Machine Learning

Lecture 12: Generative Adversarial Networks

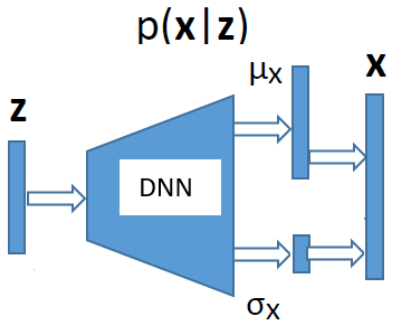
Nevin L. Zhang

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and references listed at the end.

Outline

Review of VAE

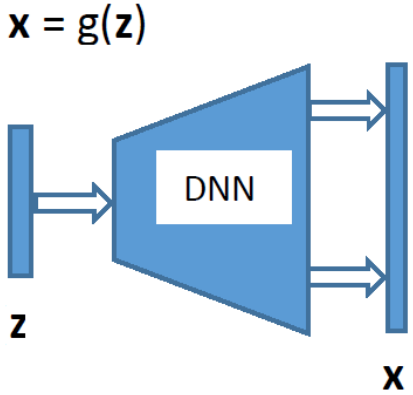


- The purpose of VAE is to learn a **decoder** which maps a latent vector $\mathbf{z}$ to a probability distribution $p(\mathbf{x}|\mathbf{z})$ over data space using a deep neural network.
- The decoder can be used to generate new samples, mostly images:

$$z \sim p(\mathbf{z}), \mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$$

- The decoder **explicitly** defines a distribution $p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$ over $\mathbf{x}$: The density $p(\mathbf{x})$ at a point $\mathbf{x}$ can be approximately computed.
- The encoder $q(\mathbf{z}|\mathbf{x})$ can be used to obtain a latent representation of data.

Generative Adversarial Networks (GAN)

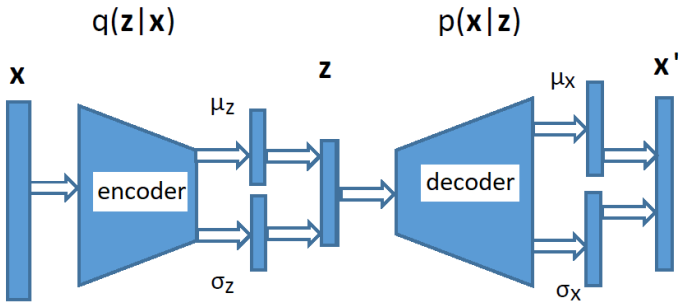


- The purpose of Generative Adversarial Networks (GAN) is to learn a **generator** that maps a a latent vector $\mathbf{z}$ to a vector $\mathbf{x} = g(\mathbf{z})$ in data space using a deep neural network.
- The generator can be used to generate new samples, mostly images:

$$z \sim p(\mathbf{z}), \mathbf{x} = g(\mathbf{z})$$

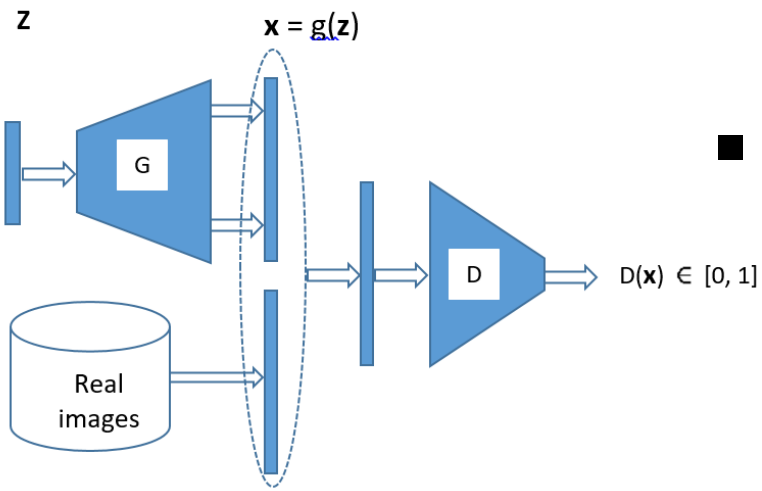
- The generator **implicitly** defines a distribution over $\mathbf{x}$. The density $p(\mathbf{x})$ at $p(\mathbf{x}) = p(g^{-1}(\mathbf{x}))$ at point $\mathbf{x}$ **cannot be computed**.
- GAN does not give a latent representation of data.

Review of VAE



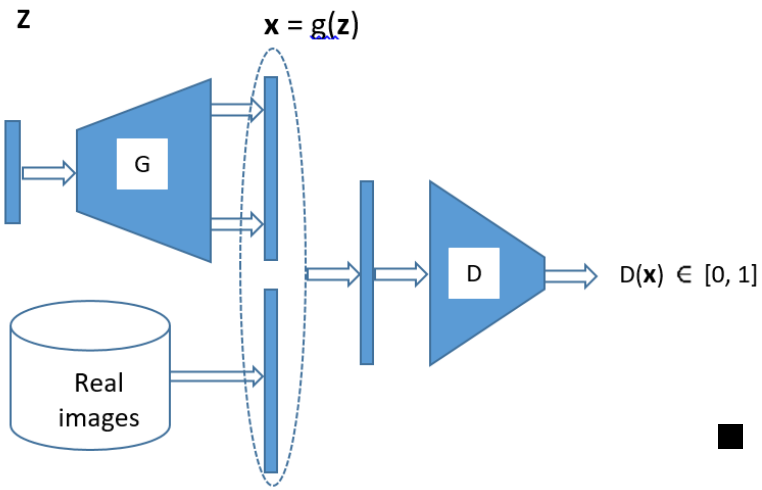
- VAE learns the parameters θ for the decoder by maximizing the empirical likelihood $\sum_{i=1}^{N} \log p_\theta(\mathbf{x}^{(i)})$,
- which asymptotically amounts to **minimizing the KL divergence** $KL(p_r||p_\theta)$ between the real data distribution p_r and the model distribution p_θ.
- The optimization problem is intractable. So, a **encoder** $q(\mathbf{z}|\mathbf{x})$ is used to obtain a variational lower bound of the empirical likelihood.
- In practice, VAE learns the parameters θ by maximizing the variational lower bound.

Generative Adversarial Networks (GAN)



- GAN learns the parameters θ for the generator by **minimizing the Jensen-Shannon divergence** $JS(p_r||p_\theta)$ between the real data distribution p_r and the model distribution p_θ.
- A **discriminator** D is used to approximate the intractable divergence $JS(p_r||p_\theta)$.
  - The discriminator is also a deep neural network. It maps a vector $\mathbf{x}$ of data space to a real number in [0, 1]. Its input can be either a **real example**, or a **fake example** generated by the generator.The output $D(\mathbf{x})$ is the probability that $\mathbf{x}$ being an real example.

