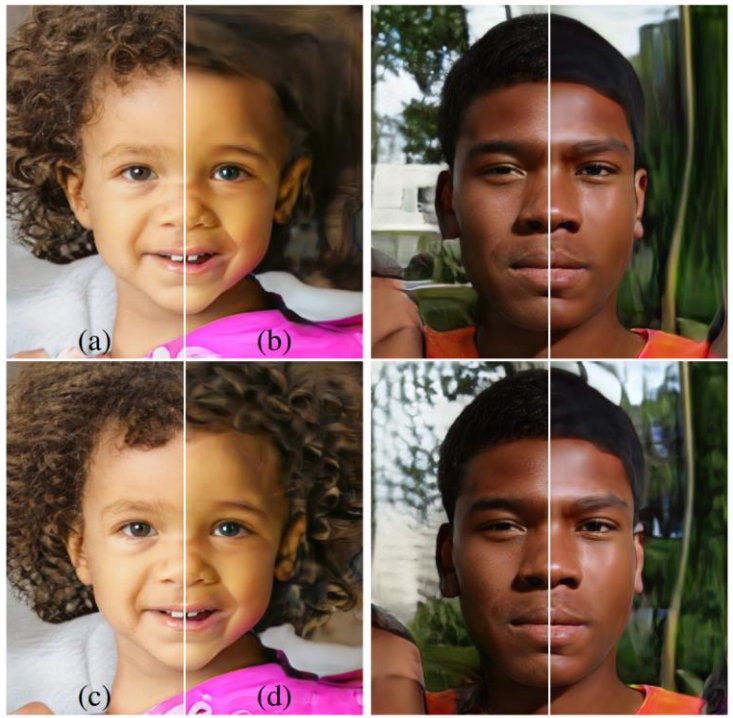


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. 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.

