Generative Adversarial Networks (GAN)

Georgia Fargetta, Ph.D.

Erasmus +
Master Degree in Computer Science
University of Rouen, France

IFOSS International Forensics Summer School

ETHICAL AND LEGAL CHALLENGES IN AI-DRIVEN FORENSIC SCIENCE

JULY 14-20, 2024

Watch a preview!

School Directors

PROF. SEBASTIANO BATTIATO, PH.D.
University of Catania

PROF. DONATELLA CURTOTTI, PH.D.
University of Foggia

PROF. GIOVANNI ZICCARDI, PH.D.
University of Milan

Speakers

others coming soon..

ALESSANDRO TRIVILINI
Scuola universitaria professionale della svizzera italiana (SUPSI)

MARTIN DRAHANSKÝ
Faculty of Information Technology, Brno University of Technology

PROF. DR. DIDIER MEUWLY
University of Twente

School location

The school will take place at Sampieri, Sicily
https://www.hotelbaiasamuele.it/en/

Social Network

IFOSS

@ifoss_official

@ifoss_official

IFOSS

www.ifoss.it

info@ifoss.it

Generative Adversarial Networks (GAN)

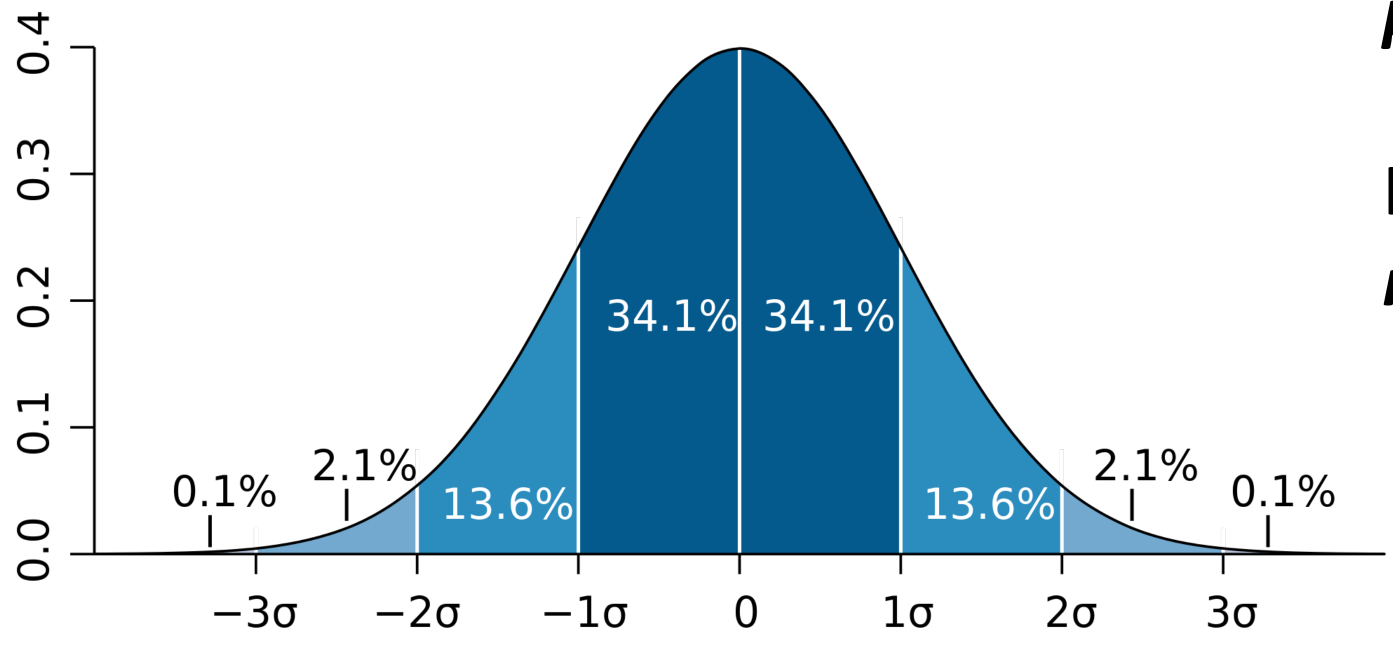
GANs are Neural Networks able to generate outputs (e.g. a picture of a human face) that appear to be a sample from the distribution of the training set (e.g. set of other human faces).

Applications:

- When training data is insufficient, GANs can learn about your data and generate synthetic images that augment your dataset.
- Create images that look like photographs of human faces.
- Generate images from descriptions (text to image synthesis).
- Improve the resolution (low-resolution to high-resolution).
- GANs can be used to produce synthetic, high-fidelity audio or perform voice translations.

Probability Distribution Function (PDF)

Is a function that expresses the probability of a random variable X to have a certain value.



Gaussian distribution with mean ***μ*** and standard deviation ***σ***.

If ***μ*** *= 0* and ***σ*** *= 1 is known as* ***normal distribution.***

Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution

Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)

Inverse transfrom sampling method

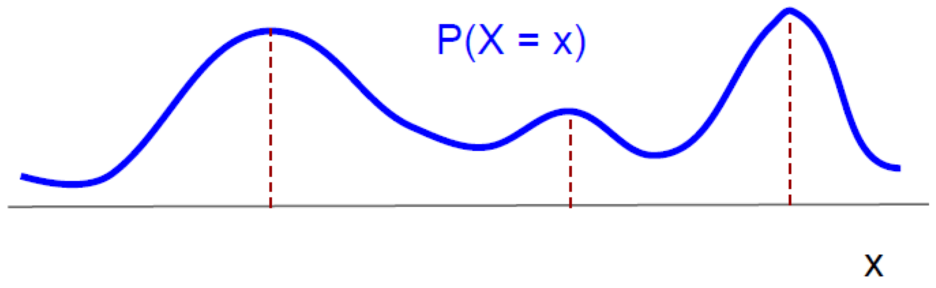
- Generate a random number *u* from the uniform distribution
- Find the inverse of the desidered CDF $F_X^{-1}(x)$
- Generate a random sample with $F_X^{-1}(u)$, the computed value has CDF $F_X\ (x)$

Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don’t know the law of the function we want to estimate?

Generating Elements of a Given Distribution

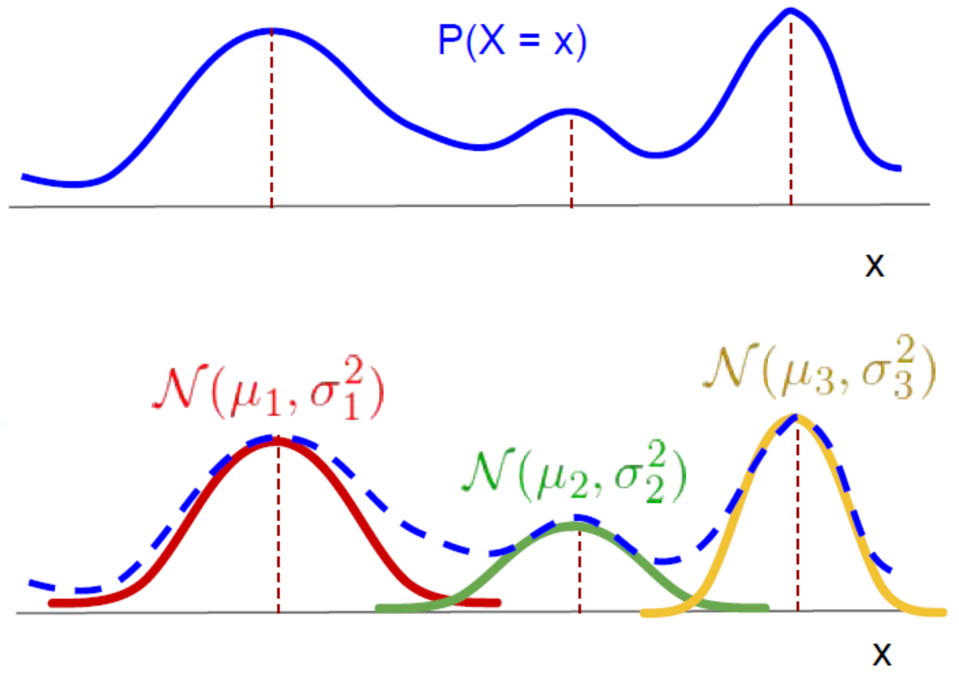
- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don't know the law of the function we want to estimate?



Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don't know the law of the function we want to estimate?

Fitting a Gaussian Mixture Model with EM algorithm

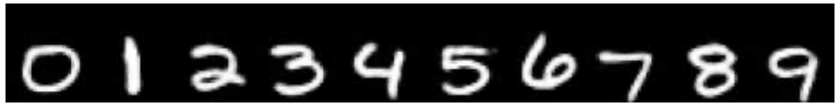


Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don’t know the law of the function we want to estimate?
- What if we augment the dimensionality of the data?

Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don't know the law of the function we want to estimate?
- What if we augment the dimensionality of the data?



Generating Elements of a Given Distribution

- We are able to generate pseudo-random numbers from a uniform distribution
- We are also able to generate random numbers from a given distribution (e.g., Gaussian)
- What if we don’t know the law of the function we want to estimate?
- What if we augment the dimensionality of the data?

- Probability distribution functions of images are too much complex to be modelled with a Gaussian Mixture Model or other parametric specification of the estimated PDF.

Kullback-Leibler (KL) divergence

Is used to measure the difference between two probability distributions over the same variable.

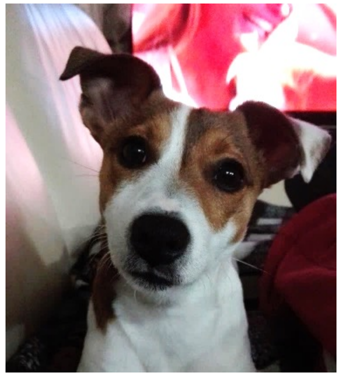
Specifically, the KL divergence of $q(x)$ from $p(x)$ is a measure of the information lost when $q(x)$ is used to approximate $p(x)$.

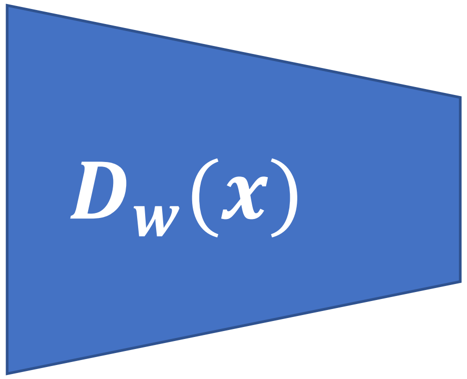
$$D_{KL}(p(x)||q(x)) = \int_{-\infty}^{\infty} p(x) \ln \frac{p(x)}{q(x)} dx$$

