

---

# Visión por Computadora II - CEAI - FIUBA



Profesores:

- Cavalieri Juan Ignacio - [juanignaciocavalieri@gmail.com](mailto:juanignaciocavalieri@gmail.com)
- Cornet Juan Ignacio - [juanignaciocornet@gmail.com](mailto:juanignaciocornet@gmail.com)
- Khodadad Pakdaman - [khodadad.pakdaman@gmail.com](mailto:khodadad.pakdaman@gmail.com)

# Quinta clase:

---

- Transferencia de estilos
- Autoencoders
- Generación de imágenes y aplicaciones

# Transferencia de estilo

---

A



B



"A Neural Algorithm of Artistic Style" . [Link](#)



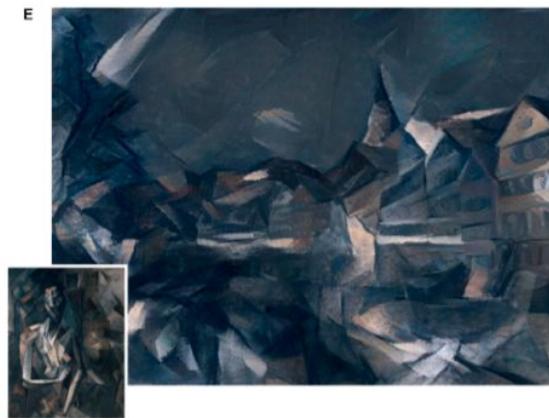
C



D



E



F



# ¿Qué aprenden las ConvNets?

---

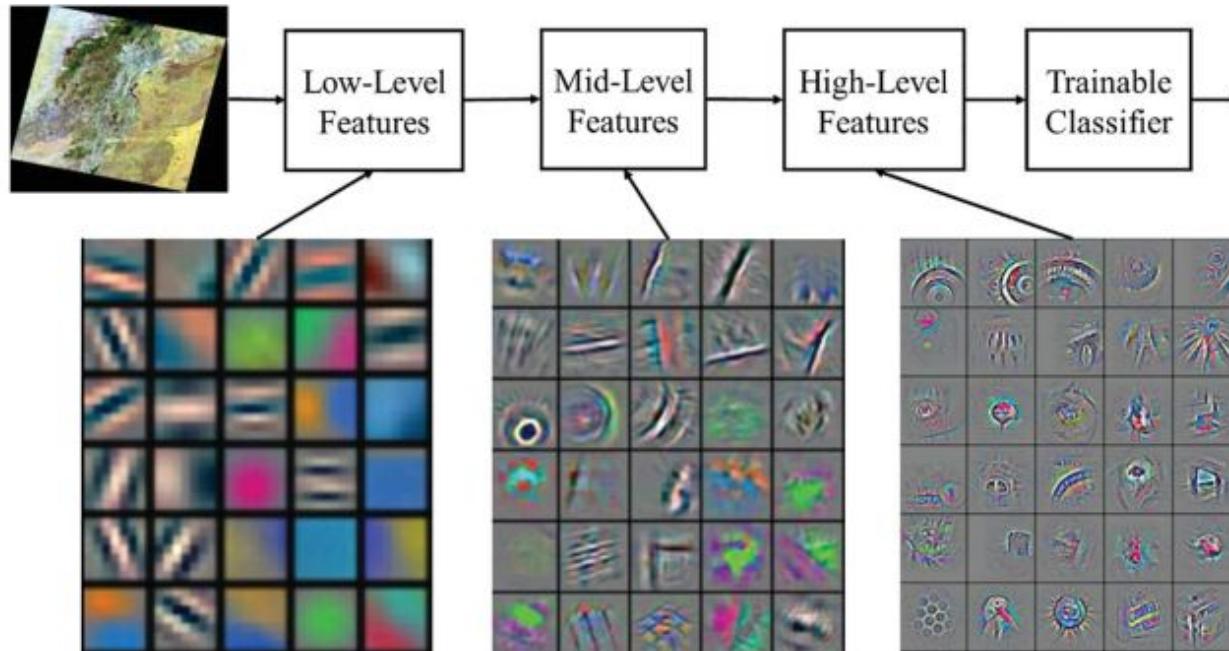


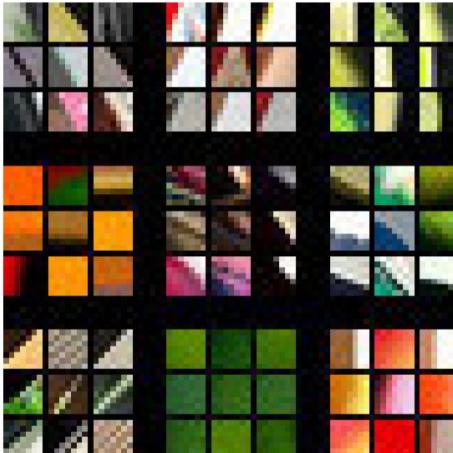
Fig. 3. Deep Learning: Representations are hierarchical and trained (LeCun et al., 2015).

# ¿Qué aprenden las ConvNets?

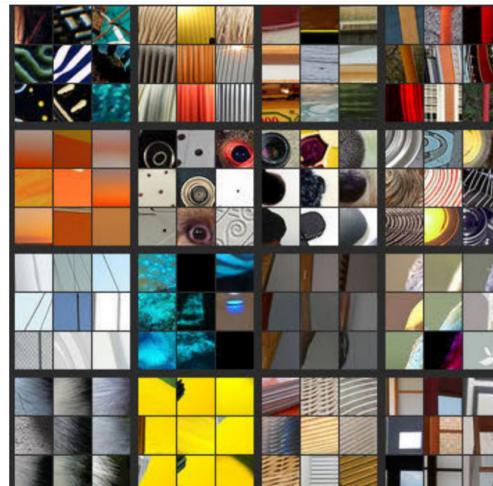
---

- Mirar patches que maximizan activaciones
- ¿Qué pasa en diferentes capas? Se hacen más complejos. Ejemplo de red similar a AlexNet:

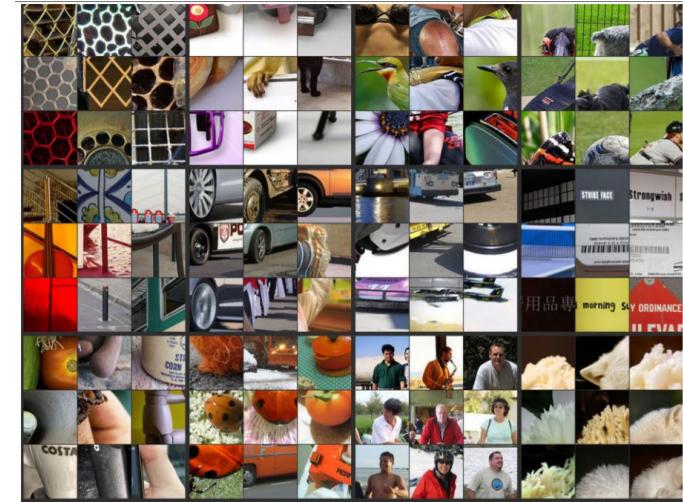
Capa 1



Capa 2



Capa 3

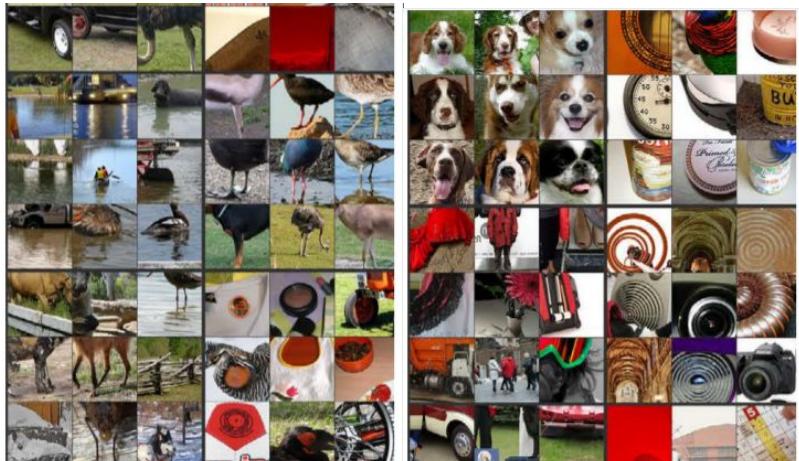


“Visualizing and understanding Convolutional Networks”. [Link](#)

# ¿Qué aprenden las ConvNets?

---

Capa 4:



Capa 5:



# Cómo definir la función de costo

---



Contenido: C

Estilo: S

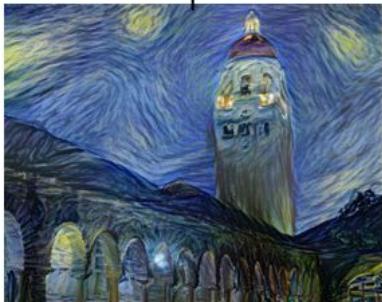


Imagen generada: G

$$J(G) = \alpha J_{\text{content}}(C, G) + \beta J_{\text{style}}(S, G)$$

# Procedimiento

---

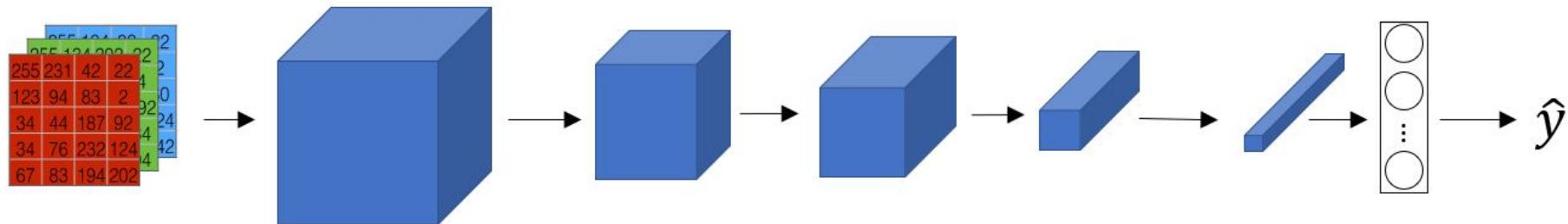
- Inicialización aleatoria de una imagen  $G$
- Descendemos en la dirección del gradiente de  $J(G)$
- $G := G - \partial J / \partial G (G)$

