Machine Learning

Lecture 11: Generative Adversarial Networks (I)

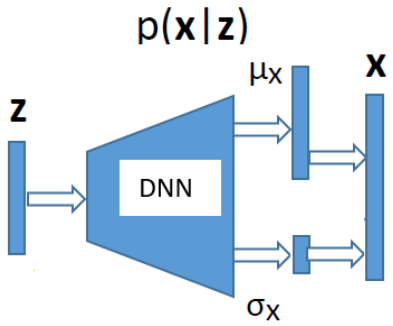
Nevin L. Zhang
lzhang@cse.ust.hk

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

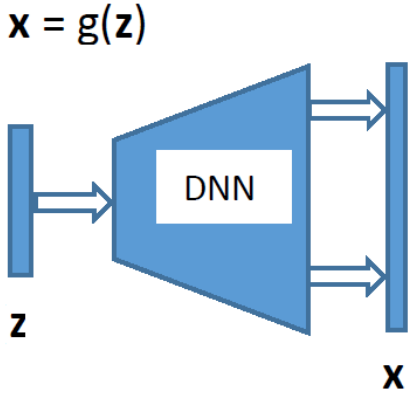


- The purpose of VAE is to learn a **decoder** which maps a latent vector $\mathbf{z}$ to a probability distribution $p(\mathbf{x}|\mathbf{z})$ over data space using a deep neural network.
- The decoder can be used to generate new samples, mostly images:

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

- 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)

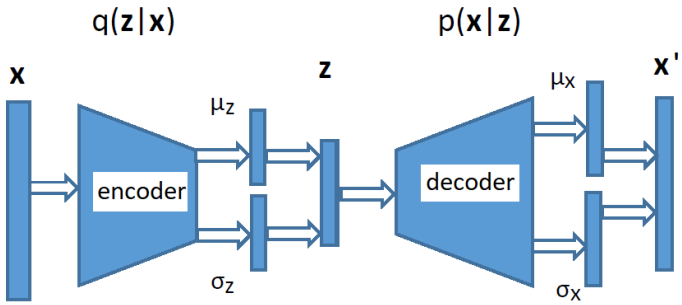


- The purpose of Generative Adversarial Networks (GAN) is to learn a **generator** that maps a a latent vector $\mathbf{z}$ to a vector $\mathbf{x} = g(\mathbf{z})$ in data space using a deep neural network.
- The generator can be used to generate new samples, mostly images:

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

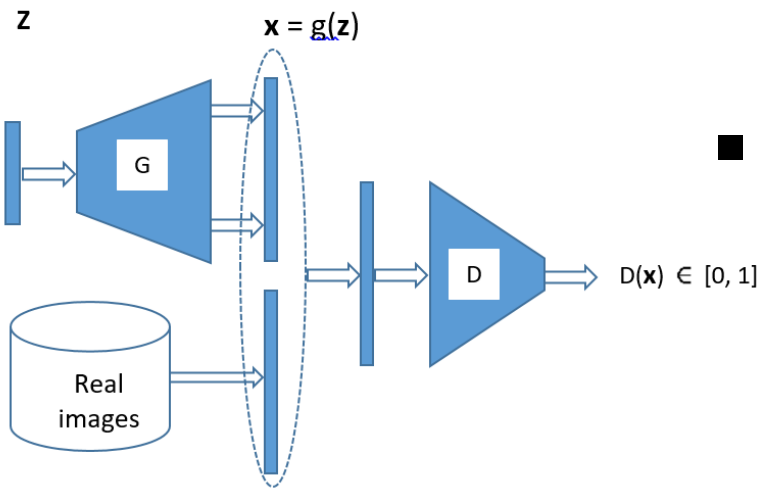
- The generator **implicitly** defines a distribution over $\mathbf{x}$. The density $p(\mathbf{x})$ at $p(\mathbf{x}) = p(g^{-1}(\mathbf{x}))$ at point $\mathbf{x}$ **cannot be computed**.
- GAN does not give a latent representation of data.

Review of VAE



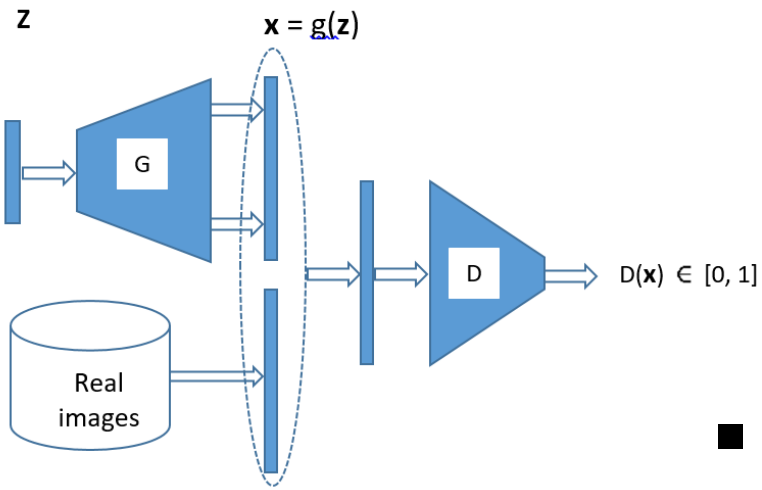
- VAE learns the parameters θ for the decoder by maximizing the empirical likelihood $\sum_{i=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$,
- which asymptotically amounts to **minimizing the KL divergence** $KL(p_r||p_{\boldsymbol{\theta}})$ between the real data distribution p_r and the model distribution $p_{\boldsymbol{\theta}}$.
- 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)



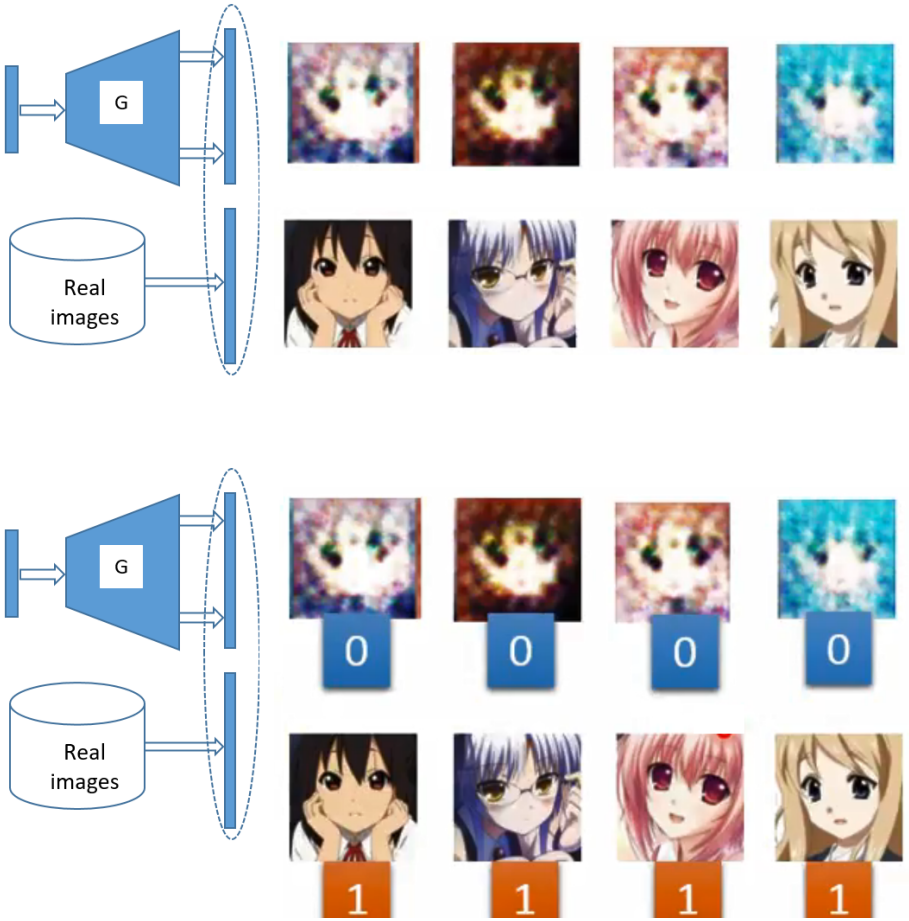
- The generator G and the discriminator D are trained alternately.
- When training D, the objective is to tell real and fake examples apart, i.e., to determine $\boldsymbol{\theta}_d$ such that
  - $D(\mathbf{x})$ is 1 or close to 1 if $\mathbf{x}$ is a real example.
  - $D(\mathbf{x})$ is 0 or close 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: $\boldsymbol{\theta}_g$: parameters for the generator. $\boldsymbol{\theta}_d$: parameters for the discriminator.

