GANs II

Clase 13

Dra. Wendy Aguilar

Modelos Generativos Profundos

UN ENFOQUE DESDE LA CREATIVIDAD COMPUTACIONAL

GANs

- La GAN original (Goodfellow et al., 2014) introdujo la idea revolucionaria del juego minimax adversarial entre Generador y Discriminador.

- Sin embargo, su implementación práctica era inestable:
  - Gradientes explosivos o que se desvanecen.
  - Colapso de modo (mode collapse).

    El generador produce siempre imágenes muy similares, perdiendo variedad.

    El modelo se estanca generando un único patrón que engaña al discriminador.

- Entrenamiento altamente inestable → difícil mantener el equilibrio G–D.
- Aun así, demostraron que la competencia podía generar muestras plausibles.

MNIST

TFD (Toronto Face Database)

CIFAR-10

Deep Convolutional GAN (DCGAN)

2015

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Objetivos

1. Demostrar que los GANs pueden escalar a arquitecturas convolucionales profundas

- Antes de este trabajo, los GANs usaban redes totalmente conectadas y eran inestables al entrenarse.

- El objetivo fue **incorporar convoluciones y des-convoluciones (transposed convolutions)** para procesar imágenes de mayor resolución (por ejemplo, de 64×64 píxeles) sin colapsar el entrenamiento.

- Se propusieron **principios de arquitectura específicos** que estabilizan el entrenamiento y mejoran la calidad visual.

Deep Convolutional GAN (DCGAN)

2015

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Objetivos

2. Aprender representaciones jerárquicas sin supervisión

- DCGAN se planteó como una forma de **aprendizaje no supervisado**.

  entrenar un modelo generativo sin etiquetas que pueda aprender **características semánticamente significativas**.

- El **discriminador** actúa como un extractor de características visuales útiles, las cuales pueden reutilizarse en tareas supervisadas como clasificación de imágenes o detección de objetos.

- Es decir, el modelo no solo genera imágenes, sino que **aprende una base representacional robusta**.

Deep Convolutional GAN (DCGAN)

2015

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Objetivos

3. Establecer buenas prácticas de arquitectura para GANs

- Se sistematizaron varias **"reglas empíricas"** que luego se volvieron estándar:
    - Sustituir pooling por convoluciones con stride y deconvoluciones (transposed convolutions).
    - Usar Batch Normalization en casi todas las capas.
    - Eliminar fully connected layers profundas.
    - Activaciones ReLU o LeakyReLU en el generador y discriminador.

Deep Convolutional GAN (DCGAN)

2015

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

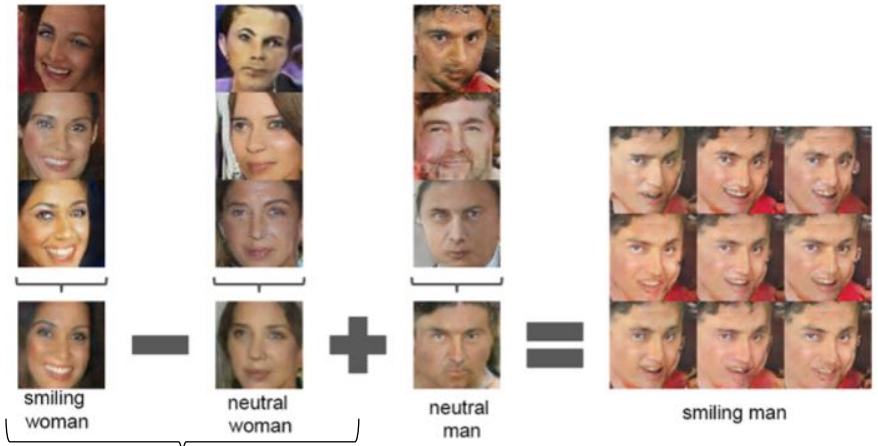
ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Objetivos

4. Visualizar y analizar el espacio latente aprendido

- Mostrar que las operaciones en el espacio latente tienen **significado semántico lineal**:



- **dirección en el espacio latente** que corresponde a "añadir una sonrisa"
- Este vector apunta desde una imagen "neutra" hacia una imagen "sonriente".

- Esta demostración fue fundamental para argumentar que las **representaciones aprendidas son *disentangled* y manipulables**.

cada dimensión codifica un factor de variación independiente y semánticamente interpretable de los datos

Deep Convolutional GAN (DCGAN)

2015

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

Objetivos

5. Evidenciar el potencial de las GANs como extractores de características transferibles.

- Para probarlo, los autores **congelaron el discriminador** y lo usaron como **extractor de características (feature extractor)** en tareas de clasificación en datasets como CIFAR-10, mostrando resultados competitivos frente a métodos no supervisados de la época.

Construyendo una Deep Convolutional GAN (DCGAN)

dcgan.ipynb Google colab

Conjunto de datos Bricks

40,000 imágenes de 50 bloques de juguete, desde distintas perspectivas.

```python
# Crea el pipeline de datos base para su entrenamiento en TensorFlow,
# de forma eficiente y con paralelismo.
train_data = tf.keras.utils.image_dataset_from_directory(
    DATASET_PATH,            # Ruta de las imágenes
    labels=None,
    color_mode="grayscale",
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,   # Agrupa las imágenes para entrenamiento eficiente
    shuffle=True,            # Garantiza que las imágenes se mezclen cada época
    seed=42,                 # de manera reproducible(mismo orden/misma semilla)
    interpolation="bilinear",
)
```

El resultado `train_data` **no son las imágenes en memoria**, sino un *dataset pipeline* que se puede:

- mapear (`.map(preprocess)`)
- almacenar en caché (`.cache()`)
- y conectar directamente al entrenamiento (`model.fit(train_data)`)

Esto permite entrenamiento en streaming, sin cargar todo el dataset en RAM.

```python
#Define y aplica una tubería de preprocesamiento (pipeline)
#sobre el dataset que creó con image_dataset_from_directory.
def preprocess(img):
    """
    Normaliza las imágenes al rango [-1, 1]
    y asegura el tipo float32.
    """
    img = tf.cast(img, tf.float32)
    img = (img - 127.5) / 127.5
    return img

train = (
    train_data
    # Preprocesa cada imagen del dataset, de forma paralela (varios hilos).
    .map(preprocess, num_parallel_calls=tf.data.AUTOTUNE)
    # Almacena en memoria las imágenes ya procesadas para evitar releer y
    # reprocesar el dataset en cada época
    .cache()
    .shuffle(1000) # Desordena las imágenes cada época en buffers de 1000.
    # Permite que la carga del siguiente batch se haga en paralelo al
    # entrenamiento del actual.
    .prefetch(tf.data.AUTOTUNE)
)
```

- El generador usa activación **`tanh`**, que también produce valores en [-1, 1].

Construyendo Deep Convolutional GAN (DCGAN)

Discriminador

- Su objetivo es predecir si una imagen es real o falsa.
-]Actúa como un **clasificador supervisado binario.**
- Podemos usar una arquitectura de capas convolucionales, con un solo nodo de salida.

Entrada → una imagen (real o generada).

Salida → un escalar entre 0 y 1 que representa la *probabilidad de ser real*.

Construyendo Deep Convolutional GAN (DCGAN)

Discriminador

Layer (type)	Output shape	Param #
InputLayer	(None, 64, 64, 1)	0
Conv2D	(None, 32, 32, 64)	1,024
LeakyReLU	(None, 32, 32, 64)	0
Dropout	(None, 32, 32, 64)	0
Conv2D	(None, 16, 16, 128)	131,072
BatchNormalization	(None, 16, 16, 128)	512
LeakyReLU	(None, 16, 16, 128)	0
Dropout	(None, 16, 16, 128)	0
Conv2D	(None, 8, 8, 256)	524,288
BatchNormalization	(None, 8, 8, 256)	1,024
LeakyReLU	(None, 8, 8, 256)	0
Dropout	(None, 8, 8, 256)	0
Conv2D	(None, 4, 4, 512)	2,097,152
BatchNormalization	(None, 4, 4, 512)	2,048
LeakyReLU	(None, 4, 4, 512)	0
Dropout	(None, 4, 4, 512)	0
Conv2D	(None, 1, 1, 1)	8,192
Flatten	(None, 1)	0

Total params	2,765,312
Trainable params	2,763,520
Non-trainable params	1,792

```python
discriminator_input = layers.Input(shape=(64, 64, 1)) ❶
x = layers.Conv2D(64, kernel_size=4, strides=2, padding="same", use_bias = False)(
    discriminator_input
) ❷
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(
    128, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum = 0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(
    256, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum = 0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(
    512, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum = 0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Dropout(0.3)(x)
x = layers.Conv2D(
    1,
    kernel_size=4,
    strides=1,
    padding="valid",
    use_bias = False,
    activation = 'sigmoid'
)(x)
discriminator_output = layers.Flatten()(x) ❸
discriminator = models.Model(discriminator_input, discriminator_output) ❹
```

64x64x1

32x32x64

16x16x128

8x8x256

4x4x512

Capa final de decisión

1x1x1

Aplica un solo filtro que aprende a **combinar toda la información global** (texturas, formas, patrones) en un solo valor por mapa espacial.
Un kernel de 4x4 que cubre **toda el área espacial.**

Un valor escalar que representa el *"grado de realismo"* de la imagen completa.

