UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

Sesión 5.0

Generative adversarial network

WGAN, cGAN, StyleGAN

TRANSFORMATEC

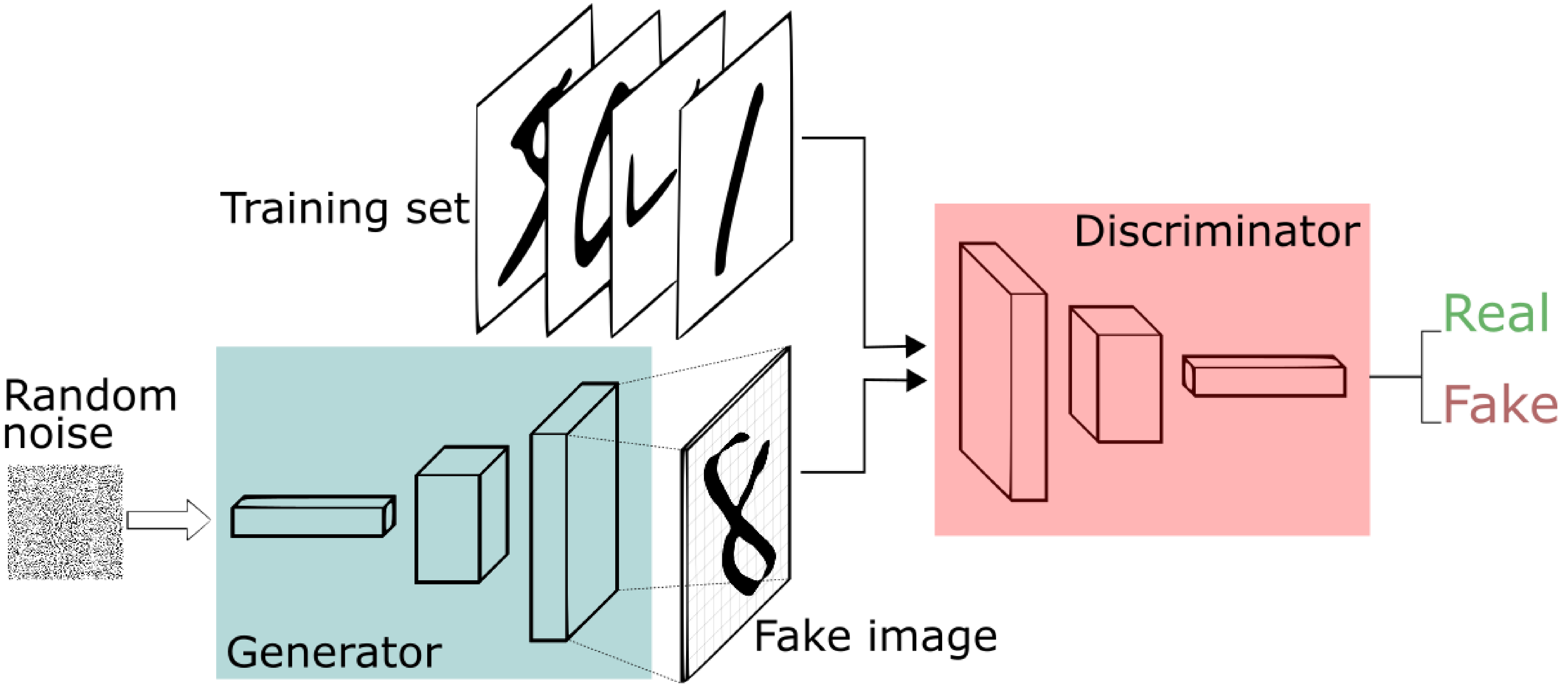
UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

1.

Generative Adversarial *Network (GAN)*

TRANSFORMATEC

GAN



Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

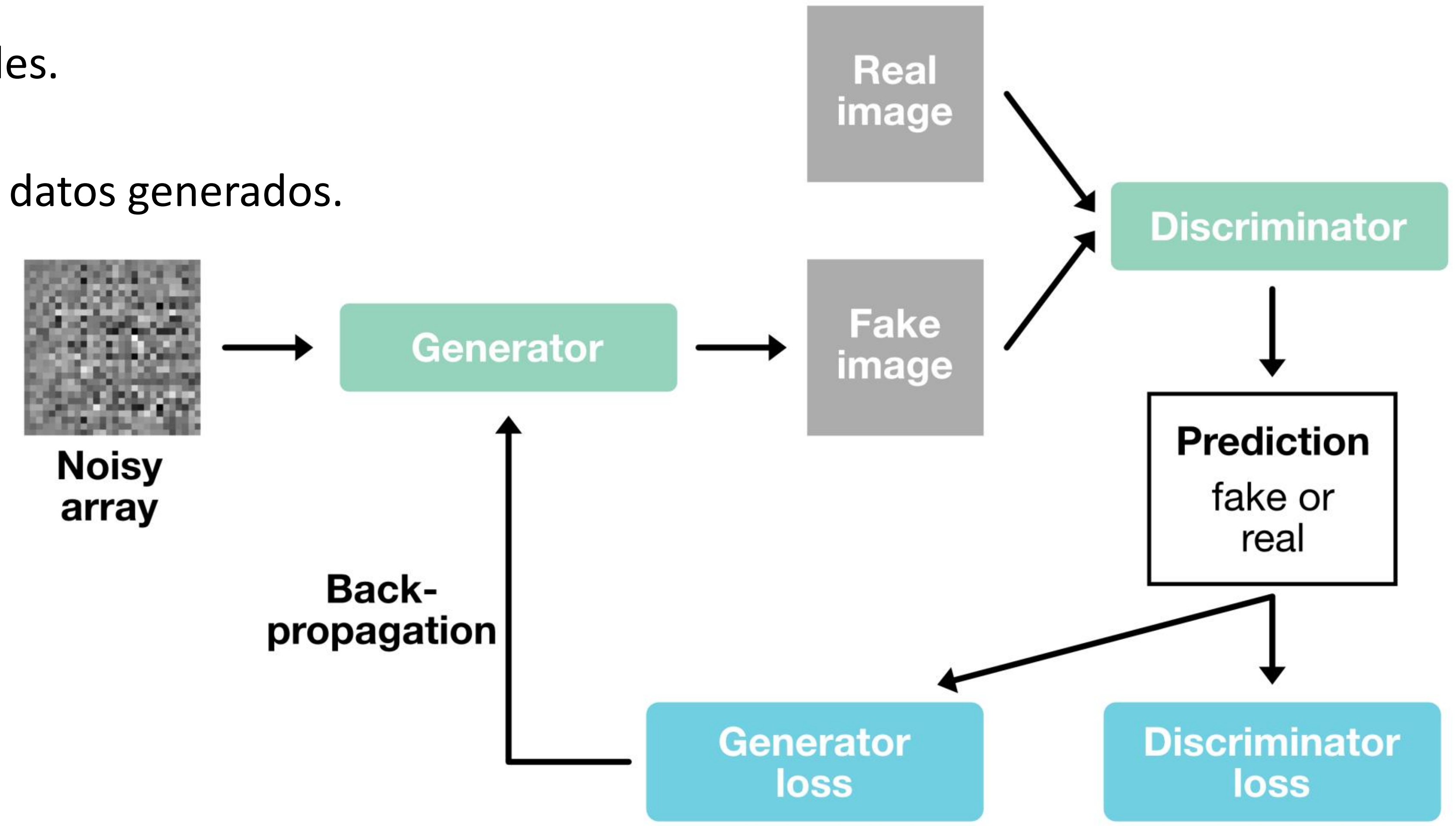
GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

donde:

- $p_{data}(x)$ es la distribución de los datos reales.
- $p_z(z)$ es la distribución del vector de ruido.
- $G(z)$ induce la distribución $p_g(x)$ sobre los datos generados.



Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_{G} \max_{D} V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D, G) = \int_x p_{data}(x) \log D(x) + p_g(x) \log\left(1 - D(x)\right) \partial x$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\left(D,G\right) = E_{x\sim p_{data}(x)}[\log D\left(x\right)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D,G) = \int_x p_{data}(x)\log D\left(x\right) + p_g(x)\log\big(1 - D(x)\big)\,\partial x$$

El objetivo es encontrar, para cada x, la función $D(x)$ que maximice $V(D,G)$. Es decir, para cada x debemos maximizar la función:

$$f\big(D(x)\big) = p_{data}(x)\log D\left(x\right) + p_g(x)\log\big(1 - D(x)\big)$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\,(D,G) = E_{x\sim p_{data}(x)}[\log D\,(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D,G) = \int_x p_{data}(x) \log D\,(x) + p_g(x)\log\big(1 - D(x)\big)\,\partial x$$

El objetivo es encontrar, para cada x, la función $D(x)$ que maximice $V(D,G)$. Es decir, para cada x debemos maximizar la función:

$$f\big(D(x)\big) = p_{data}(x)\log D\,(x) + p_g(x)\log\big(1 - D(x)\big)$$

Como $f\big(D(x)\big)$ es una función concava en $D(x)$ (por la concavidad del logaritmo), podemos encontrar el máximo derivando respecto a $D(x)$ y igualando a cero

$$\frac{\partial f}{\partial D(x)} = \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\,(D, G) = E_{x \sim p_{data}(x)}[\log D\,(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D, G) = \int_x p_{data}(x) \log D\,(x) + p_g(x) \log\big(1 - D(x)\big)\,\partial x$$

El objetivo es encontrar, para cada x, la función $D(x)$ que maximice $V(D, G)$. Es decir, para cada x debemos maximizar la función:

$$f\big(D(x)\big) = p_{data}(x) \log D\,(x) + p_g(x) \log\big(1 - D(x)\big)$$

Como $f\big(D(x)\big)$ es una función concava en $D(x)$ (por la concavidad del logaritmo), podemos encontrar el máximo derivando respecto a $D(x)$ y igualando a cero

$$\frac{\partial f}{\partial D(x)} = \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0$$

Resolviendo la ecuación: $p_{data}(x)\big(1 - D(x)\big) = p_g(x) D(x)$

Reorganizando los términos: $p_{data}(x) = D(x)\left(p_{data}(x) + p_g(x)\right)$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D, G) = \int_x p_{data}(x) \log D(x) + p_g(x) \log\left(1 - D(x)\right) \partial x$$

Por lo tanto, la solución óptima es:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

Dado un generador G fijo, definamos la distribución inducida por G en el espacio de datos como $p_g(x) = G(z)$ con $z \sim p_z(z)$. Entonces, la función objetivo se puede escribir de forma integral:

$$V(D,G) = \int_x p_{data}(x) \log D(x) + p_g(x) \log\big(1 - D(x)\big)\, \partial x$$

Por lo tanto, la solución óptima es:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

