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] [4], [2], [7], [3]

[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, haciendo hincapié en las redes convolucionales.

2.1. Conceptos básicos

Definición 1 (Divergencia de Kullback-Leibler)**.** *KL div*

Definición 2 (Divergencia de Jensen-Shannon)**.** *JS div*

Definición 3 (Equilibrio de Nash)**.** *Eq. Nash*

2.2. Redes neuronales

2.2.1. 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.

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 tamaños de conjuntos de datos muy superiores a otros métodos más tradicionales.

2.2.2. 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 (convolutional neural networks o CNNs en inglés).

Debemos sus primeros desarrollos a Kunihiko Fukushima, que en 1980 introdujo el neocognitrón [1], que posteriormente sería tomado por Yann LeCun, que introdujo la mayoría de conceptos por los que conocemos este tipo de redes hoy en día.

La particularidad de las CNNs es que introducen una visión local del espacio. Pensemos en una imagen de un perro, parece razonable que intentemos identificar que efectivamente lo que vemos es un perro y no un gato fijándonos en pequeñas partes de la imagen como ojos, hocico, patas, orejas, etc. en lugar de todos los píxeles a la vez sin ningún tipo de relación espacial entre si. Es destacable que este tipo de filtros ya se hacían antes de las redes convolucionales, seleccionando regiones de interés manualmente. Sin embargo, las redes convolucionales van más allá, automatizando el proceso de extracción de características, aprendiendo durante el entrenamiento los filtros más idóneos.

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 4. *Una imagen a color se puede definir como un tensor, con altura, y anchura el tamaño usual, por ejemplo* 64×64 *píxeles, a la que se añade otra dimensión, que llamaremos profundidad en la que se encuentran los canales de color rojo, verde y azul (Red Green Blue o RGB 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* **??** *tenemos una representación de esta idea.*

Ahora que tenemos la definición formal de imagen, comencemos con la descripción de la operación más importante, y la que da nombre a estas redes, la convolución. Tiene la función de extraer características de manera automática. Para ello, recorre el tensor de entrada de manera secuencial con una cuadrícula, aplicando a cada cuadrícula uno o varios filtros o **kernels**, de tamaño predeterminado y cuyos coeficientes son aprendidos por la red. La salida de una operación de convolución recibe el nombre de mapa de características. Para entender mejor el funcionamiento de la convolución veamos la Figura **??**. Tenemos a la derecha, representada en color azul una imagen de 5×5 píxeles en 3 canales, rojo, verde y azul a la que se añade un relleno o **padding** de ceros, representado en gris rodeando cada canal de la imagen, cuya función es recoger mejor la información sobre los bordes de la imagen y modular las dimensiones de salida. Recorremos la imagen mediante una cuadrícula 3×3 con un desplazamiento o **stride** de una posición. En la parte central de la imagen aparecen dos columnas con recuadros de color rojo. Dichos recuadros son los filtros. En este caso se utilizan dos, para obtener dos mapas de características distintos, representados a la izquierda en verde. Para obtener los mapas de características se realiza un producto elemento a elemento entre las cuadrículas de cada canal de color con las componentes de los filtros y finalmente se suman por posición.

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 $O = \frac{5-3+2}{1} + 1 = 5$ y, como hemos utilizado dos filtros, tendrá profundidad 2, en notación común $5 \times 5 \times 2$.

Observación 1. *A la vista de la fórmula, el parámetro stride también influye en la dimensión de salida, cosa que utilizaremos esto extensivamente en las GANs.*

Una vez completada la fase de convolución, esta se suele acompañar de una reducción de muestreo o **pooling**, donde tomamos cada uno de los mapas de características obtenidos y lo dividimos en regiones. A los elementos de cada región les aplicamos una operación para comprimir la información que contienen. Las operaciones más utilizadas son el máximo, la media o el mínimo. En la Figura **??** nos encontramos ante un mapa de características de tamaño 4×4 que se ha dividido en 4 regiones 2×2. A los valores de cada región se les aplica la función máximo, obteniendo la matriz de la izquierda, reduciendo así la dimensión del mapa de características.

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 consultar [5].

Algoritmo 1: Entrenamiento de una red generativa antagónica genérica.

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: $y_i = \gamma x_i + \beta$;

Definición 5 (ReLU)**.** *La función de activación unidad lineal rectificada (REctified Linear Unit o ReLU en inglés) se define como:*

$$f(x) = \text{máx}(0, x)$$

También será de utilidad la función LeakyReLU, definida como:

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

Estas funciones se utilizan en aprendizaje profundo en lugar de las sigmoides y tangentes hiperbólicas porque son más eficientes computacionalmente y por su capacidad de evitar el desvanecimiento del gradiente.

Definición 6. *El desvanecimiento del gradiente se produce cuando la función de pérdida se acerca demasiado a cero. Ocurre cuando se añaden muchas capas que utilizan ciertos tipos de funciones de activación. Por ejemplo la sigmoide, que, para cambios grandes en la entrada, produce cambios muy pequeños en la salida, es decir, con derivada casi nula. Dado que para el cálculo del gradiente se utiliza el algoritmo de retropropagación, dichos valores casi nulos se multiplican entre si, resultando un gradiente despreciable.*

Capítulo 3

Redes generativas antagónicas

En este capítulo describiremos las ideas principales, tanto teóricas como prácticas de las redes generativas antagónicas, generative adversarial networks o GANs 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 $p_g(z)$, 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 sepa discernir los billetes falsos de los verdaderos, obteniendo así los ladrones una falsificación realista. Una representación esquemática se puede ver en la Figura 3.1. Podemos traducir esta idea a términos matemáticos

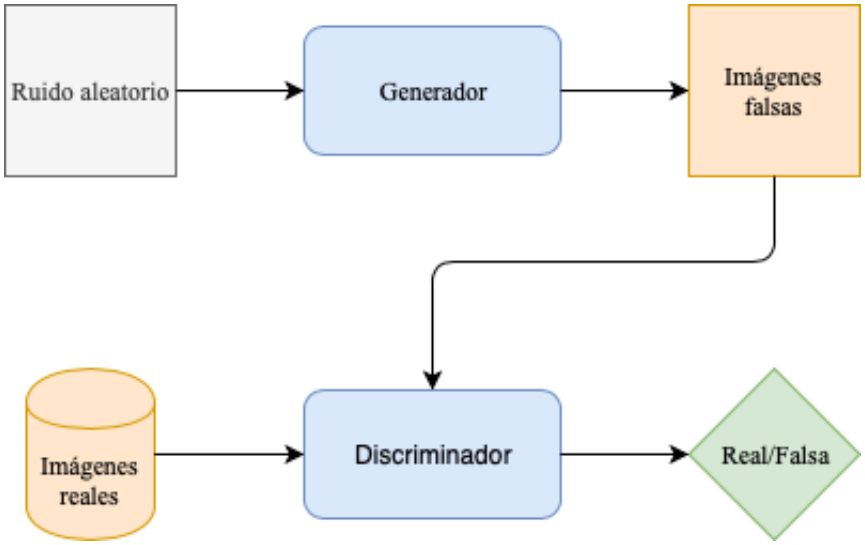

Figura 3.1: Diagrama conceptual de las redes generativas antagónicas.

de la siguiente manera:

Para aprender la distribución p_g del generador sobre los datos x, definimos una distribución sobre las variables de ruido de entrada $p_z(z)$. Mediante un perceptrón multicapa con parámetros θ_g definimos la función $G(z;\theta_g)$. Definimos también mediante otro perceptrón multicapa $D(x;\theta_d)$ la probabilidad de que x es una muestra de los datos y no del generador. Entrenamos D para que maximice la probabilidad de asignar la etiqueta correcta tanto a los ejemplos de entrenamiento como a los generados. Al mismo tiempo, entrenamos G para que minimice $\log(1 - D(G(z)))$. Es decir, tenemos el siguiente juego minimax:

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

En el Algoritmo 2 describimos lo anterior de manera más concisa y damos una primera idea de cómo se lleva a cabo el entrenamiento.

3.2. Bases teóricas

En esta sección realizaremos un análisis de la teoría que hay detrás de las redes generativas antagónicas. Mostraremos que el criterio de entrenamiento nos permite recuperar la distribución de los datos si damos a G y D capacidad suficiente.

Teorema 1. *Para un generador G fijo, el discriminador D óptimo es:*

$$D^*_G(x) = \frac{p_{datos}(x)}{p_{datos}(x) + p_g(x)} \tag{3.2}$$

Demostración. El discriminador busca maximizar su función de utilidad, dada por $V(G,D)$

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

Algoritmo 2: Entrenamiento de una red generativa antagónica genérica.

for *Iteraciones de entrenamiento* **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:

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

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

□

3.3. Otros modelos generativos

Variational autoencoders, noise contrastive estimation etc.

Capítulo 4

Generación de arte

En esta sección nos vamos a apoyar sobre el artículo [7] para implementar en `Python`, haciendo uso de `PyTorch` una red convolucional profunda generativa antagónica, con la cual vamos a tratar de generar imágenes de cuadros realistas.

4.1. DCGAN

Las redes convolucionales profundas generativas antagónicas (deep convolutional generative adversarial networks, DCGAN) son una extensión de las GANs tradicionales, que implementan una serie de recomendaciones para incrementar la estabilidad y obtener imágenes de mejor calidad. Los puntos más relevantes que se pueden extraer de [7] son:

- En el discriminador, utilizar convoluciones con stride en lugar de capas de pooling.
- En el generador, utilizar convoluciones fraccionales con stride en lugar de capas de pooling.
- Utilizar BatchNorm tanto para generador como discriminador.
- No utilizar capas fully connected.
- Utilizar funciones de activación ReLU en el generador, salvo para la última capa, en la que se propone usar tanh.
- Utilizar funciones de activación LeakyReLU en el discriminador.
- Inicializar los pesos de ambas redes con una distribución normal.

RECORDAR QUE STRIDE Y POOL SON PARECIDOS Y DESCRIBIR LEAKYRELU.

El objetivo que nos hemos propuesto es la generación de cuadros realistas, para ello, es necesario un amplio conjunto de datos con cuadros de distintos artístas, épocas y estilos. Hemos encontrado en una competición de Kaggle un conjunto de datos con las características deseadas, con más de 100000 imágenes, ocupando aproximadamente 49 GB. Una muestra de dichas imágenes se puede observar en la Figura 4.1

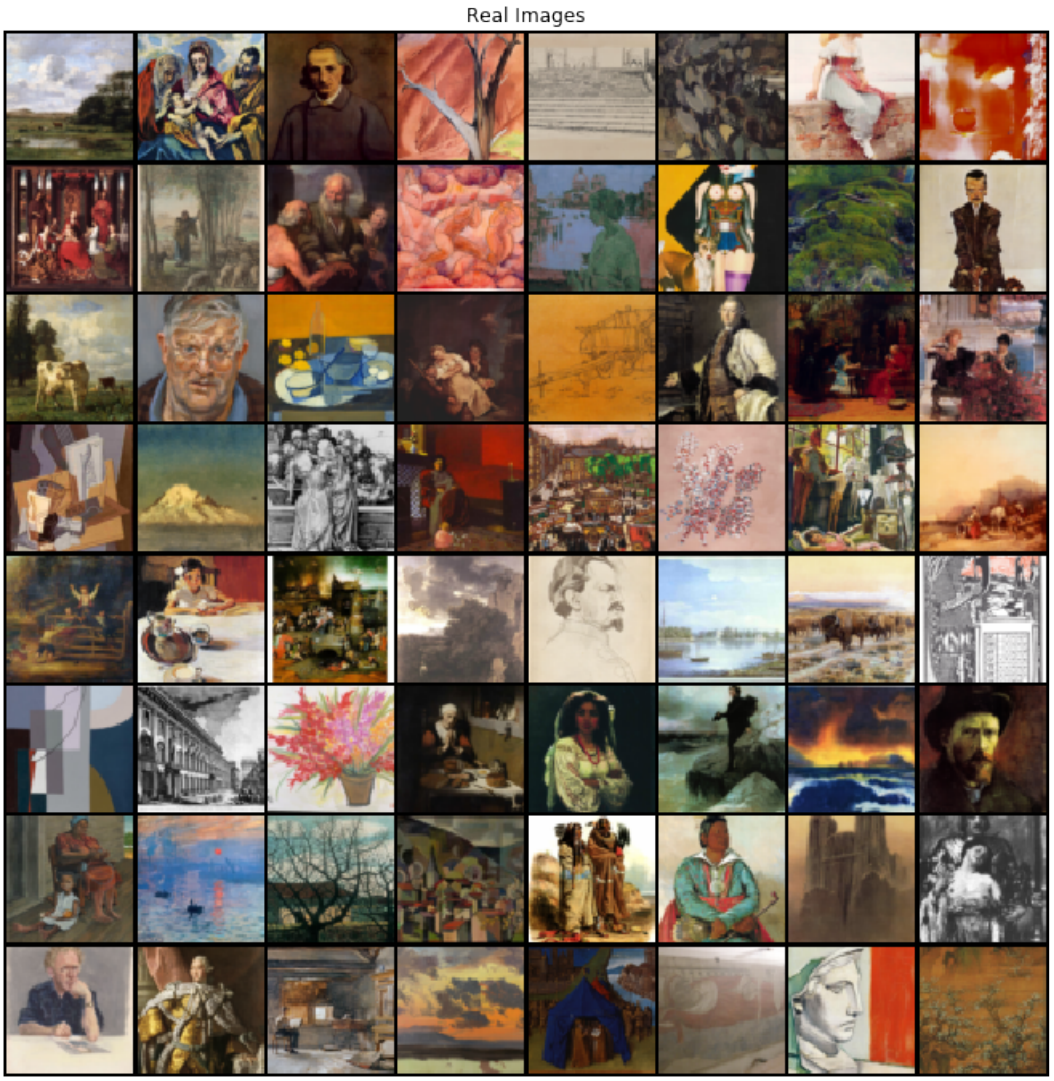

Figura 4.1: Imágenes del conjunto de datos.

4.1.1. Preprocesado

El conjunto de datos tenía algunos defectos, como por ejemplo imágenes descargadas incorrectamente o imágenes corruptas en el propio conjunto de datos. Se ha tomado la decisión de eliminar dichas imágenes. ya que son pocas y no hay una manera directa de recuperarlas.

Una vez limpio el conjunto de datos, es necesario realizar algunas transformaciones para que, posteriormente, nuestras redes neuronales puedan utilizar las imágenes. En primer lugar hemos homogeneizado los tamaños y las proporciones. Pasamos 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.

4.1.2. Arquitectura

Siguiendo las recomendaciones del artículo [7], hemos decidido utilizar la arquitectura que se muestra en la Figura 4.1.2. También hemos inicializado

los pesos con una distribución normal $\mathcal{N}()$ y hemos seleccionado un tamaño de batch de 128. La red se ha entrenado durante 30 épocas, utilizando el algoritmo de optimización Adam [6] con una tasa de aprendizaje de 0,0002 y $\beta = (0{,}5, 0{,}999)$.

4.1.3. Recursos

Cabe destacar que ha sido imprescindible el uso de un ordenador con GPU compatible con CUDA, en concreto se ha utilizado una Nvidia Quadro P5000. El entrenamiento de principio a fin ha tardado aproximadamente 24 horas. También se han realizado pruebas con CPU en un portátil de prestaciones modestas y se ha observado que el rendimiento era unas 20 veces menor.

4.2. Resultados

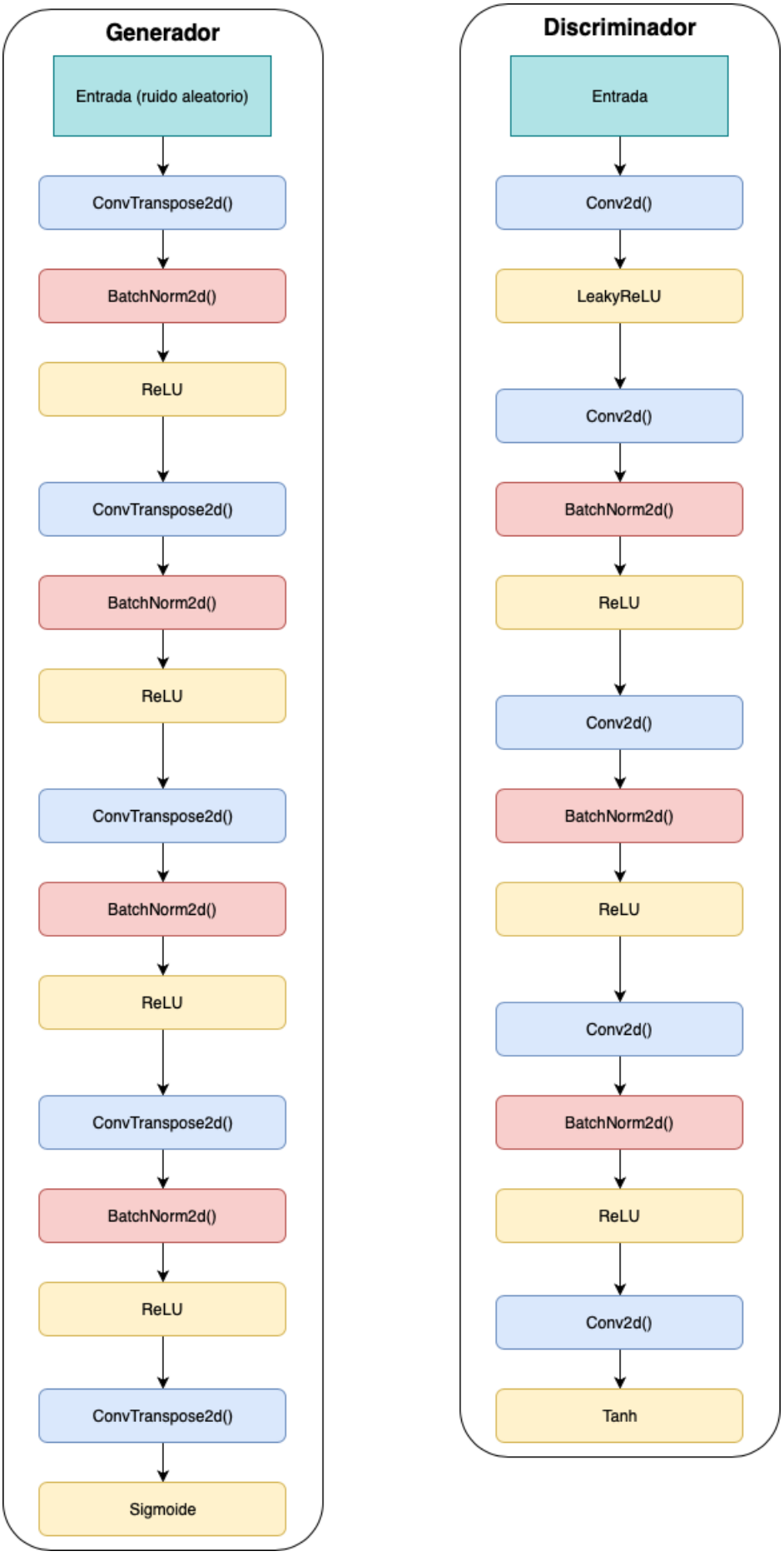

Figura 4.2: Arquitectura de las redes (Cambiar, poner tamaños de entrada y salida).

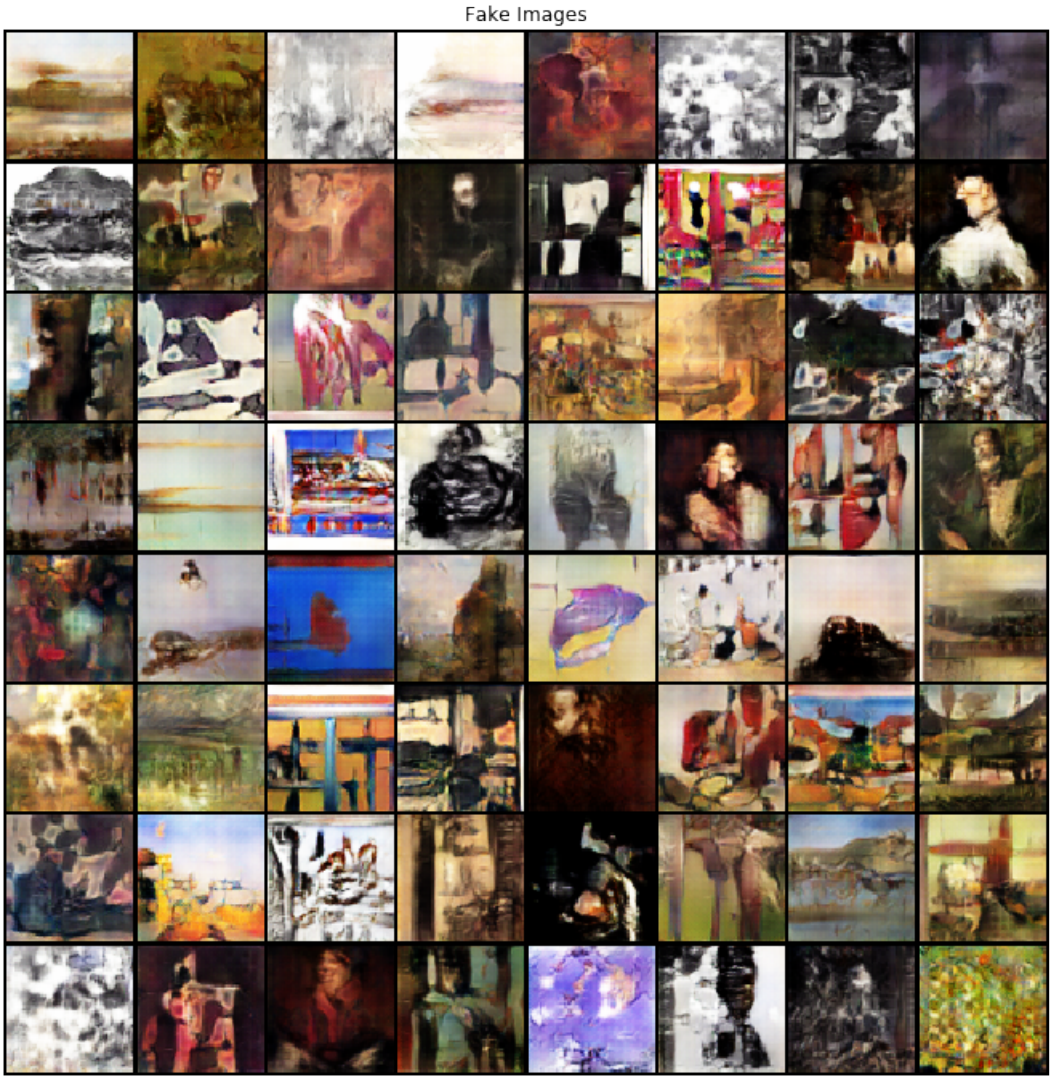

Figura 4.3: Imágenes generadas mediante la DCGAN implementada después de 30 épocas de entrenamiento.

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