Generative Adversarial Networks (GAN)

Each iteration of GAN learning proceeds as follows:

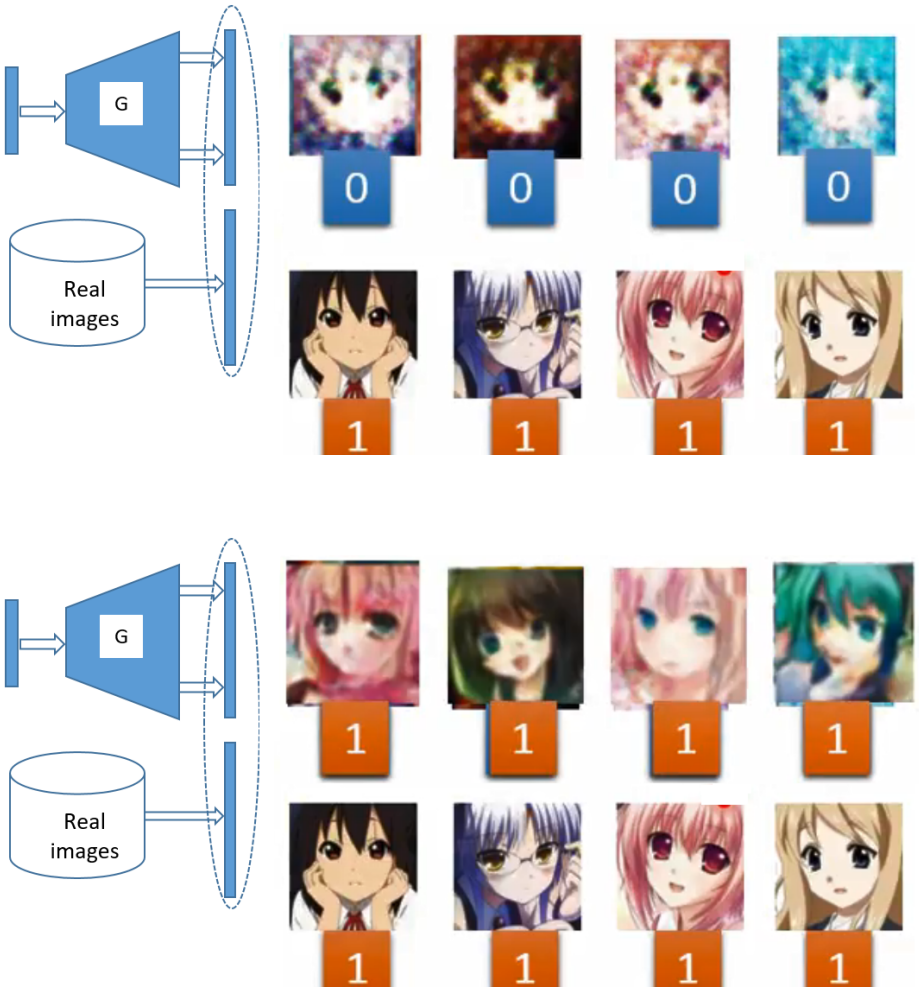
1. Improve the $\boldsymbol{\theta}_d$ so that the discriminator becomes better at distinguishing between fake and real examples:Run the following k times:
   - Sample m **real examples** $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ from training data.
   - Generate m **fake examples** $g(\mathbf{z}^{(1)}), \ldots, g(\mathbf{z}^{(m)})$ using the current generator g, where $\mathbf{z}^{(i)} \sim p(\mathbf{z})$.
   - Improve $\boldsymbol{\theta}_d$ so that the discriminator can better distinguish between **those** fake examples and **those** real examples.
2. Improve generator $\boldsymbol{\theta}_g$ 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 $\boldsymbol{\theta}_g$ to generate examples that would be more like real images than **those** fake images.

Illustration (Lee 2017)



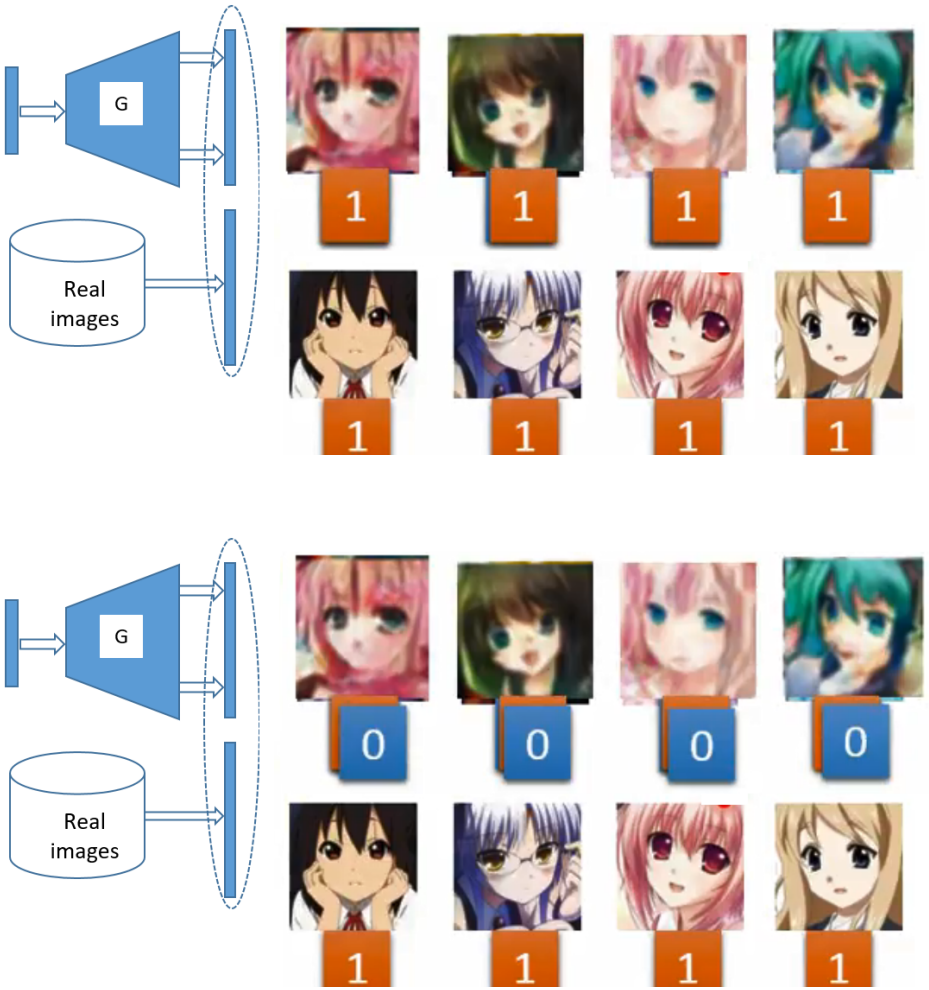
- At the beginning, the parameters of G are random. It generates poor images.
- GAN learns a discriminator D tell to real and fake images apart.

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 $\boldsymbol{\theta}_d$ 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**

$$J(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = -\frac{1}{2}\sum_{i=1}^{m} \log D(\mathbf{x}^{(i)}) - \frac{1}{2}\sum_{i=1}^{m} \log(1 - D(g(\mathbf{z}^{(i)}))$$

- 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 $\boldsymbol{\theta}_d$ by minimizing $J(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d)$.
- This is the same as maximizing

$$V(\boldsymbol{\theta}_g, \boldsymbol{\theta}_d) = \sum_{i=1}^{m} \log D(\mathbf{x}^{(i)}) + \sum_{i=1}^{m} \log(1 - D(g(\mathbf{z}^{(i)})))$$

  which can be achieved by gradient ascent.

Cost Function for the Generator

- How should the generator determine its parameters $\boldsymbol{\theta}_g$?
  - 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 $\boldsymbol{\theta}_g$
  - 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

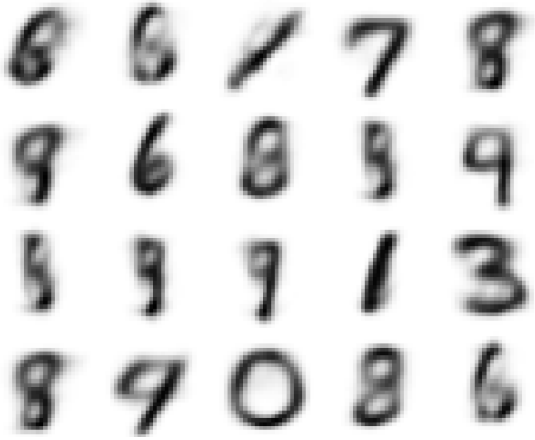
Goodfellow *et al.* (2014) use FNNs for both the generator and the discriminator:

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

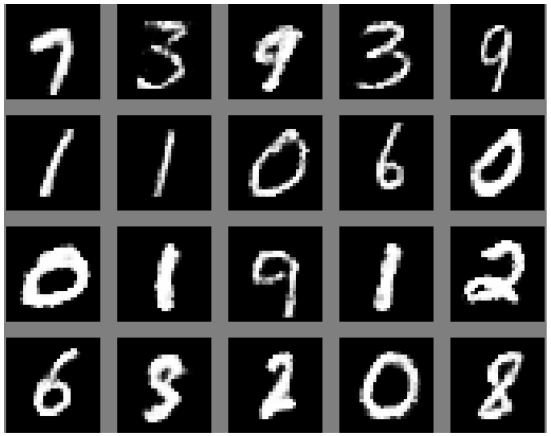
Empirical Results

- GAN generates sharper images than VAE.

