UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER EN TRATAMIENTO ESTADÍSTICO COMPUTACIONAL DE LA INFORMACIÓN

TRABAJO DE FIN DE MÁSTER

Redes Generativas Antagónicas

Antón Makarov Samusev

Director

Francisco Javier Yáñez Gestoso

Madrid, 2019

Resumen

Resumen

Abstract

Abstract

Índice general

Capítulo 1

Introducción

Introducción[1].

[1]Todo el código de este trabajo se puede encontrar en el repositorio de GitHub `https://github.com/ant-mak/tfm`, donde se incluyen las instrucciones para la reproducción de los resultados del proyecto.

Capítulo 2

Preliminares

En este capítulo realizaremos un breve resumen de los conceptos fundamentales del aprendizaje automático, introduciremos las definiciones necesarias para el desarrollo del resto del trabajo y daremos unas nociones básicas sobre aprendizaje profundo, prestando especial atención a las redes convolucionales.

2.1. Aprendizaje automático

El aprendizaje automático tiene como objetivo principal el desarrollo de técnicas que permitan aprender a las máquinas de manera autónoma, sin ser programadas explícitamente para ello, a partir de datos. Dentro del aprendizaje automático existen problemas de muy diversa índole; una manera de clasificarlos es la basada en el tipo de información disponible. En las siguientes secciones describiremos brevemente la clasificación más habitual.

2.1.1. Aprendizaje supervisado

Los problemas de aprendizaje supervisado se caracterizan por la disponibilidad tanto de una serie de muestras X como de la información sobre cuál debe ser el resultado y para cada una de ellas. Buscan encontrar una función que, a partir de una muestra, devuelva el resultado y correspondiente. Algunos de los problemas típicos ante los que nos podemos encontrar dentro del aprendizaje supervisado son:

- Predecir el precio de un piso a partir del número de habitaciones, de los metros cuadrados que tiene, del barrio, etc.
- Predecir si una transacción es o no fraudulenta a partir de la cantidad transferida, del lugar en el que ha sido realizada, de las transacciones realizadas durante los días previos, etc.
- Determinar si una imagen es un perro o un gato.

Algunos de los algoritmos más conocidos para la resolución de problemas de aprendizaje supervisado son la regresión lineal y logística, las máquinas de vector soporte, los árboles de decisión o las redes neuronales.

2.1.2. Aprendizaje no supervisado

En el caso de los problemas de aprendizaje no supervisado tan solo se dispone de las muestras X, sin etiqueta alguna, y se busca recuperar la estructura de los datos. Algunos de los problemas típicos del aprendizaje no supervisado son:

- Segmentación para campañas de publicidad, en las que se dispone de datos socioeconómicos de clientes y se les quiere dividir en varias clases según sus intereses.
- Generación de cuadros nuevos a partir de imágenes de cuadros reales.

Algunos algoritmos utilizados en aprendizaje no supervisado son el K-means o las redes neuronales.

2.1.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo busca determinar la mejor acción a realizar ante una situación concreta para maximizar la recompensa obtenida por dicha acción. Algunos problemas dentro de este tipo de aprendizaje son:

- Enseñan a una máquina a jugar al ajedrez, Go, StarCraft, etc.
- Enseñar a una máquina a conducir.

Los algoritmos más conocidos dentro de este área son el Q-learning, SARSA o DQN.

2.2. Conceptos básicos

Dedicaremos esta sección a la definición de algunos conceptos que serán de utilidad en capítulos posteriores.

Definición 2.1. *La divergencia de Kullback-Leibler es una medida de la diferencia entre dos distribuciones. Se define como:*

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) \mathrm{d}x. \tag{2.1}$$

Definición 2.2. *La divergencia de Jensen-Shannon se define a partir de la de Kullback-Leibler como:*

$$D_{\mathrm{JS}}(P\|Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M), \tag{2.2}$$

donde $M = \frac{P+Q}{2}$. Tiene la ventaja de ser simétrica y finita.

Definición 2.3. *Un equilibrio de Nash es un concepto de solución para juegos no cooperativos de dos o más jugadores en el que cada jugador conoce las estrategias de equilibrio de los demás. Si cada jugador elige una estrategia y ninguno de ellos tiene incentivos para cambiar la suya suponiendo que los demás no lo hacen, se dice que el conjunto de estrategias está en equilibrio.*

Finalmente, introducimos la **ley del inconsciente estadístico**, utilizada para calcular la esperanza de una función $f(X)$ de una variable aleatoria X cuando se conoce la distribución de X pero no la de $f(X)$:

$$\mathbb{E}_{x\sim p}[f(x)] = \int p(x)f(x)\,\mathrm{d}x. \tag{2.3}$$

2.3. Redes neuronales

Las redes neuronales son una clase de algoritmos de aprendizaje automático inspiradas en el estudio biológico de las neuronas en el cerebro. Cabe destacar que pese a que, toman la idea de las neuronas biológicas, no pretenden modelizarlas ni replicar su funcionamiento. Están formadas por una gran cantidad de unidades de procesamiento (o neuronas) interconectadas entre sí. Las fuerzas (o pesos) de cada una de dichas conexiones se modifica en el proceso de entrenamiento con el objetivo de minimizar una cierta función de coste. La minimización se lleva a cabo mediante el algoritmo de descenso en la dirección del gradiente o alguna de sus variantes creadas específicamente para las redes neuronales (SGD, RMSprop, Adam, etc.). Para calcular el gradiente se utiliza el algoritmo de retropropagación, basado en la regla de la cadena. Existe una gran variedad de redes neuronales según el tipo de problema de aprendizaje que se quiera resolver con ellas, la caracterización de las neuronas y la estructura de conexiones de la red. Algunas de las más conocidas son el perceptrón multicapa, las redes convolucionales o las redes recurrentes.

2.4. Aprendizaje profundo

El aprendizaje profundo es un subgrupo de algoritmos de aprendizaje automático que buscan abstraer características de alto nivel presentes en los datos, utilizando generalmente arquitecturas basadas en redes neuronales con una gran cantidad de capas. Para profundizar en las matemáticas detrás del aprendizaje profundo una excelente referencia es [3].

En la actualidad son muy populares debido a su enorme potencial para resolver problemas complejos en ámbitos como la visión por computador o el procesamiento del lenguaje natural, entre otros. Dicho potencial no ha podido ser aprovechado hasta hace muy poco debido a la escasez, por un lado, de conjuntos de datos adecuados y, por otro, de recursos computacionales, ya que estos métodos requieren unas capacidades de cálculo y dimensiones de conjuntos de datos muy superiores a otros métodos más tradicionales.