Rango entre 0 (falso) y 1 (real)

Construyendo Deep Convolutional GAN (DCGAN)

Discriminador

Layer (type)	Output shape	Param #
InputLayer	(None, 64, 64, 1)	0
Conv2D	(None, 32, 32, 64)	1,024
LeakyReLU	(None, 32, 32, 64)	0
Dropout	(None, 32, 32, 64)	0
Conv2D	(None, 16, 16, 128)	131,072
BatchNormalization	(None, 16, 16, 128)	512
LeakyReLU	(None, 16, 16, 128)	0
Dropout	(None, 16, 16, 128)	0
Conv2D	(None, 8, 8, 256)	524,288
BatchNormalization	(None, 8, 8, 256)	1,024
LeakyReLU	(None, 8, 8, 256)	0
Dropout	(None, 8, 8, 256)	0
Conv2D	(None, 4, 4, 512)	2,097,152
BatchNormalization	(None, 4, 4, 512)	2,048
LeakyReLU	(None, 4, 4, 512)	0
Dropout	(None, 4, 4, 512)	0
Conv2D	(None, 1, 1, 1)	8,192
Flatten	(None, 1)	0

Total params	2,765,312
Trainable params	2,763,520
Non-trainable params	1,792

```
discriminator_input = layers.Input(shape=(64, 64, 1)) ❶
x = layers.Conv2D(64, kernel_size=4, strides=2, padding="same", use_bias = False)(
```

64x64x1

32x32x64

x128

56

12

¿Se les hace conocida esta arquitectura?

Capa final de decisión

```
x = layers.Conv2D(
    1,
    kernel_size=4,
    strides=1,
    padding="valid",
    use_bias = False,
    activation = 'sigmoid'
)(x)
discriminator_output = layers.Flatten()(x) ❸
discriminator = models.Model(discriminator_input, discriminator_output) ❹
```

1x1x1

Un valor escalar que representa el *"grado de realismo"* de la imagen completa.

Aplica un solo filtro que aprende a **combinar toda la información global** (texturas, formas, patrones) en un solo valor por mapa espacial.
Un kernel de 4x4 que cubre **toda el área espacial.**

Rango entre 0 (falso) y 1 (real)

Construyendo Deep Convolutional GAN (DCGAN)

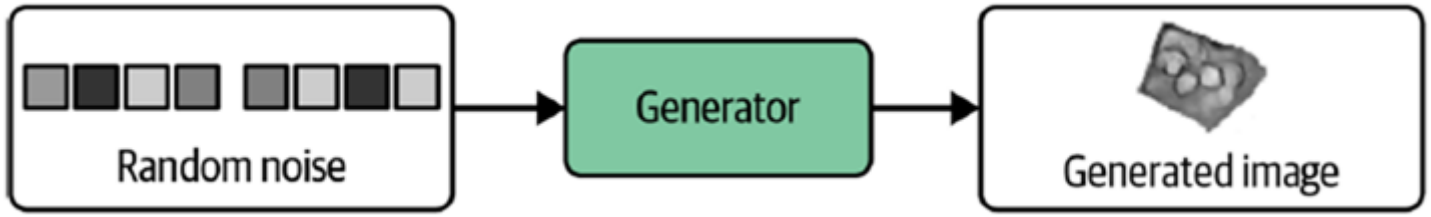
Etapa	Autoencoder (encoder)	Discriminador
Entrada	Imagen	Imagen
Convoluciones	Extraen características jerárquicas	Extraen características jerárquicas
Reducción espacial	Por *strides* o *pooling*	Por *strides*
Representación latente	Vector z	Vector/logit para clasificación real/falso

Es, esencialmente, un encoder + una capa de decisión.

- Ambos aprenden una representación compacta de la imagen.
- La diferencia está en su función:
  - el encoder reconstruye,
  - el discriminador juzga.

Construyendo Deep Convolutional GAN (DCGAN)

Generador



La entrada es un vector tomado de una distribución normal estándar multivariada.

La salida es una imagen del mismo tamaño que la imagen en el conjunto de entrenamiento original.

- El generador de una GAN **cumple exactamente el mismo propósito que el decodificador de un VAE**: convertir un vector en el espacio latente en una imagen.
- El concepto de mapear de un espacio latente de vuelta al dominio original es muy común en los modelos generativos,
- Esto nos permite manipular vectores en el espacio latente para cambiar características de alto nivel en las imágenes.

El generador "aprende" a mapear regiones del espacio latente a patrones visuales —formas, texturas, combinaciones estructurales— que resultan plausibles para el discriminador.

Construyendo Deep Convolutional GAN (DCGAN)

Generador

Layer (type)	Output shape	Param #
InputLayer	(None, 100)	0
Reshape	(None, 1, 1, 100)	0
Conv2DTranspose	(None, 4, 4, 512)	819,200
BatchNormalization	(None, 4, 4, 512)	2,048
ReLU	(None, 4, 4, 512)	0
Conv2DTranspose	(None, 8, 8, 256)	2,097,152
BatchNormalization	(None, 8, 8, 256)	1,024
ReLU	(None, 8, 8, 256)	0
Conv2DTranspose	(None, 16, 16, 128)	524,288
BatchNormalization	(None, 16, 16, 128)	512
ReLU	(None, 16, 16, 128)	0
Conv2DTranspose	(None, 32, 32, 64)	131,072
BatchNormalization	(None, 32, 32, 64)	256
ReLU	(None, 32, 32, 64)	0
Conv2DTranspose	(None, 64, 64, 1)	1,024

Total params	3,576,576
Trainable params	3,574,656
Non-trainable params	1,920

```
generator_input = layers.Input(shape=(100,)) ❶
x = layers.Reshape((1, 1, 100))(generator_input) ❷
x = layers.Conv2DTranspose(
    512, kernel_size=4, strides=1, padding="valid", use_bias = False
)(x) ❸
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(
    256, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(
    128, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
x = layers.Conv2DTranspose(
    64, kernel_size=4, strides=2, padding="same", use_bias = False
)(x)
x = layers.BatchNormalization(momentum=0.9)(x)
x = layers.LeakyReLU(0.2)(x)
generator_output = layers.Conv2DTranspose(
    1,
    kernel_size=4,
    strides=2,
    padding="same",
    use_bias = False,
    activation = 'tanh'
)(x) ❹
generator = models.Model(generator_input, generator_output) ❺
```

Convierte el vector latente z de forma `(Z_DIM,)` en un tensor de 4 dimensiones `(1, 1, Z_DIM)`

4x4*512

8x8*256

16x16*128

32x32*64

64x64*1

Limita los valores en el rango [-1, 1], coincidiendo con el preprocesamiento del dataset.
Así, el generador y el discriminador trabajan en el mismo rango de valores

Construyendo Deep Convolutional GAN (DCGAN)

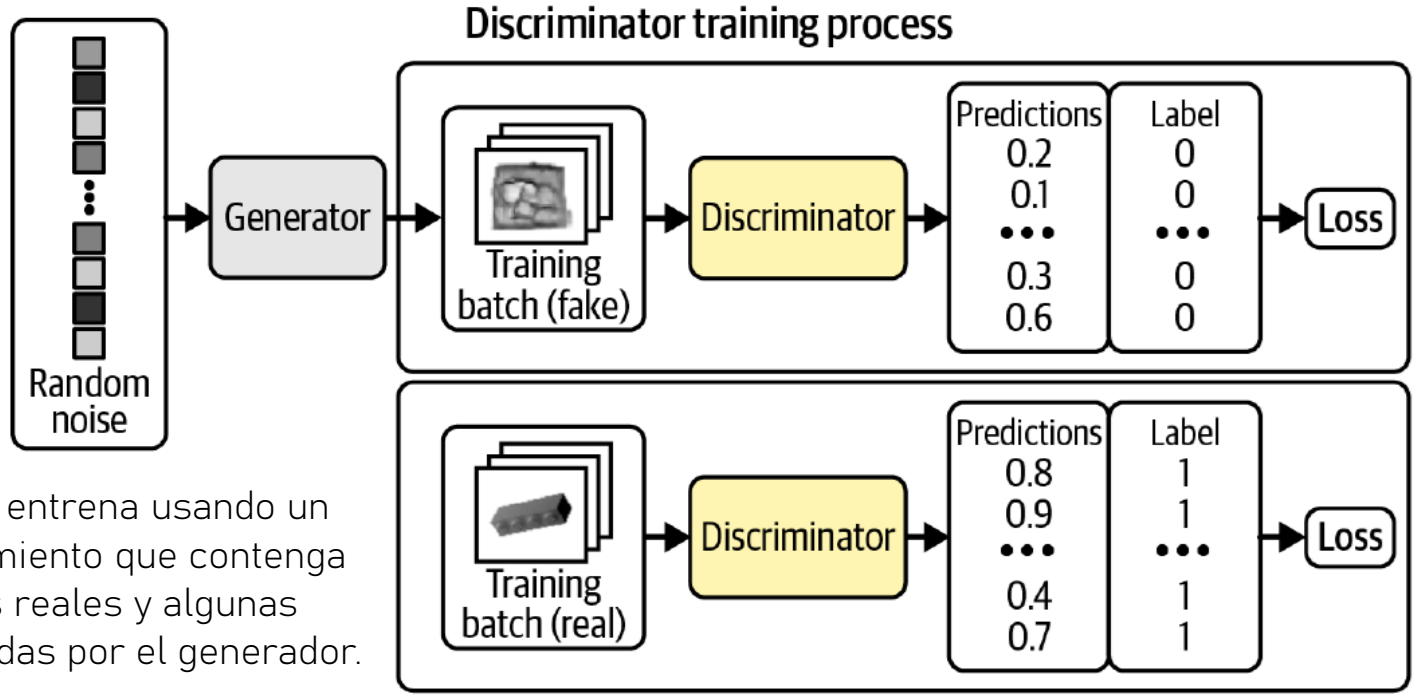
Entrenamiento

- El generador y el discriminador son muy simples y no muy diferentes de los modelos VAE.

