

# Generative Adversarial Networks (GANs)

ML Instruction Team, Fall 2022

CE Department  
Sharif University of Technology

# Discriminative vs Generative Models

- Machine learning models can be classified into two types: Discriminative and Generative.
- A **discriminative model** makes predictions on the unseen data based on conditional probability and can be used for classification or regression problems.
- A **generative model** focuses on the latent distribution of a dataset to return a probability for an example.

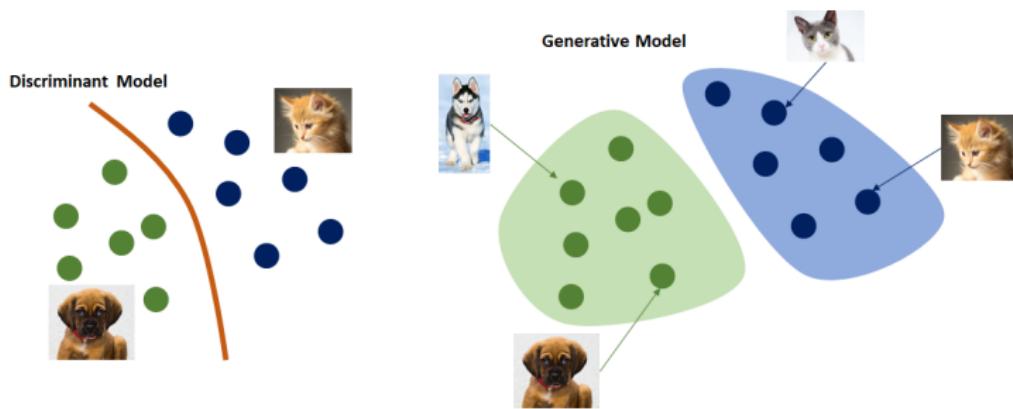


Figure: Discriminative vs Generative Models Source

# Generative Models

- Generative models have two types:

- ▶ **Explicit likelihood models** are defined by an explicit specification of the density, and so their unnormalized complete likelihood can be usually expressed in closed form.
- ▶ **Implicit probabilistic models** are defined naturally in a sampling procedure and often induce a likelihood function that cannot be expressed explicitly.

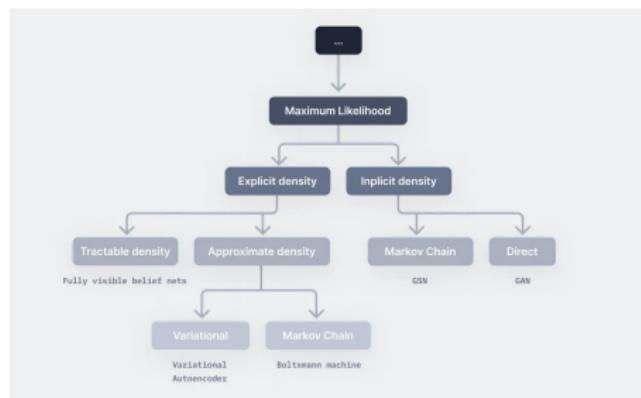


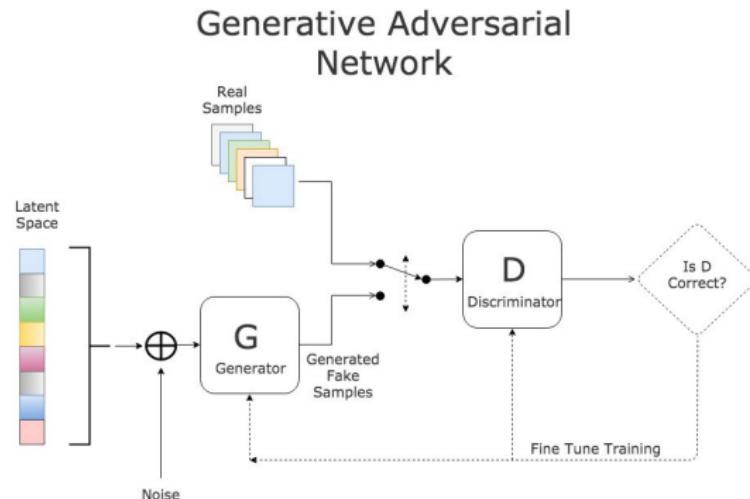
Figure: IAN Goodfellow's presentation of Generative Models Taxonomy [Source](#)

