

# Generative Adversarial Networks (GANs)

---

## Generative Adversarial Networks (GANs)

---

### Introducción

# Introducción

## Modelos generativos

### Aprendizaje no supervisado:

Extraer conocimiento de un conjunto de datos  $x$  para el que no tenemos etiquetas  $y$ .

### Tareas más comunes:

- Clustering
- Autoencoders
- Modelos generativos

### Objetivo de los modelos generativos

Generar datos sintéticos pero realistas de alta dimensión  $\mathbf{x} \sim p_{\theta}(\mathbf{x})$ , que se asemejen lo más posible a la distribución desconocida de datos  $p(\mathbf{x})$ .

# Introducción

## Generative Adversarial Networks (GANs)

Las GANs son uno de los modelos de generación de datos de alta dimensión más populares. En ellas, **dos redes** se entrenan conjuntamente. Propuestas por Goodfellow et al. (2014).

Están compuestas de:

- Un **discriminador D** que clasifica ejemplos en 'reales' y 'falsos'.
- Un **generador G** para crear nuevos datos que 'engaños' al discriminador.

Estas redes se denominan **adversarias** por que los objetivos de ambas redes son antagónicos.

# Introducción

## Arquitectura:



**No existen dos D, el mismo discriminador evalúa ambos ejemplos.**

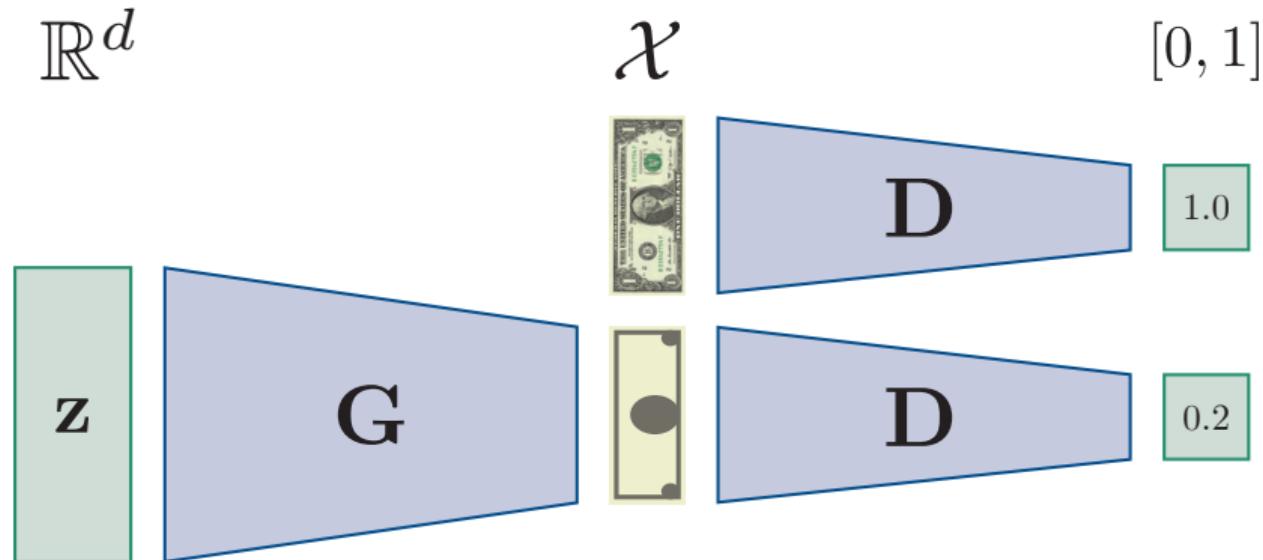
# Introducción

Analogía: Falsificador de billetes (**G**) y Policía (**D**)



# Introducción

Analogía: Falsificador de billetes (**G**) y Policía (**D**)



# Introducción

Analogía: Falsificador de billetes (**G**) y Policía (**D**)



# Introducción

Analogía: Falsificador de billetes (**G**) y Policía (**D**)



# Introducción

## Notación

Siendo  $\mathcal{X}$  el espacio de los datos de entrada, y  $d$  la dimensión del espacio latente.