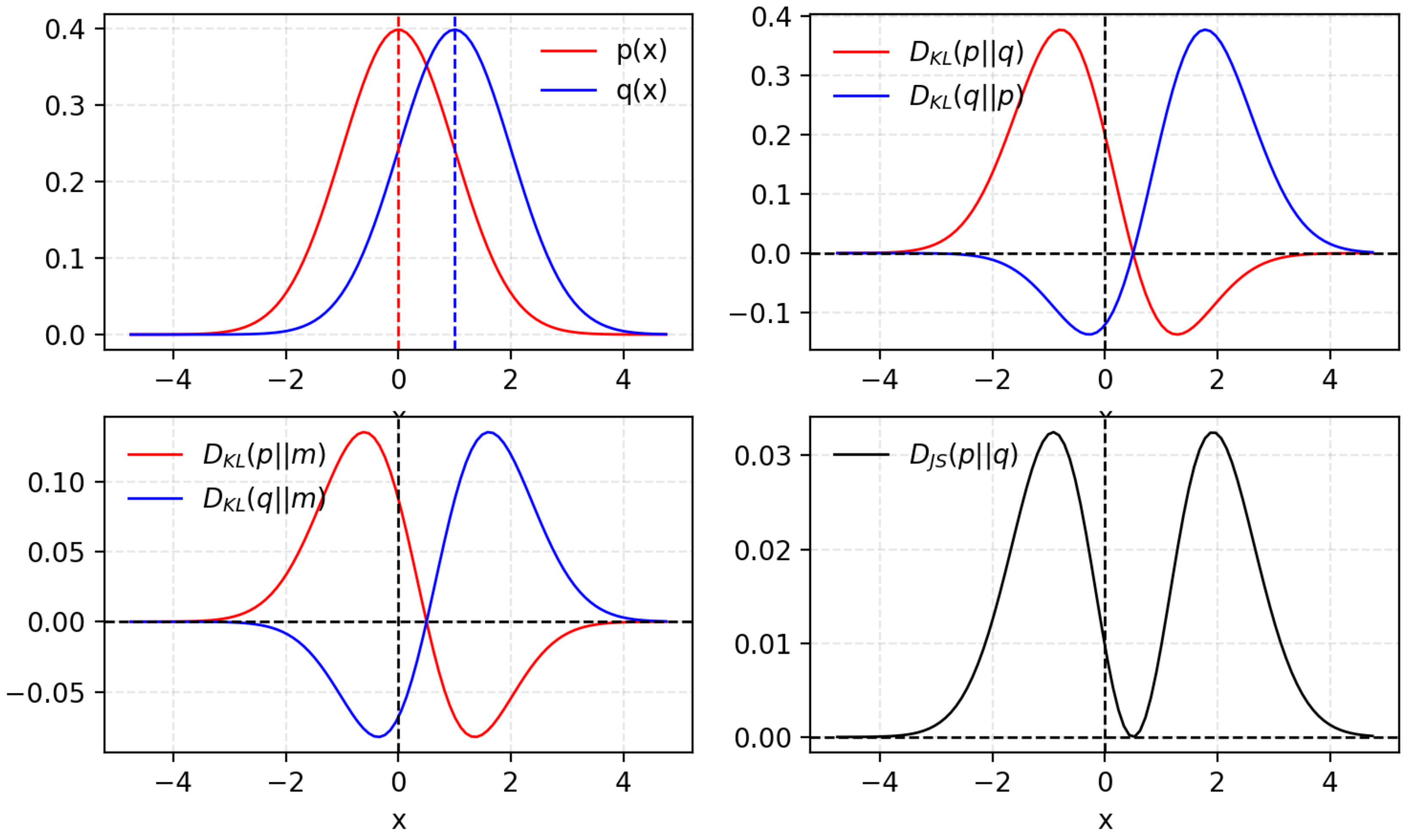
Reemplazando en $V(D,G)$:

$$V(D^*,G) = \int_x p_{data}(x) \log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) + p_g(x) \log\left(\frac{p_g(x)}{p_{data}(x) + p_g(x)}\right) \partial x$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

Jensen–Shannon *Divergence*

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



Jianhua Lin (1991) "Divergence Measures Based on the Shannon Entropy".
IEEE Transactions on Information Theory, 37(1), 145–151.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Entonces: $V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{1}{2}\cdot\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{1}{2}\cdot\frac{p_g(x)}{M(x)}\right)\partial x$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\,(D,G) = E_{x\sim p_{data}(x)}[\log D\,(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Entonces: $V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{1}{2}\cdot\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{1}{2}\cdot\frac{p_g(x)}{M(x)}\right)\partial x$

$$V(D^*,G) = \int_x p_{data}(x)\left[\log\left(\frac{p_{data}(x)}{M(x)}\right) - \log 2\right] + p_g(x)\left[\log\left(\frac{p_g(x)}{M(x)}\right) - \log 2\right]\partial x$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_{G}\max_{D} V\,(D,G) = E_{x\sim p_{data}(x)}[\log D\,(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x)+p_g(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x)+p_g(x)}\right)\partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Entonces: $V(D^*,G) = \int_x p_{data}(x)\log\left(\frac{1}{2}\cdot\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{1}{2}\cdot\frac{p_g(x)}{M(x)}\right)\partial x$

$$V(D^*,G) = \int_x p_{data}(x)\left[\log\left(\frac{p_{data}(x)}{M(x)}\right) - \log 2\right] + p_g(x)\left[\log\left(\frac{p_g(x)}{M(x)}\right) - \log 2\right]\partial x = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{M(x)}\right)\partial x - 2\log 2$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*, G) = \int_x p_{data}(x) \log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) + p_g(x) \log\left(\frac{p_g(x)}{p_{data}(x) + p_g(x)}\right) \partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Entonces: $V(D^*, G) = \int_x p_{data}(x) \log\left(\frac{1}{2} \cdot \frac{p_{data}(x)}{M(x)}\right) + p_g(x) \log\left(\frac{1}{2} \cdot \frac{p_g(x)}{M(x)}\right) \partial x$

$$V(D^*, G) = \int_x p_{data}(x) \left[\log\left(\frac{p_{data}(x)}{M(x)}\right) - \log 2\right] + p_g(x) \left[\log\left(\frac{p_g(x)}{M(x)}\right) - \log 2\right] \partial x = \int_x p_{data}(x) \log\left(\frac{p_{data}(x)}{M(x)}\right) + p_g(x) \log\left(\frac{p_g(x)}{M(x)}\right) \partial x - 2 \log 2$$

Finalmente tenemos: $V(D^*, G) = [\mathrm{D}_{KL}(p \parallel M) + \mathrm{D}_{KL}(q \parallel M)] - 2 \log 2$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\,(D, G) = E_{x\sim p_{data}(x)}[\log D\,(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo:

$$V(D^*, G) = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{p_{data}(x) + p_g(x)}\right)\partial x$$

Hacemos: $M(x) = \frac{1}{2}\left(p_{data}(x) + p_g(x)\right)$

Entonces: $V(D^*, G) = \int_x p_{data}(x)\log\left(\frac{1}{2}\cdot\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{1}{2}\cdot\frac{p_g(x)}{M(x)}\right)\partial x$

$$V(D^*, G) = \int_x p_{data}(x)\left[\log\left(\frac{p_{data}(x)}{M(x)}\right) - \log 2\right] + p_g(x)\left[\log\left(\frac{p_g(x)}{M(x)}\right) - \log 2\right]\partial x = \int_x p_{data}(x)\log\left(\frac{p_{data}(x)}{M(x)}\right) + p_g(x)\log\left(\frac{p_g(x)}{M(x)}\right)\partial x - 2\log 2$$

Finalmente tenemos: $V(D^*, G) = [\mathrm{D}_{KL}(p \parallel M) + \mathrm{D}_{KL}(q \parallel M)\,] - 2\log 2 = 2\cdot \mathrm{D}_{JS}\big(p_{data} \parallel p_g\big) - \log 4$

Ian J. Goodfellow (2014) "Generative Adversarial Networks". Advances in neural information processing systems, 27.

GAN

Se tiene un generador G y un discriminador D, definimos la función de valor:

$$\min_G \max_D V\,(D,G) = E_{x\sim p_{data}(x)}[\log D\,(x)] + E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

La función objetivo, evaluado con el discriminado optimo, queda expresado como:

$$V(D^*,G) = -\log 4 + 2\cdot \mathrm{D}_{JS}\left(p_{data} \parallel p_g\right)$$

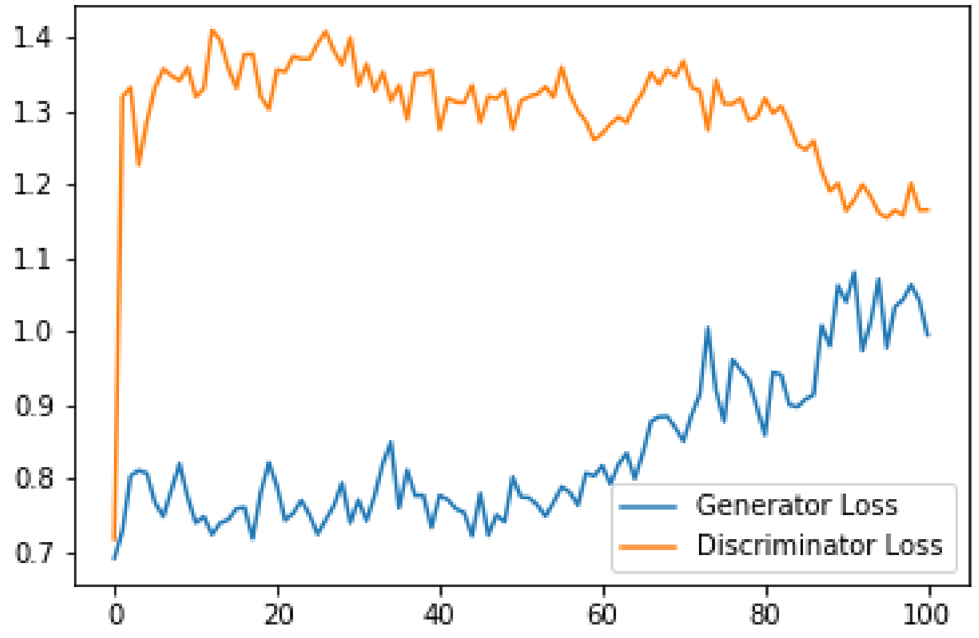
Por lo tanto, minimizar $V(D^*,G)$ con respecto a G equivale a minimizar Jensen-Shannon divergence entre p_{data} y p_g.

En el caso ideal en que $p_g = p_{data}$, se tiene:

$$\mathrm{D}_{JS}\left(p_{data} \parallel p_g\right) = 0 \quad \Longrightarrow \quad V(D^*,G) = -\log 4$$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

GAN



Discriminator loss: $\mathcal{L}_D = -E_{x\sim p_{data}(x)}[\log D\,(x)] - E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$

Generator loss: $\mathcal{L}_G = E_{z\sim p_z(z)}\left[\log\left(1 - D\big(G(z)\big)\right)\right]$

Sin embargo, en la práctica se utiliza a menudo una versión no saturante para: $\mathcal{L}_G = -E_{z\sim p_z(z)}\left[\log D\big(G(z)\big)\right]$

Ian J. Goodfellow (2014) "Generative Adversarial Networks".
Advances in neural information processing systems, 27.