# Generative Models

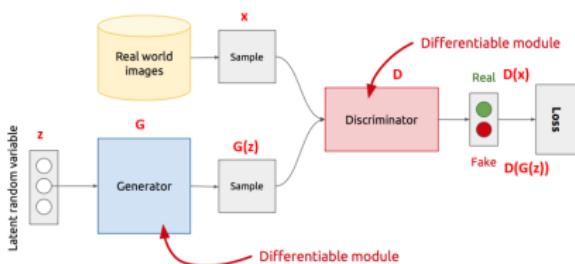
- Generative models aim for a complete probabilistic description of the data. With these models, the goal is to construct the joint probability distribution  $P(X, Y)$  – either directly or by first computing  $P(X|Y)$  and  $P(Y)$  – and then inferring the conditional probabilities required to classify new data.
- Generative models are used for:
  - ① **Density estimation** is the task of estimating the probability density function (PDF) of a random variable.
  - ② **Data generation** is the task of generating new data samples from a given distribution.
  - ③ **Data imputation** is the task of filling in missing data.
  - ④ **Data compression** is the task of reducing the amount of information required to represent a dataset.

# Generative Adversarial Networks - Architecture

- So how do Generative Adversarial Networks work?
- GAN composes of two deep networks:
  - ▶ The generator
  - ▶ The discriminator.

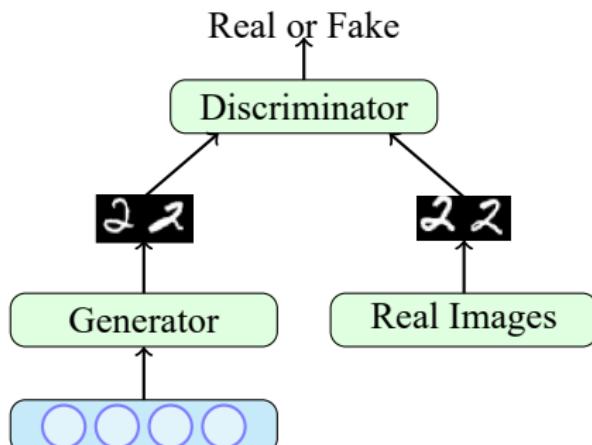


# Generative Adversarial Networks - Structure



- So let's look at the full picture
- Let  $G_\phi$  be the generator and  $D_\theta$  be the discriminator ( $\phi$  and  $\theta$  are the parameters of  $G$  and  $D$ , respectively)
- We have a neural network based generator which takes as input a noise vector  $z \sim N(0, I)$  and produces  $G_\phi(z) = X$
- We have a neural network based discriminator which could take as input a real  $X$  or a generated  $X = G_\phi(z)$  and classify the input as real/fake

# GAN: Loss Function



$$z \sim N(0, I)$$

[scale=0.7]

- What should be the objective function of the overall network?
- Let's look at the objective function of the generator first
- Given an image generated by the generator as  $G_\phi(z)$  the discriminator assigns a score  $D_\theta(G_\phi(z))$  to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake

# GAN: Loss Function

- For a given  $z$ , the generator would want to maximize  $\log D_\theta(G_\phi(z))$  (log likelihood) or minimize  $\log(1 - D_\theta(G_\phi(z)))$
- This is just for a single  $z$  and the generator would like to do this for all possible values of  $z$ ,
- For example, if  $z$  was discrete and drawn from a uniform distribution (*i.e.*,  $p(z) = \frac{1}{N} \forall z$ ) then the generator's objective function would be

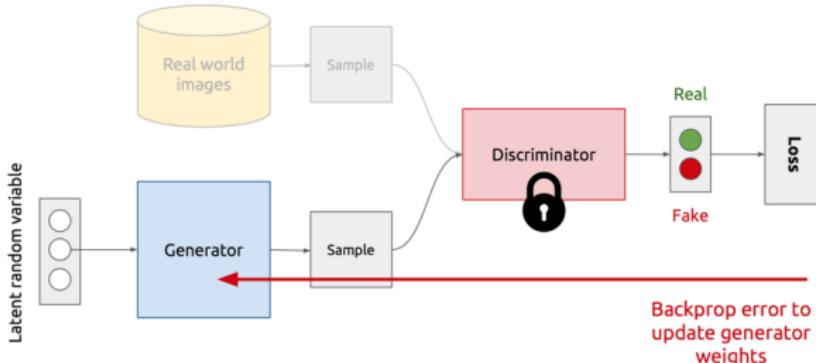
$$\min_{\phi} \sum_{i=1}^N \frac{1}{N} \log(1 - D_\theta(G_\phi(z)))$$