- El punto clave de las GANs radica en comprender el proceso de entrenamiento para el generador y el discriminador.

Construyendo Deep Convolutional GAN (DCGAN)

Entrenamiento



El discriminador se entrena usando un conjunto de entrenamiento que contenga algunas imágenes reales y algunas imágenes falsas creadas por el generador.

Se vuelve un problema de aprendizaje supervisado, en donde 1 corresponde a imagen real y 0 a imagen falsa.

Se usa **binary cross-entropy** como función de pérdida.

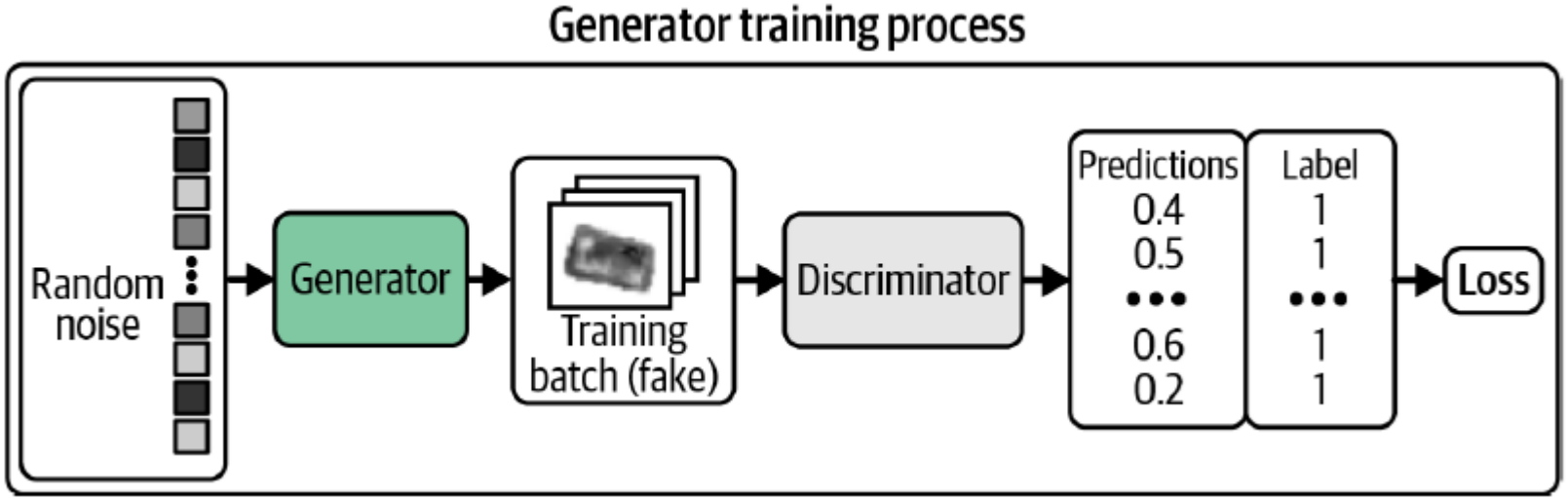
Construyendo Deep Convolutional GAN (DCGAN)

Entrenamiento

Tenemos que encontrar una manera de calificar cada imagen generada de tal forma que podamos optimizar hacia las imágenes con mayor puntaje.

Afortunadamente contamos con un discriminador que hace precisamente eso.

La función de pérdida para el generador es simplemente la binary cross-entropy entre estas probabilidades y un vector de unos.



Esto tiene sentido porque queremos entrenar un generador que produzca imágenes que el discriminador piense que son reales.

Podemos generar un batch de imágenes y pasárselas al discriminador para que le asigne una calificación a cada imagen.

Construyendo Deep Convolutional GAN (DCGAN)

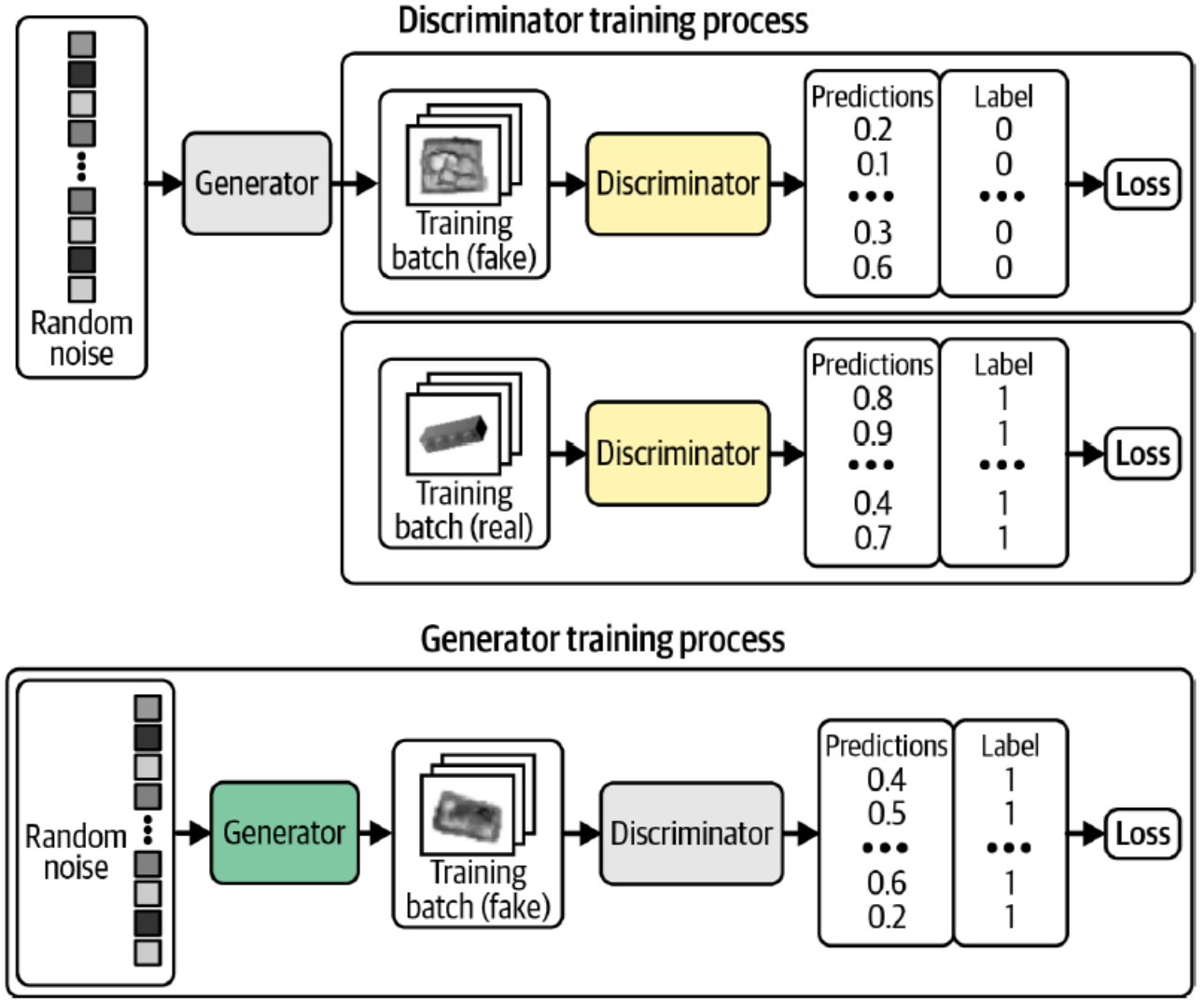
Debemos de alternar el entrenamiento de estas dos redes, asegurándonos de que solo actualicemos los pesos de una de las redes a la vez.

Durante el proceso del entrenamiento del generador, solo se actualizan los pesos del generador.

Si permitiéramos que los pesos del discriminador también se actualizaran, entonces éste se ajustaría para predecir que las imágenes generadas son reales, que no es lo que deseamos.

Queremos que a las imágenes generadas se les asigne un valor cercano a 1 (real) porque el generador es bueno, no porque el discriminador es malo.

Entrenamiento



Construyendo Deep Convolutional GAN (DCGAN)

Encapsula **toda la lógica de entrenamiento conjunto** del **generador** y el
discriminador dentro de una única subclase de `tf.keras.models.Model`.

```
class DCGAN(models.Model):
    def __init__(self, discriminator, generator, latent_dim):
        super(DCGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.latent_dim = latent_dim

    def compile(self, d_optimizer, g_optimizer):
        super(DCGAN, self).compile()
        self.loss_fn = losses.BinaryCrossentropy() ❶
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.d_loss_metric = metrics.Mean(name="d_loss")
        self.g_loss_metric = metrics.Mean(name="g_loss")

    @property
    def metrics(self):
        return [self.d_loss_metric, self.g_loss_metric]
```

Sirven solo para **monitorear el progreso**, sin afectar el entrenamiento.