2.4.1. Redes neuronales convolucionales

Dado que nuestro objetivo es describir las redes generativas antagónicas y, más concretamente, utilizarlas para la generación de imágenes, es imprescindible introducir algunos de los conceptos más importantes de la familia de arquitecturas de aprendizaje profundo que están a la vanguardia en el campo de la visión por computador, las redes neuronales convolucionales (**CNNs** o Convolutional Neural Networks en inglés).

Debemos sus primeros desarrollos a Kunihiko Fukushima, que en 1980 introdujo el Neocognitrón [1] que, posteriormente, sería tomado por Yann LeCun [7], que describió la mayoría de conceptos tal como los conocemos hoy en día.

La particularidad de las CNNs es que introducen una visión local del espacio. Pensemos, por ejemplo, en una imagen de un perro. Parece razonable que intentemos identificar que efectivamente nos encontramos ante un perro y no un gato fijándonos en pequeñas partes de la imagen como ojos, hocico, patas, orejas, etc., en lugar de en todos los píxeles a la vez sin ningún tipo de relación espacial entre sí. Es destacable que este tipo de filtros ya se utilizaba antes de las redes convolucionales, detectando por ejemplo líneas horizontales o verticales y seleccionando regiones de interés manualmente. Sin embargo, las redes convolucionales van más allá, automatizando el proceso de extracción de características y aprendiendo durante el entrenamiento los filtros más idóneos para el problema.

Veamos ahora en detalle el funcionamiento de las redes convolucionales a la vez que desarrollamos los conceptos básicos que nos serán de utilidad a lo largo del resto del trabajo.

Definición 2.4. *Una imagen a color se puede definir como un tensor, con altura, anchura y profundidad. Las dos primeras dimensiones nos indican el tamaño de la imagen, por ejemplo* 64×64 *píxeles. La profundidad contiene los canales de color, generalmente rojo, verde y azul (**RGB** o Red Green Blue en inglés) con una cierta intensidad representada en un rango de* 0 *a* 255*. Mediante la combinación de dichos canales se forman las imágenes a color a las que estamos acostumbrados. En la Figura 2.1 tenemos una representación de esta idea.*

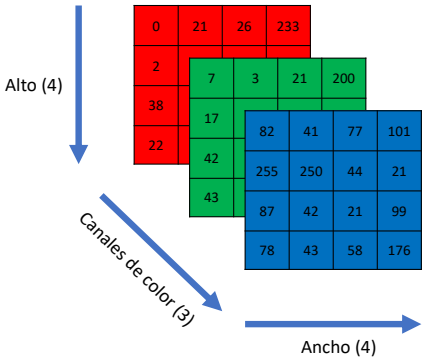

Figura 2.1: Representación esquemática de una imagen.

Una vez definida la imagen formalmente, podemos comenzar con la descripción de la capa de convolución. La función que desempeña es la de extraer características de una imagen de manera automática. Para ello, recorre

el tensor de entrada secuencialmente con una cuadrícula de tamaño predeterminado, aplicando a cada una de las posiciones de la cuadrícula uno o varios filtros (**Kernels** en inglés) cuyos coeficientes son aprendidos por la red. La salida de una operación de convolución recibe el nombre de mapa de características.

En la Figura 2.2 tenemos una representación esquemática de todos los elementos involucrados. En la parte superior está representada una imagen de 5×5 píxeles en 3 canales: rojo, verde y azul. Los bordes de color gris son un relleno (**Padding** en inglés) de ceros cuya función es recoger mejor la información de los bordes de la imagen y modular las dimensiones de salida. La imagen está siendo recorrida por una cuadrícula 3×3 con un desplazamiento (**Stride** en inglés) de tamaño 2, es decir, en cada paso se mueve dos posiciones hacia la derecha y al llegar al final de la línea baja dos posiciones, colocándose de nuevo a la izquierda. En la parte inferior se sitúan los filtros, que generan de el mapa de características representado a la derecha. Estos filtros funcionan de la manera siguiente: primero se realiza un producto elemento a elemento entre la cuadrícula y el filtro en cada canal de color, se suman el resultado del producto y, finalmente, se suman los resultados de cada filtro, dando lugar a un elemento del mapa de características.

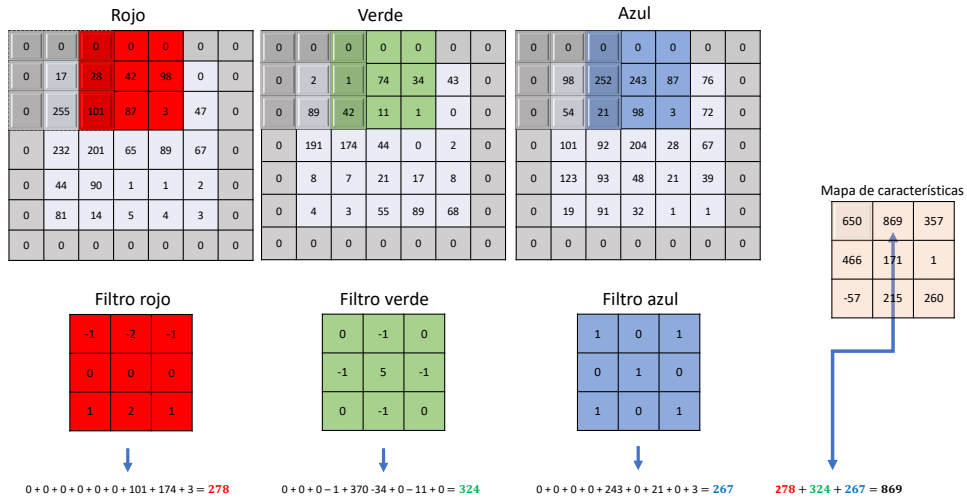

Figura 2.2: Diagrama ilustrativo de la operación de convolución.

En general se puede calcular la dimensión de salida de una capa convolucional mediante la siguiente fórmula:

$$O = \frac{W - K + 2P}{S} + 1,$$

donde W es el tamaño de entrada, K el tamaño del filtro, P el padding y S la longitud del paso. En nuestro ejemplo, por tanto, tendremos un tamaño de salida $O = \frac{5-3+2}{2} + 1 = 3$. En general se suele utilizar más de un filtro en cada capa convolucional, obteniendo así también dimensiones de profundidad.