# GAN: Loss Function

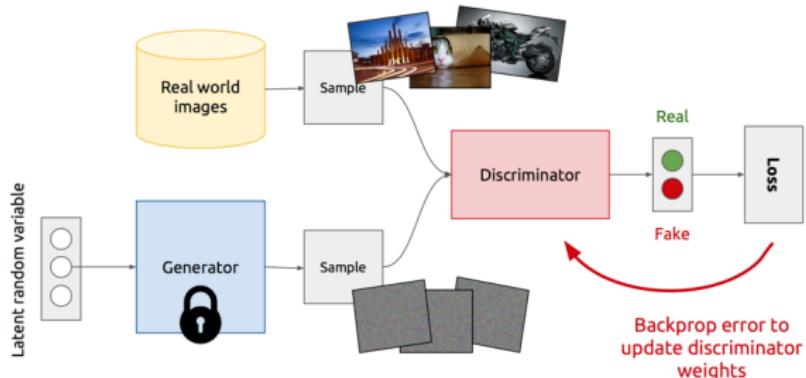
- However, in our case,  $z$  is continuous and not uniform ( $z \sim N(0, I)$ ) so the equivalent objective function would be

$$\min_{\phi} \int p(z) \log(1 - D_{\theta}(G_{\phi}(z)))$$

$$\min_{\phi} E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$



# GAN: Loss Function



- Now let's look at the discriminator
- The task of the discriminator is to assign a high score to real images and a low score to fake images
- And it should do this for all possible real images and all possible fake images
- In other words, it should try to maximize the following objective function

$$\max_{\theta} E_{x \sim p_{data}} [\log D_{\theta}(x)] + E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$

# GAN: Loss Function

- If we put the objectives of the generator and discriminator together we get a minimax game

$$\begin{aligned} \min_{\phi} \max_{\theta} & [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) \\ & + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))] \end{aligned}$$

- The first term in the objective is only w.r.t. the parameters of the discriminator ( $\theta$ )
- The second term in the objective is w.r.t. the parameters of the generator ( $\phi$ ) as well as the discriminator ( $\theta$ )
- The discriminator wants to maximize the second term whereas the generator wants to minimize it (hence it is a two-player game)

# GAN - Training

- So the overall training proceeds by alternating between these two step
- Step 1:** Gradient Ascent on Discriminator

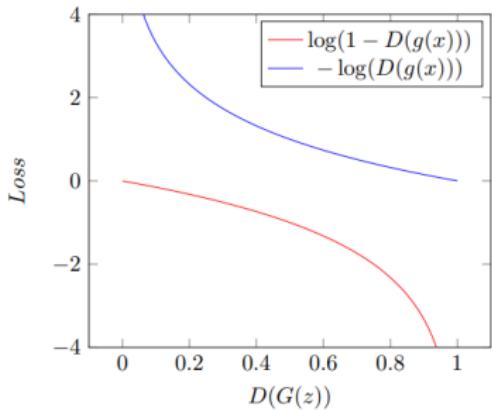
$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \text{Log } D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \text{Log}(1 - D_{\theta}(G_{\phi}(z)))]$$

- Step 2:** Gradient Descent on Generator

$$\min_{\phi} \mathbb{E}_{z \sim p(z)} \text{Log}(1 - D_{\theta}(G_{\phi}(z)))$$

- In practice, the above generator objective does not work well and we use a slightly modified objective
- Let us see why

# GAN - Training



- When the sample is likely fake, we want to give a feedback to the generator (using gradients)
- However, in this region where  $D(G(z))$  is close to 0, the curve of the loss function is very flat and the gradient would be close to 0
- Trick: Instead of minimizing the likelihood of the discriminator being correct, maximize the likelihood of the discriminator being wrong
- In effect, the objective remains the same but the gradient signal becomes better

# GAN: Problems

- Many GAN models suffer the following major problems:
  - ▶ **Mode collapse:** the generator collapses which produces limited varieties of samples
  - ▶ **Non-convergence:** the model parameters oscillate, destabilize and never converge
  - ▶ **Diminished gradient:** the discriminator gets too successful that the generator gradient vanishes and learns nothing
  - ▶ Unbalance between the generator and discriminator causing overfitting
  - ▶ Highly sensitive to the hyperparameter selections.