# Función de costo de contenido

---

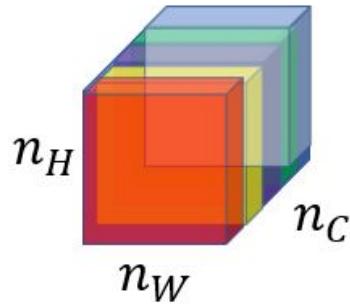
- Consideremos una capa intermedia  $l$  de una red de reconocimiento de imágenes
- Usamos una red pre-entrenada VGG, o Resnets, etc.
- Supongamos que  $a^{[l]}(C)$  y  $a^{[l]}(G)$  son las activaciones correspondientes a  $C$  y  $G$
- Si estas dos activaciones son similares entonces las imágenes tienen contenido similar
- $J_{\text{content}}(G) = \|f(a^{[l]}(C)) - f(a^{[l]}(G))\|^2$

# Función de costo de estilo

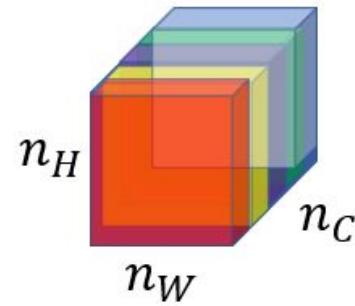


- Imaginemos que queremos usar la activación de la capa / para el **estilo**. Definimos el estilo como la correlación de activaciones a lo largo de los canales
- Pregunta: ¿cuán correlacionadas están las activaciones a lo largo de canales?
- Usando eso calculamos la ‘distancia de estilo’

## Imagen estilo



## Imagen generada



- Queremos que las correlaciones entre canales sean parecidas en ambas imágenes
- ¿Cómo logramos esto?

Sea  $a_{i,j,k}^{[l]} = \text{activación en } (i,j,k)$ .  $G^{[l]}$  es  $n_c^{[l]} \times n_c^{[l]}$

Gram matrix:

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

$$G^{[l]} = A^{[l]} (A^{[l]})^T$$

Usamos la norma de Frobenius entre matrices como distancia:

$$J_S^{[l]}(G^{[l](S)}, G^{[l](G)}) = \frac{1}{4(n_W^{[l]} n_H^{[l]})^2} \left\| G^{[l](S)} - G^{[l](G)} \right\|_{\mathcal{F}}^2 \quad \|G\|_{\mathcal{F}} = \sqrt{\sum_{ij} (g_{ij})^2}$$

Hacemos esto con varias capas:

$$J_S(\mathbf{x}, \mathbf{y}) = \sum_{l=0}^L \lambda_l J_S^{[l]}(G^{[l](S)}, G^{[l](G)})$$

# Ejemplo de programación

---

# Generación de imágenes

---

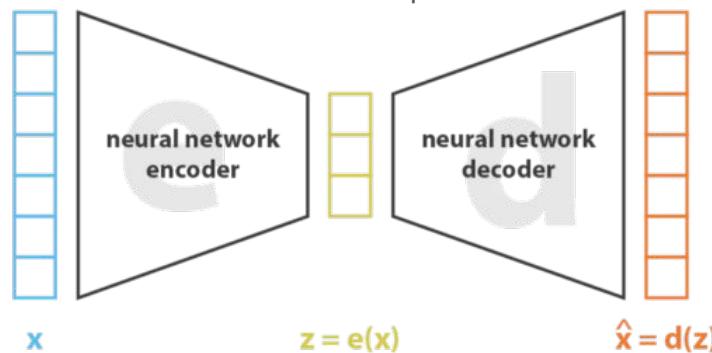
- Generación de imágenes parecidas a las de un dataset

# Autoencoders

---

Son algoritmos de compresión de datos donde el mecanismo de compresión y descompresión es:

- Específico de los datos
- Aprendido automáticamente a partir de ejemplos
- Inherenteamente tiene asociado una pérdida de información



---

$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$

# Usos de los autoencoders

---

A pesar de que los autoencoders no presentan mejoras en cuanto a compresión si los comparamos con algoritmos específicos de compresión de imágenes, los mismo son ampliamente usados, principalmente en:

- Eliminación de ruido en imágenes
- Reducción de dimensionalidad para visualización de datos

Representa un ejemplo de aprendizaje autosupervisado.

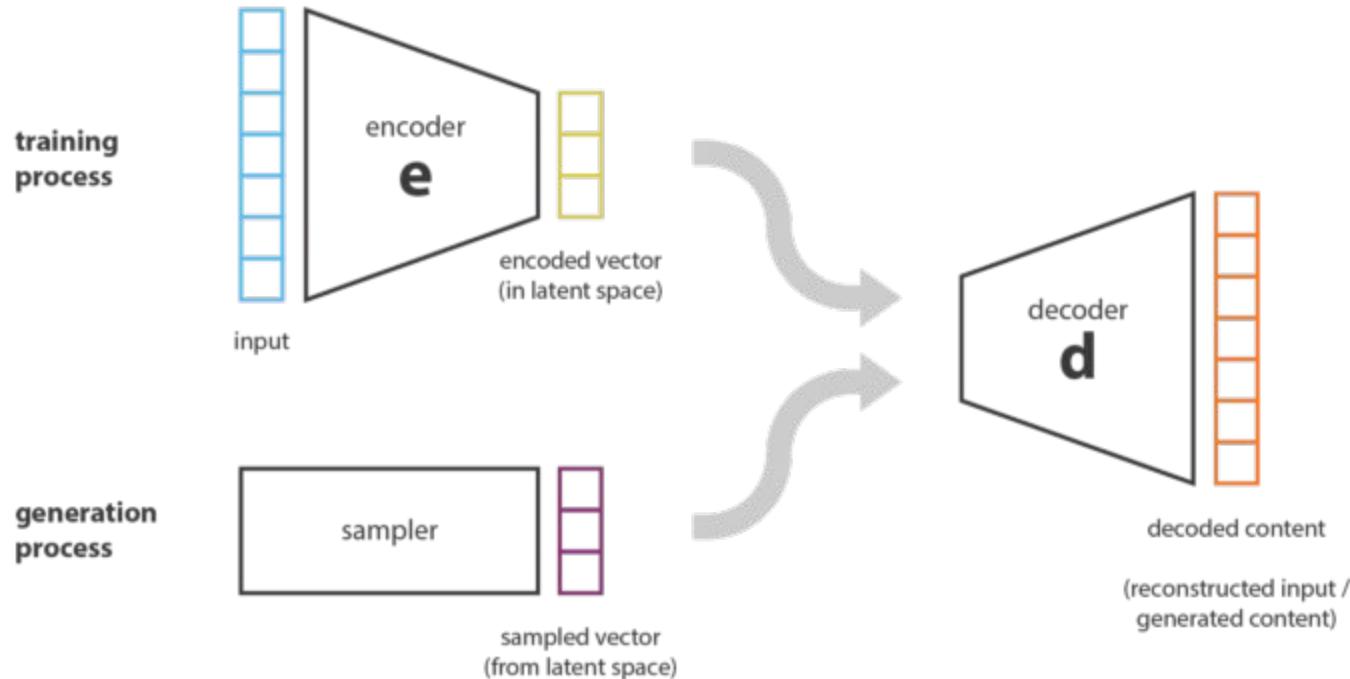
Puede agregarse regularización para forzar el aprendizaje de representaciones esparsas.

- Regularización L1
- Divergencia KL

Puede utilizarse con otras arquitecturas, e.g. con LSTM para trabajar con secuencias.

# Limitaciones de los autoencoders

---



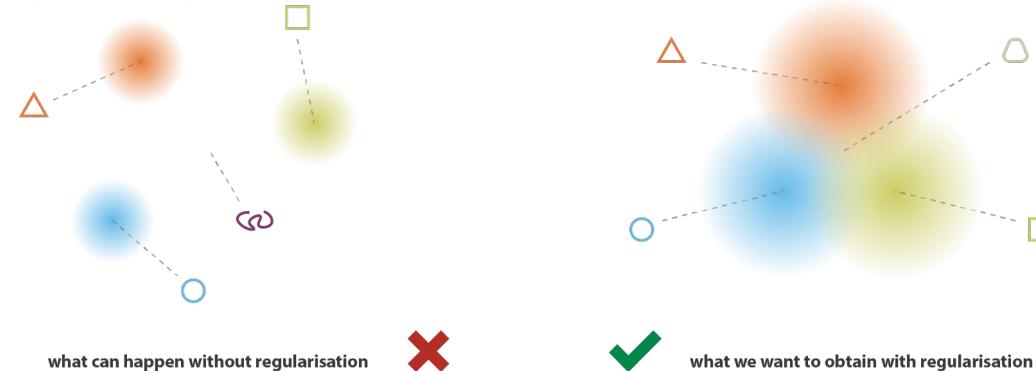
# Limitaciones de los autoencoders

---

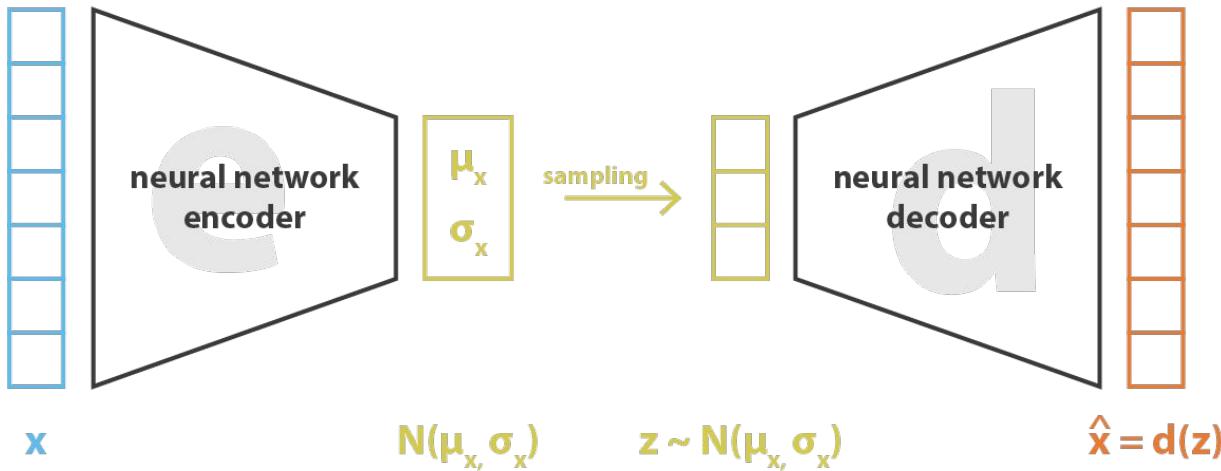
El alto grado de libertad del autoencoder que hace posible codificar y decodificar sin pérdida de información (a pesar de la baja dimensionalidad del espacio latente) conduce a un severo **sobreajuste** que implica que algunos puntos del espacio latente darán contenido sin sentido una vez decodificados.