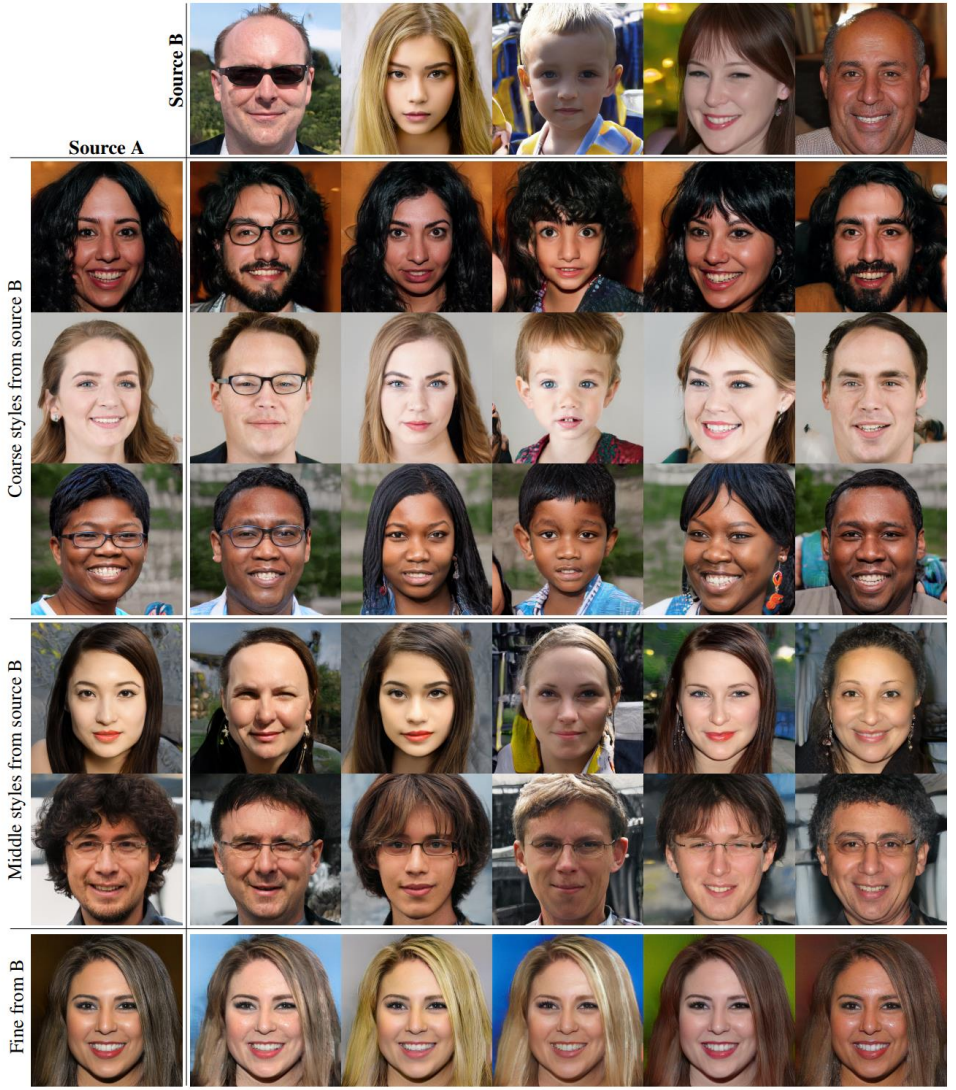


Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions (4^2 – 8^2) brings high-level aspects such as pose, general hair style, face shape, and eyeglasses from source B, while all colors (eyes, hair, lighting) and finer facial features resemble A. If we instead copy the styles of middle resolutions (16^2 – 32^2) from B, we inherit smaller scale facial features, hair style, eyes open/closed from B, while the pose, general face shape, and eyeglasses from A are preserved. Finally, copying the fine styles (64^2 – 1024^2) from B brings mainly the color scheme and microstructure.

[Karras et al., 2019] https://arxiv.org/pdf/1812.04948.pdf

https://github.com/NVlabs/stylegan

GANs for Image-to-Image Translation

CycleGAN

Unpaired Image-to-Image Translation

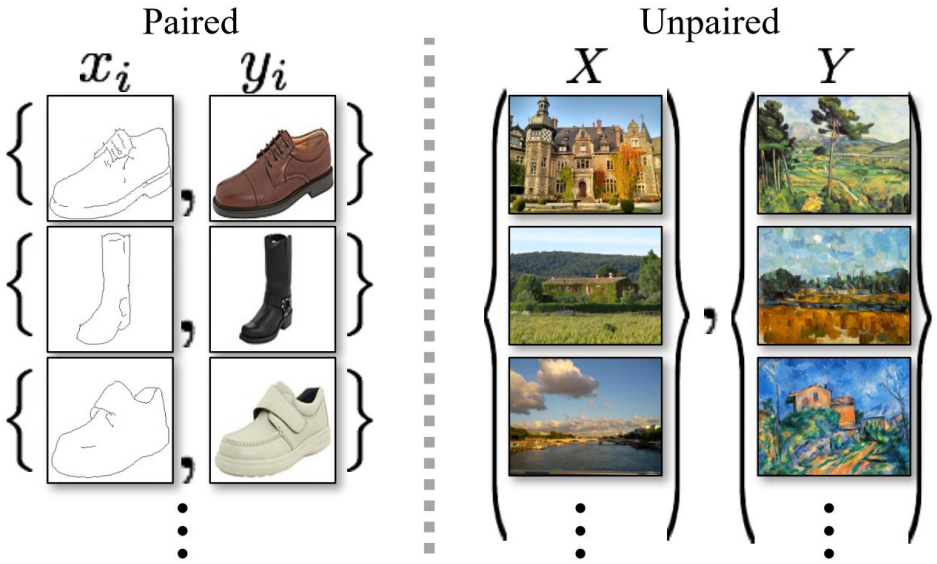


Figure 2: *Paired* training data (left) consists of training examples $\{x_i, y_i\}_{i=1}^N$, where the correspondence between x_i and y_i exists [22]. We instead consider *unpaired* training data (right), consisting of a source set $\{x_i\}_{i=1}^N$ ($x_i \in X$) and a target set $\{y_j\}_{j=1}^M$ ($y_j \in Y$), with no information provided as to which x_i matches which y_j.

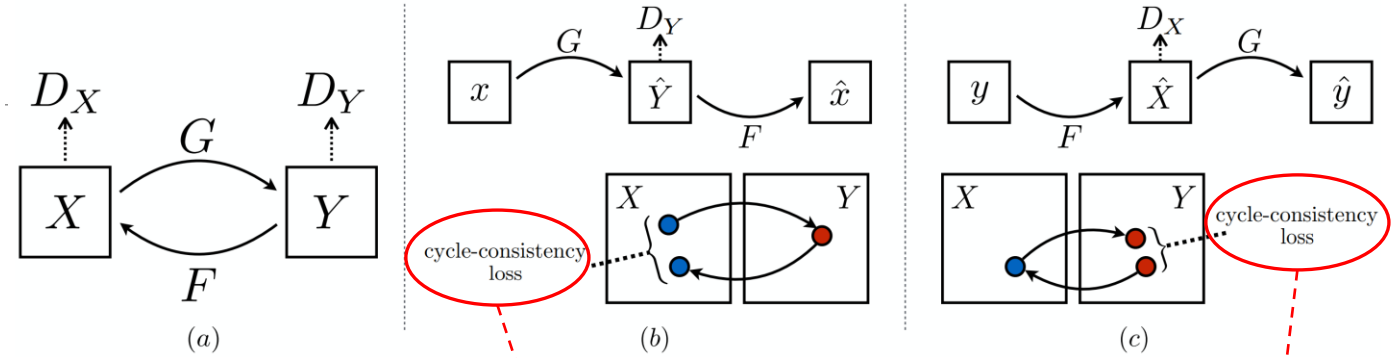


Figure 3: (a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators D_Y and D_X. D_Y encourages G to translate X into outputs indistinguishable from domain Y, and vice versa for D_X and F. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))]$$

