UNIVERSITY OF TECHNOLOGY
IN THE EUROPEAN CAPITAL OF CULTURE
CHEMNITZ

Neurocomputing

Generative Adversarial Networks

Julien Vitay
Professur für Künstliche Intelligenz - Fakultät für Informatik

1 - Generative adversarial network

Generative models

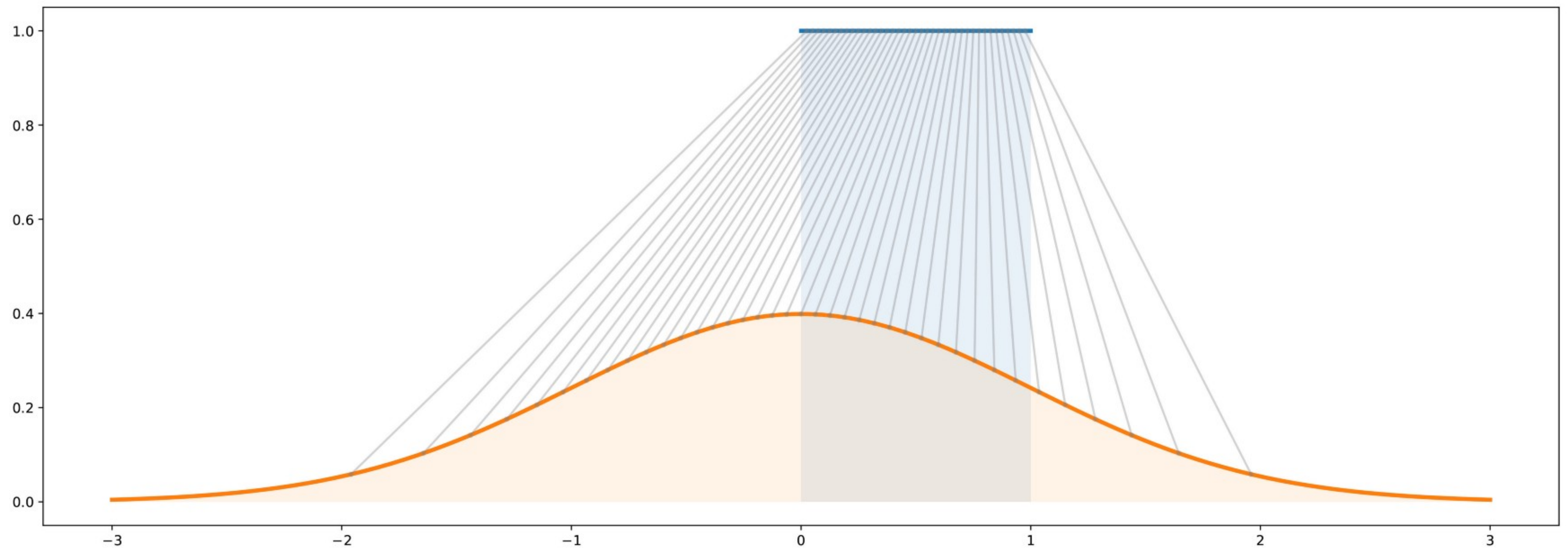
- An autoencoder learns to first encode inputs in a **latent space** and then use a generative model to model the data distribution.

$$\mathcal{L}_\text{autoencoder}(\theta, \phi) = \mathbb{E}_{\mathbf{x} \in \mathcal{D}, \mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [- \log p_\theta(\mathbf{z})]$$

- Couldn't we learn a decoder using random noise as input but still learning the distribution of the data?

$$\mathcal{L}_\text{GAN}(\theta, \phi) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, 1)} [- \log p_\theta(\mathbf{z})]$$

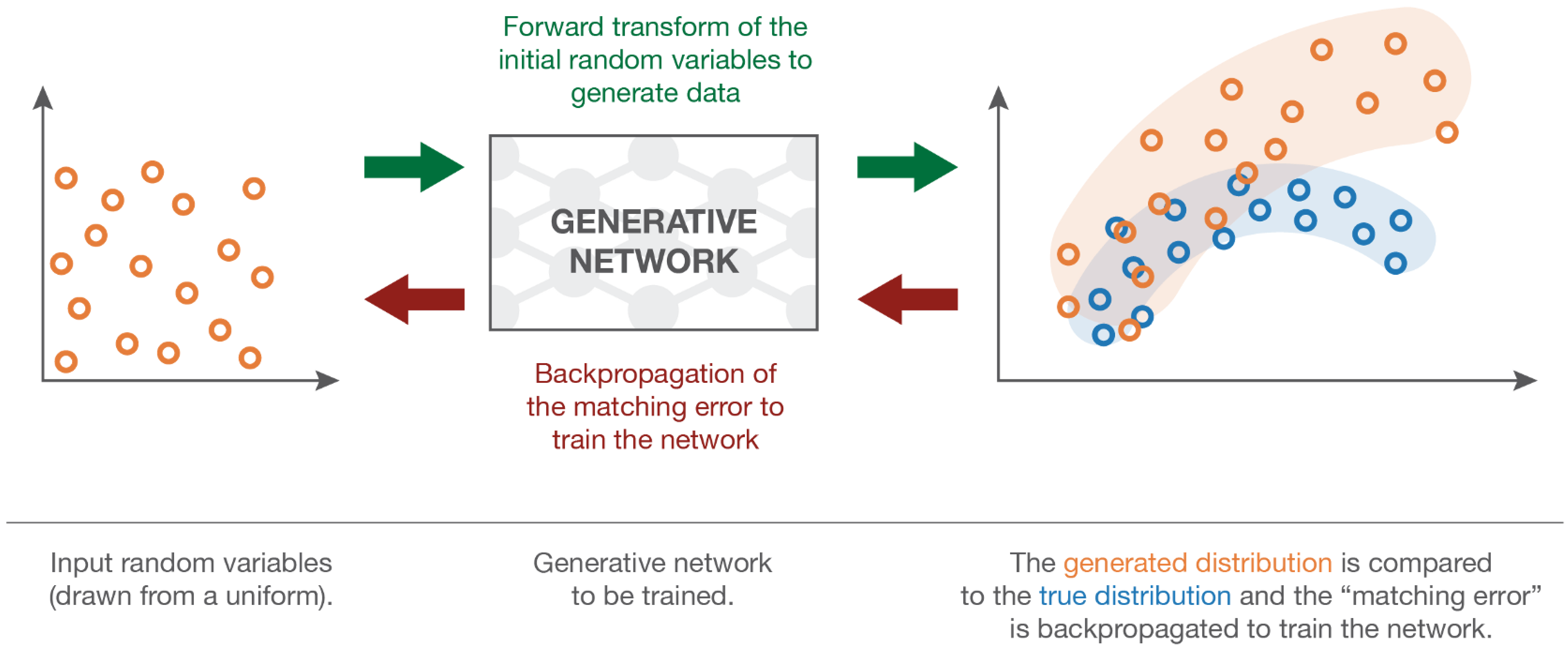
- After all, this is how random numbers are generated: a uniform distribution of pseudo-random numbers is transformed into samples of another distribution using a mathematical formula.



Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Generative models

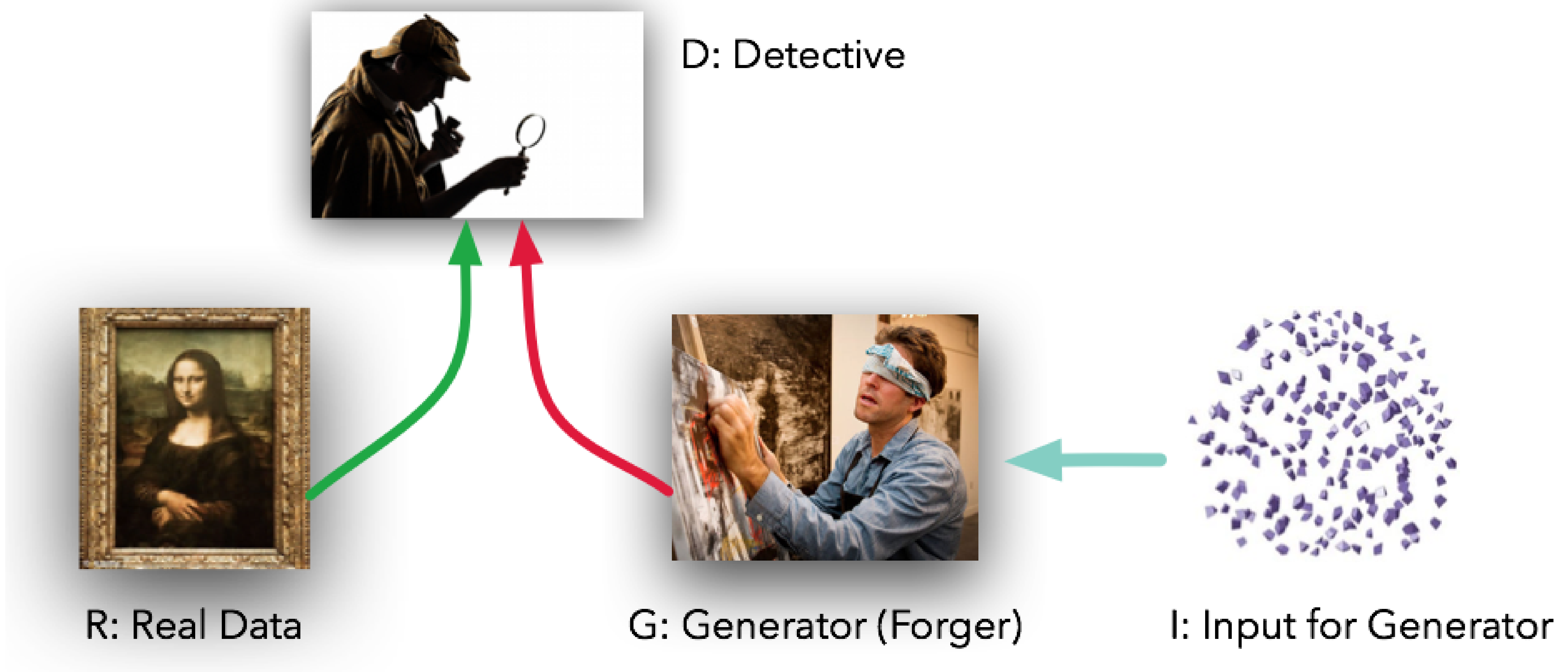
- The problem is how to estimate the discrepancy between the true distribution and the generated distribution when we only have samples.
- The Maximum Mean Discrepancy (MMD) approach allows to do that, but does not work very well in highly-dimensional spaces.


Forward transform of the initial random variables to generate data
GENERATIVE NETWORK
Backpropagation of the matching error to train the network
Input random variables (drawn from a uniform).
Generative network to be trained.
The generated distribution is compared to the true distribution and the "matching error" is backpropagated to train the network.

Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Generative adversarial network

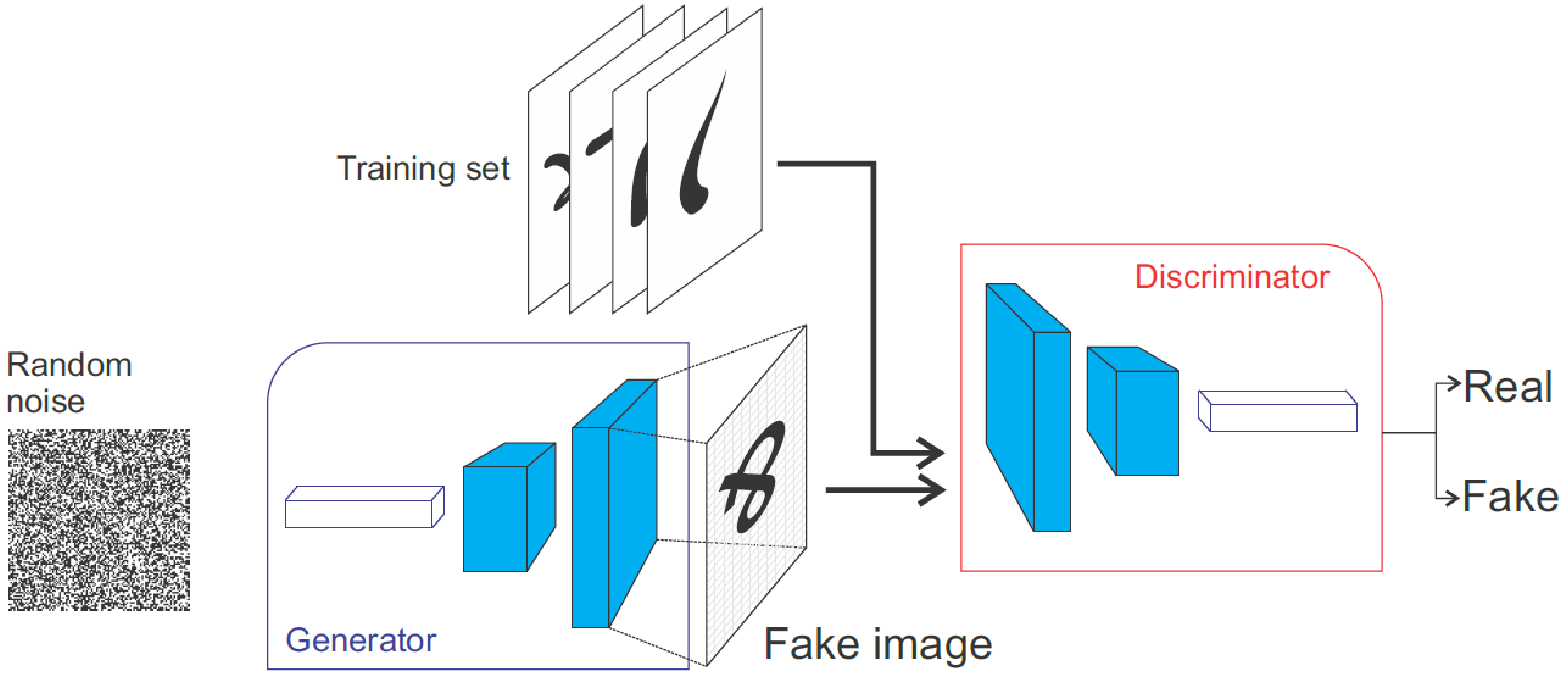
- The **Generative Adversarial Network** (GAN, Goodfellow at al., 2014) is a smart way of providing a loss function to the generative model. It is composed of two parts:
    - The **Generator** (or decoder) produces an image based on latent variables sampled from some random distribution (e.g. uniform or normal).
    - The **Discriminator** has to recognize real images from generated ones.



Source: https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

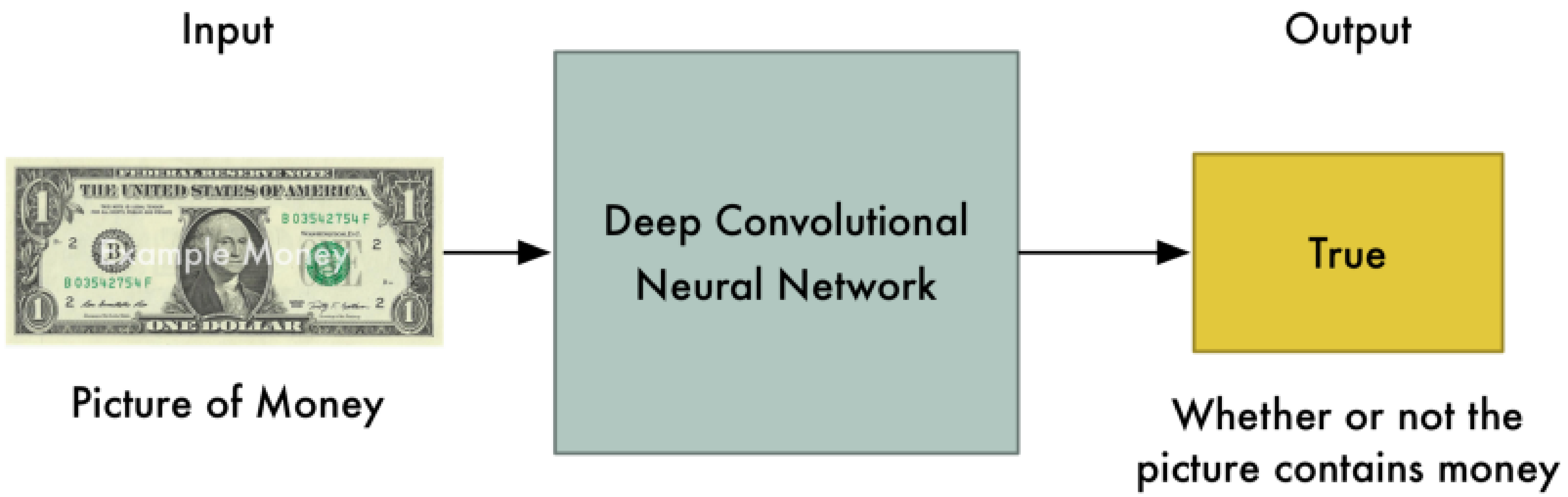
Generative adversarial network

- The generator only sees noisy latent representations and outputs a reconstruction.
- The discriminator gets alternatively real or generated inputs and predicts whether it is real or fake.