La misma función de pérdida para el generador y para el discriminador, pero su **rol y su interpretación son diferentes:**

Red	Qué intenta lograr	Etiqueta que usa	Fórmula	Intuición
Discriminador	Distinguir entre reales y falsas	Reales→1, Falsas→0	BCE(1, D(x_real)) + BCE(0, D(G(z)))	"Que aprenda a reconocer falsificaciones"
Generador	Engañar al discriminador	Falsas→1 (se hacen pasar por reales)	BCE(1, D(G(z)))	"Que el D crea que sus imágenes son reales"

```python
def train_step(self, real_images):
    batch_size = tf.shape(real_images)[0]
    random_latent_vectors = tf.random.normal(
        shape=(batch_size, self.latent_dim)
    ) ❷

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = self.generator(
            random_latent_vectors, training = True
        ) ❸
        real_predictions = self.discriminator(real_images, training = True) ❹
        fake_predictions = self.discriminator(
            generated_images, training = True
        ) ❺

        real_labels = tf.ones_like(real_predictions)
        real_noisy_labels = real_labels + 0.1 * tf.random.uniform(
            tf.shape(real_predictions)
        )
        fake_labels = tf.zeros_like(fake_predictions)
        fake_noisy_labels = fake_labels - 0.1 * tf.random.uniform(
            tf.shape(fake_predictions)
        )

        d_real_loss = self.loss_fn(real_noisy_labels, real_predictions)
        d_fake_loss = self.loss_fn(fake_noisy_labels, fake_predictions)
        d_loss = (d_real_loss + d_fake_loss) / 2.0 ❻

        g_loss = self.loss_fn(real_labels, fake_predictions) ❼

    gradients_of_discriminator = disc_tape.gradient(
        d_loss, self.discriminator.trainable_variables
    )
    gradients_of_generator = gen_tape.gradient(
        g_loss, self.generator.trainable_variables
    )

    self.d_optimizer.apply_gradients(
        zip(gradients_of_discriminator, discriminator.trainable_variables)
    ) ❽
    self.g_optimizer.apply_gradients(
        zip(gradients_of_generator, generator.trainable_variables)
    )

    self.d_loss_metric.update_state(d_loss)
    self.g_loss_metric.update_state(g_loss)

    return {m.name: m.result() for m in self.metrics}
```

Forward pass

Dos “grabadoras” de gradientes independientes:

Para entrenar la red, primero muestreamos un batch de vectores de una distribución normal estándar multivariable.

Los pasamos a través del generador para producir un batch de imágenes generadas.

Le pedimos al discriminador que prediga qué tan reales se ven las imágenes reales.

Lo mismo para las generadas.

Cálculo de la pérdida

Label smoothing y label noise (etiquetado ruidoso).

Agregan ruido controlado a las etiquetas verdaderas y falsas para que el discriminador no se vuelva demasiado preciso ni rígido, manteniendo el entrenamiento de la GAN más estable y fluido, especialmente en las primeras épocas.

La pérdida del discriminador es el promedio de la binary cross-entropy tanto de las imágenes reales (con etiqueta 1) como as generadas (con etiqueta 0).

La pérdida del generador es la binary cross-entropy entre las predicciones de las imágenes generadas y una etiqueta de 1.

Backward pass

Actualización de pesos

Actualiza los pesos del discriminador y del generador por separado.

```
dcgan = DCGAN(
    discriminator=discriminator, generator=generator, latent_dim=100
)

dcgan.compile(
    d_optimizer=optimizers.Adam(
        learning_rate=0.0002, beta_1 = 0.5, beta_2 = 0.999
    ),
    g_optimizer=optimizers.Adam(
        learning_rate=0.0002, beta_1 = 0.5, beta_2 = 0.999
    ),
)

dcgan.fit(train, epochs=300)
```

GANs

Entrenamiento en forma de **juego minimax**:

El entrenamiento de una GAN se plantea como un **juego de suma cero** entre dos redes neuronales:

situación en la que **lo que uno gana, el otro lo pierde** en la misma proporción.

$$\text{Ganancia de G} + \text{Ganancia de D} = 0$$

Si el Discriminador (D) mejora su capacidad para distinguir falsos → el Generador (G) "pierde".

Si el Generador (G) logra engañar a D → entonces D "pierde".

El Generador G **busca minimizar** esa misma función, es decir, hacer que D se equivoque.

El Discriminador D **busca maximizar** la probabilidad de clasificar correctamente:
·reales como reales,
·falsas como falsas.

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

GANs

Entrenamiento en forma de **juego minimax**:

El equilibrio teórico de Nash

Goodfellow demostró que el equilibrio ideal ocurre cuando:

$$p_g(x) = p_{data}(x)$$

El generador reproduce perfectamente la distribución real de datos.

En ese punto:

$$D(x) = 0.5$$

La probabilidad de que una imagen sea real.

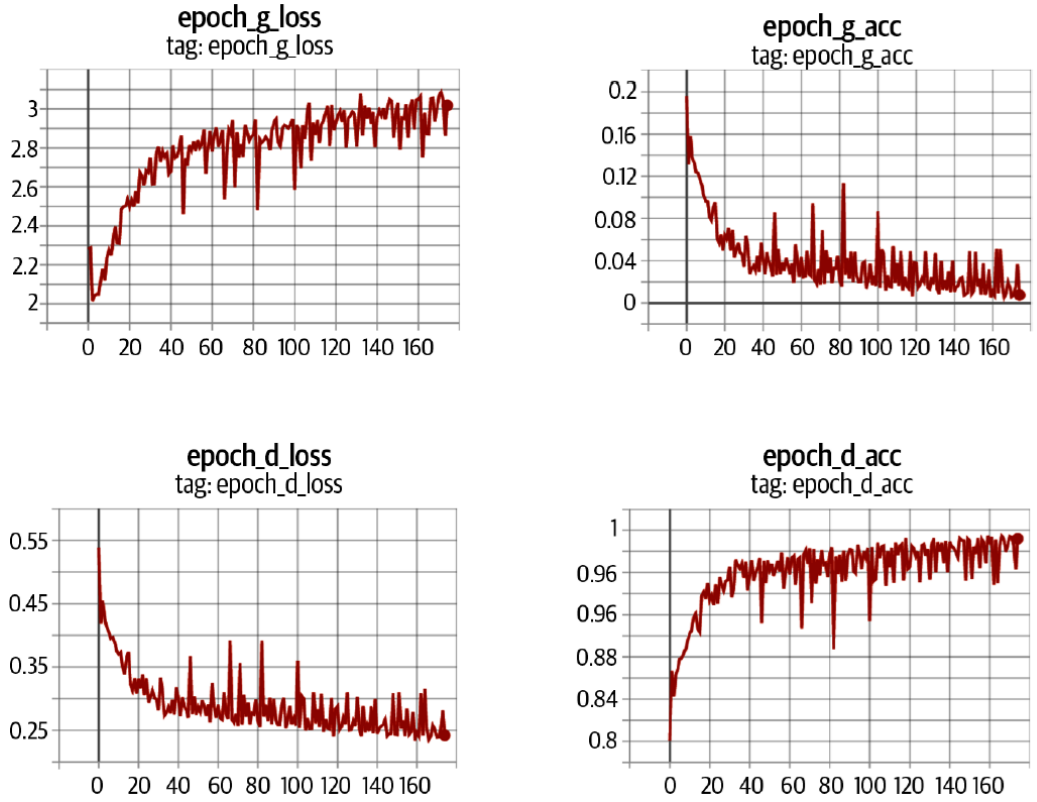
Esto significa que el discriminador ya no puede distinguir si una imagen viene del dataset real o del generador.

Por tanto:

- D predice 0.5 para todo → máximo nivel de incertidumbre.
- G ha aprendido la distribución verdadera → éxito.

Deep Convolutional GAN (DCGAN)

- Pero en la práctica de entrenamiento:
  - El **discriminador aprende más rápido** (es una tarea supervisada más sencilla).
  - El **generador tarda más en mejorar** (recibe gradientes indirectos y a menudo ruidosos).
- Al estar en constante pelea el generador y el discriminador tiende a hacer que el proceso del entrenamiento se vuelva inestable.
  - Idealmente, el proceso de entrenamiento encontrará un equilibrio que le permita aprender al generador información significativa del discriminador de tal forma que se vaya incrementando la calidad de las imágenes generadas.
  - Después de suficientes épocas, el discriminador tiende a terminar dominando la competencia, pero esto podría no ser un problema su para este punto el generador ya aprendió a producir imágenes de suficiente calidad.



Pérdida y exactitud del discriminador y del generador durante el entrenamiento.

Deep Convolutional GAN (DCGAN)

Situación	Interpretación
D_acc ≈ 0.5 y G_loss moderado	✓ Buen equilibrio: el generador logra engañar al discriminador a veces, pero no siempre.
D_acc ≈ 1.0 y G_loss muy alto	✗ Discriminador dominante → el generador no recibe gradientes útiles.
D_acc ≈ 0.0 y G_loss muy bajo	✗ Generador dominante → el discriminador está confundido o colapsado.
D_loss y G_loss oscilan suavemente sin divergir	✓ Competencia saludable: la dinámica de aprendizaje sigue viva.