Observación 2.1. *Como se puede ver en la fórmula, el parámetro stride influye en la dimensión de salida, cosa que utilizaremos extensivamente en las GANs.*

Después de la capa de convolución se suele realizar una reducción de muestreo (**Pooling** en inglés), con el objetivo de reducir la dimensión de los mapas de características y abstraer su contenido para conseguir que la red generalice mejor. Pensemos por ejemplo en un mapa de características que identifica líneas horizontales. No necesitamos saber exactamente dónde tenemos una línea horizontal, nos basta con saber que hay una en el centro de la imagen. La capa de pooling toma cada uno de los mapas de características obtenidos y lo divide en regiones. A los elementos de cada región les aplica una operación para comprimir la información que contienen. Las operaciones más utilizadas son el máximo, la media o el mínimo. En la Figura 2.3 observamos de manera esquemática un mapa de características 4×4 que se ha dividido en 4 regiones 2×2, a las que se aplica la función máximo.

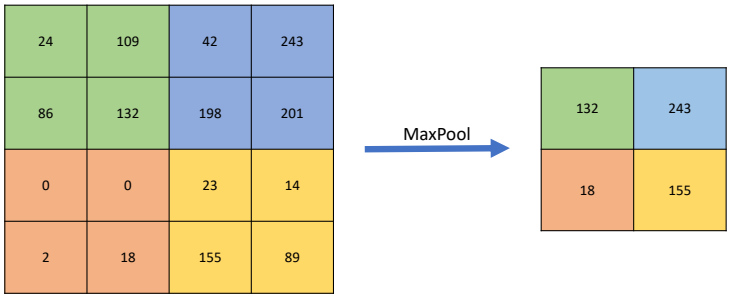

Figura 2.3: Diagrama ilustrativo de reducción de muestreo mediante máximo.

Antes de seguir adelante, es preciso introducir el concepto de **batch**, que, aunque aplica a todo tipo de redes neuronales, en nuestro caso tiene especial relevancia. El batch es un hiperparámetro que controla el número de muestras con el que es entrenada la red antes de proceder a una actualización de pesos. El tamaño del batch puede ir desde 1 hasta el número de muestras del conjunto de datos. Cuando se recorre todo el conjunto de datos se dice que ha pasado una **época**. Por ejemplo, en el caso en que el tamaño de batch sea 100 y el conjunto de datos tenga 1000 muestras, actualizaremos los pesos de la red cada 100 muestras y diremos que hemos completado una época cuando lleguemos a 1000.

Recientemente se ha incorporado una técnica para mejorar la estabilidad en el entrenamiento de redes convolucionales, llamada **Batch Normalization** [5]. La idea fundamental es, si normalizamos los datos antes de introducirlos en la capa de entrada para reducir las influencias de las magnitudes de las variables y ajustarlos a una distribución, ¿por qué no hacer lo mismo en todas las capas?

Dado que los algoritmos de optimización pueden deshacer la normalización que introduzcamos si con ello reducen el valor de la función objetivo, es necesario introducir dos parámetros adicionales γ y β que serán aprendidos

por la red. En el Algoritmo 1 mostramos un breve esquema del funcionamiento. Para más detalles se puede consultar [5].

Algoritmo 1: Batch Normalization

1 Calcular la media del batch: $\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$;
2 Calcular la varianza del batch: $\sigma_b^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$;
3 Normalizar: $\hat{x_i} = \frac{x_i - \mu_b}{\sqrt{\sigma_B}}$;
4 Escalar y trasladar: $y_i = \gamma x_i + \beta$;

Finalmente hablemos de las funciones de activación. Un problema común de las funciones tradicionales como la sigmoide o la tangente hiperbólica es que en redes profundas puede dar lugar al fenómeno conocido como desvanecimiento del gradiente, que ocurre cuando el gradiente se aproxima excesivamente a cero, impidiendo el entrenamiento de la red. Por ejemplo, en el caso de la tangente hiperbólica, tenemos que su derivada se encuentra entre cero y uno. Dado que las redes neuronales utilizan el algoritmo de retropropagación, que a su vez está basado en la regla de la cadena, estaríamos multiplicando cantidades menores que uno tantas veces como capas tengamos, reduciendo de este modo de manera considerable el gradiente y haciendo el uso de estas funciones en redes de muchas capas inapropiado.

Por este motivo es necesario definir otras funciones que sigan siendo no lineales pero tengan un mejor comportamiento ante estos escenarios.

Definición 2.5. *La función de activación unidad lineal rectificada (**ReLU** o REctified Linear Unit en inglés) se define como:*

$$f(x) = \text{máx}(0, x)$$

*De manera similar, se define la función **LeakyReLU**:*

$$f(x) = \begin{cases} x & si\ x > 0 \\ 0{,}01x & en\ otro\ caso \end{cases}$$

Como mencionamos anteriormente, describir exhaustivamente las CNNs no es el objetivo de este trabajo, por lo que nos hemos dejado muchos elementos relevantes en el tintero. Aún así, ya estamos en condiciones de proporcionar la arquitectura básica de una red convolucional. Suele constar de dos partes; en la primera se busca extraer características de las imágenes mediante varias capas de convolución con pooling y activaciones ReLU, aumentando progresivamente la profundidad a la vez que se disminuye anchura y altura. Una vez construidos los mapas de características, se aplanan y se procede a la parte de clasificación, en la que se puede utilizar por ejemplo un perceptrón multicapa. En la Figura 2.4 se puede observar una arquitectura de red convolucional como la que hemos descrito.