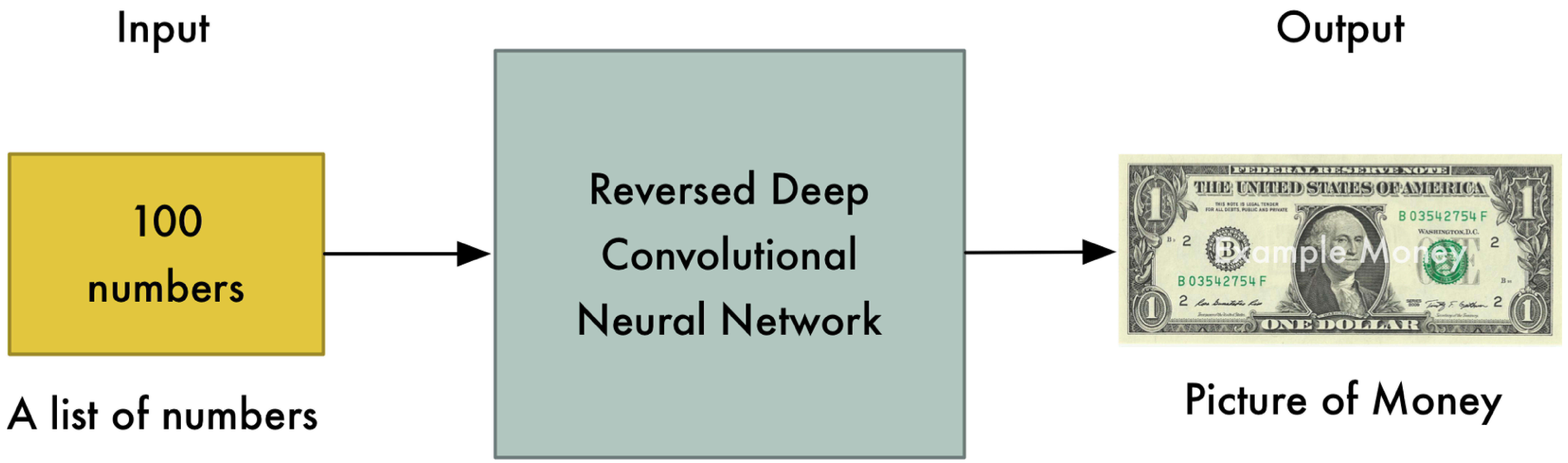
Source: https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml

The discriminator should be able to recognize false bills from true ones


Input
Picture of Money
Deep Convolutional Neural Network
Output
True
Whether or not the picture contains money

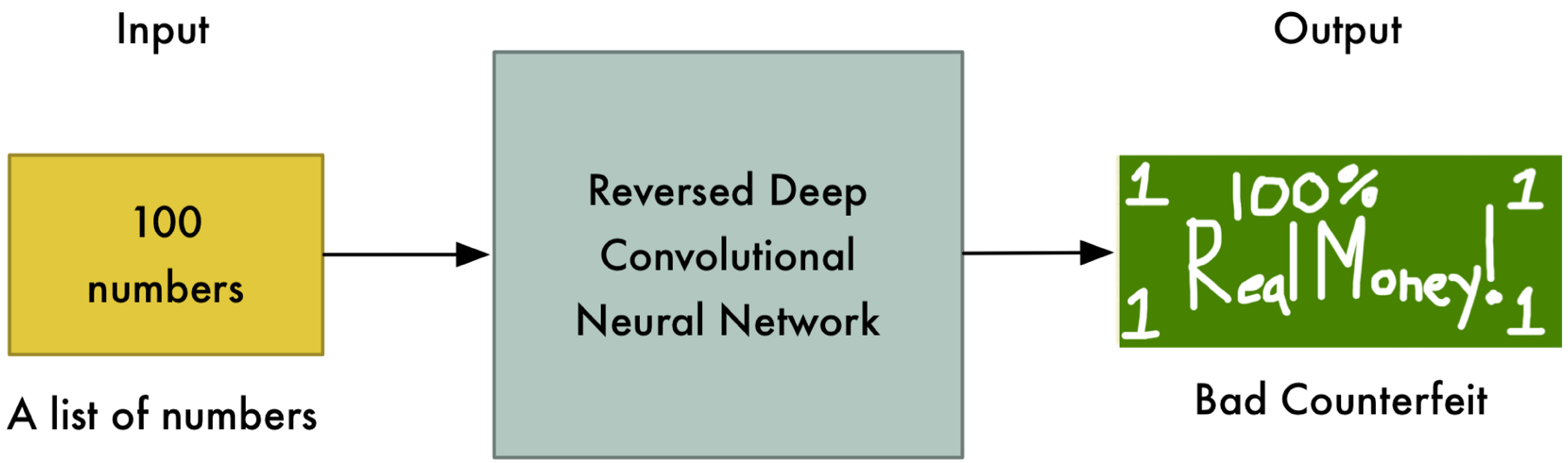
Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

The generator should be able to generate realistic bills



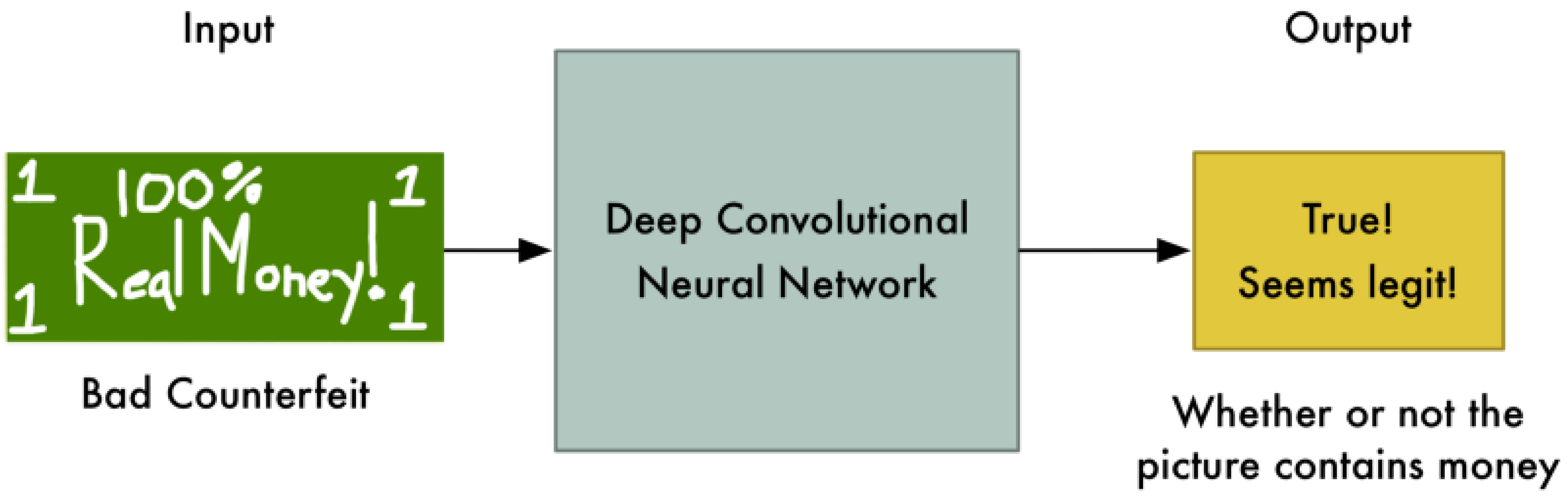
Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

The generator is initially very bad...



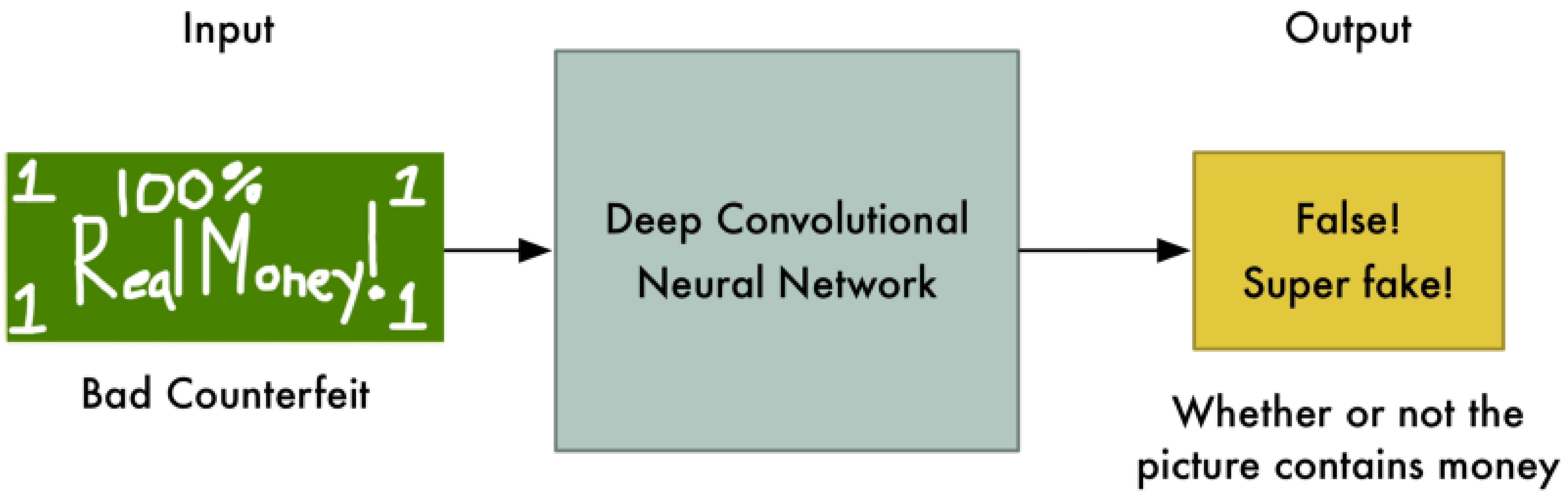
Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

… but the discriminator too!


Input
1 100% 1
1 Real Money! 1
Bad Counterfeit
Deep Convolutional Neural Network
Output
True! Seems legit!
Whether or not the picture contains money

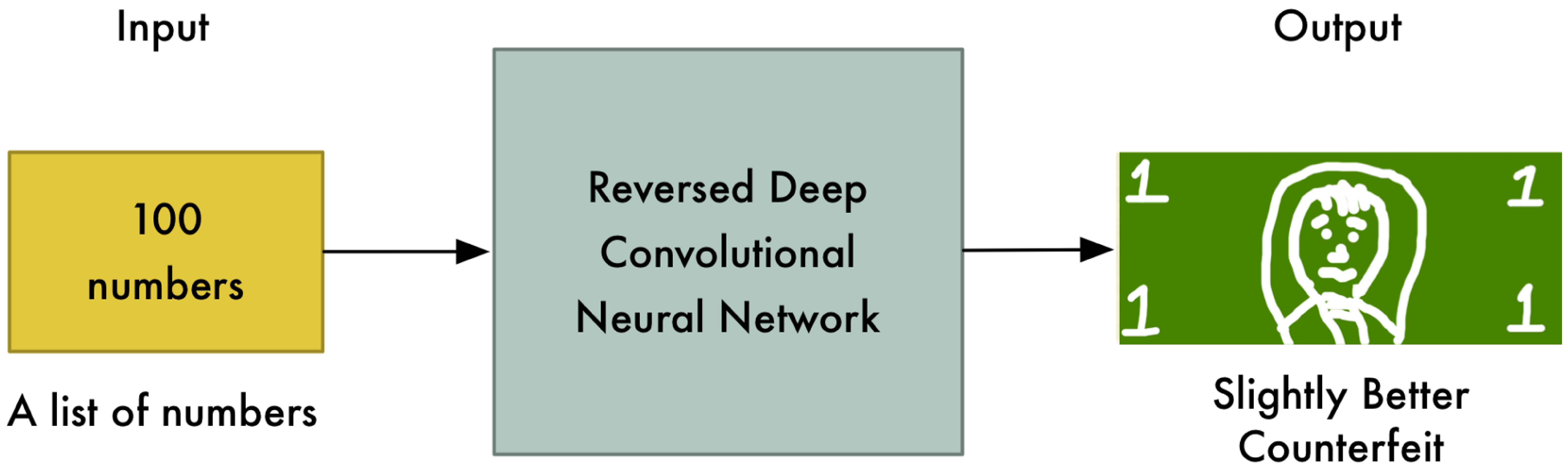
Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

After a while, the discriminator gets better...


Input
1 100% 1
Real Money!
1 1
Bad Counterfeit
Deep Convolutional Neural Network
Output
False! Super fake!
Whether or not the picture contains money

Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

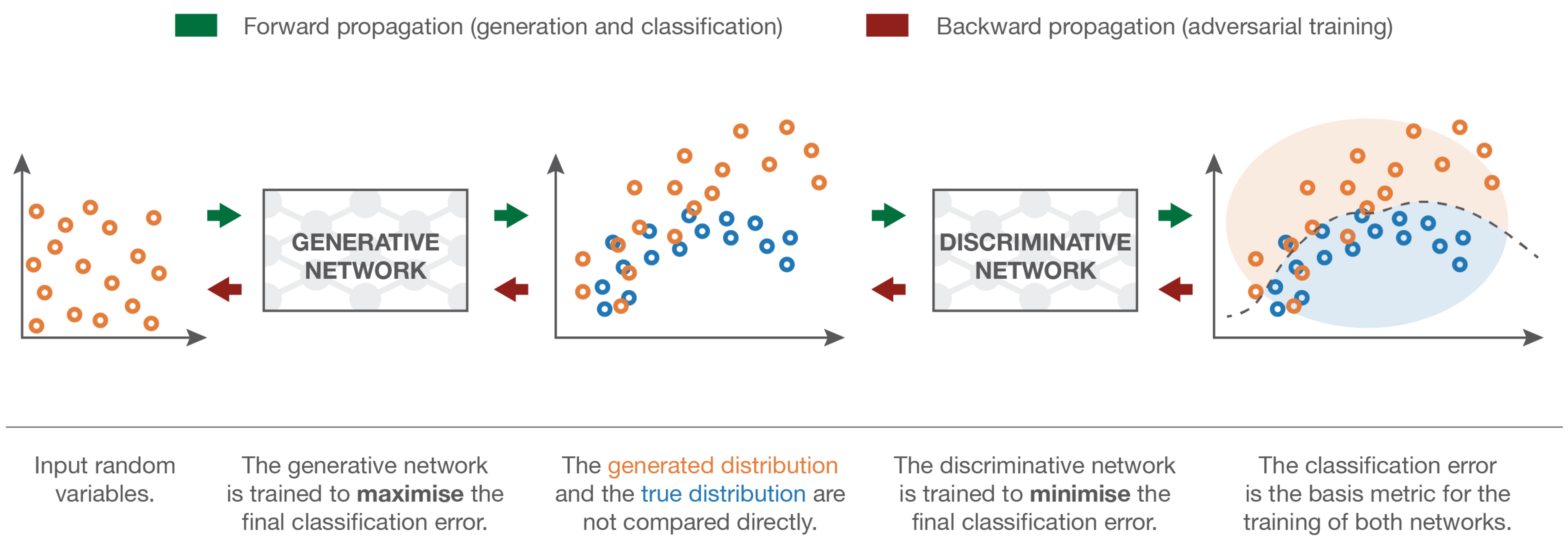
So the generator also has to improve


Input
100 numbers
A list of numbers
Reversed Deep Convolutional Neural Network
Output
1
1
1
1
Slightly Better Counterfeit

Source: https://medium.com/@ageitgey/abusing-generative-adversarial-networks-to-make-8-bit-pixel-art-e45d9b96cee7

Generative adversarial network

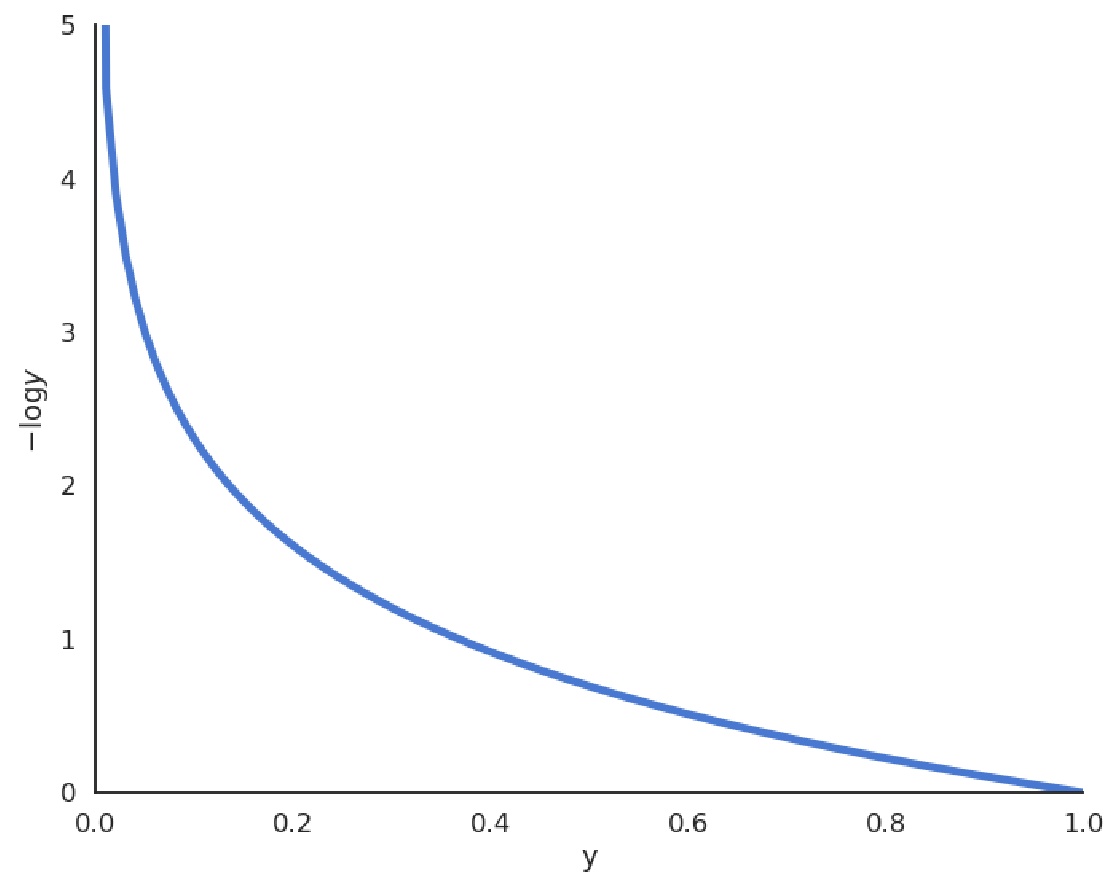
- The generator and the discriminator are in competition with each other.
- The discriminator uses pure **supervised learning**: we know if the input is real or generated (binary classification) and train the discriminator accordingly.
- The generator tries to fool the discriminator, without ever seeing the data!