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

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

$$G^*, F^* = \arg\min_{G,F} \max_{D_x, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

[Zhu et al., 2017] https://arxiv.org/pdf/1703.10593.pdf

https://junyanz.github.io/CycleGAN/

CycleGAN

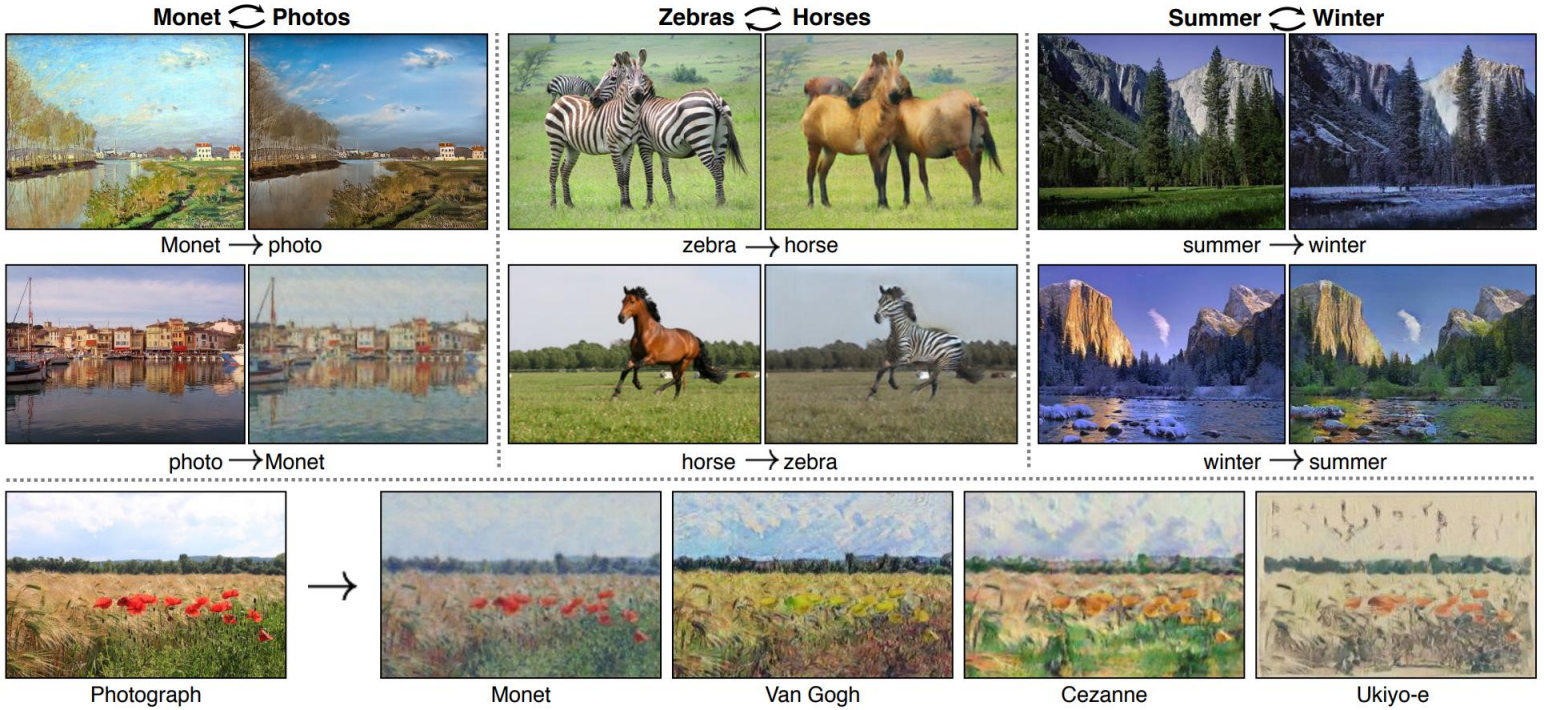


Figure 1: Given any two unordered image collections X and Y, our algorithm learns to automatically "translate" an image from one into the other and vice versa: *(left)* Monet paintings and landscape photos from Flickr; *(center)* zebras and horses from ImageNet; *(right)* summer and winter Yosemite photos from Flickr. Example application *(bottom)*: using a collection of paintings of famous artists, our method learns to render natural photographs into the respective styles.

[Zhu et al., 2017] https://arxiv.org/pdf/1703.10593.pdf

https://junyanz.github.io/CycleGAN/