VAE



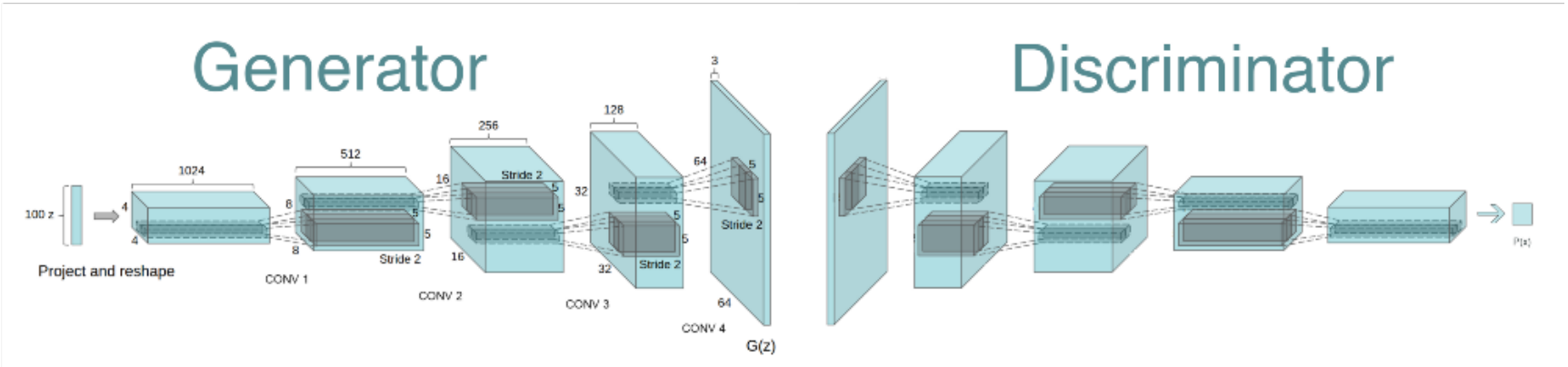
GAN



Outline

DCGANs (Radford *et al.* 2016)

More stable architecture for training GANs. The generator (given below) and the discriminator are symmetric.



Most current GANs are at least loosely based on the DCGANs architecture.

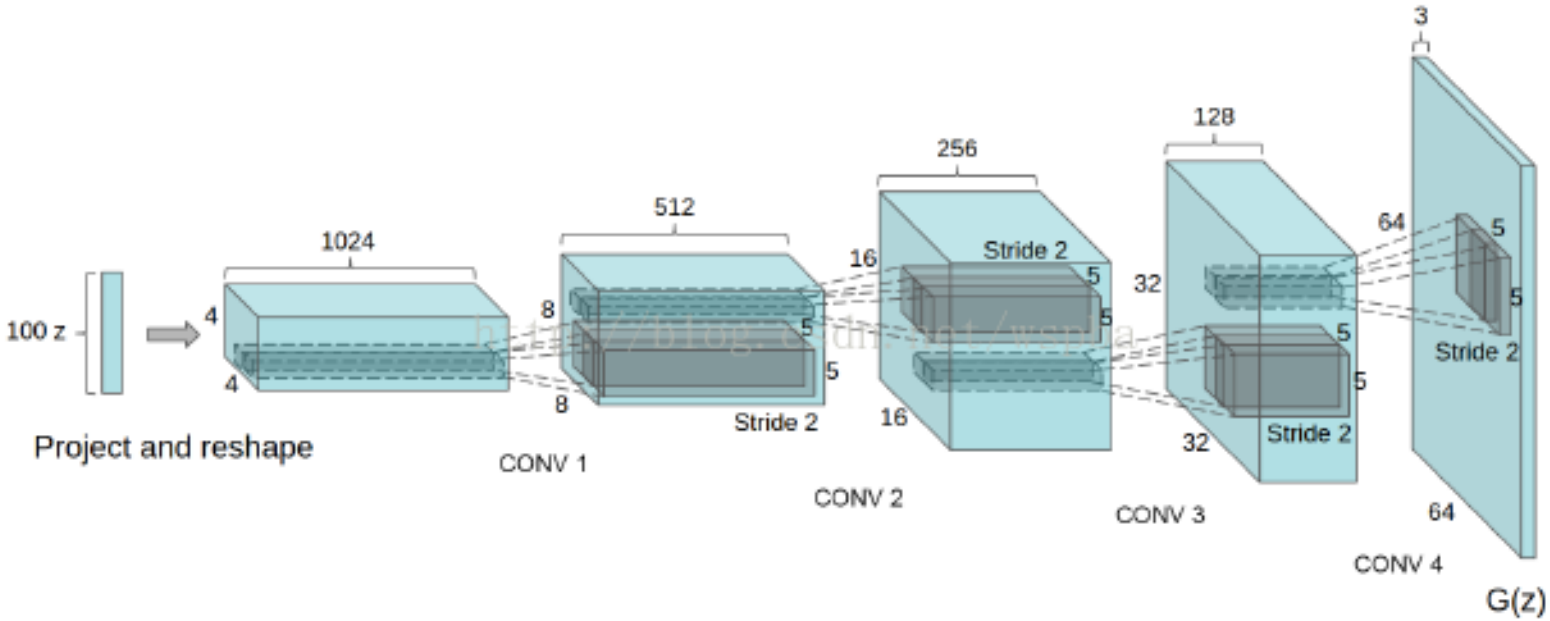
Code:

`https://wizardforcel.gitbooks.io/tensorflow-examples-aymericdamien/content/3.12_dcgan.html`

DCGANs

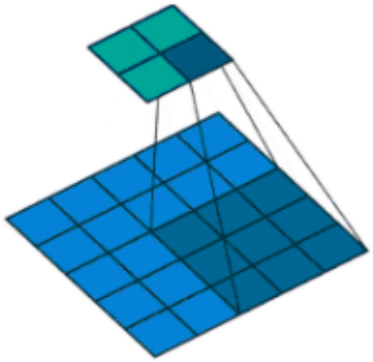
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

DCGANs

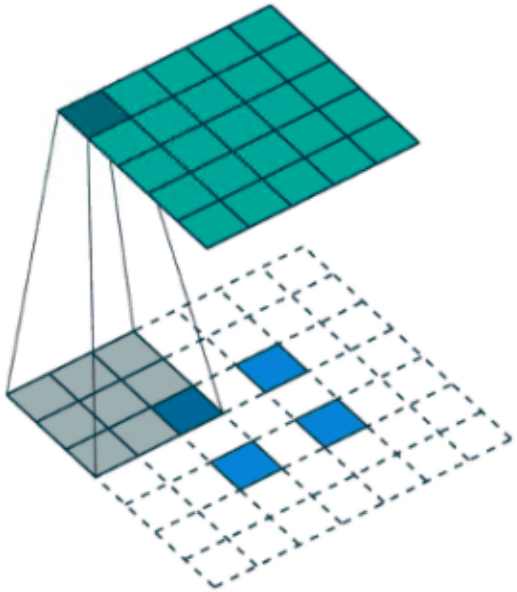


- G starts by reshaping Z into a 4-dimensional. D flattens the last convolution layer and feeds it to sigmoid output.
- A series of four fractionally-strided convolutions then convert this high level representation into a 64×64 pixel image.