🟩 Forward propagation (generation and classification)
🟥 Backward propagation (adversarial training)
GENERATIVE NETWORK
DISCRIMINATIVE NETWORK
Input random variables.
The generative network is trained to **maximise** the final classification error.
The generated distribution and the true distribution are not compared directly.
The discriminative network is trained to **minimise** the final classification error.
The classification error is the basis metric for the training of both networks.

Source: https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Loss of the discriminator

- Let's define $x \sim P_\text{data}(x)$ as a real image from the dataset and $G(z)$ as an image generated by the generator, where $z \sim P_z(z)$ is a random input.
- The output of the discriminator is a single sigmoid neuron:
  - $D(x) = 1$ for real images.
  - $D(G(z)) = 0$ for generated images
- We want both $D(x)$ and $1 - D(G(z))$ to be close from 1.



- The goal of the discriminator is to **minimize** the negative log-likelihood (binary cross-entropy) of classifying correctly the data:

$$\mathcal{L}(D) = \mathbb{E}_{x \sim P_\text{data}(x)} [- \log D(x)] + \mathbb{E}_{z \sim P_z(z)} [- \log(1 - D(G(z)))]$$

- It is similar to logistic regression: x belongs to the positive class, $G(z)$ to the negative class.

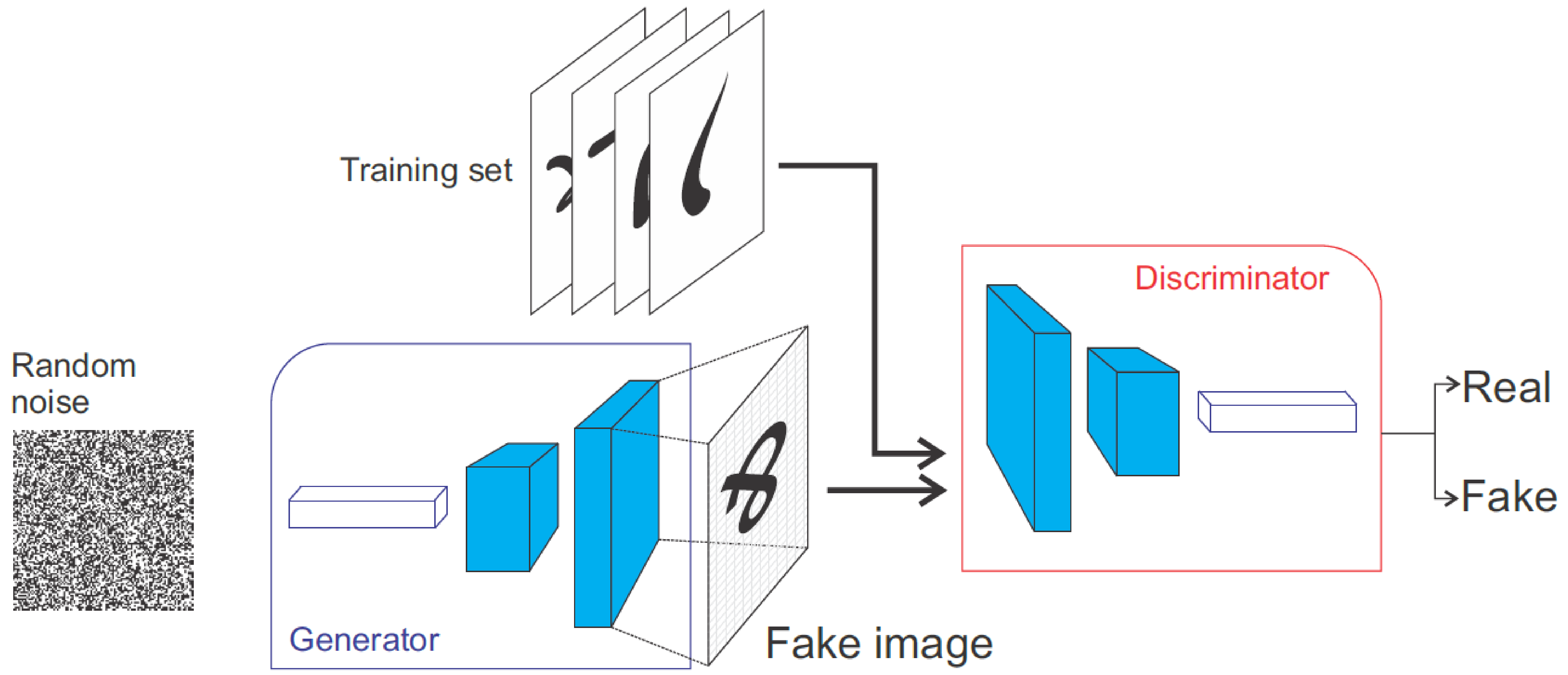
$$\mathcal{L}(\mathbf{w}, b) = - \sum_{i=1}^{N} [t_i \, \log y_i + (1 - t_i) \, \log(1 - y_i)]$$

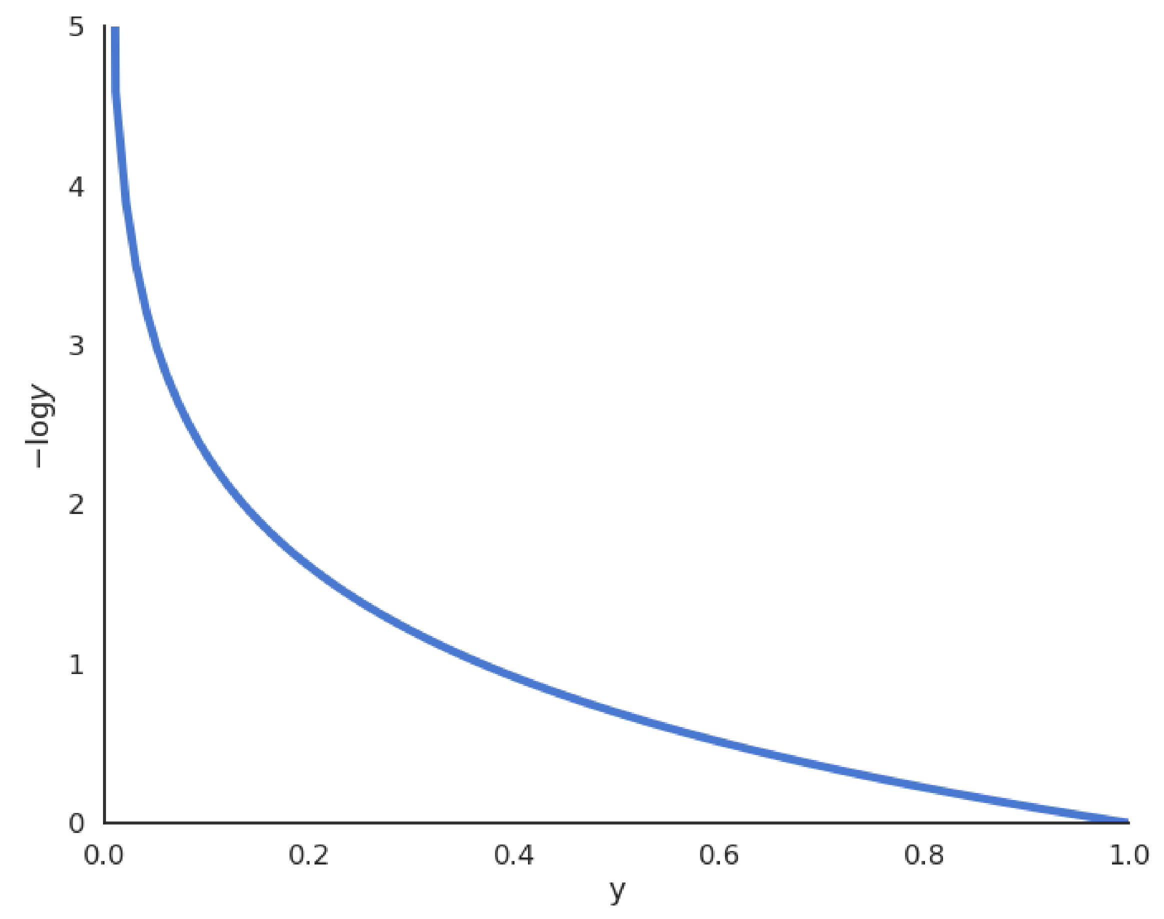
Loss of the generator

- The goal of the generator is to **maximize** the negative log-likelihood of the discriminator being correct on the generated images, i.e. fool it:

$$\mathcal{J}(G) = \mathbb{E}_{z \sim P_z(z)}[- \log(1 - D(G(z)))]$$

- The generator tries to maximize what the discriminator tries to minimize.





Source: https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml

GAN loss

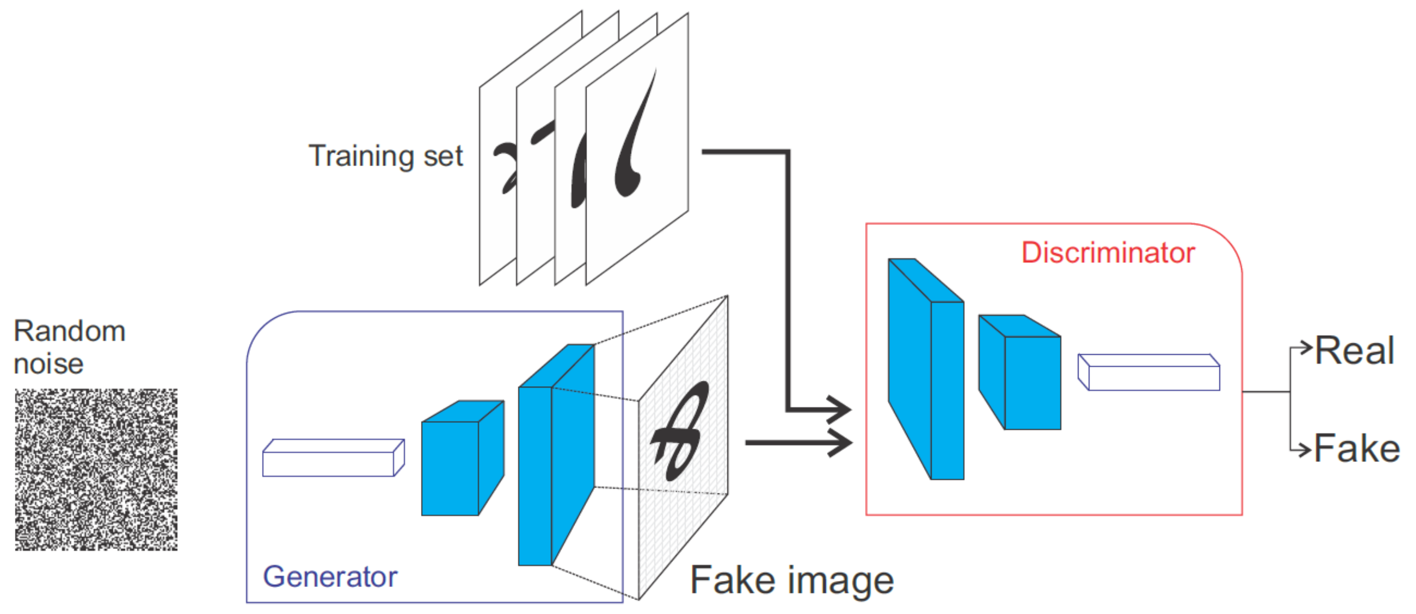
- Putting both objectives together, we obtain the following **minimax** problem:

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

- D and G compete on the same objective function: one tries to maximize it, the other to minimize it.
- Note that the generator G never sees the data x: all it gets is a **backpropagated gradient** through the discriminator:

$$\nabla_{G(z)} \mathcal{V}(D, G) = \nabla_{D(G(z))} \mathcal{V}(D, G) \times \nabla_{G(z)} D(G(z))$$

- It informs the generator which **pixels** are the most responsible for an eventual bad decision of the discriminator.



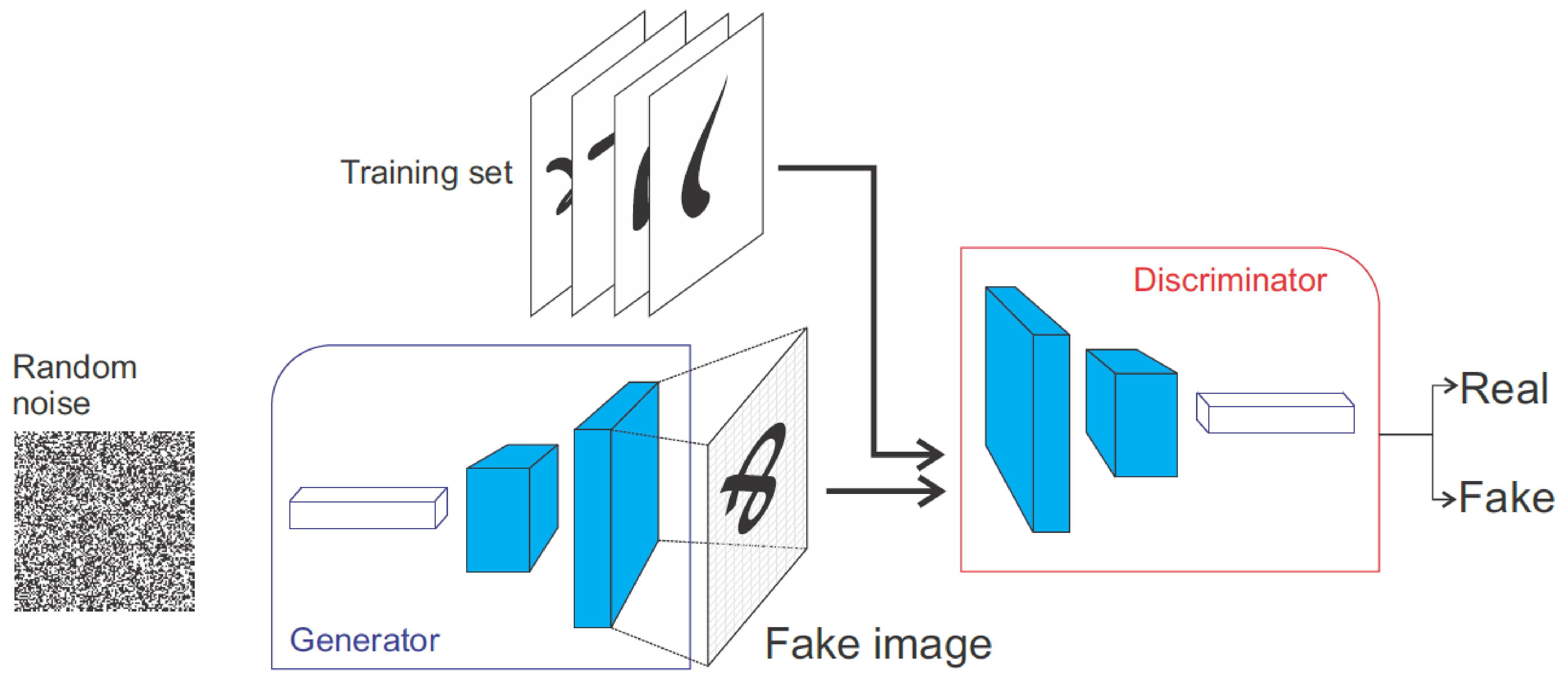
Source: https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml

GAN loss

- This objective function can be optimized when the generator uses gradient descent and the discriminator gradient ascent: just apply a minus sign on the weight updates!

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

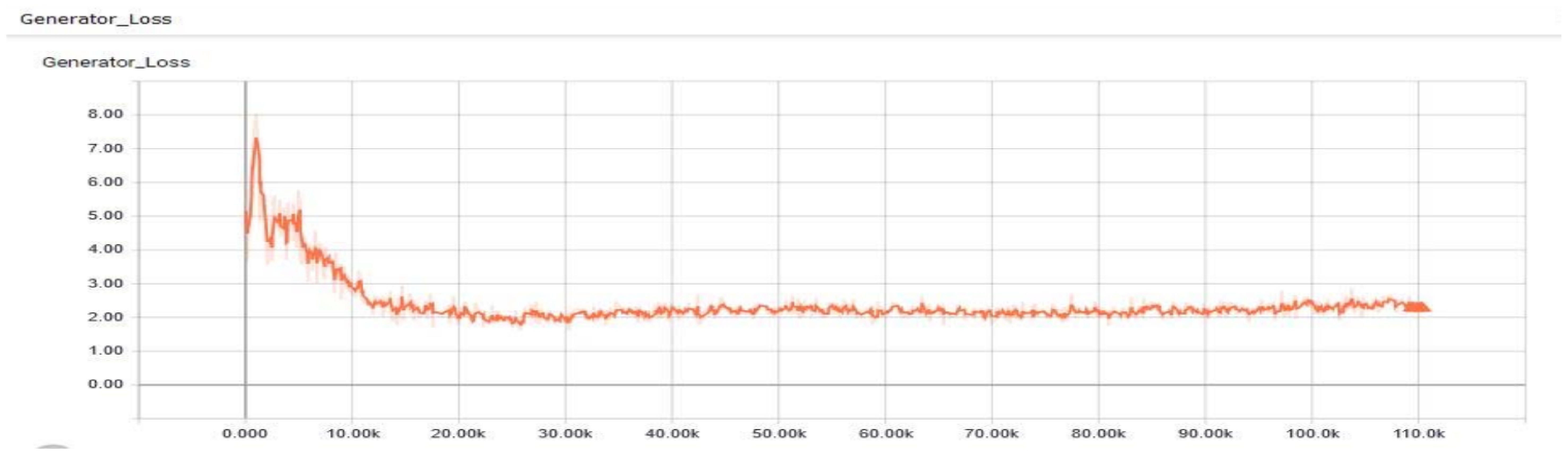
- Both can therefore use the usual **backpropagation** algorithm to adapt their parameters.
- The discriminator and the generator should reach a **Nash equilibrium**: they try to beat each other, but both become better over time.