Generative Adversarial Networks (GAN)



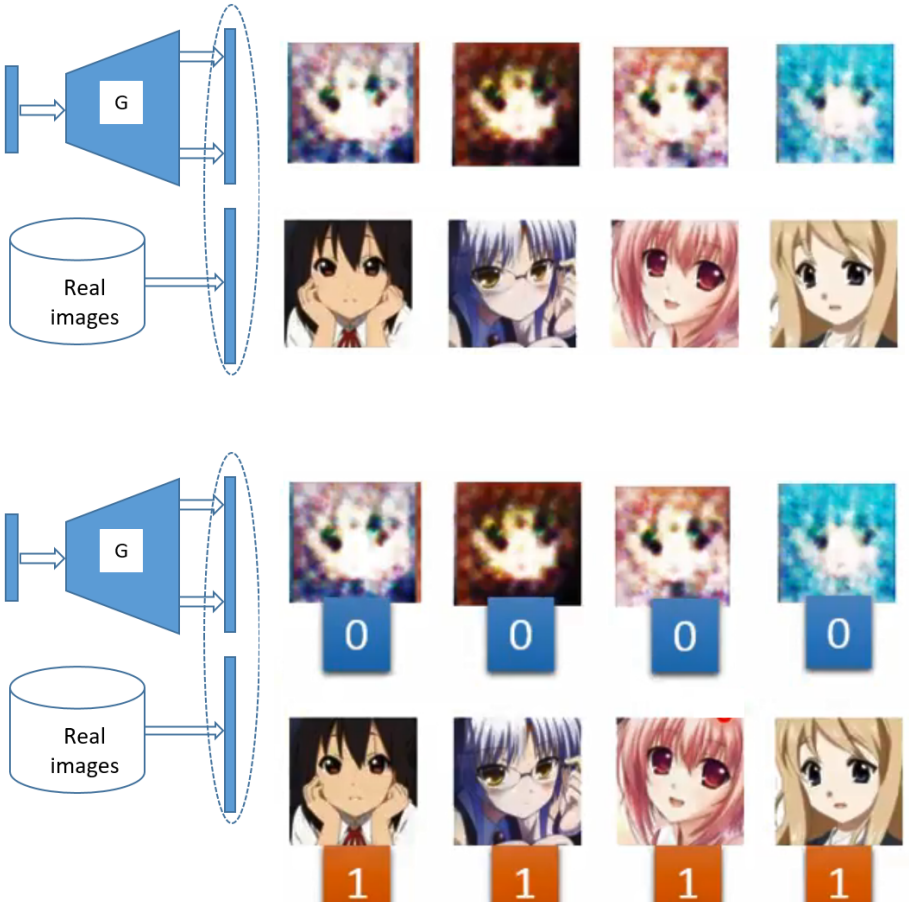
- The generator G and the discriminator D are trained alternately.
- When training D, the objective is to tell real and fake examples apart, i.e., to determine θ_d such that
  - $D(\mathbf{x})$ is 1 or close to 1 if $\mathbf{x}$ is a real example.
  - $D(\mathbf{x})$ is 0 or close to 0 if $\mathbf{x}$ is a fake example.
- When training G, the objective is to fool D, i.e, to generate fakes examples that D cannot tell from real examples.
- Notation: θ_g: parameters for the generator. θ_d: parameters for the discriminator.

Generative Adversarial Networks (GAN)

Each iteration of GAN learning proceeds as follows:

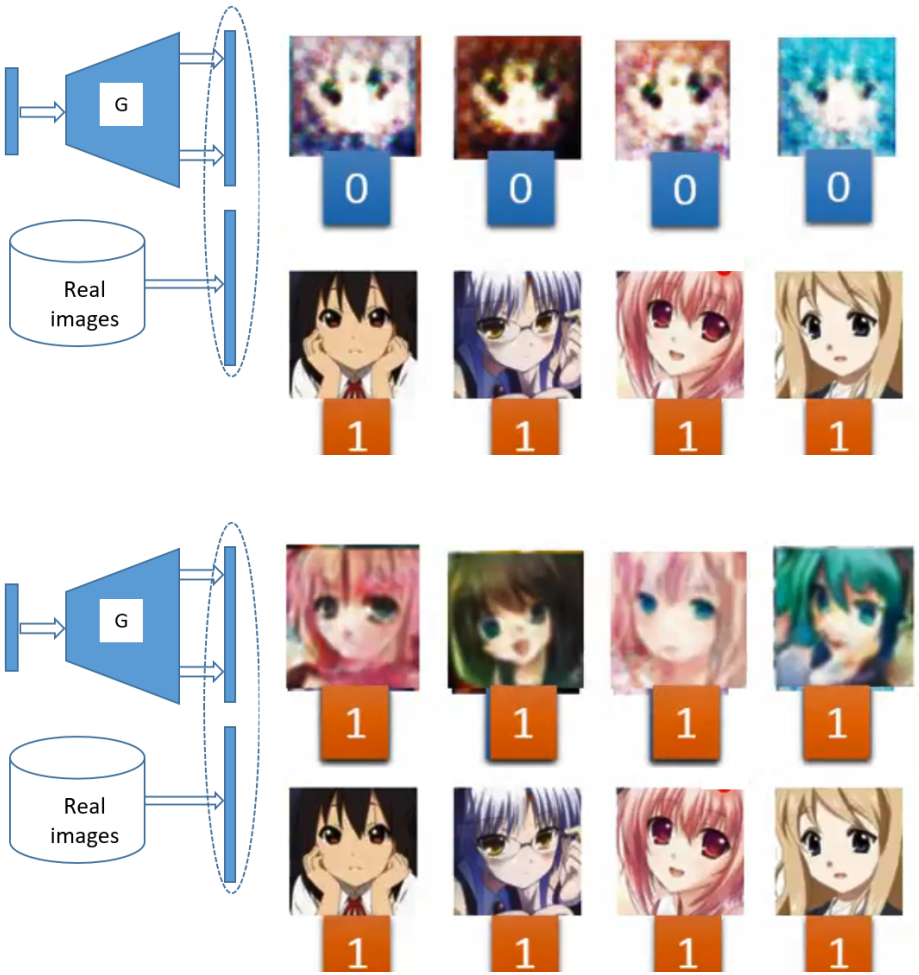
1. Improve the θ_d so that the discriminator becomes better at distinguishing between fake and real examples:Run the following k times:
   - Sample m **real examples** $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ from training data.
   - Generate m **fake examples** $g(\mathbf{z}^{(1)}), \ldots, g(\mathbf{z}^{(m)})$ using the current generator g, where $\mathbf{z}^{(i)} \sim p(\mathbf{z})$.
   - Improve θ_d so that the discriminator can better distinguish between **those** fake examples and **those** real examples.
2. Improve generator θ_g so as to generate examples the discriminator would find hard to tell fake or real:
   - Generate m **new fake examples** $g(\mathbf{z}^{(1)}), \ldots, g(\mathbf{z}^{(m)})$ using the current generator g. (Those are examples that the discriminator can tell as fake with high confidence because of the training in step 1.)
   - Improve θ_g to generate examples that would be more like real images than **those** fake images.