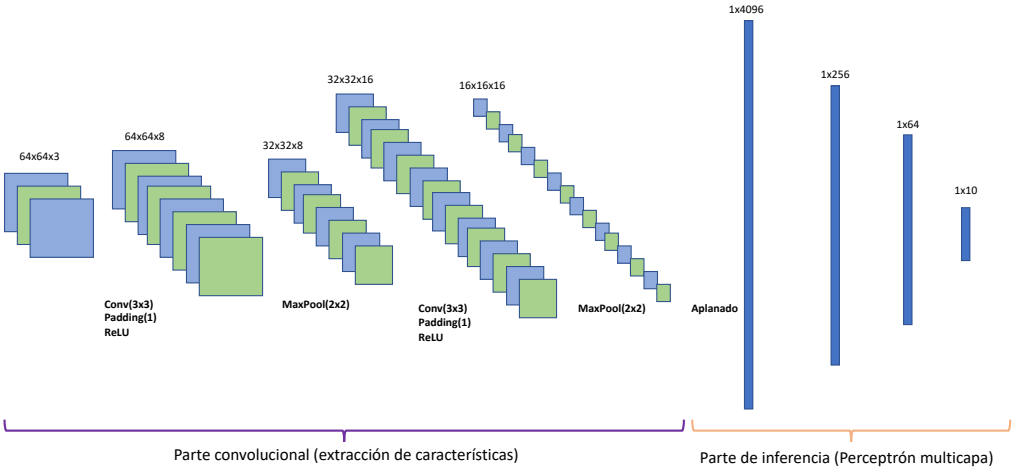

Figura 2.4: Arquitectura básica de una red convolucional

Capítulo 3

Redes generativas antagónicas

En este capítulo describiremos las ideas principales, tanto teóricas como prácticas de las redes generativas antagónicas (**GANs** o Generative Adversarial Networks en inglés). Fueron propuestas por primera vez por Ian Goodfellow en 2014 [4], mezclando conceptos de aprendizaje automático no supervisado, supervisado y teoría de juegos. Desde entonces han suscitado una actividad investigadora muy importante, con aplicaciones en prácticamente todos los campos relacionados con el aprendizaje automático.

3.1. Idea general

El objetivo de las GANs, y en general de los modelos generativos, es aprender la distribución que siguen los datos, pudiendo obtener así, en última instancia, muestras de dicha distribución. En general, las distribuciones que queremos modelar son muy complejas. Supongamos que nuestro objetivo es tomar muestras de la distribución de imágenes de perros, o dicho de otro modo, generar fotos de perros que sean realistas pero que no existan en la realidad ni sean una mezcla de imágenes de nuestro conjunto de entrenamiento. Tenemos la seguridad de que la distribución es extremadamente intrincada, existen perros de distintos colores, tamaños, razas, etc. Este problema es el que van a tratar de atacar las GANs.

La idea fundamental y más novedosa detrás de las GANs es poner dos redes neuronales a competir entre sí. Una red, llamada generadora (G), está dedicada a obtener imágenes a partir de ruido aleatorio con distribución $p_z(z)$, mientras que otra, llamada discriminadora (D), trata de averiguar si la imagen es real o ficticia. Es frecuente ilustrar esta idea mediante la analogía de falsificadores de billetes que tratan de engañar a la policía. Los falsificadores empiezan dibujando billetes que no tienen nada que ver con los reales, intentando utilizarlos para realizar pagos, momento en el que son atrapados por la policía. Los falsificadores por tanto se dan cuenta de que están dibujando los billetes de manera incorrecta y modifican su técnica, mientras que la policía va aprendiendo a su vez a detectar mejor los billetes falsos. De este modo, a lo largo del entrenamiento se busca llegar a un equilibrio, en el que la policía no sea capaz de discernir los billetes falsos de los verdaderos, obteniendo así los ladrones una falsificación realista.

Traduzcamos ahora esta idea a términos matemáticos. Sea el generador G

un PMC con parámetros θ_g que tiene por objetivo encontrar una distribución p_g lo más parecida posible a la distribución de los datos p_d. Para ello toma un vector aleatorio z distribuido según una cierta distribución p_z y lo lleva al espacio definido por los datos. Por otro lado, sea el discriminador D otro PMC con parámetros θ_d, que, para una muestra, devuelve la probabilidad de que esta sea real (x) o falsa ($G(z)$). Entrenaremos D para que maximice la probabilidad de asignar la etiqueta correcta tanto a los ejemplos de entrenamiento como a los generados, mientras que, al mismo tiempo, entrenamos G para que minimice $\log(1 - D(G(z)))$. Es decir, tenemos el siguiente juego minimax con función de utilidad $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_d(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]. \quad (3.1)$$

En la Figura **??** mostramos una representación esquemática del funcionamiento de las GANs y en el Algoritmo 2 lo describimos de manera más concisa, dando además, una primera idea de cómo se lleva a cabo el entrenamiento.

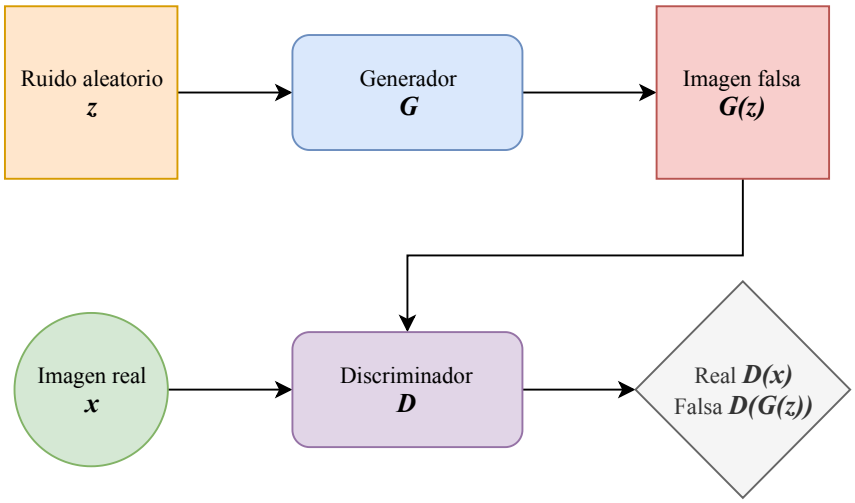

Figura 3.1: Diagrama conceptual de las redes generativas antagónicas.

3.2. Bases teóricas

En esta sección realizaremos un análisis de la teoría que hay detrás de las redes generativas antagónicas, utilizando como referencias principales [4, 2]. En primer lugar veremos cuales son el generador y discriminador óptimos, mostrando después que el Algoritmo 2 logra que p_g converja a p_d.

3.2.1. Óptimo global

Veamos en primer lugar cual es el discriminador óptimo para un generador dado.