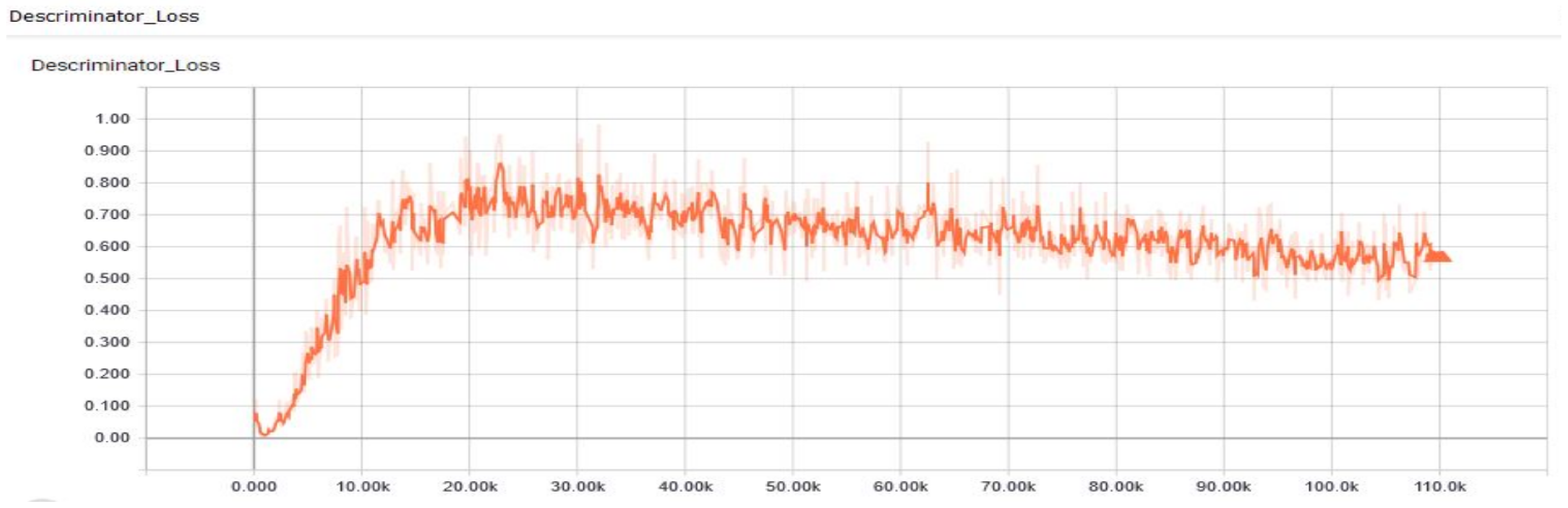


Source: https://www.oreilly.com/library/view/java-deep-learning/9781788997454/60579068-af4b-4bbf-83f1-e988fbe3b226.xhtml

Generative adversarial network

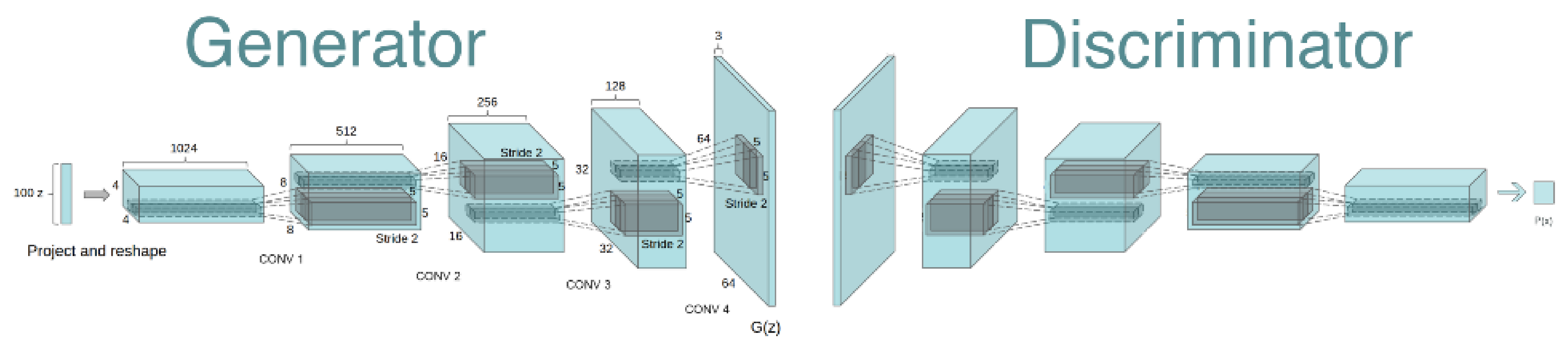
- The loss functions reach an equilibrium, it is quite hard to tell when the network has converged.





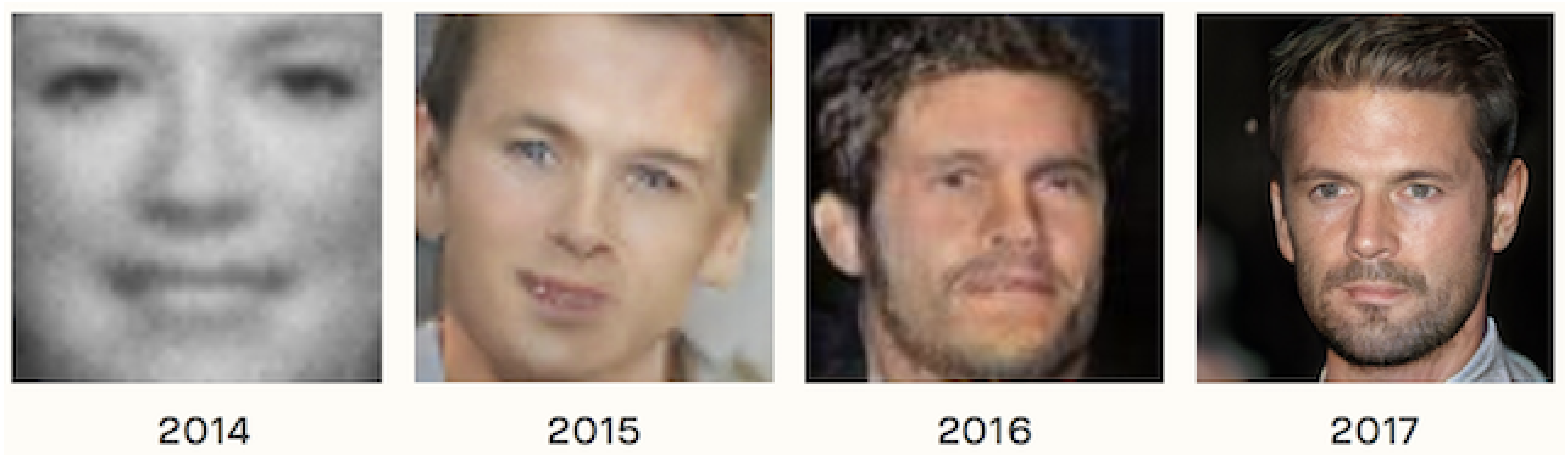
DCGAN : Deep convolutional GAN

- DCGAN is the convolutional version of GAN, using transposed convolutions in the generator and concolutions with stride in the discriminator.



Generative adversarial networks

- GAN are quite sensible to train: the discriminator should not become too good too early, otherwise there is no usable gradient for the generator.
- In practice, one updates the generator more often than the discriminator.
- There has been many improvements on GANs to stabilizes training:
  - Wasserstein GAN (relying on the Wasserstein distance instead of the log-likelihood).
  - f-GAN (relying on any f-divergence).
- But the generator often **collapses**, i.e. outputs always the same image regarless the input noise.
- Hyperparameter tuning is very difficult.



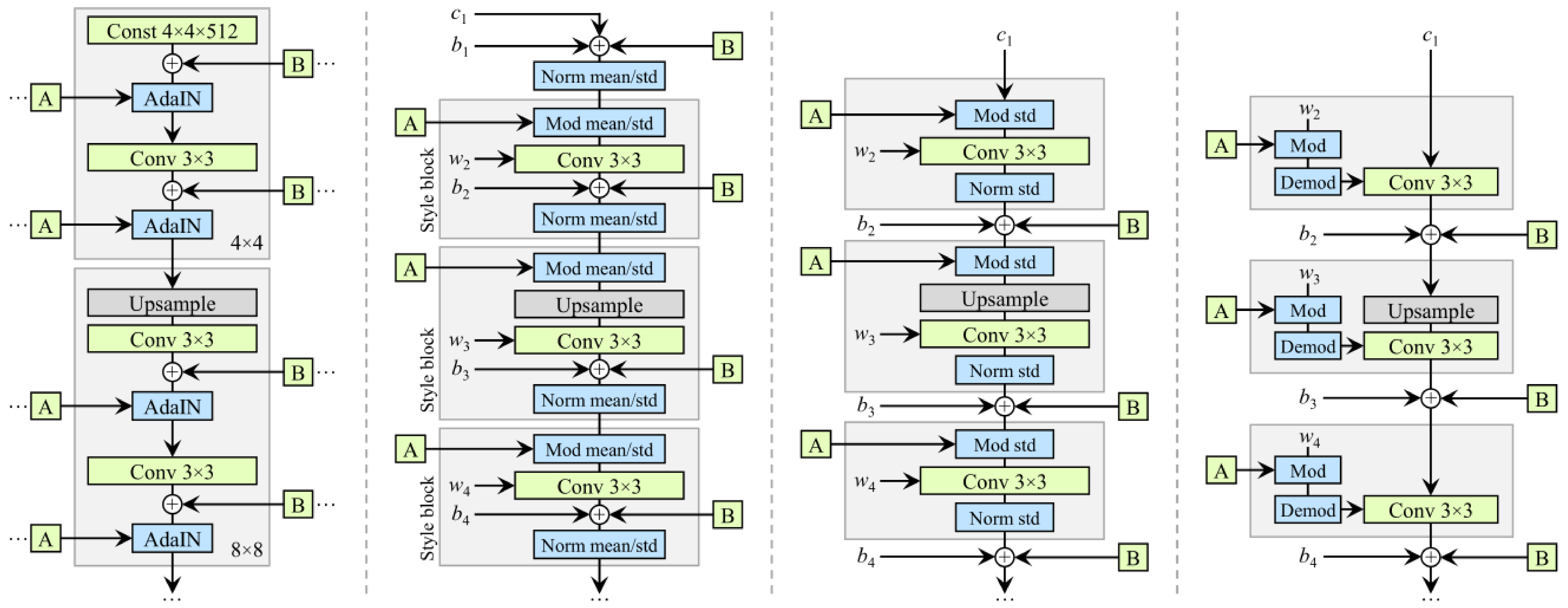
Source: Brundage et al. (2018). The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation. arXiv:180207228

Salimans et al. (2016). Improved techniques for training GANs. In Advances in Neural Information Processing Systems.

StyleGAN2

- StyleGAN2 from NVIDIA is one of the most realistic GAN variant. Check its generated faces at:

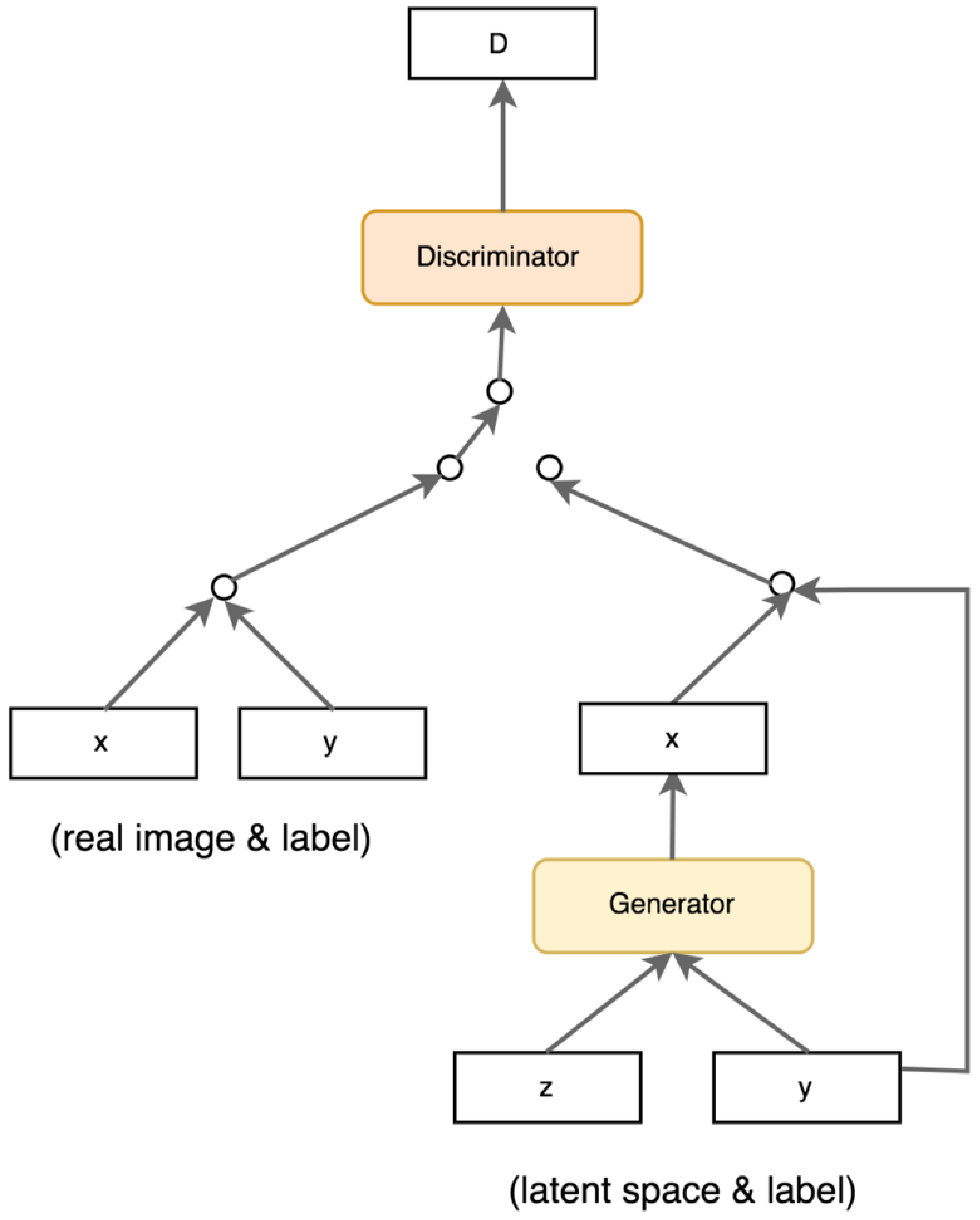
https://thispersondoesnotexist.com/



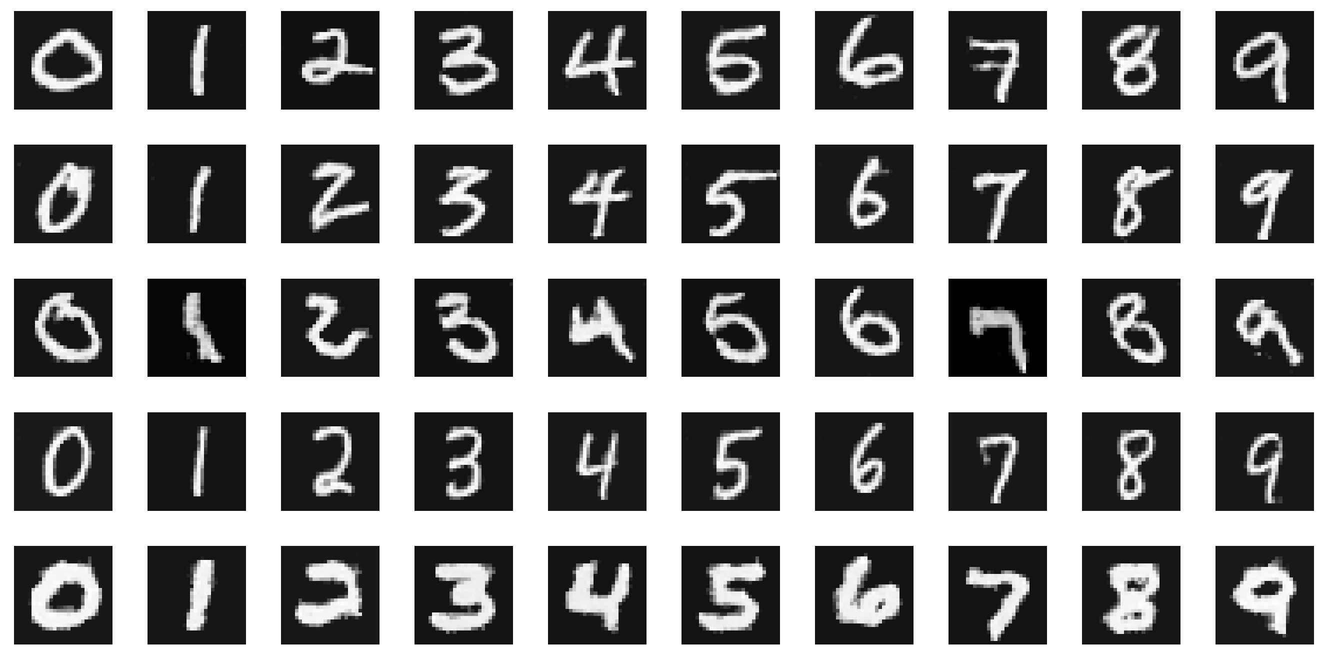
(a) StyleGAN (b) StyleGAN (detailed) (c) Revised architecture (d) Weight demodulation