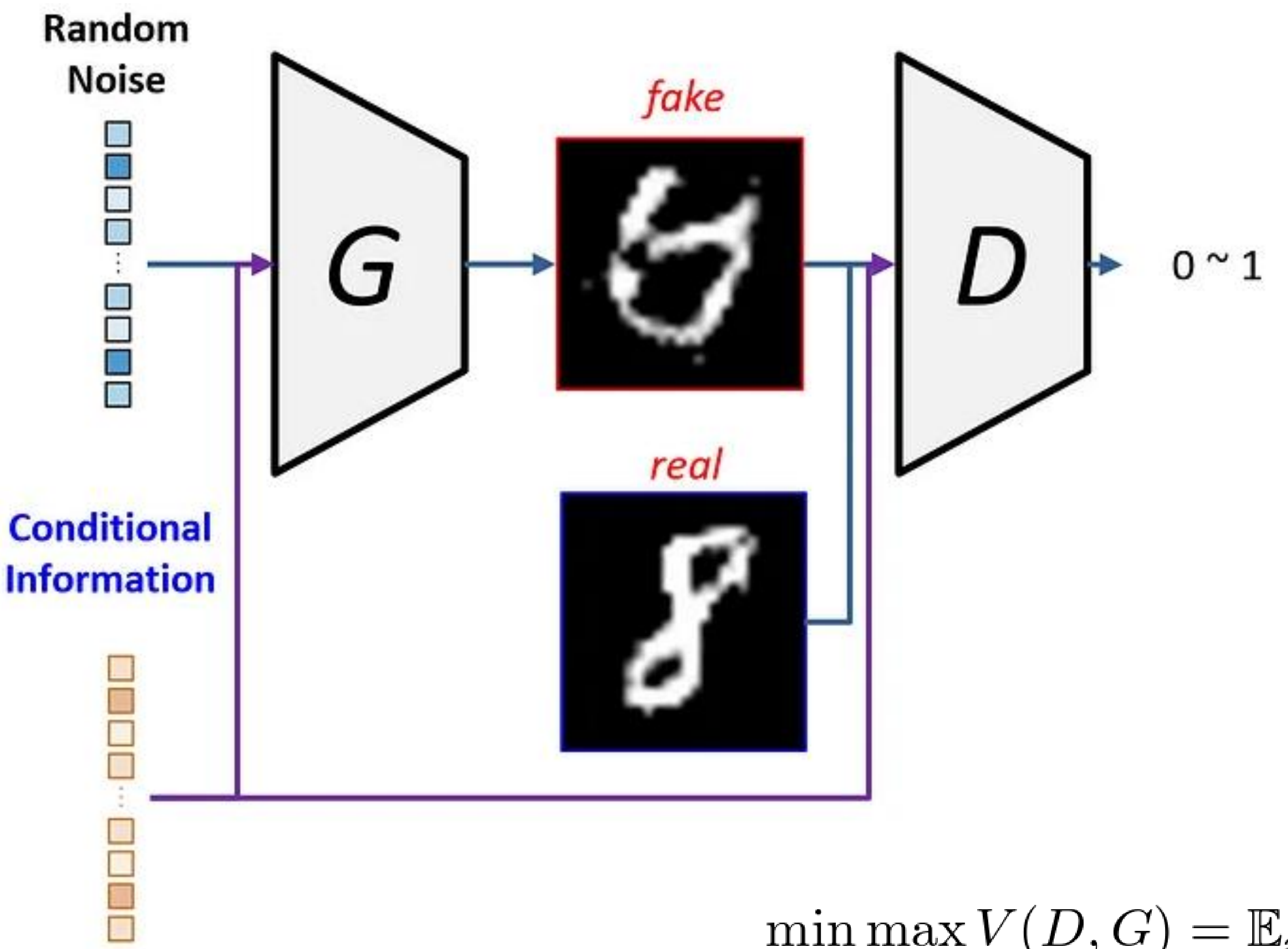
UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

2.

Conditional Generative *Adversarial Nets (cGAN)*

TRANSFORMATEC

cGAN



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

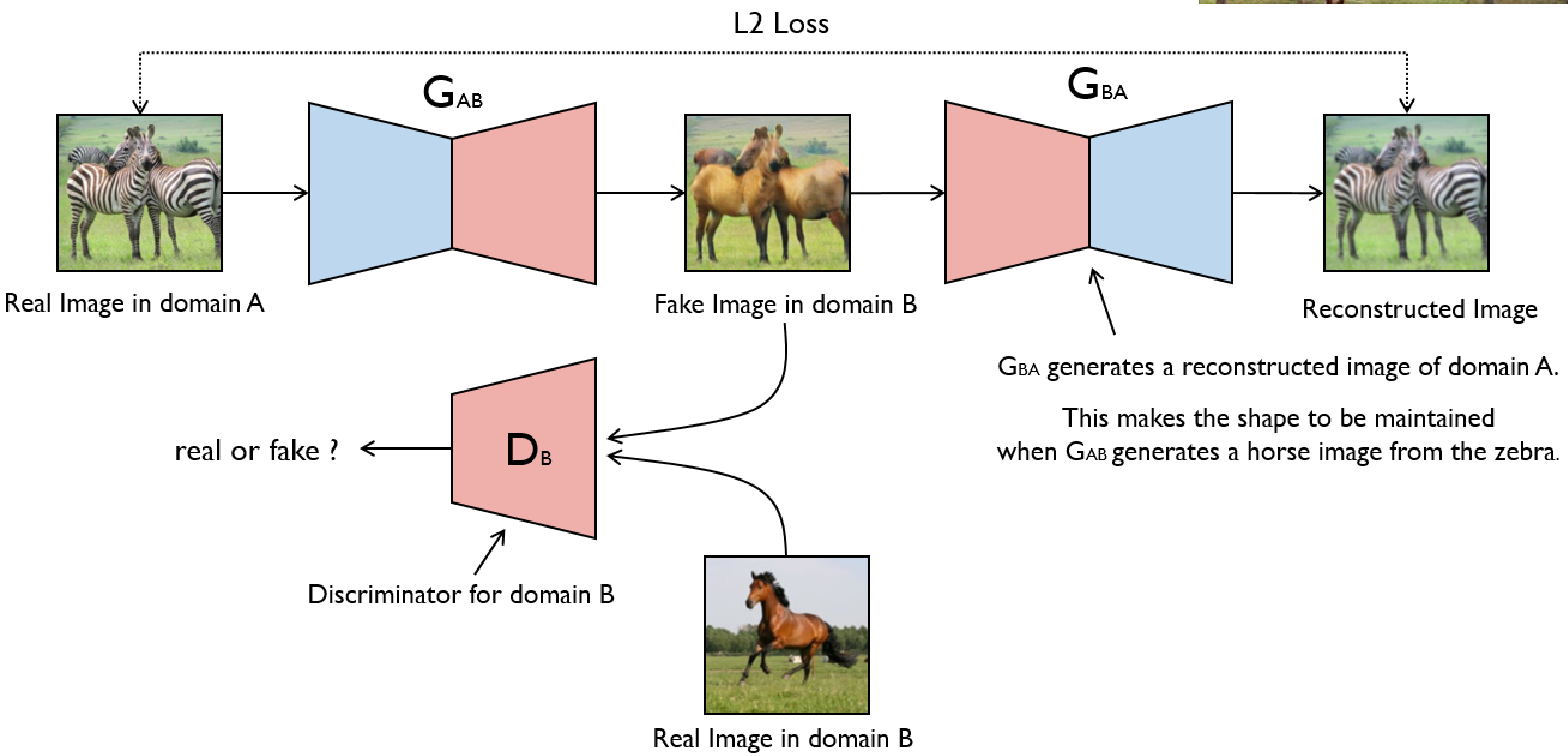
Mehdi Mirza and Simon Osindero (2014) "Conditional Generative Adversarial Nets". arXiv preprint arXiv:1411.1784.

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

3.

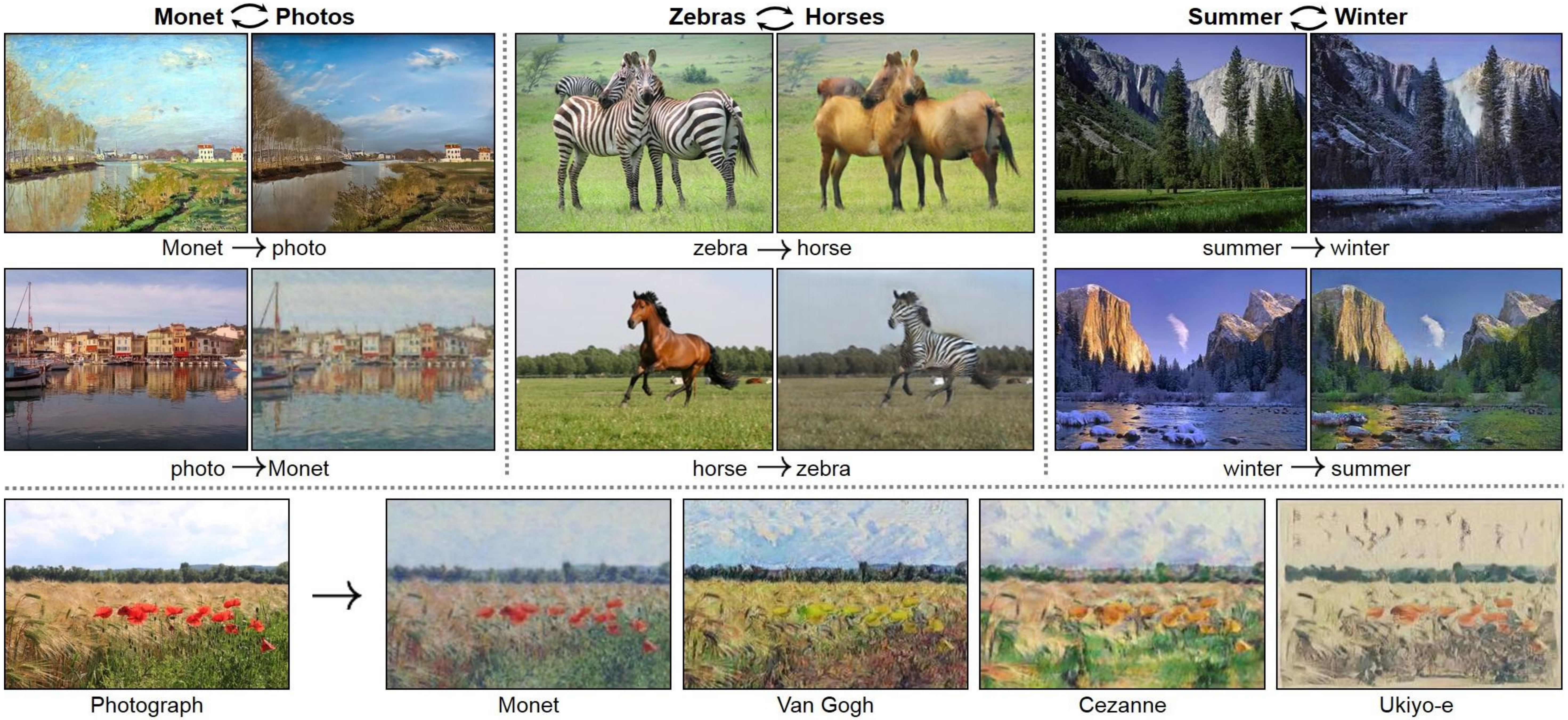
CycleGAN

Cycle*GAN*



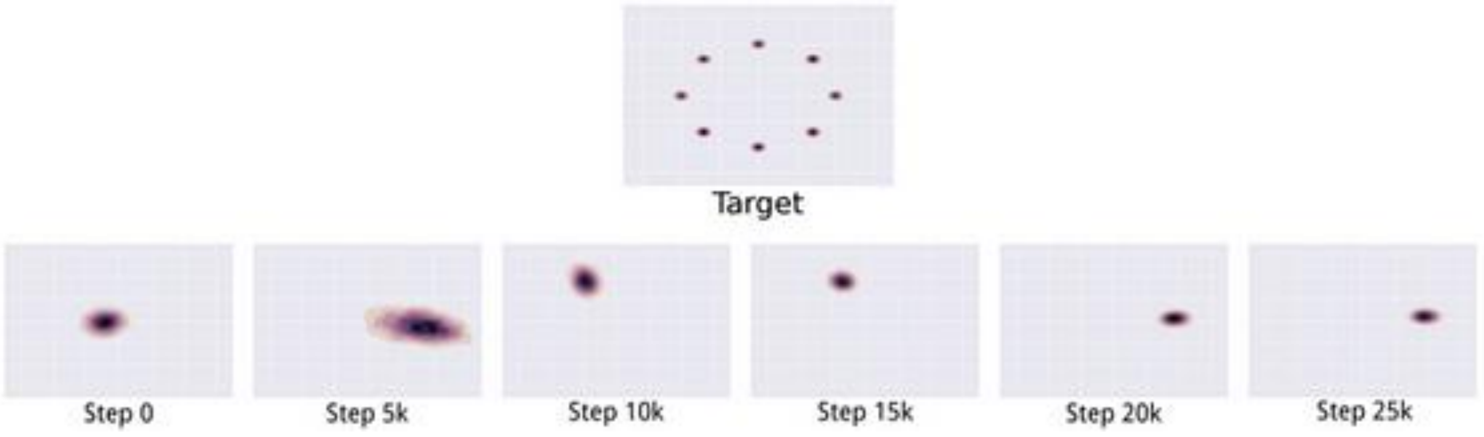
Jun-Yan Zhu et al. (2017) "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).