Teorema 3.1. *Para un generador G fijo, el discriminador D óptimo es:*

$$D^*_G(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}. \quad (3.2)$$

Algoritmo 2: Entrenamiento minibatch de una red generativa antagónica. El valor de k es un hiperparámetro que se puede elegir libremente.

for *Iteraciones de entrenamiento* **do**
  for k *pasos* **do**
    Tomar muestra $(z^{(1)}, z^{(2)}, \ldots, z^{(m)})$ de tamaño m de $p_g(z)$;
    Tomar muestra $(x^{(1)}, x^{(2)}, \ldots, x^{(m)})$ de tamaño m de $p_d(x)$;
    Actualizar el discriminador ascendiendo su gradiente:

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

  end
  Tomar muestra $(z^{(1)}, z^{(2)}, \ldots, z^{(m)})$ de tamaño m de $p_g(z)$;
  Actualizar el generador ascendiendo su gradiente:

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

end

Demostración. El discriminador busca maximizar la Ecuación (3.1), que podemos reescribir, utilizando la ley del inconsciente estadístico como:

$$\begin{aligned} V(G, D) &= \int_x p_d(x) \log(D(x))\, \mathrm{d}x + \int_z p_z(z) \log(1 - D(g(z)))\, \mathrm{d}z \\ &= \int_x \left(p_d(x) \log(D(x)) + p_g(x) \log(1 - D(x))\right) \mathrm{d}x. \end{aligned} \tag{3.3}$$

Maximizando el integrando obtendremos el discriminador óptimo. Para ello, reescribimos el integrando como:

$$f(y) = a \log y + b \log(1 - y),$$

derivamos e igualamos a cero:

$$f'(y) = 0 \implies \frac{a}{y} - \frac{b}{1 - y} = 0 \implies y = \frac{a}{a + b}.$$

Suponiendo que $a + b \neq 0$, hacemos la segunda derivada en $\frac{a}{a+b}$, obteniendo:

$$f''\left(\frac{a}{a+b}\right) = -\frac{a}{\left(\frac{a}{a+b}\right)^2} - \frac{b}{\left(1 - \frac{a}{a+b}\right)^2},$$

que es menor que cero para $(a.b) \in (0, 1)$. Tomando por tanto $D = \frac{p_d}{p_d + p_g}$ tenemos el resultado, que además es único, ya que f tiene un solo máximo en el intervalo. □

Es importante considerar que en la práctica, al desconocer $p_d(x)$, no podemos obtener el D óptimo, pero su existencia nos permitirá demostrar ahora que existe un óptimo para G.

Teorema 3.2. *El mínimo global de* $C(G) = \max_D V(G, D)$ *se alcanza si y solo si* $p_g = p_d$.

Demostración. Para $p_g = p_d$, sustituyendo en la Ecuación (3.2), obtenemos que $D^*_G = \frac{1}{2}$. Podemos por tanto escribir:

$$\begin{aligned} V(G, D^*_G) &= \int_x \left(p_d(x) \log \frac{1}{2} + p_g(x) \log \left(1 - \frac{1}{2} \right) \right) \mathrm{d}x \\ &= \int_x \log \frac{1}{2} (p_d(x) + p_g(x)) \, \mathrm{d}x \\ &= -\log 2 \left(\int_x p_d(x) \, \mathrm{d}x + \int_x p_g(x) \, \mathrm{d}x \right) \\ &= -2 \log 2 = -\log 4. \end{aligned}$$

Así, nuestro valor candidato para ser el mínimo global es $-\log 4$. Veamos qué pasa si descartamos la hipótesis de que $p_g = p_d$. Para todo G, podemos sustituir D^*_G en $C(G)$:

$$C(G) = \int_x \left(p_d(x) \log \left(\frac{p_d(x)}{p_g(x) + p_d(x)} \right) + p_g(x) \log \left(\frac{p_g(x)}{p_g(x) + p_d(x)} \right) \right) \mathrm{d}x. \tag{3.4}$$

Donde el segundo sumando de la integral proviene de:

$$\begin{aligned} 1 - D^*_G(x) &= 1 - \frac{p_d(x)}{p_g(x) + p_d(x)} \\ &= \frac{p_g(x) + p_d(x)}{p_g(x) + p_d(x)} - \frac{p_d(x)}{p_g(x) + p_d(x)} \\ &= \frac{p_g(x)}{p_g(x) + p_d(x)}. \end{aligned}$$

Sumando y restando $p_d(x) \log 2$ a ambos sumandos de (3.4) se tiene que:

$$\begin{aligned} C(G) = \int_x &(\log 2 - \log 2) p_d(x) + p_d(x) \log \left(\frac{p_d(x)}{p_g(x) + p_d(x)} \right) \\ &+ (\log 2 - \log 2) p_g(x) + p_g(x) \log \left(\frac{p_g(x)}{p_g(x) + p_d(x)} \right) \mathrm{d}x. \end{aligned} \tag{3.5}$$

Podemos escribir esto como:

$$\begin{aligned} C(G) = &-\log 2 \int_x p_d(x) + p_g(x) \, \mathrm{d}x \\ &+ \int_x p_d(x) \left(\log 2 + \log \left(\frac{p_d}{p_g + p_d} \right) \right) \\ &+ p_g(x) \left(\log 2 + \log \left(\frac{p_g}{p_g + p_d} \right) \right) \mathrm{d}x. \end{aligned} \tag{3.6}$$

Simplificando términos obtenemos:

$$-\log 4 + \int_x p_d(x) \log \left(\frac{p_d}{\frac{p_g + p_d}{2}} \right) \mathrm{d}x + \int_x p_g(x) \log \left(\frac{p_g}{\frac{p_g + p_d}{2}} \right) \mathrm{d}x. \tag{3.7}$$

Ahora podemos identificar la ecuación 3.7 como la divergencia de Kullback-Leibler:

$$C(G) = -\log 4 + D_{\mathrm{KL}}\left(p_d \middle\| \frac{p_d + p_g}{2}\right) + D_{\mathrm{KL}}\left(p_g \middle\| \frac{p_d + p_g}{2}\right). \tag{3.8}$$

Podemos escribir la ecuación anterior utilizando la divergencia de Jensen-Shannon:

$$C(G) = -\log 4 + 2D_{\mathrm{JS}}(p_d \| p_g). \tag{3.9}$$

Dado que D_{JS} es no negativa y vale cero en el caso de que las distribuciones sean iguales, tenemos que $-\log 4$ es el mínimo global de $C(G)$ y la única solución es $p_g = p_d$. □

3.2.2. Convergencia

Una vez hemos probado que el óptimo global del juego minimax existe y es único, necesitamos una manera de llegar a él, es decir, demostrar que existe un algoritmo que converge a dicho óptimo.

Teorema 3.3. *Si G y D tienen suficiente capacidad y en cada paso del Algoritmo 2 se permite al discriminador alcanzar su óptimo para dado el generador G, actualizando p_g tal que se mejore el criterio:*

$$\mathbb{E}_{x\sim p_d}[\log(D^*_G(x))] + \mathbb{E}_{x\sim p_g}[\log(1 - D^*_G(x))], \tag{3.10}$$

entonces p_g converge a p_d.