2 - Conditional GANs

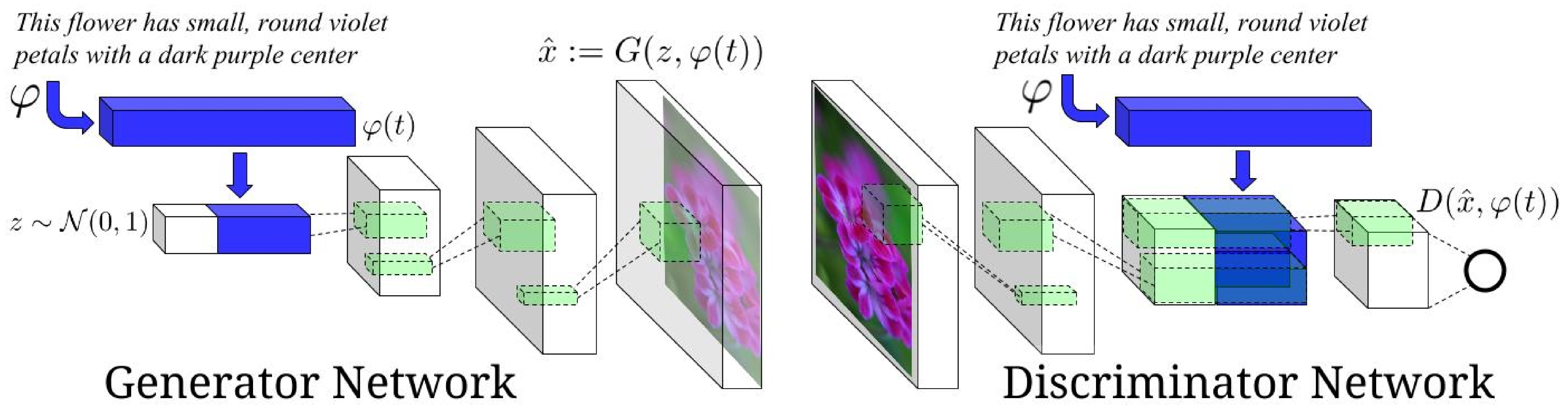
Conditional GAN (cGAN)

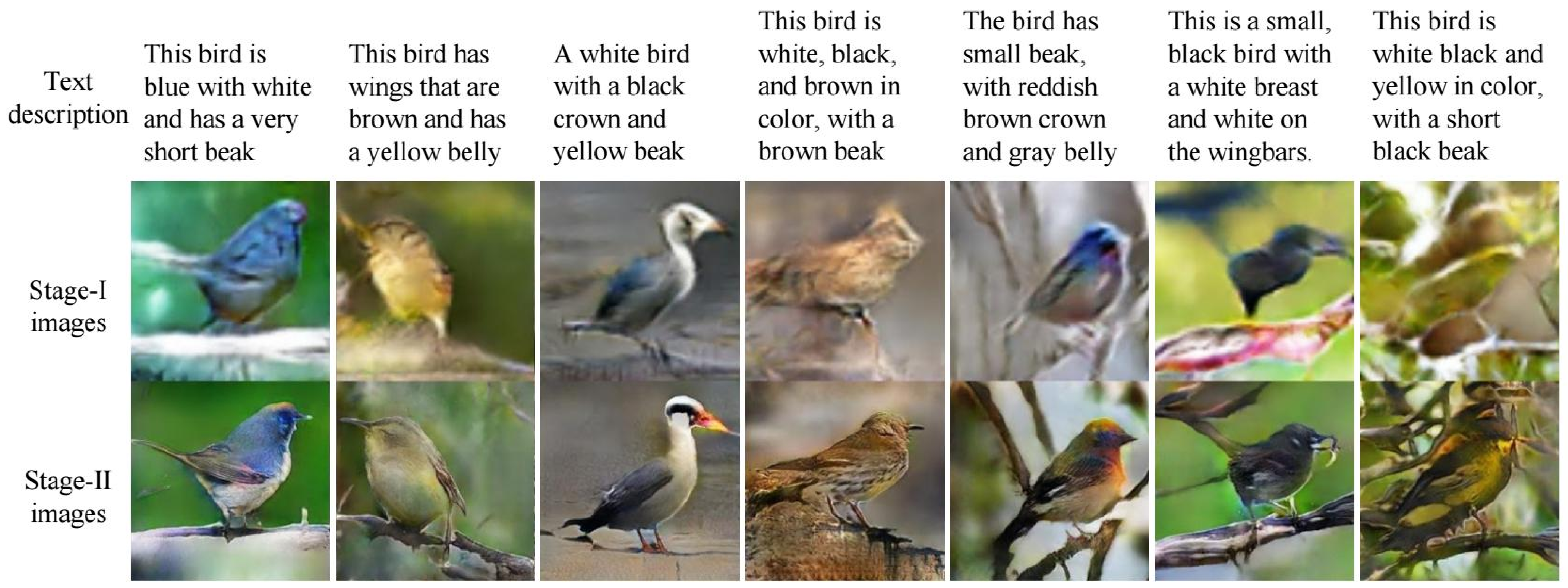


- The generator can also get additional **deterministic** information to the latent space, not only the random vector z.
- One can for example provide the **label** (class) in the context of supervised learning, allowing to generate many **new** examples of each class: data augmentation.
- One could also provide the output of a pre-trained CNN (ResNet) to condition on images.



cGAN: text-to-image synthesis

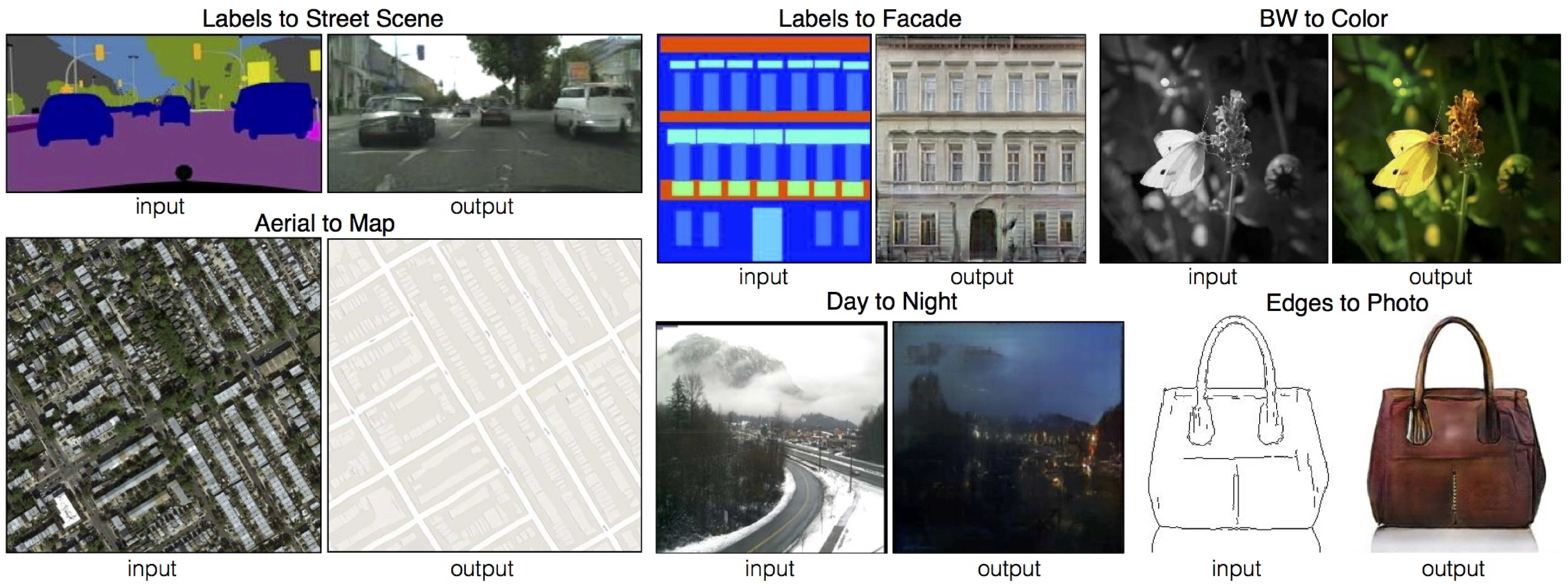

This flower has small, round violet petals with a dark purple center
φ
$\varphi(t)$
$z \sim \mathcal{N}(0,1)$
$\hat{x} := G(z, \varphi(t))$
This flower has small, round violet petals with a dark purple center
φ
$D(\hat{x}, \varphi(t))$
Generator Network
Discriminator Network


Text description
This bird is blue with white and has a very short beak
This bird has wings that are brown and has a yellow belly
A white bird with a black crown and yellow beak
This bird is white, black, and brown in color, with a brown beak
The bird has small beak, with reddish brown crown and gray belly
This is a small, black bird with a white breast and white on the wingbars.
This bird is white black and yellow in color, with a short black beak
Stage-I images
Stage-II images

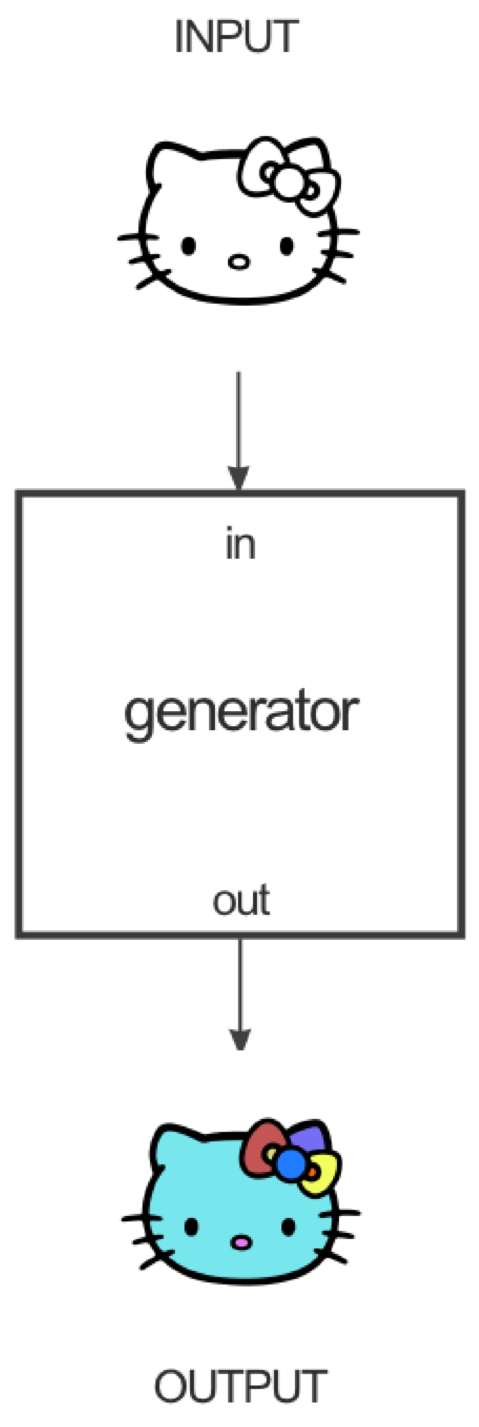
Source: Reed et al. (2016). Generative Adversarial Text to Image Synthesis. arXiv:1605.05396

pix2pix: image-to-image translation

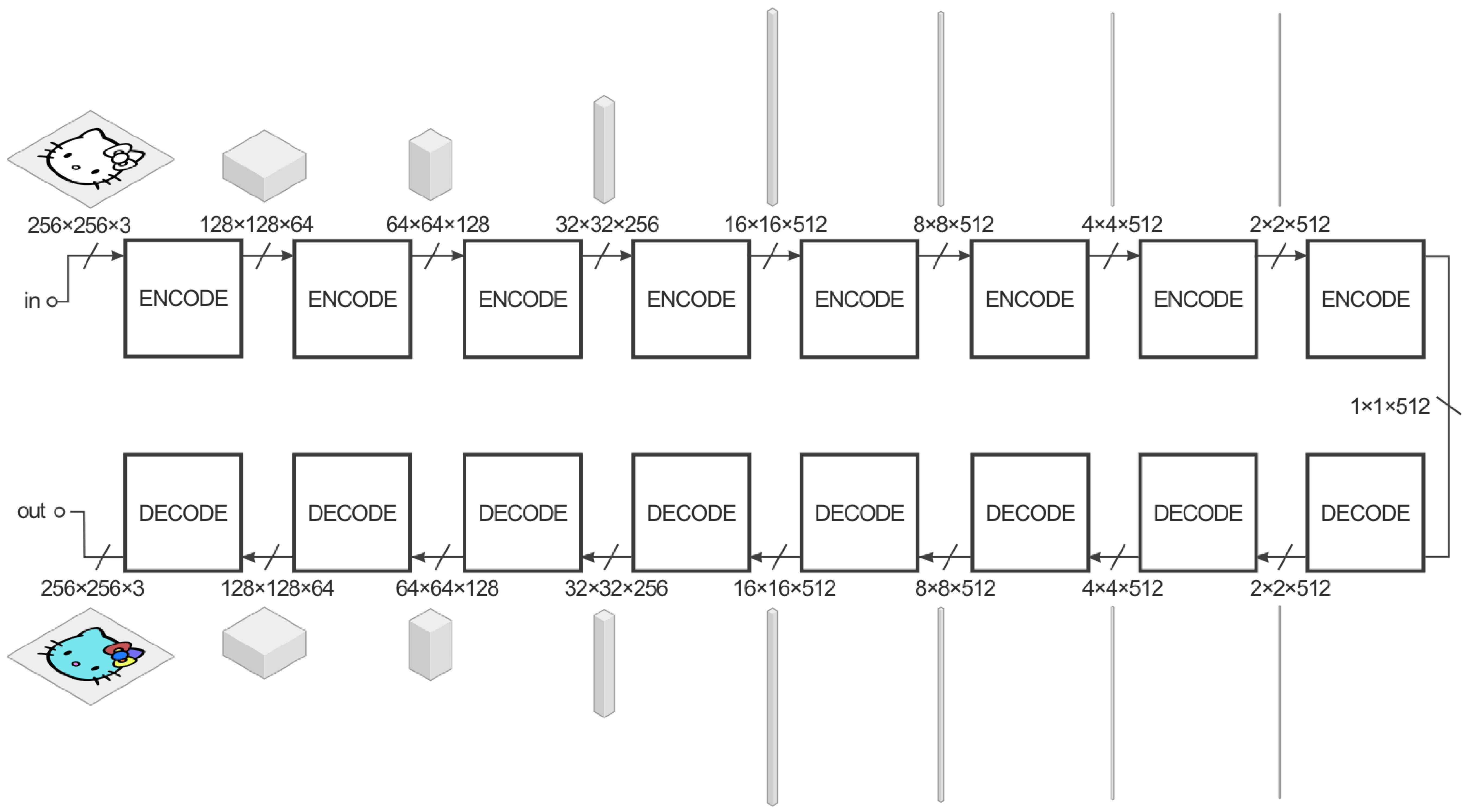
- cGAN can be extended to have an autoencoder-like architecture, allowing to generate images from images.
- **pix2pix** is trained on pairs of similar images in different domains. The conversion from one domain to another is easy in one direction, but we want to learn the opposite.



pix2pix: image-to-image translation

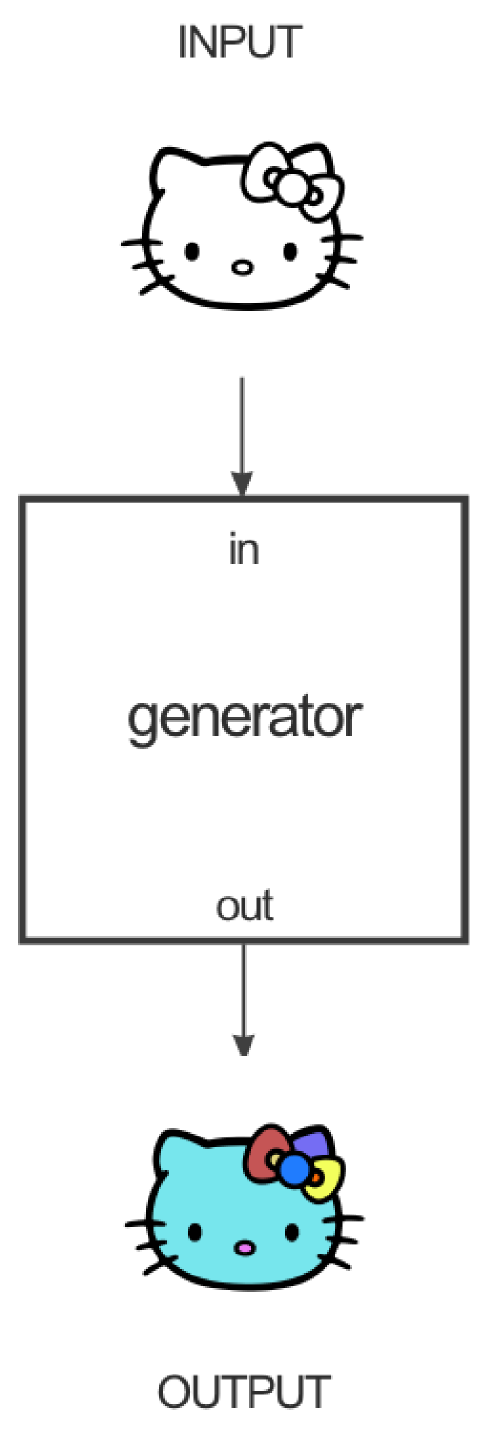


- The goal of the generator is to convert for example a black-and-white image into a colorized one.
- It is a deep convolutional autoencoder, with convolutions with strides and transposed convolutions (SegNet-like).