# Mode Collapse

- Real-life data distributions are multimodal.
- In MNIST, there are 10 major modes from digit ‘0’ to digit ‘9’.
- The samples below are generated by two different GANs.
- We can see the generator creates a single mode only. This is called mode collapse.

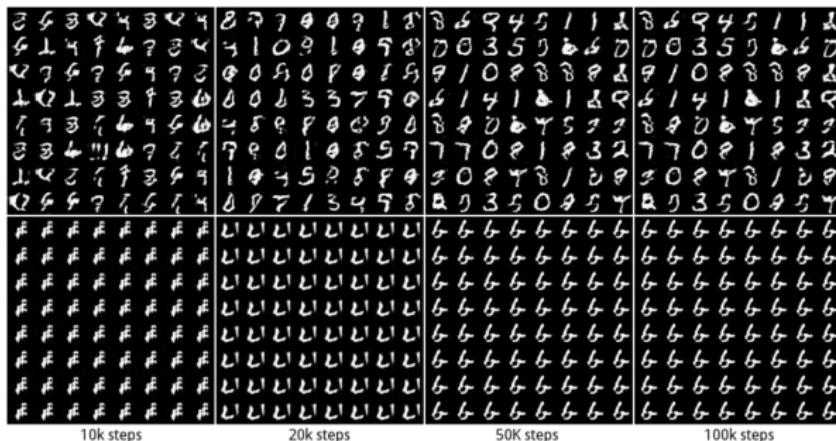


Figure: GAN mode collapse (Source)

# Why Mode Collapse in GAN?

- The images below with the same underlined color look similar and the mode starts collapsing.



Figure: Modified from (Source)

- Let's see how it may occur. The objective of the GAN generator is to create images that can fool the discriminator D the most.

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

- But let's consider one extreme case where G is trained extensively without updates to D.

# Why Mode Collapse in GAN?

- The generated images will converge to find the optimal image  $x^*$  that fool  $D$  the most, the most realistic image from the discriminator perspective.
- In this extreme,  $x^*$  will be independent of  $z$ .

$$x^* = \operatorname{argmax}_x D(x)$$

- The mode collapses to a single point.
- The gradient associated with  $z$  approaches zero.

$$\frac{\partial J}{\partial z} \approx 0$$

- Solution?
- Unrolled GAN

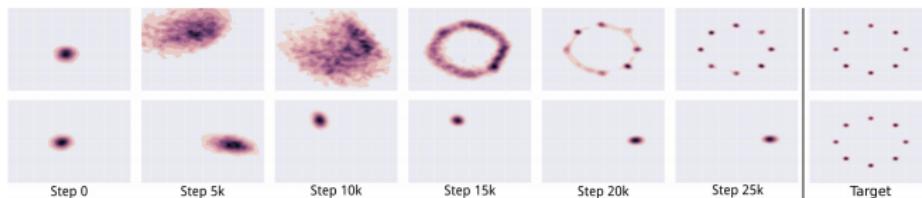


Figure: Unrolled GAN vs Vanilla GAN (Source)

# Nash Equilibrium

- GAN is based on the zero-sum non-cooperative game.
- A zero-sum game is also called **minimax**.
- In game theory, the GAN model converges when the discriminator and the generator reach a **Nash equilibrium**

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

- Nash equilibrium happens when one player will not change its action regardless of what the opponent may do.

# Nash Equilibrium

- Consider two player A and B which control the value of  $x$  and  $y$  respectively. Player A wants to maximize the value  $xy$  while B wants to minimize it.

$$\min_B \max_A V(D, G) = xy$$

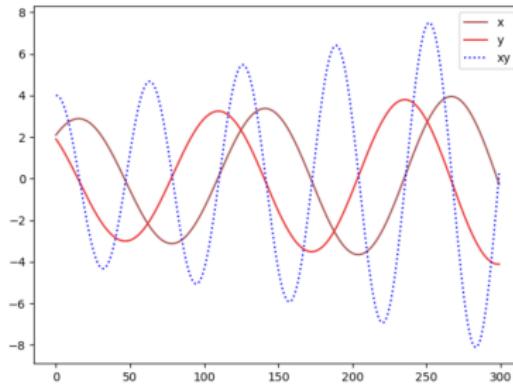
- The Nash equilibrium is  $x = y = 0$
- This is the only state where the action of your opponent does not matter. It is the only state that any opponents' actions will not change the game outcome.