Fractionally-strided convolutions



2D convolution with no padding, stride of 2 and kernel of 3

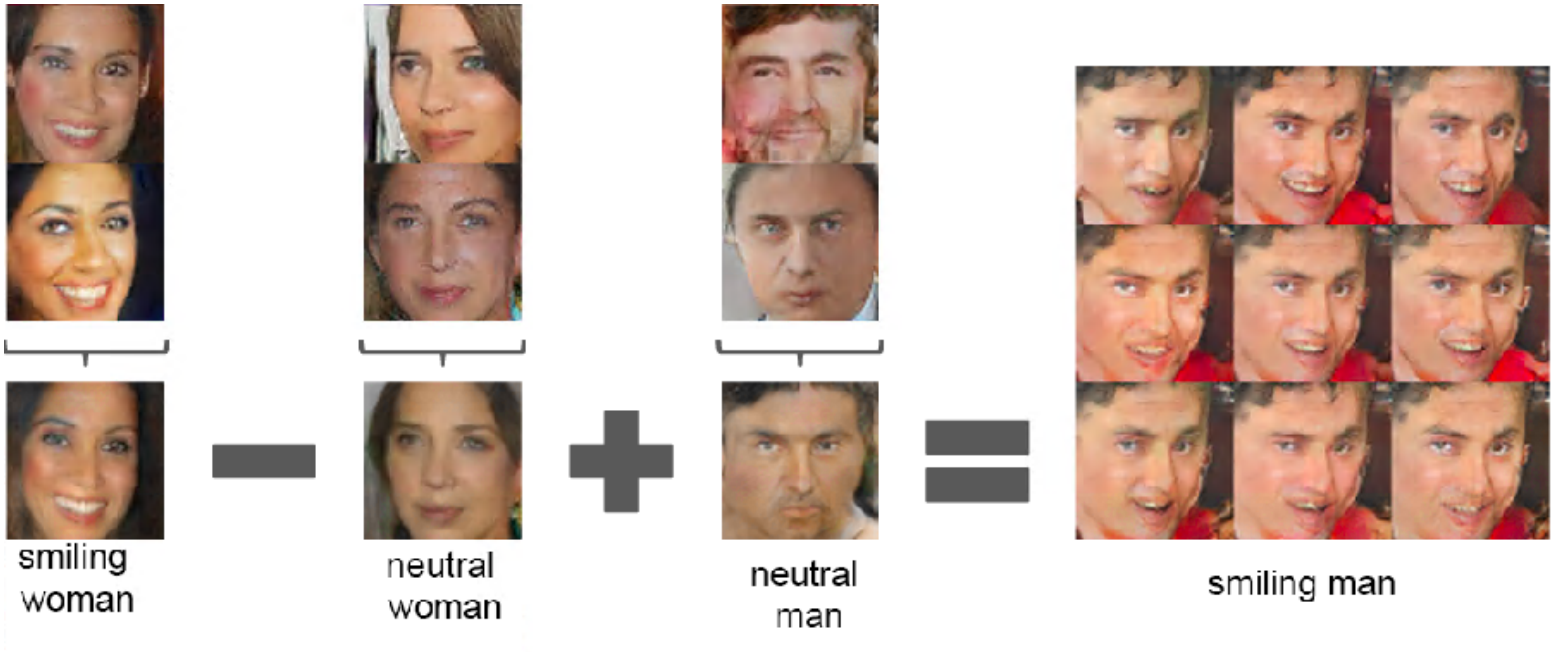


Transposed 2D convolution with no padding, stride of 2 and kernel of 3

- Left: Convolution maps a 5×5 feature map to a 2×2 feature map.
- Right: Fractionally-strided convolution (or transposed convolution) maps a 2×2 feature map to a 5×5 feature map. It is padding plus standard convolution.

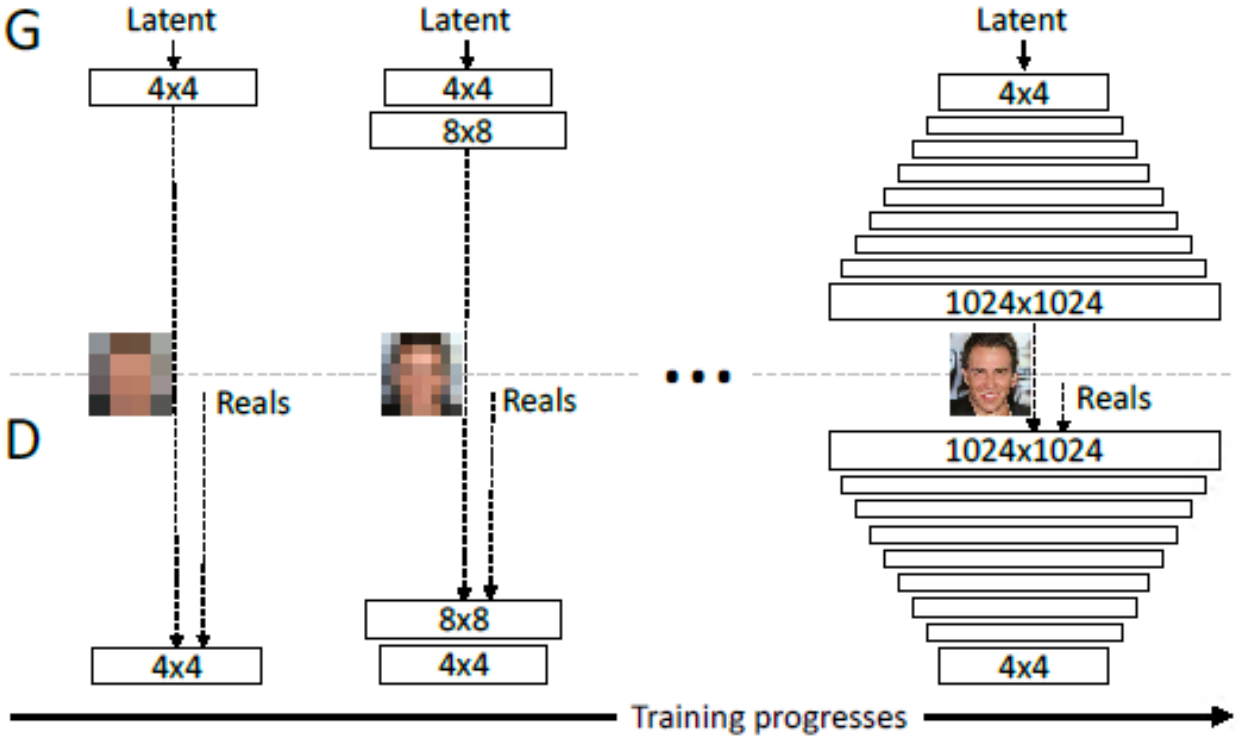
DCGAN Results

DCGAN generates better images than the original GAN.



Progressive Growing GANs (Karras *et al.* 2018)

- A training methodology for GANs where we start with low-resolution images, and then progressively increase the resolution by adding layers to the networks.



Progressive Growing GANs (Karras *et al.* 2018)

- Training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels.
- As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images.
- All existing layers remain trainable throughout the process.

Advantages:

- Early on, the generation of smaller images is substantially more stable because there is less class information and fewer modes.
- By increasing the resolution little by little we are continuously asking a much simpler question compared to the end goal of discovering a mapping from latent vectors to 1024×1024 images.

Progressive Growing GANs (Karras *et al.* 2018)

Figure 5: 1024 × 1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations.

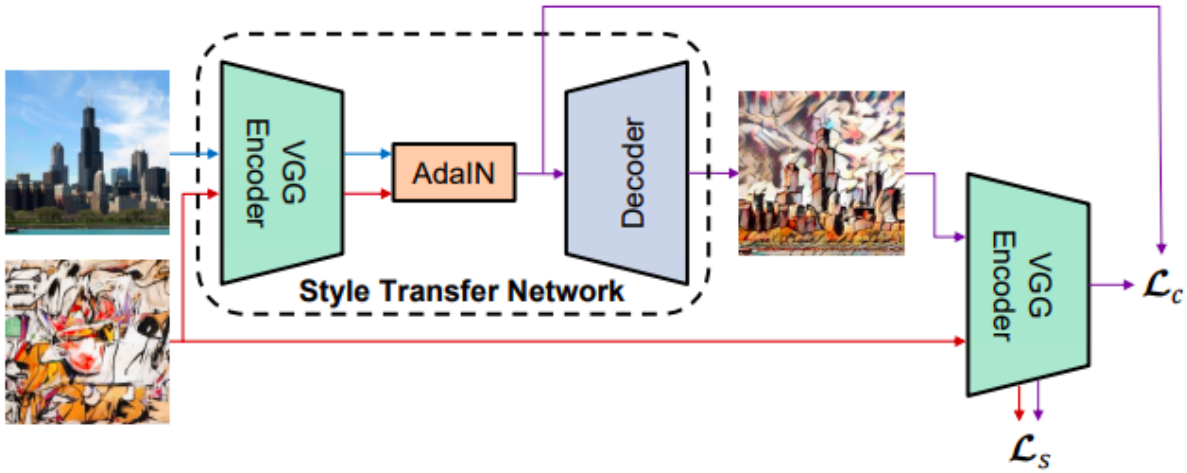
See `https://youtu.be/G06dEcZ-QTg`. Training process at 0.49.

A Style-Based Generator Architecture for GAN (Kerra *et al.* 2019)

- A new generator architecture for GAN motivated by the style transfer literature.
- Capable to separate
  - High-level attributes (e.g., pose and identity when trained on human faces), and
  - Stochastic variations in the generated images (e.g., freckles, hair)
- Enables intuitive, **scale-specific** control of the synthesis.

`https://www.youtube.com/watch?v=kSLJriaOumA & t=190s`

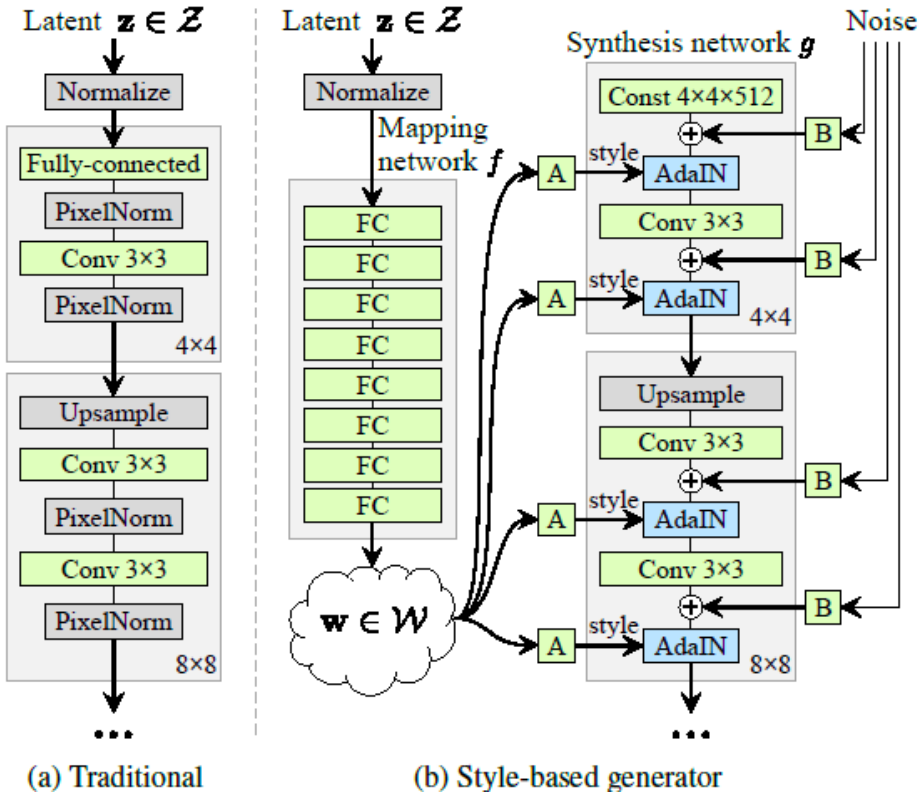
Style Transfer with Adaptive Instance Normalization (Huang 2017)



- AdaIN receives a content input x and a style input y
- Aligns the channel-wise mean and variance of x to match those of y

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

For each feature map, $\mu(x) = \mu(y)$ and $\sigma(x) = \sigma(y)$, which makes x of the same style as y.