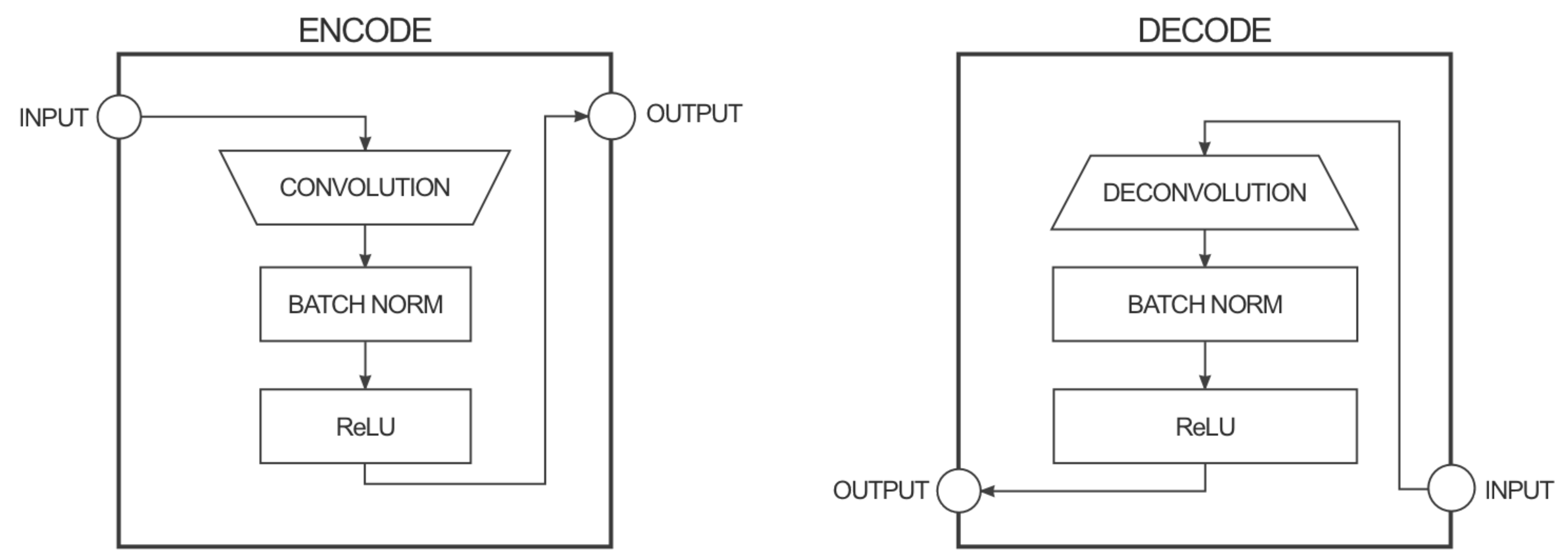


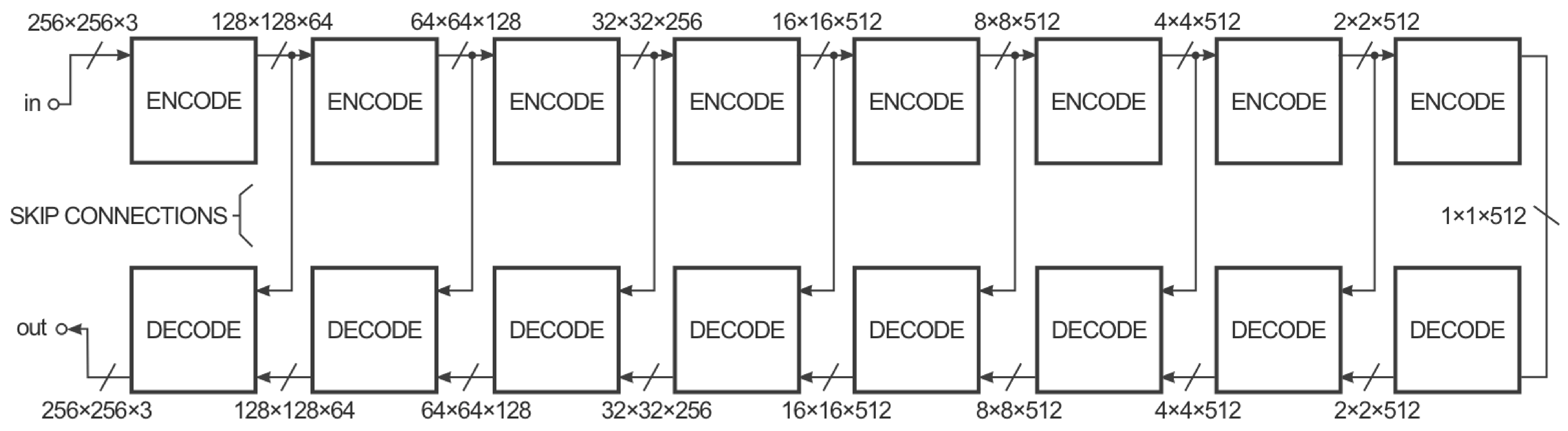
Source: https://affinelayer.com/pix2pix/

pix2pix: image-to-image translation



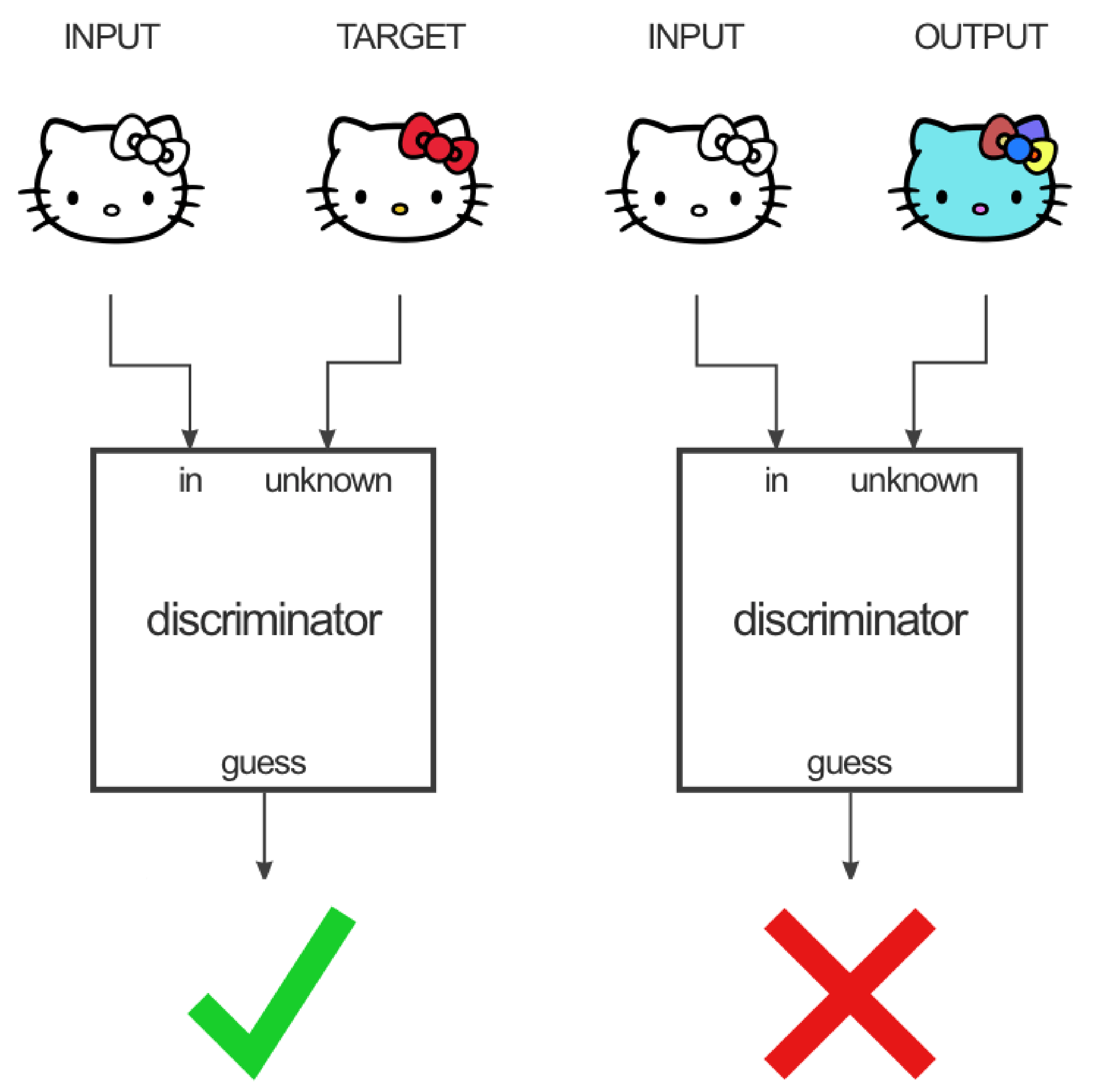
- In practice, it has a **U-Net** architecture with skip connections to generate fine details.



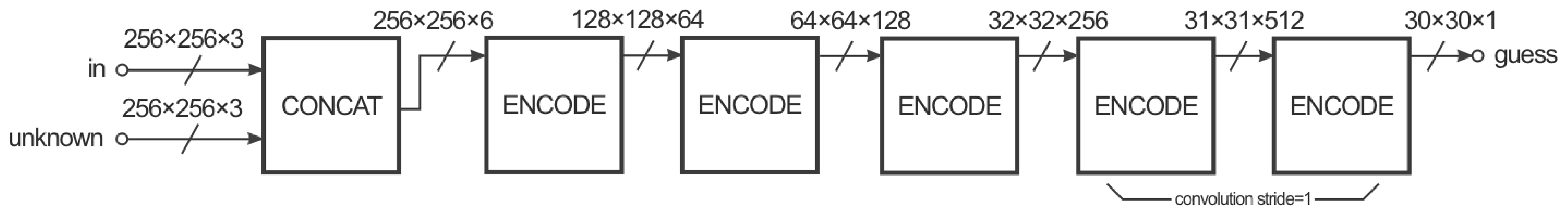


Source: https://affinelayer.com/pix2pix/

pix2pix: image-to-image translation



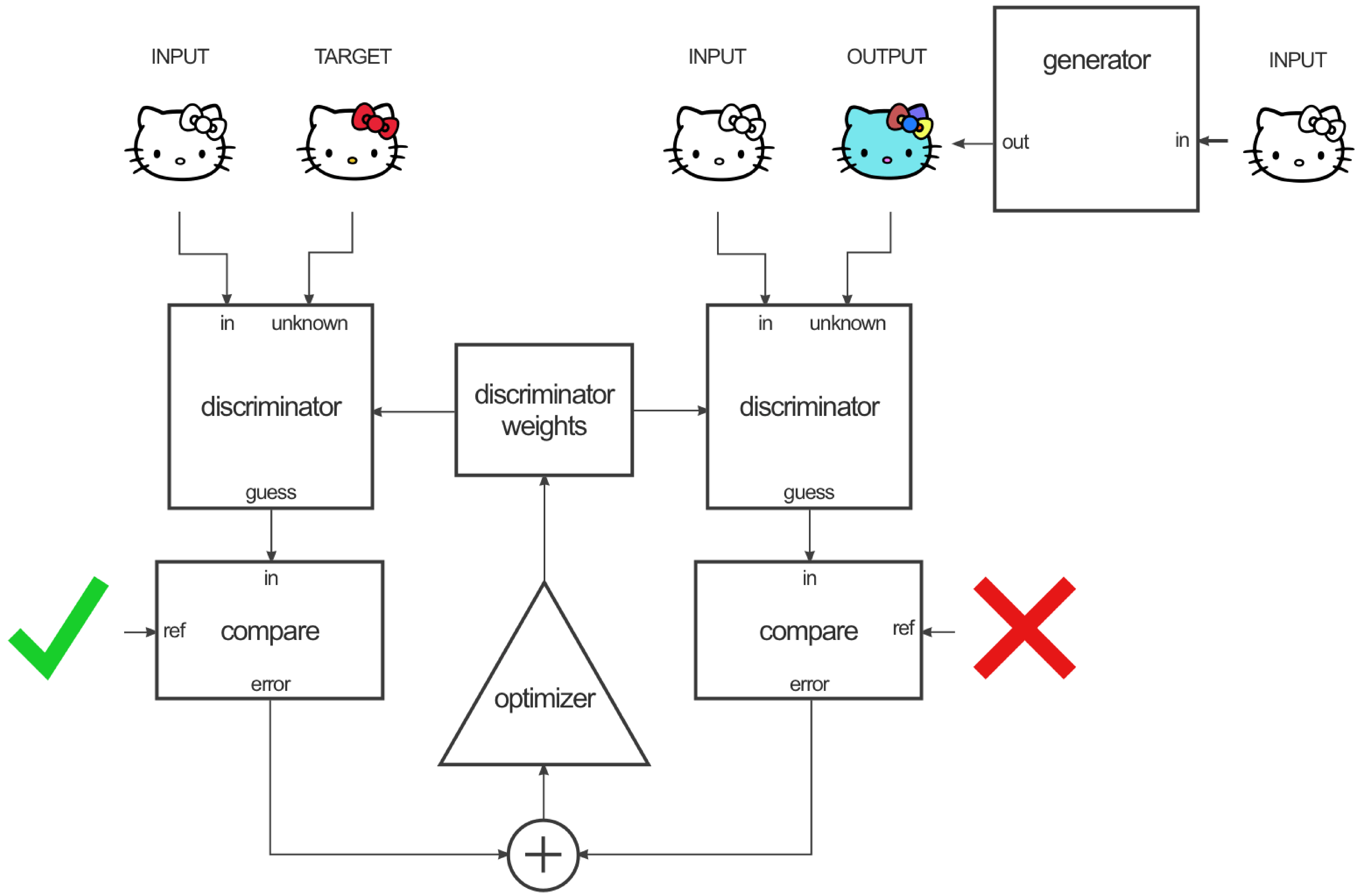
- The discriminator takes a **pair** of images as input: input/target or input/generated.
- It does not output a single value real/fake, but a 30x30 “image” telling how real or fake is the corresponding **patch** of the unknown image.
- Patches correspond to overlapping 70x70 regions of the 256x256 input image.
- This type of discriminator is called a **PatchGAN**.



Source: https://affinelayer.com/pix2pix/

pix2pix: image-to-image translation

- The discriminator is trained like in a regular GAN by alternating input/target or input/generated pairs.