Illustration (Lee 2017)



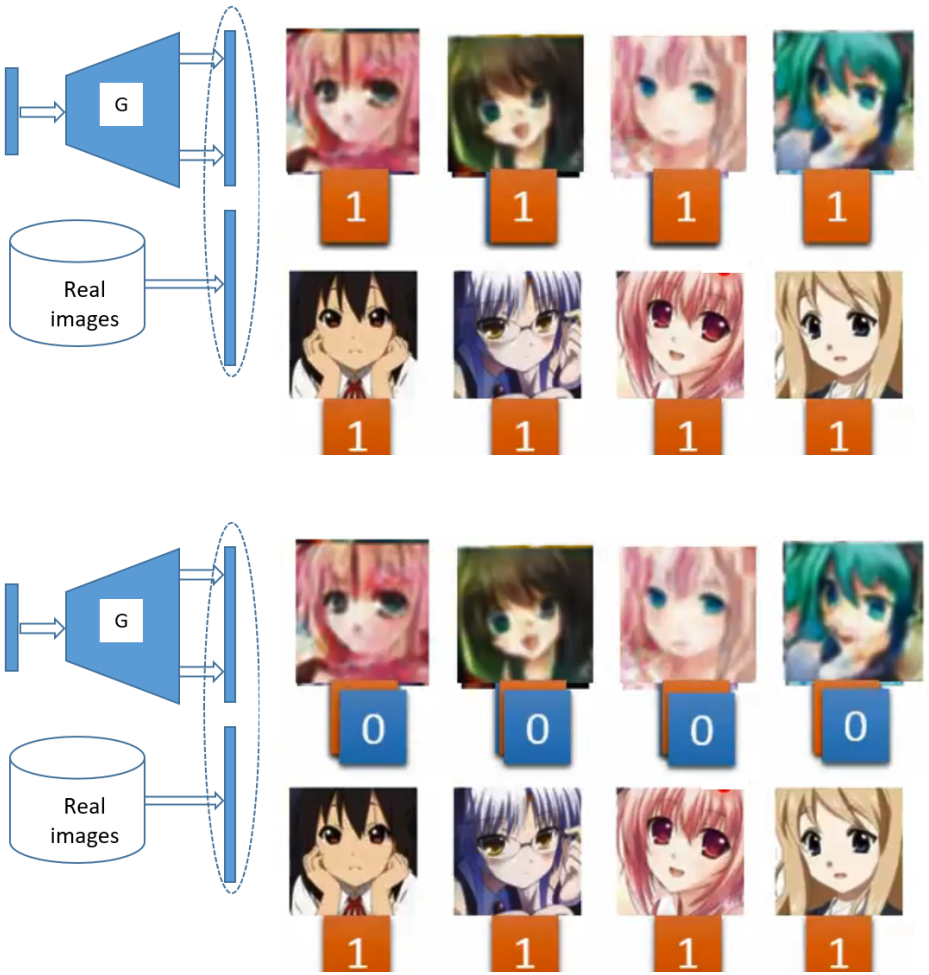
- At the beginning, the parameters of G are random. It generates poor images.
- GAN learns a discriminator D tell to real and fake images apart.

Example (Hung-Yi Lee 2017)



- Because the initial fake images are of poor quality, the discriminator learned from them (and real images) is rather weak.
- Then GAN improves G.
- The improved G can generate images that fool the initial weak D.

Illustration (Lee 2017)



- Then, D is told that the images on the first row are actually fake.
- It is therefore improved using this knowledge.
- The new version of D can now tell the better quality fake images from real images.
- Next, G will learn to improve further to fool this smarter D.

Cost Function for the Discriminator

- At each iteration, the discriminator is given the following data:
  - m **real examples** $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ from training data.
  - m **fake examples** $g(\mathbf{z}^{(1)}), \ldots, g(\mathbf{z}^{(m)})$ using the current generator g, where $\mathbf{z}^{(i)} \sim p(\mathbf{z})$.
- It needs to change its parameters θ_d so as to label all real examples with 1 and label all fake examples with 0.
- A natural cost function to use here is the **cross-entropy cost**

$$J(\theta_g, \theta_d) = -\frac{1}{2}\sum_{i=1}^{m} \log D(\mathbf{x}^{(i)}) - \frac{1}{2}\sum_{i=1}^{m} \log(1 - D(g(\mathbf{z}^{(i)}))$$

- The minimum value of J is 0. It is achieved when
  - All real examples are labeled with 1, i.e., $D(\mathbf{x}^{(i)}) = 1$ for all i, and
  - All fake examples are labeled with 0, i.e., $D(g(\mathbf{z}^{(i)}) = 0$ for all i.

Cost Function for the Discriminator

- So, the discriminator should determine θ_d by minimizing $J(\theta_g, \theta_d)$.
- This is the same as maximizing

$$V(\theta_g, \theta_d) = \sum_{i=1}^{m} \log D(\mathbf{x}^{(i)}) + \sum_{i=1}^{m} \log(1 - D(g(\mathbf{z}^{(i)})))$$

  which can be achieved by gradient ascent.

Cost Function for the Generator

- How should the generator determine its parameters θ_g?
  - The discriminator wants V to be as large as possible, because large V means it can tell the real and fake image apart with small error.
  - The generator wants to fool the discriminator. Hence, it wants V to as small as possible.
  - Note that the first term in V does not depend on θ_g
  - So, the generator should minimize

$$\sum_{i=1}^{m} \log(1 - D(g(\mathbf{z}^{(i)})))$$

- The GAN training algorithm is given on the next page.

The GAN training algorithm (Goodfellow et al 2014)

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

for number of training iterations **do**

  for k steps **do**

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

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

  end for

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

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

end for

The GAN training algorithm: Notes

- At each iteration, the discriminator is not trained to optimum. Instead, its parameters are improved only once by gradient ascent.
- Similarly, the parameters of the generators are also improved only once in each iteration by gradient descent.
- The reason for this will discussed in the next part.