(a) Traditional (b) Style-based generator

- Traditional: Latent code **z** is fed to input layer of generator.
- StyleGAN: **z** is mapped to **w**, which is via affine transformation turned into
  - **style codes**: $\mathbf{y} = \{(y_{s,i}, y_{b,i})\}$, where i indexes conv layer.
  - Adaptive instance normalization *AdaIN* is applied to feature map $\mathbf{x}_i$ from conv layer:
$$AdaIN(\mathbf{x}_i, \mathbf{y}) = y_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + y_{b,i}$$
  - Gaussian noise is added after each convolution.

- **Style codes**: $\mathbf{y} = \{(y_{s,i}, y_{b,i})\}$, where i indexes conv layer.
- Adaptive instance normalization *AdaIN* is applied to feature map $\mathbf{x}_i$ from conv layer:

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

To generate image of style $\mathbf{y}$, we ensure the pixel values of $\mathbf{x}_i$ be around $y_{b,i}$, and their variance be $y_{s,i}$.

- Style codes for coarse resolutions (small i) determine high-level features such as pose, face shape, eyeglasses, general hair style, etc.
- Style codes for fine resolutions (large i) determine low-level features such smaller scale facial features, hair style, color scheme.
- Noise affects only inconsequential stochastic variation.

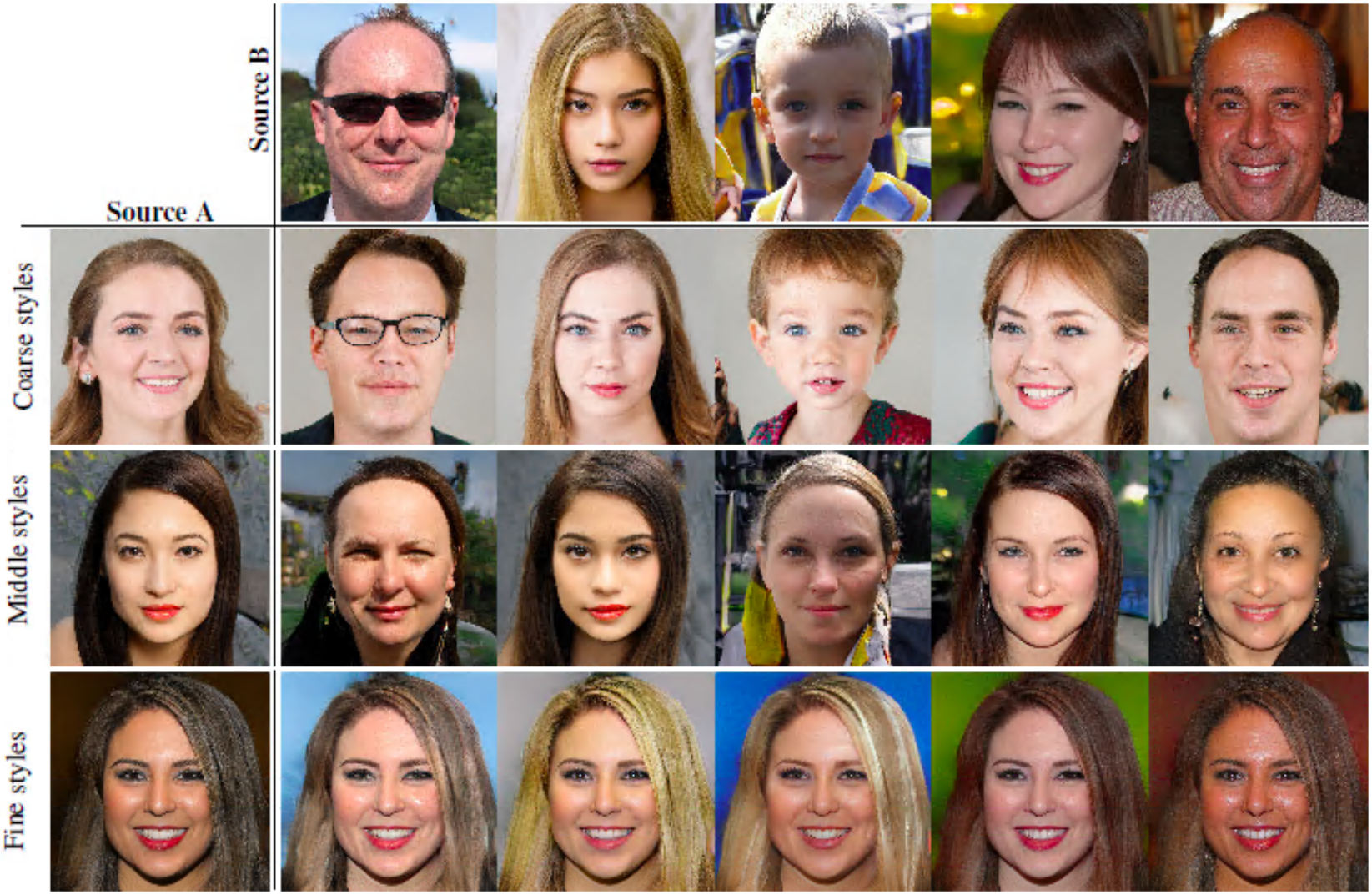
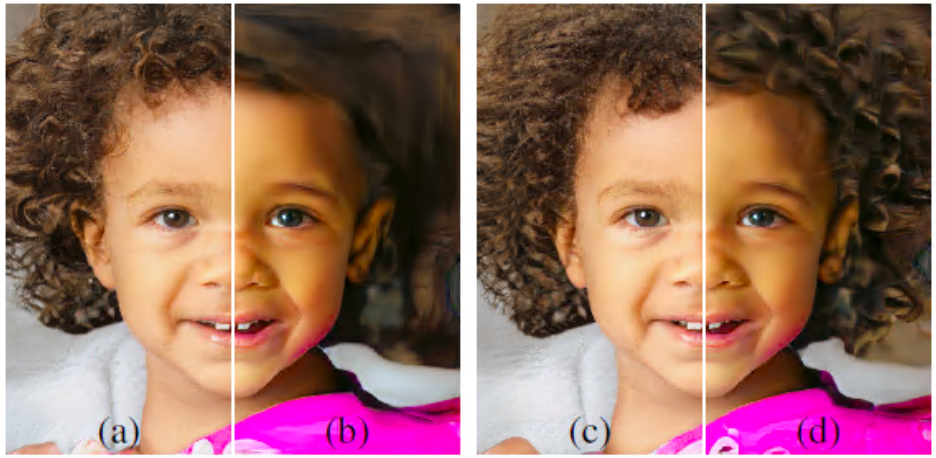


Figure 3. Two sets of images were generated from their respective latent codes (sources A and B); the rest of the images were generated by copying a specified subset of styles from source B and taking the rest from source A. Copying the styles corresponding to coarse spatial resolutions ($4^2 - 8^2$) 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.

- Fix $\mathbf{z}$, generate 100 images, each time with different noise.
- Areas affected by noise include hair, silhouettes, and parts of background.



- (a) Noise at all layers; (b) No noise; (c) Noise at fine fine layers; (d) Noise at coarse layers.