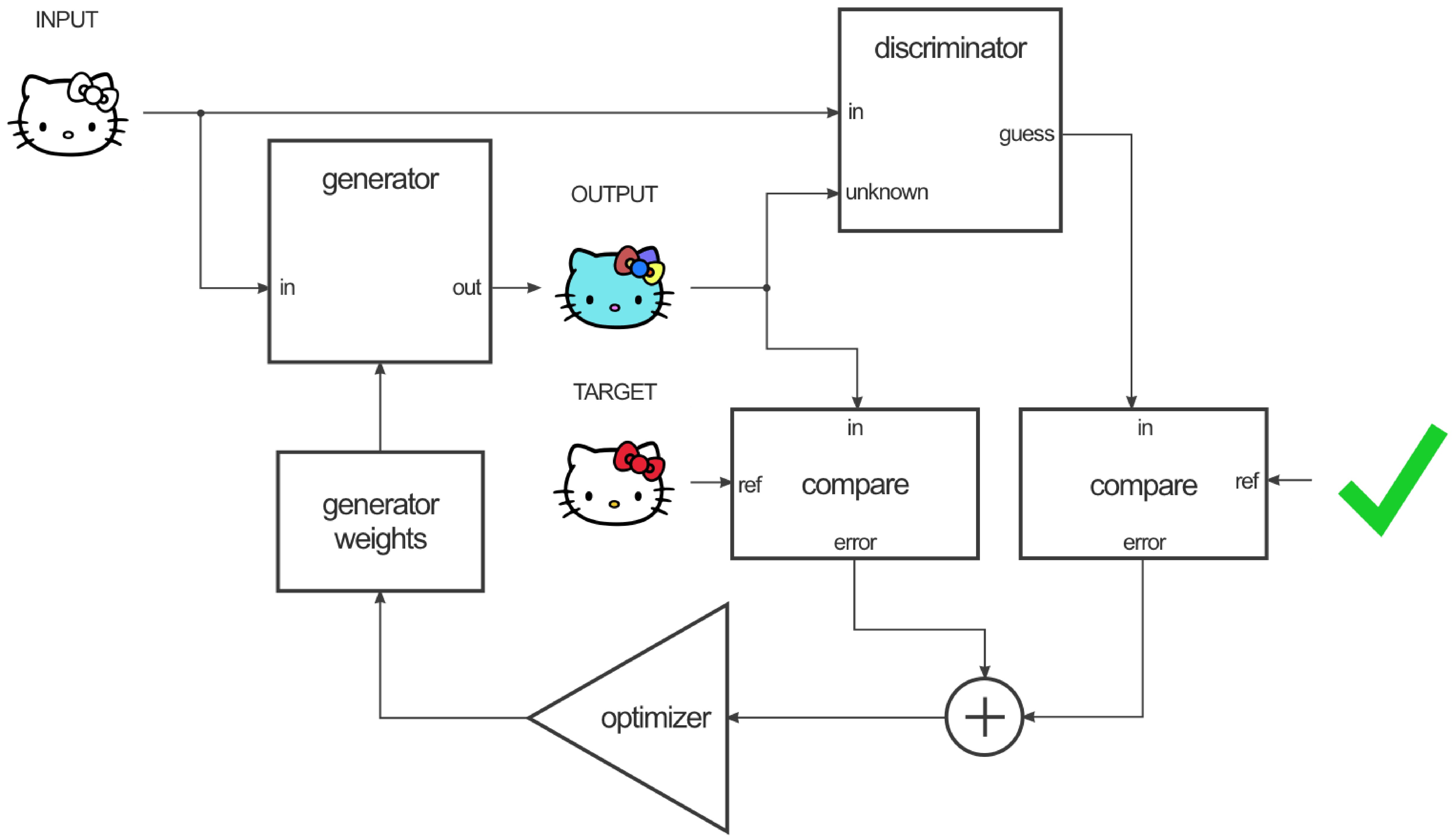


Source: https://affinelayer.com/pix2pix/

pix2pix: image-to-image translation

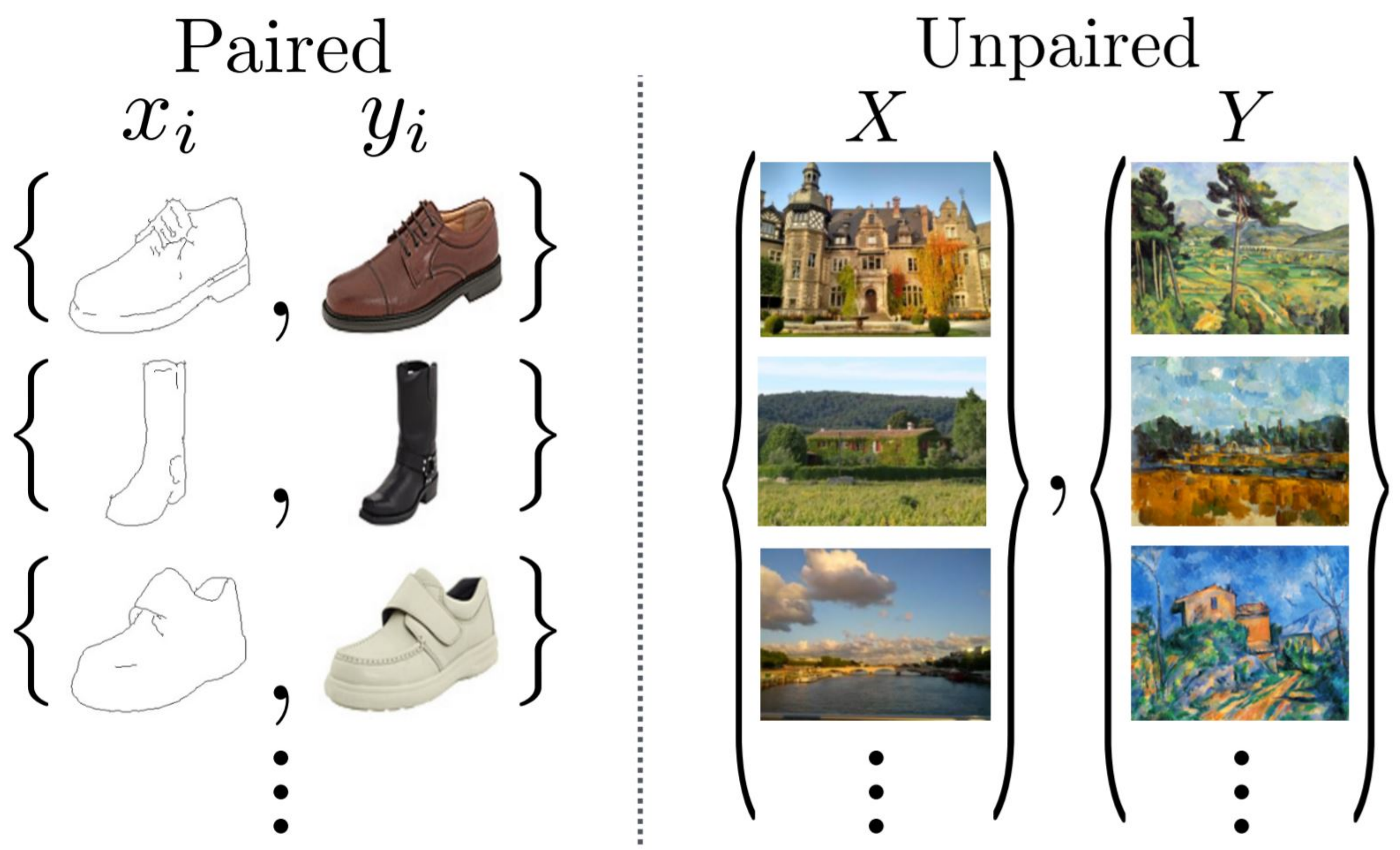
- The generator is trained by maximizing the GAN loss (using gradients backpropagated through the discriminator) but also by minimizing the L1 distance between the generated image and the target (supervised learning).

$$\min_G \max_D V(D,G) = V_{\text{GAN}}(D,G) + \lambda \mathbb{E}_\mathcal{D}[|T - G|]$$



Source: https://affinelayer.com/pix2pix/

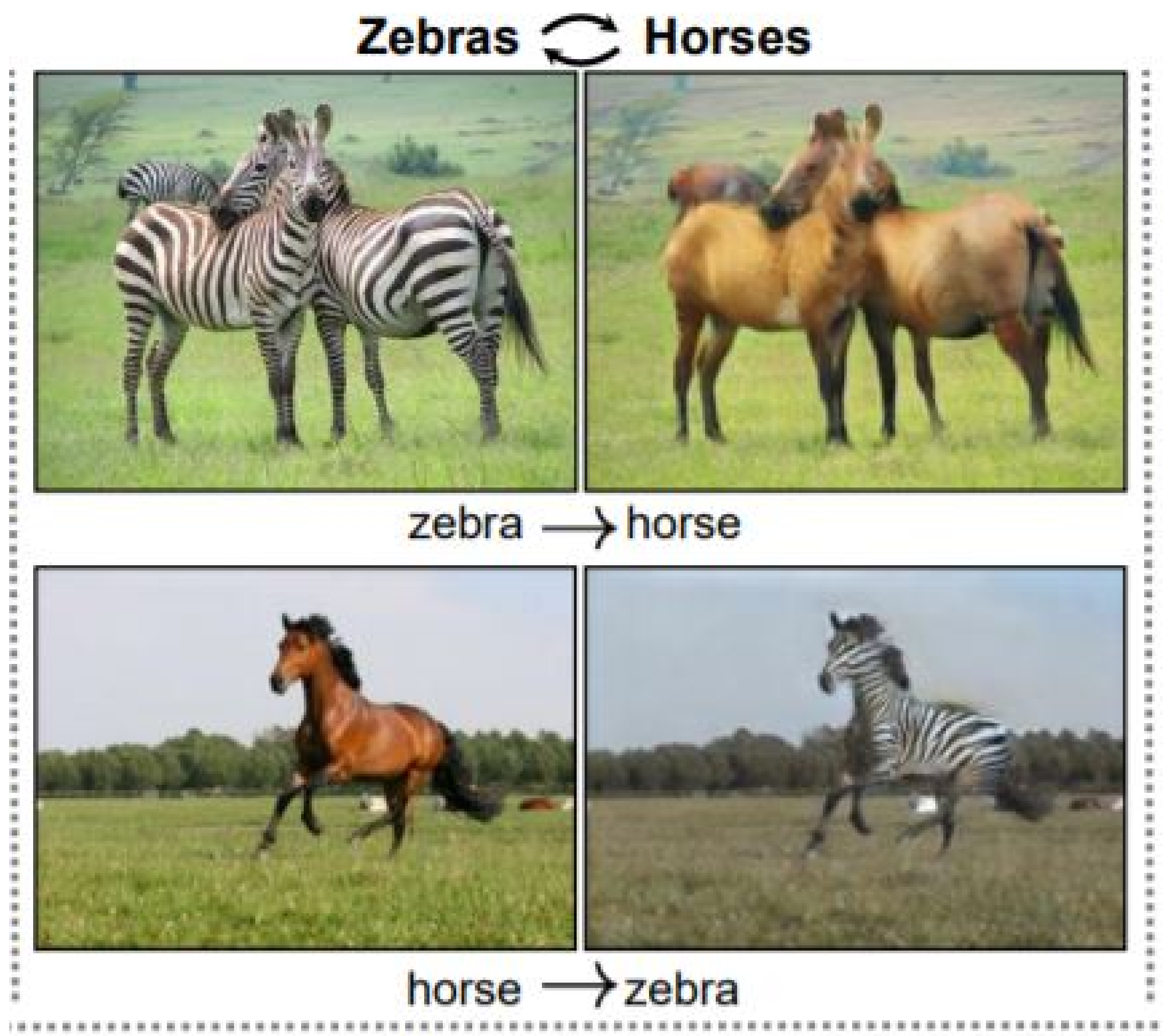
CycleGAN : Neural Style Transfer



- The drawback of pix2pix is that you need **paired** examples of each domain, which is sometimes difficult to obtain.
- In **style transfer**, we are interested in converting images using unpaired datasets, for example realistic photographies and paintings.
- **CycleGAN** is a GAN architecture for neural style transfer.

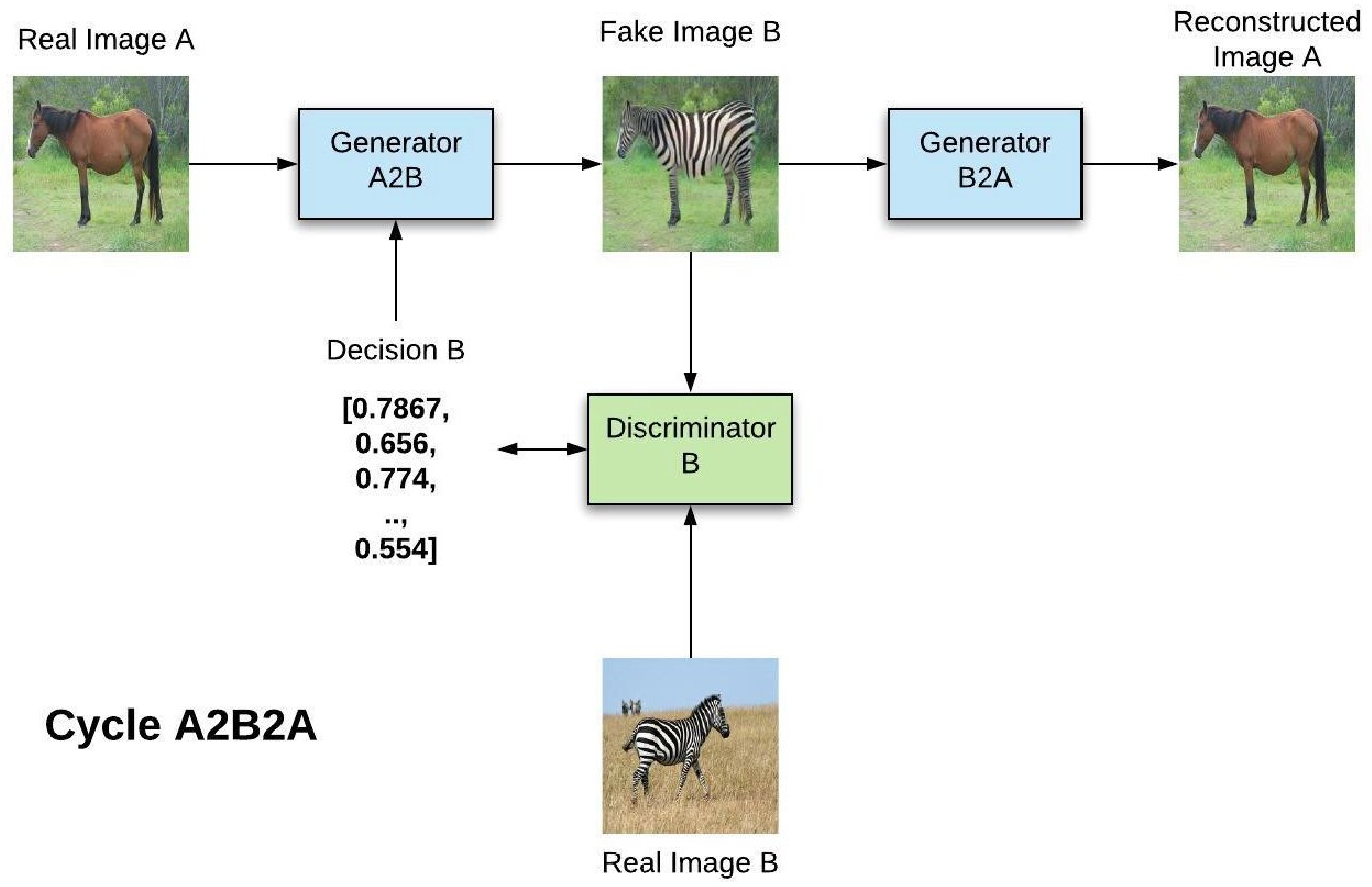
Source: https://hardikbansal.github.io/CycleGANBlog/

CycleGAN : Neural Style Transfer



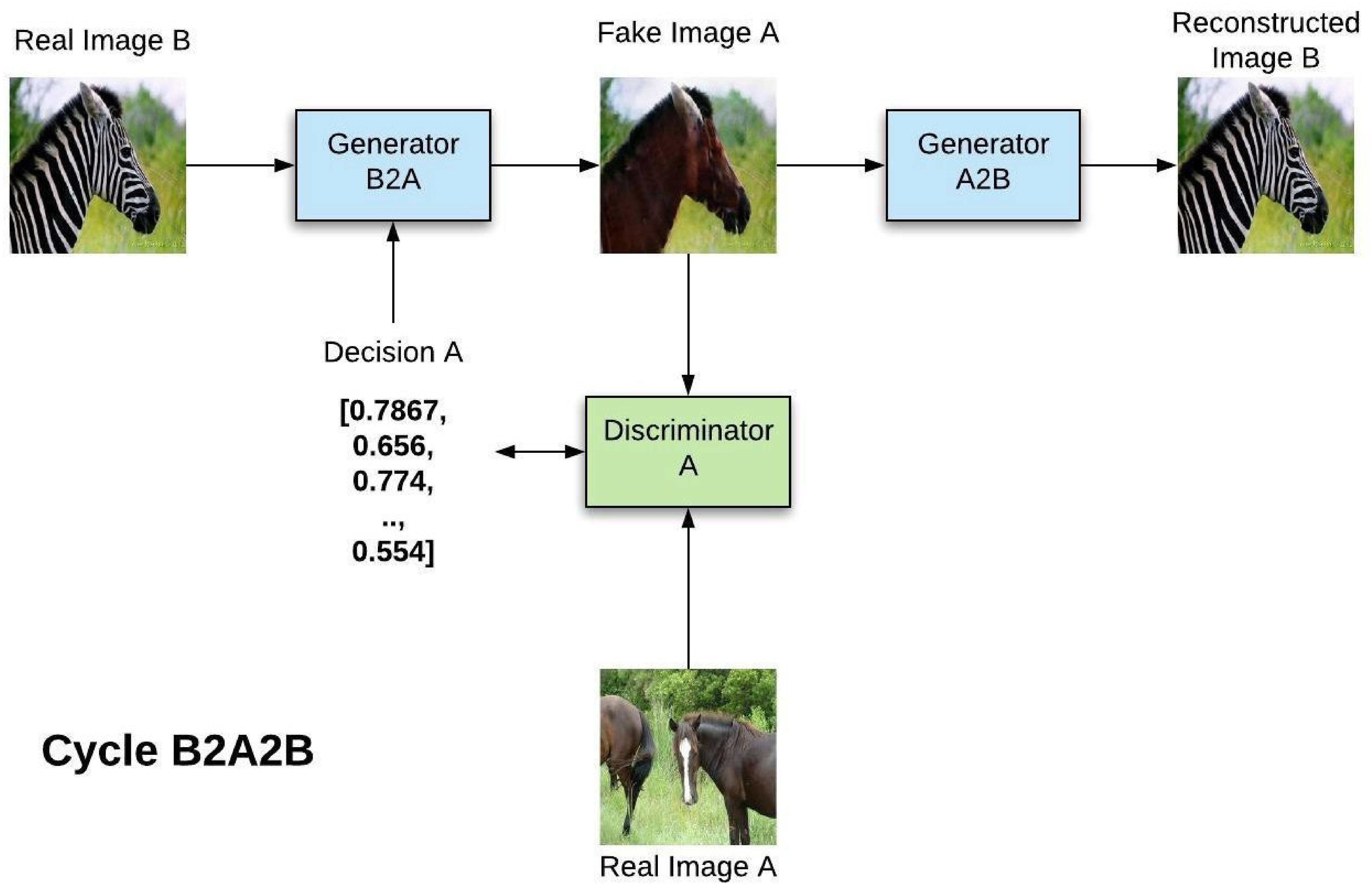
- Let's suppose that we want to transform **domain A** (horses) into **domain B** (zebras) or the other way around.
- The problem is that the two datasets are not paired, so we cannot provide targets to pix2pix (supervised learning).
- If we just select any zebra target for a horse input, pix2pix would learn to generate zebras that do not correspond to the input horse (the shape may be lost).
- How about we train a second GAN to generate the target?