- The cost function used in practice for the generator is actually the following:

$$\frac{1}{m}\sum_{i=1}^{m}[-\log D(g(\mathbf{z}^{(i)}))]$$

- The reason for this will also be discussed in the next part.

Empirical Results

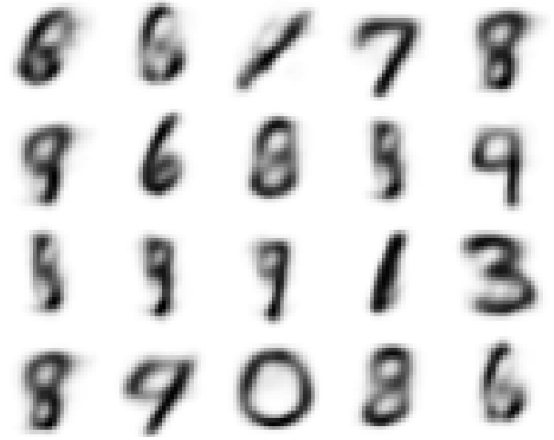
Goodfellow *et al.* (2014) use FNNs for both the generator and the discriminator:

- The generator nets used a mixture of rectifier linear activations and sigmoid activations.
- The discriminator net used maxout activations.

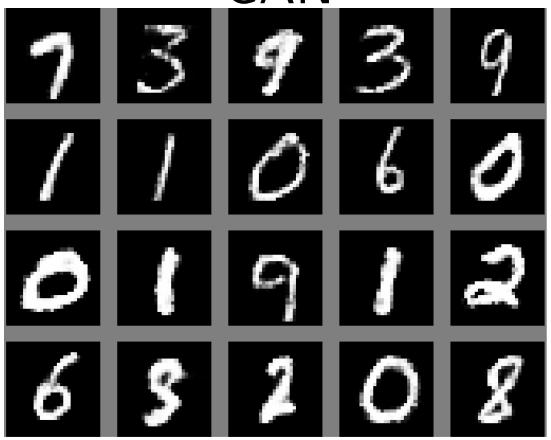
Empirical Results

- GAN generates sharper images than VAE.

VAE



GAN



Outline

GAN solves a minimax game

- The objective function $\frac{1}{m}\sum_{i=1}^{m}\log D(\mathbf{x}^{(i)}) + \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(g(\mathbf{z}^{(i)})))$ used in the GAN algorithm is an approximation of

$$\begin{aligned} V(\theta_g, \theta_d) &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\log(1 - D(g(\mathbf{z}))) \\ &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}\log(1 - D(\mathbf{x})) \end{aligned}$$

  We will use the latter in the analysis. Sometimes, we also write it as $V(G, D)$

- The discriminator wants to maximize V and the generator want to minimize V. So, we have a minmax game

$$\theta_g^* = arg\ min_{\theta_g} \max_{\theta_d} V(\theta_g, \theta_d)$$

Optimal Discriminator for Fixed Generator

- Assume that both G and D has sufficient capacity so that they can represent any function from $\mathbf{z}$ to $\mathbf{x}$ and any function from $\mathbf{x}$ to $[0, 1]$.
- **Theorem** (Goodfellow et al 2014): For fixed G, the optimal discriminator is

$$D^*(\mathbf{x}) = \frac{p_r(\mathbf{x})}{p_r(\mathbf{x}) + p_g(\mathbf{x})}$$

- **Proof**:

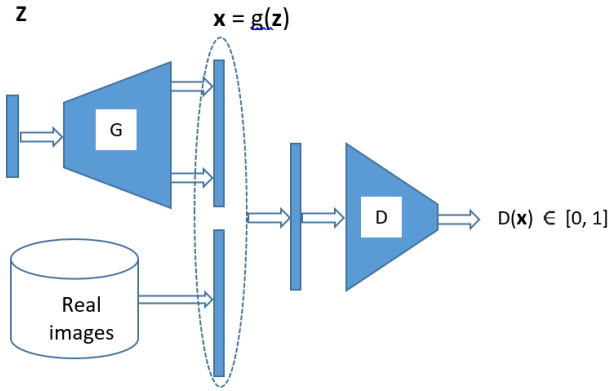
$$\begin{aligned} V(G, D) &= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x})) \\ &= \int [p_r(\mathbf{x}) \log D(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))]\, d\mathbf{x} \\ &\leq \int [p_r(\mathbf{x}) \log D^*(\mathbf{x}) + p_g(\mathbf{x}) \log(1 - D^*(\mathbf{x}))]\, d\mathbf{x} \end{aligned}$$

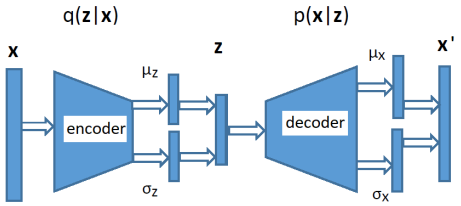
The last step follows by applying Gibbs' inequality to the integrand.

GAN and Jensen-Shannon Divergence

$$
\begin{aligned}
V(G) &= \max_D V(G,D) \\
&= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log \frac{p_r(\mathbf{x})}{p_r(\mathbf{x})+p_g(\mathbf{x})} + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - \frac{p_r(\mathbf{x})}{p_r(\mathbf{x})+p_g(\mathbf{x})}) \\
&= \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})} \log \frac{p_r(\mathbf{x})}{(p_r(\mathbf{x})+p_g(\mathbf{x}))/2} + \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(\frac{p_g(\mathbf{x})}{(p_r(\mathbf{x})+p_g(\mathbf{x}))/2}) - \log 4 \\
&= -\log 4 + 2JS(p_r||p_g)
\end{aligned}
$$

GAN and Jensen-Shannon Divergence





The above analysis leads to the following view on GAN:

- The objective of GAN is to learn the parameters of the generator so as to minimize the JS divergence $JS(p_r||p_g)$ between the real data distribution p_r and the generator distribution p_g.
- The introduction of a discriminator is to approximate the JS divergence.

Recall that the objective of VAE is to learn parameters of the decoder by minimizing the KL divergence $KL(p_r||p_g)$ between p_r and p_g (called the decoder distribution in VAE context). The introduction of an encoder $q(\mathbf{z}|\mathbf{x})$ is to provide an approximation (an upper bound) for the KL divergence

Why GAN generates more realistic images than VAE?

- VAE minimize the KL divergence

$$KL(p_r||p_g) = \int p_r(\mathbf{x}) \log \frac{p_r(\mathbf{x})}{p_g(\mathbf{x})} d\mathbf{x}$$

  - KL receives large contributions from areas of data space where $p_g(\mathbf{x})$ is small and $p_r(\mathbf{x})$ is large: High cost for not covering parts of data. Minimizing KL avoids such areas.
  - KL receives small contributions from areas of data space where $p_r(\mathbf{x})$ is small and $p_g(\mathbf{x})$ is large: Low cost for generating fake looking images. Minimizing KL does not avoid such areas. This is why VAE generates blurry images.