Outline

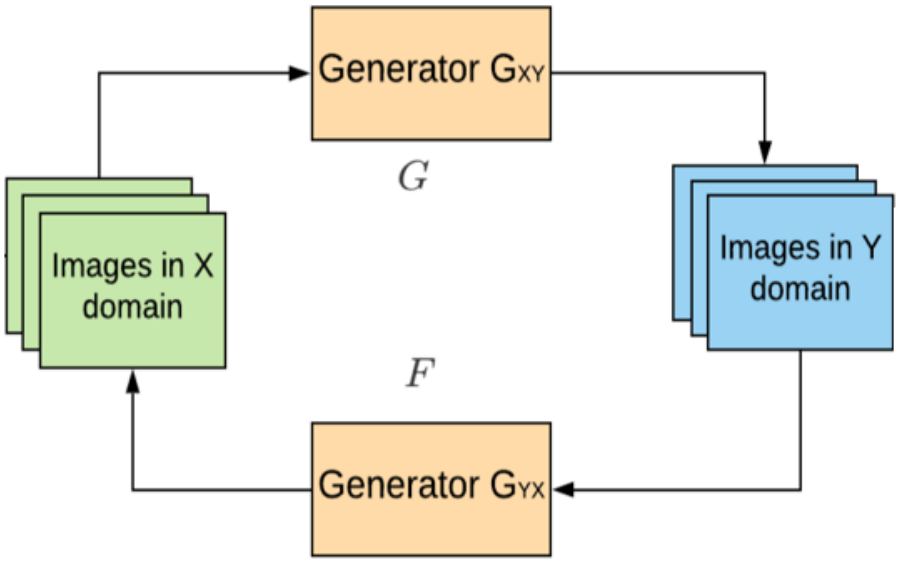
GAN Applications (Gui *et al.* 2020)

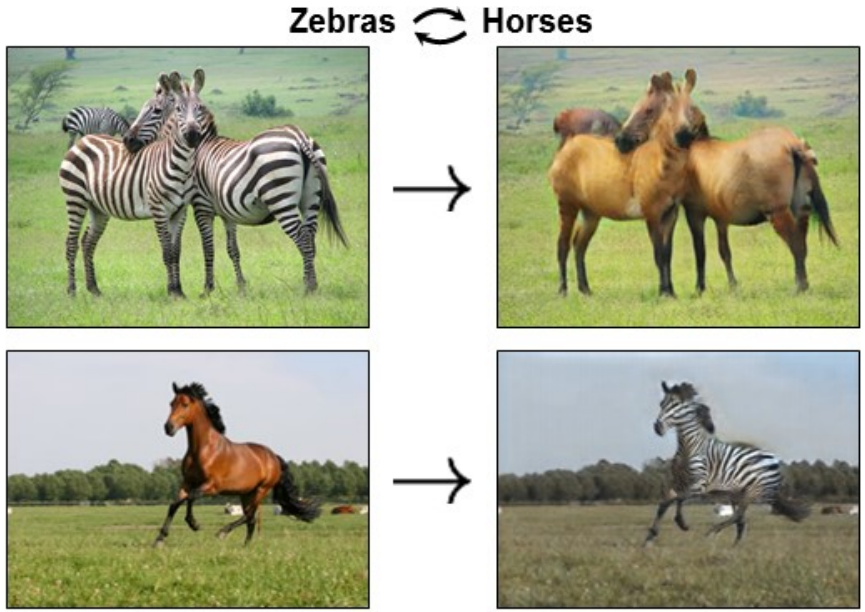
- Image applications
  - Image super-resolution (`https://www.youtube.com/watch?v=nKtE-V6LNpE`)
  - Image synthesis
  - Image texture synthesis
  - Video
  - Deepfake (`https://www.youtube.com/watch?v=dCKbRCUyop8&t=393s`)
- NLP, music, audio, etc.

We will talk about image-to-image translation.

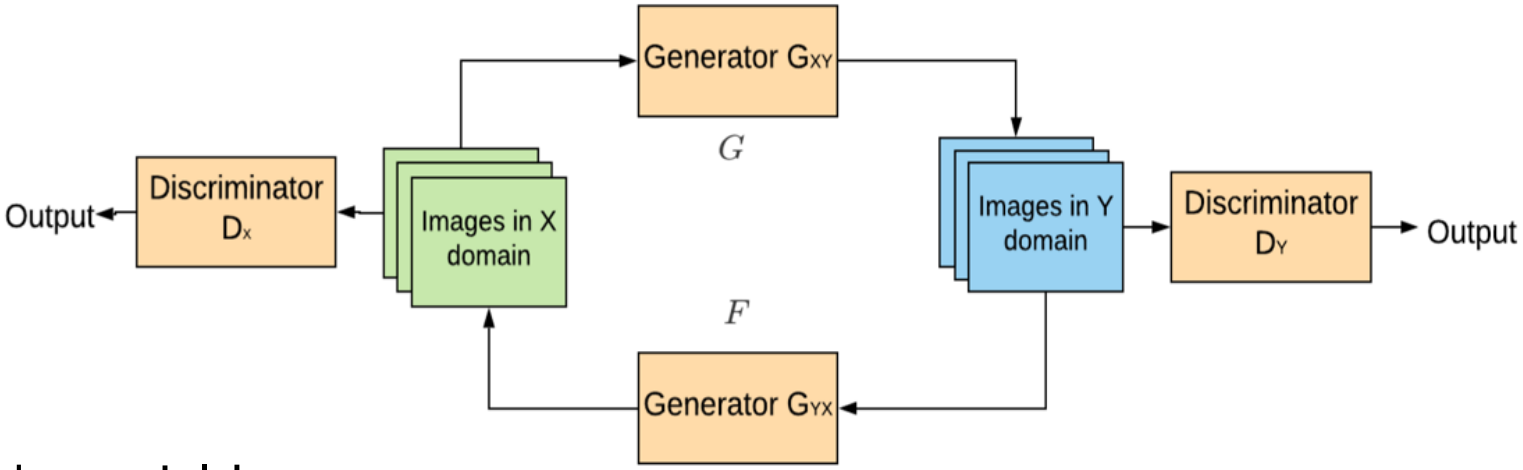
CycleGAN (Zhu *et al.* 2017)

- 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

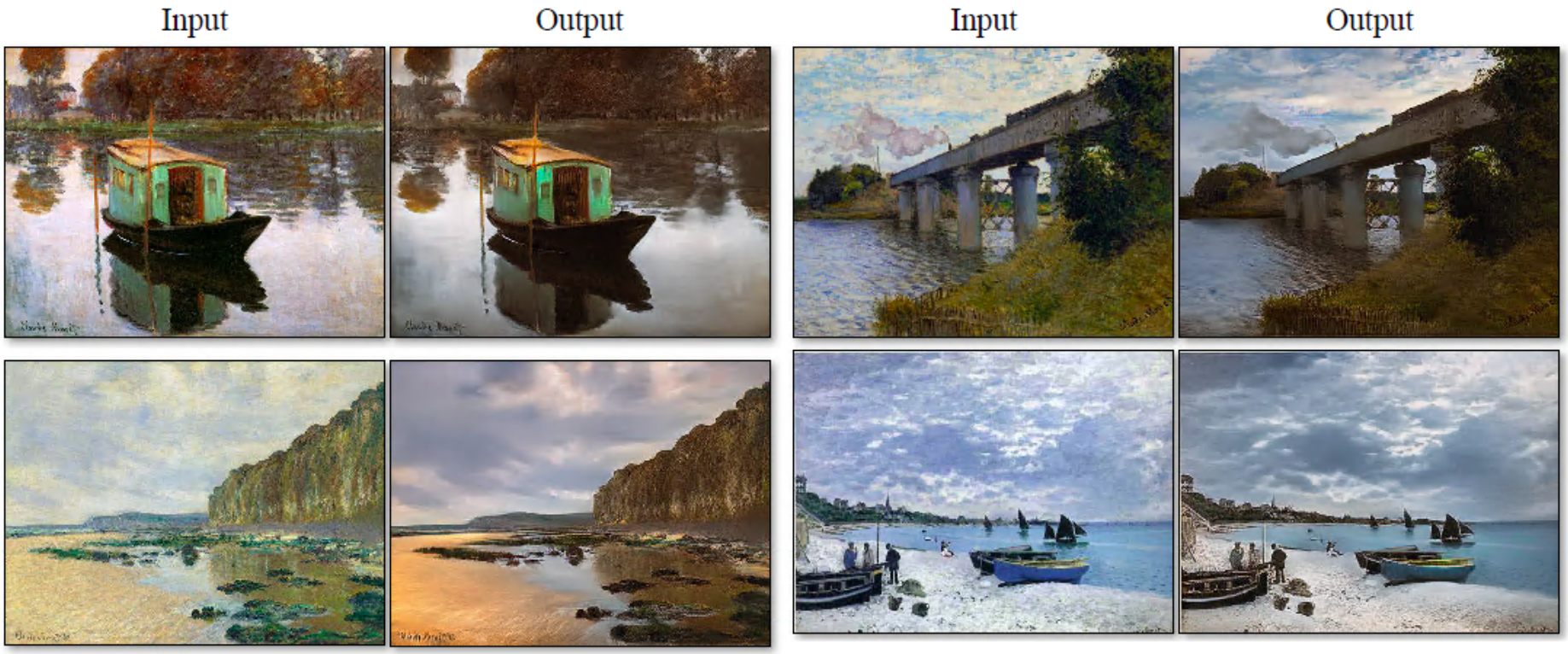
$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1]$$