Deep Convolutional GAN (DCGAN)

Agregar ruido a las etiquetas (label smoothing)

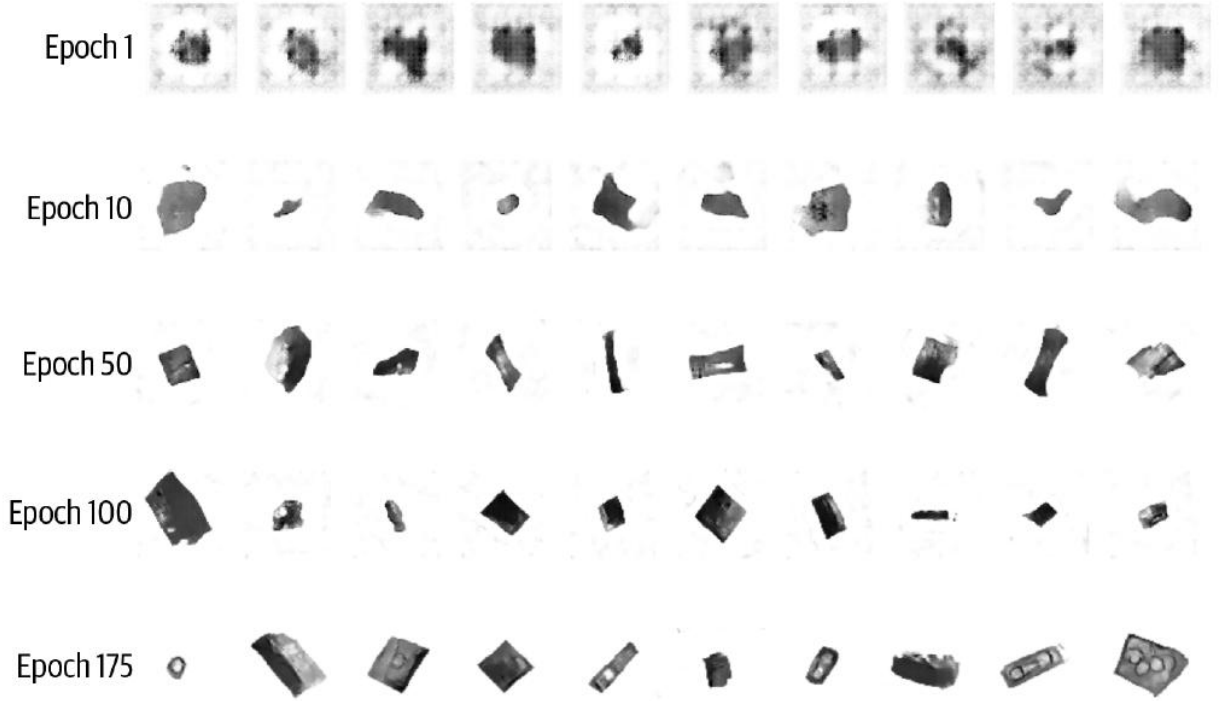
- Un **truco útil** cuando entrenamos GANs consiste en agregar una pequeña cantidad de ruido aleatorio a las etiquetas de entrenamiento.

- Esto ayuda a mejorar la estabilidad del proceso de entrenamiento y también a afinar las imágenes generadas.

- Este suavizado de etiquetas actúa como una forma de moderar al discriminador, de modo que se le presenta una tarea más desafiante y no sobrepasa al generador.

Deep Convolutional GAN (DCGAN)

Análisis de la DCGAN



- Es un tanto milagroso que una red neuronal sea capaz de convertir ruido aleatorio en algo significativo.
- Recuerda que la hemos provisto de ninguna información adicional aparte de los pixeles, de tal forma que ésta tiene que ver por sí misma cómo puede trabajar con conceptos de alto nivel como sombras, cubos y círculos.

Deep Convolutional GAN (DCGAN)

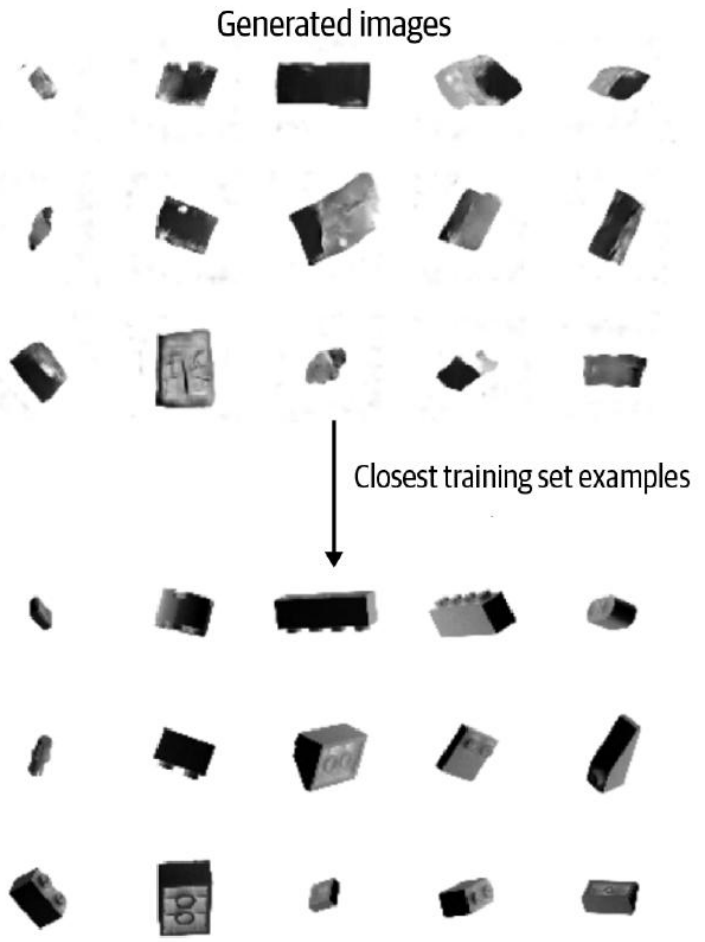
Análisis de la DCGAN

Otro requisito de un modelo generativo es que no solo pueda reproducir imágenes del conjunto de entrenamiento.

Usamos la distancia L1:

Mean Absolute Error (MAE)
Pixel a pixel

```
def compare_images(img1, img2):
    return np.mean(np.abs(img1 - img2))
```



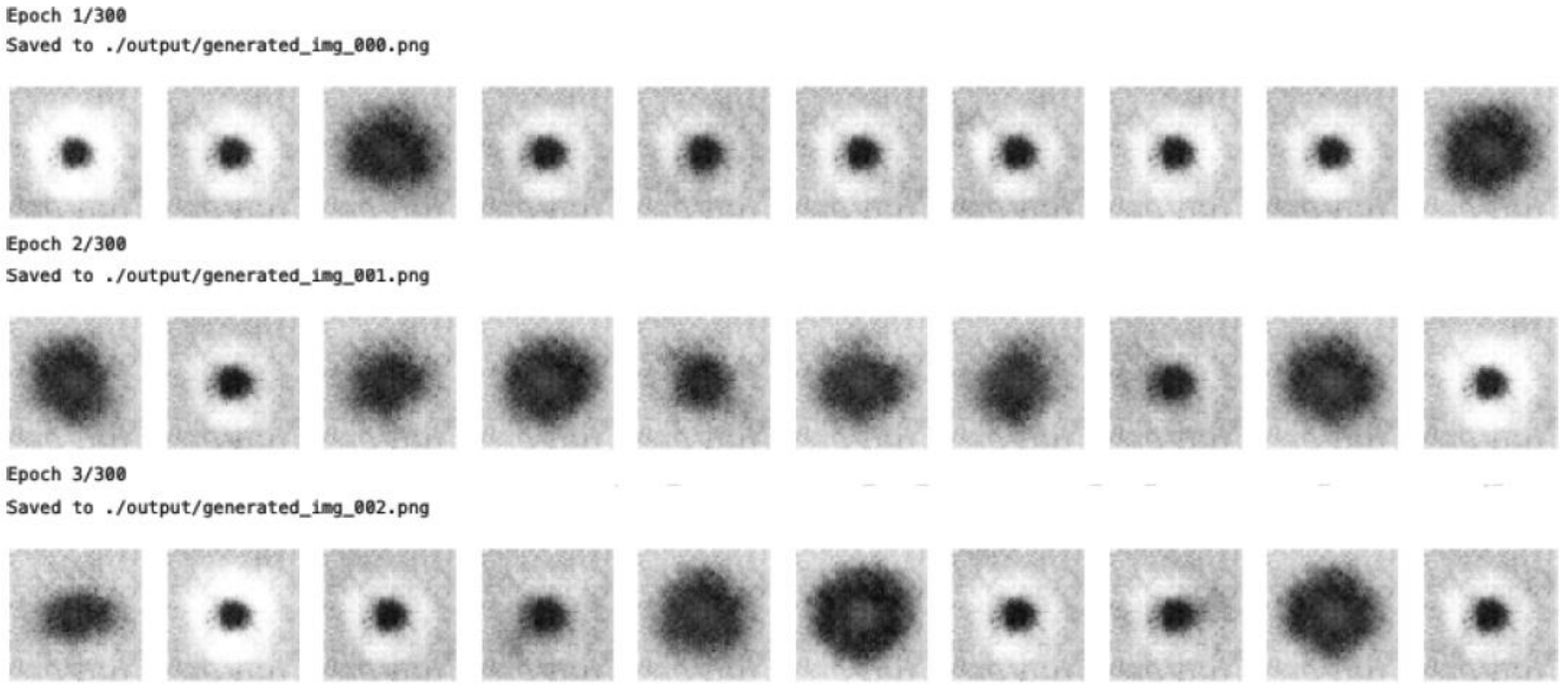
Deep Convolutional GAN (DCGAN)

Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

El discriminador sobrepasa al generador

La señal de la función de pérdida será demasiado débil como para guiar a mejoras al generador.