# Non-convergence

- Let's see whether we can find the Nash equilibrium easily using the gradient descent.
- We update the parameter x and y based on the gradient of the value function V.

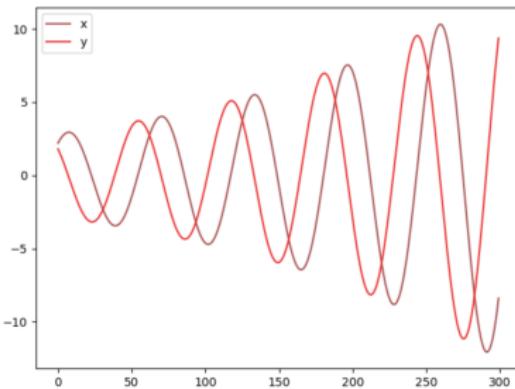
$$\Delta x = \alpha \frac{\partial(xy)}{\partial(x)}$$
$$\Delta y = -\alpha \frac{\partial(xy)}{\partial(y)}$$

- Where  $\alpha$  is the learning rate. When we plot x, y, and xy against the training iterations, we realize our solution does not converge.



# Non-convergence

- If we increase the learning rate or train the model longer, we can see the parameters  $x$ ,  $y$  is unstable with big swings.



- Our example is an excellent showcase that **some cost functions will not converge with gradient descent**, in particular for a non-convex game.

# Generative model with KL-Divergence

- Before GAN, many generative models create a model  $\theta$  that maximizes the Maximum Likelihood Estimation MLE. i.e. finding the best model parameters that fit the training data the most.

$$\hat{\theta} = \arg \max_{\theta} \prod_{i=1}^N p(x_i | \theta)$$

- KL ( $x$ ) drops to 0 for area where  $p(x) \rightarrow 0$ . For example, in the figure on the right below, the red curve corresponds to  $D(p, q)$ . It drops to zero when  $x > 2$  where  $p$  approaches 0.

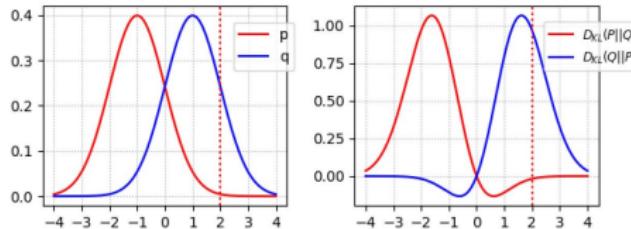


Figure:  $KL(p, q)$  is the integral of the red curve in the right.

# Generative model with JL-Divergence

- JS-divergence is defined as:

$$D_{JS}(p||q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2})$$

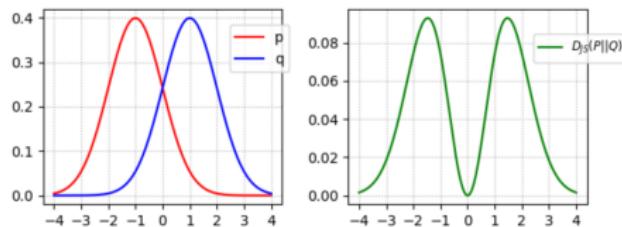


Figure:  $KL(p, q)$  is the integral of the red curve in the right.