- If $KL(p_g||p_r)$ is used instead, the story will be reversed. The generator distribution might not covering parts of the data points, and hence will not generate images similar to those in that areas. This is called mode dropping

Why GAN generates more realistic images than VAE?

- GAN minimize the JS divergence

$$JS(p_r||p_g) = \frac{1}{2}KL(p_r||p_a) + \frac{1}{2}KL(p_g||p_a)$$

  where $p_a = (p_r + p_g)/2$.

- $JS(p_r||p_g)$ is a middle ground between $KL(p_r||p_g)$ and $KL(p_q||p_r)$. Hence, it gives high cost for generating fake looking images.
- This is one reason why GAN generates more realistically looking images than VAE.
- However, GAN suffers from mode dropping.
  - Discriminator forces fake images to look real, but does not enforces coverage.

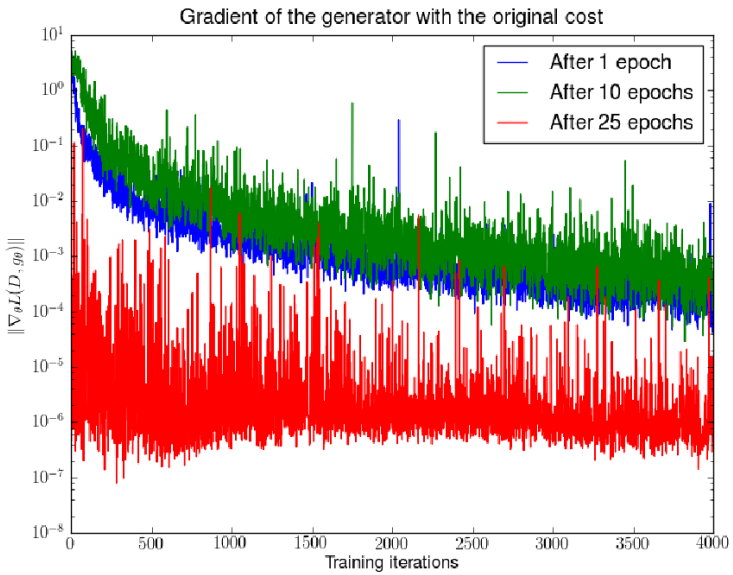
Two Cost Functions for Generator

- The original cost function for the generator is: $\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x}))$, i.e, expected log-probability that the discriminator being correct.
- In practice, an alternative cost function is used: $-\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[\log D(\mathbf{x})]$, i.e, negation of expected log-probability that the discriminator being mistaken.
- This is why:

$$\begin{aligned}\nabla_{\theta_g} \mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})} \log(1 - D(\mathbf{x})) &= \nabla_{\theta_g} \int p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) d\mathbf{x} \\ &= \int \log(1 - D(\mathbf{x})) \nabla_{\theta_g} p_g(\mathbf{x}) d\mathbf{x}\end{aligned}$$

  - At the beginning, the generator is poor. The discriminator can easily tell fakes images from real ones, i.e. $D(\mathbf{x}) = 0$ for all $\mathbf{x} \sim p_g(\mathbf{x})$.
  - Hence, the gradient for generator is 0. The generator does not get information on how to improve its parameters at a time it needs such information the most.

Gradient of Original Generator Cost Function



- How the gradient of the original cost function change as we train the discriminator to optimum.
- Arjovsky et al. ICLR 2017: " First, we trained a DCGAN for 1, 10 and 25 epochs. Then, with the generator fixed we train a discriminator from scratch and measure the gradients with the original cost function. We see the gradient norms decay quickly, in the best case 5 orders of magnitude after 4000 discriminator iterations. Note the logarithmic scale "

Two Cost Functions for Generator

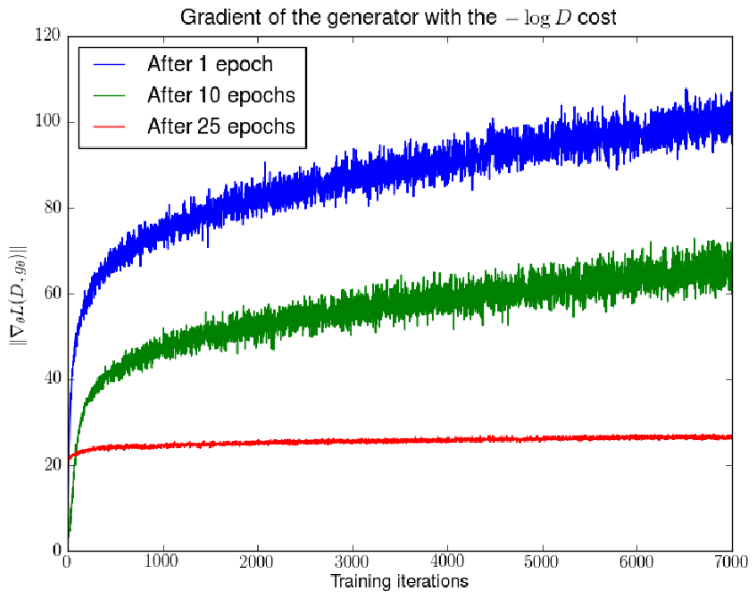
- The vanishing gradient problem is avoided if we use the alternative cost function $\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D(\mathbf{x})]$ because

$$\begin{aligned}\nabla_{\theta_g}\mathbb{E}_{\mathbf{x}\sim p_g(\mathbf{x})}[-\log D(\mathbf{x})] &= \nabla_{\theta_g}\int p_g(\mathbf{x})[-\log D(\mathbf{x})]d\mathbf{x}\\ &= \int[-\log D(\mathbf{x})]\nabla_{\theta_g}p_g(\mathbf{x})d\mathbf{x}\end{aligned}$$

  The gradient for the generator is no longer zero, because $D(\mathbf{x})$ is not close to 1 for $\mathbf{x}\sim p_g(\mathbf{x})$.

- However, when the discriminator become good ($D(\mathbf{x})\approx 0$ for $\mathbf{x}\sim p_g(\mathbf{x})$), the gradient become very large (note $\log 0 = -\infty$), making the training unstable.

Gradient of the Alternative Generator Cost Function



- How the gradient of the alternative cost function change as we train the discriminator to optimum.
- With a poor generator (the one after only one epoch of training), the gradient of the alternative cost function goes to infinite.
- The general trend is the same with a better generator (the ones after 10, or 25 epochs of training), except now the gradient increase at a slower pace.