El autoencoder se entrena únicamente para codificar y decodificar con el menor número de pérdidas posible, sin importar cómo esté organizado el espacio latente, por lo que al no prestar atención a la estructura, la red aprovechará el sobreajuste para minimizar la función de costo.

Por lo tanto, un autoencoder así como fue presentado, no presenta un buen mecanismo para generar imágenes nuevas a partir de una descripción del espacio latente.



# Autoencoder variacional



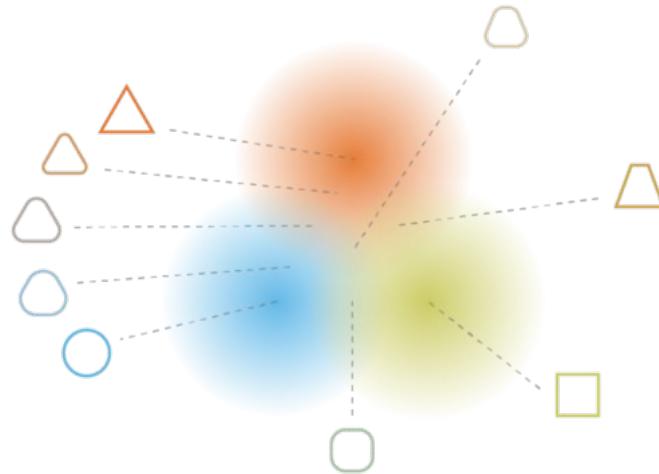
---

$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

# Autoencoder variacional

---

La regularización que se espera del espacio latente para hacer posible el proceso generativo puede expresarse a través de dos propiedades principales: la **continuidad** (dos puntos cercanos en el espacio latente no deben dar dos contenidos completamente diferentes una vez decodificados) y la **completitud** (para una distribución elegida, un punto muestrado del espacio latente debe dar un contenido "significativo" una vez descodificado).

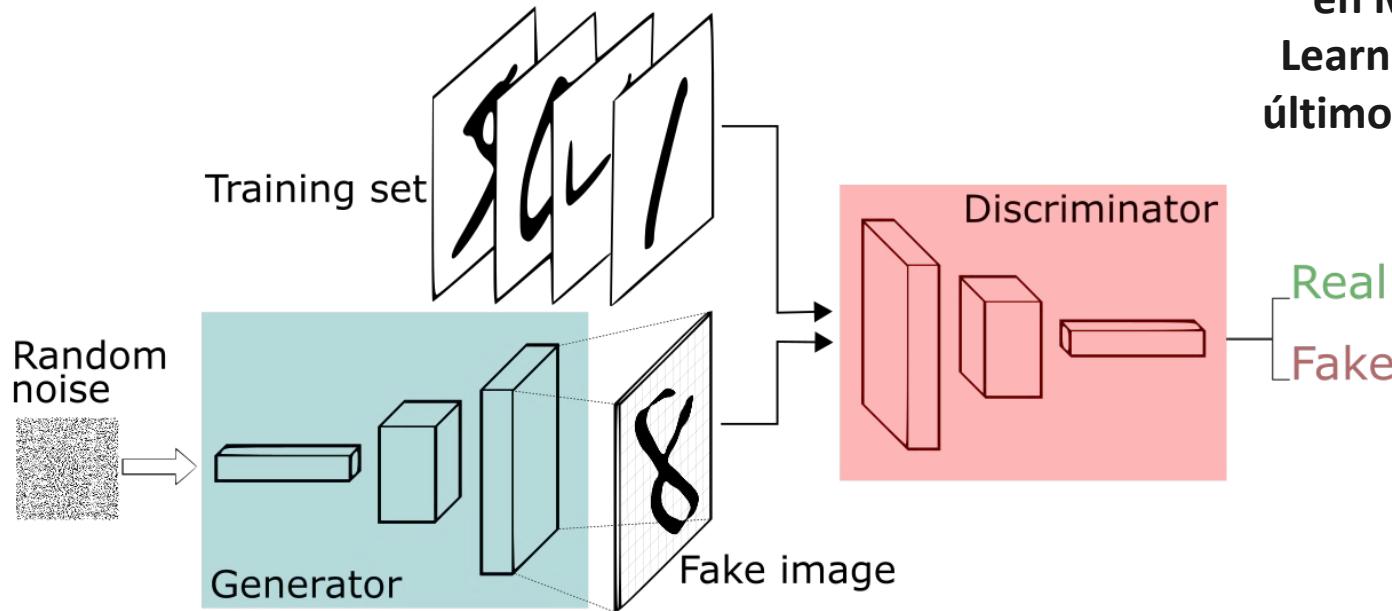


# Ejemplo de programación

---

# Generative Adversarial Networks

“La idea más cool  
en Machine  
Learning en los  
últimos 20 años”



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

---

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator,  $k$ , is a hyperparameter. We used  $k = 1$ , the least expensive option, in our experiments.

---

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

**end for**

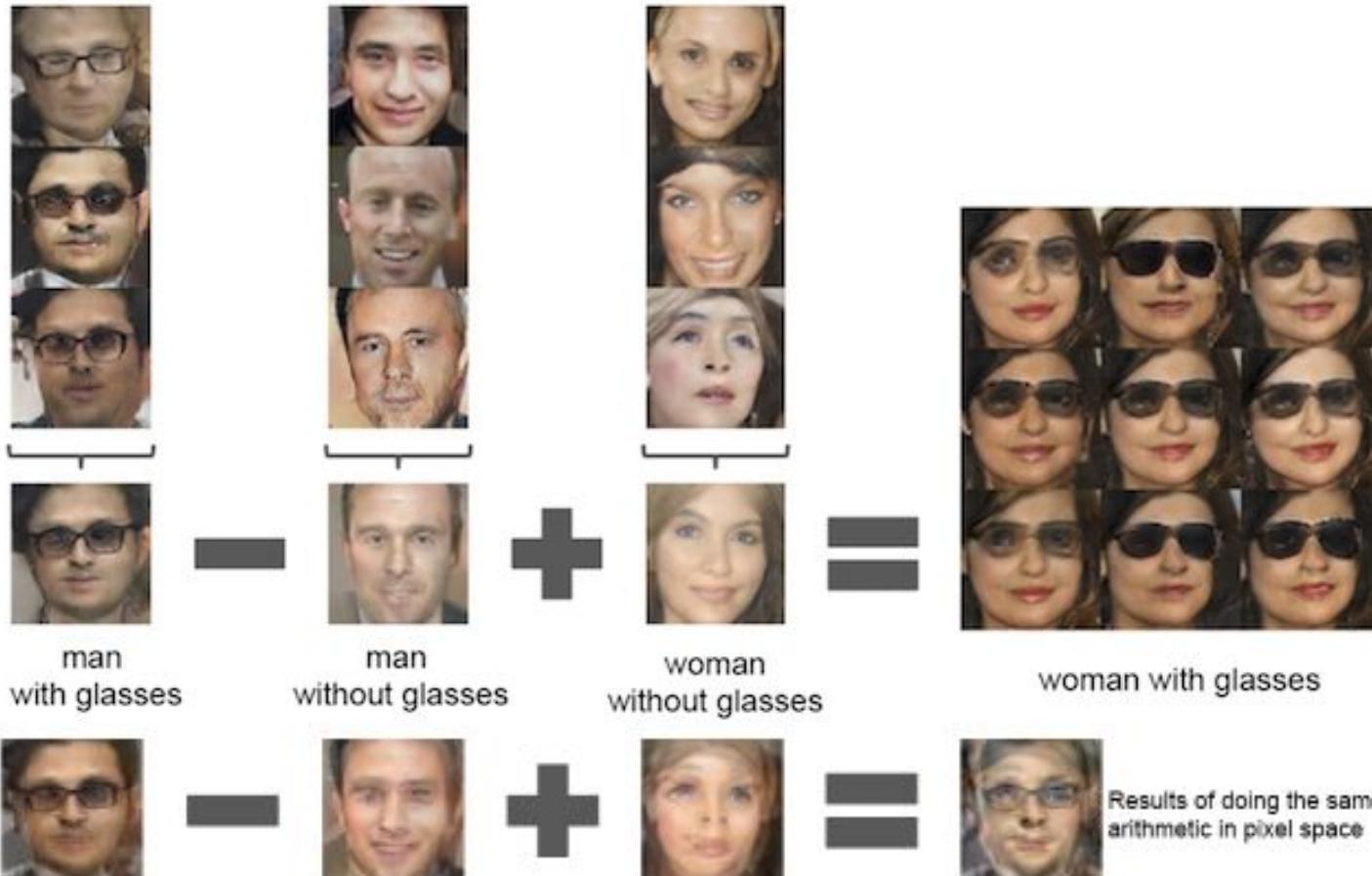
- Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
- Update the generator by descending its stochastic gradient:

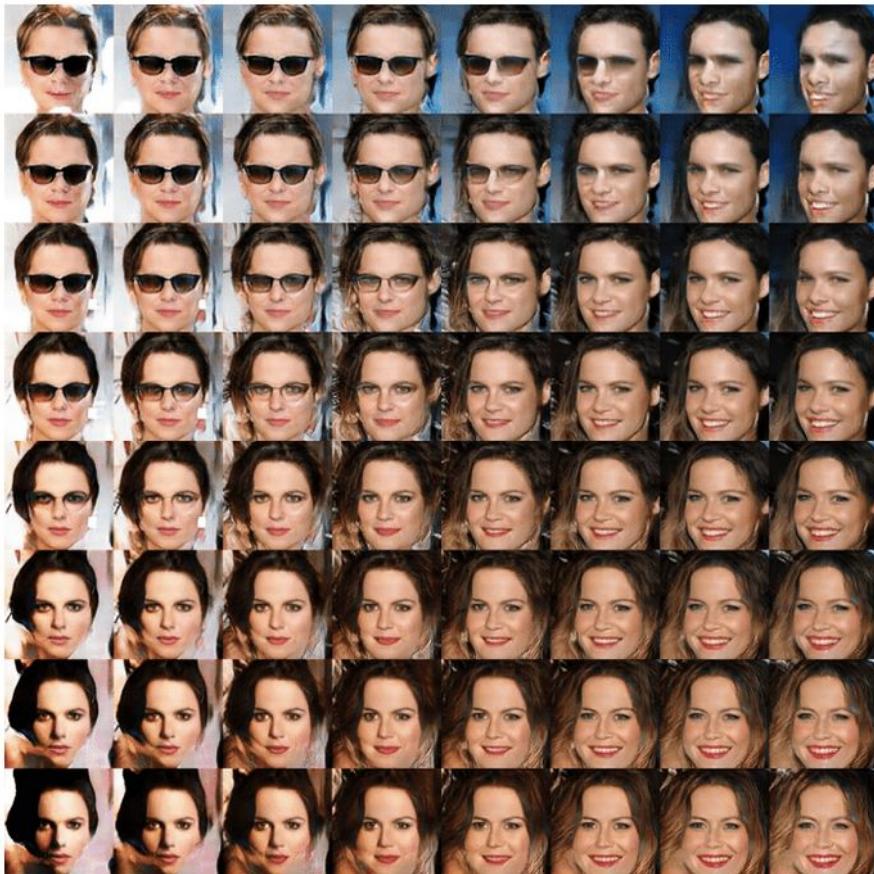
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

---





Al interpolar dos puntos diferentes en el espacio latente  $Z$ , podemos generar numerosos ejemplos intermedios que conserven características de las imágenes de referencia, en mayor o menor medida.

# Colapso de modo

---



7 1 7 / / 1 / 7  
9 1 1 1 1 9 1 /  
8 / 1 / / 7 1 9  
1 / 1 / 4 7 / /  
1 / 7 1 8 7 8 7  
1 1 1 9 1 1 1 7  
1 9 7 9 7 1 1 1  
1 1 1 1 1 1 1 2

# Colapso de modo

---

10k steps	20k steps	50K steps	100k steps

# Comparación de distribuciones

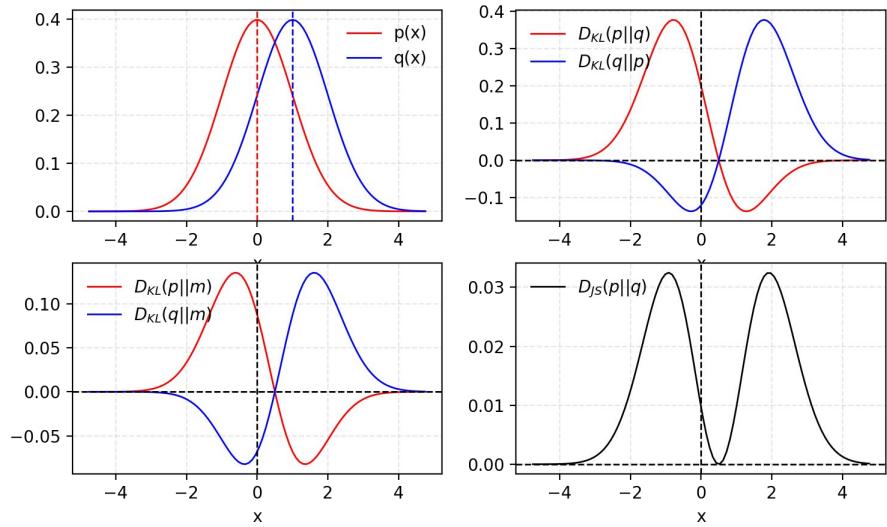
---

Kullback–Leibler divergence:

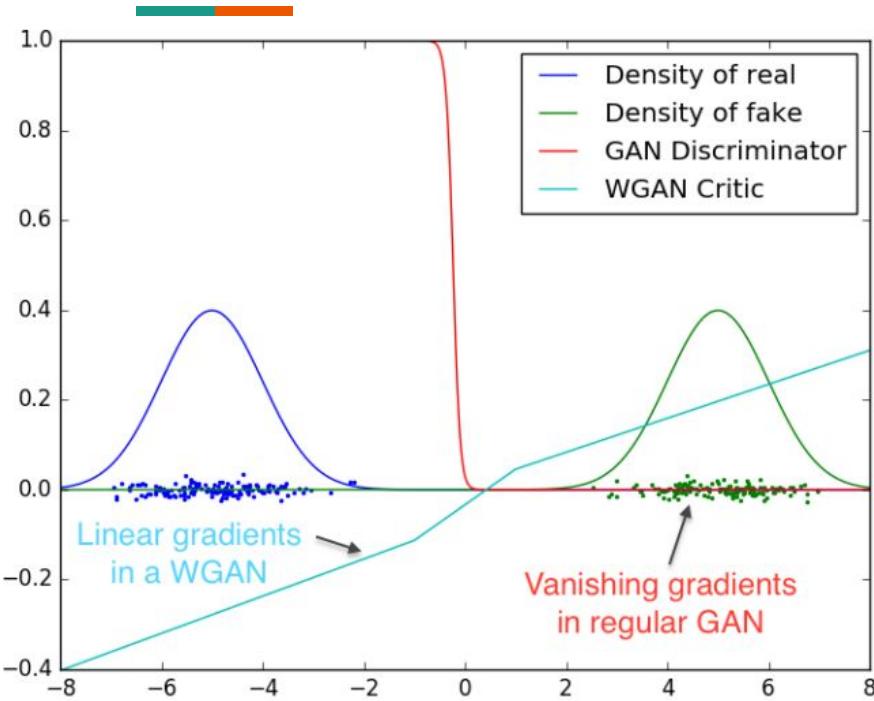
$$D_{KL}(P||Q) = \sum_{x=1}^N P(x) \log \frac{P(x)}{Q(x)}$$

Jensen–Shannon divergence:

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

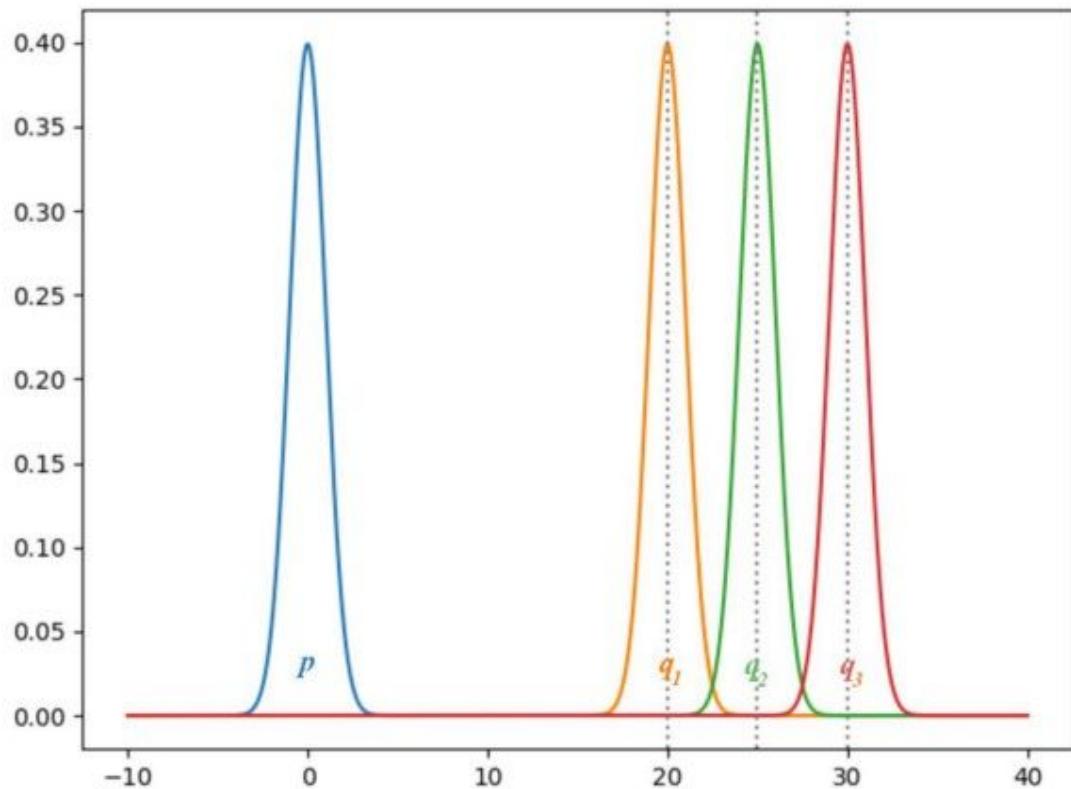


# Wasserstein GAN

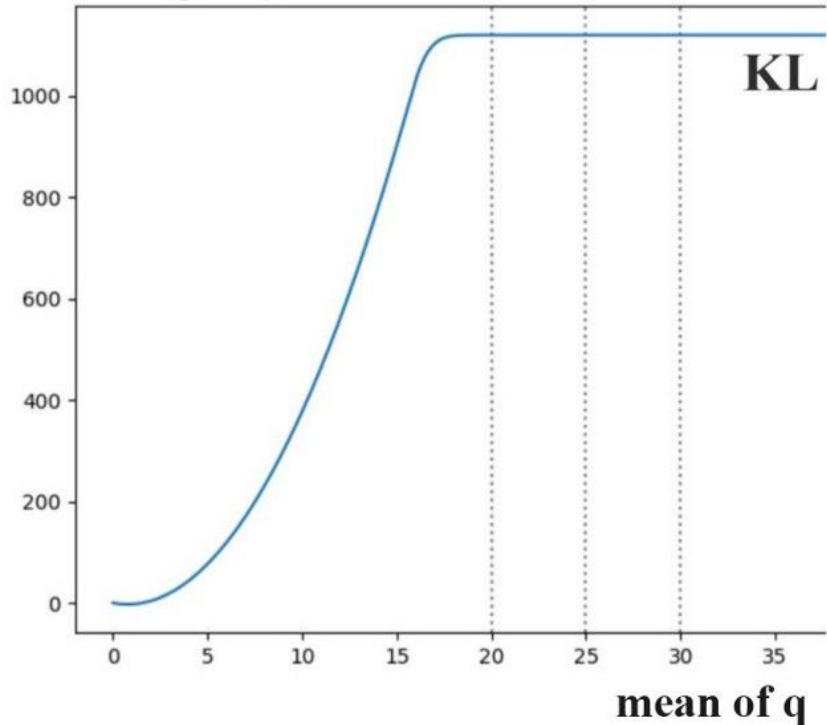


El entrenamiento de un GAN se enfrenta a un dilema:

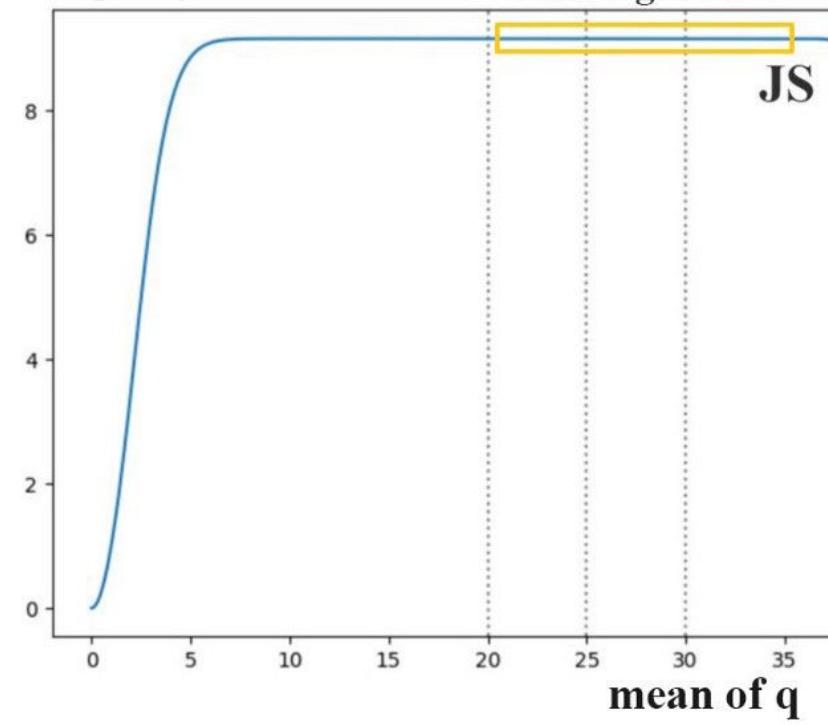
- Si el discriminador se comporta mal, el generador no tiene una retroalimentación precisa y la función de pérdida no puede representar la realidad.
- Si el discriminador hace un gran trabajo, el gradiente de la función de pérdida cae hasta cerca de cero y el aprendizaje se vuelve súper lento o incluso se atasca.



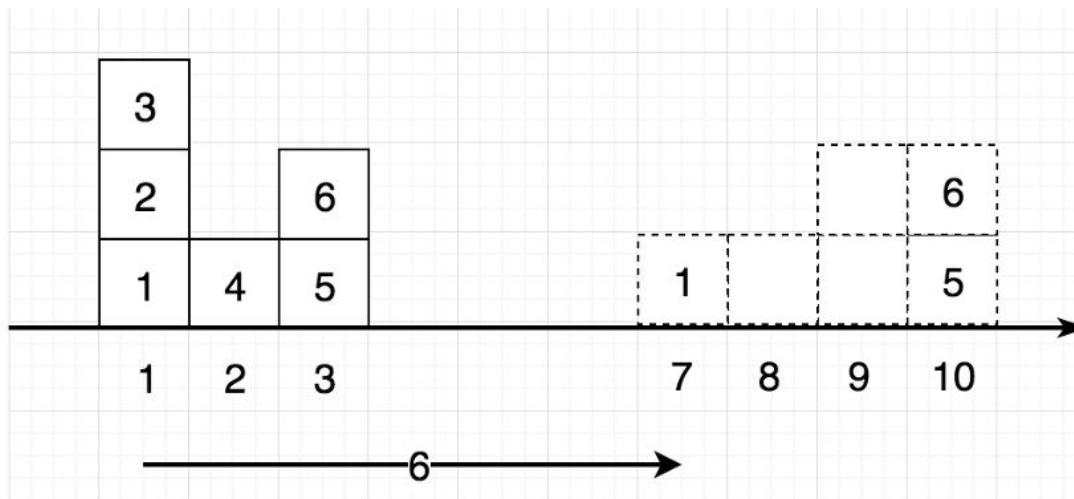
**KL-divergency**



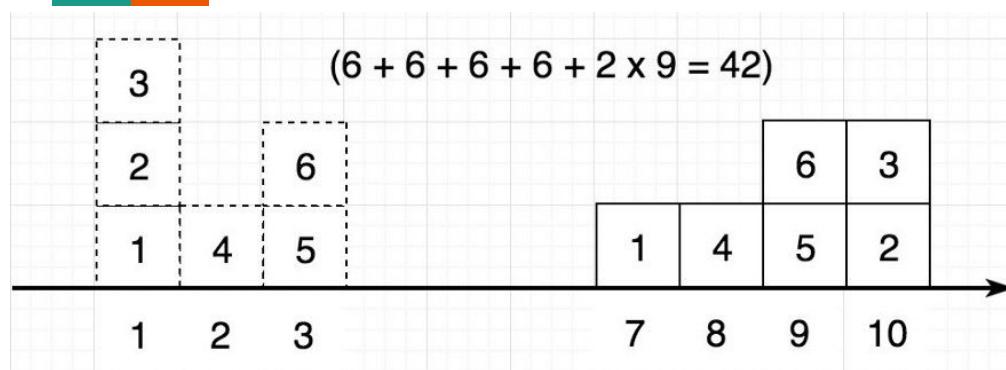
**JS-divergency**



# Earth Mover Distance

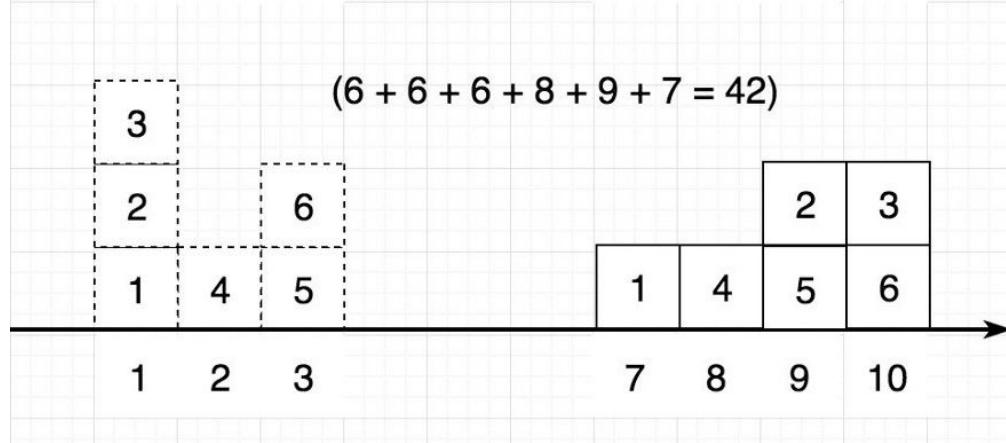


# Earth Mover Distance



	7	8	9	10
1	1	0	0	2
2	0	1	0	0
3	0	0	2	0

	7	8	9	10
1	1	0	1	1
2	0	1	0	0
3	0	0	1	1



# Earth Mover Distance

---

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|],$$

$\Pi(\mathbb{P}_r, \mathbb{P}_g)$  representa las distribuciones conjuntas  $\gamma(x, y)$  cuyas distribuciones marginales son  $\mathbb{P}_r$  y  $\mathbb{P}_g$ .

		7	8	9	10	$\Pi$
		1	0	0	2	
		2	1	0	0	$\gamma_1$
x		0	0	2	0	
	y	1	0	0	2	

		7	8	9	10	$\Pi$
		1	0	0	2	
		2	0	1	0	$\gamma_2$
x		0	1	1	0	
	y	1	0	1	0	

# Cómputo de la distancia de Wasserstein

---

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

usando la dualidad de Kantorovich-Rubinstein,  
podemos simplificar los cálculos a,

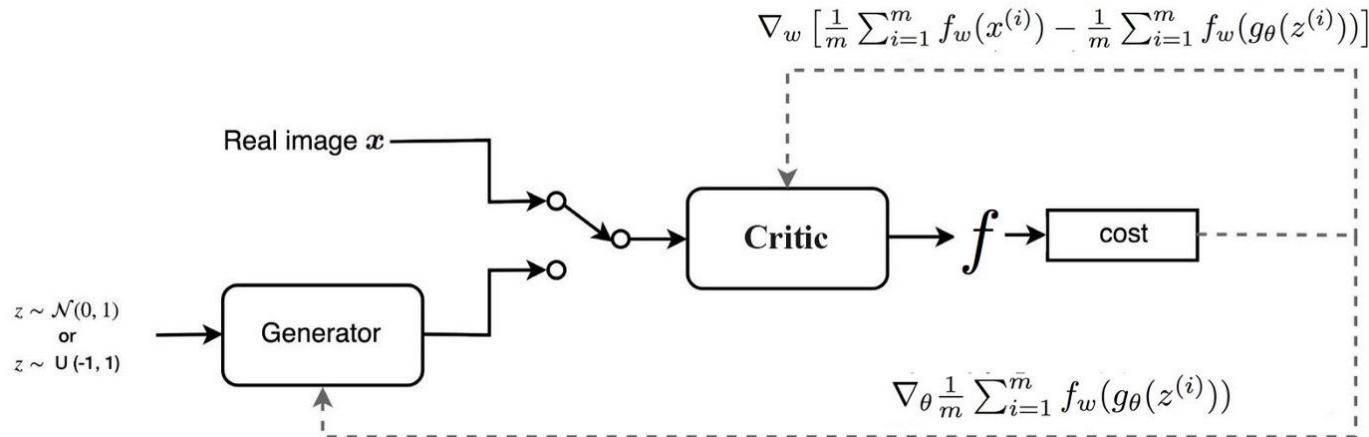
$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} [f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta} [f(x)]$$

$f$  es una función  
1-Lipschitz si:

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|$$

# WGAN

Así que para calcular la distancia de Wasserstein, sólo tenemos que encontrar una función de 1-Lipschitz. Al igual que otros problemas de aprendizaje profundo, podemos construir una red profunda para aprenderla.