Generative vs. Discriminative models

Discriminative models

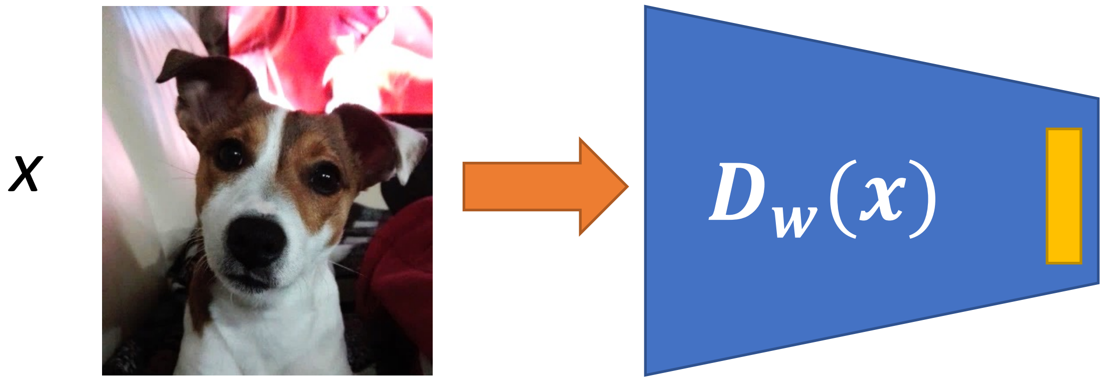
x





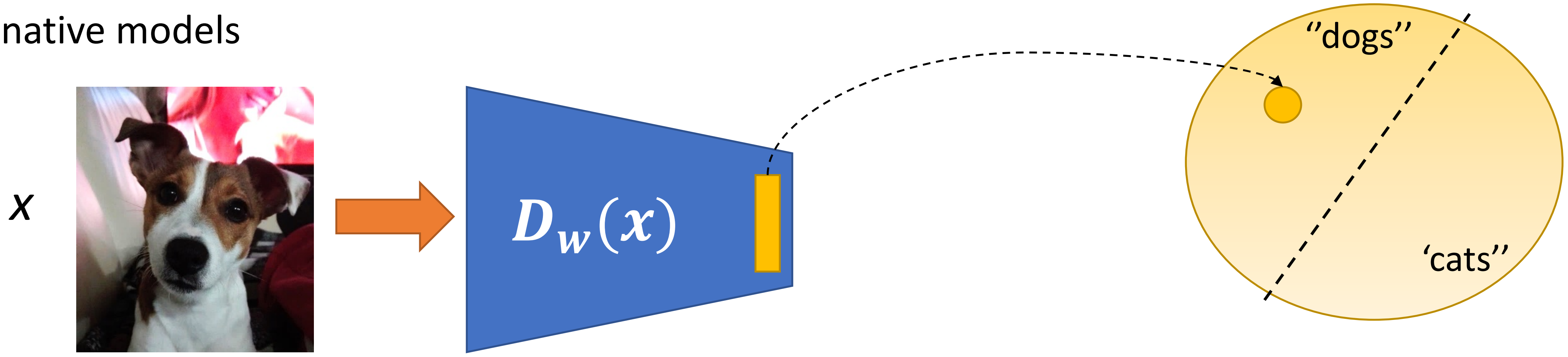
Generative vs. Discriminative models

Discriminative models



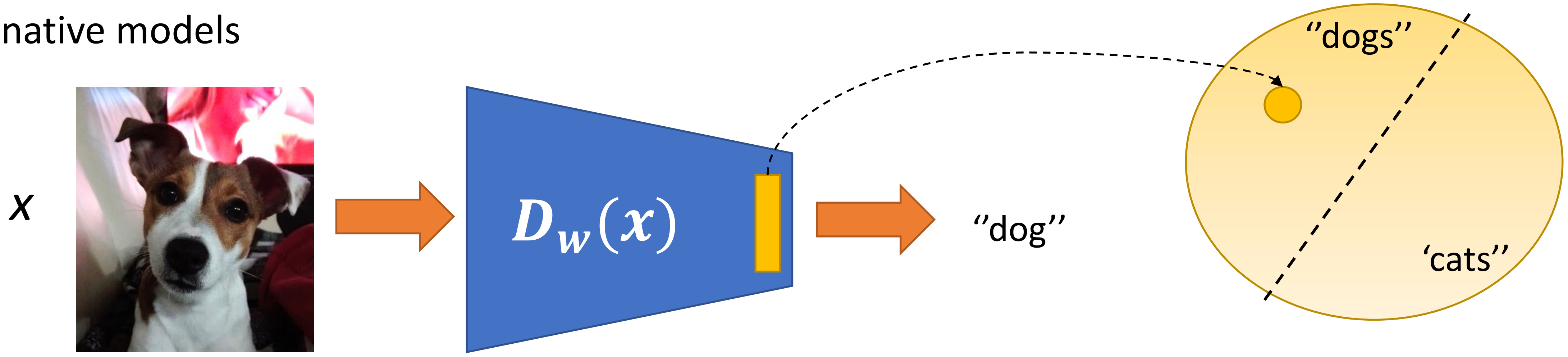
Generative vs. Discriminative models

Discriminative models



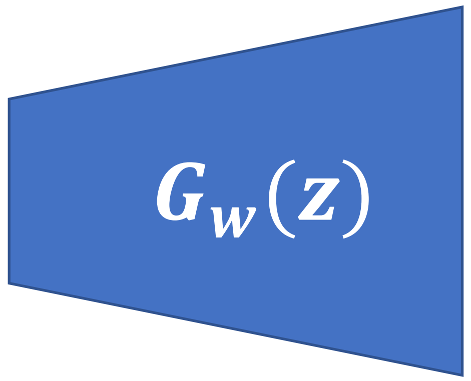
Generative vs. Discriminative models

Discriminative models



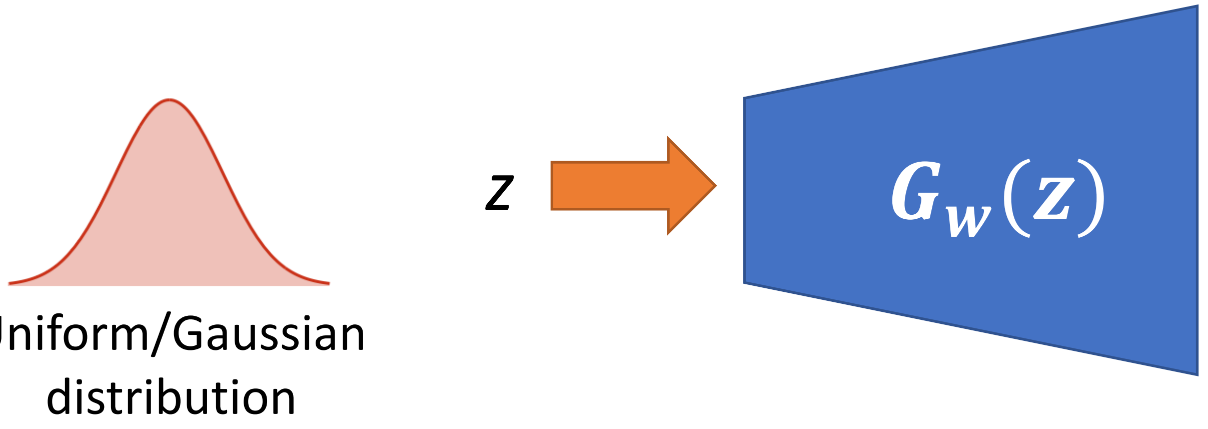
Generative vs. Discriminative models

Generative models



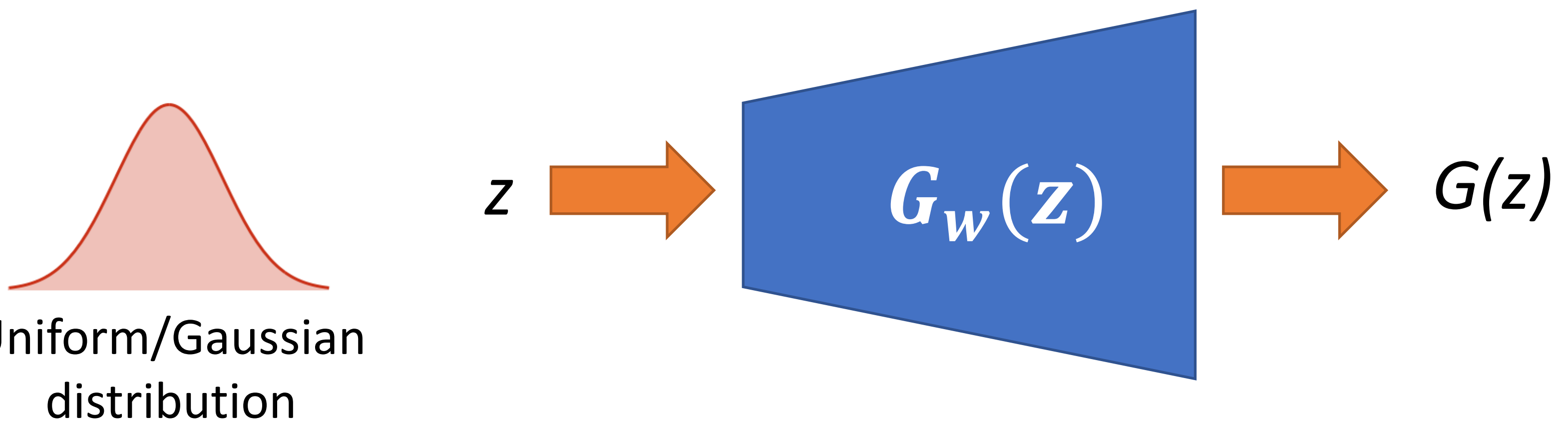
Generative vs. Discriminative models

Generative models



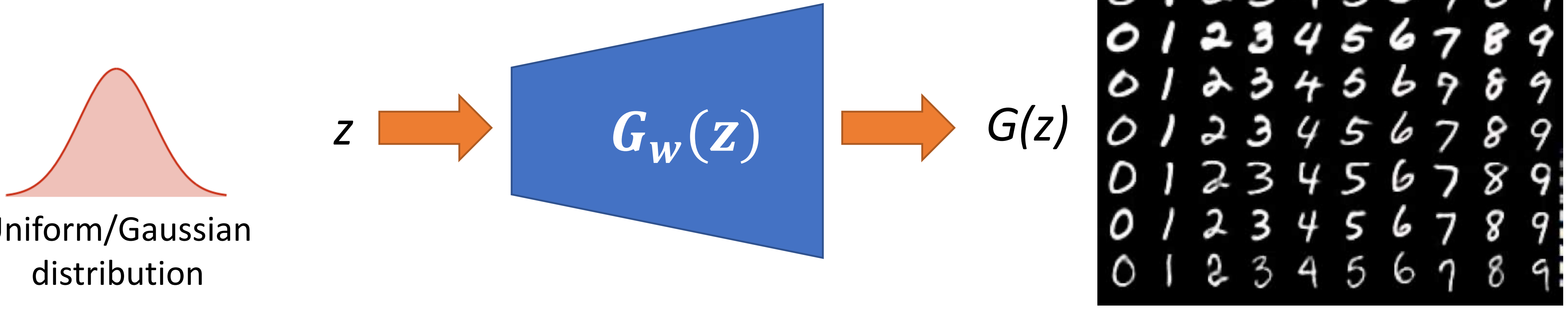
Generative vs. Discriminative models

Generative models



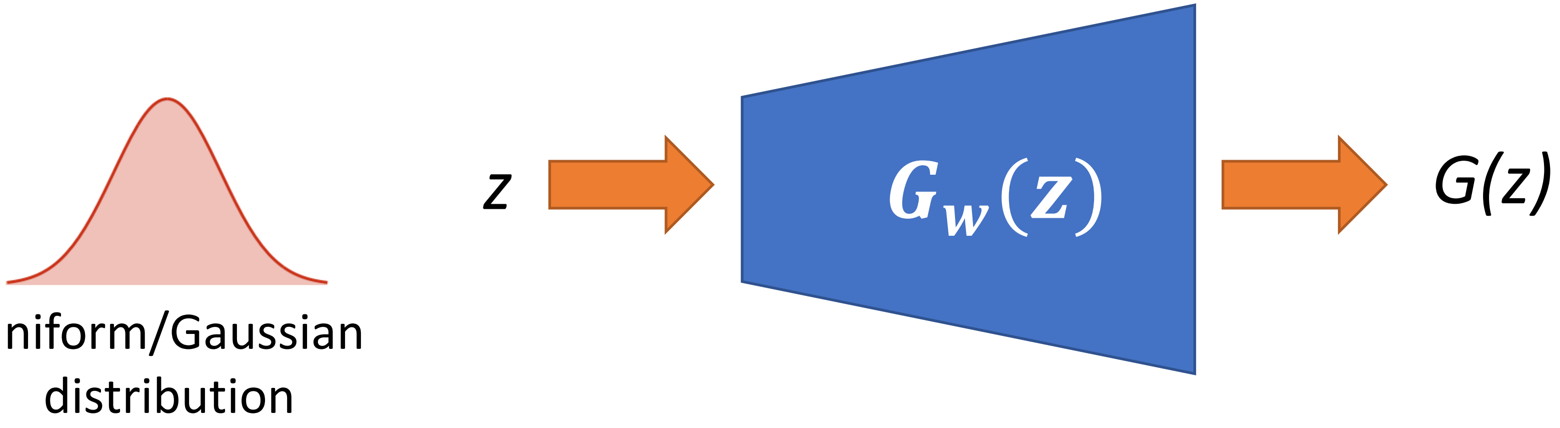
Generative vs. Discriminative models

Generative models



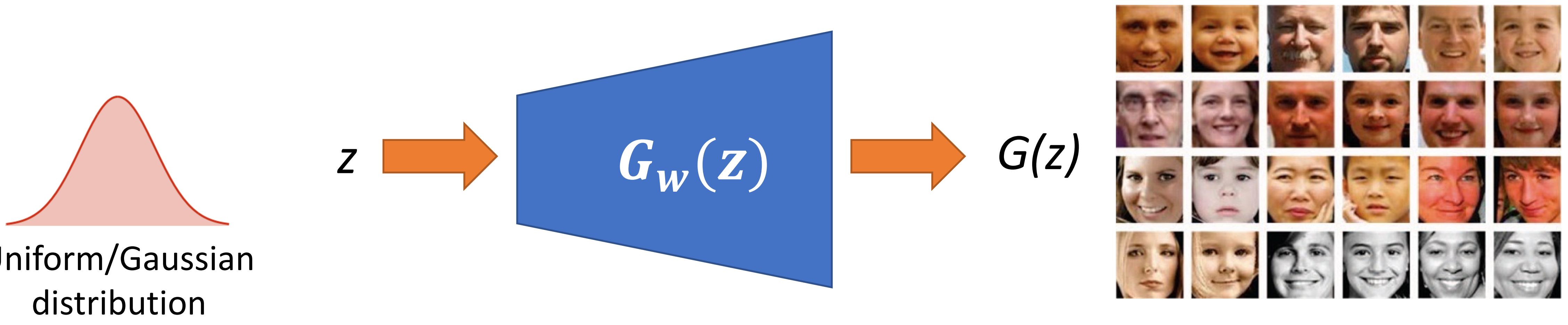
Generative vs. Discriminative models

Generative models



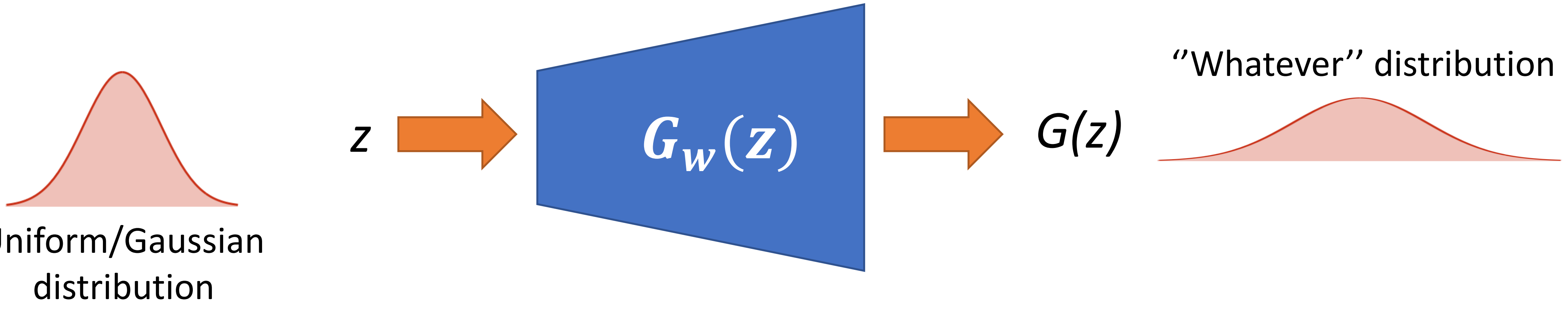
Generative vs. Discriminative models

Generative models



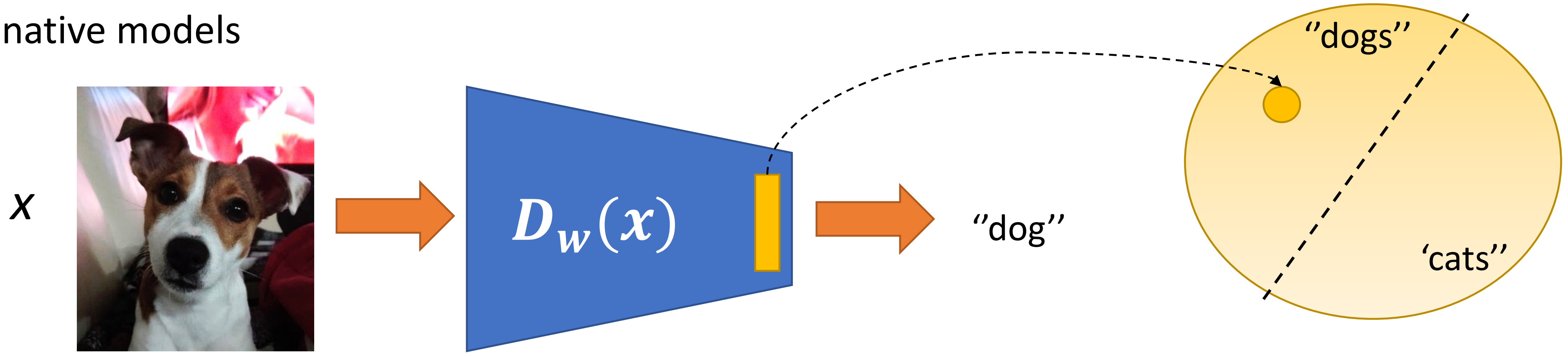
Generative vs. Discriminative models

Generative models

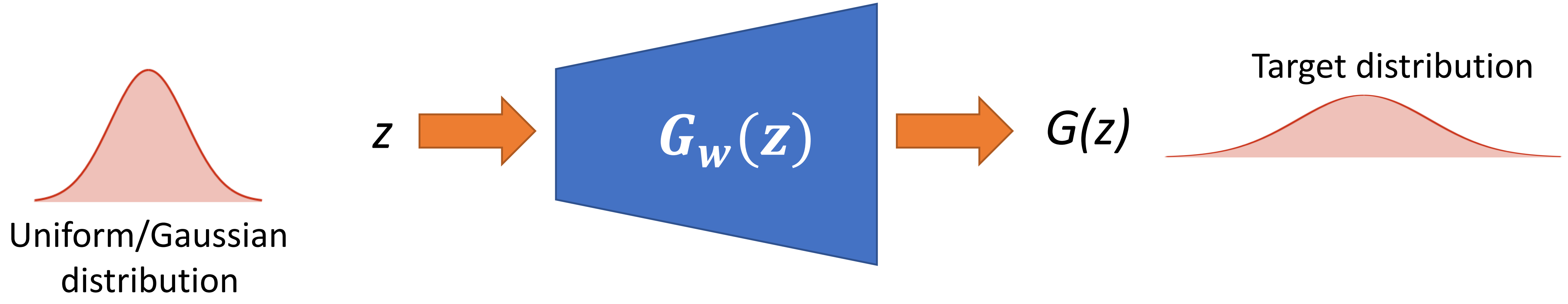


Generative vs. Discriminative models

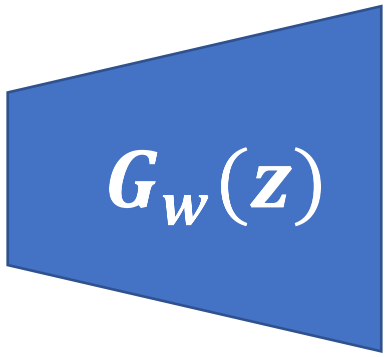
Discriminative models



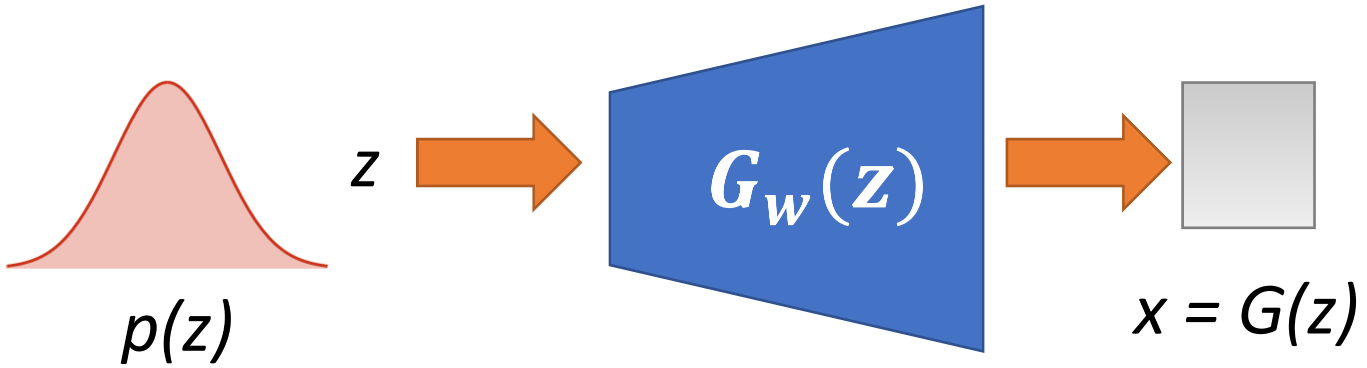
Generative models



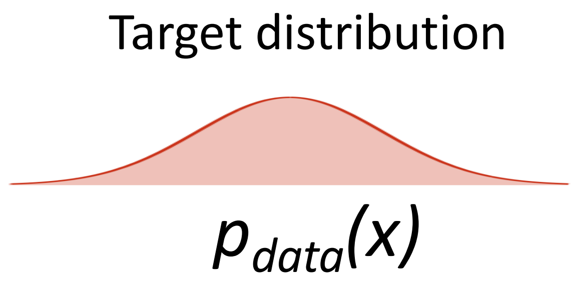
Generative Adversarial Networks

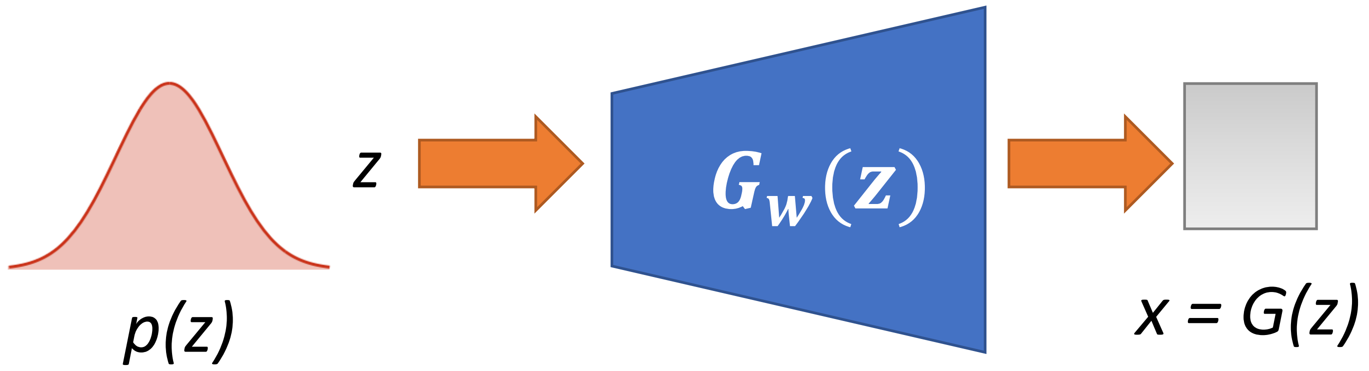


Generative Adversarial Networks

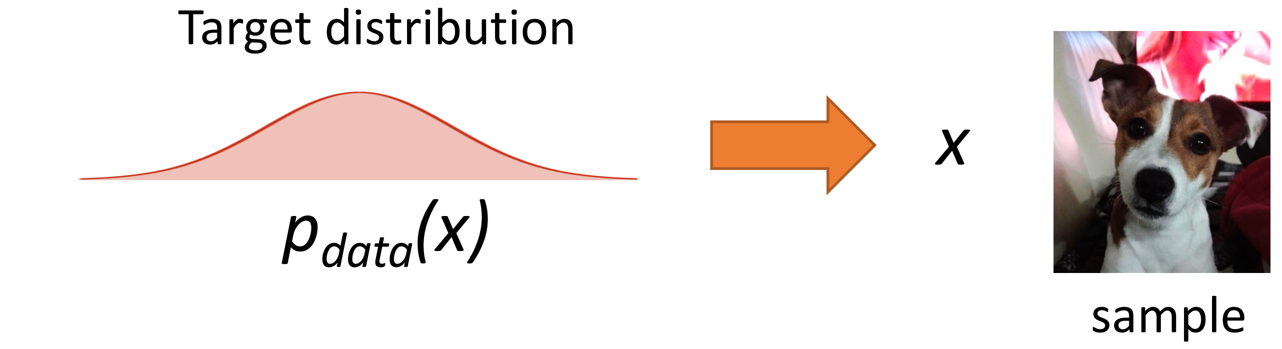


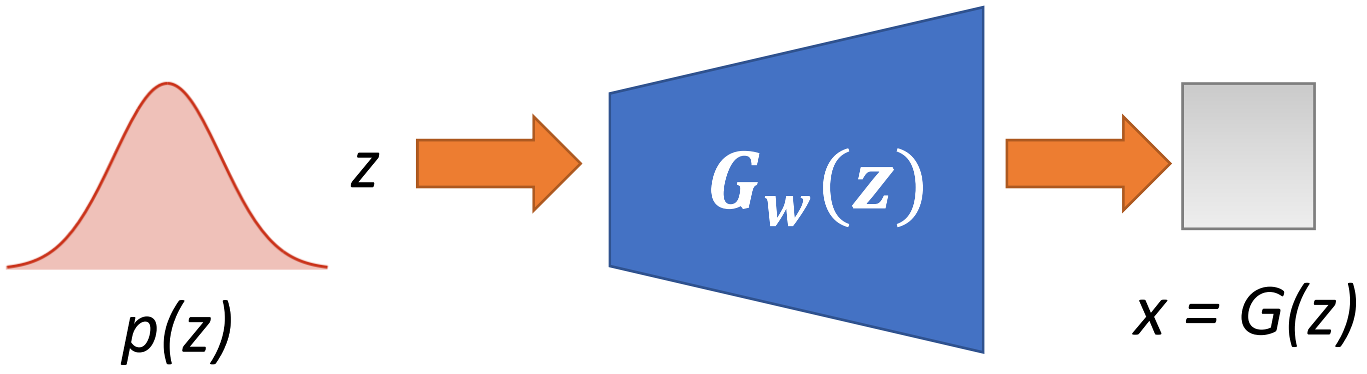
Generative Adversarial Networks



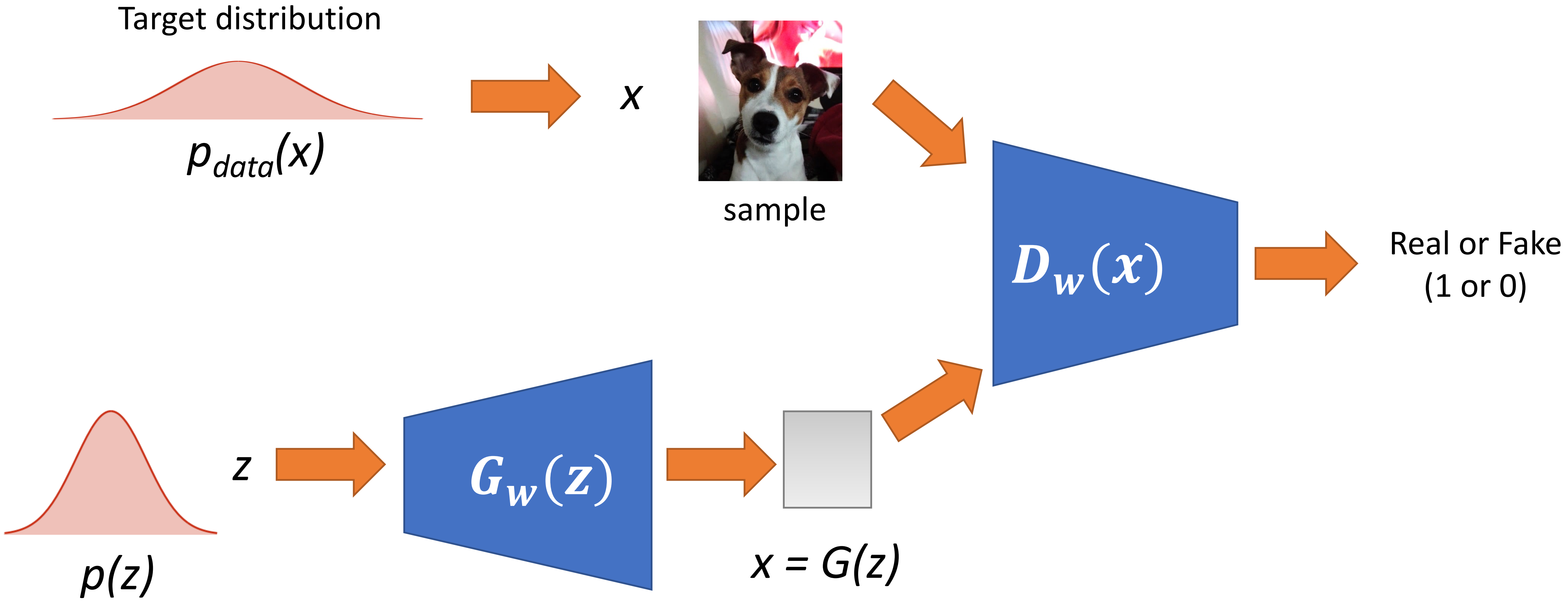


Generative Adversarial Networks

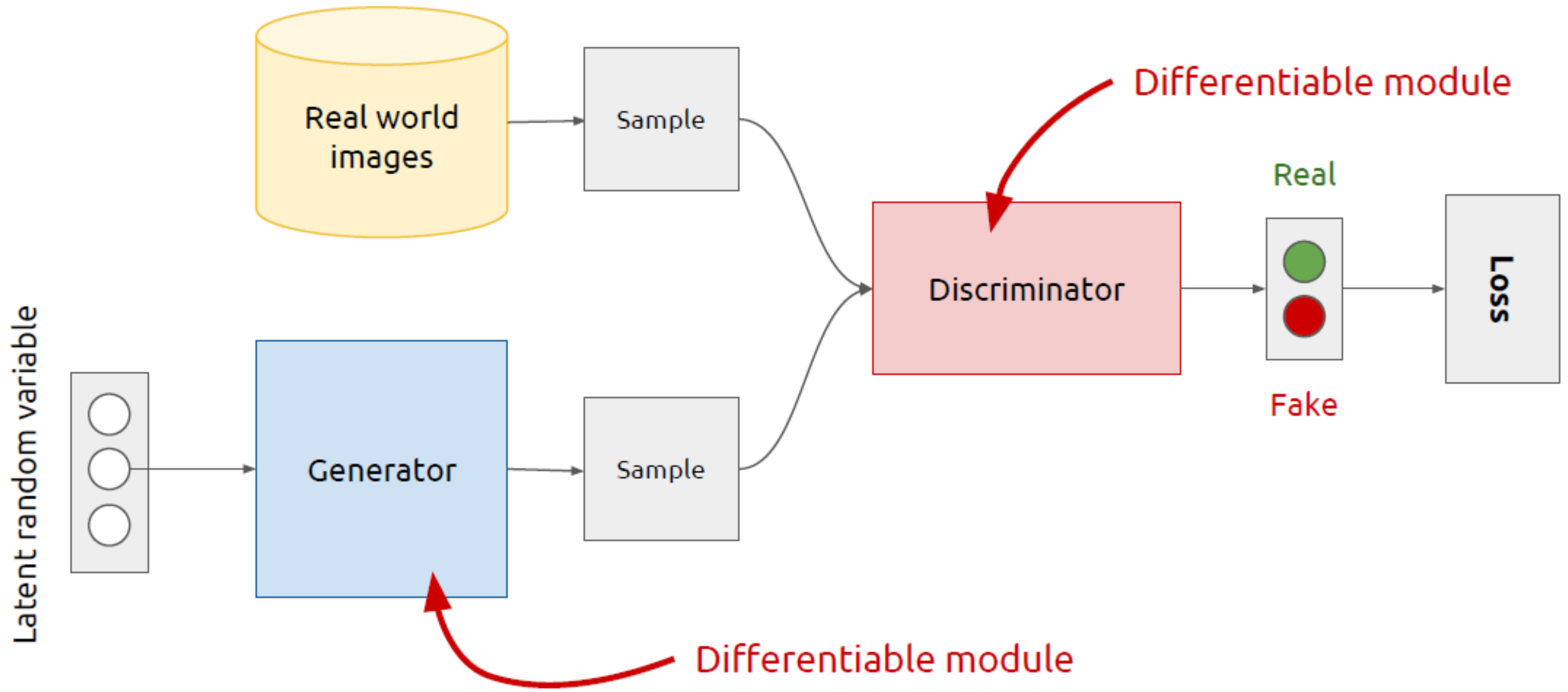




Generative Adversarial Networks

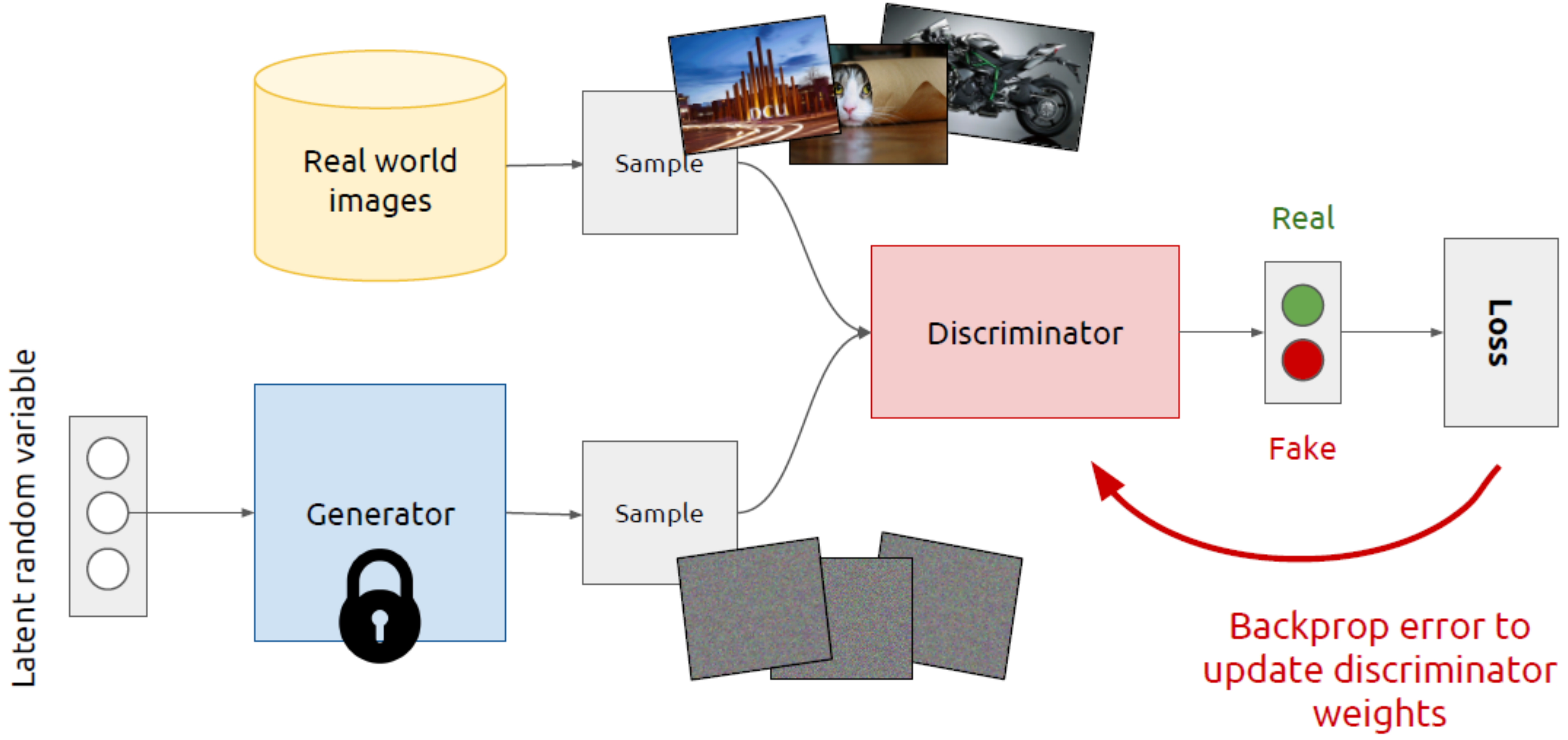


Generative Adversarial Networks



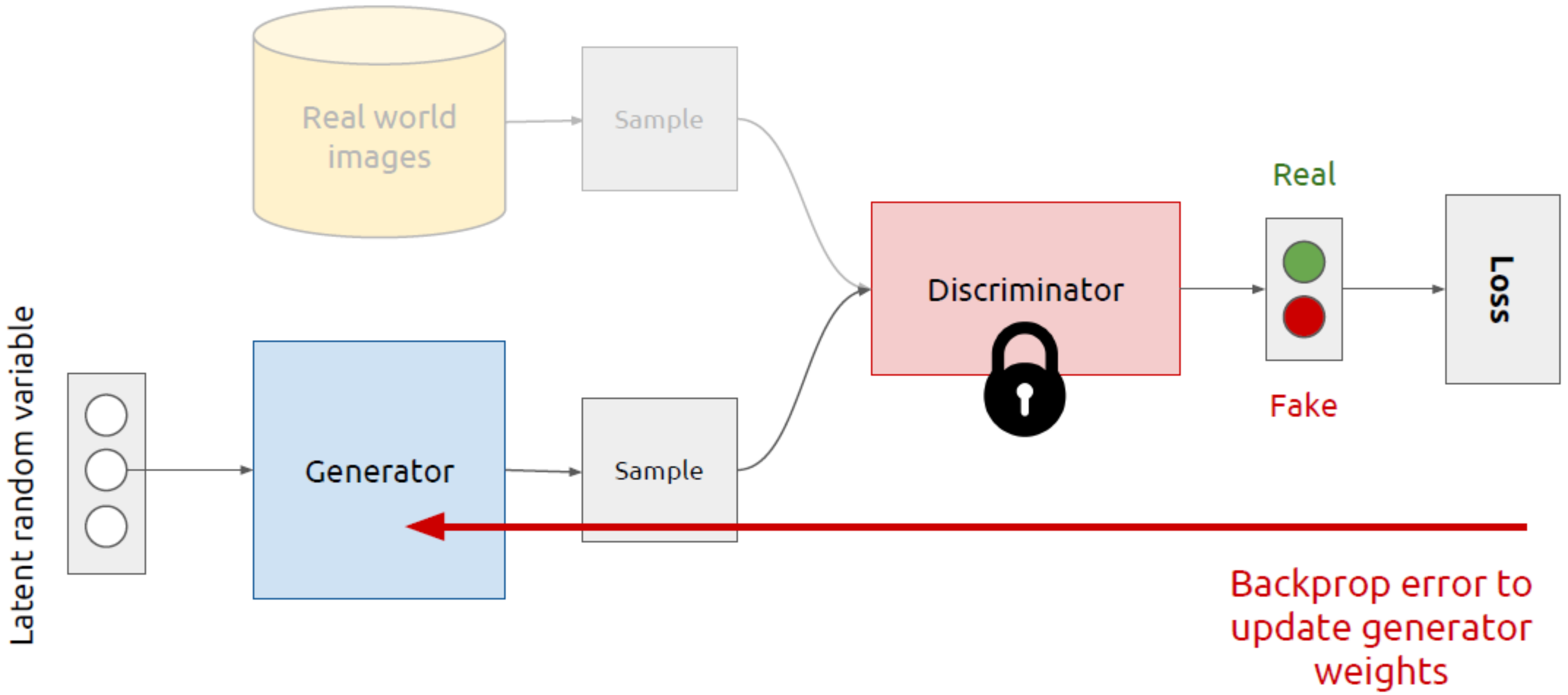
Credit: https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Generative Adversarial Networks



Credit: https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Generative Adversarial Networks



Credit: https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z; \theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x; \theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

p_g	generator's distribution
p_z	noise distribution (e.g, Gaussian, Uniform)
p_{data}	real data distribution
$G(z; \theta_g)$	generator function (i.e., $x = G(z; \theta_g)$)
$D(x; \theta_d)$	discriminator function

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z; \theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x; \theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

- **Train *D* to maximize** the probability of assigning the correct label to both training examples and samples from G
- Simultaneously **train *G* to minimize** the distance between the two distributions

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z; \theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x; \theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

- **Train *D* to maximize** the probability of assigning the correct label to both training examples and samples from G
- Simultaneously **train *G* to minimize** the distance between the two distributions
- *D* and *G* play the following minmax game

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

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z;\theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x;\theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

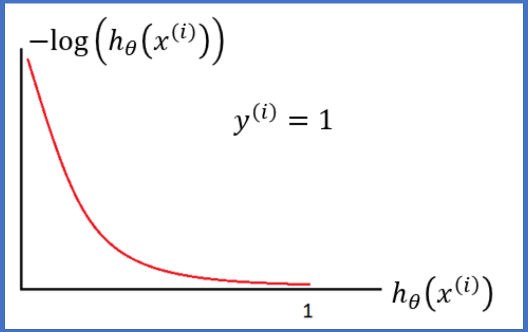
- **Train *D* to maximize** the probability of assigning the correct label to both training examples and samples from G
- Simultaneously **train *G* to minimize** the dista
- *D* and *G* play the following minmax game