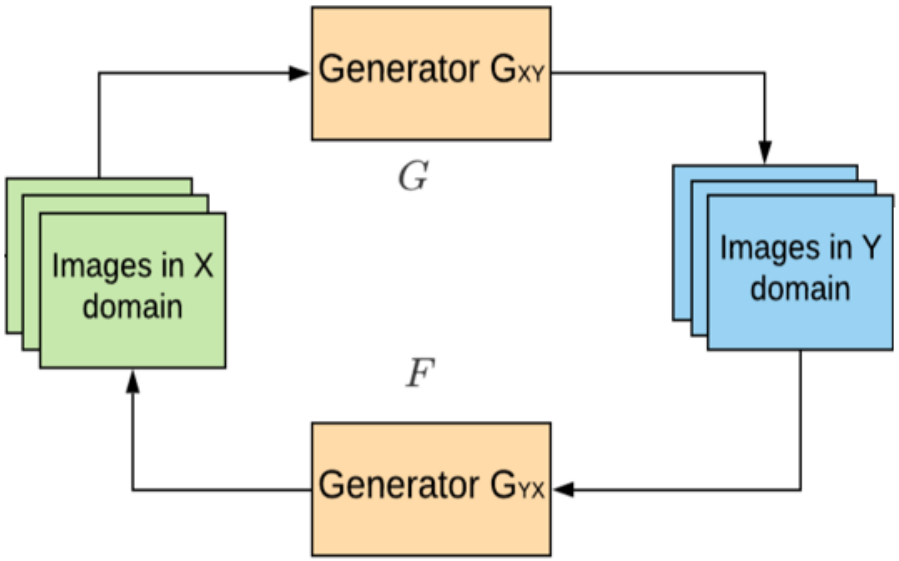
Difficulties with GAN

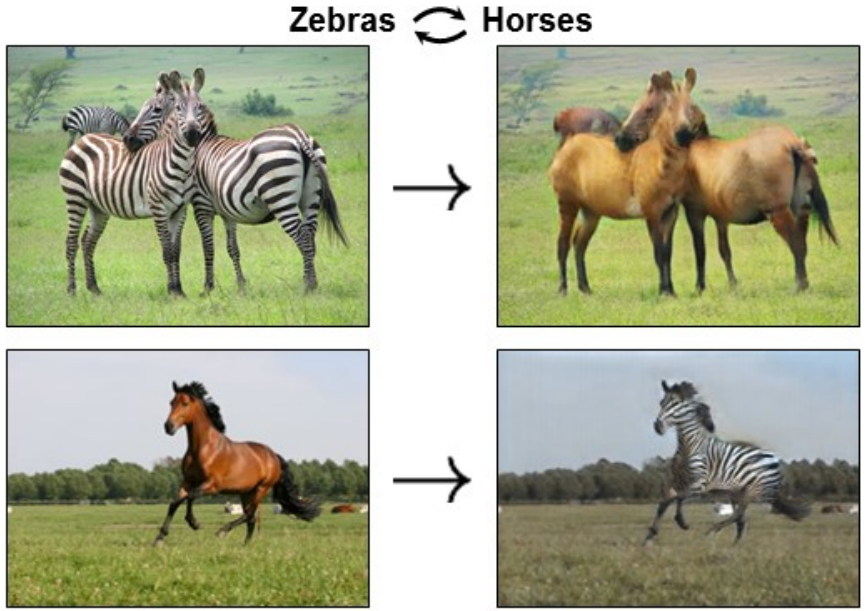
- The original generator cost function leads to slow training, and the alternative cost function leads to unstable training.
- In practice, those difficulties are dealt with by not training the discriminator to optimum, not even close to optimum.

Outline

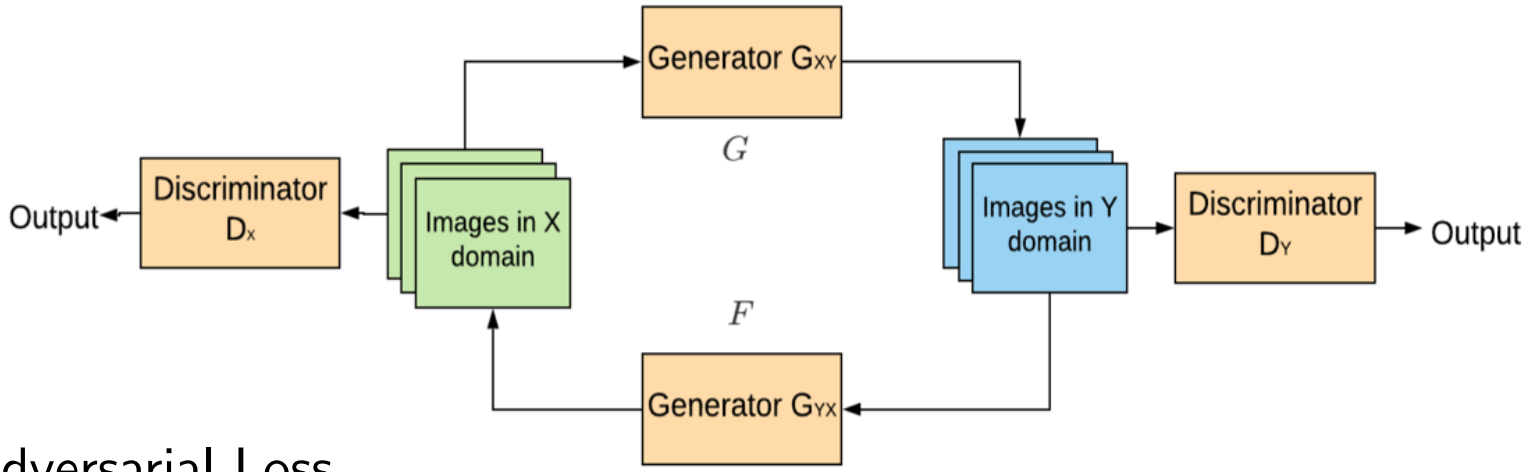
CycleGAN (Zhu *et al.* 2017)

- Given: X, Y — Two collections of images from different domains.
- To lean: G: $X \rightarrow Y$; F: $Y \rightarrow X$.