**Red generadora  $\mathbf{G} : \mathbb{R}^d \rightarrow \mathcal{X}$ :**

- Mapea desde el espacio latente  $\mathbb{R}^d$  al espacio de los datos de entrada  $\mathcal{X}$ .
- Entrenada para que, dado un ruido Gaussiano  $\mathbf{z} \in \mathbb{R}^d$ , produzca una muestra que siga la distribución de los datos de entrada.

**Red discriminadora  $\mathbf{D} : \mathcal{X} \rightarrow [0, 1]$ :**

- Entrenada para **clasificar** si un ejemplo proviene del conjunto real o del generador  $\mathbf{G}$ .

# Introducción

## Notación

Dado un conjunto de puntos 'reales'

$$X = \{x_1, \dots, x_n\}$$

si consideramos  $\mathbf{G}$  fijo, podemos entrenar  $\mathbf{D}$  generando ejemplos 'falsos'

$$X' = \{x'_1, \dots, x'_n\}$$

mediante  $x' = \mathbf{G}(z)$ . Esto resultará en un conjunto de datos de clasificación binaria

$$\mathfrak{D} = \underbrace{\{(x_1, 1), \dots, (x_N, 1)\}}_{\text{Ejemplos reales}}, \underbrace{\{(\mathbf{G}(z_1), 0), \dots, (\mathbf{G}(z_N), 0)\}}_{\text{Ejemplos generados}}$$

donde  $z \sim \mathcal{N}(0, 1)$ .

## Optimizar discriminador

Resuelve un problema de clasificación binaria, por tanto necesita minimizar la función de pérdida **Binary Cross Entropy (BCE)**.

$$\min_{\mathbf{D}} \{-y \log \mathbf{D}(\mathbf{x}) - (1 - y) \log(1 - \mathbf{D}(\mathbf{x}))\}$$

Donde  $\mathbf{D}(\mathbf{x})$  es la predicción del discriminador resultado de haber aplicado previamente una función **sigmoide**

$$D(\mathbf{x}) = \frac{1}{(1 + e^{-o})}$$

a la salida  $o \in \mathbb{R}$  de la última capa del modelo.

# Optimizar generador

Para generar cada uno de los ejemplos falsos:

- Primero obtiene  $\mathbf{z} \in \mathbb{R}^d$  números aleatorios<sup>1</sup>, típicamente de una normal  $\mathbf{z} \sim \mathcal{N}(0, 1)$ .
- Aplica el generador a los valores resultantes del paso anterior  $\mathbf{x}' = G(\mathbf{z})$ .

El objetivo del generador es **engañar al discriminador** para que clasifique  $\mathbf{x}$  como datos reales, es decir, queremos que  $D(G(\mathbf{z})) \approx 1$ .

En otras palabras, para un discriminador  $\mathbf{D}$  dado tenemos que actualizar los pesos del generador  $\mathbf{G}$  para **maximizar** la cross-entropy cuando  $y = 0$  (ejemplo falso):

$$\max_{\mathbf{G}} \{-\log(1 - \mathbf{D}(\mathbf{G}(\mathbf{z})))\}$$

<sup>1</sup>Tradicionalmente llamamos a  $\mathbf{z}$  variable latente y a  $\mathbb{R}^d$  espacio latente.

## Minimax Loss

Podemos transformar la maximización previa en una minimización forzando al modelo para que prediga  $y = 1$  en los casos generados, lo que resulta en:

$$\min_{\mathbf{G}} \{-\log(\mathbf{D}(\mathbf{G}(z)))\}$$

En resumen,  $\mathbf{D}$  y  $\mathbf{G}$  están jugando un juego *minimax* con la función objetivo:

$$\min_{\mathbf{D}} \max_{\mathbf{G}} \{-E_{x \sim \text{Data}} \log \mathbf{D}(x) - E_{z \sim \text{Noise}} \log(1 - \mathbf{D}(\mathbf{G}(z)))\}$$

# Entrenamiento

Lo más importante en una GAN es su proceso de entrenamiento. Cada batch o step se divide en dos pasos diferentes.

Para cada batch:

1 Optimizar el discriminador.

- Congelamos los pesos del generador.
- Actualizamos los pesos del discriminador para minimizar la BCE.

2 Optimizar el generador.

- Congelamos los pesos del discriminador.
- Actualizamos los pesos del generador buscando engañar al discriminador.