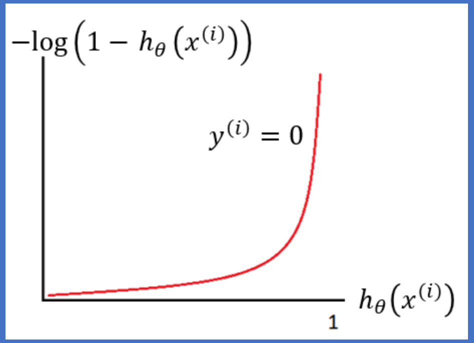
Do you remember the BCE loss?

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

Generativ

- The aim is to
- Input noise d
- The generato a differentiable function repr
- A second ML the probability that x came f





$$J(\theta) = -\frac{1}{m}\sum_i\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + (1-y^{(i)})\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

- **Train *D* to maximize** the probability of assigning the correct label both training examples and samples from G
- Simultaneously **train *G* to minimize** the dista
- *D* and *G* play the following minmax game

Do you remember the BCE loss?

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

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z; \theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x; \theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

- **Train *D* to maximize** the probability of assigning the correct label to both training examples and samples from G
- Simultaneously **train *G* to minimize** the distance between the two distributions
- *D* and *G* play the following minmax game

$$\min_{G} \max_{\underline{D}} V(D, G) = \underbrace{\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{Recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{Recognize generated images better}}.$$

Generative Adversarial Networks

- The aim is to learn the generator's distribution p_g over data x
- Input noise data distribution p_z (e.g, Gaussian, Uniform)
- The generator learns a mapping function $G(z; \theta_g)$, where G is a differentiable function represented by a MLP with parameters θ_g
- A second MLP $D(x; \theta_d)$ outputs a single scalar $D(x)$ representing the probability that x came from the data distribution p_{data} rather than p_g

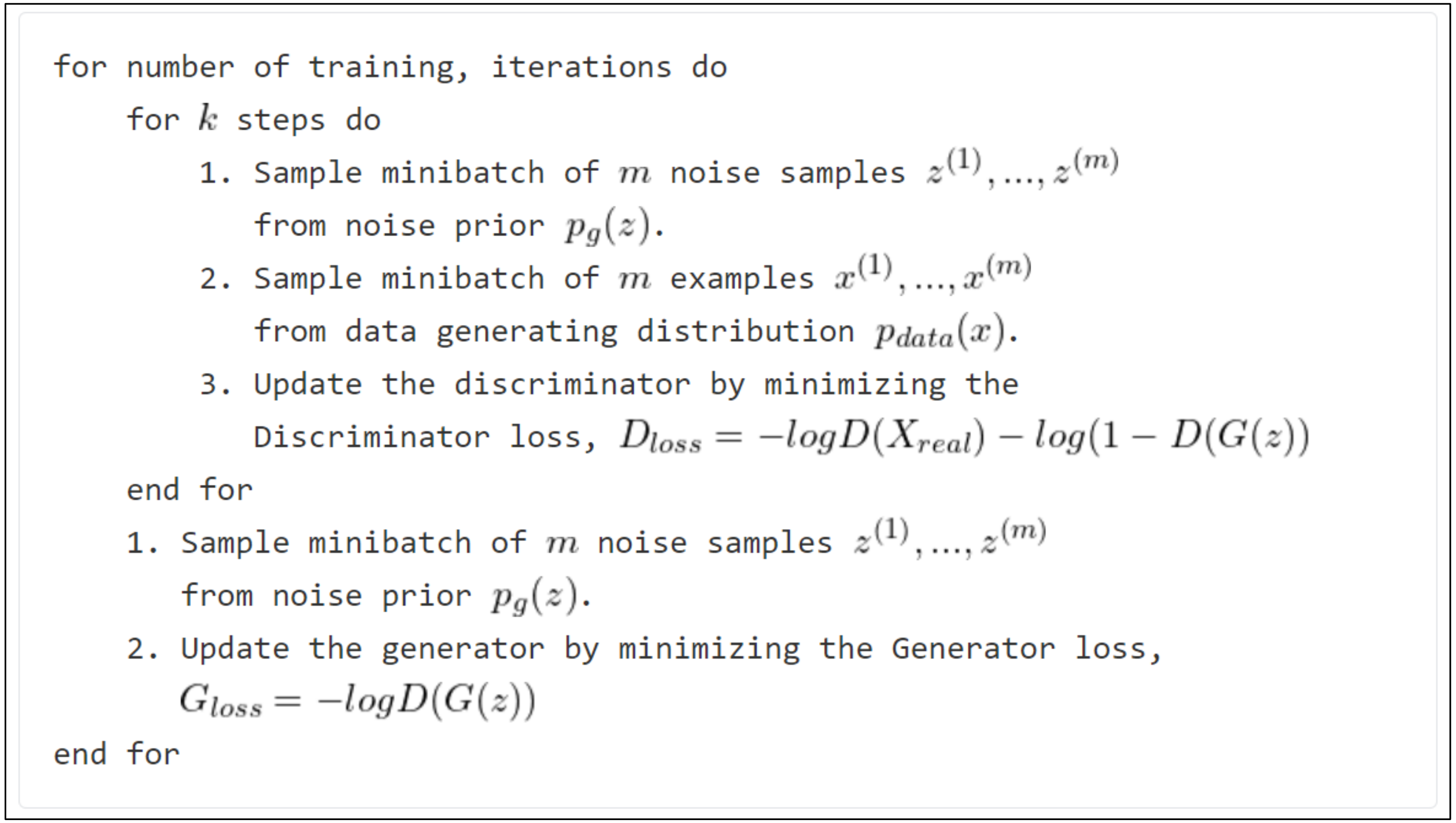
- **Train *D* to maximize** the probability of assigning the correct label to both training examples and samples from G
- Simultaneously **train *G* to minimize** the distance between the two distributions
- *D* and *G* play the following minmax game

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

Optimize G that can fool the discriminator the most

Generative Adversarial Networks

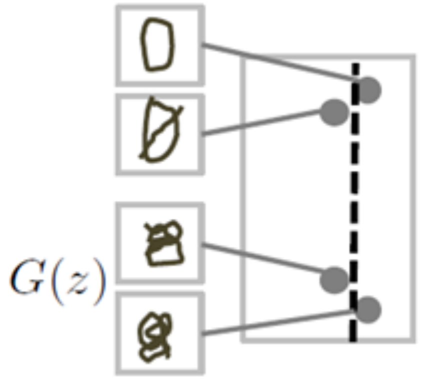
Training procedure



```
for number of training, iterations do
    for $k$ steps do
        1. Sample minibatch of $m$ noise samples $z^{(1)}, ..., z^{(m)}$
           from noise prior $p_g(z)$.
        2. Sample minibatch of $m$ examples $x^{(1)}, ..., x^{(m)}$
           from data generating distribution $p_{data}(x)$.
        3. Update the discriminator by minimizing the
           Discriminator loss, $D_{loss} = -logD(X_{real}) - log(1 - D(G(z))$
    end for
    1. Sample minibatch of $m$ noise samples $z^{(1)}, ..., z^{(m)}$
       from noise prior $p_g(z)$.
    2. Update the generator by minimizing the Generator loss,
       $G_{loss} = -logD(G(z))$
end for
```

Two-player minmax discrimination game

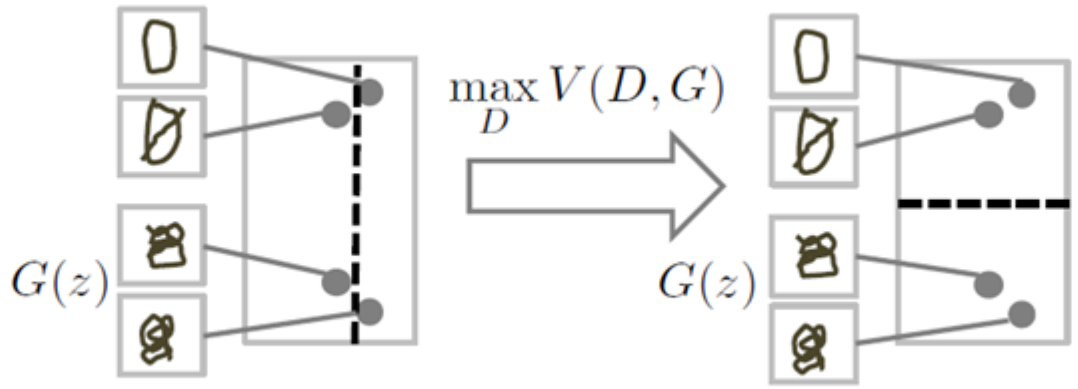
$$\min_G \max_D V(D, G)$$



https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks?qid=21ec97ec-ba8c-4edf-bb4e-9dfc2cc62db9&v=&b=&from_search=1

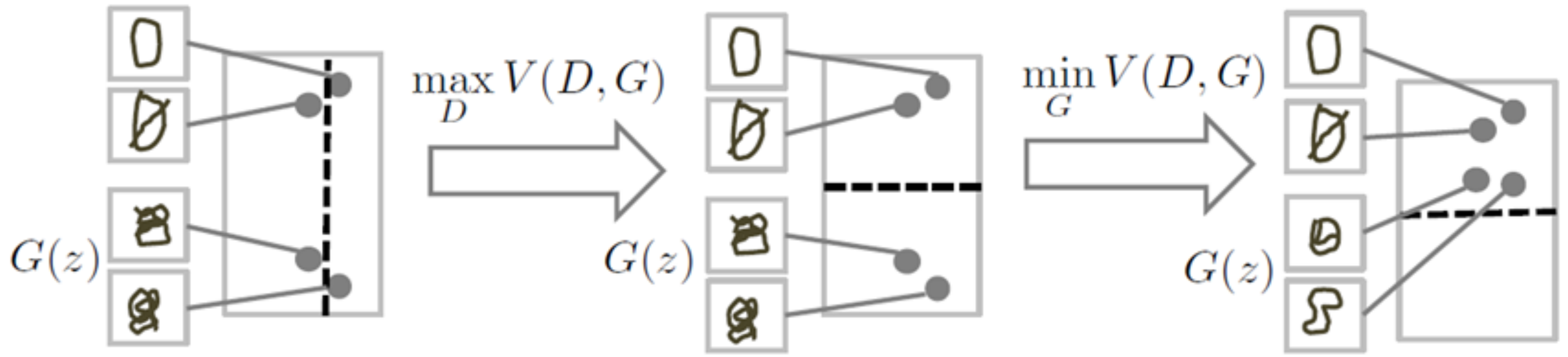
Two-player minmax discrimination game

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



Two-player minmax discrimination game

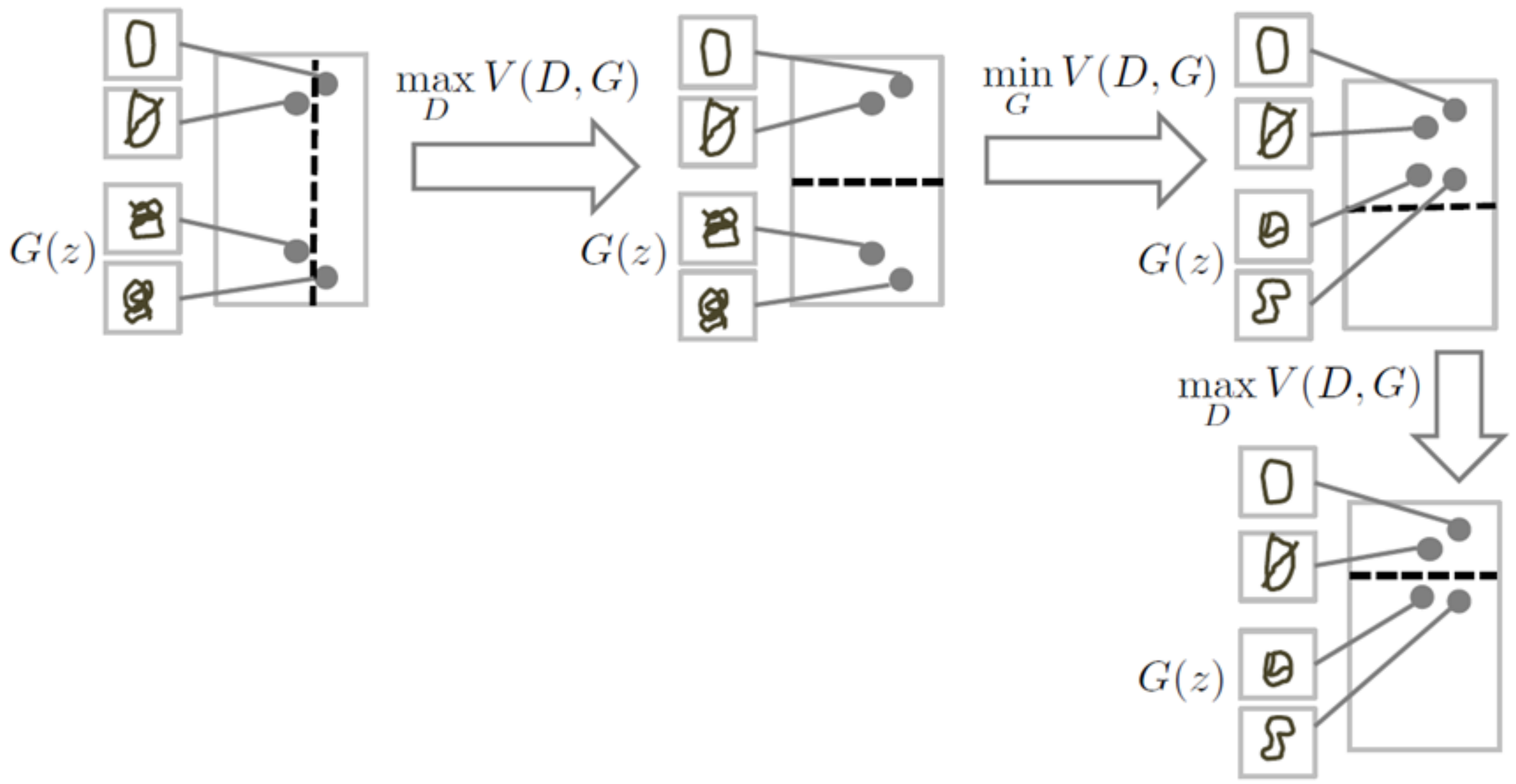
$$\min_G \max_D V(D, G)$$



https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks?qid=21ec97ec-ba8c-4edf-bb4e-9dfc2cc62db9&v=&b=&from_search=1

Two-player minmax discrimination game

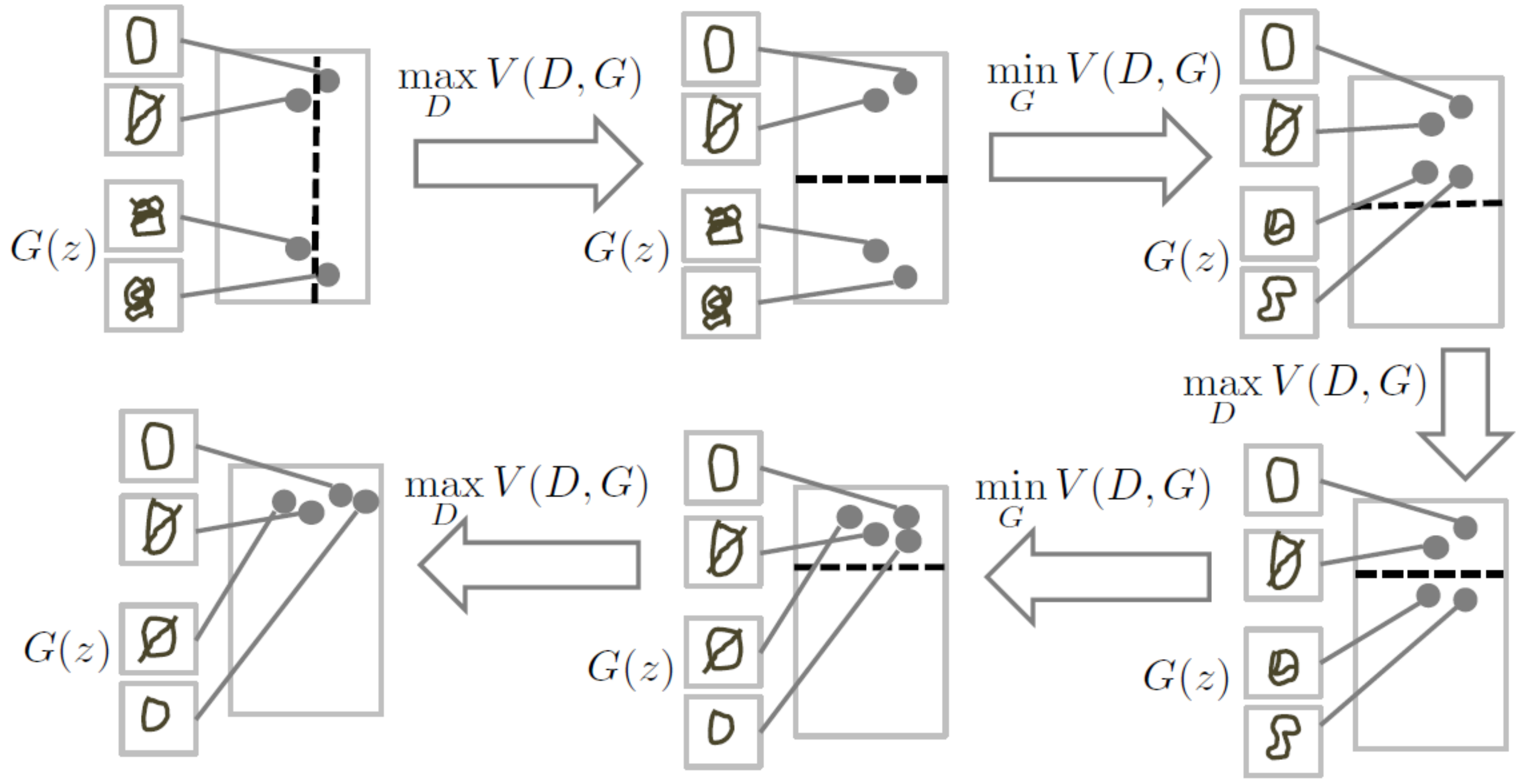
$$\min_{G} \max_{D} V(D, G)$$



https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks?qid=21ec97ec-ba8c-4edf-bb4e-9dfc2cc62db9&v=&b=&from_search=1