CycleGAN (Zhu *et al.* 2017)



- Adversarial Loss

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

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

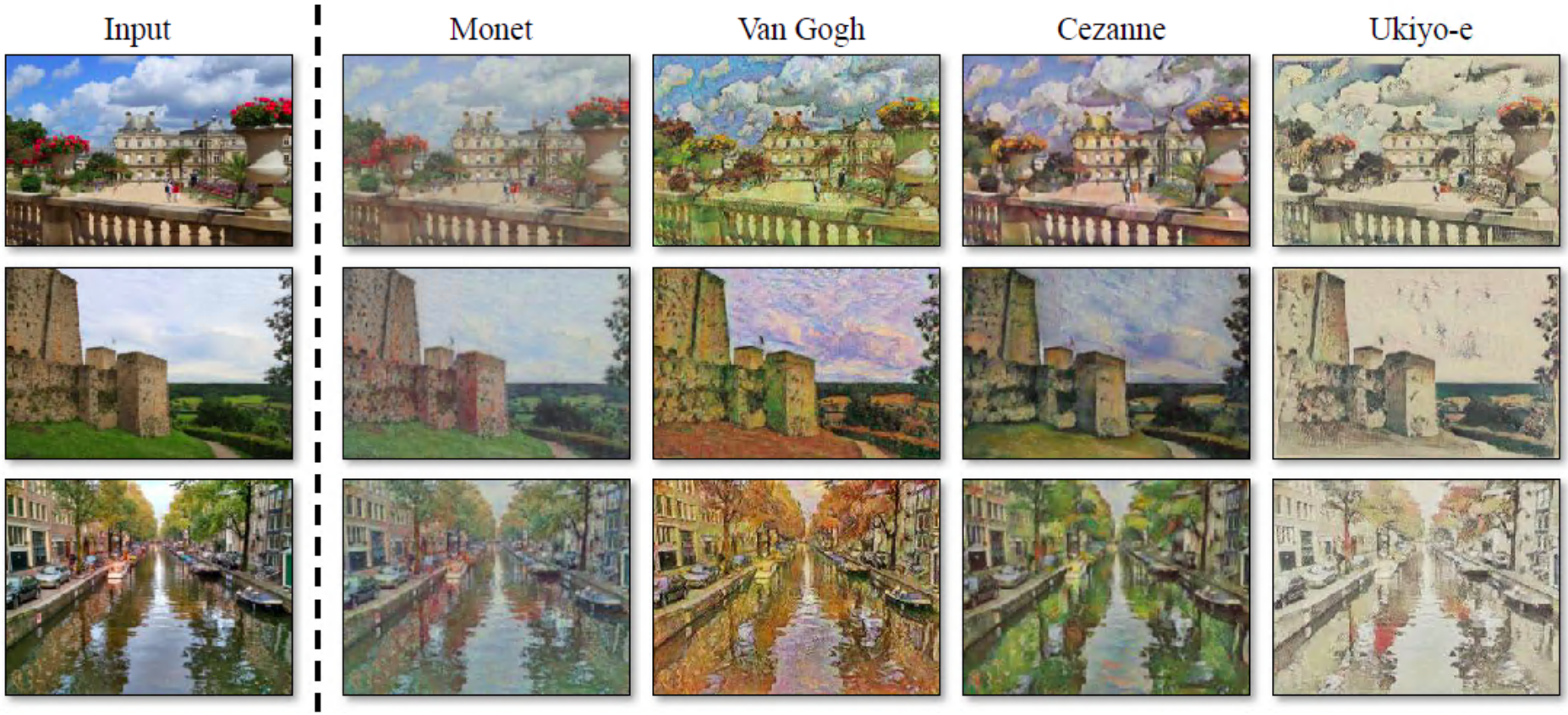
- Cycle Consistent Loss

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

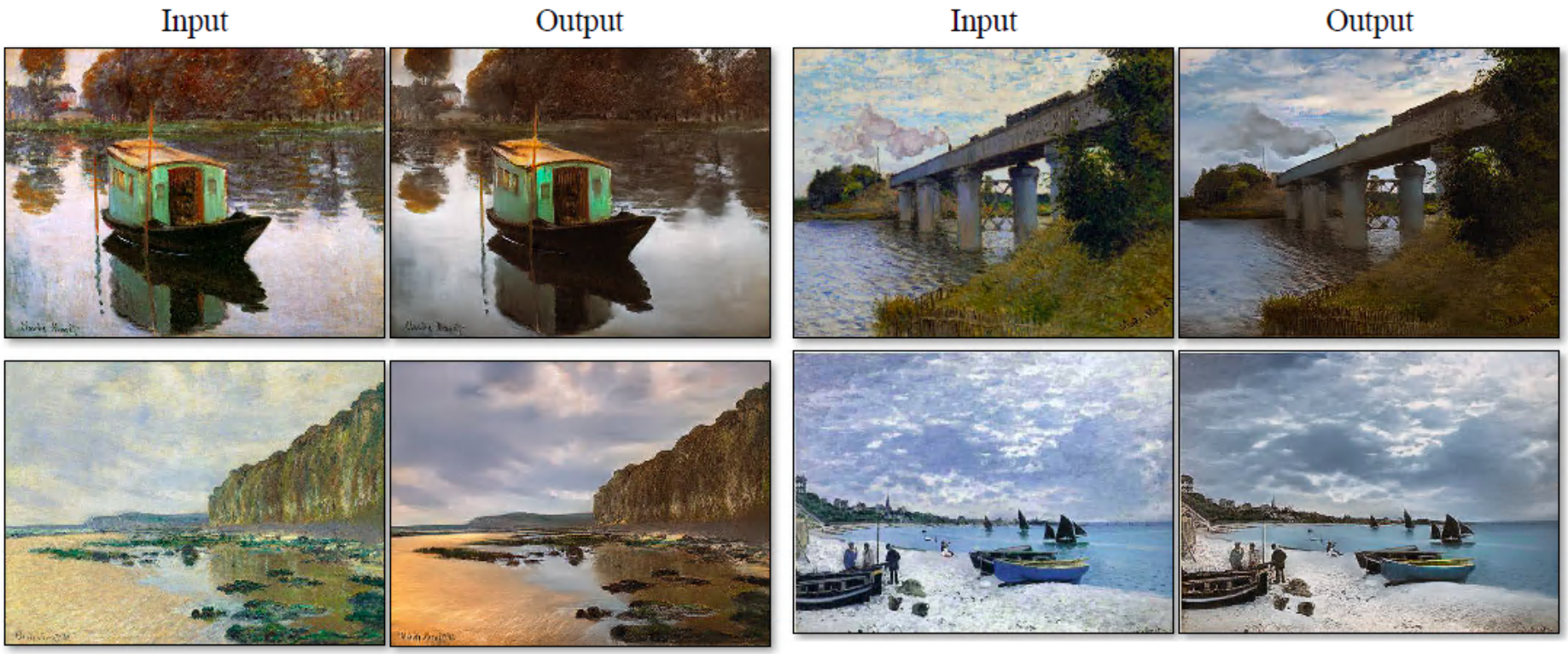
- Total Loss

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

CycleGAN Results

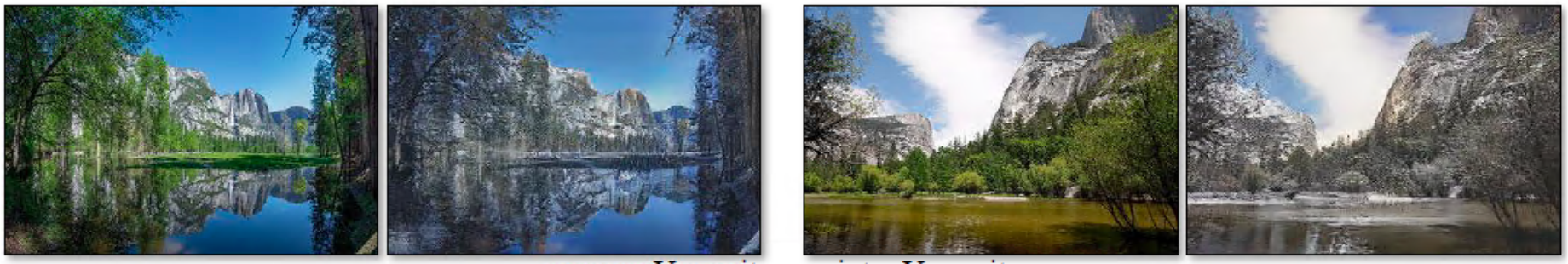


CycleGAN Results



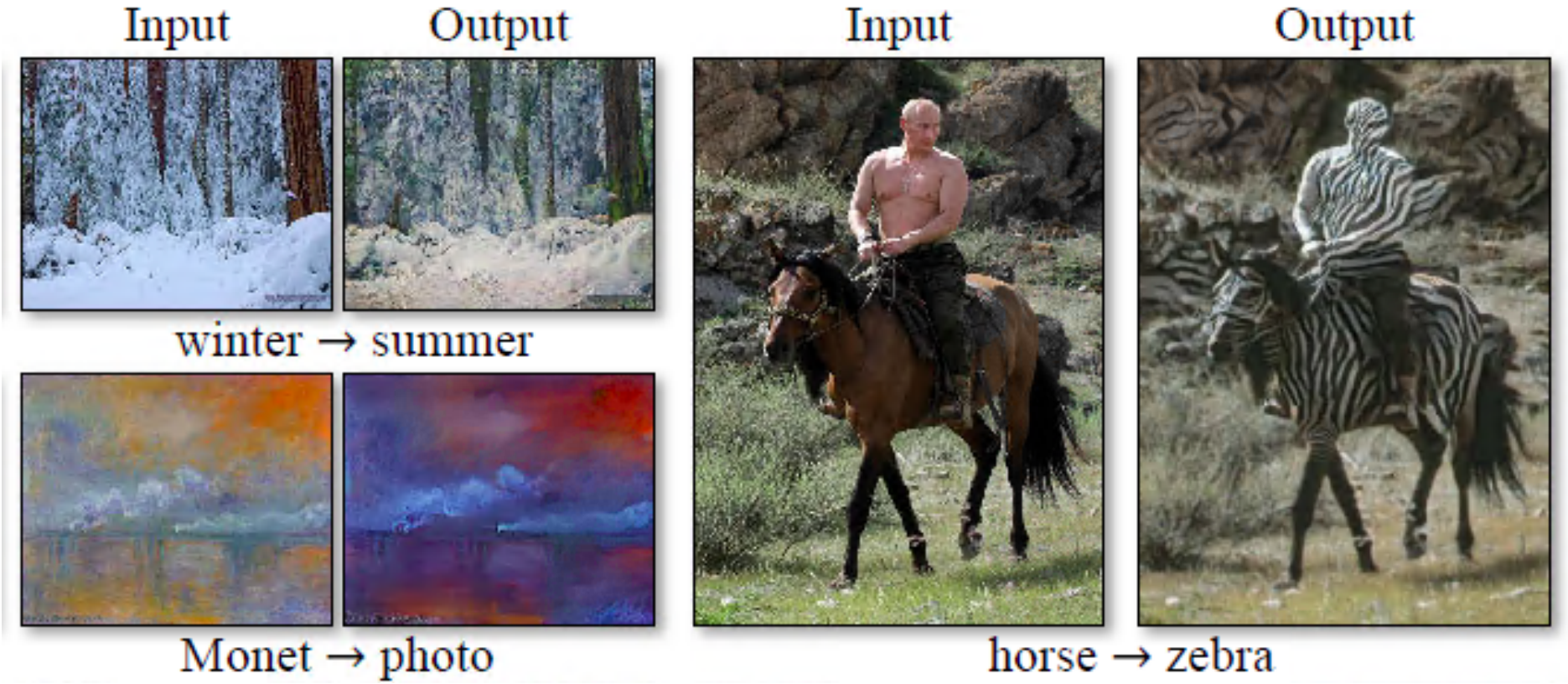
CycleGAN Results

winter Yosemite → summer Yosemite



summer Yosemite → winter Yosemite

CycleGAN Results: Failures



References

- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. arXiv preprint arXiv:1701.07875.
- Arjovsky, M., & Bottou, L. (2017). Towards principled methods for training generative adversarial networks. ICLR 2017.
- Choi, Yunjey, et al. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
- Choi, Yunjey, et al. "Stargan v2: Diverse image synthesis for multiple domains." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.
- Goodfellow, Ian, *et al.* "Generative adversarial nets." Advances in neural information processing systems. 2014.
- Goodfellow, Ian, *et al.* Generative adversarial networks, NIPS 2016 Tutorial.
- Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE International Conference on Computer Vision. 2017.
- Gui, Jie, et al. "A review on generative adversarial networks: Algorithms, theory, and applications." arXiv preprint arXiv:2001.06937 (2020).
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. In Advances in neural information processing systems (pp. 5767-5777).

References

- Vincent Herrmann. Wasserstein GAN and the Kantorovich-Rubinstein Duality. `https://vincentherrmann.github.io/blog/wasserstein/`
- Karras, Tero, *et al*. "Progressive growing of gans for improved quality, stability, and variation." ICLR 2018.
- Karras, T., Laine, S., & Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4401-4410).
- Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of stylegan. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 8110-8119).
- Hung-Yi Lee (2017). Improving GAN. `https://www.youtube.com/watch?v=KSN4QYgAtao`
- Radford, Alec, *et al.* "Unsupervised representation learning with deep convolutional generative adversarial networks." ICLR 2016.
- Cedric Villani (2009). Optimal Transport: Old and New. Grundlehren der mathematischen Wissenschaften. Springer, Berlin.
- Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." Proceedings of the IEEE international conference on computer vision. 2017.