- Total Loss

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

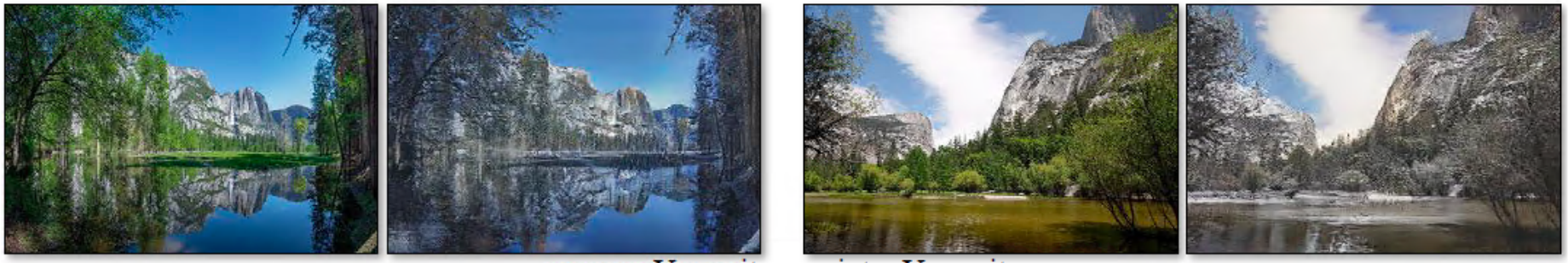
CycleGAN Results

CycleGAN Results



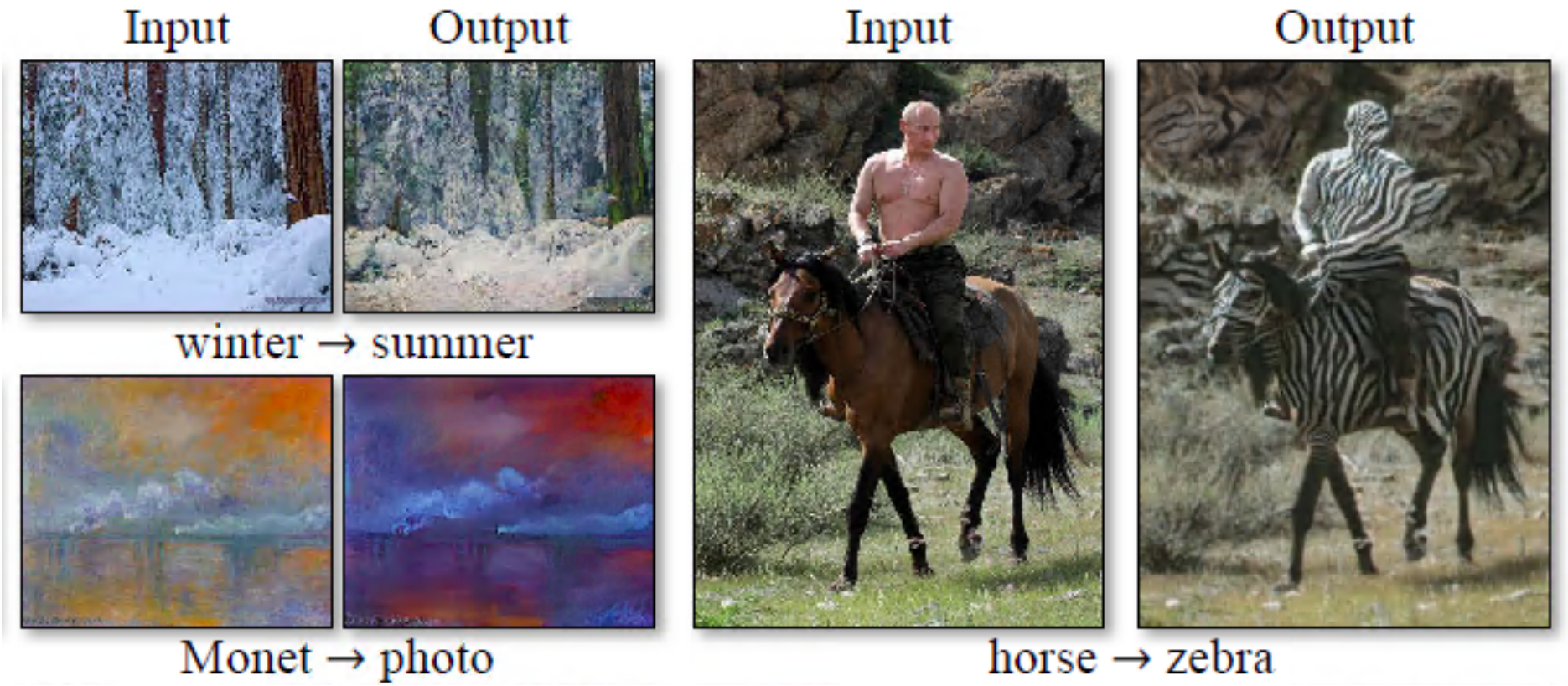
CycleGAN Results

winter Yosemite → summer Yosemite



summer Yosemite → winter Yosemite

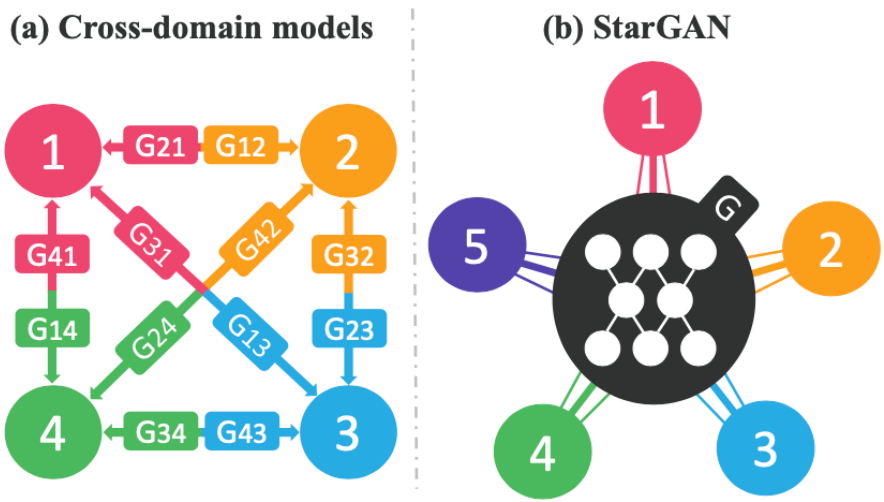
CycleGAN Results: Failures



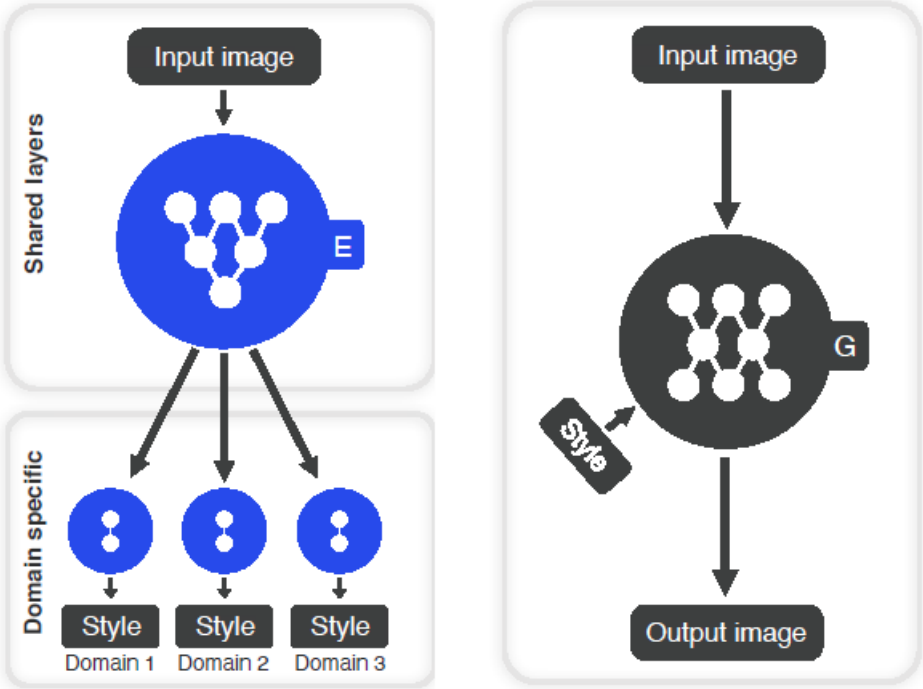
StarGAN (Choi *et al.* 2018)

- Different image attributes can be considered as different domains:
  - happy, angry, sad, ...
  - blond hair, aged, pale skin, ...

  Image style transfer can be achieved via **multi-domain image-to-image translation.**
- Instead of building a mapping between every pair of domains, **StarGAN** learns one generator to facilitate mappings among multiple domains.