Two-player minmax discrimination game

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



https://www.slideshare.net/ckmarkohchang/generative-adversarial-networks?qid=21ec97ec-ba8c-4edf-bb4e-9dfc2cc62db9&v=&b=&from_search=1

Training Generative Adversarial Networks

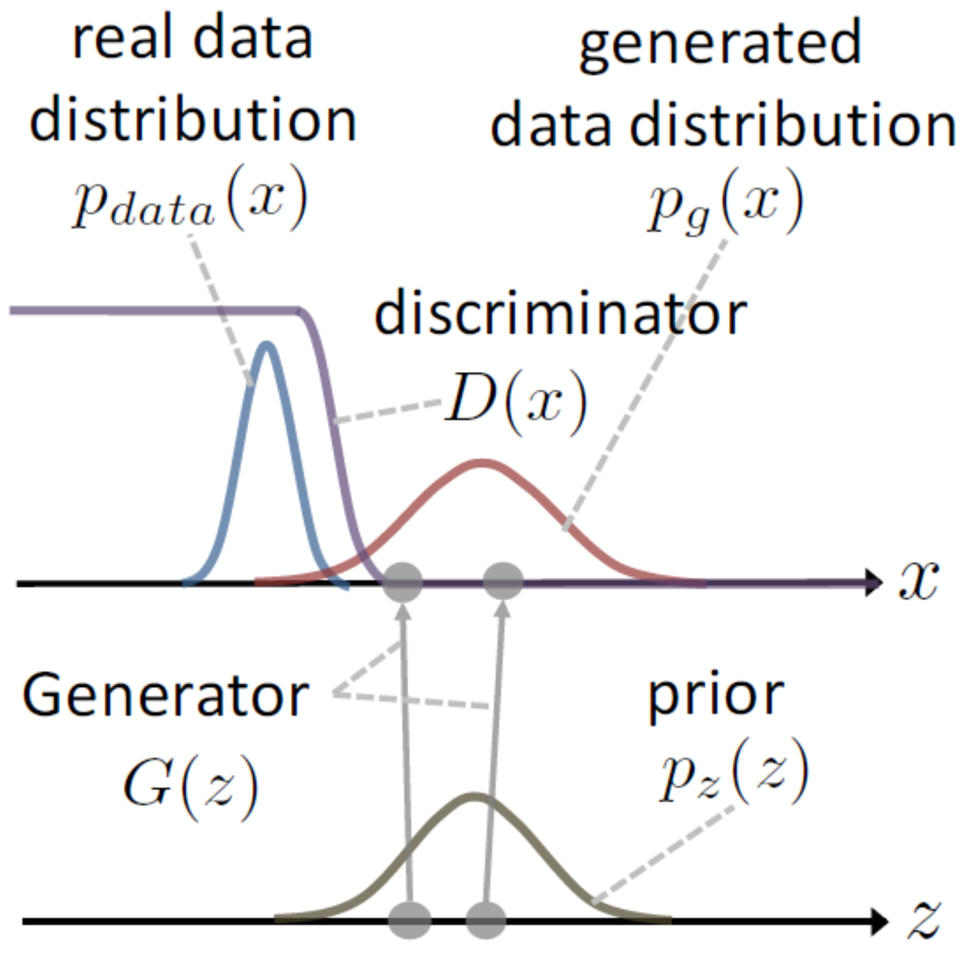
$$\min_G \max_D V(D, G)$$

Iterate the training of generator and discriminator until convergence

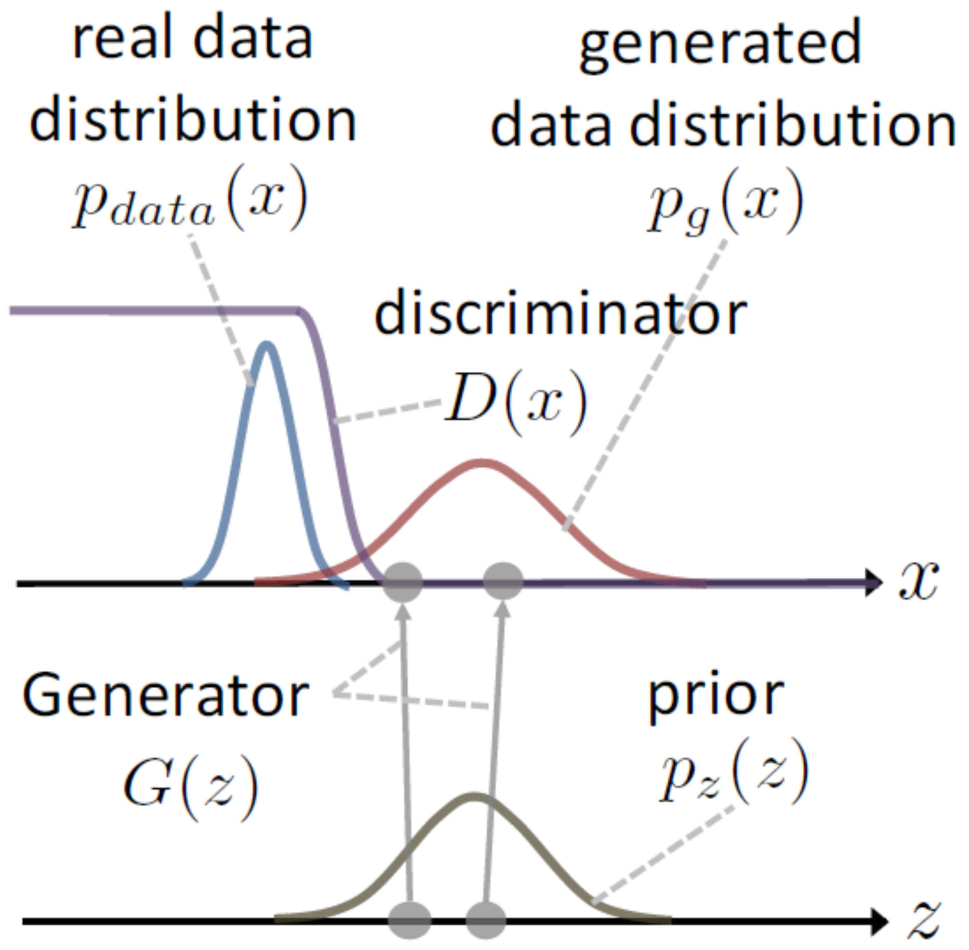
- ***Updating the discriminator*** should make it better at discriminating between real images and generated ones
- ***Updating generator*** makes it better at fooling the current discriminator

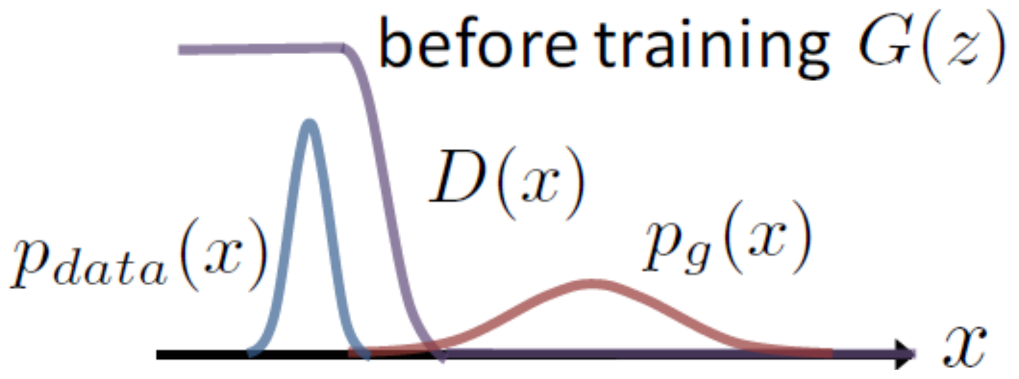
If the generator gets so good that it is impossible for the discriminator to tell the difference between real and generated images, the discriminator accuracy would be 0.5

Training Generative Adversarial Networks

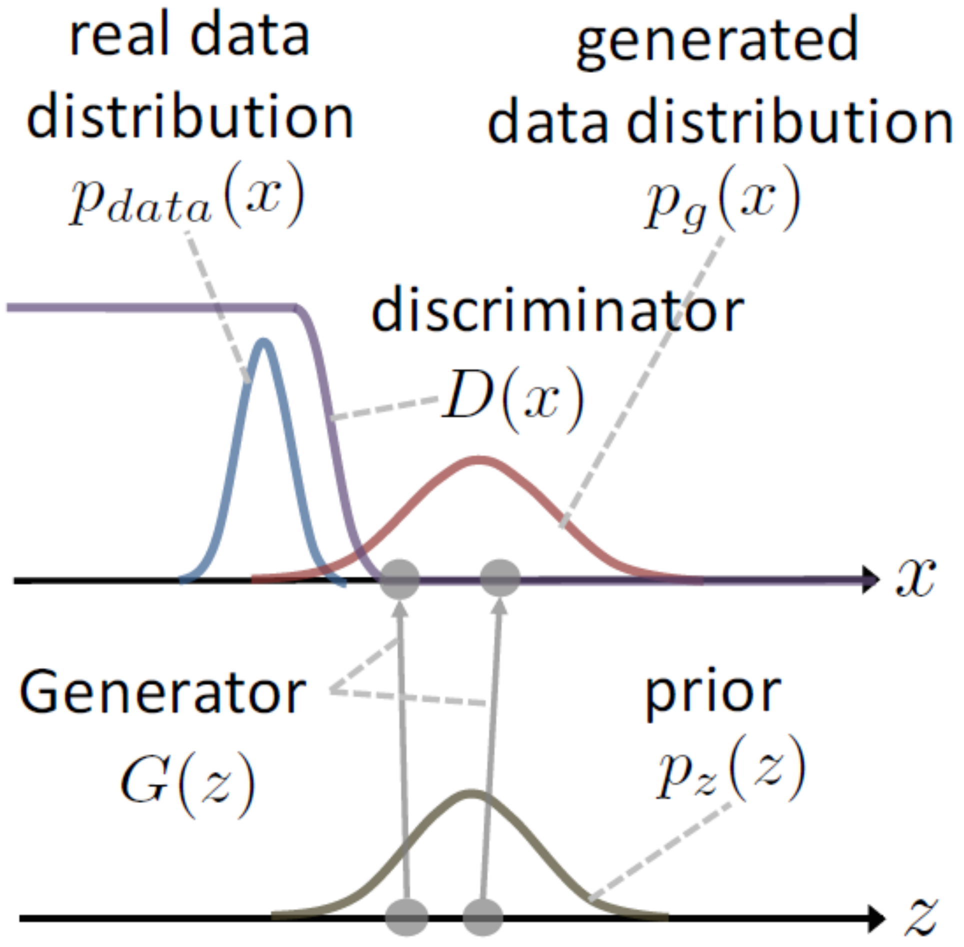


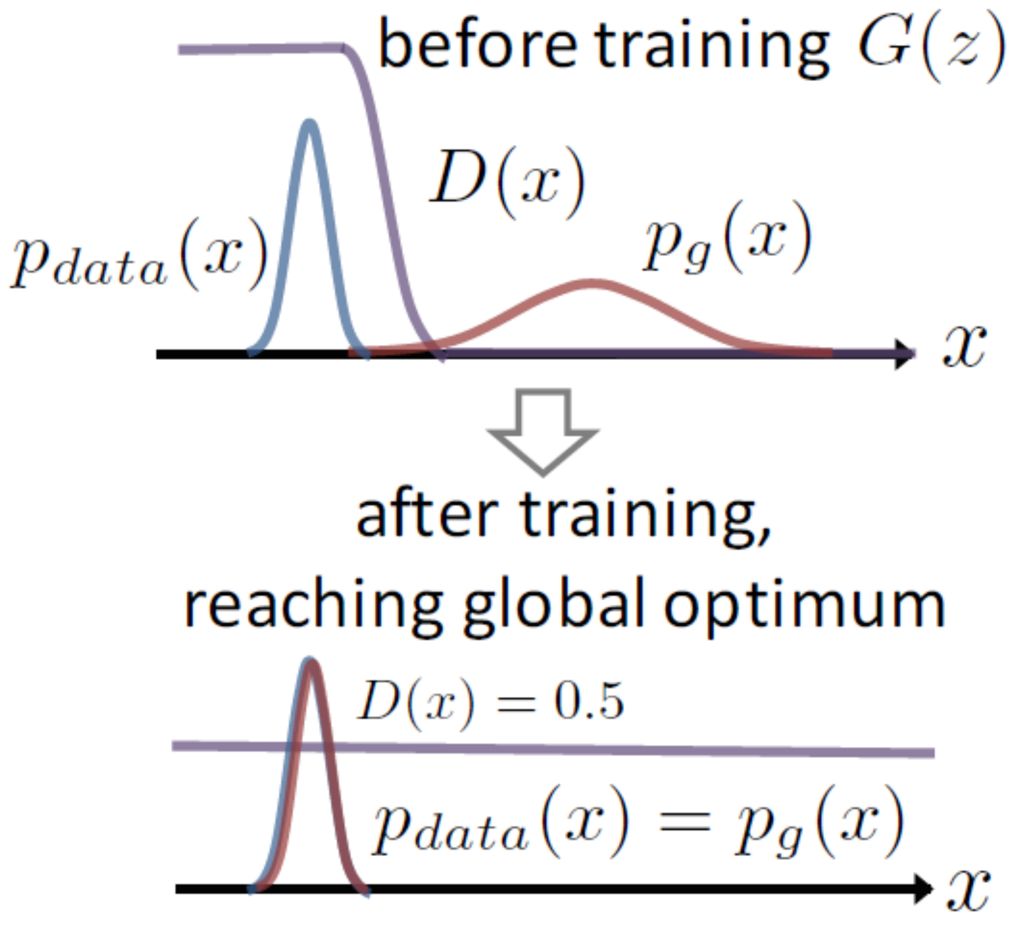
Training Generative Adversarial Networks



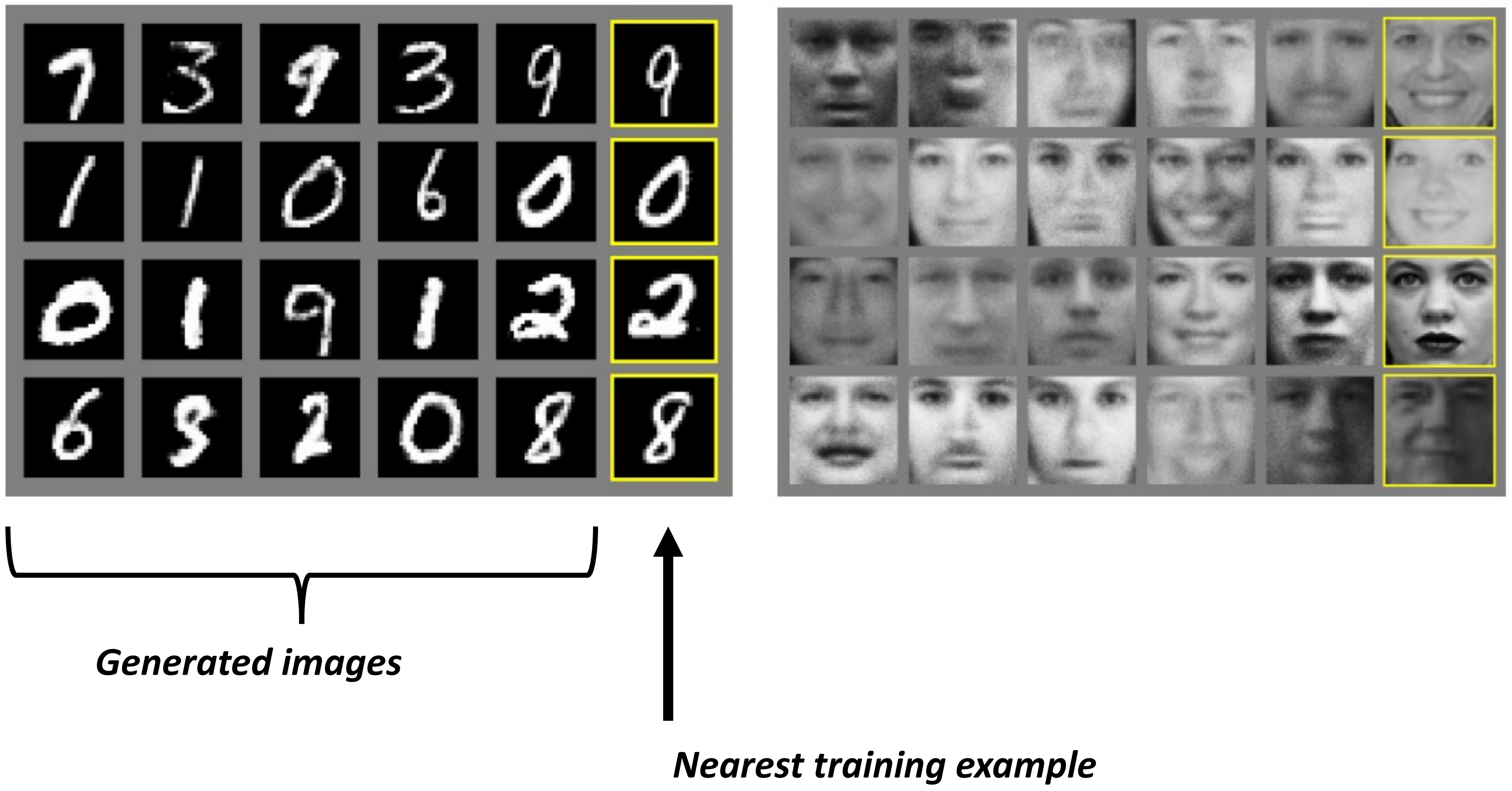


Training Generative Adversarial Networks





Examples



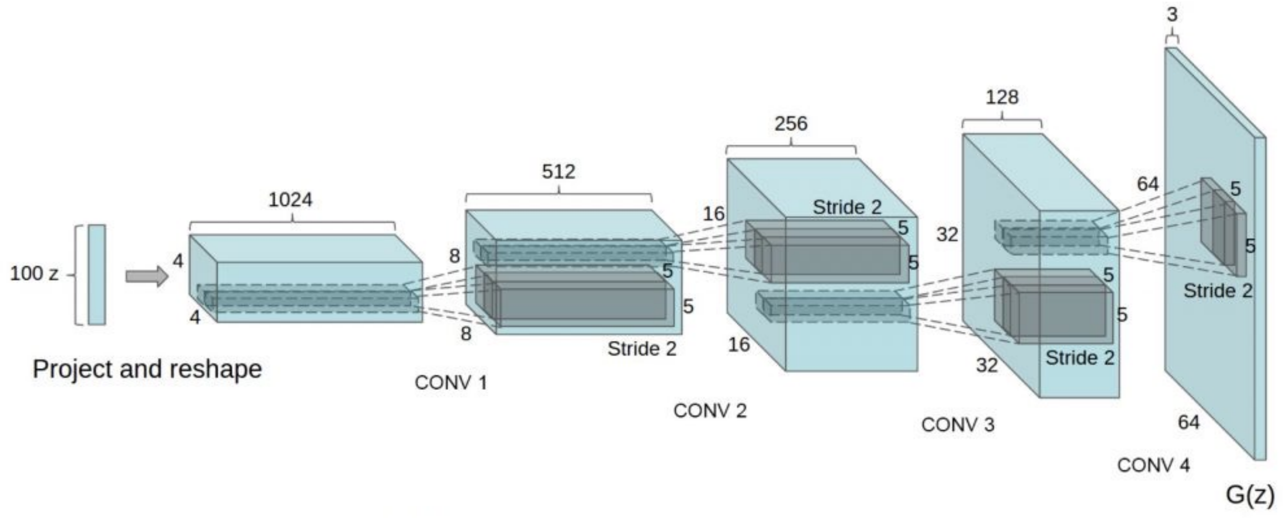
Break time!

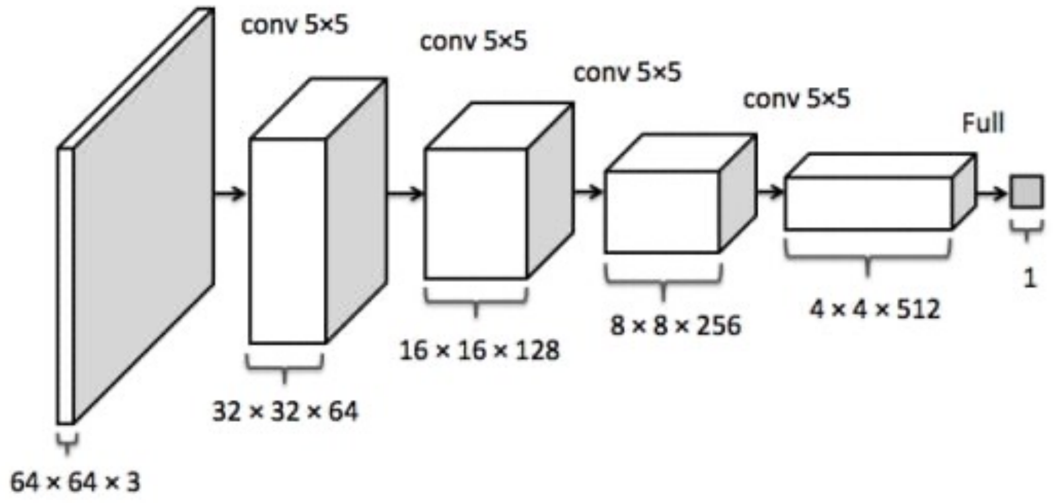
DCGAN – Deep Convolutional GAN

GANs have always used Dense Layers in the discriminator. However, in 2015 came Deep Convolutional GAN (DCGAN), which demonstrated that *convolutional layers work better than fully-connected layers* in GAN.

DCGAN generated higher quality images by

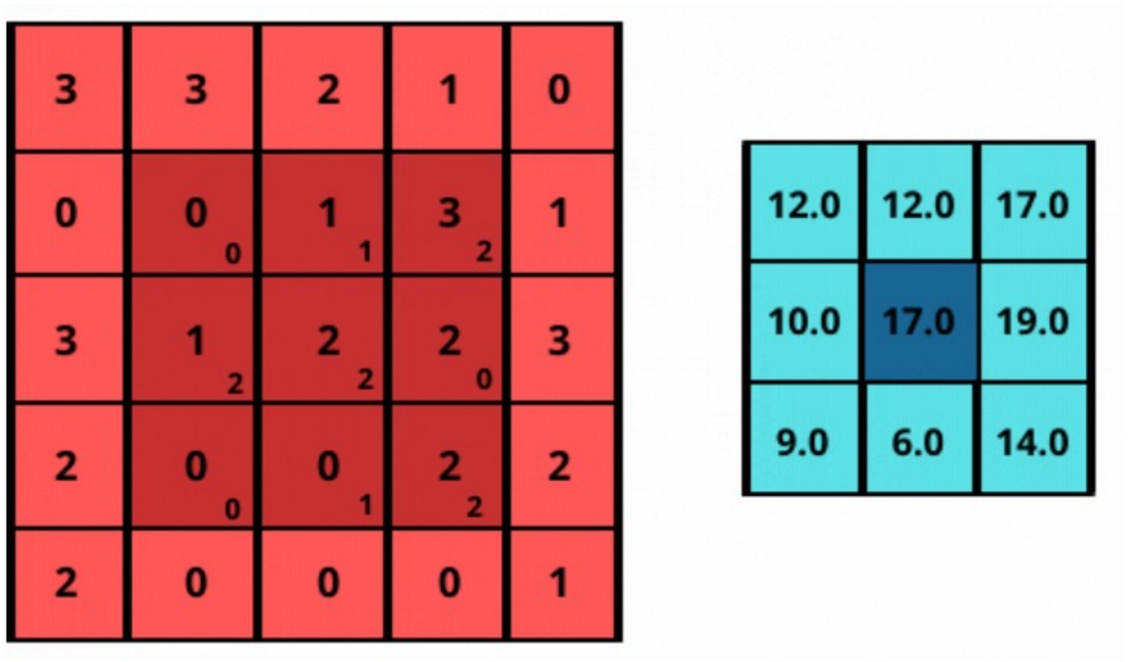
- Using ***strided convolutional layers*** in the discriminator to downsample the images.
- Using ***fractionally-strided convolutional layers*** to upsample the images.