Cycle*GAN*



Jun-Yan Zhu et al. (2017) “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”.
Proceedings of the IEEE international conference on computer vision (pp. 2223-2232).

Mode *Collapse*

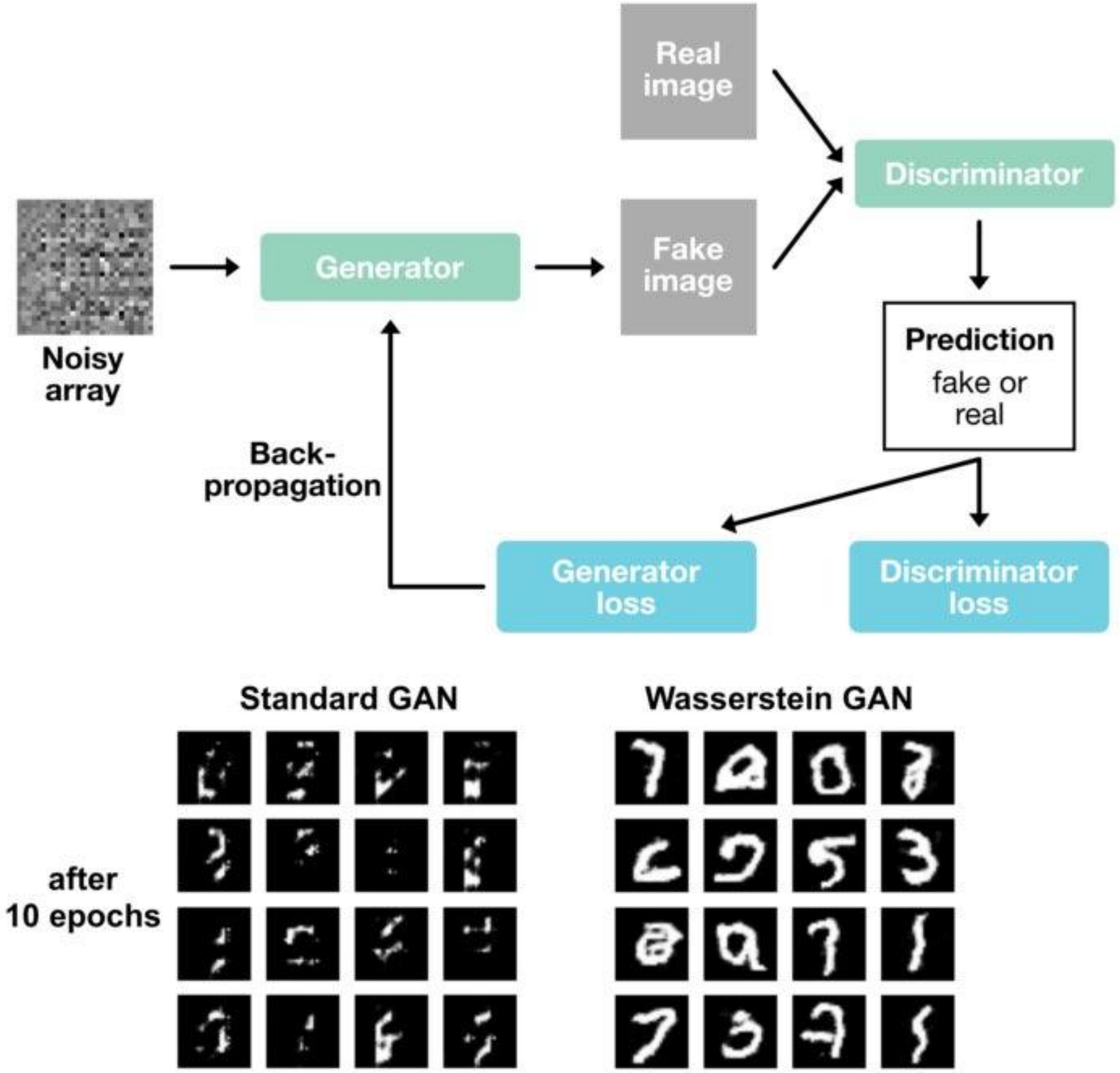


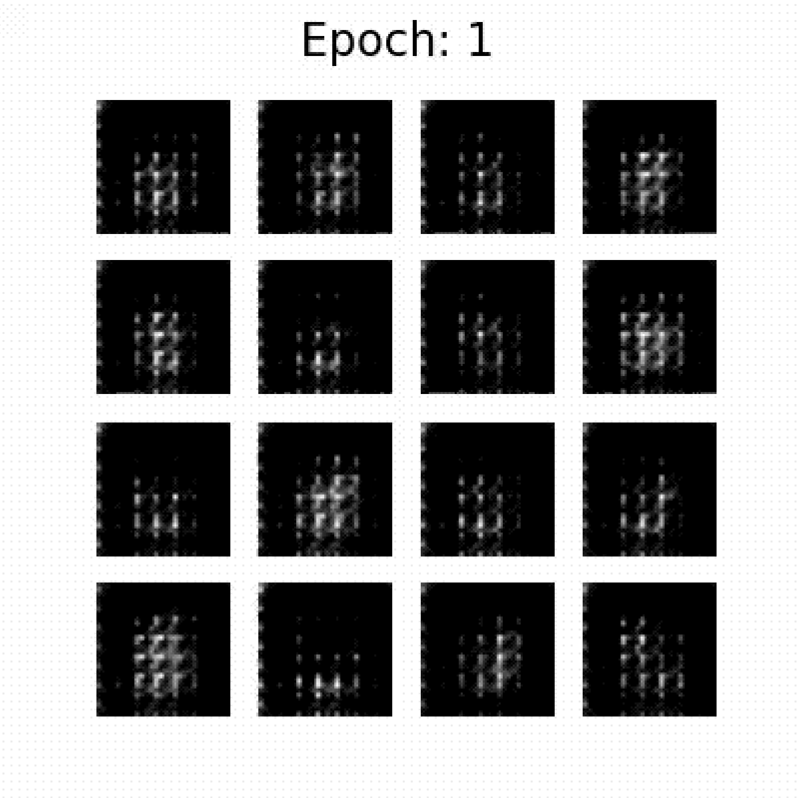
UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

4.

Wasserstein *GAN*

Wasserstein *GAN*





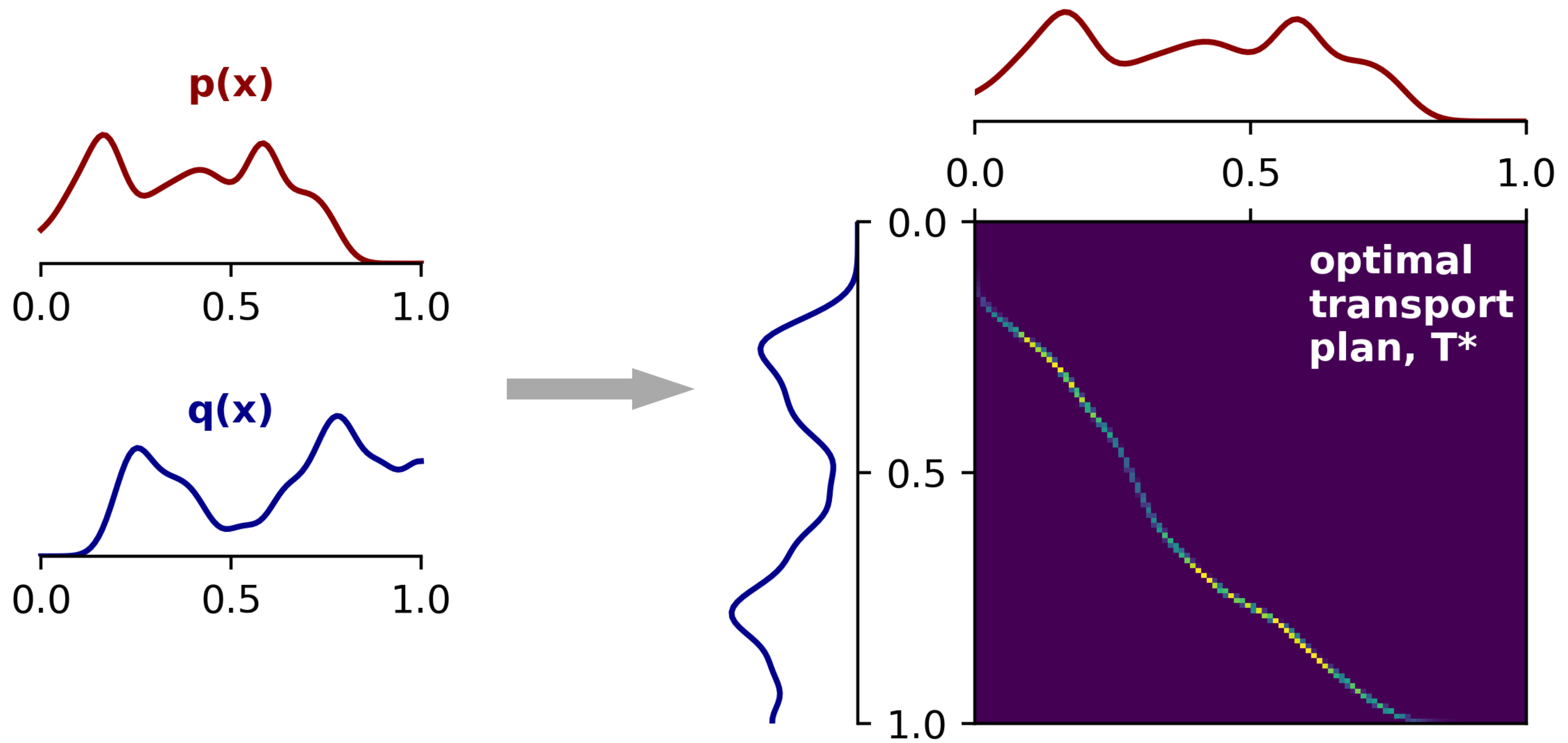
Martin Arjovsky et al. (2017) "Wasserstein GAN". arXiv:1701.07875

Wasserstein *distance*

Wasserstein distance (Earth Mover's Distance, EMD), es una medida que cuantifica la diferencia entre dos distribuciones de probabilidad.

Noción intuitiva

Imagina que tienes dos montones de tierra (representando distribuciones de probabilidad) y el objetivo es mover la tierra de un montón para que coincida exactamente con el otro. La distancia de Wasserstein mide el trabajo total necesario para hacer esta transformación.