De hecho, esta red es muy similar al discriminador D, sólo que sin la función sigmoidea y produce una puntuación escalar en lugar de una probabilidad.



## Discriminator/Critic

**GAN**

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D \left( \mathbf{x}^{(i)} \right) + \log \left( 1 - D \left( G \left( \mathbf{z}^{(i)} \right) \right) \right) \right]$$

**WGAN**

$$\nabla_w \frac{1}{m} \sum_{i=1}^m \left[ f \left( x^{(i)} \right) - f \left( G \left( z^{(i)} \right) \right) \right]$$

## Generator

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

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f \left( G \left( z^{(i)} \right) \right)$$

# Weight Clipping

---

WGAN aplica un mecanismo muy sencillo para restringir el valor máximo del peso en  $\mathbf{f}$

$$w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$$

$$w \leftarrow \text{clip}(w, -c, c)$$

*"Weight clipping is a clearly terrible way to enforce a Lipschitz constraint"*

---

**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values  $\alpha = 0.00005$ ,  $c = 0.01$ ,  $m = 64$ ,  $n_{\text{critic}} = 5$ .

---

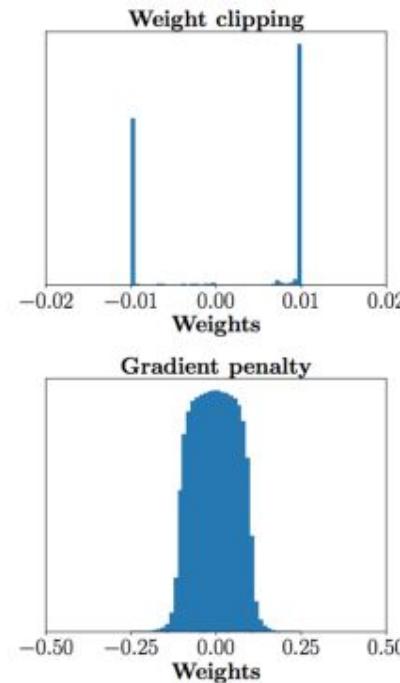
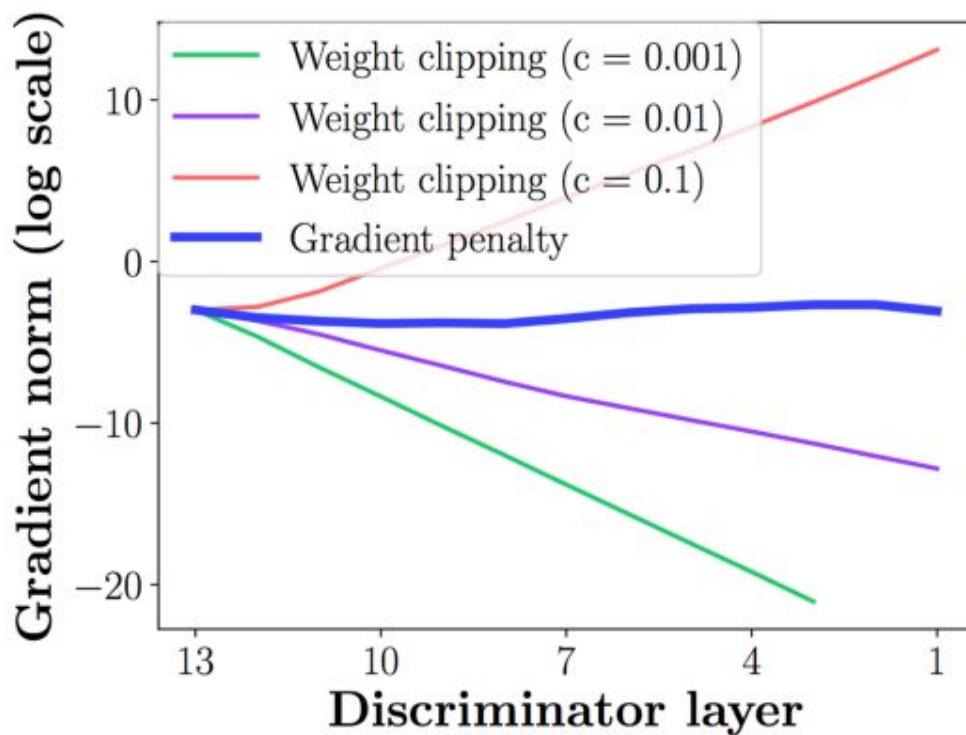
**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
12: end while
```

---

# Desventaja del Weight Clipping



# Muestreo uniforme para cálculo de gradiente

---

$$L = \underbrace{\mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x})]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}.$$

where  $\hat{\mathbf{x}}$  sampled from  $\tilde{\mathbf{x}}$  and  $\mathbf{x}$  with  $t$  uniformly sampled between 0 and 1

$$\hat{\mathbf{x}} = t \tilde{\mathbf{x}} + (1 - t) \mathbf{x} \text{ with } 0 \leq t \leq 1$$

---

**Algorithm 1** WGAN with gradient penalty. We use default values of  $\lambda = 10$ ,  $n_{\text{critic}} = 5$ ,  $\alpha = 0.0001$ ,  $\beta_1 = 0$ ,  $\beta_2 = 0.9$ .

---

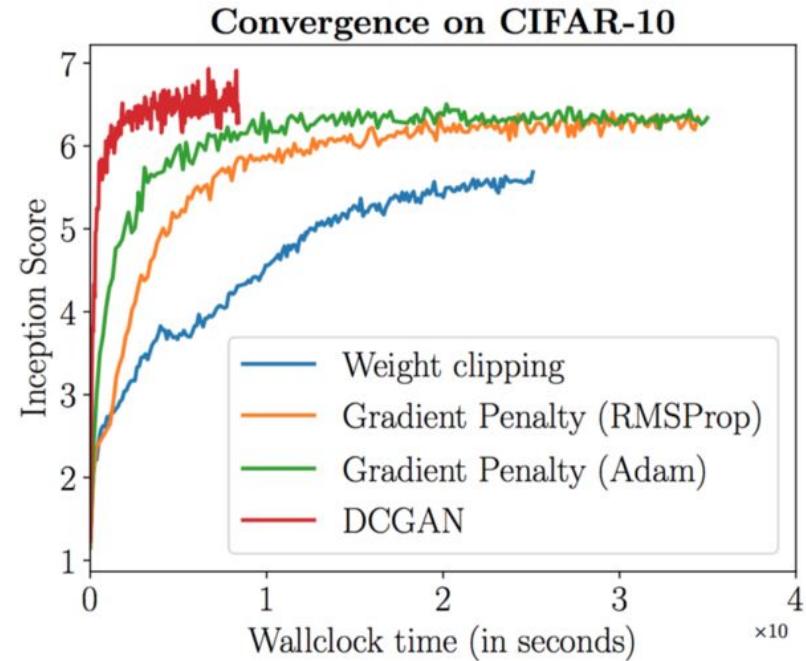
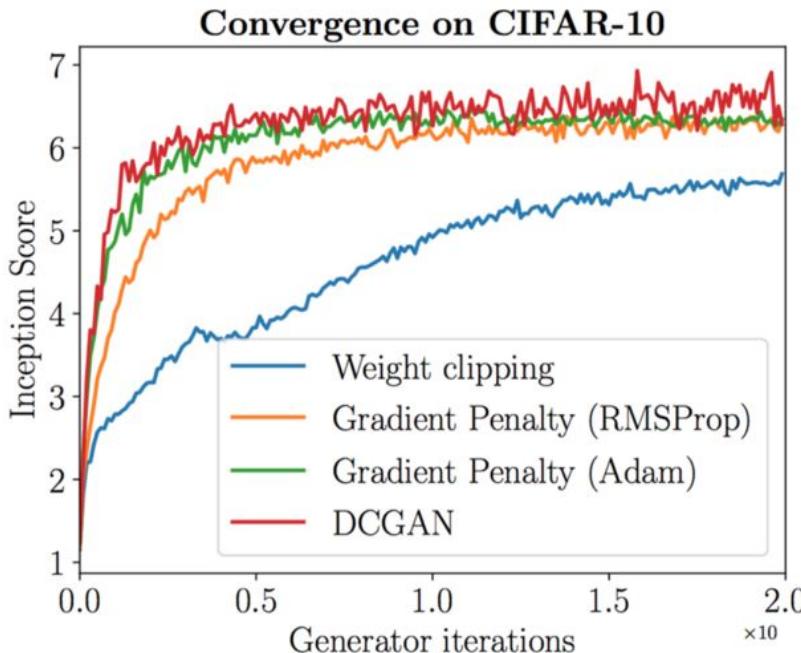
**Require:** The gradient penalty coefficient  $\lambda$ , the number of critic iterations per generator iteration  $n_{\text{critic}}$ , the batch size  $m$ , Adam hyperparameters  $\alpha, \beta_1, \beta_2$ .

**Require:** initial critic parameters  $w_0$ , initial generator parameters  $\theta_0$ .

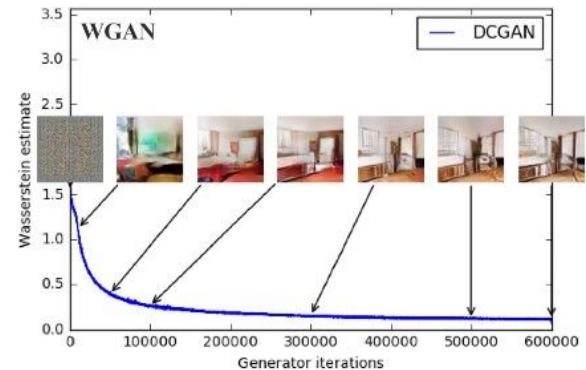
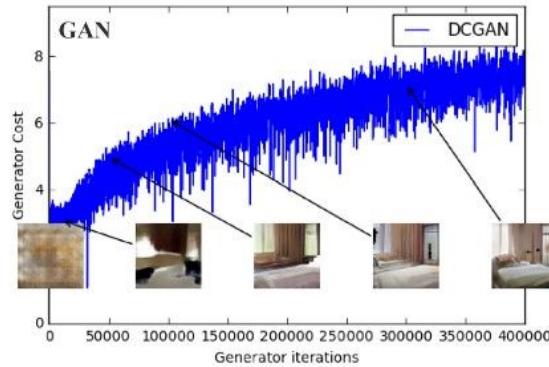
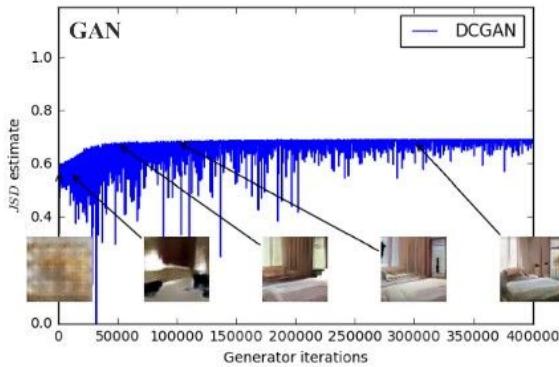
```
1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $\mathbf{x} \sim \mathbb{P}_r$ , latent variable  $\mathbf{z} \sim p(\mathbf{z})$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z})$ 
6:        $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon) \tilde{\mathbf{x}}$ 
7:        $L^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:   end for