Source: UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS https://arxiv.org/pdf/1511.06434.pdf

DCGAN – Deep Convolutional GAN

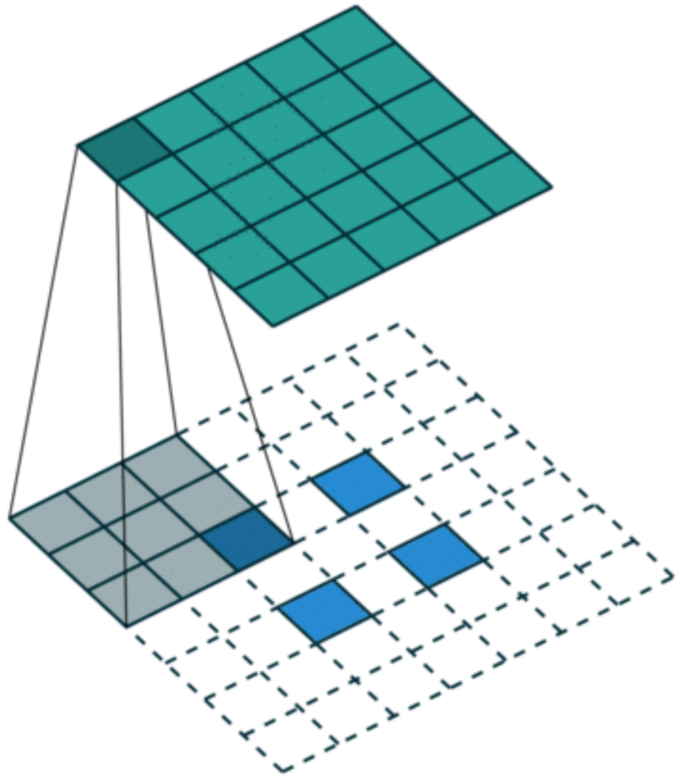


In DCGAN, the authors used a Stride of 2, meaning the filter slides through the image, moving 2 pixels per step.

The authors eliminated max-pooling, which is generally used for downsampling an image. Instead, they adopted strided convolution, with a stride of 2, to downsample the image in Discriminator.

Max-pooling has no learnable parameters. Strided convolution generally allows the network to ***learn its own spatial downsampling***.

DCGAN – Deep Convolutional GAN



Fractionally-strided convolution (or transposed convolution), is the opposite of a convolution operation.

In a convolution operation the input is processed by a kernel, producing a smaller output. In a fractionally-strided operation, an upsampled (larger) output is obtained from a smaller input.

In this figure, a 2 x 2 input matrix is upsampled to a 5 x 5 matrix.

Traditional interpolation techniques can do this upsampling, but ***they lack learnable parameters***. In this case it cannot be trained on your data.
In DCGAN, the authors used a series of four fractionally-strided convolutions to upsample the 100-dimensional input, into a 64 × 64 pixel image in the Generator.

Source: https://github.com/vdumoulin/conv_arithmetic

Conditional GAN (cGAN)

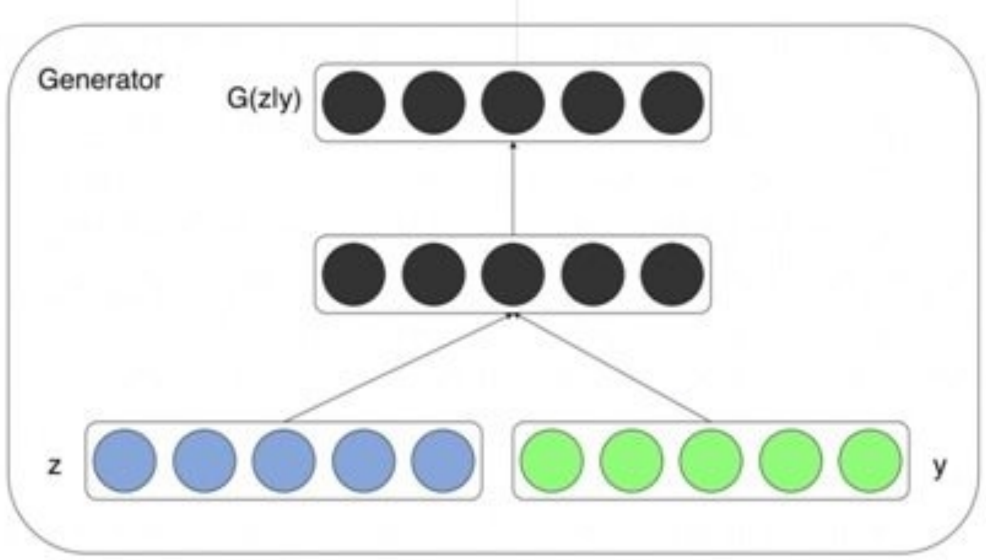
Though the GAN model is capable of generating new realistic samples for a particular distribution (i.e., dataset), ***we have zero control*** over the type of images that are generated.

But what if we want our GAN model to generate only dog images, and not random ones containing cats, cars or other random subjects included in the training dataset?

Conditional GAN (cGAN)

Generator

Ordinarily, the generator needs a noise vector to generate a sample. In a *conditional generation* however, it also *needs auxiliary information* that specifically tells the generator which particular class sample to produce (i.e., a conditioning label).



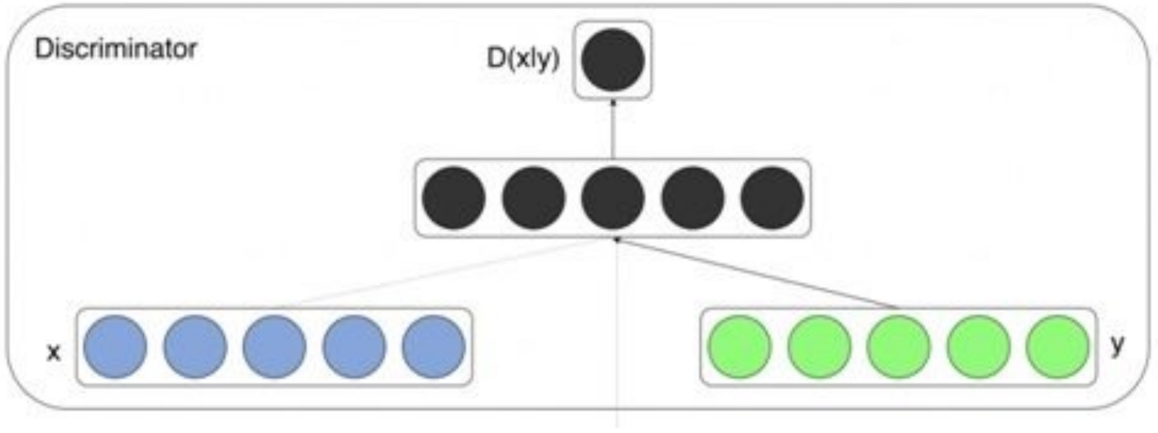
Let's call the ***conditioning label y.*** The Generator uses the noise vector z and the label y to synthesize a fake example G(z, y) = x|y (x conditioned on y, where x is the generated fake example). This fake example aims to fool the discriminator by looking as similar as possible to a real example for the given label.

Once the Generator is fully trained, you can specify what example you want the Conditional Generator to now produce, by simply passing it the desired label.

Conditional GAN (cGAN)

Discriminator

The Discriminator is fed ***both real and fake examples with labels***. It learns to not just recognize real data from fake, but also matching pairs.



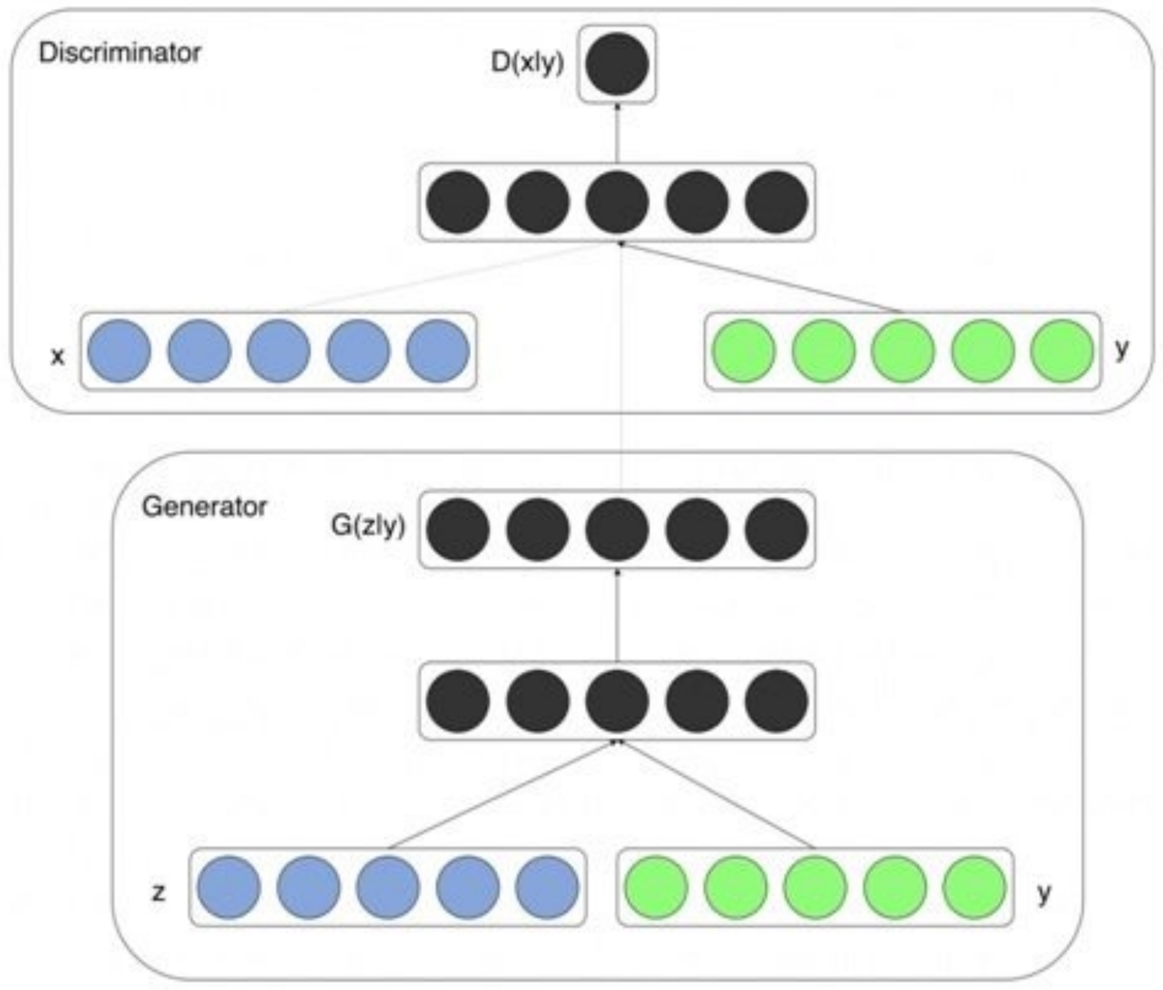
A pair is matching when the image has a correct label assigned to it. The Discriminator finally outputs a probability indicating the input is real or fake. Its goal is to learn to:

- Accept all real sample label pairs.
- Reject all image-label pairs in which ***the image is fake***, even if the label matches the image.
- Reject all samples in which the corresponding ***labels do not match***, regardless of the example being real or fake.

Conditional GAN (cGAN)

Training

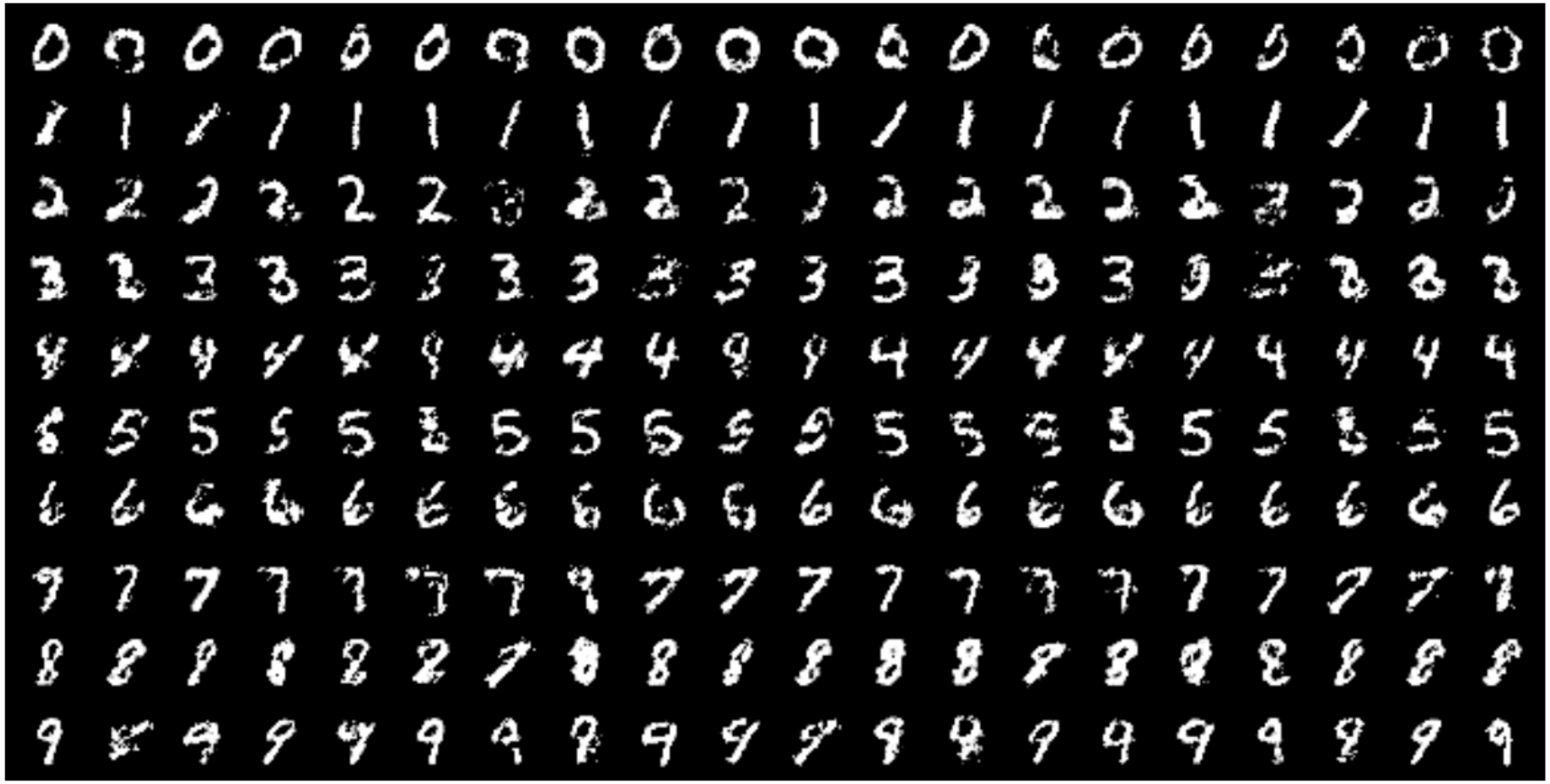
- Condition the GAN with some vector y.
- See the networks with a probabilistic point of view.
- Generator G(z, y)
  - modeling the distribution of data given z and y
- Discriminator D(x, y)
  - Labels of X and X_G are modeled with d ~ D(d|X, y)



Objective function:

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

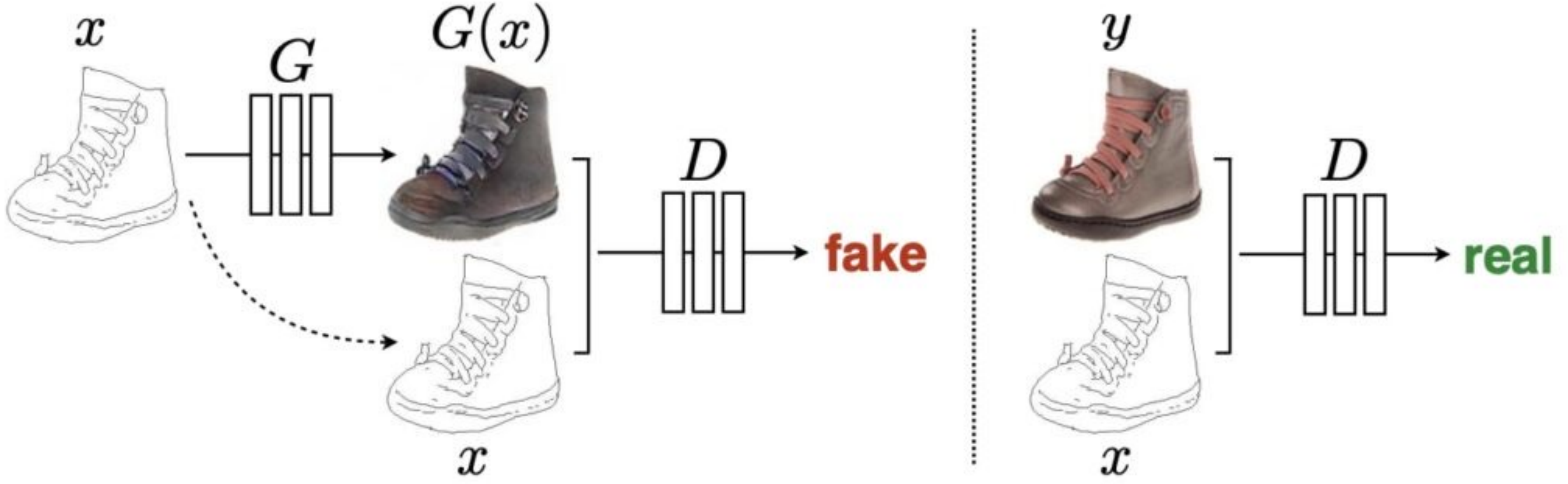
Conditional GAN (cGAN)



Generated MNIST digits, each row conditioned on one label

Pix2Pix GAN – Paired image translation

Pix2Pix GAN further extends the idea of CGAN, where the images are translated from input to an output image, ***conditioned on the input image***. Pix2Pix is a Conditional GAN that performs Paired Image-to-Image Translation.



The generator of every GAN we seen till now was fed a random-noise vector, sampled from a uniform distribution. But the Pix2Pix GAN eliminates the noise vector concept totally from the generator.

Pix2Pix GAN – Paired image translation

Applications

- Transforming a black and white image to a coloured image.
- Transforming edges into a meaningful image, e.g., given a boundary of an object, we realize a corresponding realistic image.
- Converting an aerial or satellite view to a map.
- Generating a segmentation map from a realistic image of an urban scene, comprising a road, sidewalk, pedestrians etc.
- Translating a photograph from day to a night time scenario or vice-versa.
- Transforming a low-resolution image to a high-resolution one