- JS-divergence is symmetrical.
- Unlike KL-divergence, it will penalize poor images badly.
- the generator's objective function becomes:

$$\min_G V(D^*, G) = 2D_{JS}(p_r \| p_g) - 2 \log 2$$

# Deep Convolutional GANs

- We will now look at one of the popular neural networks used for the generator and discriminator (Deep Convolutional GANs) For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)

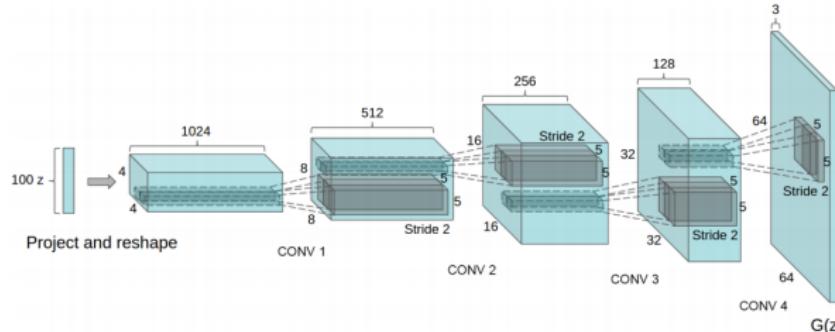


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

# Deep Convolutional GANs: Architecture

- 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

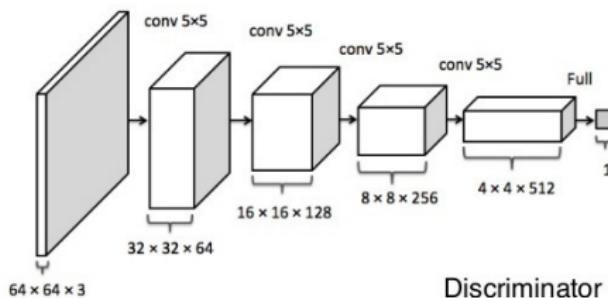


Figure: Generator (Redford et al 2015) (left) and discriminator (Yeh et al 2016) (right) used in DCGAN

# Generative Adversarial Networks - StyleGAN + CycleGAN

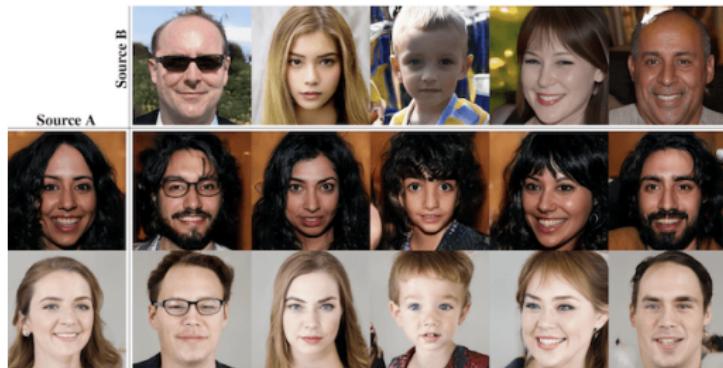


Figure: Example of One Set of Generated Faces (Left) Adopting the Coarse Style of Another Set of Generated Faces (Top)

- GAN: Lacking Control Over Synthesized Images
- Style Generative Adversarial Network(StyleGAN) Controls Style Using New Generator Model
- StyleGan is proficient in producing impressively photorealistic high-quality photos of faces and grants control over the characteristic of the created image at different specification levels by changing the style vectors and noise.