11:   Sample a batch of latent variables  $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$ .
12:    $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\mathbf{z})), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while
```

---

# Comparacion de metodos



# Comparacion de metodos



# Evolucion a traves de los años

---



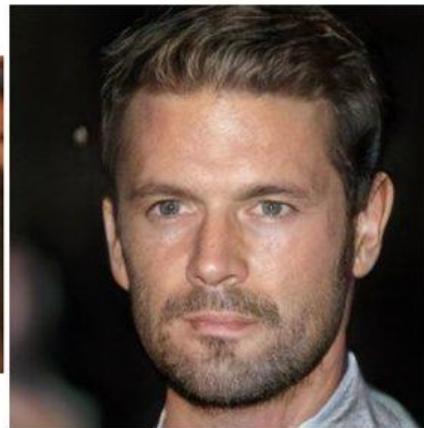
2014



2015



2016

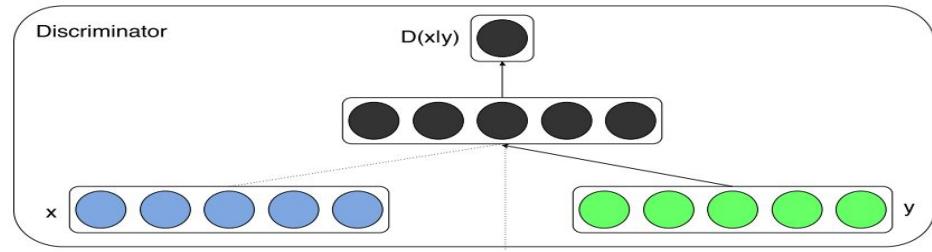
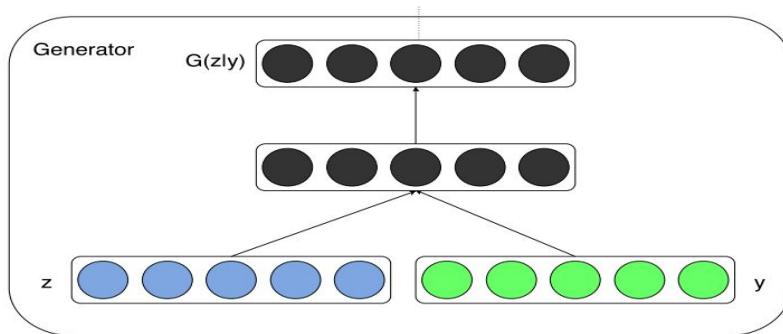


2017



2018

# Conditional GAN



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



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

# Ejemplo de programación

---

Revisar repositorio:

- <https://github.com/LynnHo>
- <http://gaugan.org/gaugan2/>

# Evaluacion de Generadores

---

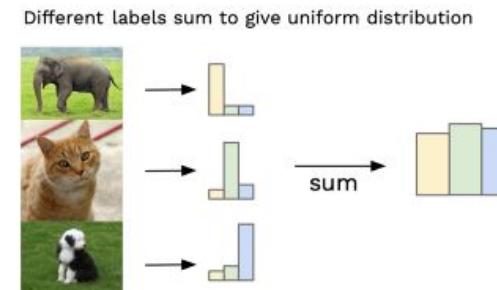
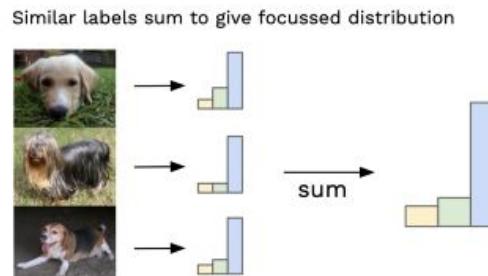
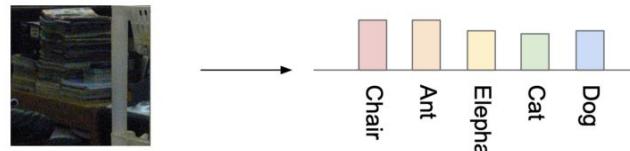
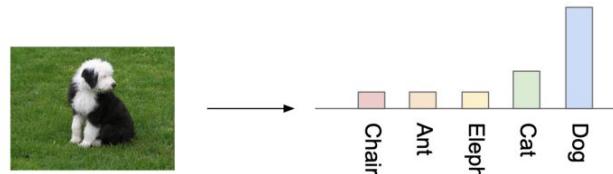


# Inception Score (IS)

Esta métrica toma su nombre de la red neuronal Inception, ya que utiliza un modelo pre entrenado de esta red para estimar que tan bien funciona un generador.

Para ellos se calculan las salidas del clasificador para un conjunto de muestras representativo de mi generador.

El clasificador debe tener entre sus clases aquellas comprendidas entre las que puede generar el generador.



# Inception Score (IS)

---

El punto final consiste en comparar estas distribuciones y combinarlas en una sola métrica. Esto lo hacemos a partir de la divergencia de Kullback-Leibler.

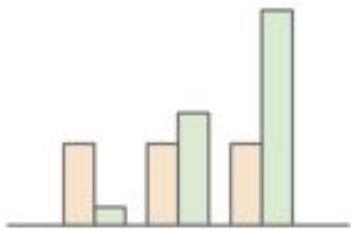
$$D(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}.$$

$$\text{IS}(G) = \exp \left( \mathbb{E}_{\mathbf{x} \sim p_g} D_{KL}(\, p(y|\mathbf{x}) \parallel p(y) \,) \right),$$

# Inception Score (IS)

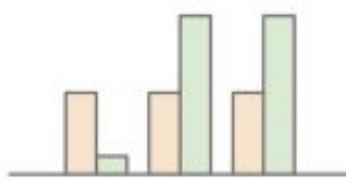
---

High KL divergence



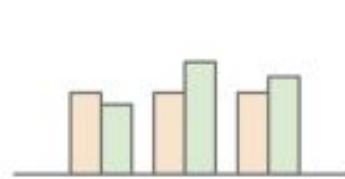
Ideal situation

Medium KL divergence



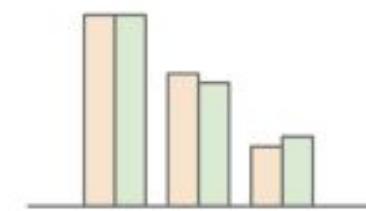
Generated images are  
not distinctly one  
label

Low KL divergence



Generated images are  
not distinctly one  
label

Low KL divergence



Generator lacks  
diversity

Label distribution

Marginal distribution

# Inception Score (IS)

---

## Limitaciones:

1. La métrica está limitada a aquellas clases que la red Inception está entrenada para clasificar.
  - a. Si el generador está aprendiendo a generar algo no presente en las clases de salida de Inception, el IS será siempre bajo.
  - b. Si el generador está aprendiendo a generar algo con etiquetas más específicas que las de Inception, el IS será siempre bajo.
2. Si el generador genera una única imagen por clase, repitiendo muchas veces la misma imagen, el IS será alto
3. Si el generador memoriza los datos de entrenamiento y los repite, el IS será alto

# Frechet Inception Distance (FID)

---

El FID intenta sobreponerse a las desventajas que presenta el IS, evaluando la similitud entre las imágenes generadas y las imágenes reales.

Para eso se utiliza la misma red Inception, pero esta vez se elimina la capa de clasificación y se toman las activaciones de las 2048 features que genera la red. Estas son analizadas estadísticamente a través del cálculo de la media y la covarianza de las mismas a través de las diferentes imágenes. Generadas y reales por separado.

$$FID = \|\mu_r - \mu_g\|^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2})$$

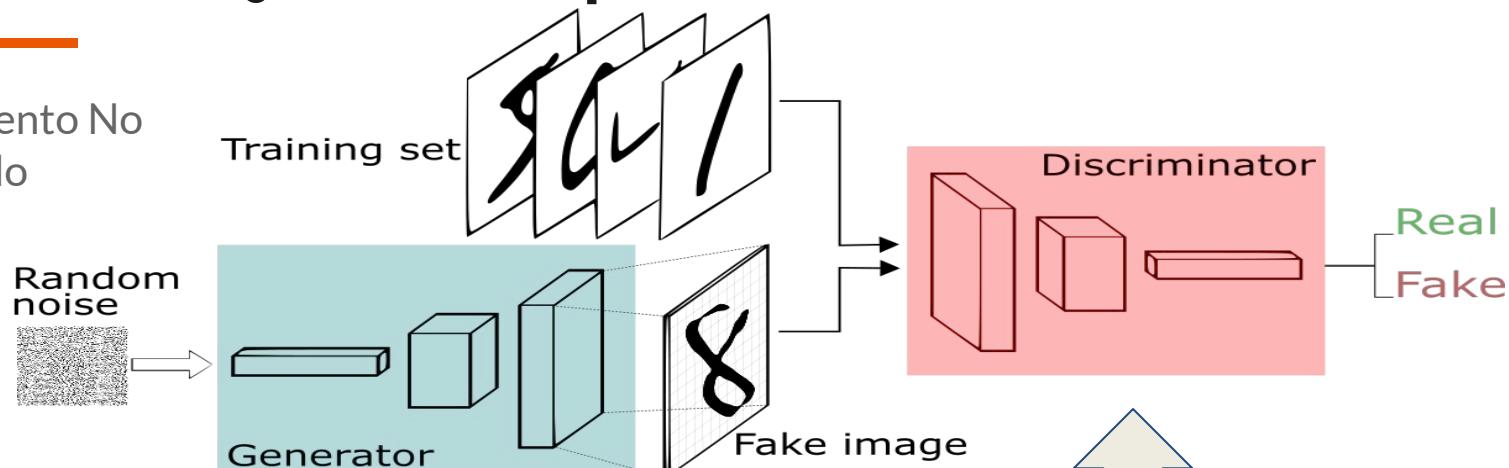
# Otras aplicaciones

---

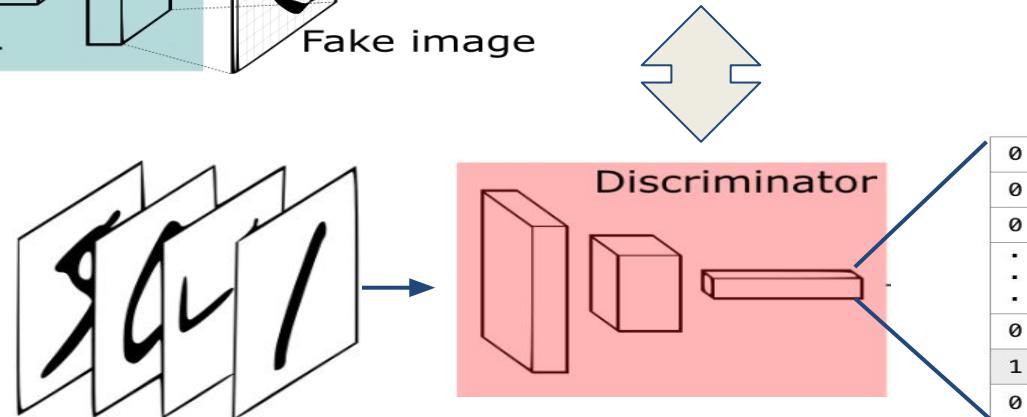


# Aprendizaje semi supervisado con GAN

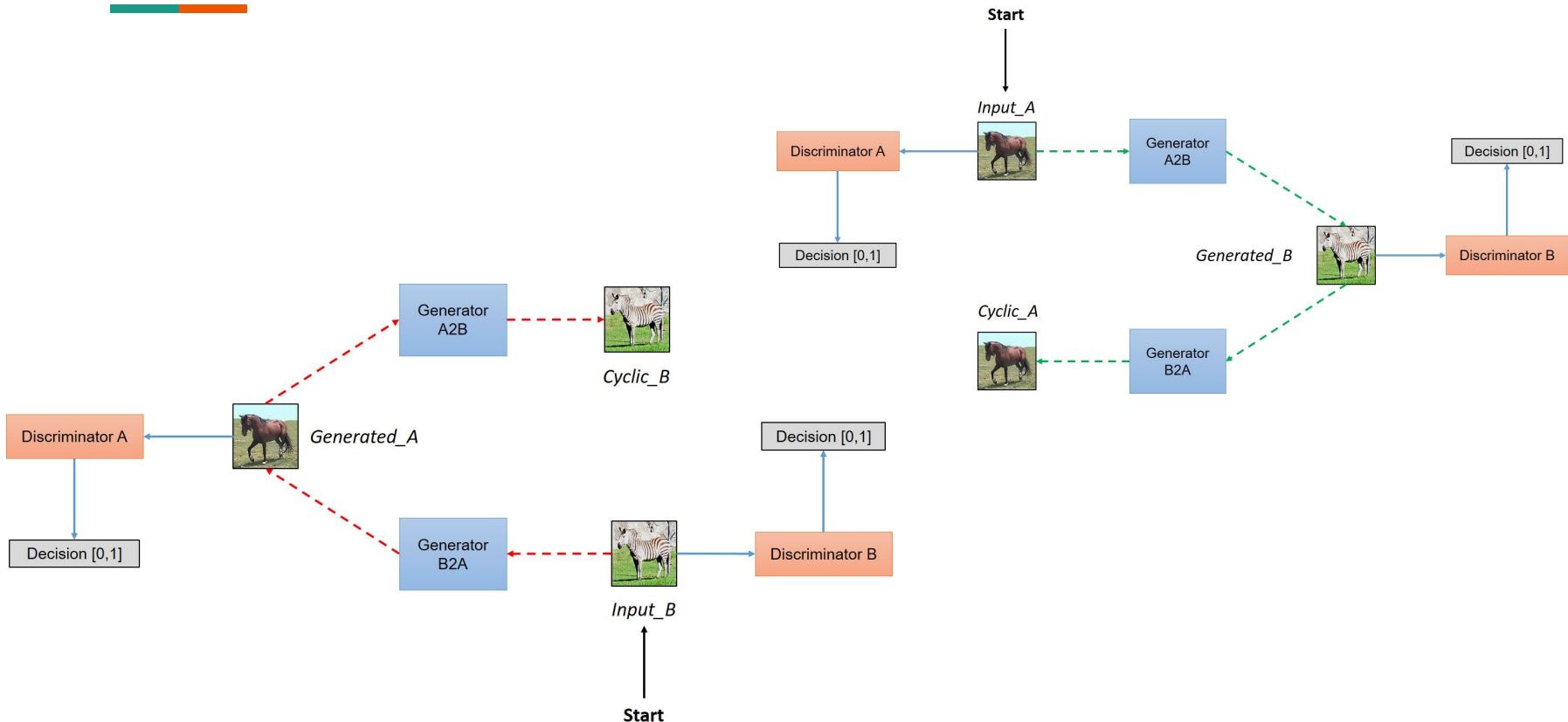
Entrenamiento No  
Supervisado

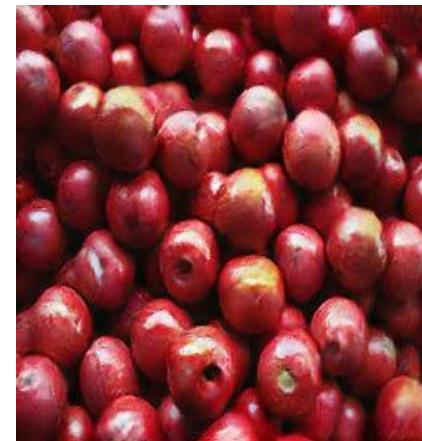
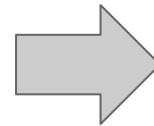
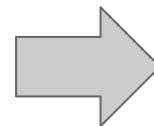


Entrenamiento  
Supervisado



# Cycle GAN





## CycleGAN conversion from Fortnite to PUBG



Fortnite



PUBG



FortG? PUBnite?

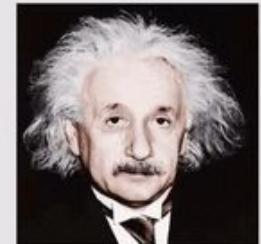
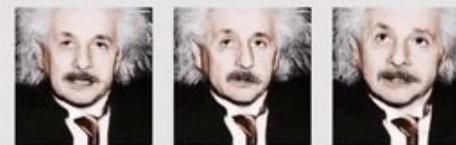
# Evolucion redes GAN

---

Living portraits



Living portraits



# Image Super Resolution

---

- El problema de Image Super Resolution, o super resolución de imágenes intenta reconstruir una imagen de alta resolución de una imagen de baja resolución
- En particular tenemos Single Image Super Resolution (SISR), que intenta hacerlo de una sola imagen
- En general la relación entre  $I^{LR}$  y  $I^{HR}$  depende de la situación
- Muchos estudios asumen que  $I^{LR}$  es una versión sub-sampleada de manera bicúbica de  $I^{HR}$ , pero hay otras transformaciones posibles: blur, ruido, etc.
- Papers:
  - [Enhanced Deep Residual Networks for Single Image Super-Resolution.](#)
  - [Wide Activation for Efficient and Accurate Image Super-Resolution,](#)
  - [Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network](#)
- Ejemplo de programación:
  - <https://github.com/krasserm/super-resolution>

# Ejemplos de aplicaciones en la industria

---

- Aplicaciones en la industria:
  - Colorización automática: <https://youtu.be/ys5nMO4Q0iY>
  - DeepFakes: [https://youtu.be/MVBe6\\_o4cMI](https://youtu.be/MVBe6_o4cMI)
  - Videocall Enhancement  
<https://blogs.nvidia.com/blog/2020/10/05/gan-video-conferencing-maxine/>
  - Aplicación práctica de super resolution: renderización 3D (ejemplo, juegos)

# NVIDIA Deep Learning Super Sampling (DLSS)

---



- <https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-2-0-a-big-leap-in-ai-rendering/>
- Mejora la imagen y a su vez la performance en más de 50%+ y a veces más de 70%
- <https://www.youtube.com/watch?v=ccPUj5cCs4c&feature=youtu.be>
- <https://youtu.be/KwDs6LrocR4>

# Trabajo Final

---