Demostración. Sea $V(G, D) = U(p_g, D)$, dependiente de p_g, con $U(p_g, D)$ convexa. Las subderivadas del supremo de una función convexa incluyen la derivada de la función donde se alcanza el máximo. Es decir, si $f(x) = \sup_{a\in\mathcal{A}} f_\alpha(x)$ y $f_\alpha(x)$ es convexa para todo α, entonces:

$$\partial f_\beta(x) \in \partial f, \tag{3.11}$$

si $\beta = \arg\sup_{\alpha\in\mathcal{A}} f_\alpha(x)$. Esto es equivalente a actualizar p_g según el descenso en la dirección del gradiente para el discriminador óptimo dado el generador G. Como hemos visto en el Teorema 3.2 $\sup U(p_g, D)$ es convexo en p_g y con óptimo global único, por tanto, con actualizaciones lo suficientemente pequeñas de p_g, obtendremos la convergencia deseada. □

Finalmente, es importante resaltar que, aunque el juego minimax tiene estas excelentes propiedades teóricas, en la práctica es necesario hacer una ligera modificación. Al principio del entrenamiento, cuando G aún tiene bajas prestaciones, D rechaza sus muestras con mucha confianza, lo que provoca que $\log(1 - D(G(z)))$ se sature con facilidad. Por ello, es frecuente entrenar G para que maximice $\log D(G(z))$, que converge al mismo equilibrio pero permite entrenar con mayor estabilidad.

Capítulo 4

Generación de arte

En esta sección nos vamos a apoyar sobre el artículo [8] para implementar en `Python`, haciendo uso de `PyTorch`, una red convolucional generativa antagónica profunda (**DCGAN** o Deep Convolutional Generative Adversarial Networks en inglés), con la cual vamos a generar imágenes de cuadros realistas.

4.1. DCGAN

Las redes convolucionales generativas antagónicas profundas son una extensión de las GANs tradicionales. Una de sus características principales es el uso de redes convolucionales con arquitecturas relativamente sencillas para el discriminador y redes con convoluciones fraccionales para el generador.

Definición 4.1. *Una convolución fraccional (también llamada deconvolución o convolución traspuesta en algunos textos) es.*

Además, en el artículo [8] los autores proporcionan una serie de recomendaciones empíricas para incrementar la estabilidad y obtener imágenes de mayor calidad. Los puntos más relevantes que se pueden extraer del artículo son:

- En el discriminador, utilizar convoluciones con stride en lugar de convoluciones con pooling.
- En el generador, utilizar convoluciones fraccionales con stride en lugar de convoluciones con pooling.
- No utilizar capas totalmente conectadas.
- Utilizar funciones de activación ReLU en todas las capas del generador salvo en la última, en la que utilizar tanh.
- Utilizar funciones de activación LeakyReLU en todas las capas del discriminador.
- Utilizar batch normalization tanto para el generador como el discriminador.
- Inicializar los pesos de ambas redes según una distribución normal.

El objetivo del trabajo es la generación de cuadros realistas, para ello, es necesario una base de datos de gran tamaño con cuadros de distintos artistas, épocas y estilos. En una competición de Kaggle[1] hemos encontrado un conjunto de datos con más de 100000 imágenes, ocupando aproximadamente 49 GB en disco. Una muestra de dichas imágenes se puede observar en la Figura 4.1

(a) (b) (c)

Figura 4.1: Imágenes del conjunto de datos original sin preprocesar.

4.1.1. Preprocesado

Como paso previo a la definición de la arquitectura de las redes y su posterior entrenamiento, ha sido necesario procesar las imágenes, paso que nos ha servido también para formarnos una idea de cómo es el conjunto de datos.

En un principio teníamos imágenes etiquetadas con el nombre del autor y estilo de cada cuadro. Dado que dicha información no es relevante para nuestro propósito, la hemos descartado (se podría haber aprovechado, pero excede el alcance de este trabajo). Por otro lado, teníamos las imágenes divididas en conjuntos de entrenamiento y test, que hemos unificado, ya que nuestro problema pertenece al ámbito del aprendizaje no supervisado. Finalmente nos hemos encontrado con imágenes corruptas que hemos tenido que eliminar, por ejemplo, imágenes descargadas incorrectamente, de tamaños excesivamente grandes o en formatos incorrectos.

Una vez limpio el conjunto de datos, hemos realizado algunas transformaciones para que, posteriormente, nuestras redes reciban elementos de entrada uniformes. En primer lugar hemos escalado los tamaños y las proporciones, pasando así de cuadros de todos los tamaños y de distintas formas a cuadrados de 64×64 píxeles. Posteriormente, realizamos un recorte centrado y finalmente las cargamos como tensores, que es el formato utilizado por `PyTorch`. En la Figura **??** se puede observar una muestra de las imágenes preprocesadas.

[1] `https://www.kaggle.com/c/painter-by-numbers/data`.

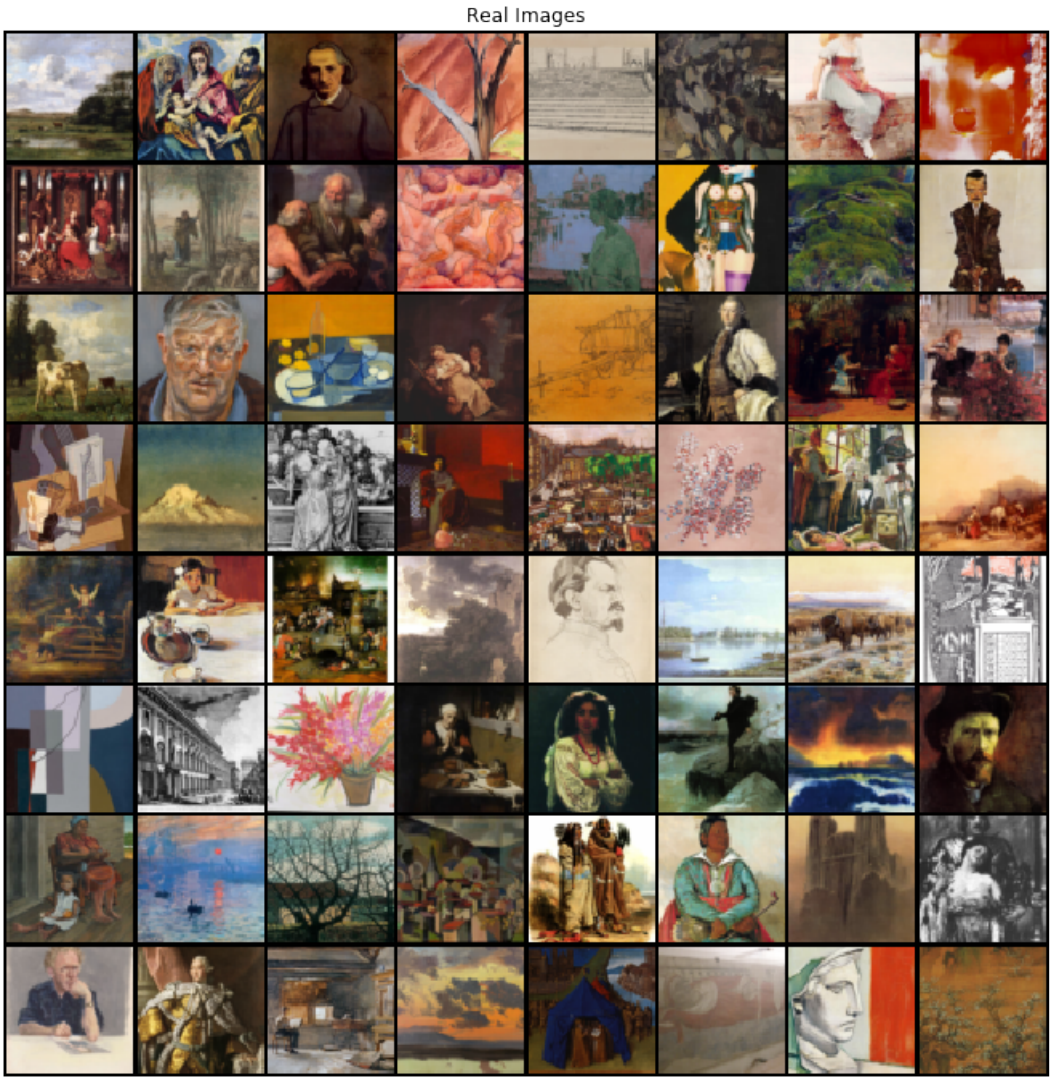

Figura 4.2: Imágenes del conjunto de datos original con el preprocesado realizado.

4.1.2. Arquitectura

Basándonos en las recomendaciones mencionadas anteriormente, hemos decidido utilizar la arquitectura que mostramos en la Figura **??**, con una red convolucional sencilla para el discriminador y una red de convoluciones fraccionales para el generador, para consultar los detalles a más bajo nivel, recomendamos visitar la página correspondiente al código en el repositorio creado para este trabajo[2].

Hemos inicializado los pesos de ambas redes según una distribución normal $\mathcal{N}(0, 0{,}0002)$ y seleccionado un tamaño de batch de 128. El algoritmo de optimización por el que hemos optado es Adam [6], seleccionando una tasa de aprendizaje de 0,0002 y $\beta = (0{,}5, 0{,}999)$. Dado que el entrenamiento es muy costoso, hemos considerado prudente entrenar durante 30 épocas como máximo, guardando no obstante los modelos en cada época para después

[2] `https://github.com/ant-mak/tfm/blob/master/src/tfm_teci_antonmakarov_gan-paintings.ipynb`.

poder volver a ellos y realizar comparaciones en la calidad de las imágenes generadas. Para poder hacer las comparaciones algo más fiables, hemos fijado un vector de ruido aleatorio al principio y generado una muestra a partir de él cada 200 batches y al final de cada época, obteniendo de este modo una visualización de la evolución del entrenamiento[3].

4.1.3. Recursos

Como hemos ido comentando a lo largo de todo el trabajo, el entrenamiento de las GANs no es para nada sencillo, siendo necesaria una cuidadosa elección del conjunto de datos, de la arquitectura de la red y de los hiperparámetros. Sin embargo esto es tan solo una parte de la dificultad. También es necesario disponer de unas capacidades de cómputo generosas. Ha sido imprescindible el uso de un ordenador con GPU compatible con CUDA y una gran cantidad de memoria RAM. En concreto el equipo que hemos utilizado contaba con una GPU Nvidia Quadro P5000 y 32GB de RAM. Con esta configuración, las 30 épocas de entrenamiento se prolongaron durante aproximadamente 24 horas. Para contrastar, también hemos realizado pruebas entrenando con CPU en un portátil de modestas prestaciones, observando un rendimiento aproximadamente 20 veces menor.

4.2. Resultados

En la Figura **??** mostramos medio batch de imágenes generadas con nuestra GAN a partir de un vector de entrada aleatorio, sin realizar ningún tipo de selección ni procesado. Observamos que, en general y a primera vista, algunas podrían hacerse pasar por obras de arte, incluso podríamos distinguir paisajes, retratos o composiciones abstractas. Sin embargo, tras una inspección más cuidadosa, es claro que las imágenes que no pueden ser catalogadas como abstractas carecen de detalles o presentan formas inusuales, bastante típicas en las imágenes generadas con GANs. Con todo, consideramos que las imagenes obtenidas son visualmente agradables.

[3]Una animación de la evolución de la muestra desde ruido aleatorio hasta imágenes realistas se puede ver en el repositorio.