Pix2Pix GAN – Paired image translation

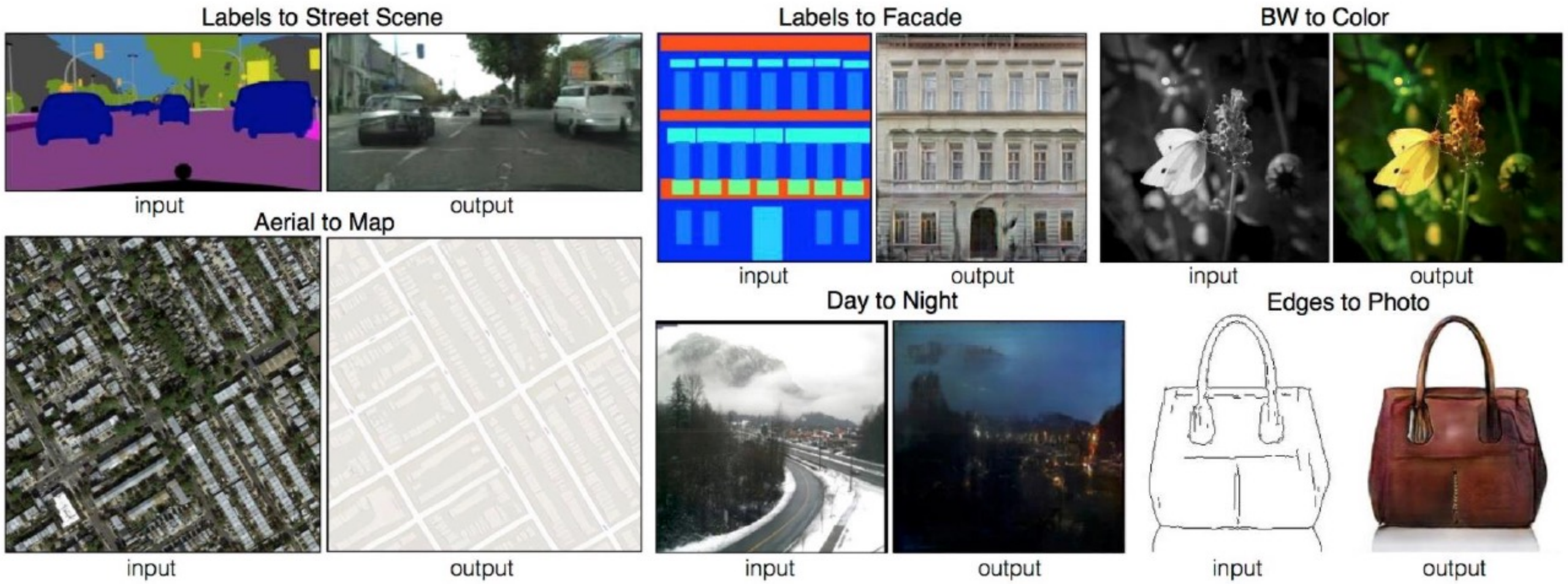


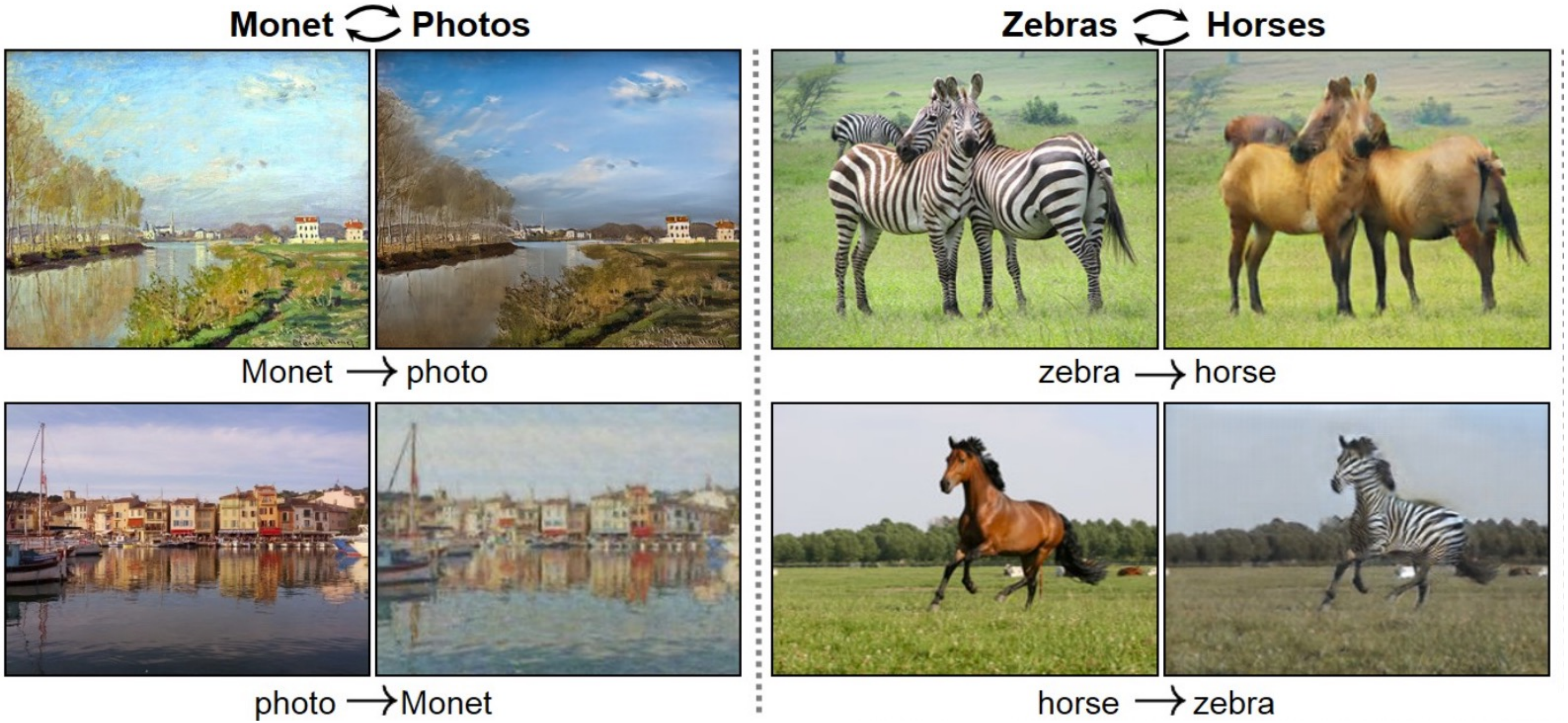
Image-to-Image Demo

Interactive Image Translation with pix2pix-tensorflow

Written by Christopher Hesse — February 19th, 2017

https://affinelayer.com/pixsrv/

CycleGAN - Unpaired image translation

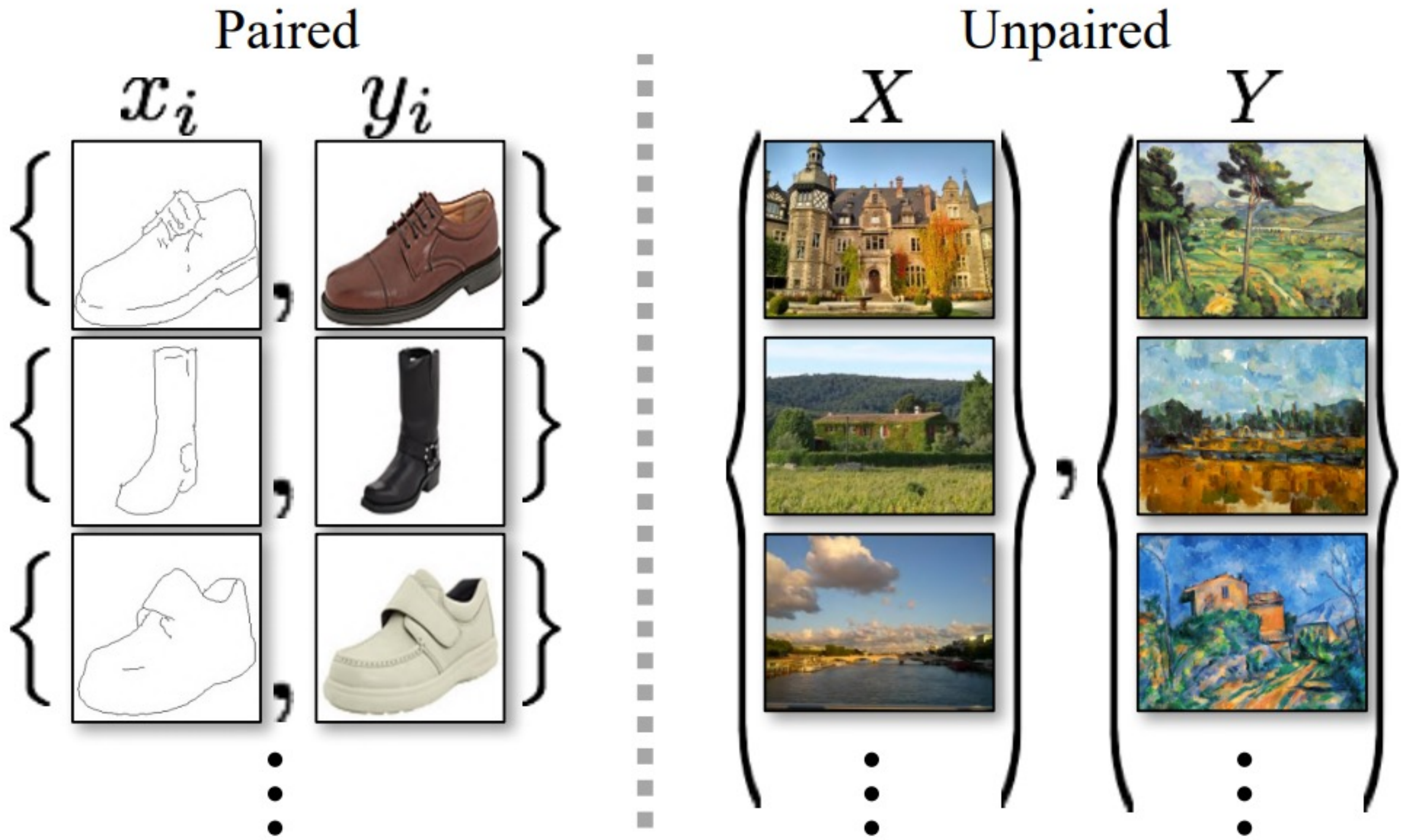


CycleGAN - Unpaired image translation

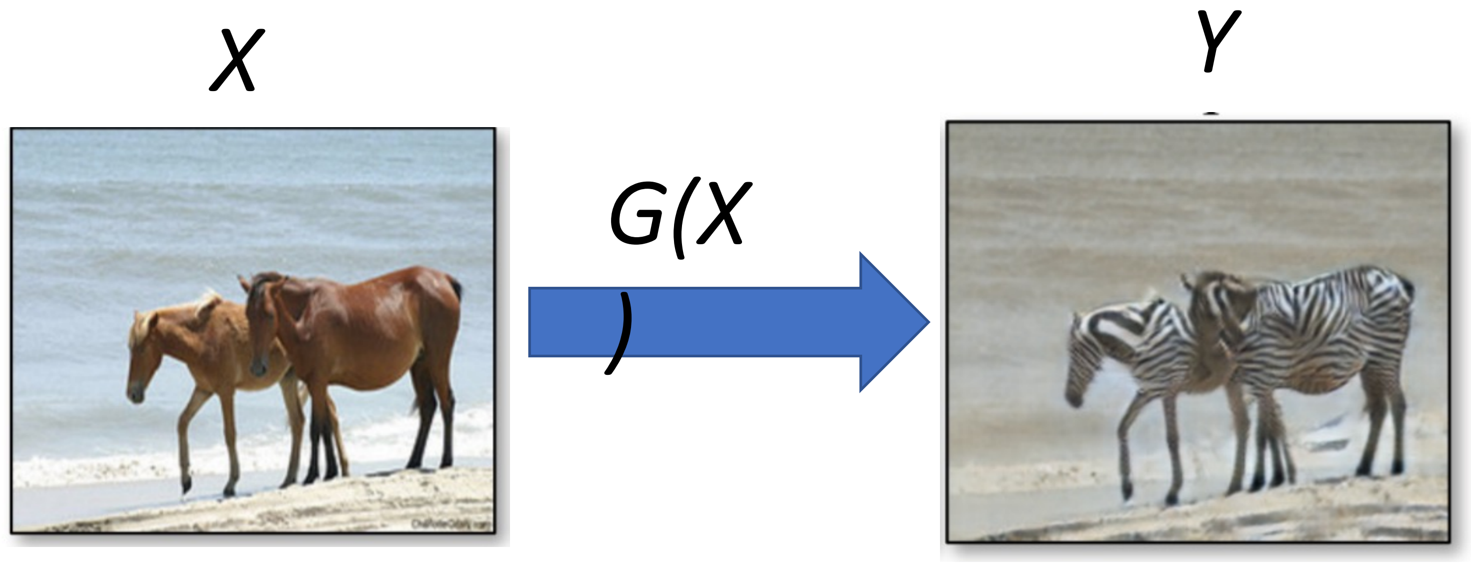
CycleGAN introduced two fundamental elements:

- Unpaired images
- Cycle consistency

CycleGAN - Unpaired image translation



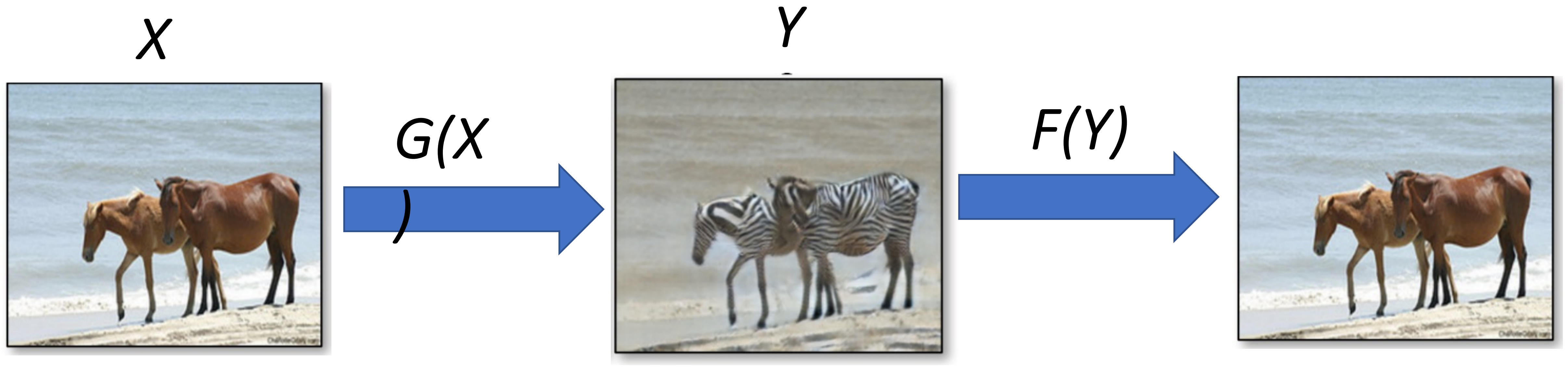
CycleGAN - Cycle Consistency



X: horses *Y: zebras*

G(X) -> Y *(horses -> zebras)*

CycleGAN - Cycle Consistency

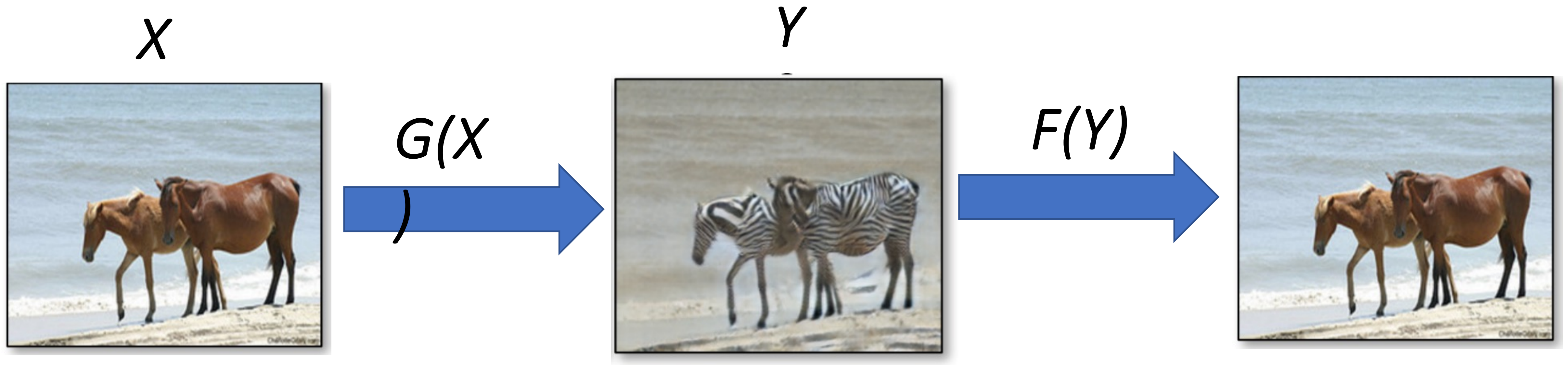


X: horses *Y: zebras*

G(X) -> Y *(horses -> zebras)*

F(Y) -> X *(zebras -> horses)*

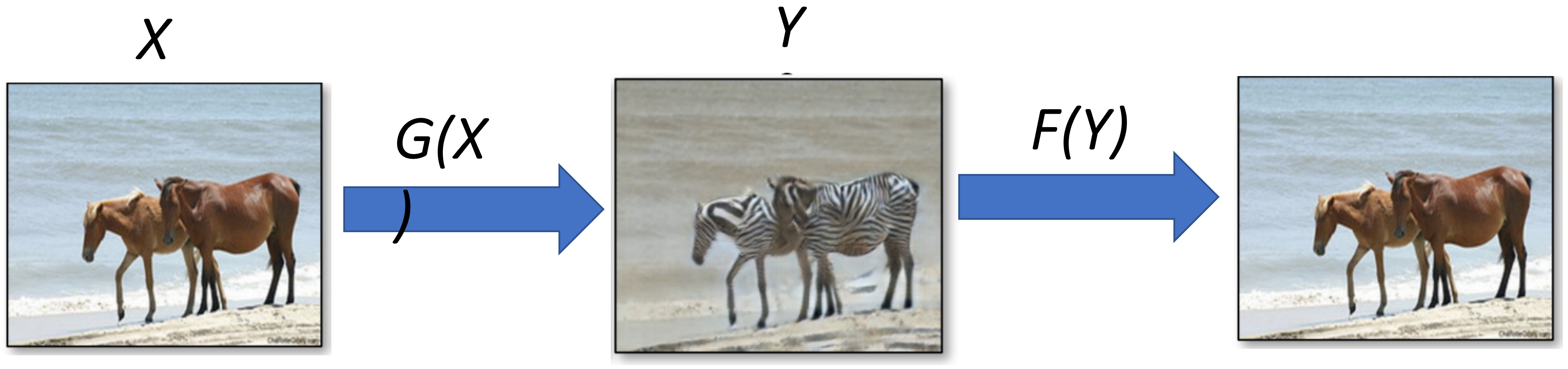
CycleGAN - Cycle Consistency



Cycle consistency: *F(G(X)) ~ X*

X: horses	*Y: zebras*
G(X) -> Y	*(horses -> zebras)*
F(Y) -> X	*(zebras -> horses)*

CycleGAN - Cycle Consistency



Cycle consistency: $F(G(X)) \sim X$

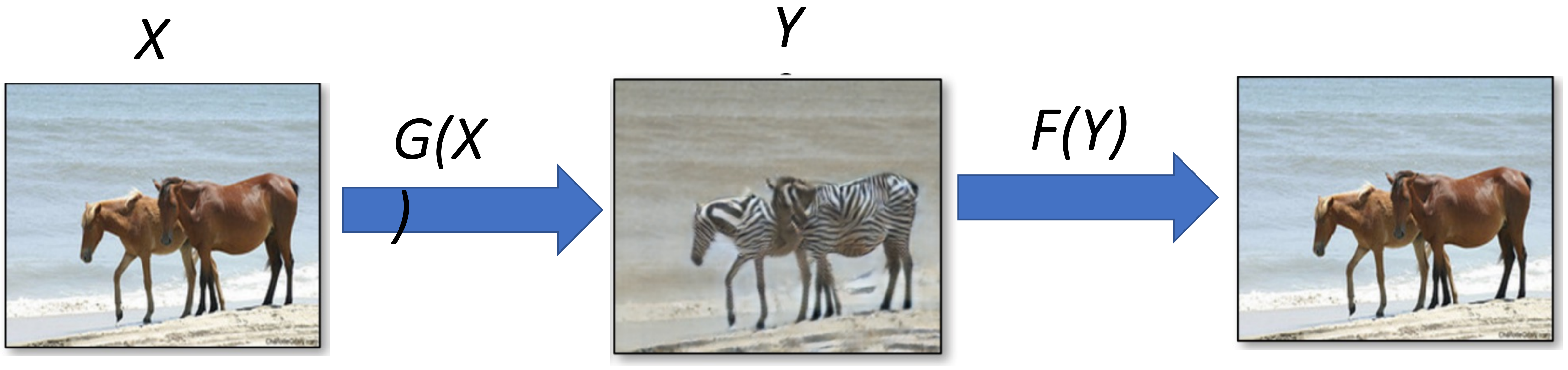
X: horse

X: horses *Y: zebras*

G(X) -> Y *(horses -> zebras)*

F(Y) -> X *(zebras -> horses)*

CycleGAN - Cycle Consistency



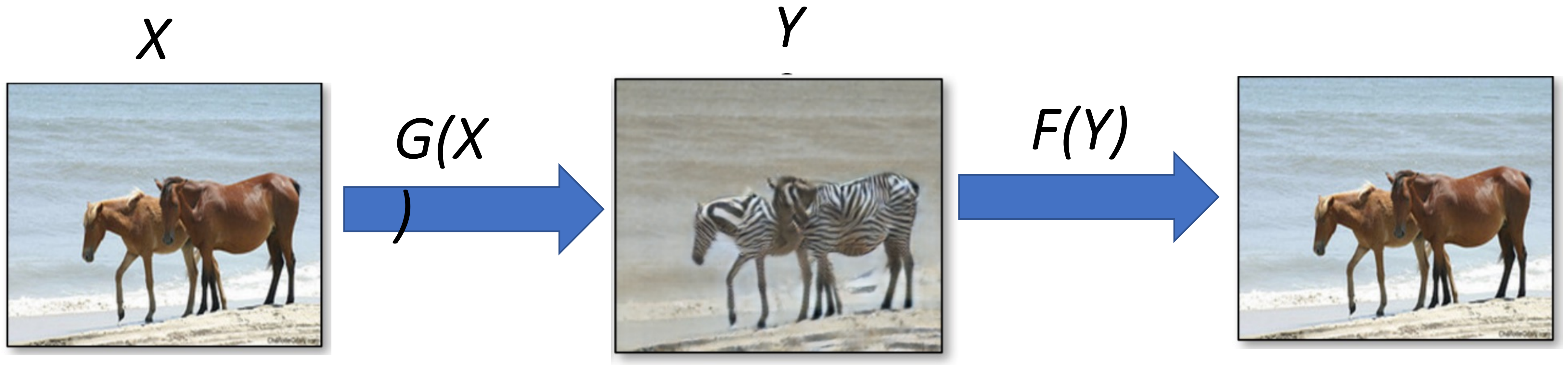
Cycle consistency: *$F(G(X)) \sim X$*

X: horse

G(X): G(horse) -> zebra

X: horses	*Y: zebras*
G(X) -> Y	*(horses -> zebras)*
F(Y) -> X	*(zebras -> horses)*

CycleGAN - Cycle Consistency



Cycle consistency: *F(G(X)) ~ X*

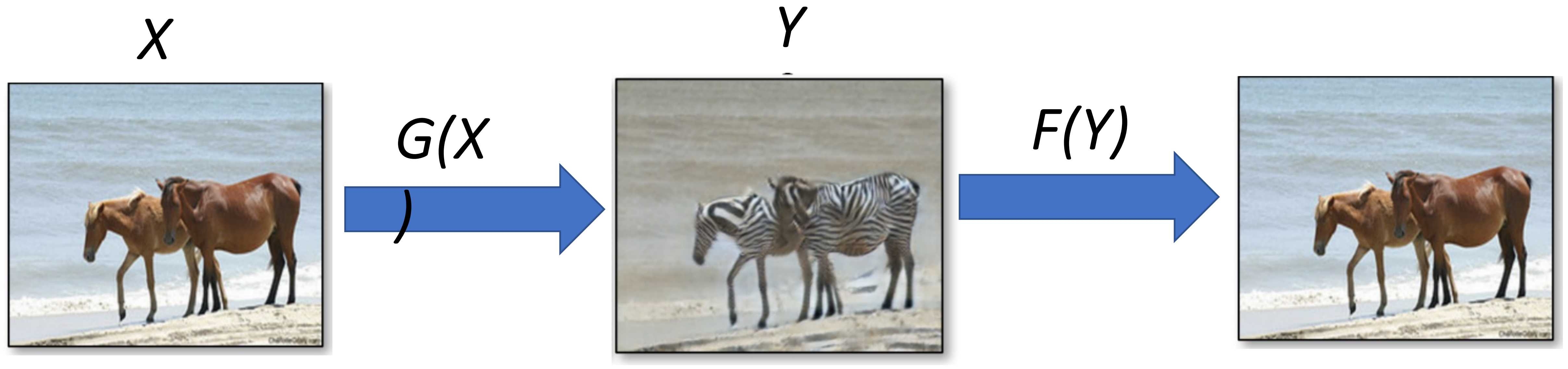
X: horses	*Y: zebras*
G(X) -> Y	*(horses -> zebras)*
F(Y) -> X	*(zebras -> horses)*