En el peor escenario, el discriminador aprenderá a separar perfectamente las imágenes reales de las falsas y los gradientes se desvanecerán completamente, llevando a que no mejore el generador.

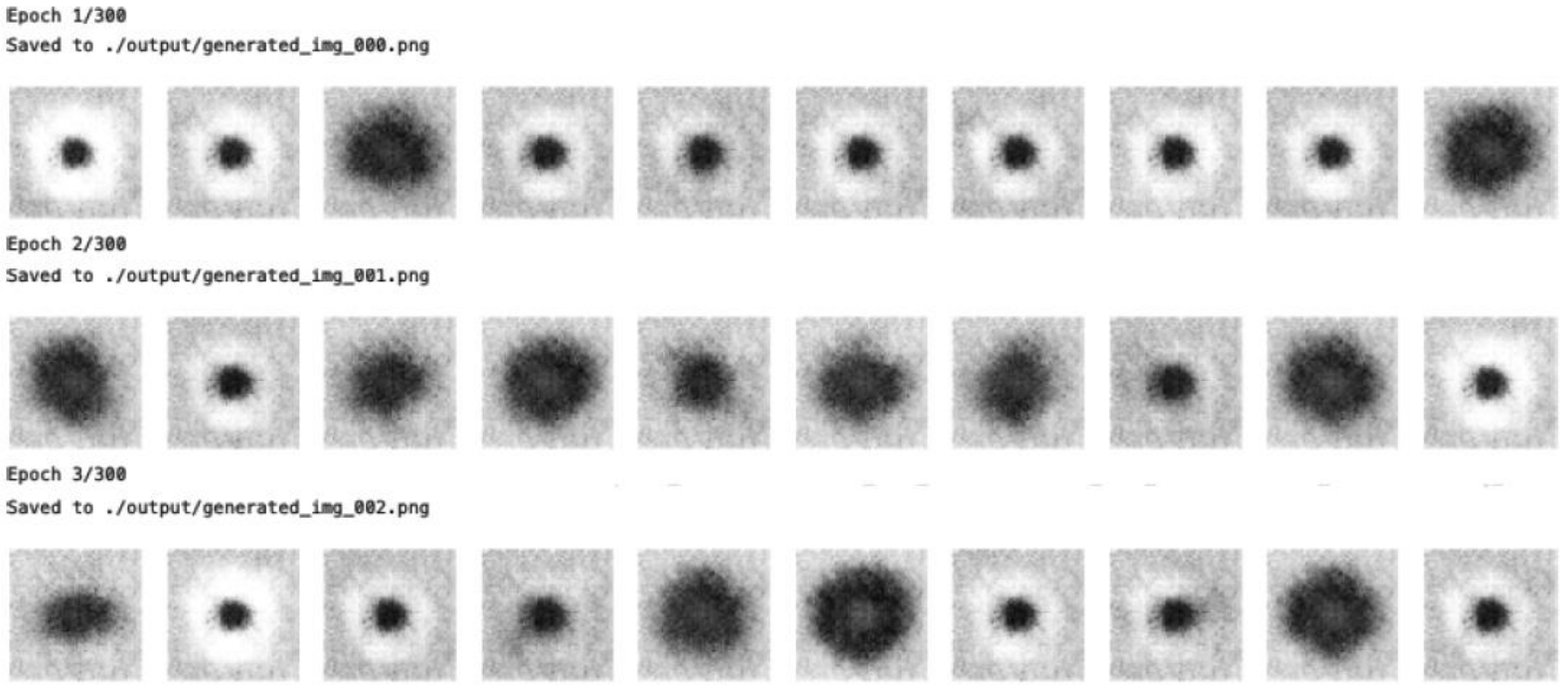


Deep Convolutional GAN (DCGAN)

Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

Solución: debilitar al discriminador

- Aumenta el parámetro de tasa de las capas Dropout en el discriminador para reducir la cantidad de información que fluye a través de la red.
- Reduce la tasa de aprendizaje del discriminador.
- Reduce el número de filtros convolucionales en el discriminador.
- Agrega ruido a las etiquetas cuando se entrene el discriminador.
- Invierte aleatoriamente las etiquetas de algunas imágenes al entrenar el discriminador.



Deep Convolutional GAN (DCGAN)

Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

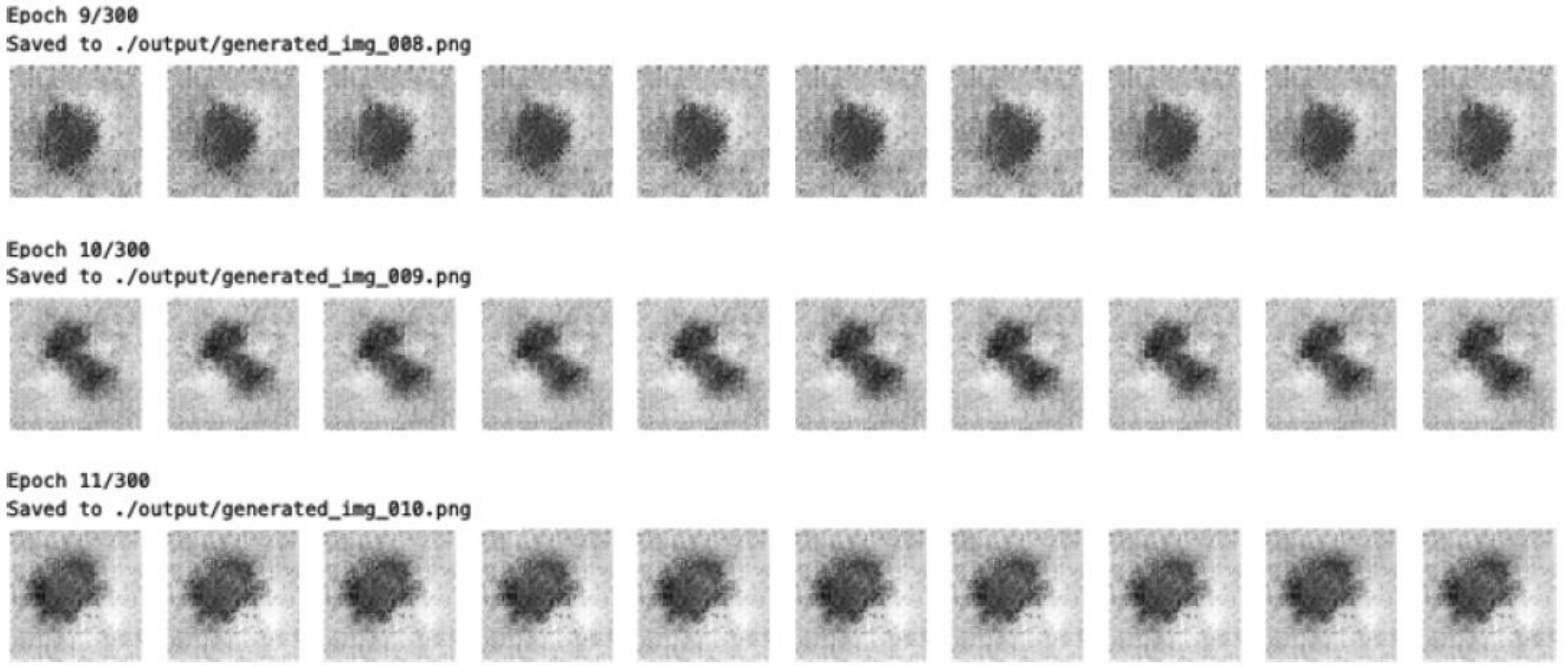
El generador sobrepasa al discriminador

El generador encontrará maneras sencillas de engañar al discriminador con una pequeña muestra de imágenes casi idénticas (*mode collapse*).

Es como si no actualizáramos los pesos del discriminador por varios batches.

El generador va a tender a encontrar una sóla observación (mode) que siempre engaña al discriminador y comenzará a mapear cada punto en el espacio latente a esta imagen.

Los gradientes de la función de pérdida colapsarán a casi 0, y no podrá recuperarse de ese estado.



Solución: mejorar al discriminador

- Haciendo lo opuesto de la diapositiva anterior.
- También puedes intentar reducir la tasa de aprendizaje de ambas redes e incrementar el tamaño del batch.

Deep Convolutional GAN (DCGAN)

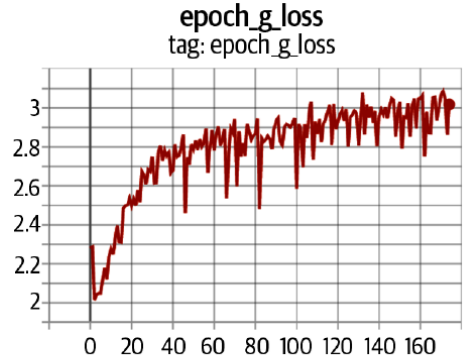
Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

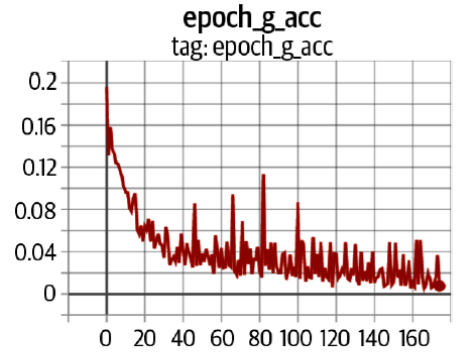
Pérdida no informativa

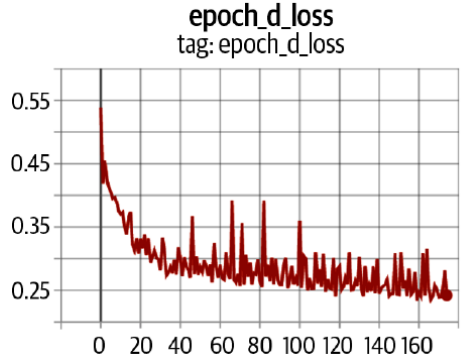
Dado que el modelo de aprendizaje profundo está compilado para minimizar la función de pérdida, sería natural pensar que, cuanto más pequeña sea la función de pérdida del generador, mejor será la calidad de las imágenes producidas.

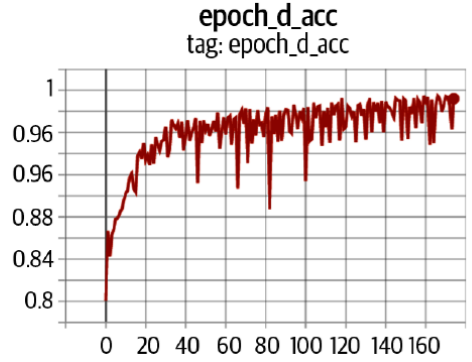
Sin embargo, dado que el generador solo se evalúa en relación con el discriminador actual y este discriminador está mejorando constantemente, no podemos comparar la función de pérdida evaluada en diferentes puntos del proceso de entrenamiento.

De hecho, en este ejemplo, la función de pérdida del generador en realidad aumenta con el tiempo, a pesar de que la calidad de las imágenes está claramente mejorando. **Esta falta de correlación entre la pérdida del generador y la calidad de las imágenes a veces dificulta el monitoreo del entrenamiento de las GAN**.









Deep Convolutional GAN (DCGAN)

Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

Hiperparámetros

- Incluso con las GANs más simples, hay una gran cantidad de hiperparámetros que ajustar.
    - Arquitectura general tanto del discriminador como del generador.
    - Parámetros que gobiernan la normalización por lotes, el dropout, la tasa de aprendizaje, las capas de activación, los filtros convolucionales, el tamaño del kernel, el paso (striding), el tamaño del lote (batch size) y el tamaño del espacio latente.
    - Las GANs son muy sensibles a cambios muy pequeños en todos estos parámetros, y encontrar un conjunto de parámetros que funcione a menudo es un caso de prueba y error informado, más que seguir un conjunto de directrices establecidas.
    - Por eso es importante entender el funcionamiento interno de la GAN y saber cómo interpretar la función de pérdida, para que puedas identificar ajustes sensatos en los hiperparámetros que puedan mejorar la estabilidad del modelo.

Deep Convolutional GAN (DCGAN)

Consejos y trucos para entrenar
(suelen ser muy difíciles de entrenar)

Abordando los desafíos de las GAN

- En los últimos años, varios avances clave han mejorado drásticamente la estabilidad general de los modelos GAN y han disminuido la probabilidad de algunos de los problemas mencionados anteriormente, como el colapso de modo.

- Por ejemplo el Wasserstein GAN con Penalización de Gradiente (WGAN-GP), que realiza varios ajustes clave en el marco de las GAN que hemos explorado hasta ahora para mejorar la estabilidad y la calidad del proceso de generación de imágenes.

Línea de tiempo de IA Generativa

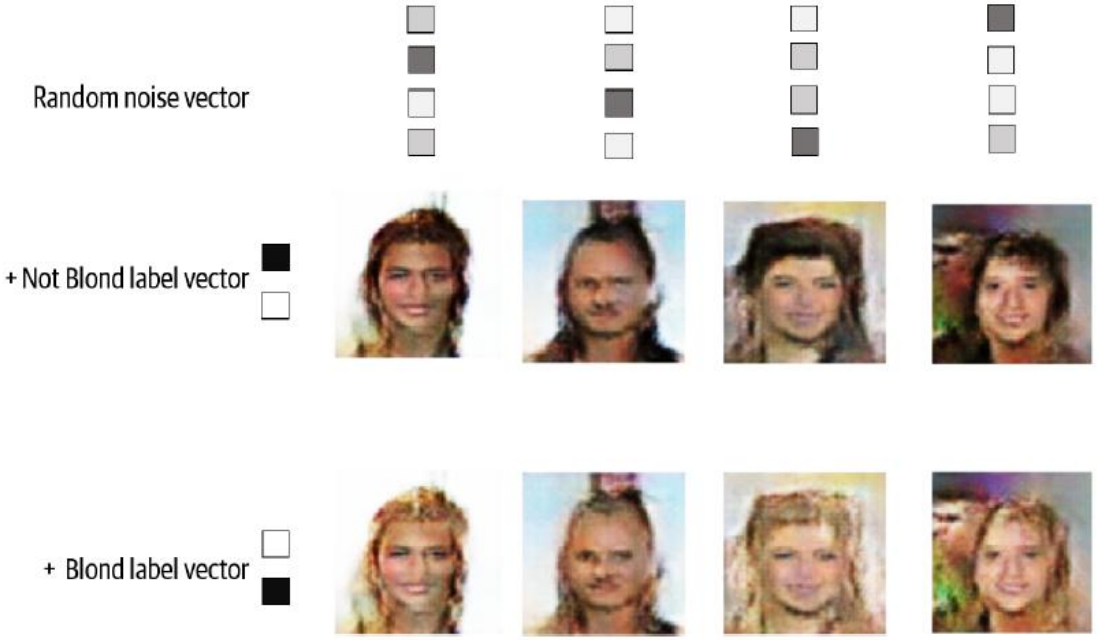
1. La era de las VAEs y GANs (2013-2017)

CGAN (Conditional GAN)

Extensión de la GAN original para permitir el control explícito sobre la salida del generador mediante condiciones, como etiquetas de clase. Esto marca el inicio del enfoque condicional en modelos generativos, donde se puede guiar la generación con información adicional.

2014

Mirza, M., & Osindero, S. (2014). *Conditional generative adversarial nets*. arXiv preprint arXiv:1411.1784.



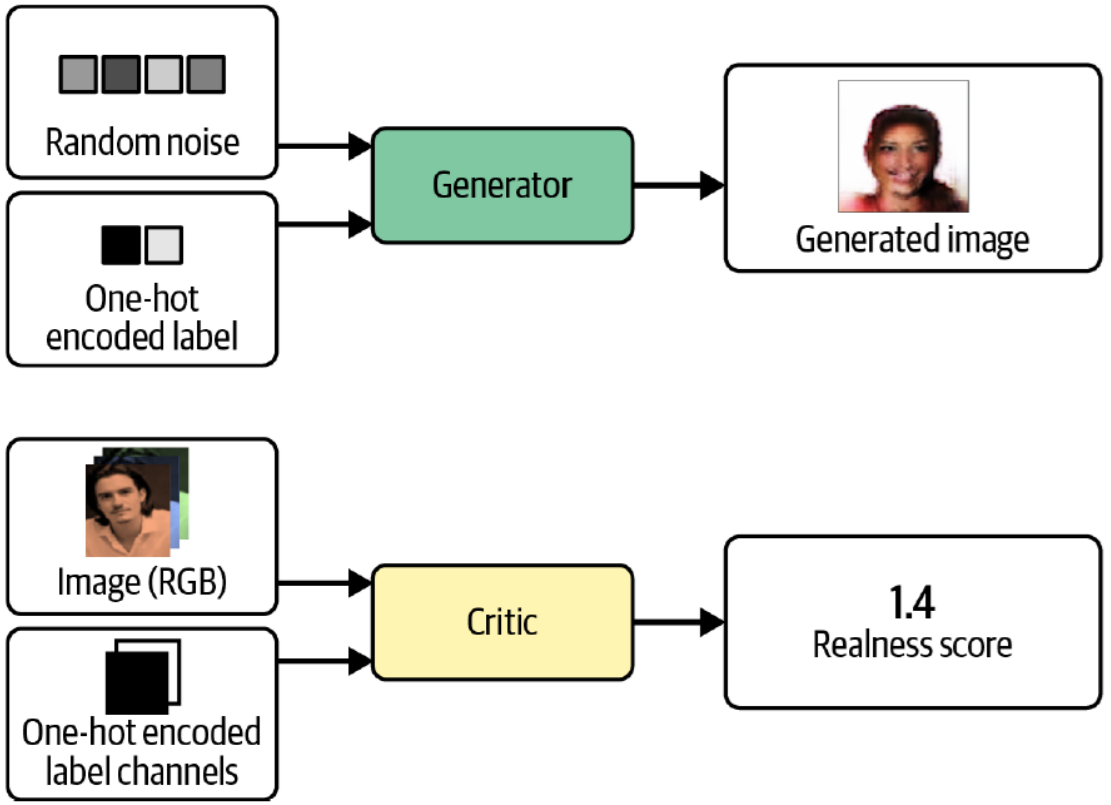
Conditional GAN (CGAN)

- Hasta ahora, hemos construido GANs que son capaces de generar imágenes realistas a partir de un conjunto de entrenamiento dado.
- Sin embargo, no hemos podido controlar el tipo de imagen que nos gustaría generar, por ejemplo, un rostro masculino o femenino, o un bloque de lego grande o pequeño.
- Podemos muestrear un punto aleatorio del espacio latente, pero no tenemos la capacidad de entender fácilmente qué tipo de imagen se generará dada la elección de la variable latente.
- Ahora construiremos un GAN en el que podamos controlar la salida: un llamado GAN condicional.
- Esta idea, presentada por primera vez en "Conditional Generative Adversarial Nets" por Mirza y Osindero en 2014, es una extensión relativamente simple de la arquitectura GAN.

Conditional GAN (CGAN)

Arquitectura

- En este ejemplo, condicionaremos nuestro CGAN sobre el atributo de cabello rubio del conjunto de datos de rostros.
- Podremos especificar explícitamente si queremos generar una imagen con cabello rubio o no.
- Esta etiqueta se proporciona como parte del conjunto de datos CelebA.
- Los CGAN funcionan porque el crítico ahora tiene acceso a información adicional sobre el contenido de la imagen, por lo que el generador debe asegurarse de que su salida esté de acuerdo con la etiqueta proporcionada, para seguir engañando al crítico.
- Si el generador produjera imágenes perfectas que no coincidieran con la etiqueta de la imagen, el crítico sería capaz de detectar que son falsas simplemente porque las imágenes y las etiquetas no coinciden.



Entradas y salidas del generador y del crítico en una CGAN

Conditional GAN (CGAN)

Arquitectura

El único cambio que necesitamos hacer en la arquitectura es concatenar la información de la etiqueta a las entradas existentes del generador y el crítico:

Input layers in the CGAN

```
critic_input = layers.Input(shape=(64, 64, 3)) ❶
label_input = layers.Input(shape=(64, 64, 2))
x = layers.Concatenate(axis = -1)([critic_input, label_input])
...
generator_input = layers.Input(shape=(32,)) ❷
label_input = layers.Input(shape=(2,))
x = layers.Concatenate(axis = -1)([generator_input, label_input])
x = layers.Reshape((1,1, 34))(x)
...
```

Los canales de imagen y los canales de etiqueta se pasan por separado al crítico y se concatenan.

El vector latente y las clases de etiquetas se pasan por separado al generador y se concatenan antes de ser remodelados.

Conditional GAN (CGAN)

Entrenando la CGAN

También debemos realizar algunos cambios en el paso de entrenamiento (train_step) del CGAN para que coincidan con los nuevos formatos de entrada del generador y el crítico.

```
def train_step(self, data):
    real_images, one_hot_labels = data ❶

    image_one_hot_labels = one_hot_labels[:, None, None, :] ❷
    image_one_hot_labels = tf.repeat(
    image_one_hot_labels, repeats=64, axis = 1
)
image_one_hot_labels = tf.repeat(
    image_one_hot_labels, repeats=64, axis = 2
)

batch_size = tf.shape(real_images)[0]
```

❶ Las imágenes y las etiquetas se desempaquetan de los datos de entrada.

❷ Los vectores codificados en one-hot se expanden a imágenes codificadas en one-hot que tienen el mismo tamaño espacial que las imágenes de entrada (64 × 64).

Conditional GAN (CGAN)

Entrenando la CGAN

```
for i in range(self.critic_steps):
    random_latent_vectors = tf.random.normal(
        shape=(batch_size, self.latent_dim)
    )

    with tf.GradientTape() as tape:
        fake_images = self.generator(
            [random_latent_vectors, one_hot_labels], training = True
        ) ❸

        fake_predictions = self.critic(
            [fake_images, image_one_hot_labels], training = True
        ) ❹
        real_predictions = self.critic(
            [real_images, image_one_hot_labels], training = True
        )

        c_wass_loss = tf.reduce_mean(fake_predictions) - tf.reduce_mean(
            real_predictions
        )
        c_gp = self.gradient_penalty(
            batch_size, real_images, fake_images, image_one_hot_labels
        ) ❺
        c_loss = c_wass_loss + c_gp * self.gp_weight

    c_gradient = tape.gradient(c_loss, self.critic.trainable_variables)
    self.c_optimizer.apply_gradients(
        zip(c_gradient, self.critic.trainable_variables)
    )
```

Ahora, el generador recibe una lista de dos entradas: los vectores latentes aleatorios y los vectores de etiquetas codificados en one-hot.

Ahora, el crítico recibe una lista de dos entradas: las imágenes reales/falsas y los canales de etiquetas codificados en one-hot.

La función de penalización del gradiente también requiere que los canales de etiquetas codificados en one-hot sean pasados, ya que utiliza el crítico.

Conditional GAN (CGAN)

Entrenando la CGAN

```
random_latent_vectors = tf.random.normal(
    shape=(batch_size, self.latent_dim)
)

with tf.GradientTape() as tape:
    fake_images = self.generator(
        [random_latent_vectors, one_hot_labels], training=True
    ) ❻
    fake_predictions = self.critic(
        [fake_images, image_one_hot_labels], training=True
    )
    g_loss = -tf.reduce_mean(fake_predictions)

gen_gradient = tape.gradient(g_loss, self.generator.trainable_variables)
```

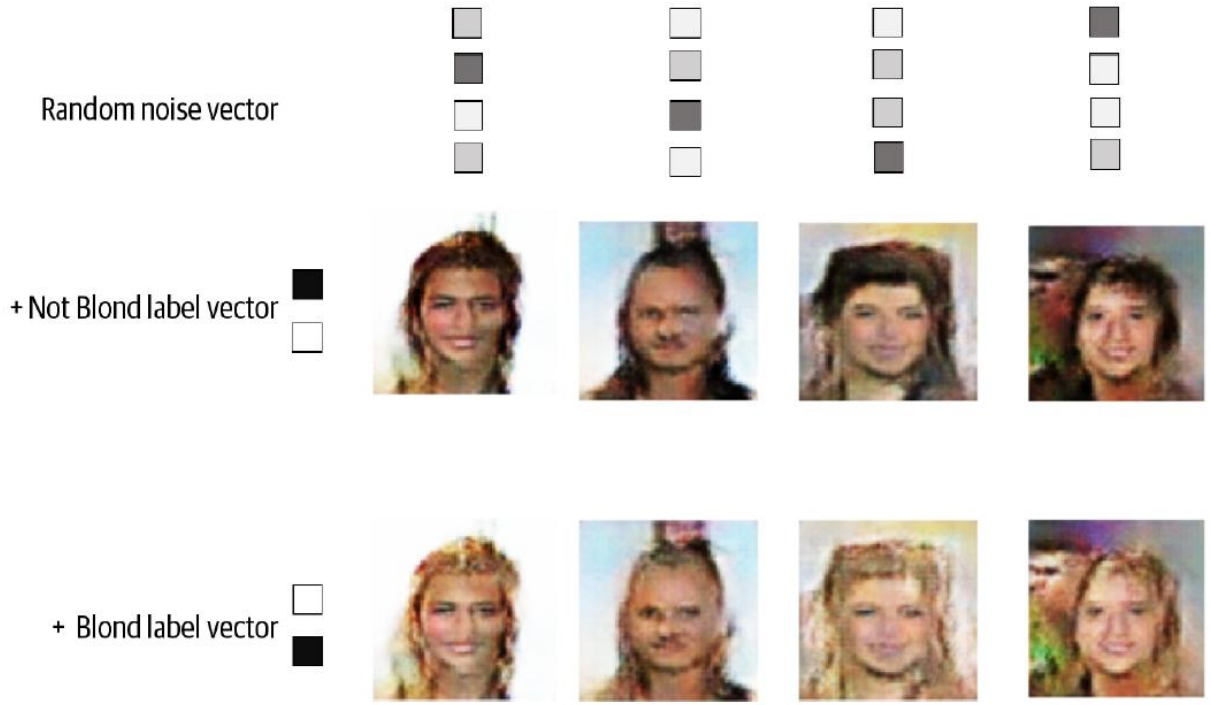
Los cambios realizados en el paso de entrenamiento del crítico también se aplican al paso de entrenamiento del generador.

Conditional GAN (CGAN)

Análisis de la CGAN

- Podemos controlar la salida del CGAN pasando una etiqueta codificada en one-hot particular como entrada al generador.

- Por ejemplo, para generar una cara con cabello no rubio, pasamos el vector [1, 0]. Para generar una cara con cabello rubio, pasamos el vector [0, 1].



Conditional GAN (CGAN)

Análisis de la CGAN

- Si están disponibles etiquetas para tu conjunto de datos, generalmente es una buena idea incluirlas como entrada en tu GAN, incluso si no necesitas condicionar la salida generada en la etiqueta, ya que tienden a mejorar la calidad de las imágenes generadas.

- Puedes pensar en las etiquetas como una extensión altamente informativa de la entrada de píxeles.

"Mujer, sonriendo, con cabello rubio y sin lentes".

Concatenando un **vector multietiqueta** a las entradas del generador y del discriminador.

Permite generar imágenes con **combinaciones específicas de atributos**.

CelebA tiene 40 atributos binarios.