# StyleGAN Model Architecture

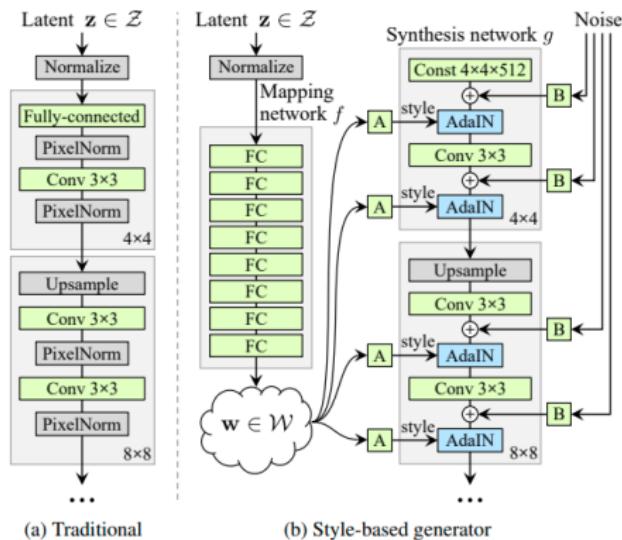


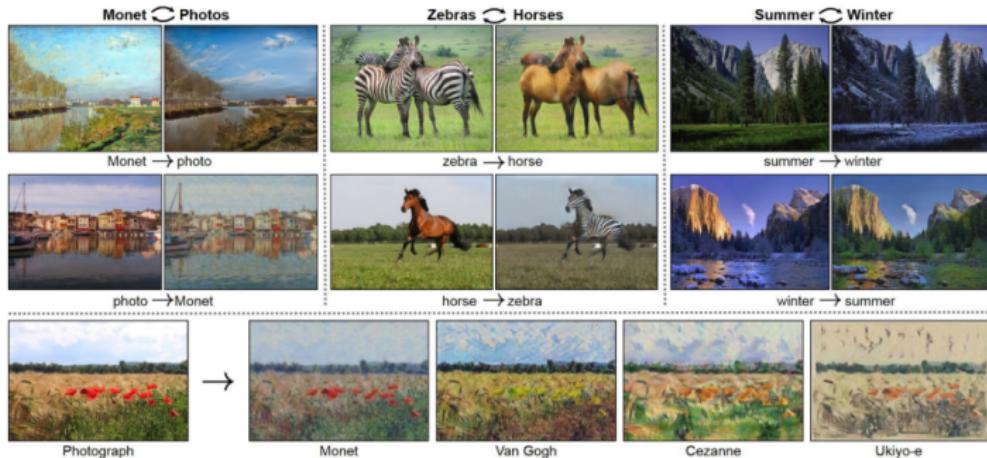
Figure: StyleGAN Architecture (Source)

# StyleGAN Model Architecture

- The StyleGAN is described as a progressive growing GAN architecture with five modifications:
  - ▶ Baseline Progressive GAN.
  - ▶ Addition of tuning and bilinear upsampling.
  - ▶ Addition of mapping network and AdaIN (styles).
  - ▶ Removal of latent vector input to generator.
  - ▶ Addition of noise to each block.
  - ▶ Addition Mixing regularization.

# CycleGAN: Unpaired Image-to-Image Translation

- Style transfer problem: change the style of an image while preserving the content.