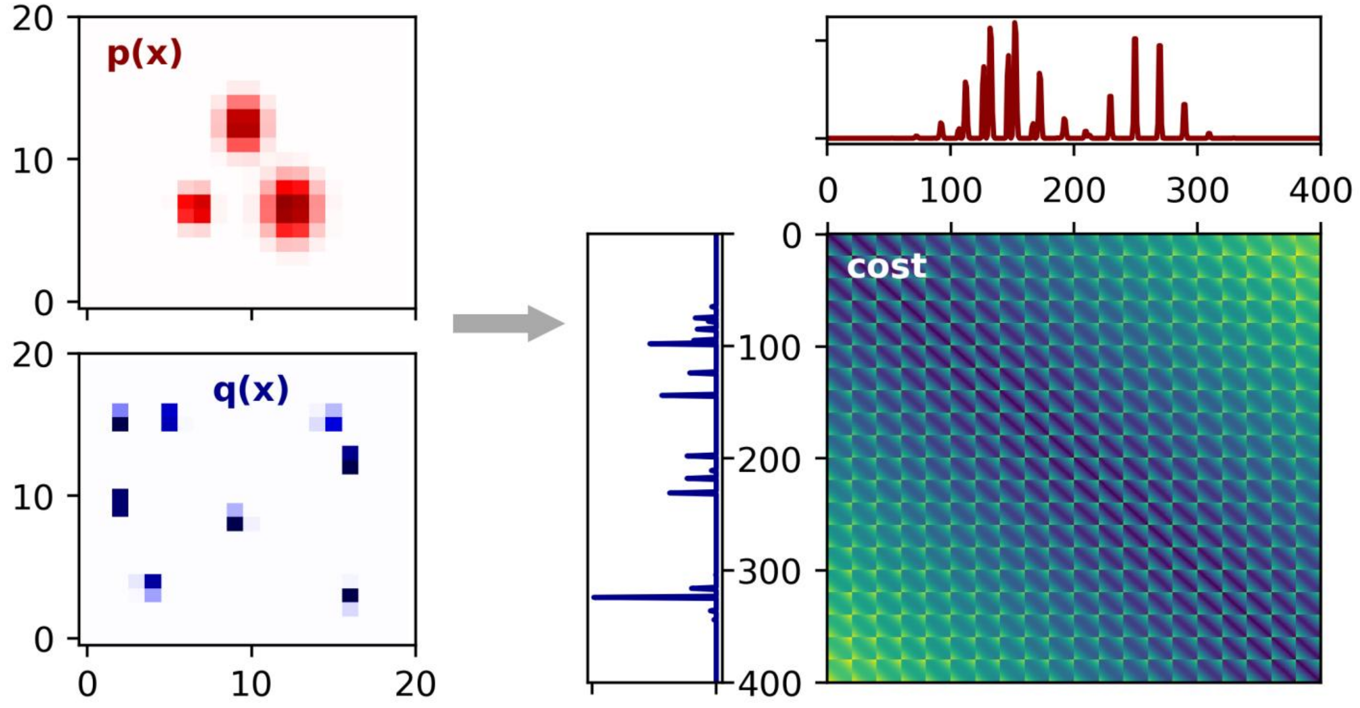
Gaspard Monge (1781) "Mémoire sur la théorie des déblais et des remblais". Mem. Math. Phys. Acad. Royale Sci., 666-704.

Wasserstein *distance*

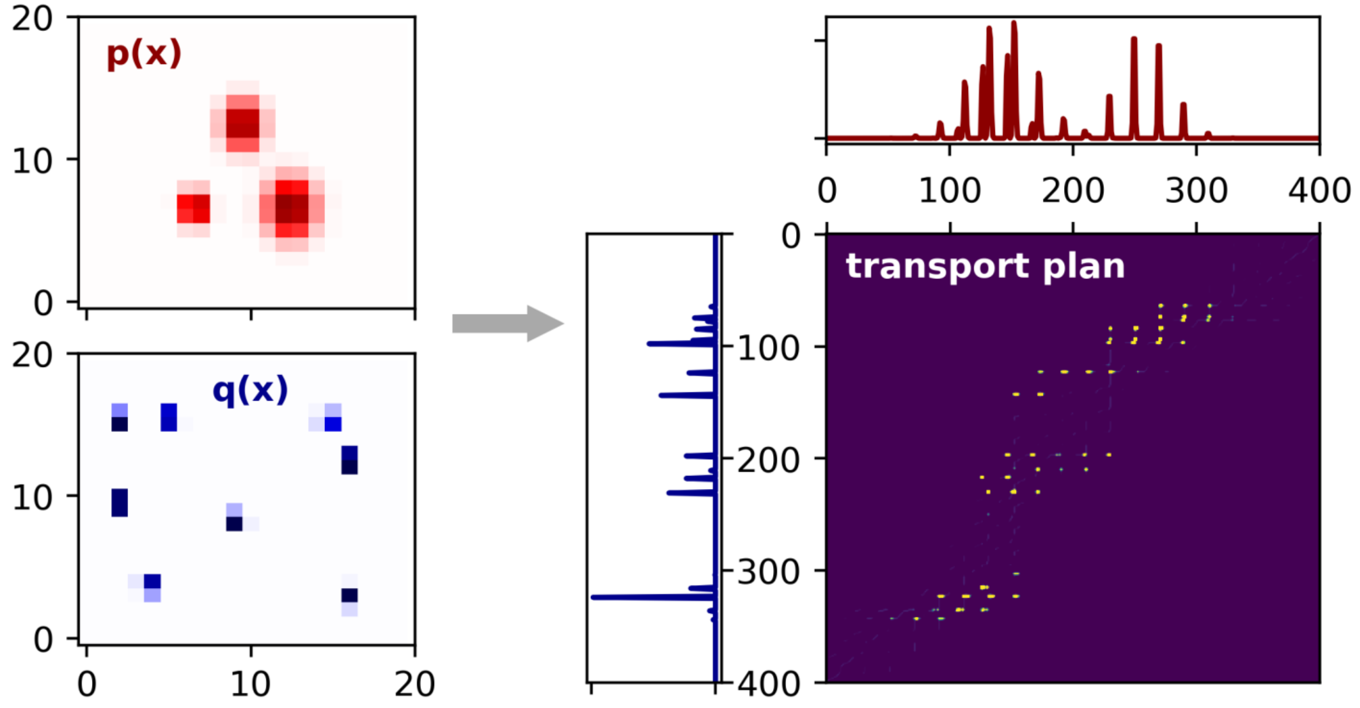
Kantorovich–Rubinstein metric

Wasserstein distance (Earth Mover's Distance, EMD), es una medida que cuantifica la diferencia entre dos distribuciones de probabilidad.

$$W_p(\mu,\nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \mathbf{E}_{(x,y)\sim\gamma} d(x,y)^p \right)^{1/p}$$



Transport cost matrix in 2D



Optimal transport plan in 2D

Leonid V. Kantorovich (1942) "On the translocation of masses".
Dokl. Akad. Nauk. USSR (NS) (Vol. 37, pp. 199-201).

Wasserstein *distance*

Kantorovich–Rubinstein metric

Sea P_r la distribución real y P_g la distribución generada. El teorema de Kantorovich-Rubinstein permite reescribir la distancia de Wasserstein de la siguiente forma:

$$W\left(P_r, P_g\right) = \sup_{\|f\|_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)]$$

donde la supremum se toma sobre todas las funciones f que son 1-Lipschitz (es decir, $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$ para todo x_1, x_2).

Leonid V. Kantorovich (1942) "On the translocation of masses". Dokl. Akad. Nauk. USSR (NS) (Vol. 37, pp. 199-201).

Wasserstein *distance*

Kantorovich–Rubinstein metric

Sea P_r la distribución real y P_g la distribución generada. El teorema de Kantorovich-Rubinstein permite reescribir la distancia de Wasserstein de la siguiente forma:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} E_{x\sim P_r}[f(x)] - E_{x\sim P_g}[f(x)]$$

donde la supremum se toma sobre todas las funciones f que son 1-Lipschitz (es decir, $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$ para todo x_1, x_2).

Una función **1-Lipschitz** tiene una pendiente acotada, lo que garantiza que los gradientes que se computan durante el entrenamiento no sean excesivamente grandes ni se vuelvan inestables. Esto es especialmente importante cuando se actualizan los parámetros del generador, ya que gradientes más suaves conducen a una mejora más estable en el proceso de optimización.

Leonid V. Kantorovich (1942) "On the translocation of masses".
Dokl. Akad. Nauk. USSR (NS) (Vol. 37, pp. 199-201).

Wasserstein *distance*

Kantorovich–Rubinstein metric

Sea P_r la distribución real y P_g la distribución generada. El teorema de Kantorovich-Rubinstein permite reescribir la distancia de Wasserstein de la siguiente forma:

$$W(P_r, P_g) = \sup_{\|f\|_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_g}[f(x)]$$

donde la supremum se toma sobre todas las funciones f que son 1-Lipschitz (es decir, $|f(x_1) - f(x_2)| \leq |x_1 - x_2|$ para todo x_1, x_2).

Una función **1-Lipschitz** tiene una pendiente acotada, lo que garantiza que los gradientes que se computan durante el entrenamiento no sean excesivamente grandes ni se vuelvan inestables. Esto es especialmente importante cuando se actualizan los parámetros del generador, ya que gradientes más suaves conducen a una mejora más estable en el proceso de optimización.

Un aspecto crucial es que la función f_θ (**critic**) debe ser 1-Lipschitz. En la implementación original de WGAN se utiliza el **weight clipping** (recorte de pesos) para forzar esta propiedad, es decir, se limita cada peso del **critic** a un intervalo fijo $[-c, c]$.

Desventajas:

- **Limita la capacidad del modelo:** El recorte de pesos puede afectar la capacidad de la red para aprender funciones complejas.
- **Gradientes Inestables:** El clipping puede inducir gradientes erráticos y comportamientos subóptimos durante el entrenamiento.

Martin Arjovsky et al. (2017) "Wasserstein GAN". arXiv:1701.07875

WGAN

GAN

Discriminator loss: $\mathcal{L}_D = -E_{x \sim p_{data}(x)}[\log D(x)] - E_{z \sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$

Generator loss: $\mathcal{L}_G = -E_{z \sim p_z(z)}\left[\log D\left(G(z)\right)\right]$

WGAN

Critic loss: $\mathcal{L}_C = -E_{x \sim p_r(x)}[f_\theta(x)] + E_{z \sim p_z(z)}\left[f_\theta\left(G(z)\right)\right]$

Generator loss: $\mathcal{L}_G = -E_{z \sim p_z(z)}\left[f_\theta\left(G(z)\right)\right]$

(En todos los casos, el objetivo es minimizar los losses)

Martin Arjovsky et al. (2017) "Wasserstein GAN". arXiv:1701.07875

WGAN-GP
(Wasserstein GAN + Gradient Penalty)

Impone de manera más estable y efectiva la condición de 1-Lipschitz en el critic, sin recurrir al método de weight clipping, el cual puede limitar la capacidad del modelo.