Este tipo de optimización se conoce como *Alternating Stochastic Gradient Descent*.

# Entrenamiento

## Paso 1: Optimizar el discriminador:

- ① Obtenemos las muestras reales del conjunto de datos  $X = \{x_1, \dots, x_n\}$ .
- ② Con  $\mathbf{G}$  congelado, generamos las muestras  $X' = \{x'_1, \dots, x'_n\}$  mediante  $\mathbf{G}(\mathbf{z})$ .
- ③ Optimizamos el discriminador mediante la Binary Cross Entropy:
  - Primero obtenemos la loss para las muestras reales buscando predecir un 1.

$$BCE(y = 1) = -\log(\mathbf{D}(\mathbf{x}))$$

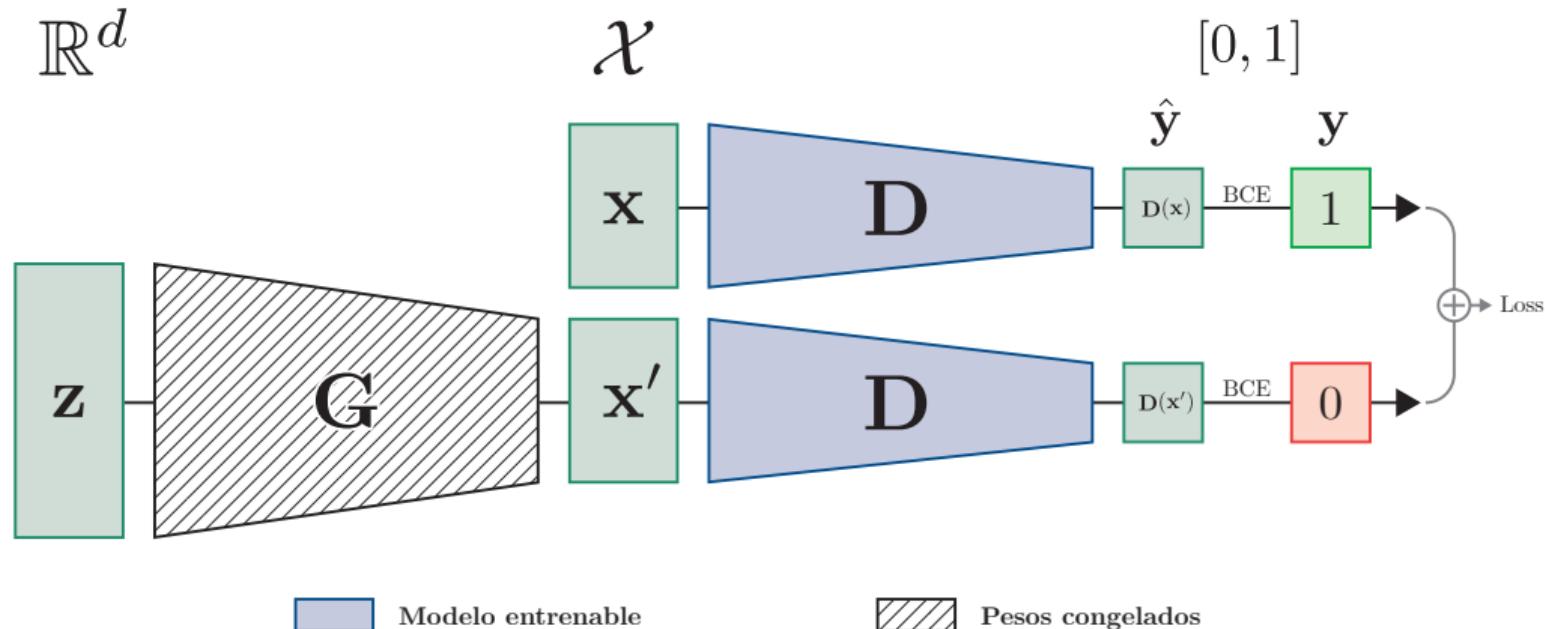
- A continuación, lo mismo para las muestras generadas pero buscando predecir un 0.

$$BCE(y = 0) = -\log(1 - \mathbf{D}(\mathbf{G}(\mathbf{z})))$$

- Optimizamos utilizando  $BCE(y = 1) + BCE(y = 0)$ .