CycleGAN : Neural Style Transfer



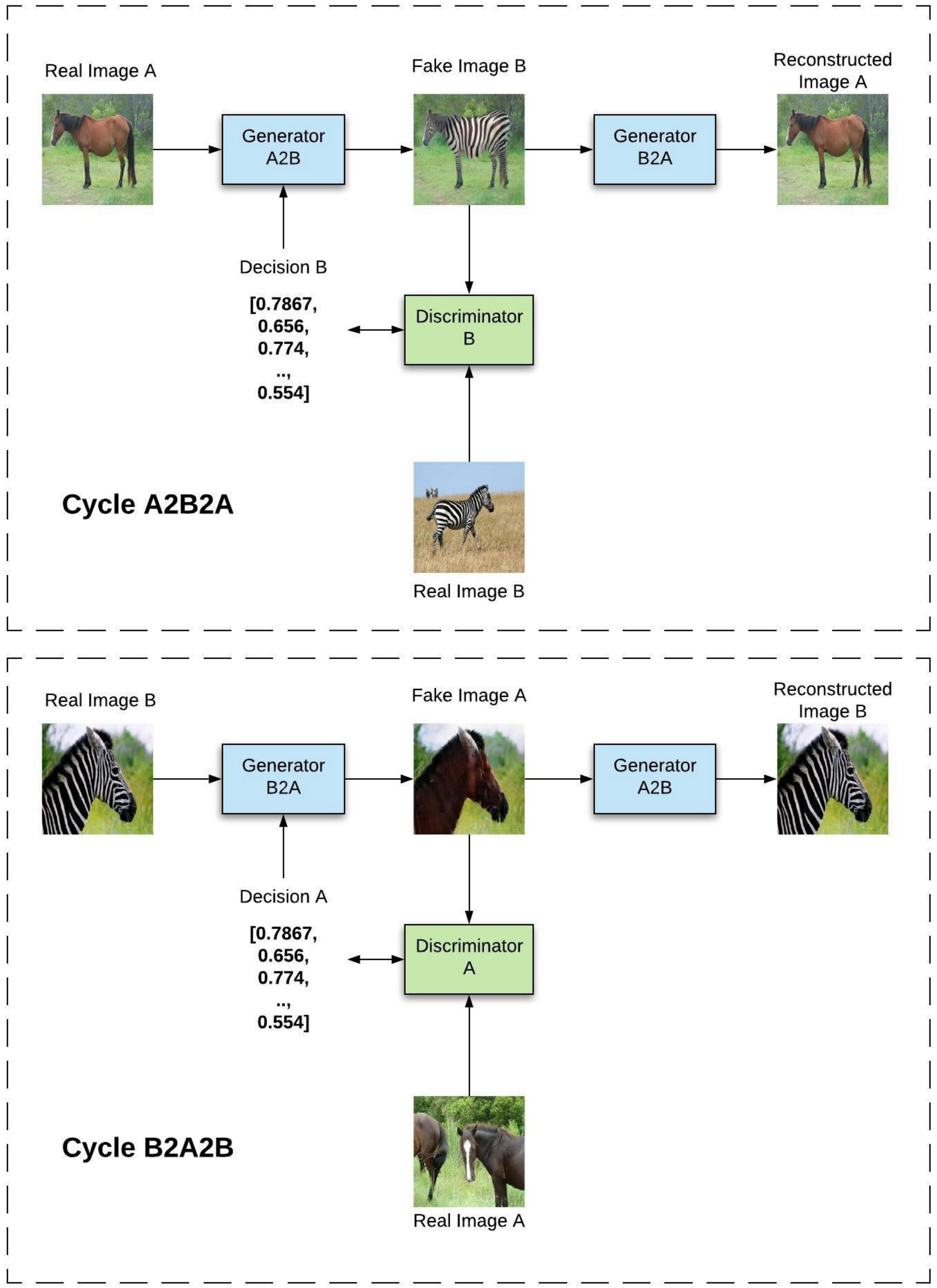
Source: https://towardsdatascience.com/gender-swap-and-cyclegan-in-tensorflow-2-0-359fe74ab7ff

CycleGAN : Neural Style Transfer



Source: https://towardsdatascience.com/gender-swap-and-cyclegan-in-tensorflow-2-0-359fe74ab7ff

CycleGAN : Neural Style Transfer



Source: https://towardsdatascience.com/gender-swap-and-cyclegan-in-tensorflow-2-0-359fe74ab7ff

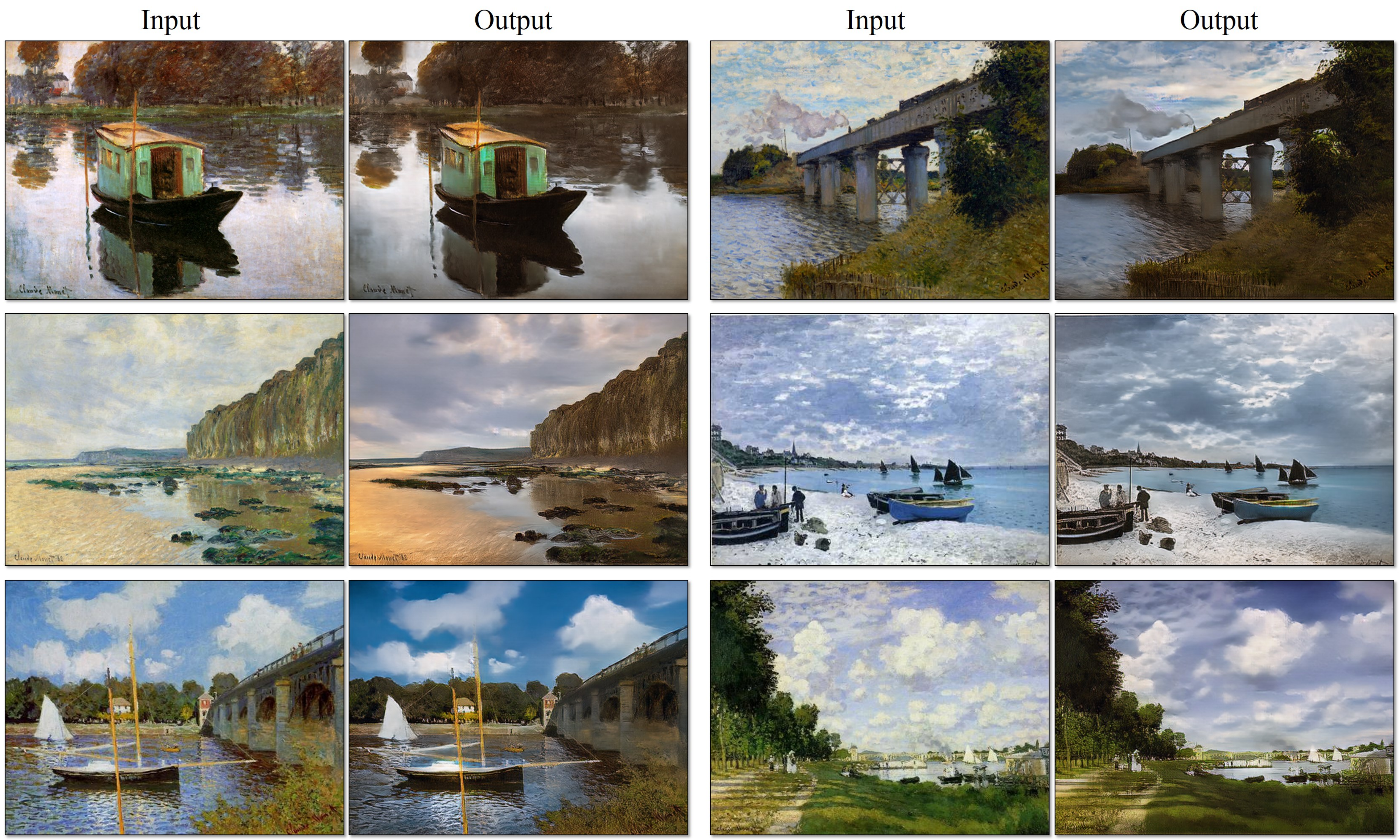
Cycle A2B2A

- The A2B generator generates a sample of B from an image of A.
- The B discriminator allows to train A2B using real images of B.
- The B2A generator generates a sample of A from the output of A2B, which can be used to minimize the L1-reconstruction loss (shape-preserving).

Cycle B2A2B

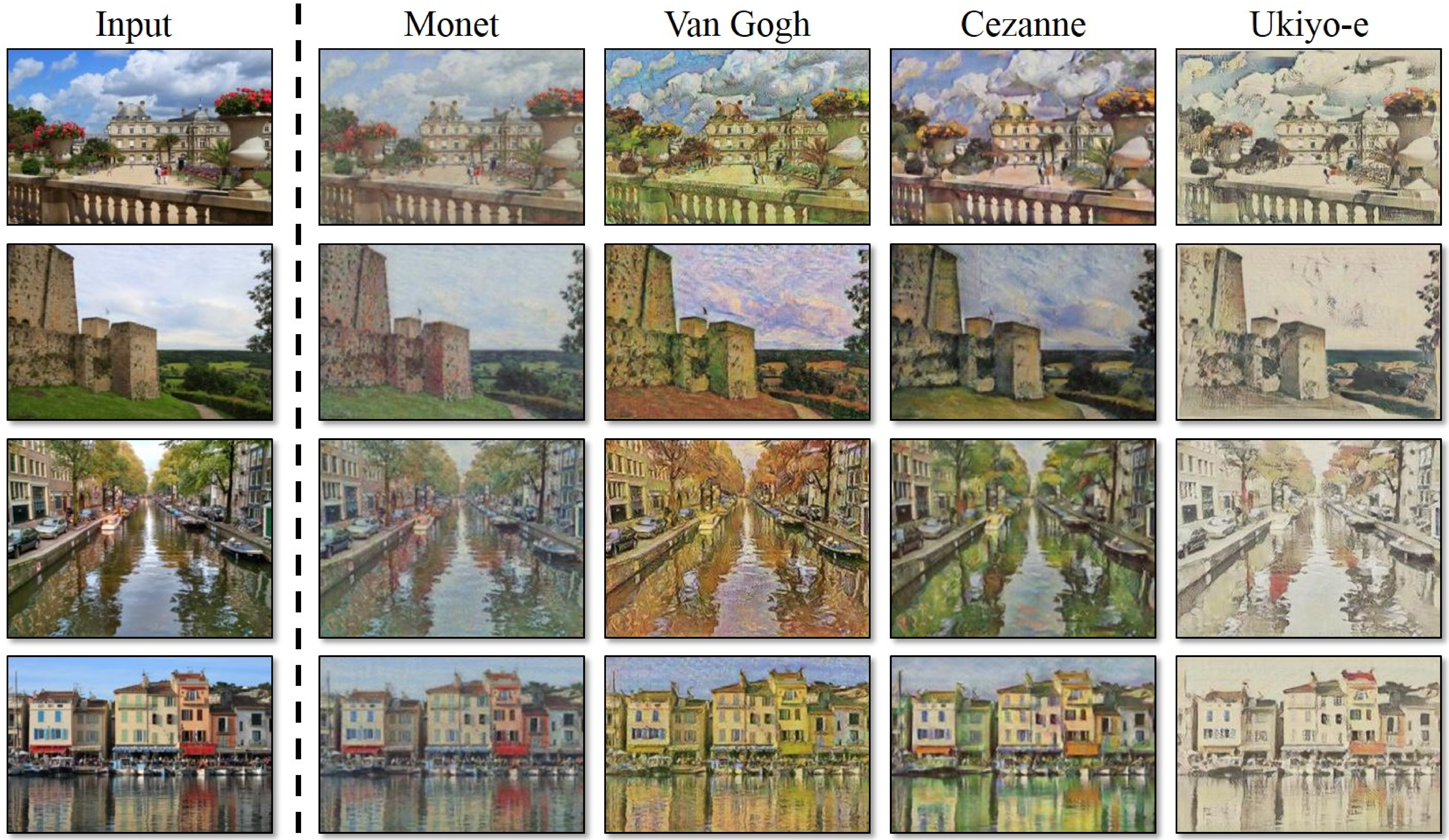
- In the B2A2B cycle, the domains are reversed, what allows to train the A discriminator.
- This cycle is repeated throughout training, allowing to train both GANS concurrently.

CycleGAN : Neural Style Transfer



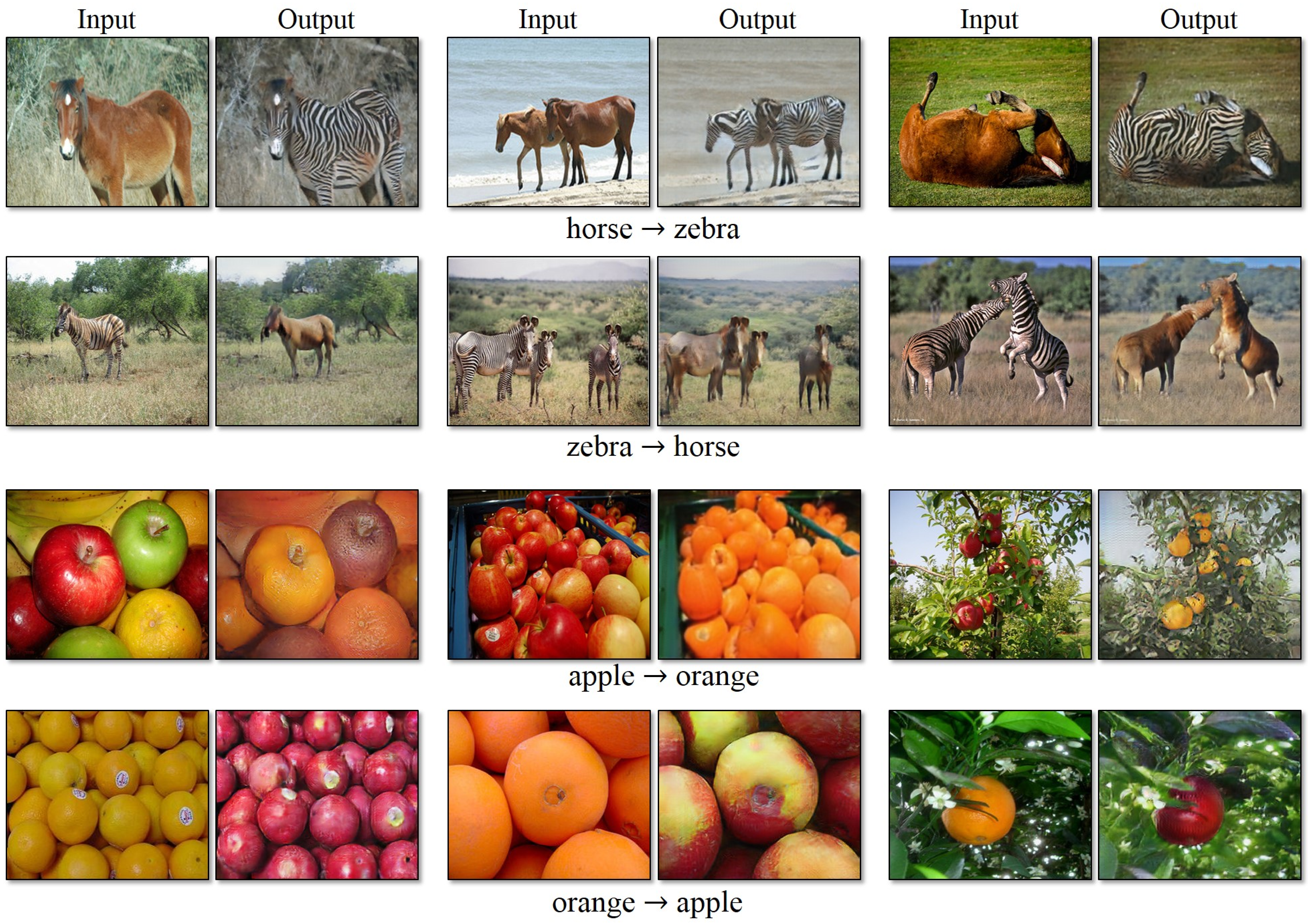
Source: https://github.com/junyanz/CycleGAN

CycleGAN : Neural Style Transfer



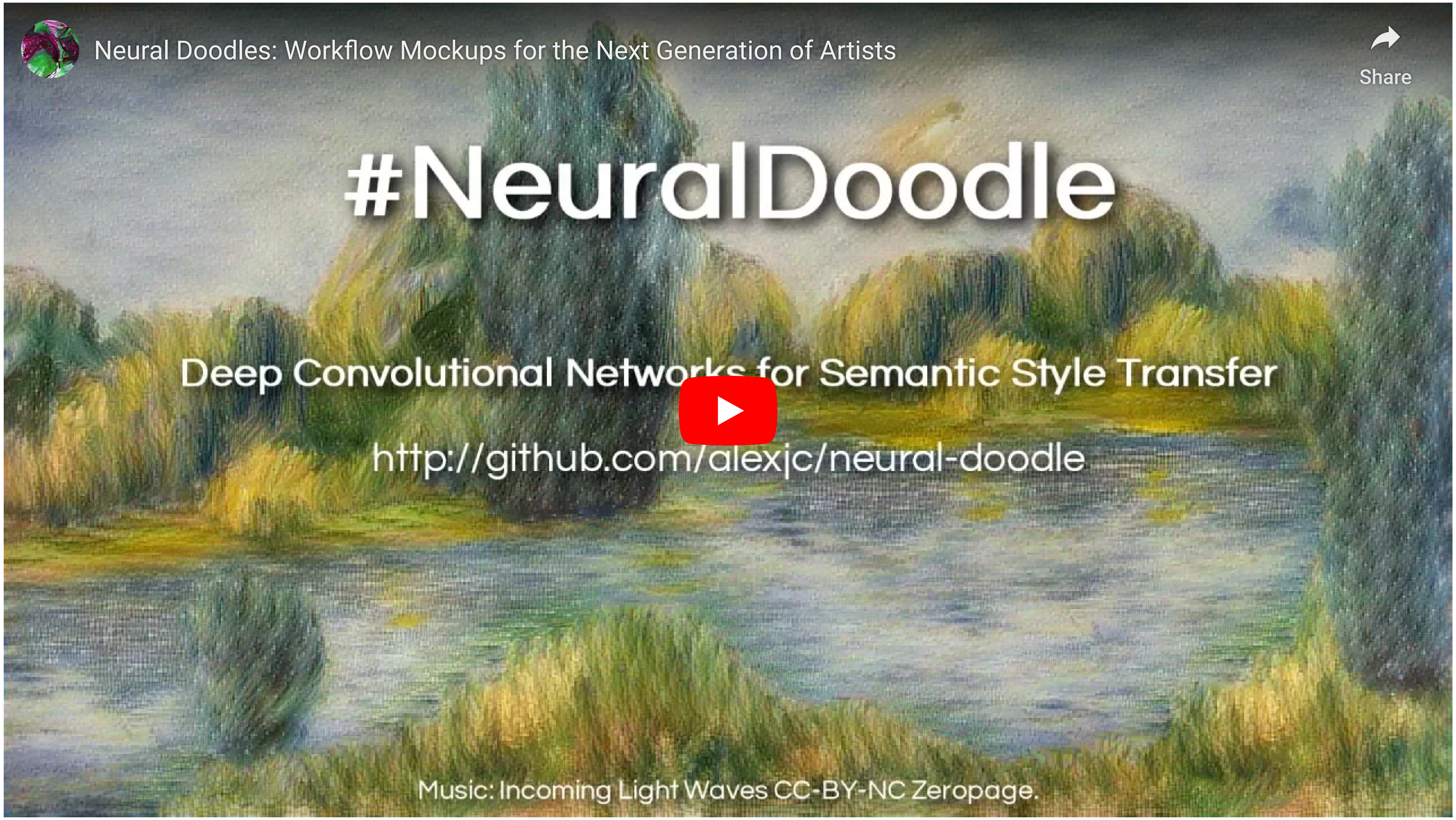
Source: https://github.com/junyanz/CycleGAN

CycleGAN : Neural Style Transfer



Source: https://github.com/junyanz/CycleGAN

Neural Doodle


Neural Doodles: Workflow Mockups for the Next Generation of Artists
Share
#NeuralDoodle
Deep Convolutional Networks for Semantic Style Transfer
http://github.com/alexjc/neural-doodle
Music: Incoming Light Waves CC-BY-NC Zeropage.