En lugar de limitar directamente los pesos, WGAN-GP introduce un término de penalización que castiga las violaciones de la condición de 1-Lipschitz de forma suave y continua. Para cualquier función f 1-Lipschitz se debe cumplir:

$$\|\nabla_{\hat{x}} f(\hat{x})\|_2 \leq 1 \quad \forall \hat{x}$$

Critic loss:

$$L_C = \mathbb{E}_{x\sim P_r}[f(x)] - \mathbb{E}_{z\sim p(z)}\left[f\left(G(z)\right)\right] + \lambda\, \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(|\nabla_{\hat{x}} f(\hat{x})|_2 - 1)^2]$$

donde:

- P_r es la distribución de datos reales.
- P_g es la distribución generada (obtenida al muestrear $z \sim p(z)$ y calcular $G(z)$).
- $P_{\hat{x}}$ es la distribución de puntos $\hat{x}$ obtenidos interpolando entre muestras reales y generadas. Es decir, para $\epsilon \sim \text{Uniform}(0,1)$:

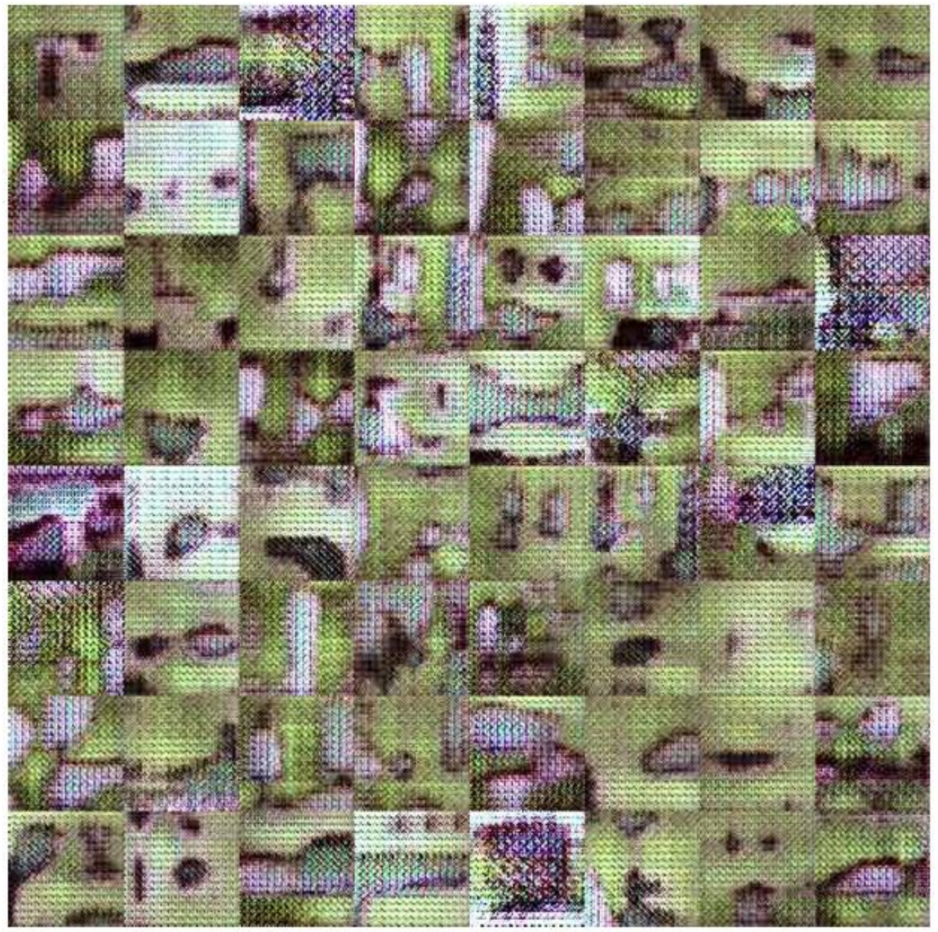
  $$\hat{x} = \epsilon\, x_{real} + (1-\epsilon)\, x_{gen}$$

  lo que asegura que se evalúa la penalización en regiones de la interpolación entre ambas distribuciones.
- λ es un hiperparámetro que controla la fuerza de la penalización del gradiente.

Ishaan Gulrajani et al. (2017) "Improved Training of Wasserstein GANs".
Advances in neural information processing systems, 30.

WGAN-GP

(Wasserstein GAN + Gradient Penalty)



WGAN (weight clipping)

Ambos con ResNet-101

WGAN-GP

Ishaan Gulrajani et al. (2017) "Improved Training of Wasserstein GANs".
Advances in neural information processing systems, 30.

WGAN-GP

(Wasserstein GAN + Gradient Penalty)

```
def computeGradientPenalty(critic, realSamples, fakeSamples, λgp=10):
    n_batch = realSamples.size(0)
    device  = realSamples.device

    # Generamos un tensor epsilon con valores aleatorios entre 0 y 1
    ε = torch.rand(n_batch, 1, 1, 1, device=device) # Le damos la forma de realSamples

    # Interpolamos entre muestras reales y fakes
    interSamples = ε * realSamples + (1 - ε) * fakeSamples
    interSamples.requires_grad_(True)

    # Evaluamos el critic en las muestras interpoladas
    interCritic = critic(interSamples)

    # Calculamos el gradiente de la salida respecto a las muestras interpoladas
    gradients = autograd.grad(
        outputs=interCritic,
        inputs=interSamples,
        grad_outputs=torch.ones(interCritic.size(), device=device),
        create_graph=True,
        retain_graph=True,
        only_inputs=True
    )[0]

    # Reestructuramos el gradiente para calcular la norma L2 por muestra.
    gradients = gradients.view(n_batch, -1)
    gradientNorm = gradients.norm(2, dim=1)

    # Calculamos el Gradient Penalty
    return λgp * ((gradientNorm - 1) ** 2).mean()
```

→ Valor inicial para la propagación del gradiente
Estamos indicando que la derivada de la salida respecto a sí misma es 1

Ishaan Gulrajani et al. (2017) "Improved Training of Wasserstein GANs".
Advances in neural information processing systems, 30.

WGAN-GP

(Wasserstein GAN + Gradient Penalty)

```
import torch
import torch.optim as optim
import torch.autograd as autograd

n_epochs = 100
n_critic = 5
lr   = 1e-4
λgp = 10
n_latent = 100  # Depende de tu modelo

optimizerG = optim.Adam(generator.parameters(),
                        lr=lr, betas=(0.5, 0.9))
optimizerC = optim.Adam(critic.parameters(),
                        lr=lr, betas=(0.5, 0.9))
# El paper dice β1=0, lo podrías probar
```

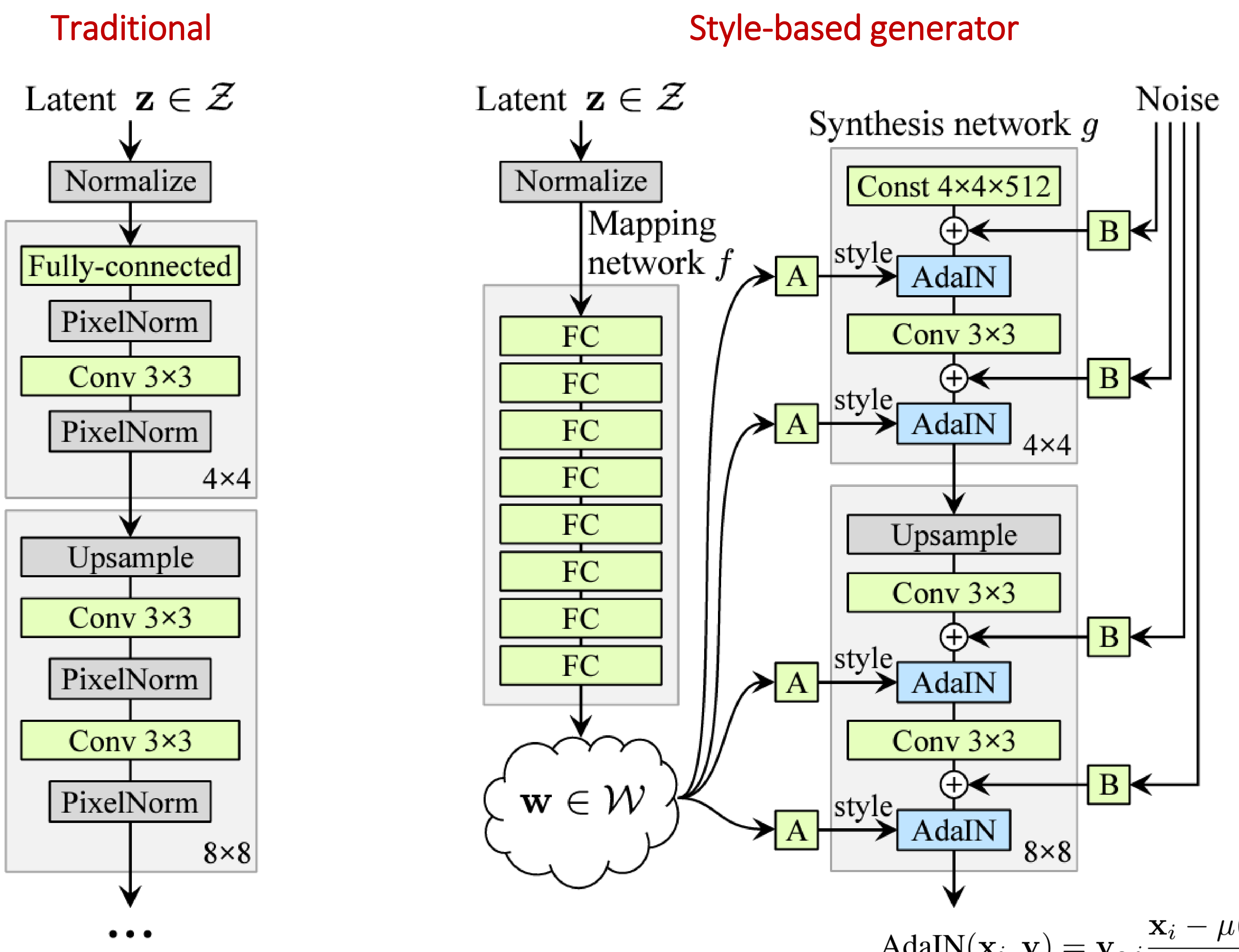
```
for epoch in range(n_epochs):
    for i, (realSamples, _) in enumerate(dataloader):
        n_batch     = realSamples.size(0)
        realSamples = realSamples.to(device)

        # Critic
        for _ in range(n_critic):
            z = torch.randn(n_batch, n_latent, device=device)
            fakeSamples = generator(z).detach()

            criticReal = critic(realSamples)
            criticFake = critic(fakeSamples)

            gp = computeGradientPenalty(critic, realSamples, fakeSamples, λgp=λgp)
            lossCritic = -torch.mean(criticReal) + torch.mean(criticFake) + gp

            optimizerC.zero_grad()
            lossCritic.backward()
            optimizerC.step()

        # Generator
        z = torch.randn(n_batch, n_latent, device=device)
        fakeSamples = generator(z)
        lossGenerator = -torch.mean(critic(fakeSamples))

        optimizerG.zero_grad()
        lossGenerator.backward()
        optimizerG.step()
```


Ishaan Gulrajani et al. (2017) "Improved Training of Wasserstein GANs".
Advances in neural information processing systems, 30.

3.