# Entrenamiento

Paso 1: Optimizar el discriminador:



# Entrenamiento

## Paso 2: Optimizar el generador:

- ① Generamos las muestras  $X' = \{x'_1, \dots, x'_n\}$  mediante  $\mathbf{G(z)}$ .
- ② Con  $\mathbf{D}$  congelado, optimizamos el generador mediante la Binary Cross Entropy:
  - Obtenemos la loss para las muestras generadas **buscando predecir un 1**.

$$BCE(y = 1) = -\log(\mathbf{D}(\mathbf{G(z)}))$$

- Optimizamos  $\mathbf{G}$  utilizando la loss calculada.

## Engañar al discriminador

Al dar ejemplos falsos y entrenar el generador para que sea capaz de generar 1, el generador aprende a engañar al discriminador.

# Entrenamiento

Paso 2: Optimizar el generador:



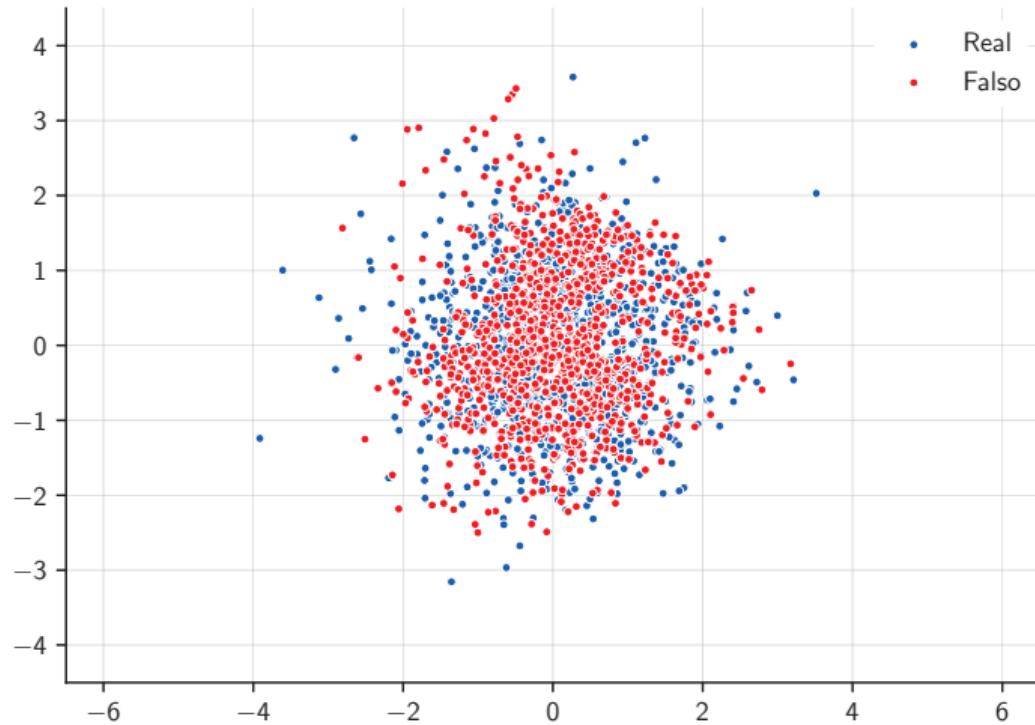
# Ejemplos

$$\mathbf{G} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$



# Ejemplos

$$\mathbf{G} : \mathbb{R}^8 \rightarrow \mathbb{R}^2$$



# Ejemplos

$$\mathbf{G} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$$



# Ejemplos

$$\mathbf{G} : \mathbb{R}^8 \rightarrow \mathbb{R}^2$$



# Ejemplos

## Generación de imágenes

Figura del artículo original de Goodfellow et al., 2014 donde se presentaron las GANs. Todas son generadas salvo las de borde amarillo, que son las de entrenamiento más próximas.



a)



b)



c)



d)

# Ejemplos

Imágenes de habitaciones generadas por una GAN tras la primera época de entrenamiento.



# Ejemplos

Imágenes de habitaciones generadas por una GAN tras la quinta época de entrenamiento.