X: horse

G(X): G(horse) -> zebra

F(G(X)): F(G(horse)) -> F(zebra) -> horse

CycleGAN - Cycle Consistency

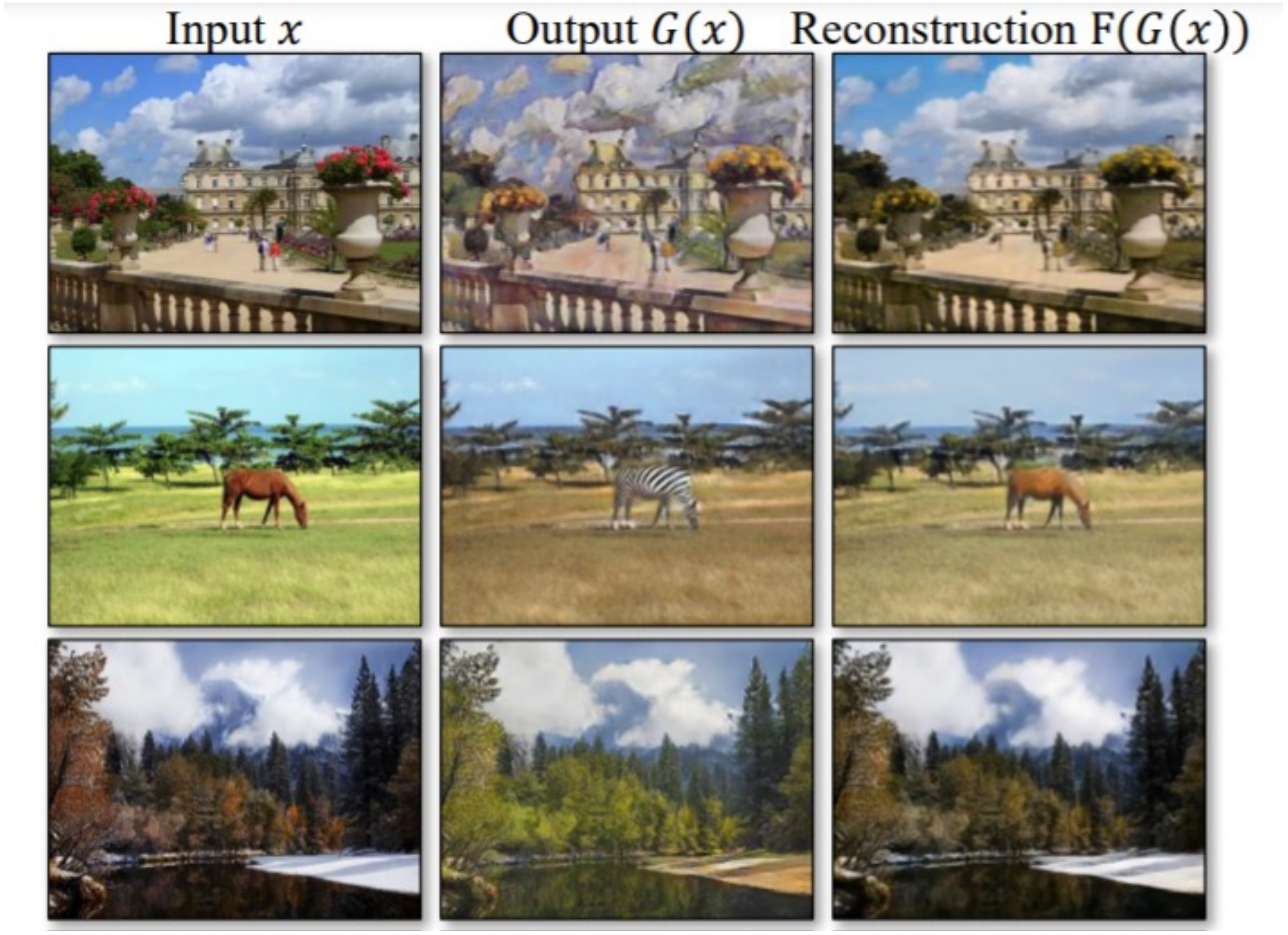


Cycle consistency: *F(G(X)) ~ X*

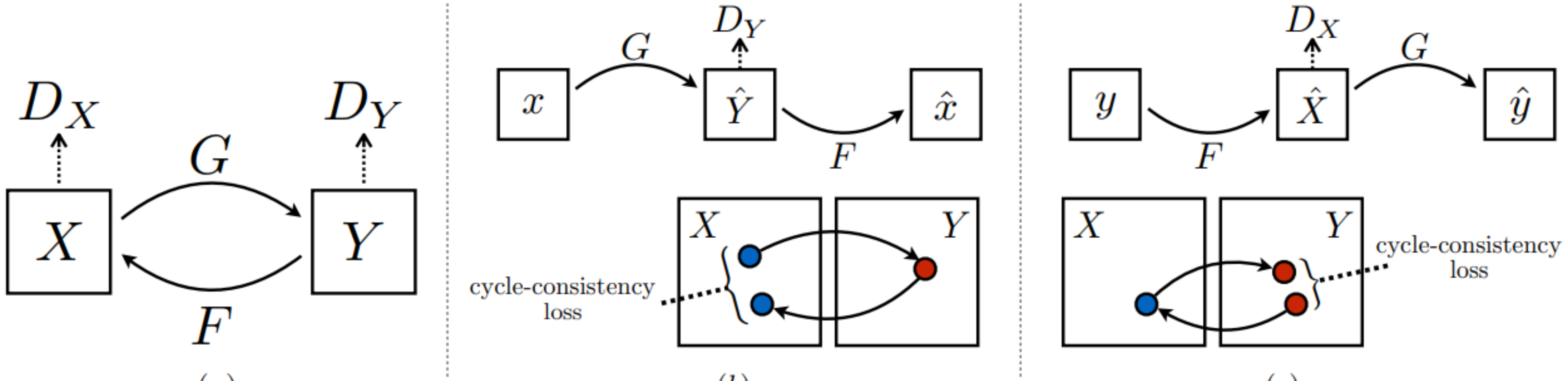
X: horses	*Y: zebras*
G(X) -> Y	*(horses -> zebras)*
F(Y) -> X	*(zebras -> horses)*

Formalized,
if G: X -> Y, F: Y -> X;
expect,
X -> G(X) -> F(G(X)) ~ X

CycleGAN - Cycle Consistency



CycleGAN – Cycle Consistency



(a) (b) (c)

CycleGAN – Objective function

Adversarial loss (for G: X -> Y)

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

CycleGAN – Objective function

Adversarial loss (for G: X -> Y)

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

Cycle-consistency loss

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

CycleGAN – Objective function

Adversarial loss (for G: X -> Y)

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

Cycle-consistency loss

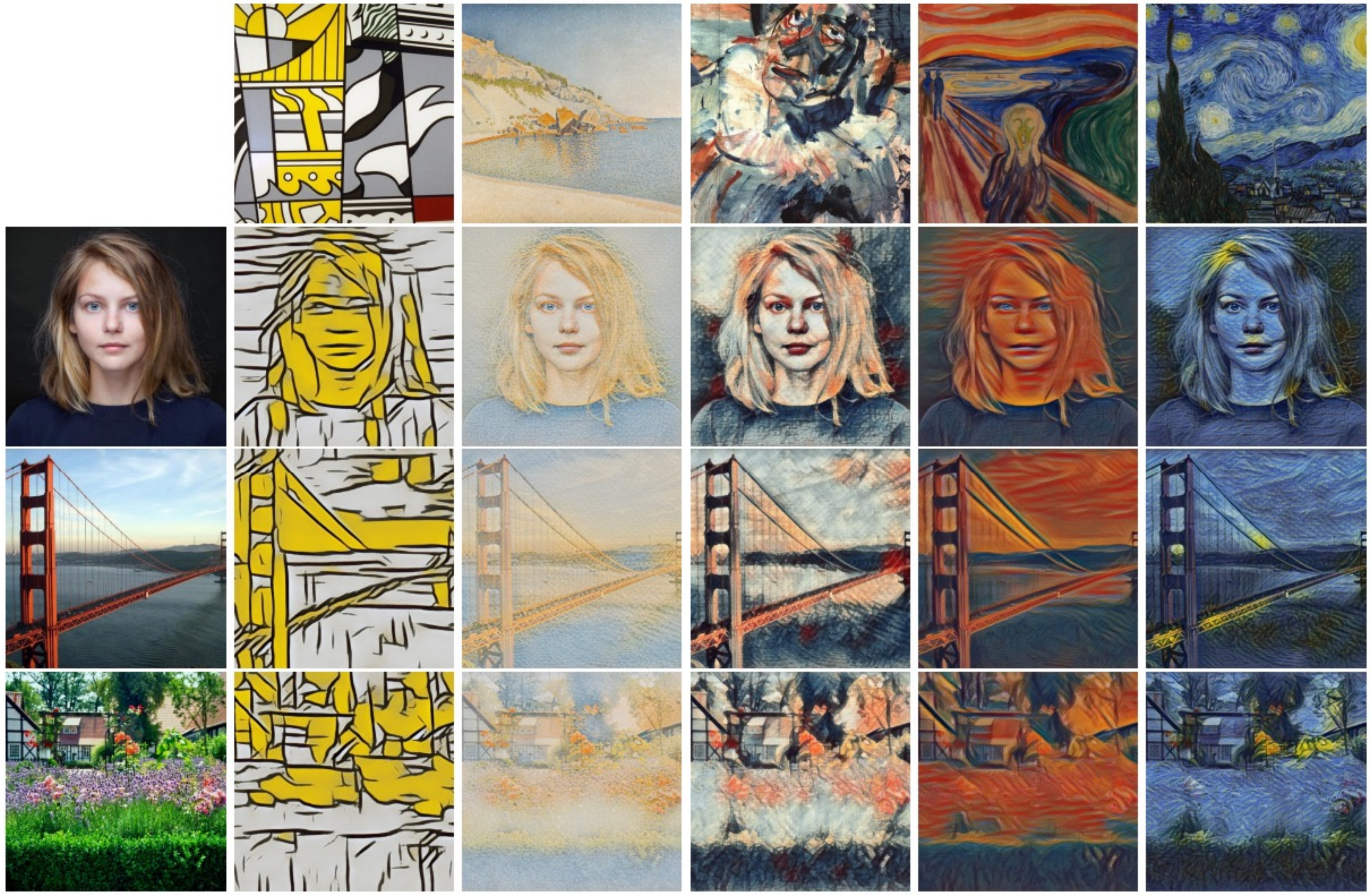
$$\begin{aligned}\mathcal{L}_{\text{cyc}}(G, F) = & \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] \\ & + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].\end{aligned}$$

Full objective

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

GAN Examples

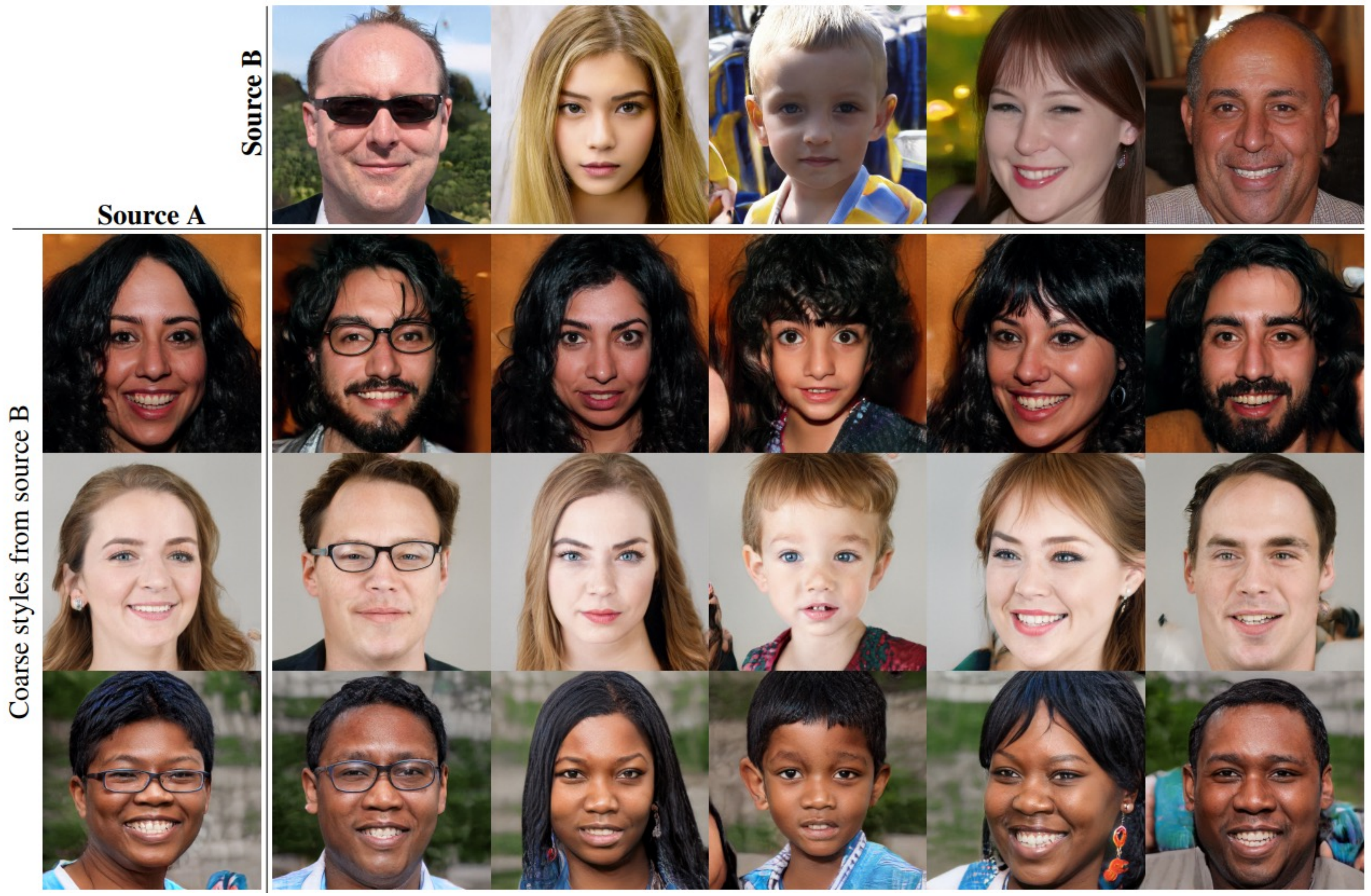
Conditional Style Transfer Network



Dumoulin, Vincent, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style." arXiv preprint arXiv:1610.07629 (2016).

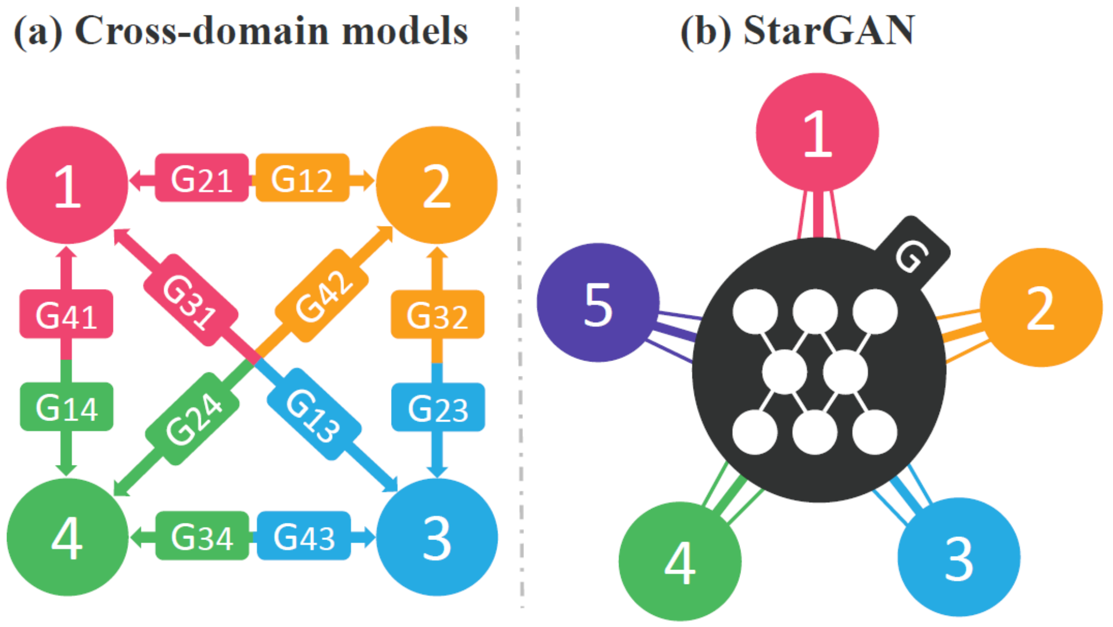
GAN Examples

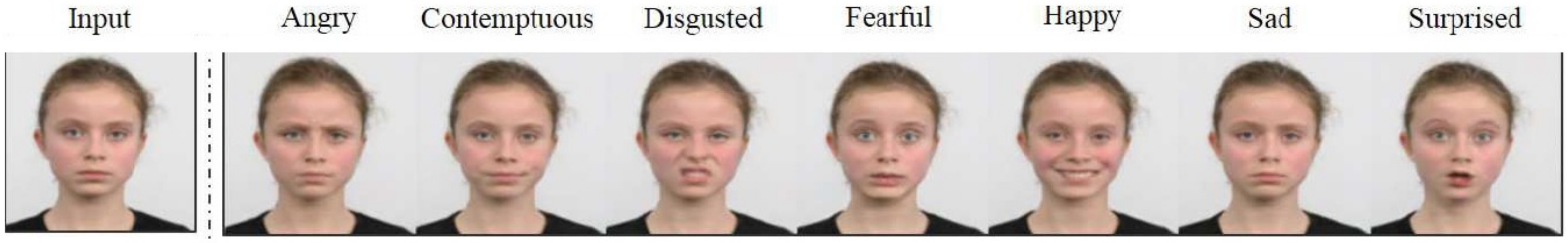
StyleGAN



Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

StarGAN – Multiple domains translation





StarGAN – Multiple domains translation

Background

- **GAN:** generates fake samples that are indistinguishable from real ones.
- **Conditional GAN:** generates samples conditioned on certain classes.
- **Pix2pix/CycleGAN:** changes a particular aspect of a given image to another (paired/unpaired).

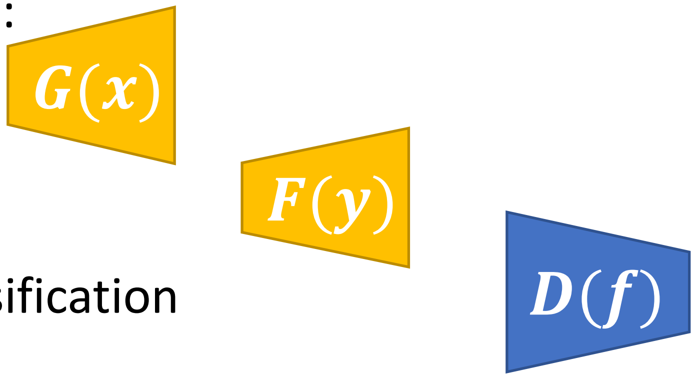
StarGAN extends the previous models to multiple domains. The main concepts taken from previous models are:

- Conditioned input: the input of the generator is conditioned by an input label (cGAN)
- Cycle consistency (CycleGAN)

StarGAN – Multiple domains translation

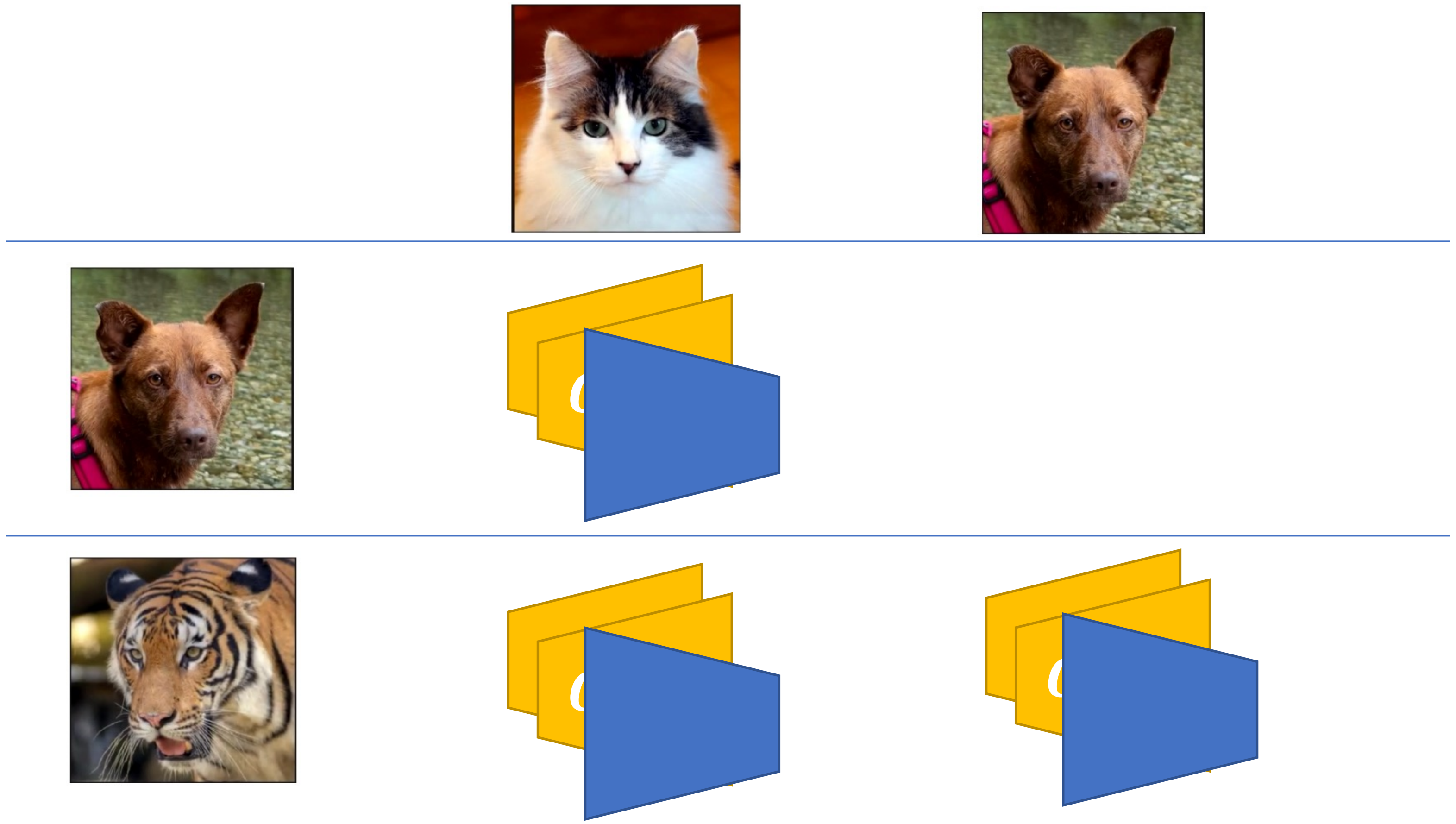
Consider a three class problem (cats, dogs, wildlife). With previous architectures (e.g., CycleGAN) we need three models for each pair of classes:

- A generator from class x to class y
- A generator from class y to class x
- A discriminator for fake/real image classification



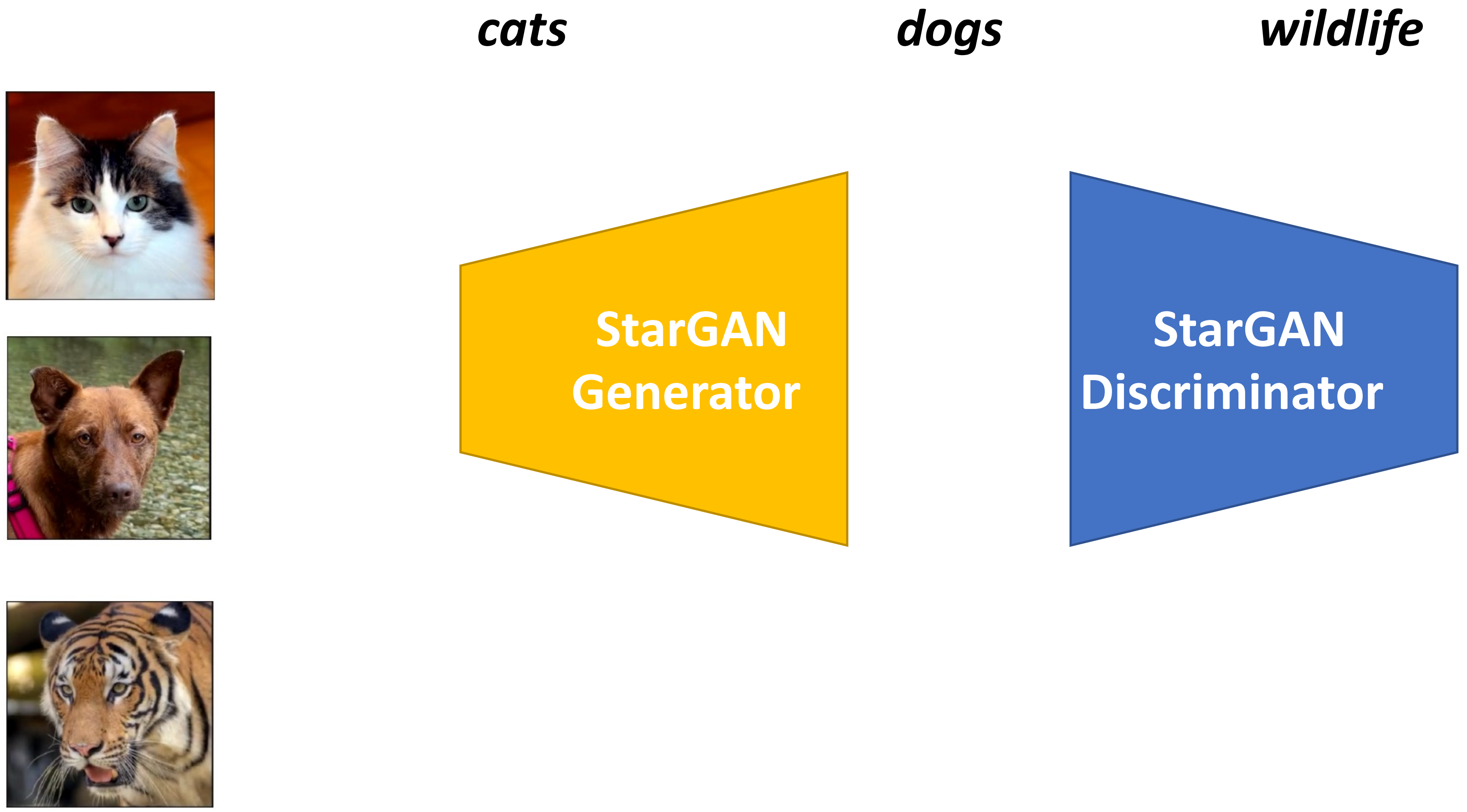
StarGAN – Multiple domains translation

With previous architectures (e.g., CycleGAN) we need three models for each pair of classes.



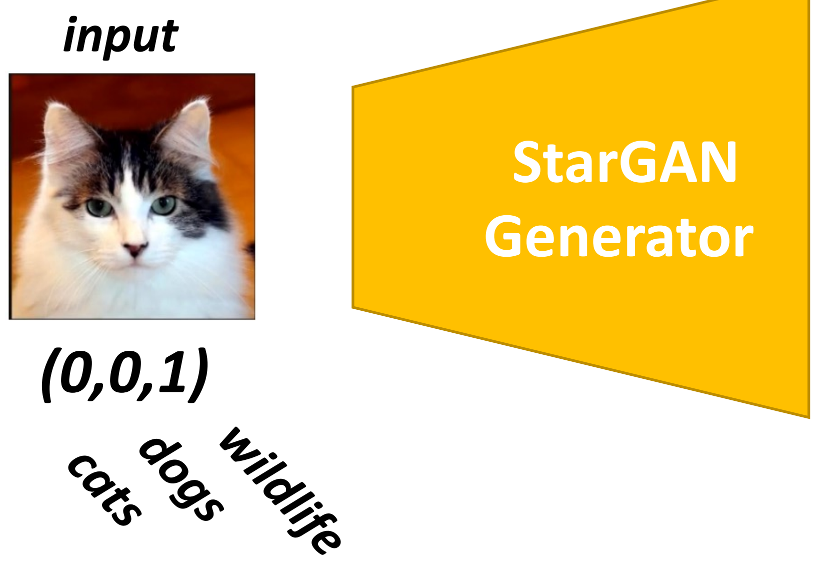
StarGAN – Multiple domains translation

With StarGAN we have only one generator and discriminator which work for each pair of classes.



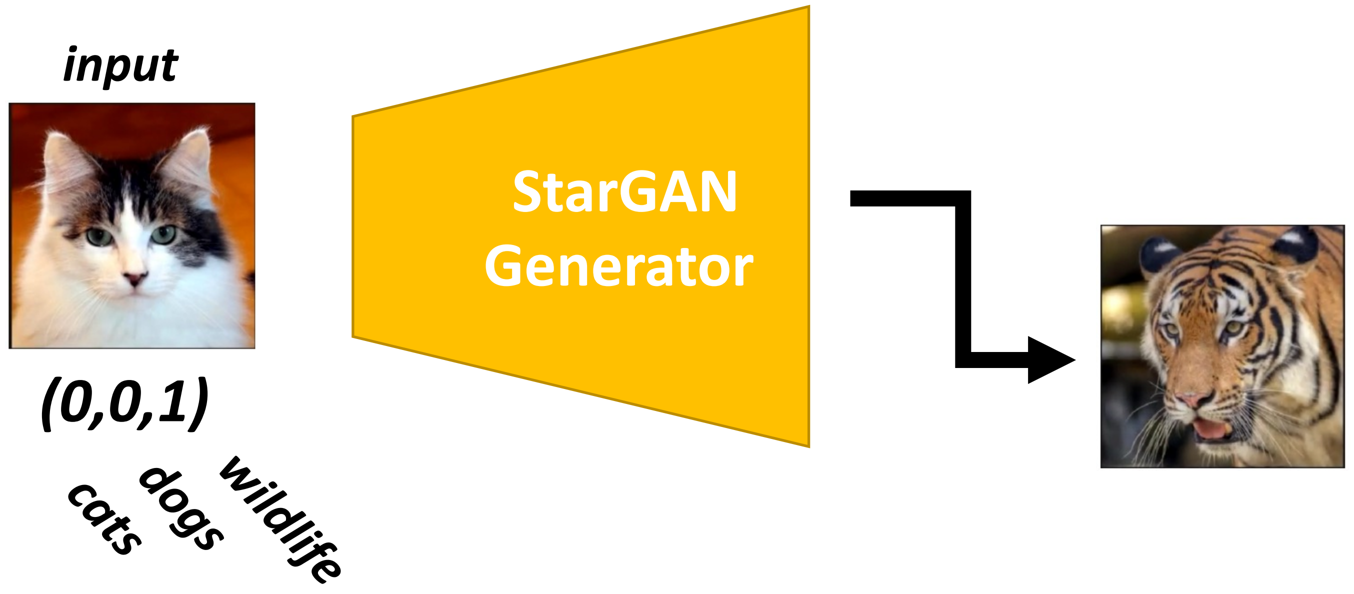
StarGAN – Multiple domains translation

The input of the generator is conditioned by a class label.



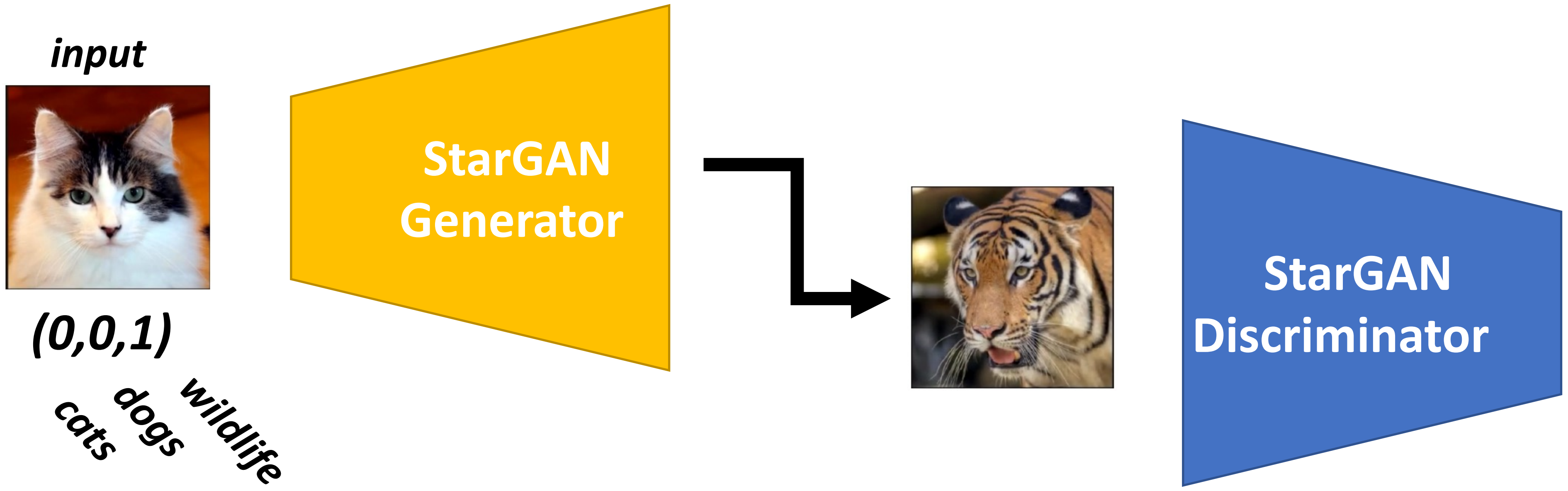
StarGAN – Multiple domains translation

The input of the generator is conditioned by a class label.



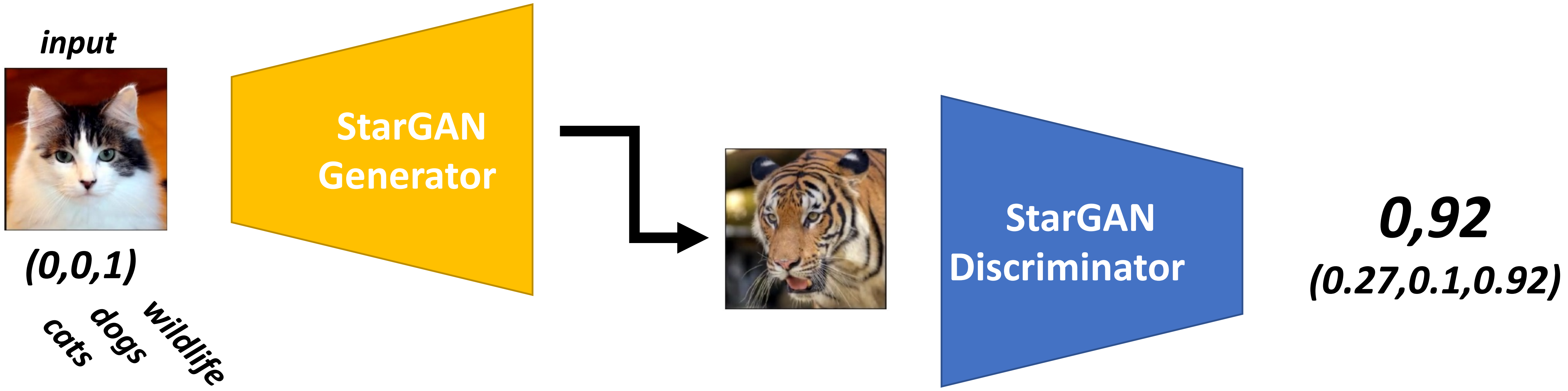
StarGAN – Multiple domains translation

The input of the generator is conditioned by a class label.



StarGAN – Multiple domains translation

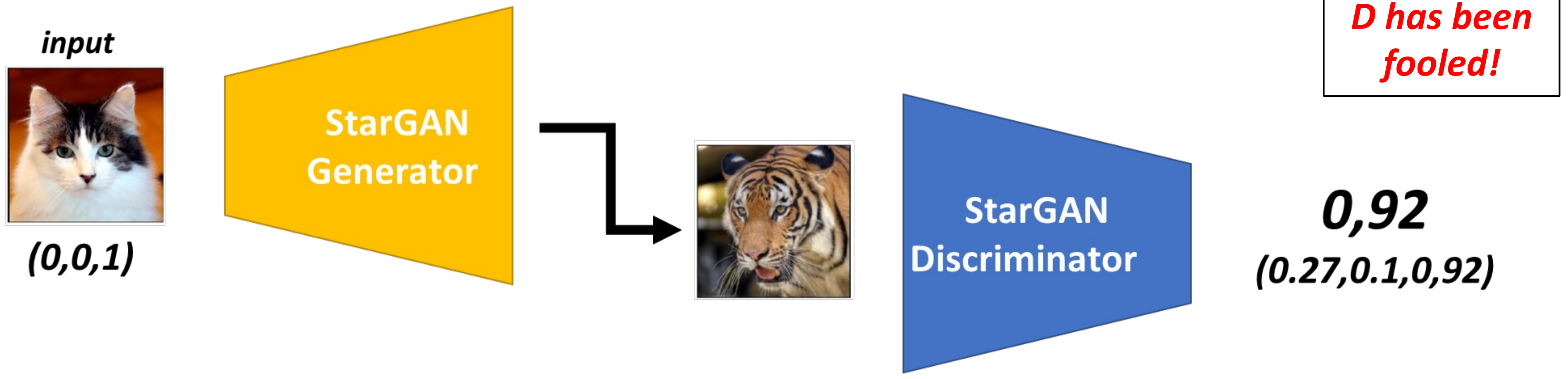
The input of the generator is conditioned by a class label.



StarGAN – Multiple domains translation

The generator has three learning objectives.

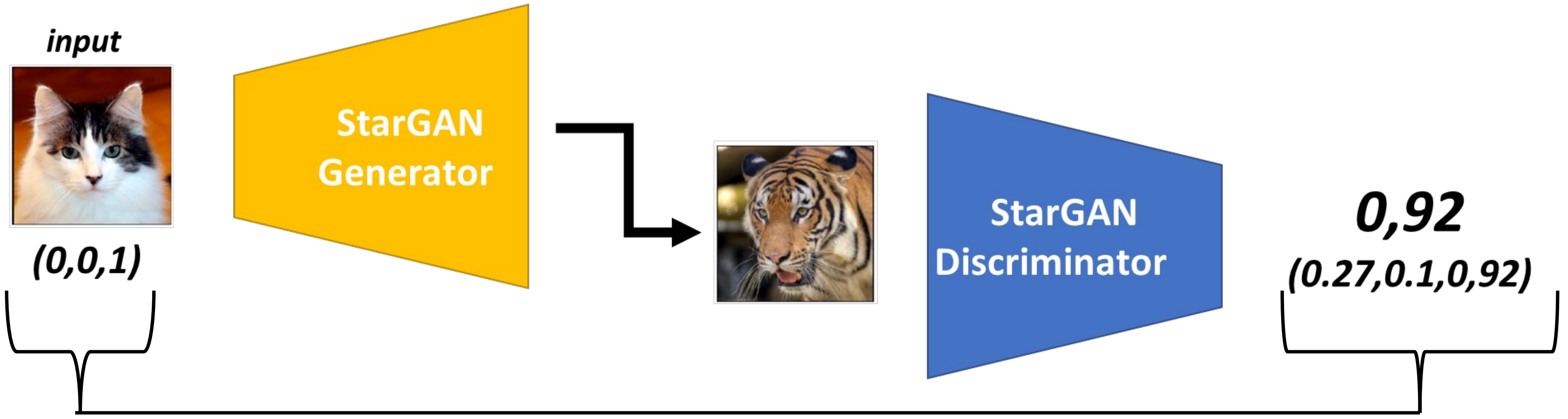
1. How real does this image look?
   $\Rightarrow$ Uses the output (real/fake) of the discriminator. If *D* is fooled, then *G* did a good job.



StarGAN – Multiple domains translation

The generator has three learning objectives.

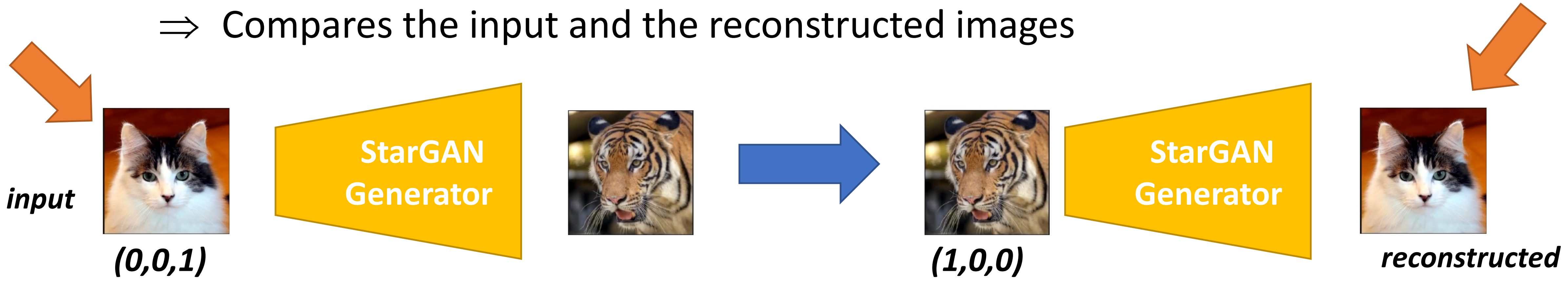
1. How real does this image look?
   ⇒ Uses the output (real/fake) of the discriminator. If *D* is fooled, then *G* did a good job.
2. How much does the image look like it belongs to the target domain ?
   ⇒ Uses the discriminator's classification output in comparison to the target label



StarGAN – Multiple domains translation

The generator has three learning objectives.

1. How real does this image look?
   ⇒ Uses the output (real/fake) of the discriminator. If *D* is fooled, then *G* did a good job.
2. How much does the image look like it belongs to the target domain ?
   ⇒ Uses the discriminator's classification output in comparison to the target label
3. Cycle Consistency
   ⇒ Compares the input and the reconstructed images

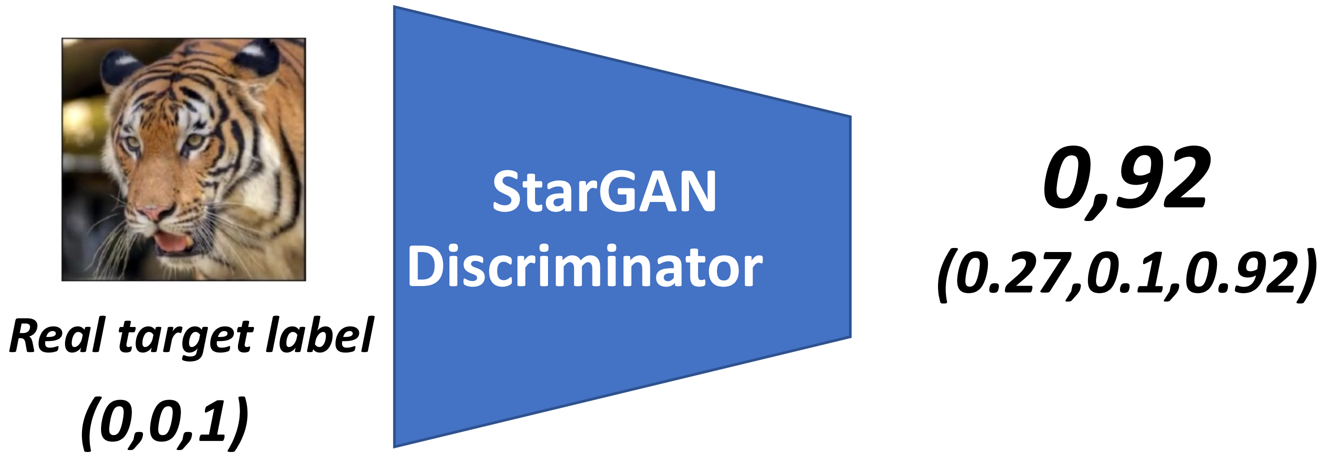


StarGAN – Multiple domains translation

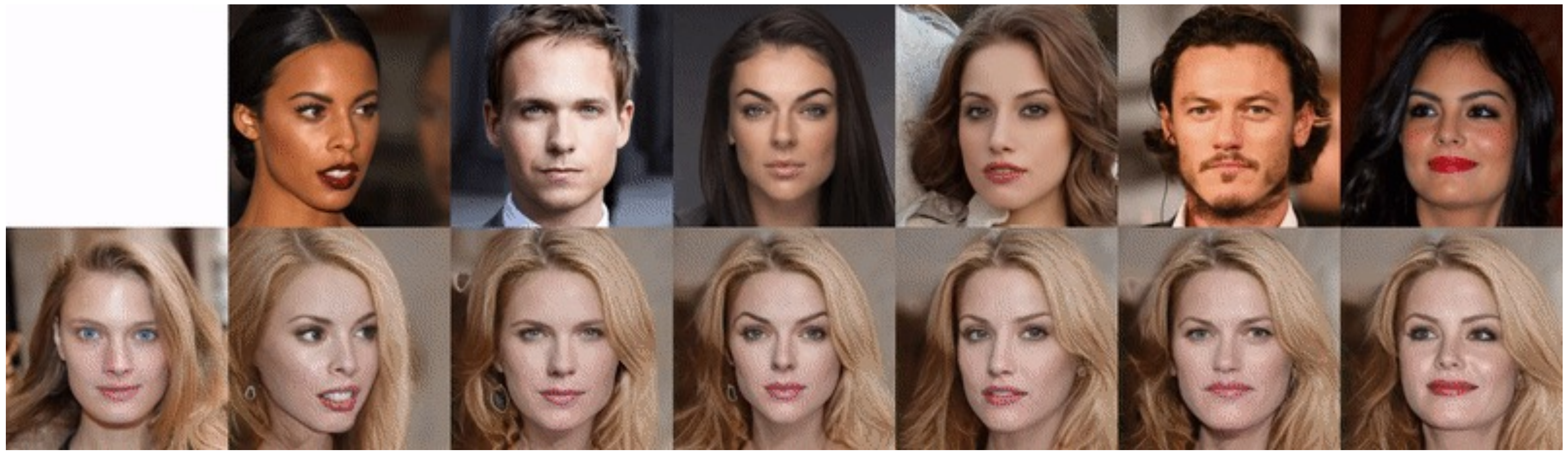
The discriminator has two learning objectives.

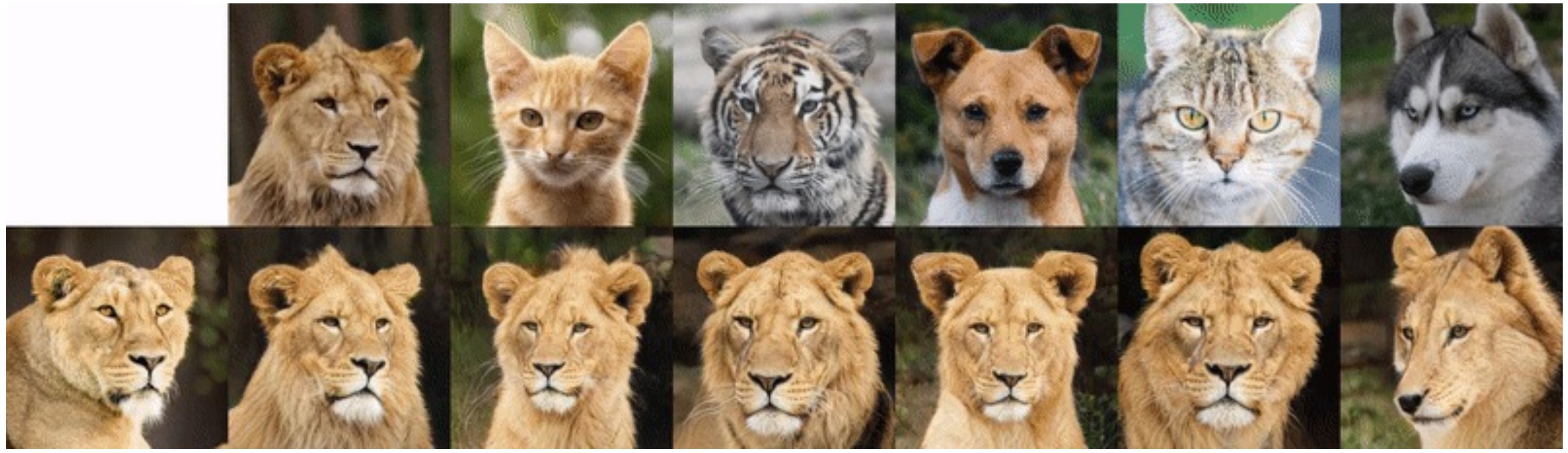
1. Correctly classify real/fake image
   ⇒ Learns the properties of a real/fake image.

2. Determine the class label
   ⇒ Compares the (real) target labels and the obtained scores.



StarGANv2 – Multiple domains translation





Y. Choi et al., StarGAN v2: Diverse Image Synthesis for Multiple Domains – CVPR 2020

References

- *"Generative Adversarial Networks." Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. ArXiv 2014.*
- *Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).*
- *Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).*
- *Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.*
- *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.*
- *Choi, Yunjey, et al. "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation." Proceedings of the IEEE*