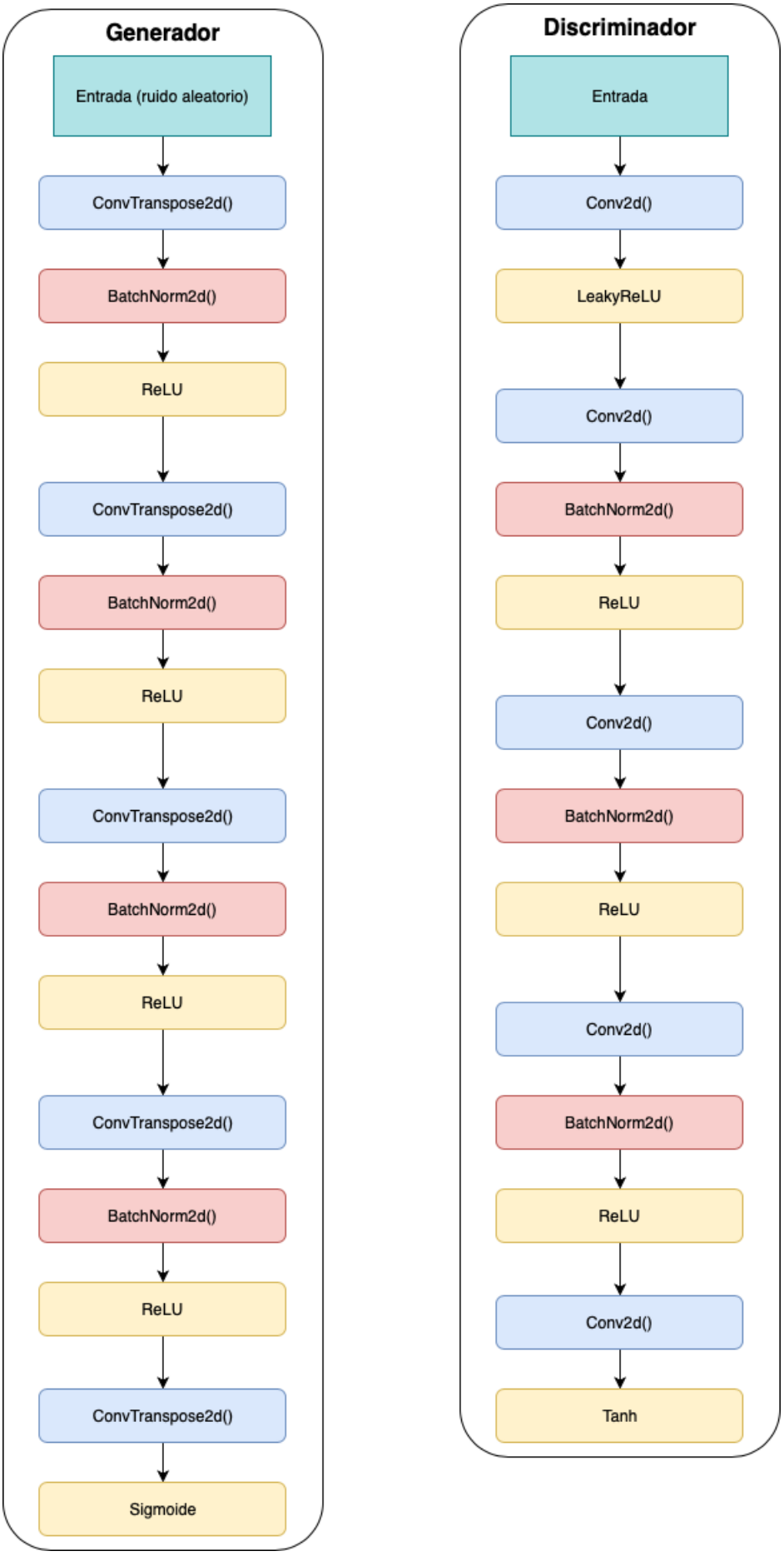

Figura 4.3: Arquitectura de las redes (Cambiar, poner tamaños de entrada y salida).

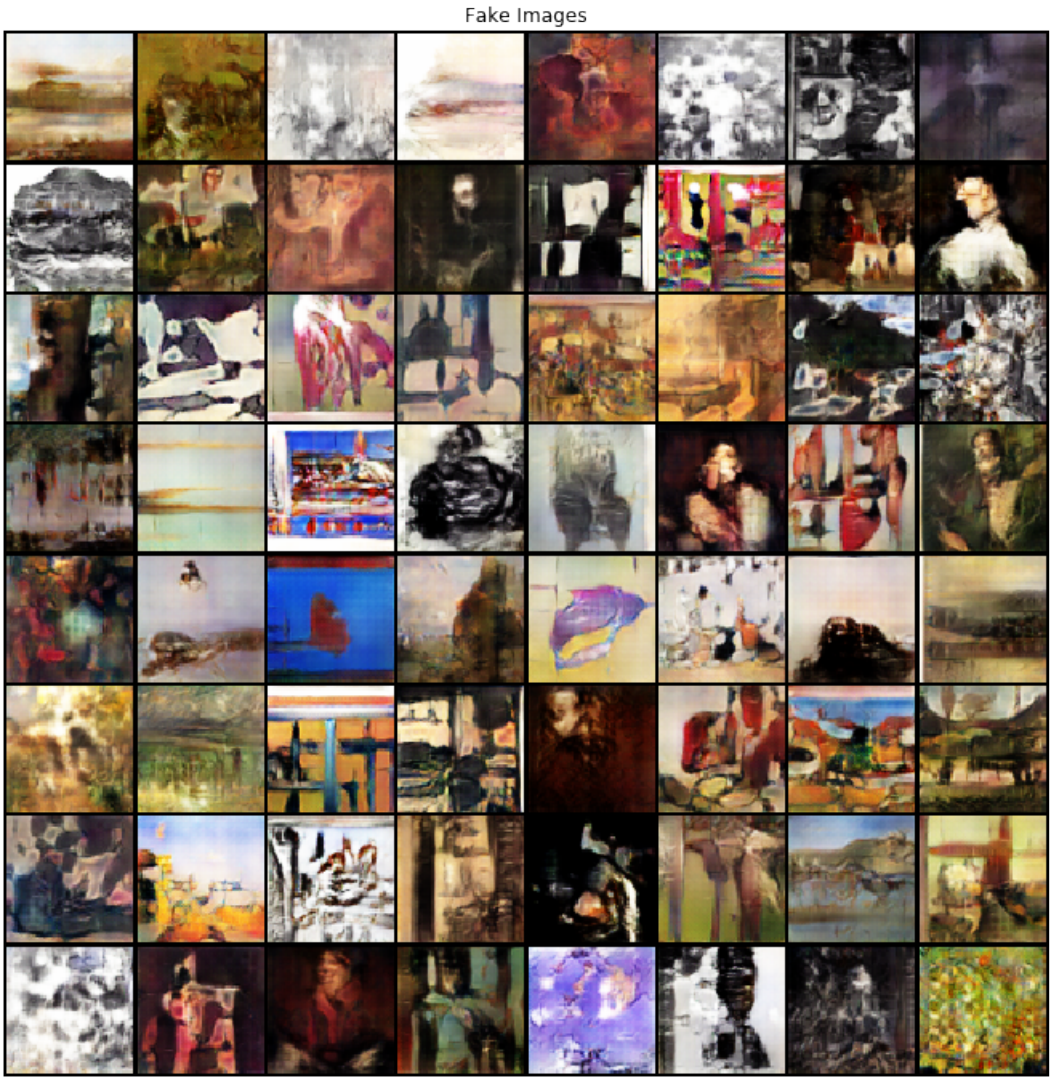

Figura 4.4: Imágenes generadas mediante la DCGAN implementada después de 30 épocas.

Capítulo 5

Conclusión

Bibliografía

[1] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, (36):193–302, 1980.

[2] I. Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[5] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[8] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.