# Problemas de las GANs

Entrenar una red GAN tradicional puede tener los siguientes problemas:

- **Oscilaciones sin convergencia:** A diferencia de la minimización de loss estándar, el Alternating Stochastic Gradient Descent no tiene garantía de convergencia.
- **Desvanecimiento de gradientes:** Cuando el discriminador es demasiado bueno, puede que el generador no sea capaz de aprender.
- **Colapso de modo:** El generador puede dejar de generar diversidad en las muestras y producir siempre el mismo tipo de ejemplos.
- **Evaluación del rendimiento:** La evaluación del rendimiento de una GAN es difícil en la práctica. ¿Cómo de buena es una imagen generada? ¿Con quién la comparamos?.
- **Control de la generación:** Al generar a partir de ruido  $z$ , ¿Cómo puedo generar ejemplos de una clase concreta?

Generative Adversarial Networks (GANs)

---

## GANs condicionales (cGANs)

# GANs condicionales

## Control de la generación

Los modelos que hemos visto hasta ahora nos permiten generar nuevas muestras a partir de un vector de ruido, pero no permiten controlar la generación.

Muchas aplicaciones requieren de una generación condicionada:

- Predicción del siguiente fotograma en un video.
- Rellenar partes faltantes o dañadas de una imagen.
- Transferencia del estilo artístico de una imagen a otra.

# GANs condicionales

## The Conditional GAN

Este modelo propuesto por Mirza y Osindero en 2014 añade la capacidad de generación condicionada a las GANs tradicionales.

Para ello, parametriza **G** y **D** con un valor condicionante **y**:

$$\min_{\mathbf{D}} \max_{\mathbf{G}} \left\{ -E_{x,y \sim \text{Data}} \log \mathbf{D}(\mathbf{x}, \mathbf{y}) - E_{z \sim \text{Noise}, y \sim \text{Data}} \log (1 - \mathbf{D}(\mathbf{G}(\mathbf{z}, \mathbf{y}), \mathbf{y})) \right\}$$

# GANs condicionales

The Conditional GAN

## Ejemplo MNIST:

Queremos ser capaces de generar imágenes de un dígito concreto:

- La  $y$  sería un vector One-Hot con la clase correspondiente al dígito.
- Tenemos que introducirlo tanto en el generador como en el discriminador.

**En este caso, como necesitamos datos etiquetados estamos ante un problema de aprendizaje supervisado.**

# GANs condicionales

The Conditional GAN

Ejemplo MNIST:



# GANs condicionales

The Conditional GAN

Ejemplo MNIST:



# GANs condicionales

The Conditional GAN

Ejemplo MNIST:



# GANs condicionales

The Conditional GAN

Ejemplo MNIST:



Cada fila está condicionada a una clase y en cada columna a un vector de ruido diferente.

# GANs condicionales

## Image to image

Existen otro tipo de problemas donde queremos generar una imagen a partir de otra dada. Isola et al. presentaron en 2016 un método de resolver estos problemas denominado *pix2pix*.

En concreto, evaluaban su método en las siguientes tareas:

- Bordes a imágenes.
- Segmentación semántica.
- Coloreado de imágenes.
- Dia a noche.

# GANs condicionales

Image to image

En este caso, proponen dos arquitecturas para **G**:

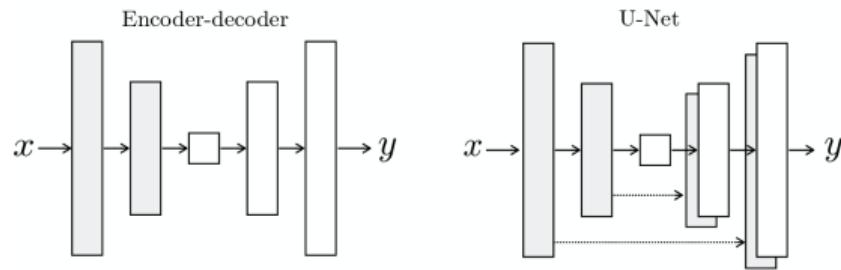


Figure 3: Two choices for the architecture of the generator. The “U-Net” [50] is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

## Nota

**G** solo recibe un elemento (la imagen de origen) y la aleatoriedad proviene de un DropOut.  
En este caso denominan  $y$  a la imagen generada y no  $x$  como en artículos previos.

# GANs condicionales

Image to image

El discriminador **D** es una red convolucional denominada *PatchGAN*.