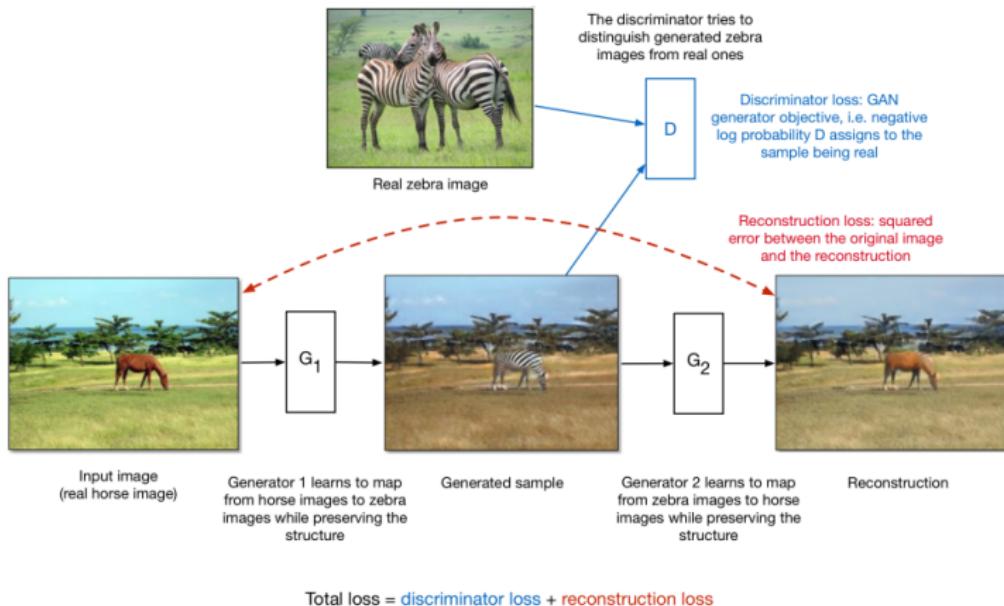


- Data: Two unrelated collections of images, one for each style

# CycleGAN: Unpaired Image-to-Image Translation

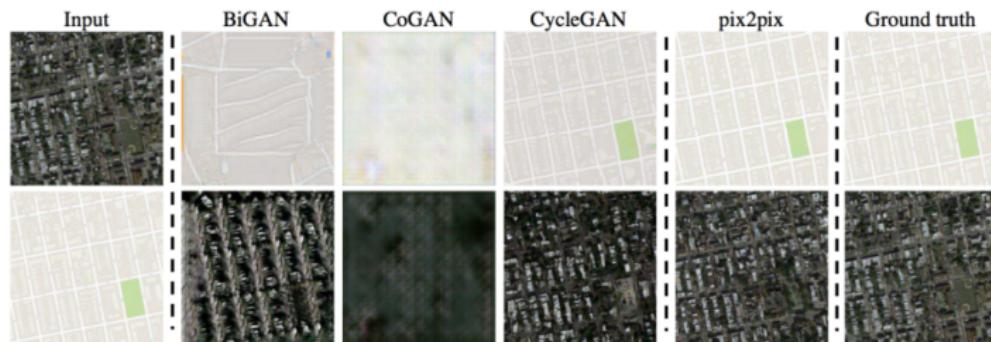
- If we had paired data (same content in both styles), this would be a supervised learning problem. But this is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
  - ▶ Train two different generator nets to go from style 1 to style 2, and vice versa.
  - ▶ Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
  - ▶ Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

# CycleGAN



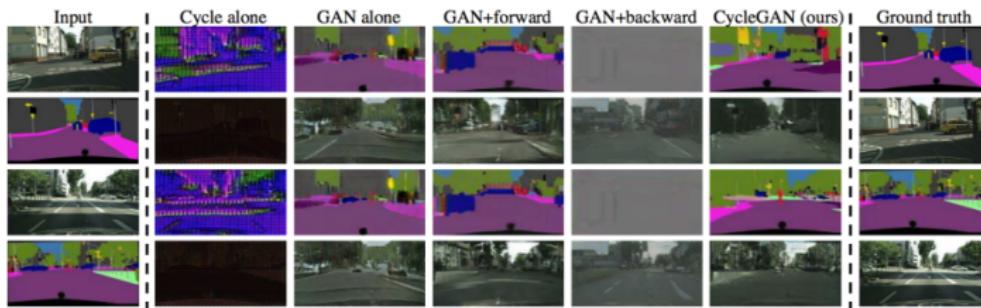
# CycleGAN

Style transfer between aerial photos and maps:



# CycleGAN: Example

- Style transfer between road scenes and semantic segmentations (labels of every pixel in an image by object category):



# Image References

- <https://www.cse.iitm.ac.in/miteshk/CS7015>
- Alec Radford, Luke Metz, Soumith Chintala Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks(<https://arxiv.org/abs/1511.06434>)
- Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros Image-to-Image Translation with Conditional Adversarial Networks(<https://arxiv.org/abs/1511.06434>)
- Tero Karras, Samuli Laine, Timo Aila A Style-Based Generator Architecture for Generative Adversarial Networks(<https://arxiv.org/abs/1812.04948>)
- Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks(<https://arxiv.org/abs/1703.10593>)
- Ian Goodfellow NIPS 2016 Tutorial: Generative Adversarial Networks(<https://arxiv.org/abs/1701.00160>)
- <https://kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html>

# Image References

- [https://www.researchgate.net/figure/Our-text-conditional-convolutional-GAN-architecture-Text-encoding-pht-is-used-by-both\\_fig2\\_303337197](https://www.researchgate.net/figure/Our-text-conditional-convolutional-GAN-architecture-Text-encoding-pht-is-used-by-both_fig2_303337197)
- <https://www.linkedin.com/in/tauil-abd-elilah-076967176/>
- <https://this-person-does-not-exist.com>
- <https://medium.com/@jordi299/about-generative-and-discriminative-models-d8958b67ad32>
- [https://www.researchgate.net/figure/Overview-of-the-GAN-training-process-Segmented-volumetric-images-are-usually-split-into\\_fig11\\_316029770](https://www.researchgate.net/figure/Overview-of-the-GAN-training-process-Segmented-volumetric-images-are-usually-split-into_fig11_316029770)
- Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio: Generative Adversarial Networks (<https://arxiv.org/abs/1406.2661>)
- [https://www.researchgate.net/figure/Progress-of-human-face-generation-with-GAN\\_fig2\\_338979046](https://www.researchgate.net/figure/Progress-of-human-face-generation-with-GAN_fig2_338979046)