StyleGAN

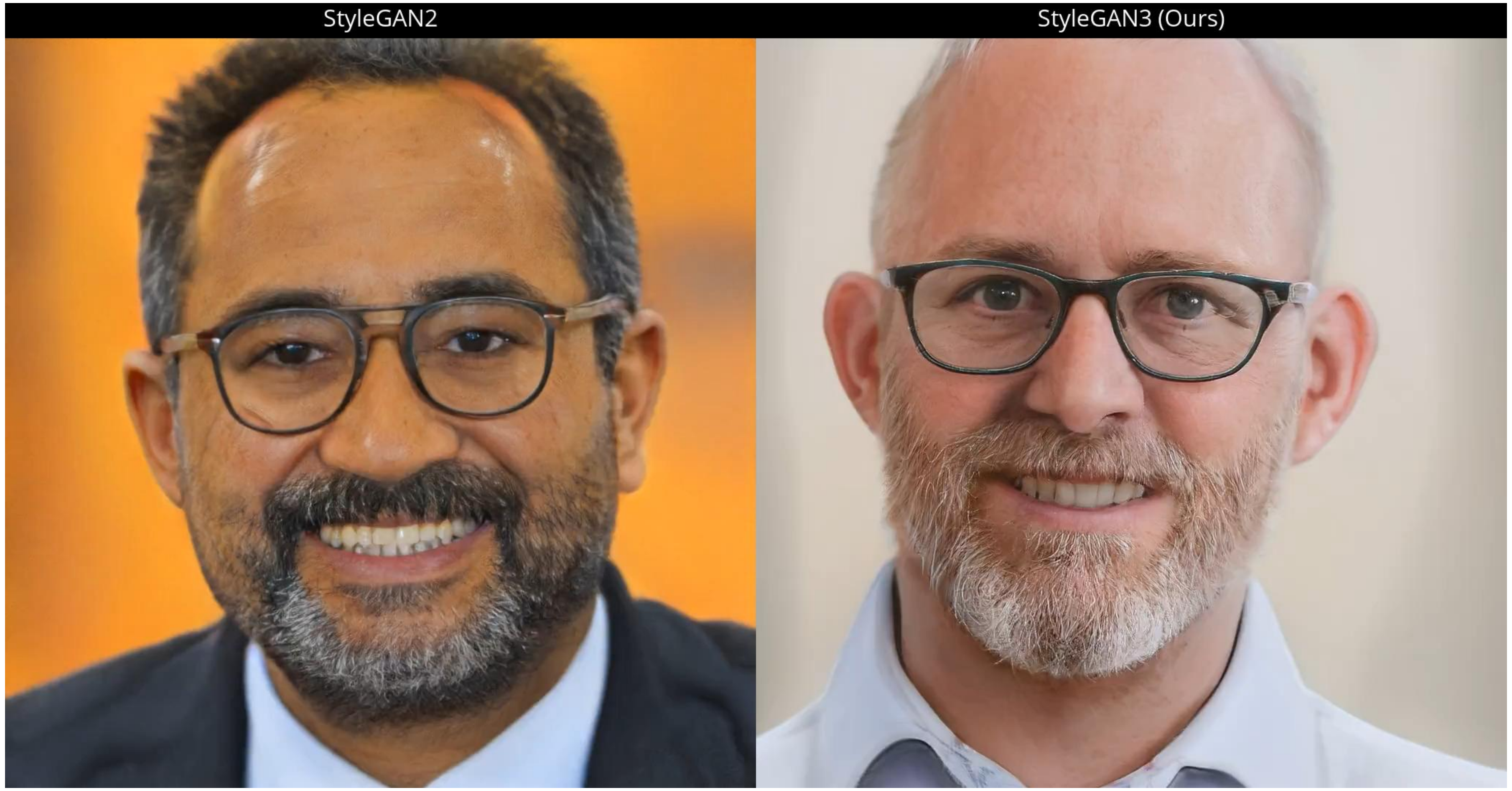
StyleGAN



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

Tero Karras et al. (2019) "A Style-Based Generator Architecture for Generative Adversarial Networks". Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 4401-4410).

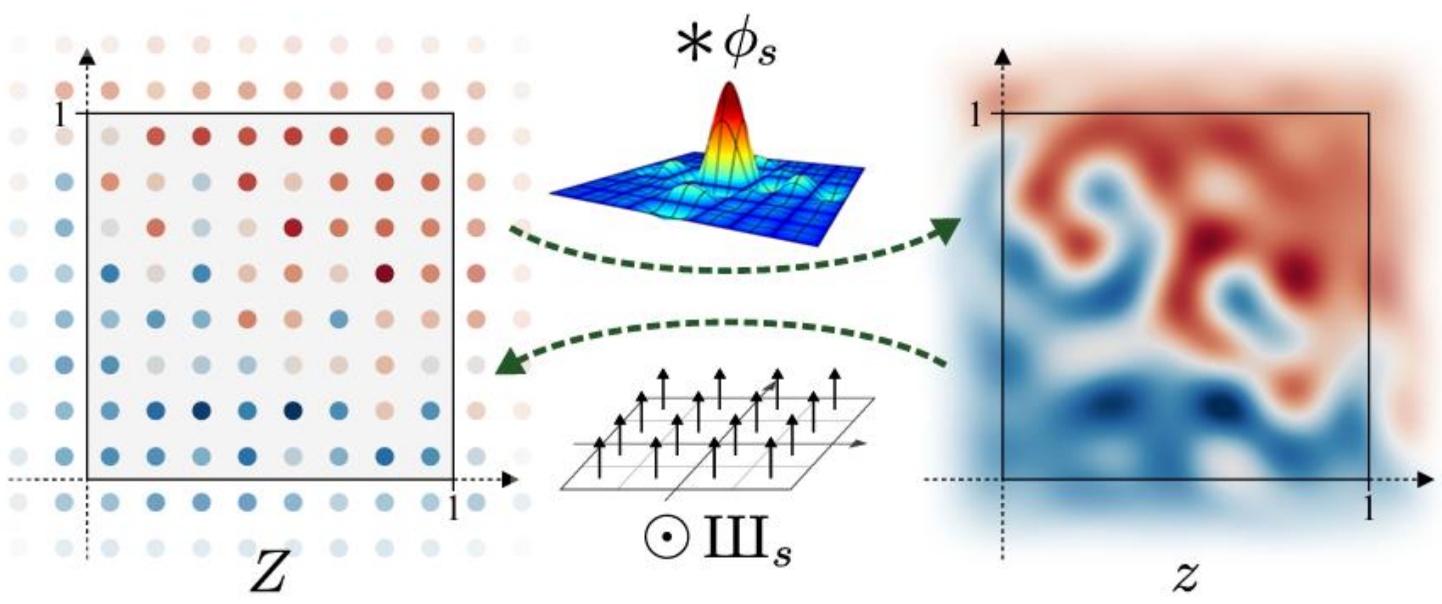
StyleGAN v3

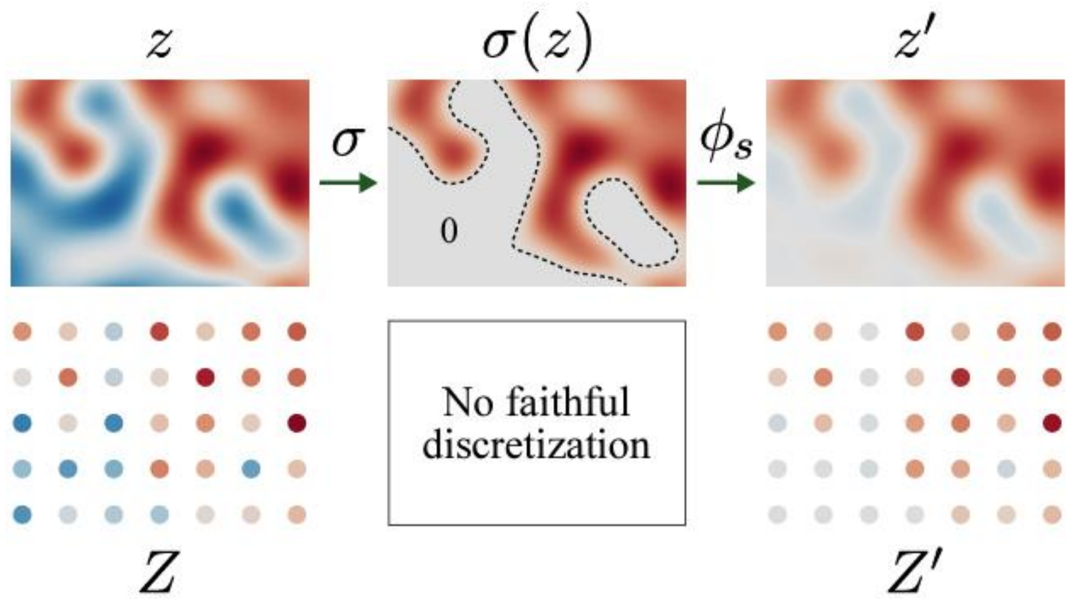


Texture sticking

Tero Karras et al. (2020) "Alias-Free Generative Adversarial Networks".
Advances in neural information processing systems, 34, 852-863.

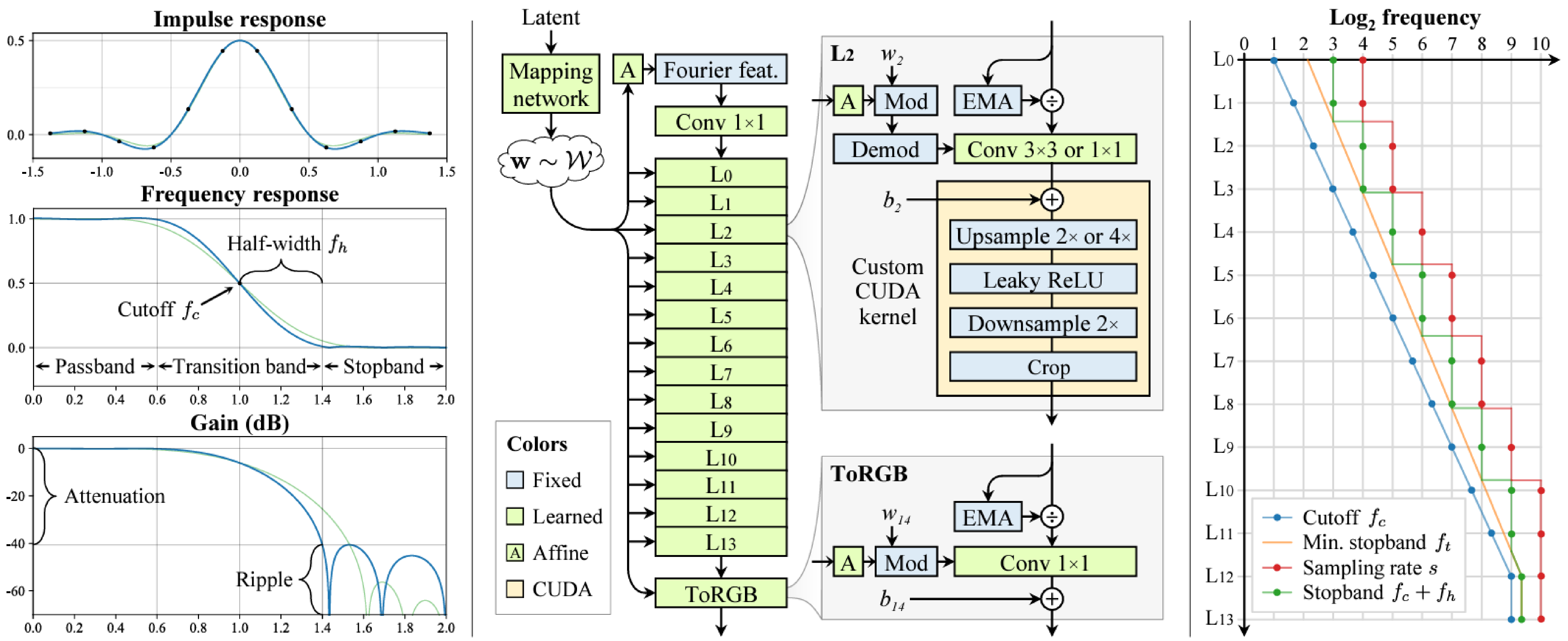
Style*GAN v3*





Tero Karras et al. (2020) "Alias-Free Generative Adversarial Networks".
Advances in neural information processing systems, 34, 852-863.

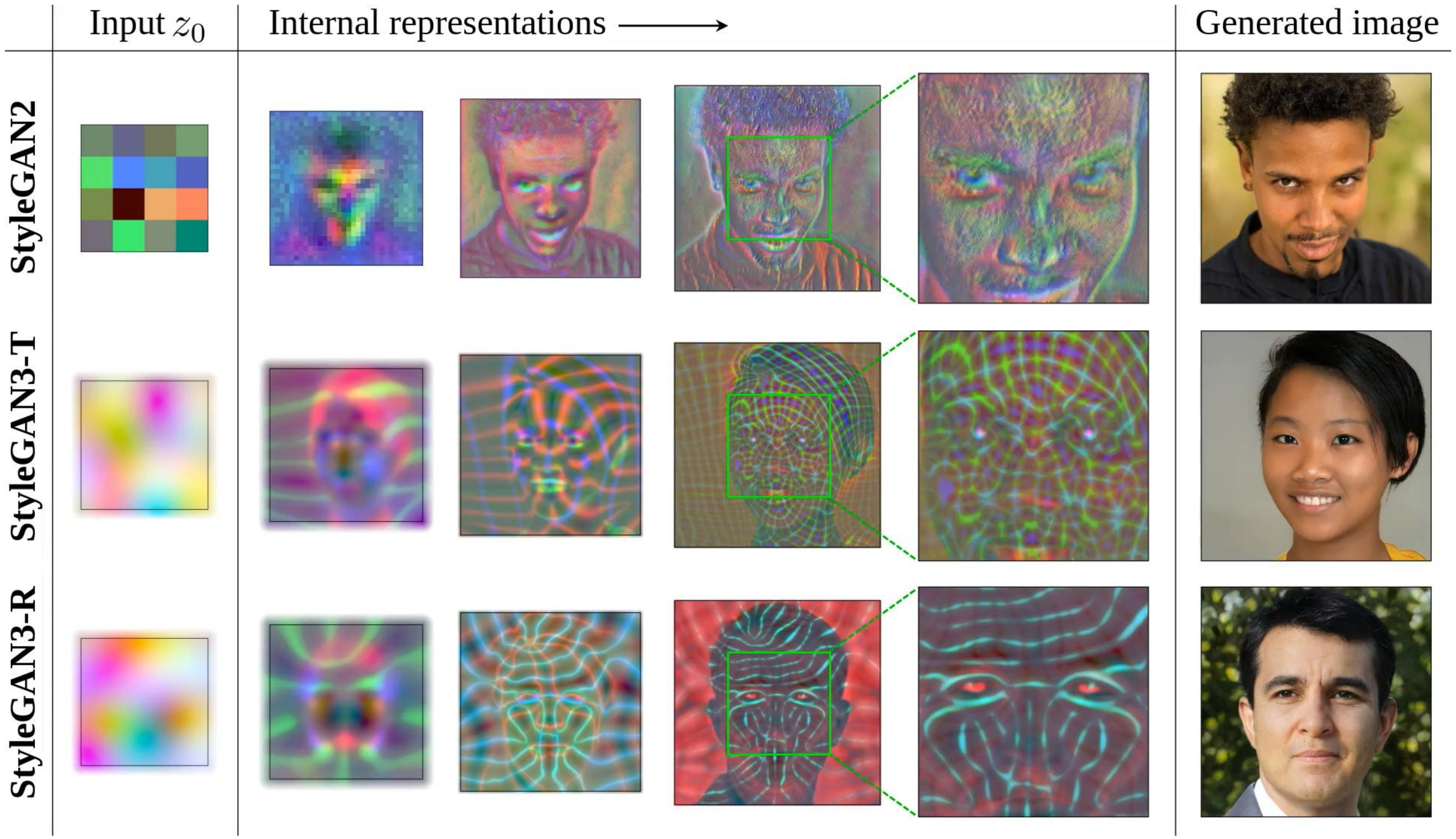
StyleGAN v3



(a) Filter design concepts (b) Our alias-free StyleGAN3 generator architecture (c) Flexible layers

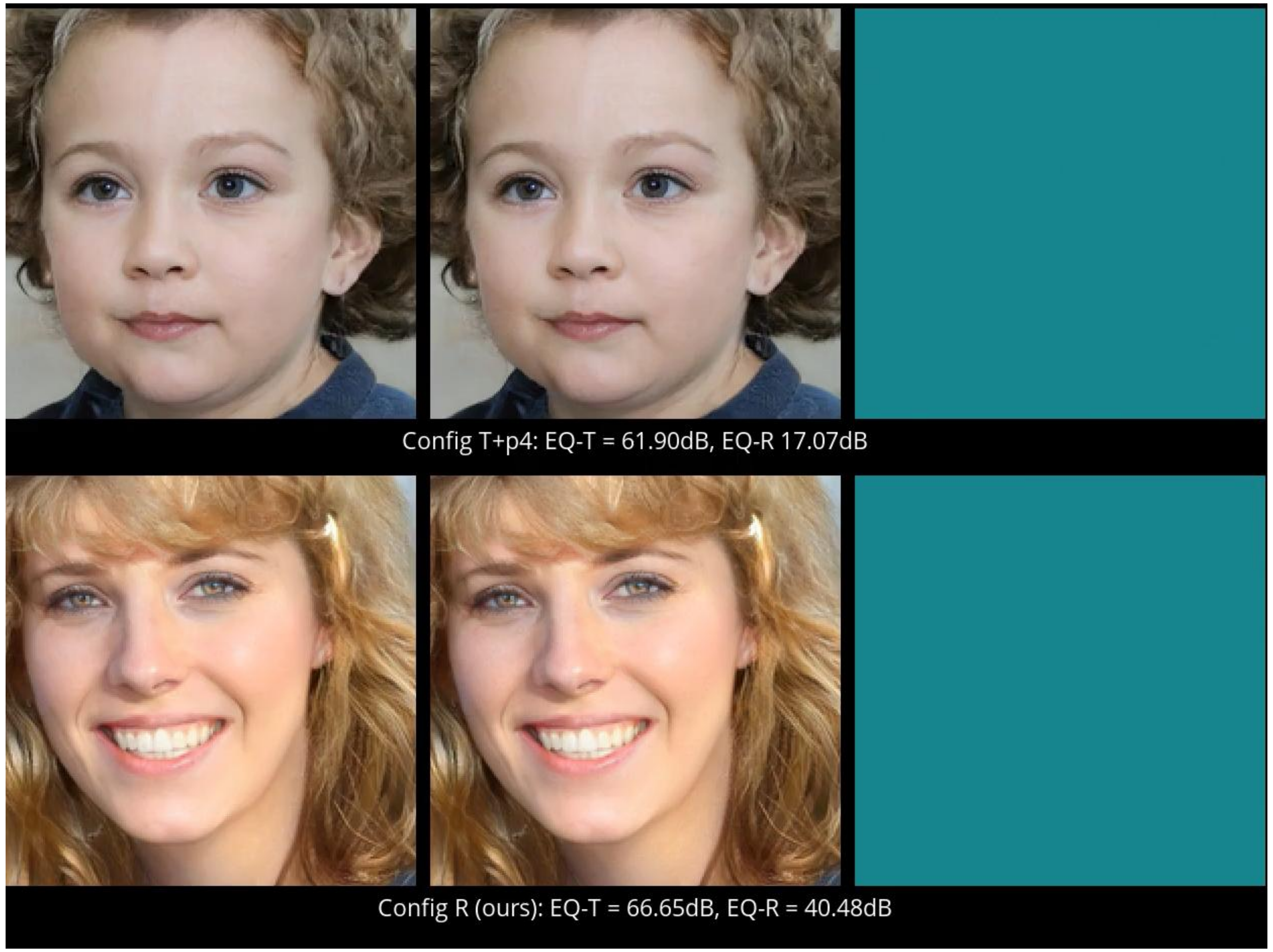
Tero Karras et al. (2020) "Alias-Free Generative Adversarial Networks".
Advances in neural information processing systems, 34, 852-863.

StyleGAN v3



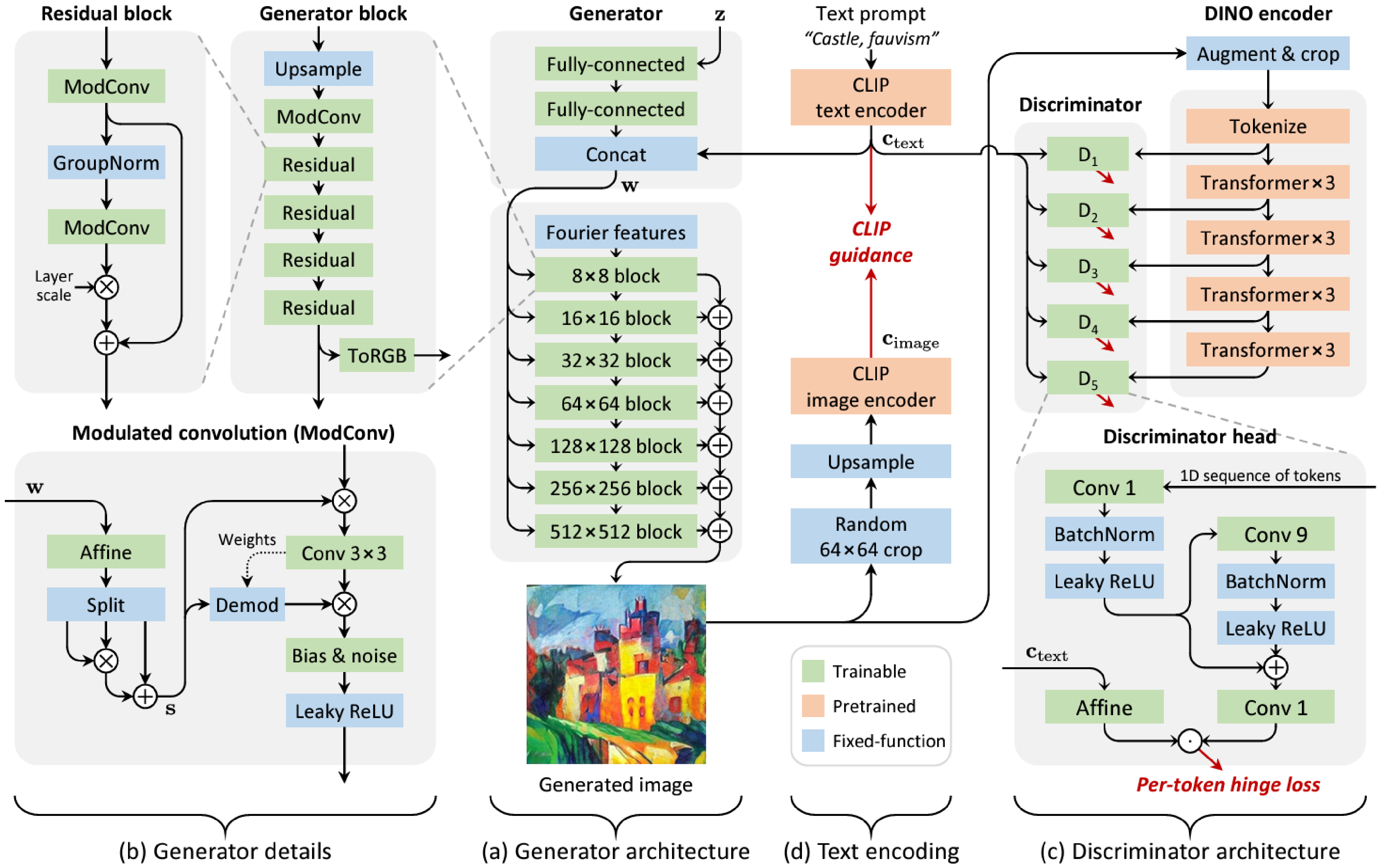
Tero Karras et al. (2020) "Alias-Free Generative Adversarial Networks".
Advances in neural information processing systems, 34, 852-863.

StyleGAN v3



Tero Karras et al. (2020) "Alias-Free Generative Adversarial Networks". Advances in neural information processing systems, 34, 852-863.

StyleGAN-T



Axel Sauer et al. (2023) "StyleGAN-T: Unlocking the Power of GANs for Fast Large-Scale Text-to-Image Synthesis". Proceedings of the 40th International Conference on Machine Learning (ICML'23), Vol. 202. JMLR.org, Article 1250, 30105–30118

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

> Reinventa el mundo <

GRACIAS

Victor Flores Benites

TRANSFORMATEC