En lugar de clasificar una imagen completa como real o falsa, clasifica varios parches de tamaño  $N \times N$  de la imagen de forma independiente y luego promedia las predicciones.



Figure 6: Patch size variations. Uncertainty in the output manifests itself differently for different loss functions. Uncertain regions become blurry and desaturated under L1. The 1x1 PixelGAN encourages greater color diversity but has no effect on spatial statistics. The 16x16 PatchGAN creates locally sharp results, but also leads to tiling artifacts beyond the scale it can observe. The 70x70 PatchGAN forces outputs that are sharp, even if incorrect, in both the spatial and spectral (coherence) dimensions. The full 256x256 ImageGAN produces results that are visually similar to the 70x70 PatchGAN, but somewhat lower quality according to our FCN-score metric (Table 2). Please see <https://phillipi.github.io/pix2pix/> for additional examples.

# GANs condicionales

Image to image

## Ejemplos:



input



output

Labels to Street Scene



input



output

Labels to Facade

BW to Color



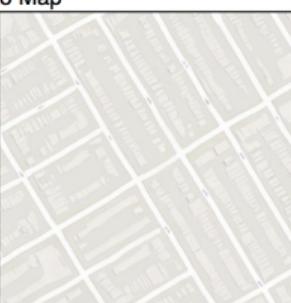
input



output



input



output

Aerial to Map



input



output

Day to Night



input



output

Edges to Photo

# GANs condicionales

## Image to image

El problema principal de esta técnica es que requiere pares de ejemplos con correspondencias pixel a pixel.

### Diferentes dominios

En ciertos casos tenemos ejemplos de diferentes dominios y queremos ser capaces de traducir entre ellos. Tengo una foto de manzanas y quiero cambiarlas por naranjas.

### CycleGAN

En 2017, Zhu et al. presentaron un método capaz de solventar este problema. En el proponen aprender **dos generadores**.

# GANs condicionales

Image to image

## Funcionamiento:



Figure 3: (a) Our model contains two mapping functions  $G : X \rightarrow Y$  and  $F : Y \rightarrow X$ , and associated adversarial discriminators  $D_Y$  and  $D_X$ .  $D_Y$  encourages  $G$  to translate  $X$  into outputs indistinguishable from domain  $Y$ , and vice versa for  $D_X$  and  $F$ . To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss:  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ , and (c) backward cycle-consistency loss:  $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

# GANs condicionales

Image to image

Ejemplos:



apple → orange



orange → apple

# GANs condicionales

Image to image

## Ejemplos:



Generative Adversarial Networks (GANs)