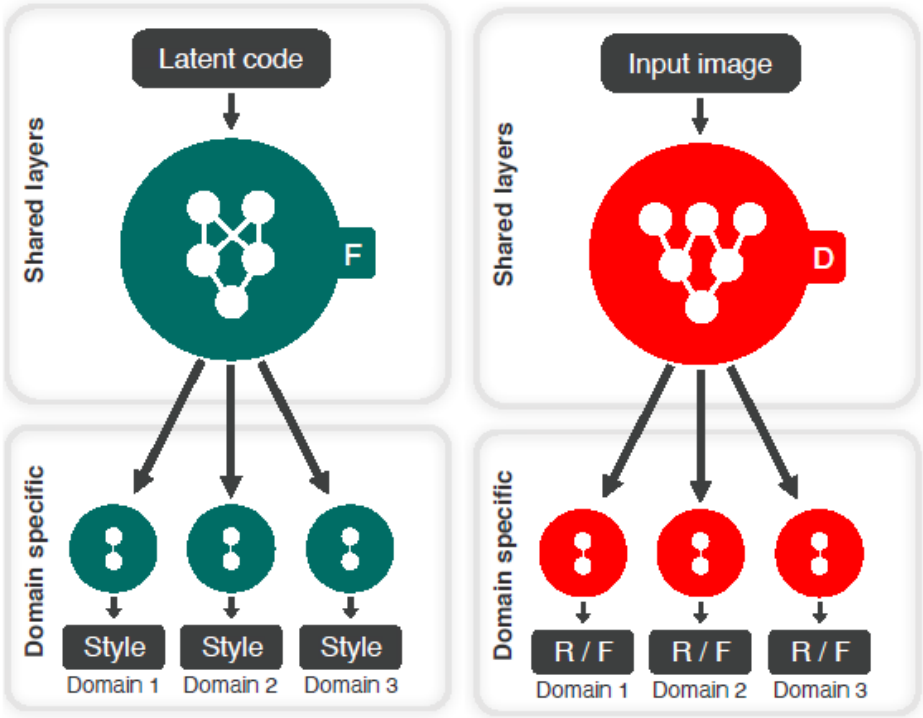
StarGAN v2 (Choi *et al.* 2020)



- It uses a **Style Encoder** $\mathbf{s} = E_y(\mathbf{x}_r)$ to extract the style code of a reference image $\mathbf{x}_r$, when it is consider from domain y.
- It uses a **Generator** $G(\mathbf{x}, \mathbf{s})$ to translate an input image $\mathbf{x}$ to the domain indicated by a style code $\mathbf{s}$. The style code is injected into G using AdaIN.

Referece-guided imagine synthesis:Style encoder + Generator

StarGAN2 (Choi *et al.* 2020)



Two auxiliary networks are used to train the style encoder and the generator:

- The **Mapping Network** $\mathbf{s} = F_y(\mathbf{z})$ maps a latent code $\mathbf{z}$ to a style code for domain y. The **Discriminator** distinguishes between real and fake images for each domain.

Latent-guided imagine synthesis: Mapping Network + Generator

Training of StarGAN2)

- **Adversarial loss**:
  - Randomly sample latent code $\mathbf{z}$ and target domain $\tilde{y}$
  - Generate a target style code $\tilde{\mathbf{s}} = F_{\tilde{y}}(\mathbf{z})$
  - Take an real image $\mathbf{x}$ from domain y, and translate it to domain $\tilde{y}$ using $G(\mathbf{x}, \tilde{\mathbf{s}})$.
  - The discriminator should be able to tell that $\mathbf{x}$ is a real image the domain y and $G(\mathbf{x}, \tilde{\mathbf{s}})$ a fake image of the domain $\tilde{y}$.

  So, we have the following loss:

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x},y}[\log D_y(\mathbf{x})] + \mathbb{E}_{\mathbf{x},\tilde{y},\mathbf{z}}[\log(1 - D_{\tilde{y}}(G(\mathbf{x}, \tilde{\mathbf{s}}))]$$

Training of StarGAN2)

- **Style reconstruction loss**: If we feed $G(\mathbf{x}, \tilde{\mathbf{s}})$ and $\tilde{y}$ to the style encoder, we should recover $\tilde{\mathbf{s}}$:

$$\mathcal{L}_{sty} = \mathbb{E}_{\mathbf{x},\tilde{y},\mathbf{z}}[||\tilde{\mathbf{s}} - E_{\tilde{y}}(G(\mathbf{x}, \tilde{\mathbf{s}}))||_1]$$

- **Style diversification loss**: Want G to produce diverse images:

$$\mathcal{L}_{ds} = \mathbb{E}_{\mathbf{x},\tilde{y},\mathbf{z}_1,\mathbf{z}_2}[||G(\mathbf{x}, \tilde{\mathbf{s}}_1) - ||G(\mathbf{x}, \tilde{\mathbf{s}}_2)||_1]$$

where $\tilde{\mathbf{s}}_i = F_{\tilde{y}}(\mathbf{z}_i)$ for $i = 1$ or 2. $\mathcal{L}_{ds}$ will be maximized in the full objective.

Training of StarGAN2)

- **Cycle loss**:

$$\mathcal{L}_{cyc} = \mathbb{E}_{\mathbf{x},y,\tilde{y},\mathbf{z}}[||\mathbf{x} - G(G(\mathbf{x}, \tilde{\mathbf{s}}), \hat{\mathbf{s}})||_1]$$

  where $\hat{\mathbf{s}} = E_y(\mathbf{x})$ is the style code of $\mathbf{x}$ given that it is considered from domain y.

- **Full objective**:

$$\min_{G,F,E} \max_{D} \mathcal{L}_{adv} + \lambda_1 \mathcal{L}_{sty} - \lambda_2 \mathcal{L}_{ds} + \lambda_3 \mathcal{L}_{cyc}$$

StarGAN2 Results

StarGAN2 Results

StarGAN2 Results

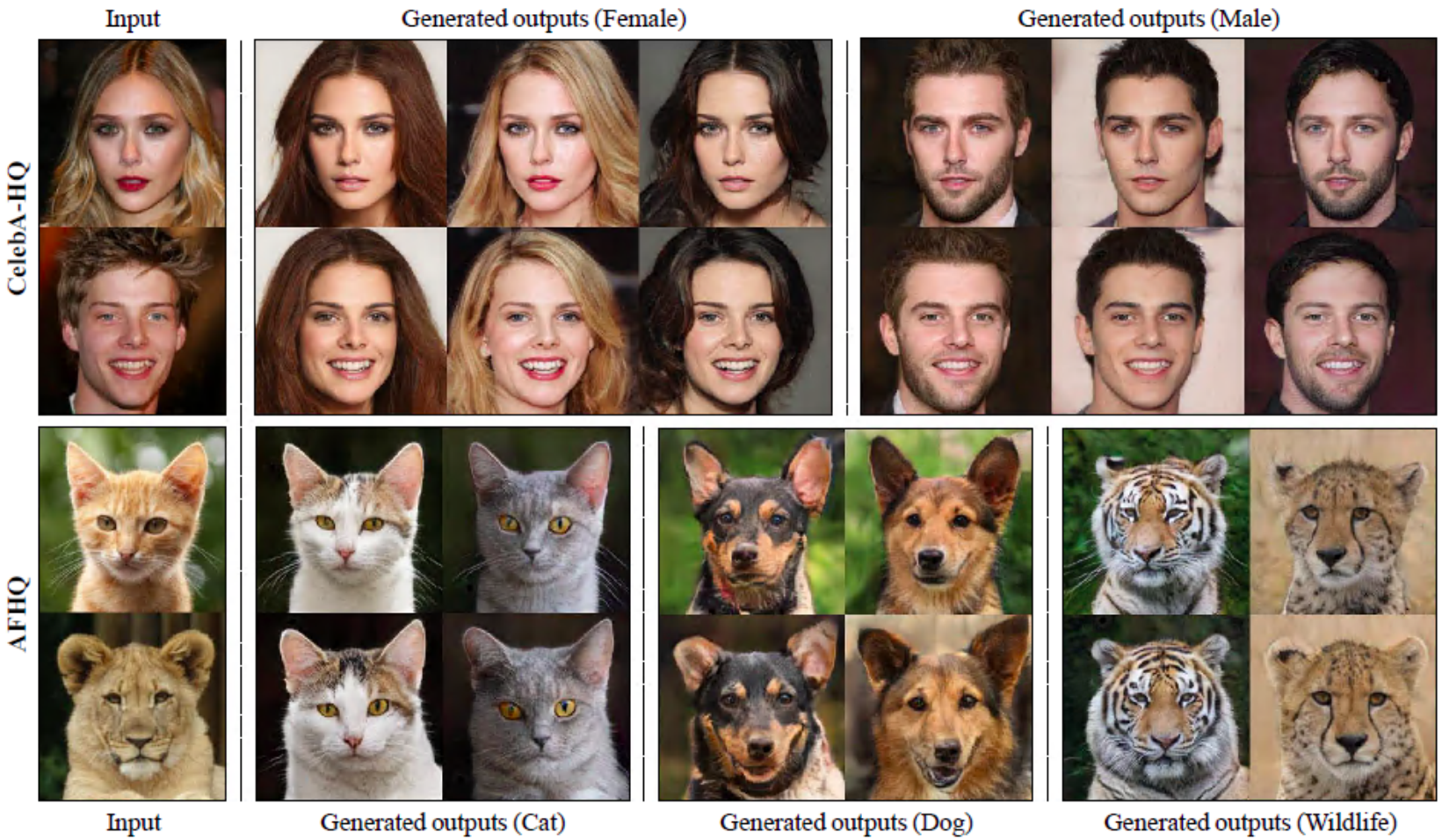


Figure 1. Diverse image synthesis results on the CelebA-HQ dataset and the newly collected animal faces (AFHQ) dataset. The first column shows input images while the remaining columns are images synthesized by StarGAN v2.

Outline

Outline

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.