---

## Otras arquitecturas y aplicaciones

# Otras arquitecturas y aplicaciones

## Introducción



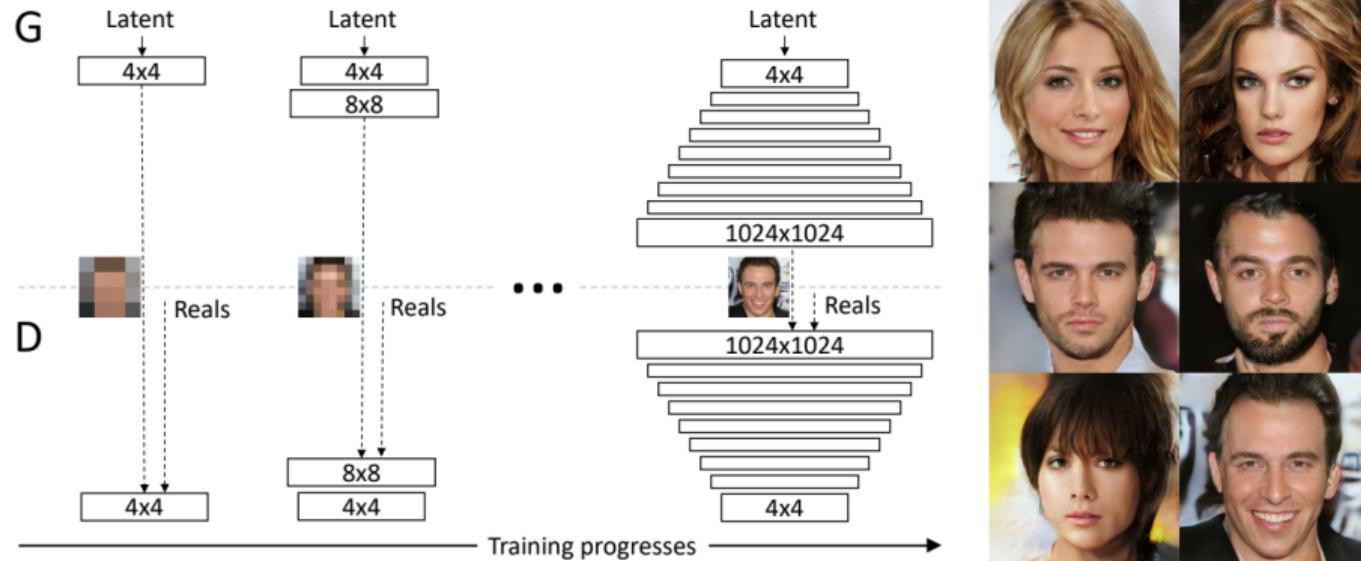
Ian Goodfellow  
@goodfellow\_ian

4.5 years of GAN progress on face generation.  
[arxiv.org/abs/1406.2661](https://arxiv.org/abs/1406.2661)  
[arxiv.org/abs/1511.06434](https://arxiv.org/abs/1511.06434)  
[arxiv.org/abs/1606.07536](https://arxiv.org/abs/1606.07536)  
[arxiv.org/abs/1710.10196](https://arxiv.org/abs/1710.10196)  
[arxiv.org/abs/1812.04948](https://arxiv.org/abs/1812.04948)



# Otras arquitecturas y aplicaciones

## Progressive growing of GANs:



# Otras arquitecturas y aplicaciones

## Progressive growing of GANs:



Figure 2: When doubling the resolution of the generator (G) and discriminator (D) we fade in the new layers smoothly. This example illustrates the transition from  $16 \times 16$  images (a) to  $32 \times 32$  images (c). During the transition (b) we treat the layers that operate on the higher resolution like a residual block, whose weight  $\alpha$  increases linearly from 0 to 1. Here  $[2\times]$  and  $[0.5\times]$  refer to doubling and halving the image resolution using nearest neighbor filtering and average pooling, respectively. The  $[toRGB]$  represents a layer that projects feature vectors to RGB colors and  $[fromRGB]$  does the reverse; both use  $1 \times 1$  convolutions. When training the discriminator, we feed in real images that are downsampled to match the current resolution of the network. During a resolution transition, we interpolate between two resolutions of the real images, similarly to how the generator output combines two resolutions.

# Otras arquitecturas y aplicaciones

## BigGANs:

Buscan generar imágenes de alta resolución y fidelidad mediante:

- Mecanismos de atención.
- Hinge Loss.
- Class-conditional Batch Normalization.
- Etc.



# Otras arquitecturas y aplicaciones

## StyleGAN (v1):

- Generador capaz de separar (automáticamente) diferentes aspectos de una imagen.
- Se basa en la idea de las Progressive Growing GANs:
- Cada aspecto se denomina estilo:
  - Estilos más genéricos en las resoluciones más bajas.
  - Detalles más finos en las resoluciones más altas



# Otras arquitecturas y aplicaciones

## StyleGAN (v1):



$\psi = 1$

$\psi = 0.7$

$\psi = 0.5$

$\psi = 0$

$\psi = -0.5$

$\psi = -1$

# Otras arquitecturas y aplicaciones

## Few-Shot Adversarial Learning of Realistic Neural Talking Head Models:

Origen de los *DeepFakes*:

- Requiere de una imagen de referencia y otra objetivo.
- Crea una versión de la imagen referencia con la pose de la imagen objetivo.
- Ambas imágenes pueden ser de diferentes dominios.



## Generative Adversarial Networks (GANs)

---

## Referencias

# Referencias

- ① **Generative Adversarial Networks, Gilles Louppe**
- ② **Generative Adversarial Networks, François Fleuret**
- ③ **Generative Adversarial Networks, Dive into DL**
- ④ **Conditional Generative Adversarial Nets**
- ⑤ **Image-to-Image Translation with Conditional Adversarial Networks**
- ⑥ **Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks**