



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Grado en Ingeniería Informática (10II)

Trabajo Fin de Grado

**Desarrollo de una Red Generativa
Antagónica para la Generación de
Imágenes de Flores Falsas**

Autor: Sergio Donís Ebri
Tutor: Daniel Manrique Gamo

Madrid, Junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería informática

Título: Desarrollo de una Red Generativa Antagónica para la Generación de Imágenes de Flores Falsas

Junio 2022

Autor: Sergio Donís Ebri

Tutor:

Daniel Manrique Gamo

Departamento de Inteligencia Artificial

ETSI Informáticos

Universidad Politécnica de Madrid

Agradecimientos

Lo primero, me gustaría agradecer a Daniel, mi tutor en este trabajo, por su orientación, dedicación y ayuda en la realización de este.

En segundo lugar, agradecerle también a mis compañeros y amigos de la carrera, porque sin ellos estos cuatro años de universidad no hubiesen sido lo mismo.

Por último, quiero agradecerle especialmente a mi familia por creer en mí en los momentos difíciles y en particular a Pablo, por su apoyo recibido. También a mi abuela Remedios.

Resumen

Uno de los ámbitos estudiados hoy en día por la Inteligencia Artificial es la generación de imágenes. Dos de las técnicas más importantes, dentro del aprendizaje profundo, son los autocodificadores variacionales (variational autoencoders, VAE) y las redes generativas antagónicas (generative adversarial networks, GAN). La arquitectura de estas últimas consiste en dos redes de neuronas, una generadora y otra discriminadora, puestas a competir entre ellas. La red generadora aprende a producir datos que se asemejan al conjunto de entrenamiento, mientras que la segunda debe aprender a distinguir entre si las muestras son reales (del conjunto de datos) o falsas (imágenes producidas por el generador).

Dentro de las GAN, existen las redes generativas antagónicas convolucionales, llamadas DCGAN. Utilizan filtros para extraer patrones cuando el conjunto de datos tiene 2 o más dimensiones, por lo que son las más adecuadas cuando lo que se quiere generar son imágenes o representaciones de objetos en tres dimensiones.

El objetivo de este trabajo es la implementación de una DCGAN para la generación de imágenes falsas de flores a partir de una entrada aleatoria distribuida de forma normal. Para el entrenamiento de la red, se emplea un repositorio público con imágenes de flores. El desarrollo de este trabajo consiste en 4 fases: adquisición de los conocimientos necesarios de programación en Python con Tensorflow y Keras así como el entendimiento de las tecnologías subyacentes a las DCGAN, diseño de la red de neuronas profunda, implementación y pruebas.

Los resultados obtenidos son favorables, dado a que en el entrenamiento de la DCGAN se ha conseguido llegar a un punto de equilibrio entre las dos redes. Las imágenes producidas por el generador se asemejan a imágenes de flores reales, comprobando los resultados obtenidos mediante Google Lens.

La principal aplicación de esta red es generar nuevas imágenes de flores para producir conjuntos de datos con el propósito de entrenar otras redes neuronales. También las imágenes generadas pueden ser empleadas para el diseño, o como inspiración para cuadros, ropa o videojuegos en el que se requieran nuevos diseños de flores no existentes o una gran cantidad de flores diferentes.

Abstract

One of the areas currently being studied by Artificial Intelligence (AI) is image generation. Two of the most important techniques in deep learning are variational autoencoders (VAE) and generative adversarial networks (GAN). The architecture of the GAN consists of two networks of neurons, one generative and the other discriminative, set to compete with each other. The generator learns to produce data that resembles the training set, while the discriminator must learn to distinguish whether the samples are real (from the data set) or fake (images produced by the generator).

A subset of GANs are deep convolutional adversarial generative networks (DCGAN). They use filters to extract patterns when the dataset has 2 or more dimensions. They are most suitable when generating images or representations of objects in three dimensions.

The objective of this work is the implementation of a DCGAN for the generation of fake images of flowers from a normally distributed random input. For the training of the network, a public repository with images of flowers is used. The development of this project consists of 4 phases: acquisition of the necessary knowledge of DCGANs and Python's APIs (Tensorflow, Keras), designing the deep neural network, the implementation and finally, testing.

The results obtained are favorable, given that in the training of the DCGAN it has been possible to reach a point of equilibrium between the two networks. The images produced by the generator resemble real flowers, which is verified by using Google Lens.

The main application of this network is to generate new images of flowers to produce data sets for the purpose of training other neural networks. The generated images can also be used for design, or as inspiration for paintings, clothing, films or video games where new designs of non-existing flowers or a large number of different flowers are required.

Tabla de contenidos

1	Introducción	1
2	Redes de neuronas artificiales.....	3
2.1	Introducción	3
2.2	La neurona artificial.....	4
2.3	Redes de neuronas multicapa.....	7
2.4	Algoritmo de aprendizaje de retropropagación del gradiente del error	7
2.5	Redes neuronales convolucionales.....	8
2.5.1	Elementos de las redes convolucionales	8
2.6	Redes generativas antagónicas (GAN)	12
2.6.1	Arquitectura de las GAN	12
2.6.2	Entrenamiento de las GAN.....	14
2.6.3	Dificultades del entrenamiento de las GAN.....	14
2.7	Generación de imágenes.....	14
3	Planteamiento del problema.....	18
4	Solución Propuesta	20
4.1	Conjunto de datos	20
4.2	Preprocesado de datos.....	21
4.3	Arquitectura de la red	22
4.4	Búsqueda de hiperparámetros.....	25
5	Resultados	28
5.1	Análisis del entrenamiento	28
5.2	Análisis de las imágenes generadas	29
5.3	Prueba de clasificación mediante Google Lens	36
6	Conclusiones y líneas futuras.....	40
6.1	Conclusiones.....	40
6.2	Lineas Futuras.....	40
7	Análisis de Impacto.....	42
8	Bibliografía.....	43

1 Introducción

Actualmente, la inteligencia artificial (IA) se encuentra completamente integrada en nuestra vida. Desde el corrector de palabras en el teclado de los teléfonos inteligentes, hasta en el correo electrónico, pasando por otras aplicaciones cotidianas como los bancos, videojuegos, realizar búsquedas en Internet o el GPS. El término de inteligencia artificial es muy amplio, ya que se usa cuando sistemas informáticos intentan replicar un comportamiento humano para resolver una tarea que se considera difícil [1].

Uno de los ámbitos dentro de la inteligencia artificial es el aprendizaje automático, en el que hay un modelo capaz de modificar su comportamiento y mejorarlo (aprender) a partir de un conjunto de datos de entrenamiento. Ejemplos de sistemas de aprendizaje automático son los filtros de *spam*, o el contenido que aparece como recomendados en las redes sociales. También es un término relativamente amplio, ya que es usado para cualquier sistema informático que sea capaz de aprender de forma autónoma mediante la modificación de un conjunto de parámetros. Recientemente ha habido un gran auge en este campo, gracias a las redes neuronales y el aprendizaje profundo [2].

Las redes neuronales son un tipo de modelo de aprendizaje automático consistente en pequeñas unidades de procesamiento que se combinan entre sí para producir sistemas de procesamiento más grandes y sofisticados. El conjunto de métodos de aprendizaje de sistemas inteligentes que combinan varias capas de abstracción de procesamiento de los datos de entrada se conoce como aprendizaje profundo. Los asistentes de voz de los móviles inteligentes, los coches autónomos o el reconocimiento de rostros, se entrena mediante estas técnicas [2].

Dentro del aprendizaje profundo, hay sistemas generadores como los autocodificadores variacionales (VAE) o las redes generativas antagónicas (GAN). Las GAN han sido clasificadas por Yann Lecun, científico jefe de inteligencia artificial en Meta (Facebook), como “la idea más interesante en aprendizaje automático de los últimos diez años” [3].

Tanto los VAE como las GAN se utilizan normalmente para aumentar el conjunto de datos (*data augmentation*) con el propósito de entrenar redes neuronales profundas cuando el conjunto de datos de entrenamiento es insuficiente. Cuando los datos que se quieren generar son imágenes u objetos en tres dimensiones, se suelen utilizar las redes generativas antagónicas profundas convolucionales (DCGAN). Esto es debido a que utilizan filtros para reconocer patrones en dos o más dimensiones. Otras de las posibles aplicaciones de las DCGAN es la de aumentar la resolución de imágenes, transferir imágenes de un estilo a otro, traducir texto a imágenes o la edición de fotografías [4][5].

El principal objetivo de este trabajo de fin de grado es el de la creación de una DCGAN con el propósito de generar imágenes de flores lo más parecidas a un conjunto de datos de entrenamiento y que un clasificador externo, en este caso Google Lens, sea capaz de reconocer las imágenes como fotografías de flores. El objetivo principal de este trabajo se ha dividido en los siguientes subobjetivos:

1. Encontrar un repositorio público de imágenes de flores a color. El conjunto de datos se considera adecuado si tiene una cantidad de imágenes suficiente para poder entrenar adecuadamente la arquitectura DCGAN y que sea capaz de generar una variedad de imágenes diferentes. Además, las imágenes del repositorio deben tener diferentes categorías para que el modelo aprenda diferentes tipos de flores. También las imágenes tienen que ser lo suficientemente similares para que la DCGAN sea capaz de aprender los patrones más comunes de las flores. Dicho de manera metafórica, que la red generadora aprenda lo que es un ramo de flores.
2. Las imágenes deben tener un formato correcto para ser la entrada de las redes de neuronas y que estas puedan ser capaces de entrenar. Para ello, es necesario realizar una preparación y limpieza de las imágenes de flores a utilizar, modificando la resolución de estas y normalizando los valores de los píxeles.
3. Adquisición de conocimientos de aprendizaje profundo, concretamente sobre las DCGAN, así como la programación en Python con Tensorflow y Keras.
4. Diseño, implementación en Python y entrenamiento de una arquitectura DCGAN en el entorno de ejecución Google Colab con el modo de ejecución de GPU.
5. Evaluación del funcionamiento y resultados de la red generativa antagónica, incluyendo la realización de experimentos con Google Lens para comprobar cuántas de las imágenes generadas las clasifica como imágenes de flores o es capaz de relacionarlas con flores.

La estructura del documento de este trabajo fin de grado es la siguiente:

- El capítulo 2 describe los algoritmos y tecnologías relacionadas con el aprendizaje profundo, más en concreto las DCGAN.
- El capítulo 3 detalla los objetivos del TFG y plantea el problema a resolver de este trabajo fin de grado.
- El capítulo 4 indica la solución propuesta para resolver el problema planteado en el capítulo anterior.
- El capítulo 5 analiza los resultados obtenidos por el modelo propuesto.
- El capítulo 6 expone las conclusiones del trabajo desarrollado y propone futuras líneas a seguir.
- El capítulo 7 recoge el análisis de impacto del proyecto, tanto a nivel personal, como social y económico.

2 Redes de neuronas artificiales

2.1 Introducción

Convencionalmente, la manera de actuar de los ordenadores es mediante algoritmos. Esto significa que, para resolver una tarea, debe ser programable. El ordenador ejecuta instrucciones paso por paso, aplicando una serie de reglas previamente programadas hasta llegar al resultado deseado. Su principal inconveniente es que se necesita saber previamente todas las posibles situaciones que se puedan dar [6].

Por eso, el aprendizaje automático supone un paso adelante en la forma en la que los ordenadores son capaces de resolver problemas. Las redes de neuronas artificiales forman parte de esta rama de la inteligencia artificial.

El aprendizaje automático, también conocido como *machine learning* es una rama de la inteligencia artificial en la que el ordenador aprende a través de conjuntos de datos a resolver tareas específicas, extrayendo patrones de los datos. Pero no solo para resolver tareas con el conjunto de datos con el que han sido entrenadas, sino a devolver salidas adecuadas ante datos que el algoritmo no haya visto previamente [7]. Por esa razón, son importantes los datos con los que se entrena un sistema de aprendizaje automático: si hay algún problema con los datos, el sistema inteligente no aprende correctamente [6].

Los algoritmos de aprendizaje automático se pueden dividir en supervisado, no supervisado, semi-supervisado y aprendizaje por refuerzo (*reinforcement learning*).

Los algoritmos de aprendizaje automático supervisado necesitan el valor de datos de entrada y la salida deseada para esos datos. Estos algoritmos aprenden el proceso de cómo a partir de esas entradas generar esas salidas y construyen un modelo capaz de ello. Las tareas más comunes para el aprendizaje automático supervisado son la clasificación y la regresión [8,2].

En el aprendizaje automático no supervisado, se tiene solo el conjunto de datos de entrada, pero no las salidas que se quieren producir gracias a ese conjunto de datos. Los datos tampoco están clasificados. Gracias a estos algoritmos se encuentran relaciones y patrones en el conjunto de datos. En este tipo de aprendizaje, las tareas más comunes son el agrupamiento (*clustering*), la detección de anomalías, la estimación de densidad y la reducción de dimensiones del conjunto de datos [8].

El aprendizaje semi-supervisado consiste en aplicar técnicas tanto de aprendizaje supervisado como no supervisado. En el aprendizaje semi-supervisado, la menor parte de los datos están etiquetados, mientras que el resto no. Primero se usa aprendizaje no supervisado para etiquetar los datos que faltan por clasificar. Después se aplican técnicas de aprendizaje supervisado para entrenar modelos [7].

El aprendizaje por refuerzo se da cuando un agente tiene que aprender cómo comportarse en un entorno dinámico interactuando con él mediante prueba y error. Como otros tipos de aprendizaje automático, esta basado en comportamientos biológicos y psicológicos, más en específico en el modo de aprendizaje de los animales. La diferencia está en cómo se produce este refuerzo

necesario para el aprendizaje. Hay dos modos en el que el agente inteligente puede interactuar con el ambiente, mediante percepción y mediante acción. Primero el agente recibe información del estado actual del entorno y después ejecuta una acción, que a su vez cambia el estado del entorno y recibe una respuesta conocida como señal de respuesta. Según el entorno, el agente se encuentra en un estado o en otro, y en cada uno puede ejecutar un número limitado de acciones, que le producen una señal de respuesta y le llevan a otro estado. El agente aprende, según el estado en el que esté, a tomar la acción que le produzca la mayor señal de respuesta positiva. Los estados del robot y la información del entorno se pueden ver como una cadena de Markov [10].

2.2 La neurona artificial

“Las redes de neuronas artificiales son un sistema de computación constituido por un gran número de elementos simples de procesamiento interconectados, que procesan información por medio de su estado dinámico como respuesta a entradas externas” [11].

Las redes neuronales artificiales están compuestas por neuronas individuales. Una neurona es una unidad de procesamiento simple: recibe datos, los agrupa, los computa y, a partir de ellos, genera una salida. Cuando la red está compuesta por una sola neurona, la red es capaz de realizar tareas simples, como por ejemplo un clasificador lineal. Pero también las neuronas se agrupan en capas y varias capas forman una red de neuronas, capaces de realizar tareas más complejas. Cuando las redes tienen muchas capas con múltiples neuronas cada una, se las conoce como redes de neuronas profundas y las técnicas empleadas para su aprendizaje se denominan aprendizaje profundo o *deep learning* [2].

Cada neurona está compuesta por entradas, salida, desplazamiento (*bias*), entrada neta y función de activación; interactuando entre sí como se puede apreciar en la Figura 1[7].

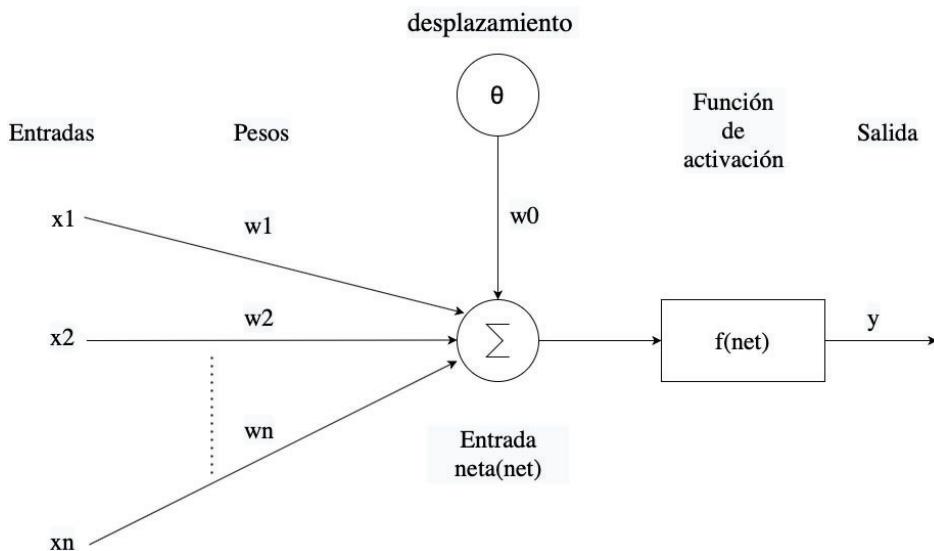


Figura 1, Neurona Artificial

Las entradas (x_1, x_2, \dots, x_n) son los valores con los que se alimenta la neurona. Estos valores pueden provenir de un elemento del conjunto de datos. Por ejemplo, cada entrada podría ser un píxel de una imagen. Pero cada elemento necesita estar en forma de vector unidimensional, por lo que puede ser necesario preprocessar los datos antes de usarlos. El valor de cada entrada puede variar dependiendo de la neurona, siendo los rangos más comunes $[-1,1]$ y $[0,1]$.

Los pesos (w_1, w_2, \dots, w_n), son valores numéricos que especifican la importancia del dato que han recibido a través de la entrada. Un valor cercano a cero significa que casi no tiene relevancia. Cuanto más alejado sea de cero, tanto positivo o negativo, quiere decir que esa conexión tiene un gran peso en la neurona o red. Se multiplica cada entrada con el peso que tiene asociado.

El sumatorio de las entradas multiplicadas por sus respectivos pesos constituyen la entrada neta total a la neurona. Pero debido a que este sumatorio es la ecuación de una recta o hiperplano que pasa por el centro del eje de coordenadas, se necesita más flexibilidad. Si no, cuando todos los valores de entrada fuesen cero, el valor del sumatorio sería cero y no se tiene por qué desear eso. Por esa razón, se añade un peso adicional conocido como desplazamiento (*bias*). Su entrada es siempre 1 y lo que cambia es el peso que va asociada a ella [7].

Una vez calculada la entrada neta a una neurona, es necesario transformarlo mediante una función denominada función de activación para que el resultado de la red no sea una combinación lineal de las entradas por sus respectivos pesos y puedan existir relaciones más complejas. También se eligen funciones de activación que tengan una derivada parcial simple, para facilitar el entrenamiento de la red. Las funciones de activación más usadas se pueden ver en la Figura 2:

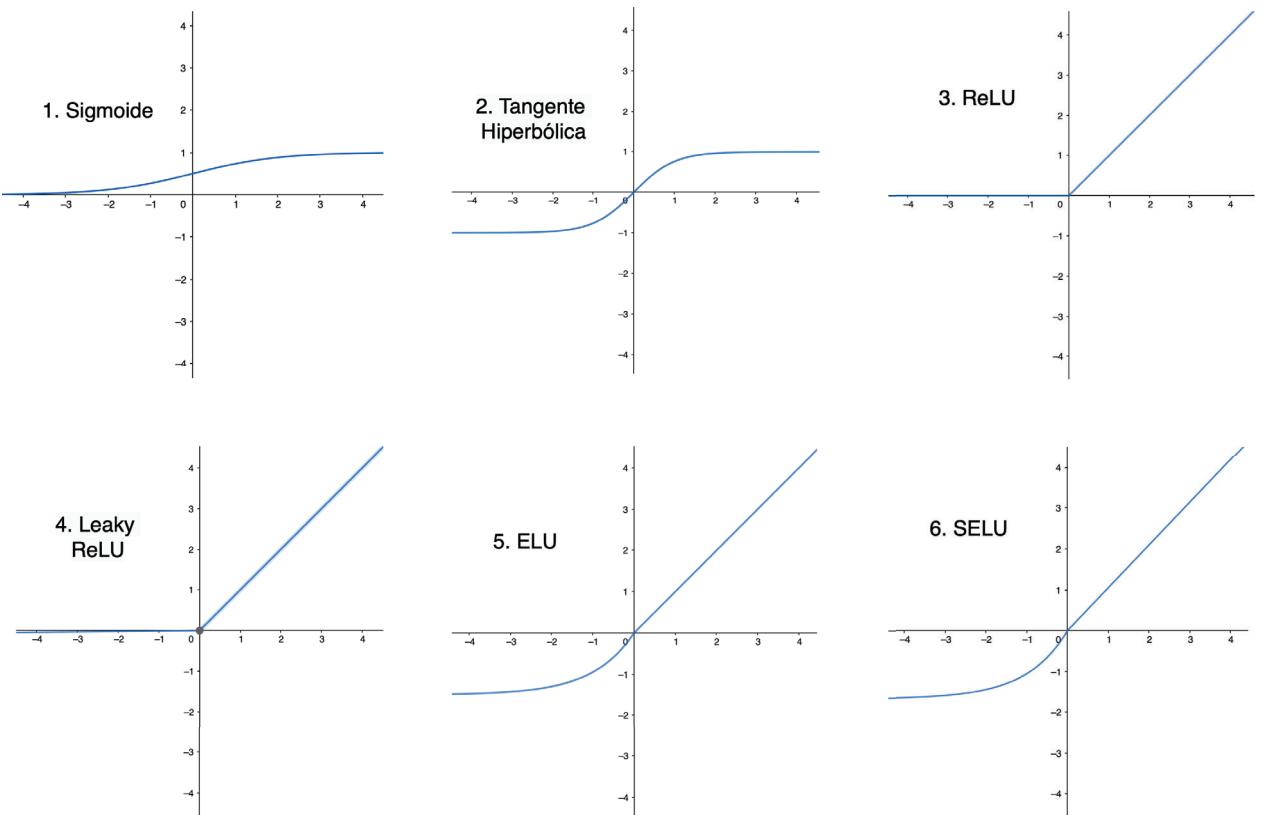


Figura 2, Funciones de Activación

Función Sigmoid (1): $f(x) = 1 / (1 + e^{-x})$

Transforma los valores de entrada a un intervalo $[0, 1]$. Su centro no es 0, ya que para $x=0$, la salida de la función es 0.5.

Función Tangente hiperbólica (2): $f(x) = (e^x - e^{-x}) / (e^x + e^{-x})$

Transforma los valores de entrada a un intervalo $[-1, 1]$.

Función ReLU (3): si $x < 0 \rightarrow f(x) = 0$
 si $x \geq 0 \rightarrow f(x) = x$

Transforma los valores negativos a 0 y los positivos los deja igual.

Función Leaky ReLU (4): si $x < 0 \rightarrow f(x) = a * x$
 si $x \geq 0 \rightarrow f(x) = x$

Se multiplican los valores negativos por un valor (parámetro a variable elegido por el usuario de la red), menor que uno, acercando los valores a 0 y los números positivos los deja igual.

Función ELU (5): si $x < 0$, $f(x) = a * (e^x - 1)$
 si $x \geq 0$, $f(x) = x$

'a' es un parámetro ajustable que se encarga de controlar los valores negativos de la x.

Función SELU (6): si $x < 0$, $f(x) = a * (e^x - 1)$
 si $x \geq 0$, $f(x) = \lambda * x$

Con valores típicos $a = 1,6732632423544$ y $\lambda = 1,0507009873555$

La salida y es el resultado de la función de activación aplicada sobre la entrada neta. Dependiendo de la función elegida, los valores pueden estar entre $[-1, 1]$,

$[0,1]$, $[-\infty, +\infty]$ o el rango deseado. Cada neurona tiene exclusivamente una salida.

2.3 Redes de neuronas multicapa

Las redes de una sola capa de neuronas son capaces de encontrar patrones simples, pero normalmente es necesario encontrar patrones más complejos. Las neuronas se agrupan en diferentes capas según se puede ver en la Figura 3. Esta red neuronal consiste en tres elementos de entrada, dos capas de 3 neuronas, y una capa de salida compuesta por una neurona.

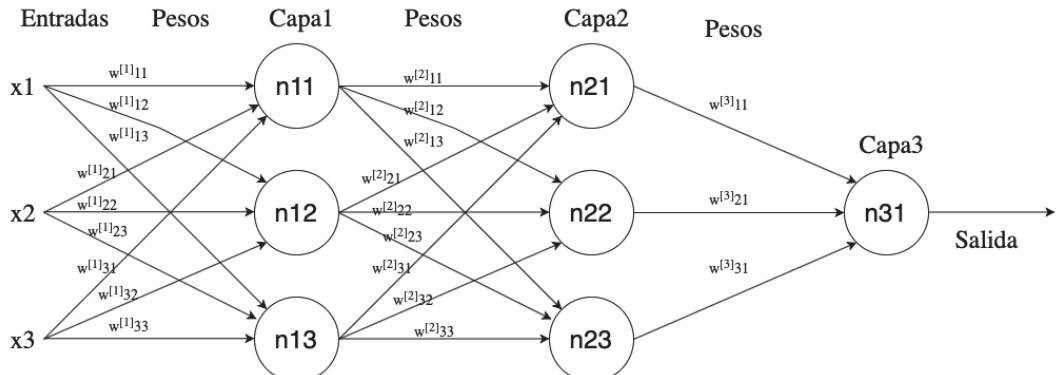


Figura 3, Red de Neuronas de 3 Entradas, 1 Salida y 3 Capas

Una capa de neuronas es una agrupación de neuronas que tienen los mismos datos de entrada. Cada neurona puede ser capaz de encontrar un patrón distinto al resto de neuronas de la misma capa. Las neuronas de una capa solo se pueden conectar con neuronas de la capa anterior y posterior.

Las entradas de cada capa son las salidas de la capa anterior, y la salida de la capa es la entrada de la posterior. Menos en la primera y última. La entrada de la primera son los datos del conjunto de datos. La salida de la última capa es la salida de la red.

Todas las neuronas de la misma capa tienen la misma función de activación, pero cada capa puede tener una función de activación diferente. Lo más común es que la salida de la red sea la que tiene la función de activación diferente, dependiendo del tipo de problema que se quiera resolver.

La salida puede ser un solo valor numérico o puede ser un vector, dependiendo de la salida deseada. Si la red es un clasificador binario, por ejemplo, solo tendrá una salida que será lo segura que está la red de que el elemento de la entrada pertenezca a esa clasificación. Si la salida de la red es una imagen, cada salida será un píxel de la imagen final.

2.4 Algoritmo de aprendizaje de retropropagación del gradiente del error

Una vez los datos han entrado en la primera capa de la red, van pasando por el resto de las capas, siendo transformados, hasta llegar a la última y se produce la salida de la red. Esta salida se compara con la salida deseada (en el caso de aprendizaje automático supervisado) y se calcula la diferencia entre estos dos términos, lo que se conoce como el error. El siguiente paso es ajustar los pesos

para disminuir el error. Un inconveniente es que se necesita saber de antemano cómo va a afectar a la salida modificar un peso de la red y si va a haber un aumento o una disminución del error. Para ello, se utiliza el gradiente, que es la derivada parcial del error con respecto al conjunto de parámetros o pesos. El gradiente nos indica hacia donde modificar un peso para minimizar el error.

Se empieza calculando el gradiente de la última capa y se va hacia atrás, hacia las primeras capas. Esto es lo que se conoce como retropropagación del gradiente. Posteriormente, se ajustan los pesos, dependiendo de un término conocido como tasa de aprendizaje (*learning rate*).

2.5 Redes neuronales convolucionales

Las redes convolucionales nacieron en los años 80. Al igual que el resto, las redes neuronales convolucionales están basadas en las neuronas biológicas, pero en este caso en las del córtex visual [8].

Las redes neuronales convolucionales son uno de los tipos de redes neuronales más comunes hoy en día y son usadas, principalmente, para el reconocimiento de formas y clasificación de imágenes.

A diferencia de las redes convencionales, donde todas las neuronas de una capa están conectadas a todas las neuronas de la capa anterior y posterior, las neuronas de una red convolucional tienen sólo unas pocas conexiones con las neuronas de la capa anterior. Otra principal diferencia es que las capas de estas redes no son unidimensionales, si no que pueden tener dos o más dimensiones, siendo las más comunes las de dos cuando los datos representan imágenes y tres cuando los datos modelan objetos tridimensionales [12].

2.5.1 Elementos de las redes convolucionales

La arquitectura típica de una red convolucional es una mezcla de capas convolucionales y capas de reducción. Entre capa y capa convolucional, se suelen colocar capas que hagan reducción (*pooling*). La última capa de las redes de neuronas convolucionales suele ser una capa típica, no convolucional. La Figura 4 muestra un ejemplo de una arquitectura típica de redes convolucionales, más en concreto de un clasificador de imágenes. Esta red neuronal esta compuesta por dos capas de convolución y reducción sucesivamente y después por una capa de m neuronas. Por ultimo, la capa de salida que es una sola neurona.

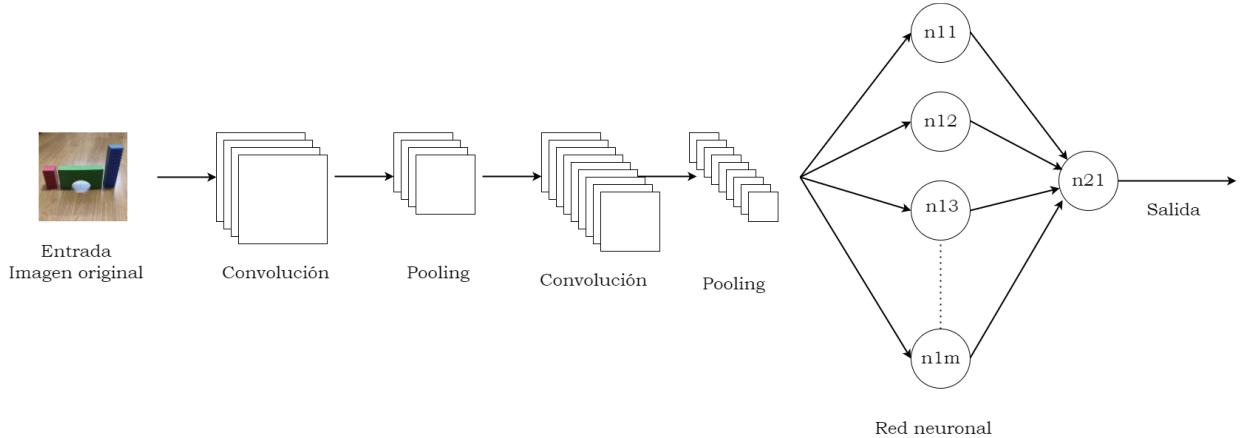
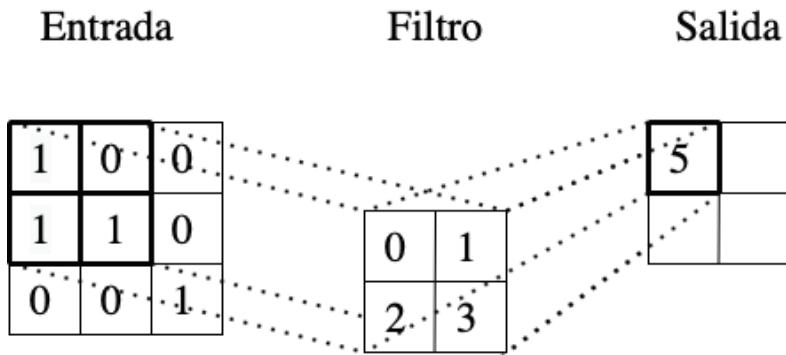


Figura 4, Arquitectura Red Convolucional

Para el caso de las redes convolucionales de dos dimensiones, cada neurona solamente está conectada con un pequeño rectángulo de neuronas de la capa anterior. Como en las redes de neuronas típicas, cada valor nuevo se calcula mediante una suma ponderada. Pero en vez de con todos los elementos de la capa anterior, se coloca una matriz de menor tamaño, conocida como filtro, sobre la imagen anterior y se realiza un producto escalar. Como se muestra en la Figura 5, el filtro empieza al inicio de la matriz, se realiza el producto escalar y el resultado se guarda en otra matriz.



$$1*0 + 0*1 + 1*2 + 1*3 = 5$$

Figura 5, Filtro Convolucional

El filtro se desplaza y se realizan los productos escalares hasta que se obtiene la matriz de salida. Esto hace que la red se pueda centrar en pequeños detalles o patrones de la capa previa y al juntarlos, se pueden reconocer patrones más grandes para la siguiente capa. De una capa a la siguiente, se pueden aplicar uno o más filtros, produciendo múltiples matrices de salida. En las redes convolucionales 2D, el filtro tiene que ser rectangular, no necesariamente cuadrado, mientras que en las de tres dimensiones, los filtros tienen forma de ortoedro.

Pero como podemos apreciar en la Figura 5, si se realiza la convolución de este modo, la matriz de salida es más pequeña que la de entrada, por lo que para solucionarlo se utiliza una técnica conocida como *zero padding*. Esta técnica consiste en aumentar la dimensión de la matriz de entrada añadiendo ceros en el exterior de la matriz, para que, al aplicar la convolución, la matriz de salida tenga las mismas dimensiones que la de entrada. En la Figura 6 se puede ver

una matriz 6x6 a la que se le ha aplicado *zero padding*, y el resultado es una matriz 8x8.

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

Figura 6, Matriz con Zero Padding

A veces puede ser necesario reducir la dimensión de matriz de salida para que, al acumular capas y filtros, la computación no sea tan costosa y no se requiera de tanto almacenamiento. Esto se puede conseguir moviendo el filtro, en vez de solo una posición cada vez, desplazándolo dos o más posiciones, lo que se conoce como desplazamiento o *stride*. En la figura 7 se puede observar una matriz de 6x6 elementos con *zero padding*, lo que la convierte en una matriz de 8x8, en la que el filtro se ha movido en dos posiciones (*stride* de 2).

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1
0	0	1	0	0	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

Figura 7, Capa con Zero Padding y Stride de 2

La técnica más utilizada para reducir la dimensión de una matriz es aplicarle una capa de reducción (*pooling*). Su funcionamiento es similar al de las capas convolucionales, pero en vez de hacer el producto escalar, se aplica una función de agregación que puede ser la media o el máximo, siendo esta última la más usada ya que preserva mejor los datos de la matriz original. Este tipo de capas se suelen usar después de capas convolucionales [8].

En la Figura 8, se puede apreciar como una matriz de 4x4 pasa a ser de 2x2 después de aplicarle una reducción solo conservando el máximo valor de cada sección (*max pooling*)

1	2	3	4
5	6	7	8
1	1	3	9
5	2	0	3

6	8
5	9

Figura 8, Reducción Mediante *Max Pooling*

También es importante tener en cuenta cuántos filtros se utilizan Dependiendo de si se está trabajando con imágenes a color o en blanco y negro. Por ejemplo, para imágenes a color, la última capa necesita 3 filtros de salida para que se produzcan 3 imágenes, Cada una se asocia a un canal de color: rojo, verde y azul; también conocido como canales RGB (*red, green, blue*).

En la Figura 9 se puede apreciar como la suma de los tres canales RGB, forma una imagen a color, mientras que la Figura 10 contiene la misma imagen, pero en blanco y negro.

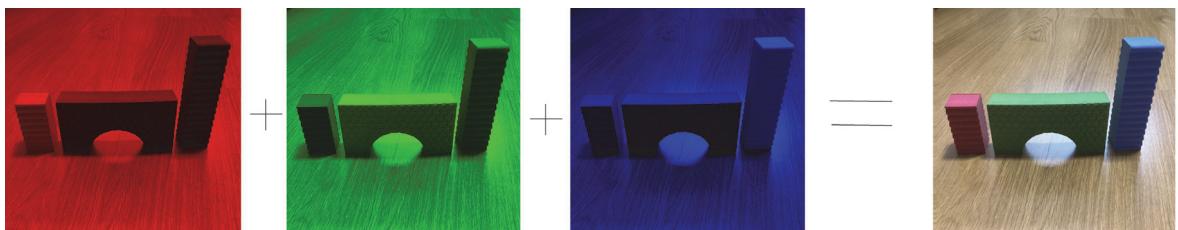


Figura 9, Imagen a Color y sus Canales RGB

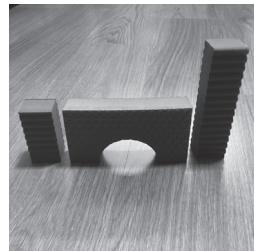


Figura 10, Imagen en Blanco y Negro

El tamaño del filtro en las redes convolucionales no tiene por qué ser cuadrado, pudiendo también ser de diferentes tamaños. Siendo lo más común 3x3, 5x5 y 7x7. Si los filtros son pequeños, son capaces de detectar pequeños patrones de las imágenes y al ir pasando los datos por las diferentes capas, se juntan los detalles pequeños en patrones más grandes. Las redes de neuronas convolucionales, por su modo de funcionar, poseen invarianza a pequeños cambios, ya sea por rotación, desplazamiento o cambio de tamaño.

Las redes convolucionales transpuestas, por su parte aumentan el tamaño de las capas, pasando, por ejemplo, de una imagen de 10x10 a una 20x20 [13].

2.6 Redes generativas antagónicas (GAN)

Las redes generativas antagónicas están compuestas por dos redes diferentes compitiendo entre sí. Son una red generadora y una discriminadora. Como sus propios nombres indican, el generador se encarga de producir nuevos datos que sean parecidos al conjunto de datos con el que las redes están siendo entrenadas y el discriminador se encarga de decir si los datos que le llegan son del *conjunto de datos* o han sido creados por el generador.

Al final el generador y el discriminador están jugando a un juego de suma cero, donde lo que uno gana, lo pierde el otro. Si el discriminador tiene poco error y clasifica los datos generados bien, significa que el generador no lo está haciendo correctamente y tendrá mucho error. También en el otro sentido, si el generador tiene poca pérdida, el discriminador tendrá mucha. Al ser dos tipos de redes trabajando juntas, es bastante importante cómo se entrena [4].

El uso típico de las GAN es para generar datos parecidos a los datos de entrada. Bien porque se desea aumentar el tamaño del conjunto de datos original ya existente para entrenar otra red neuronal o bien porque simplemente se quieren generar nuevos datos.

2.6.1 Arquitectura de las GAN

En la Figura 11 se puede ver como se relacionan el generador y el discriminador de una GAN de imágenes. El discriminador recibe imágenes que pueden ser del conjunto de datos o del generador y su misión es determinar cuál es la procedencia de la imagen recibida. Por su parte el generador se encarga de transformar valores aleatorios en imágenes que el discriminador piense que son del conjunto de datos.

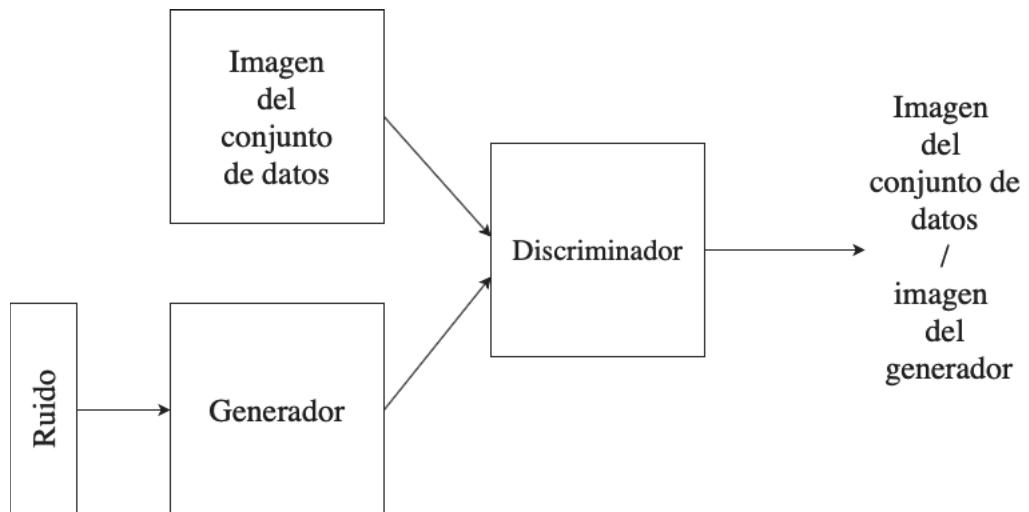


Figura 11, Red Generativa Antagónica de Imágenes

El uso de las GAN es generalizado, no específico para un tipo de datos en concreto. La Figura 12 muestra como es una GAN compuesta por un generador de dos capas y un discriminador también de dos capas. Cada capa del generador va siendo progresivamente mas grande que la anterior, mientras que con el discriminador cada capa va siendo mas pequeña hasta que la última capa esta compuesta por una neurona.

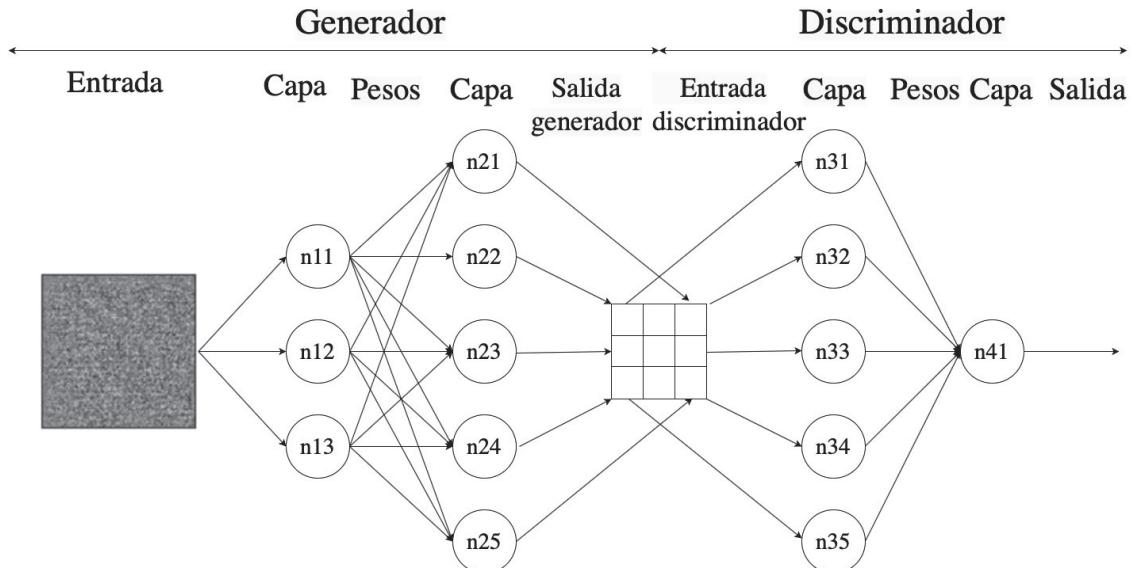


Figura 12, Red Generativa Antagónica para la Generación de Imágenes

La entrada del generador es una distribución aleatoria de tipo gaussiano. En la Figura 13 imágenes se pueden ver imágenes en blanco y negro con este tipo de distribución.



Figura 13, Imágenes de Ruido Gaussiano

La salida del generador es un dato parecido a uno del conjunto de datos. La capa de salida debe tener suficientes neuronas y en la forma correcta, para ser capaz de producir datos con la misma estructura que los del *conjunto de datos*, ya sean imágenes, audio etc. Por ejemplo, si se quieren hacer imágenes de 20x20 pixeles, la salida del generador tiene que ser de 400 neuronas (20x20). Cada capa va siendo más grande que la anterior y su activación suele ser Selu, menos la de la última capa que es Sigmoidal.

La entrada del discriminador puede ser, o los datos del conjunto de datos reales, o datos creados por el generador. Su salida es una única neurona con un solo valor numérico, para intentar distinguir entre si la entrada es del conjunto de datos o del generador. Generalmente, cada capa va teniendo menos neuronas, hasta que llega solo a 1.

2.6.2 Entrenamiento de las GAN

En el primer paso se entrena al discriminador para que aprenda a distinguir entre imágenes reales o del generador. En el segundo paso, se entrena al generador. El generador crea imágenes, pero esta vez los pesos del discriminador no se actualizan. Se etiquetan las imágenes creadas como verdaderas, como si fuesen del conjunto de datos, y a través de la retropropagación del gradiente, el generador aprende cómo ajustar los pesos para que el discriminador crea que las imágenes son reales y la próxima vez haga imágenes más parecidas al conjunto de datos.

Ahora estos dos pasos se repiten hasta que el generador produzca los resultados deseados. Se vuelve a entrenar al discriminador con las nuevas imágenes del generador y después se vuelve a entrenar al generador con lo que ha aprendido el discriminador.

2.6.3 Dificultades del entrenamiento de las GAN

Al ser un juego de suma cero, en las GAN cuanto menor error en uno de los dos elementos, mayor es en el otro. Hasta que se llega a un estado que se denomina equilibrio de Nash. Esto se produce cuando ninguna de las dos redes cambia su estrategia. En las GAN se da esta situación cuando el generador produce imágenes perfectamente realistas y el discriminador intenta clasificar con un 50% de probabilidad si la imagen es del generador o del conjunto de datos. No se puede garantizar de ninguna forma que se vaya a llegar a este equilibrio. Por esta razón, es necesario aplicar la técnica de prueba y error para entrenar una GAN.

Otra dificultad del entrenamiento de una GAN aparece cuando el conjunto de datos tiene varias clases. Por ejemplo, si hay que producir mascotas de diferentes tipos (gatos, perros, conejos). El propósito del discriminador es el mismo, distinguir si las imágenes son reales o generadas. Puede haber una categoría en la que el generador produzca más imágenes que el discriminador clasifique como originales. Por ejemplo, puede ser preciso creando imágenes de perros, pero no tanto de gatos. Entonces, el generador empieza a producir más imágenes de perros y poco a poco irá olvidando cómo generar imágenes de otras categorías. A su vez, como el discriminador solo recibe imágenes de perros por parte del generador no clasificar correctamente las imágenes de otras categorías [8].

2.7 Generación de imágenes

La generación de imágenes gracias a la inteligencia artificial ha dado un gran salto en los últimos años, gracias a dos modelos generativos, los autocodificadores variacionales (variational autoencoders, VAE) (2013)[5] y las redes generativas antagónicas (generative adversarial networks, GAN) (2014)[4], más en concreto las que usan capas convolucionales. También existen otras técnicas no relacionadas con la IA para generar imágenes, pero son usadas para otros ámbitos como el cine o los videojuegos. Para el cine y los efectos especiales se usa el término imágenes generadas por ordenador, CGI por sus siglas en inglés. Aunque ese término puede sonar muy amplio, el método consiste en

modelar objetos en tres dimensiones, aplicarles una textura y renderizarlos en dos dimensiones [14].

Las GAN con capas convolucionales (DCGAN) se utilizan típicamente para la generación de imágenes, por lo que los ejemplos que vamos a ver están enfocados a ello. Constan de los mismos elementos que las GAN tradicionales, un generador y un discriminador. La Figura 14 muestra una arquitectura típica de una DCGAN. En este caso la primera capa del generador es una capa con 3 neuronas. Después se aplican tres capas de convolución transpuesta y y normalización y la salida de esta última convolución transpuesta es una imagen a color. El discriminador recibe esa imagen y le aplica dos veces un filtro convolucional seguido de una reducción. Por último, los datos pasan por dos capas alimentadas hacia adelante.

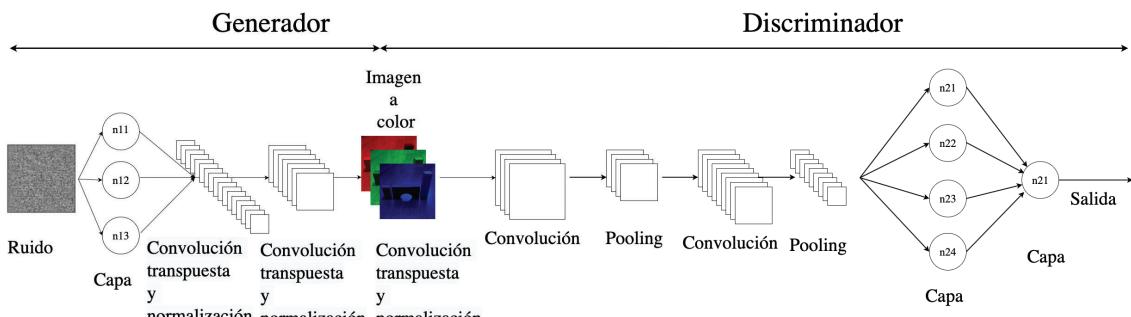


Figura 14, Arquitectura de Red Generativa Antagónica con Capas Convolucionales (DCGAN)

Menos la primera capa del generador, que es una capa de redes de neuronas alimentada hacia adelante, el resto es una red convolucional traspuesta. Cada vez usa capas más grandes, pero con menos filtros. Su entrada es una distribución aleatoria, normalmente Gaussiana o uniforme. Su salida son los datos sintéticos a generar, normalmente una imagen, con un filtro si se quiere que la imagen sea en blanco y negro o tres filtros en el caso en el que se quieran imágenes a color. Su objetivo es crear imágenes que el discriminador crea que son reales. La salida del generador está conectada con la entrada del discriminador. En Figura 15 se puede ver el generador de una DCGAN de imágenes a color. Su entrada es ruido, se procesan estos datos en una capa de 3 neuronas, después se aplican tres capas de convolución transpuesta y y normalización de los valores y se obtiene la imagen a color.

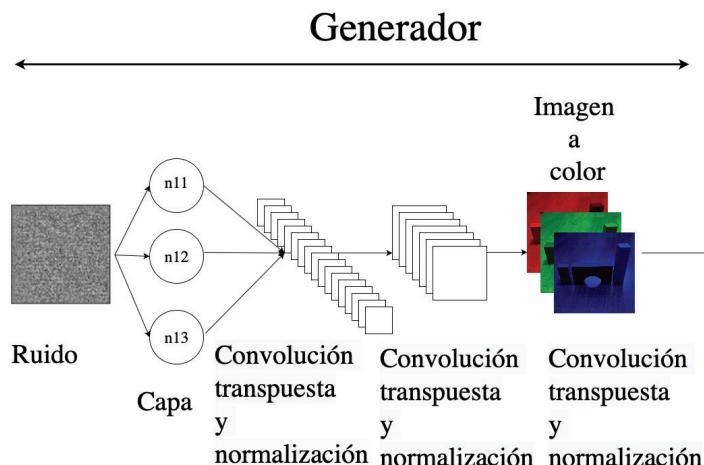


Figura 15, Generador con Capas Convolucionales

El discriminador es una red convolucional, con la estructura típica de los clasificadores binarios, menos la última parte que suele constar de una o dos capas de redes de neuronas alimentadas hacia adelante. Cada vez usa capas más pequeñas, pero más filtros. Su entrada son datos con el mismo formato que la salida del generador, lo más común son imágenes. Estas imágenes vienen del generador o del conjunto de datos. Su salida es una capa de una sola neurona, normalmente con una activación Sigmoide. Su objetivo es clasificar los datos en imágenes del conjunto de datos o creadas por el generador.

En la Figura 16 se puede observar un discriminador de una DCGAN de imágenes a color. Se aplican dos veces convolución y reducción sucesivamente. Por último, la información pasa por dos capas, siendo la ultima de una sola neurona.

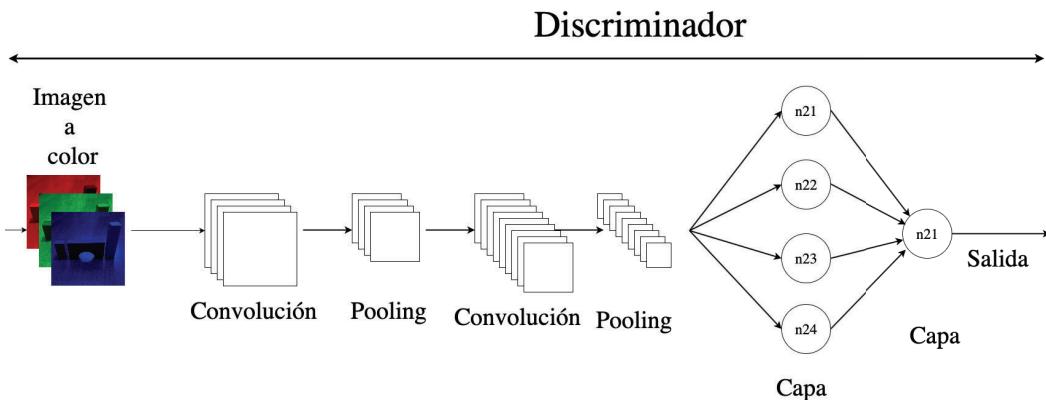


Figura 16, Discriminador con Capas Convolucionales

Otra técnica común para generar nuevas imágenes, usando redes neuronales es mediante los denominados autocodificadores variacionales (VAE por sus siglas en inglés). Tiene ciertas similitudes respecto las GAN. Los autocodificadores son redes de neuronas compuestas por dos elementos, el codificador y el decodificador.

La función de los autocodificadores es intentar que las entradas de la red y la salida sean las mismas. Para ello, el número de neuronas en la capa de entrada y en la de salida es la misma. Cada capa del codificador tiene menos neuronas que la capa anterior, por lo que se comprimen los datos hasta lo que se conoce como representación latente (*latent representation*) o código (*code*). El decodificador tiene que tomar esta representación intermedia e intentar convertirla de vuelta al conjunto de datos original.

En la Figura 17 muestra un autocodificador compuesto por 4 entradas, el codificador tiene dos capas, la primera de 4 neuronas y la segunda de 2. El decodificador tiene una capa de 4 neuronas. La representación latente de los datos se produce cuando los datos atraviesan la capa con 2 neuronas.

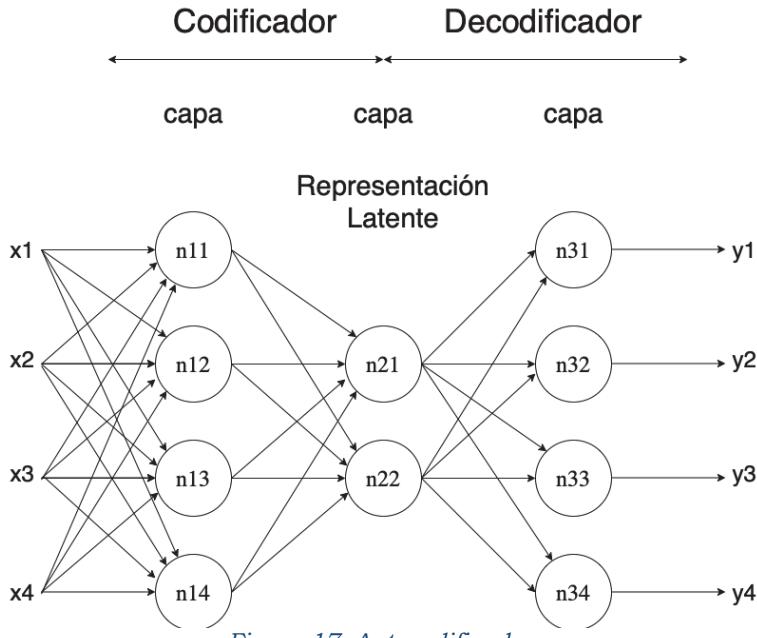


Figura 17, Autocodificador

Los VAE son un tipo de redes de neuronas generativas y probabilísticos. Son modelos generativos porque son capaces de producir datos similares al conjunto de datos con el que son entrenados. Son probabilísticos porque para las mismas entradas, pueden producir diferentes salidas. Usan la aleatoriedad, no solo para ser entrenados, si no también para producir nuevos datos. Los VAE aplican una pequeña modificación siguiendo una distribución Gaussiana a la representación latente para generar nuevos datos.

Otro uso común de los autocodificadores y los VAE es la interpolación de datos. Una vez se tiene la representación latente de dos datos. Dos datos pueden ser interpolados para producir nuevos datos, mediante la media ponderada de la representación latente de los datos. Se le puede dar mas importancia a un dato o a otro para que sean interpolados en diferentes proporciones [5].

3 Planteamiento del problema

El objetivo principal de este trabajo de fin de grado es la implementación en Python de una arquitectura GAN cuyo propósito es el de ser capaz de generar imágenes que sean, para el ojo humano, lo más similares posibles a imágenes reales. En este caso el conjunto de entrenamiento elegido es de flores. Para ello, se hace uso de las librerías Keras y Tensorflow. Debido a que Tensorflow acepta ser ejecutado en GPU y esto reduce considerablemente el tiempo de entrenamiento de la red, se emplea Google Colab (<https://colab.research.google.com>) utilizando el entorno de ejecución GPU.

La arquitectura GAN propuesta en este trabajo para la generación de imágenes de flores consta de dos redes de neuronas, una generadora y otra discriminadora. La red generadora transforma una variable aleatoria uniforme de entrada en imágenes de flores que siguen la distribución de probabilidad del conjunto de entrenamiento. El propósito de la red discriminadora es el de diferenciar si las imágenes de flores que le llegan a la entrada proceden del conjunto de datos de entrenamiento (datos reales) o de la red generadora (datos producidos por el generador). El objetivo del proceso de entrenamiento es que la red generadora produzca imágenes de flores que sean indistinguibles del conjunto de entrenamiento para la red discriminadora.

El uso de esta red puede servir para generar conjuntos de datos con el propósito de entrenar otras redes de neuronas. También como inspiración para el diseño de ropa o pintar cuadros, entre otros ejemplos. También podría ser usada para la generación de flores para videojuegos o películas en los que se requieran nuevos diseños de flores no existentes o una gran cantidad de ellas. Para ello, este trabajo se ha dividido en los siguientes cinco subobjetivos:

1. Localizar un repositorio público de imágenes de flores adecuado al problema a resolver. Para ello, debe disponer de un numero de imágenes suficiente para poder entrenar adecuadamente la arquitectura GAN y que sea capaz de generar variedad de imágenes diferentes. Asimismo, las imágenes contenidas en el repositorio deben ser diferentes entre sí, para que el sistema inteligente aprenda diferentes tipos de flores, pero también lo suficientemente similares para que la red generadora sea capaz de aprender los patrones comunes existentes en las flores. Dicho de manera metafórica, que la red generadora aprenda lo que es un ramo de flores.
2. Preparación y limpieza de las imágenes de flores a utilizar. Las imágenes deben tener la forma y estructura correctas para que puedan ser usadas por redes de neuronas.
3. Adquisición de los conocimientos necesarios para realizar el proyecto, más en concreto, sobre las APIs Keras y Tensorflow y también sobre el aprendizaje profundo y las DCGAN.
4. Implementación de una arquitectura DCGAN y su proceso de entrenamiento en Python, usando Keras y Tensorflow, dentro del entorno de ejecución Google Colab.
5. Evaluación del funcionamiento y de los resultados de la red generativa antagónica, mediante la aplicación para dispositivos inteligentes Google

Lens. En la prueba se evaluará el porcentaje de imágenes generadas que son clasificadas como imágenes de flores o relacionadas con ellas.

4 Solución Propuesta

Para la construcción de las redes neuronales, no hay fórmula exacta. Depende del conjunto de datos de entrenamiento y de los resultados que se van obteniendo en el proceso de construcción y ajuste de hiperparámetros de la red. Una opción es basarse en arquitecturas de redes ya existentes cuyo propósito sea similar e ir ajustando los hiperparámetros de la red hasta conseguir los resultados deseados. Los hiperparámetros de una red no tienen por qué ser los mismos que los de otra, aunque las dos redes tengan una arquitectura y un propósito similar.

4.1 Conjunto de datos

Para el entrenamiento de la red generativa antagónica para la generación de imágenes de flores se ha utilizado el conjunto de entrenamiento 102 Category Flower Dataset (<https://www.robots.ox.ac.uk/~vgg/data/flowers/102>), que consiste en 8195 imágenes de fotos de flores comunes de Reino Unido. Cada fotografía tiene una resolución de entre 500 y 700 píxeles en cada eje. En la siguiente figura se puede ver una matriz de 9x9 imágenes obtenidas del conjunto de datos, reduciendo la resolución a 56x56 píxeles.



Figura 18, Imágenes del Conjunto de Datos con Resolución 56x56

4.2 Preprocesado de datos

La primera cuestión que hay que afrontar para la construcción de una GAN para la generación de imágenes es la de elegir la resolución de estas y si van a ser a color o en blanco y negro. A partir de un conjunto de pruebas preliminares, se observó que la resolución más adecuada para que el generador produzca las imágenes de forma adecuada es de 56x56 píxeles. Por ello, ha sido necesario redimensionar las imágenes del conjunto de entrenamiento a esa misma resolución, mediante la librería cv2 de google.colab.patches, que hace uso de la interpolación bilineal. Cuando la resolución es mayor de 60x60 píxeles, el tiempo de entrenamiento aumenta considerablemente y la red no encuentra patrones de similitud en las imágenes, por lo que los resultados obtenidos no son adecuados. Si, por el contrario, la resolución es menor de 40x40 píxeles, no se distingue el contenido de las imágenes generadas, como se puede apreciar en la Figura 19.

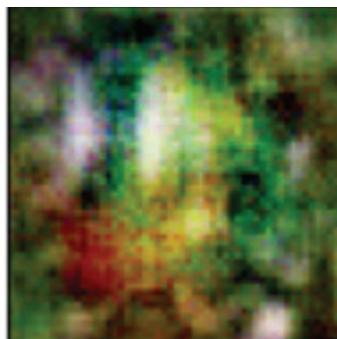


Figura 19, Flor Falsa de 40x40 Píxeles

Debido a que las imágenes contienen flores, resulta más adecuada que sean en color en lugar de blanco y negro. Así mismo, las imágenes en color del conjunto de datos tienen mayor resolución y una proporción casi cuadrada. Dado que la entrada del discriminador pueden ser imágenes del conjunto de datos o imágenes de la salida del generador, es necesario que el intervalo de valores de los píxeles sea igual en ambos casos. La salida del generador, al tener activación Sigmoide, proporciona valores en el intervalo [-1,1]. Pero al importar las imágenes del *dataset*, los valores de cada píxel van desde 0 hasta 255, por lo que hay que normalizarlos al intervalo [-1,1], empleando la siguiente fórmula: nuevo píxel=(valor_antiguo/127,5)-1. Todo este proceso se puede apreciar en la Figura 20.

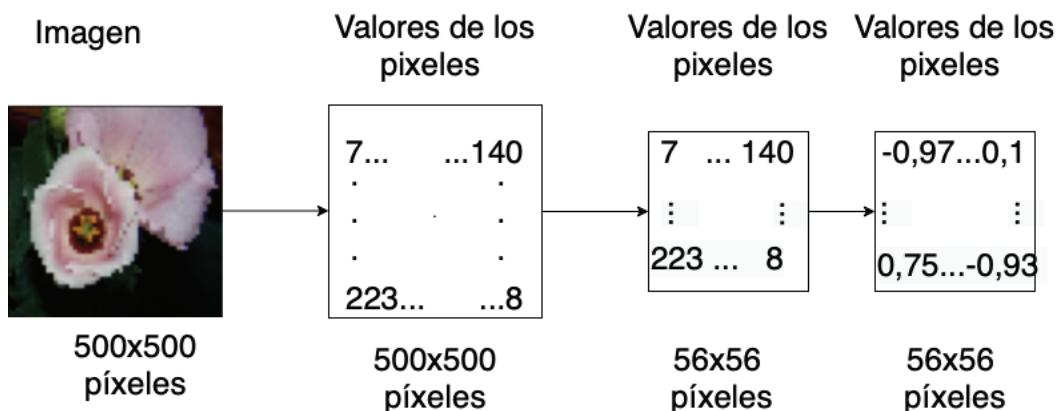


Figura 20, Preprocesado de Imágenes

4.3 Arquitectura de la red

En cuanto a la arquitectura de la GAN, al ser usada para la generación de imágenes, el tipo de neuronas utilizadas son convolucionales, por lo que la red es DCGAN. El generador y el discriminador deben tener una estructura similar, con el mismo número de capas y similar numero de filtros, pero estructurados de una manera inversa la una a la otra. Esto es debido a que deben de tener una capacidad de computación similar para que no se produzcan problemas a la hora del entrenamiento.

Para producir cada imagen, se genera un vector de 400 valores aleatorios entre -1 y 1 para la entrada del generador. Debido a que la resolución de las imágenes es de 56x56 píxeles, se ha optado por una primera capa convolucional del generador de 14x14 píxeles. Al usar dos capas convolucionales transpuestas, se cuadriplica la resolución, pasando a 56x56 píxeles.

La primera capa oculta del generador tiene 25088 neuronas. Después se redimensionan las 25088 salidas de la capa anterior en 128 imágenes de un solo canal de 14x14. La segunda es una capa convolucional transpuesta con un tamaño de filtro de 5x5 pixeles, 84 filtros, con un desplazamiento de 2, *zero padding* y una activación SELU. La salida son 84 imágenes de 28x28. La tercera y última capa es una capa convolucional transpuesta con las mismas características que la anterior, excepto que se han utilizado 3 filtros y la función de activación tangente hiperbólica. La entrada de esta son 84 imágenes de 28x28 pixeles, siendo la salida 3 imágenes de 56x56 pixeles que se combinan en una sola para generar una imagen a color. Entre capa y capa se normalizan los valores.

La red generadora propuesta tiene un total de 10.336.323 parámetros divididos en 10.335.889 parámetros entrenables y 424 no entrenables. El resumen de la estructura y parámetros de la red generadora se puede ver en las Figuras 21 y 22, respectivamente.

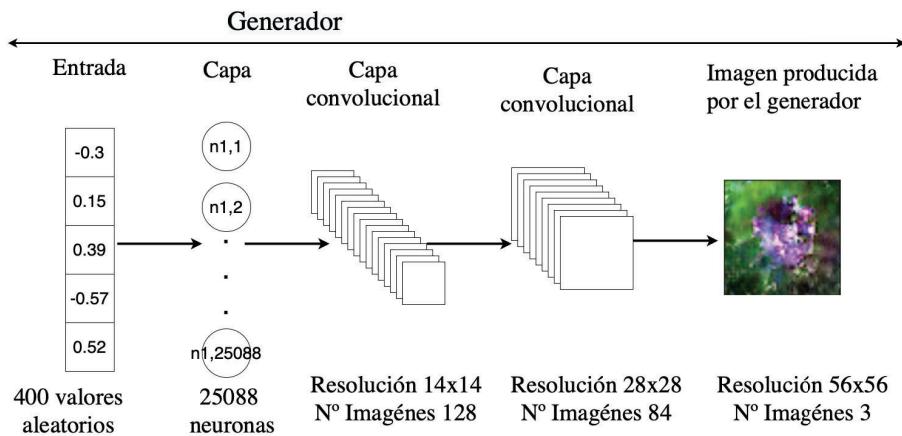


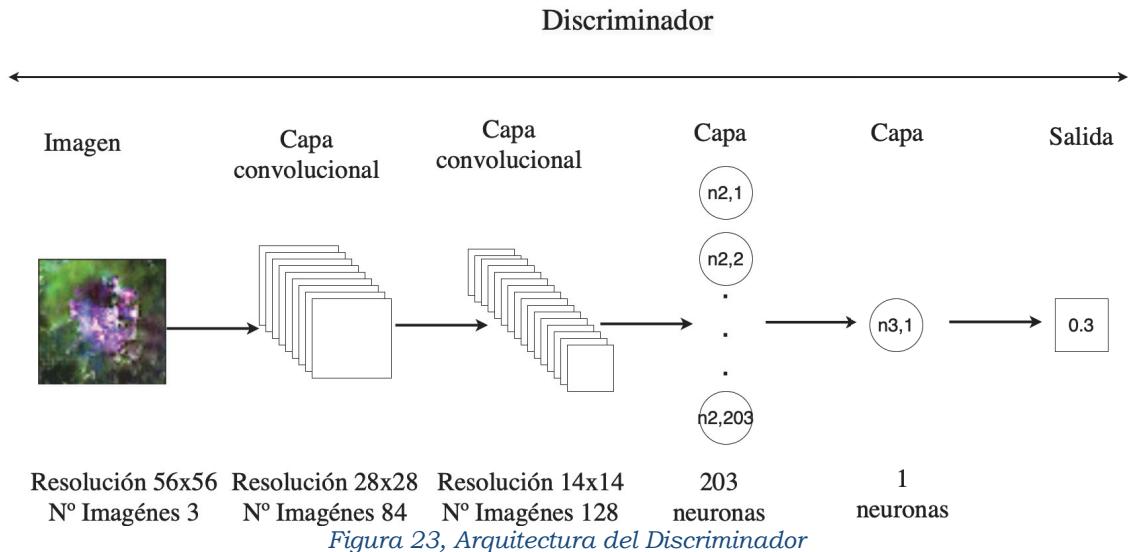
Figura 21, Arquitectura del Generador

Model: "Generador"

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 25088)	10060288
reshape_1 (Reshape)	(None, 14, 14, 128)	0
batch_normalization_2 (BatchNormalization)	(None, 14, 14, 128)	512
conv2d_transpose_2 (Conv2DTranspose)	(None, 28, 28, 84)	268884
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 84)	336
conv2d_transpose_3 (Conv2DTranspose)	(None, 56, 56, 3)	6303
<hr/>		
Total params: 10,336,323		
Trainable params: 10,335,899		
Non-trainable params: 424		

Figura 22, Parámetros del Generador

La entrada de la red discriminadora son imágenes a color de 56x56 píxeles. La primera capa es una capa convolucional 2D, con 84 filtros, un tamaño de filtro de 5x5, un desplazamiento de 2, *zero padding*, y función de activación LeakyReLU con parámetro 0,2. Esta capa convierte las 3 imágenes de 56x56 en 84 imágenes de 28x28. La siguiente capa también es convolucional, con los mismos parámetros, pero con 128 filtros. La entrada de esta capa son las 84 imágenes de 28x28 píxeles y la salida son 128 imágenes de 14x14x píxeles. La siguiente capa tiene 203 neuronas con activación SELU. Por último, hay una capa con una sola neurona y activación Sigmoide, que es la encargada de clasificar si la imagen de la entrada es una imagen real del conjunto de datos o falsa, generada por el generador. En total, la red GAN cuenta con un total de 5.368.583 parámetros, configurados para ser entrenables o no, dependiendo de la fase de aprendizaje en la que se encuentre la red. Cuando se entrena solo el discriminador, estos parámetros son todos entrenables, pero cuando se entrena la GAN, solo se modifica el generador. Un resumen de la arquitectura de esta red discriminadora junto a sus parámetros se puede ver en las Figuras 23 y 24, respectivamente.



Model: "Discriminador"		
Layer (type)	Output Shape	Param #
conv2d_16 (Conv2D)	(None, 28, 28, 84)	6384
dropout_16 (Dropout)	(None, 28, 28, 84)	0
conv2d_17 (Conv2D)	(None, 14, 14, 128)	268928
dropout_17 (Dropout)	(None, 14, 14, 128)	0
flatten_8 (Flatten)	(None, 25088)	0
dense_19 (Dense)	(None, 203)	5093067
dense_20 (Dense)	(None, 1)	204

Total params: 5,368,583
 Trainable params: 5,368,583
 Non-trainable params: 0

Figura 24, Parámetros del Discriminador

En la figura 25 se puede ver el resumen de la GAN junto con la salida de cada una de las redes y el número de parámetros de cada una.

Model: "DCGAN"		
Layer (type)	Output Shape	Param #
Generador (Sequential)	(None, 56, 56, 3)	10336323
Discriminador (Sequential)	(None, 1)	5368583
<hr/>		
Total params: 15,704,906		
Trainable params: 10,335,899		
Non-trainable params: 5,369,007		

Figura 25, Parámetros y Estructura de la DCGAN

Los hiperparámetros que he usado de la red han sido: *binary-cross entropy* como función de pérdida, Rmsprop como optimizador, la red ha sido entrenada durante 6000 épocas, se ha empleado un tamaño de *batch* de 64, dividido en 32 imágenes reales y 32 imágenes del generador y una tasa de aprendizaje de 0,1.

4.4 Búsqueda de hiperparámetros

Para la búsqueda de los hiperparámetros, se ha basado la elección en los siguientes criterios: convergencia, resultados obtenidos y tiempo de entrenamiento. En las DCGAN, cuando se llega a un estado de equilibrio entre el generador y discriminador (el entrenamiento converge), los valores de las pérdidas de estas dos redes no se modifican ligeramente como ocurre en las redes densas, si no que la variación es más amplia. No basta tampoco con elegir simplemente la configuración que produzca la menor pérdida porque se producen grandes cambios en el valor de pérdida entre una época (*epoch*) y la siguiente. Por esta razón, es necesario revisar el resultado de las imágenes obtenidas para comprobar que tienen la calidad suficiente. Se puede dar el caso que una red con mayor perdida produzca mejores resultados que otra con menor error.

Cuando los hiperparámetros no son los correctos, la pérdida del discriminador aumenta con cada época y la precisión de este no supera el 50%. En las pruebas realizadas, las imágenes producidas en esta situación son colores aleatorios. La Figura 26 muestra la evolución de la pérdida del discriminador durante 4000 épocas en las que el entrenamiento de la red no se produce de forma correcta (no converge). Se puede observar que la pérdida aumenta en la red discriminadora conforme avanza el entrenamiento porque va perdiendo su capacidad de clasificar las entradas en reales.

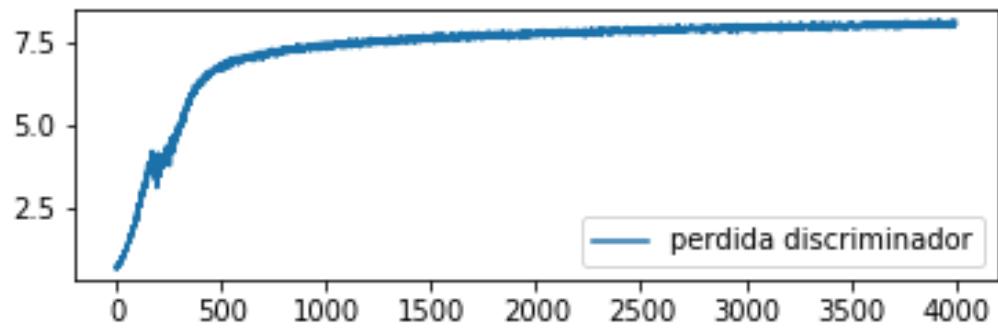


Figura 26, Pérdida del Discriminador Cuando la Red No Converge

De forma similar, la Figura 27 muestra la evolución de la precisión del discriminador en un proceso de entrenamiento que no es el adecuado y el discriminador no es capaz de distinguir de forma correcta las imágenes, lo que produce a su vez que el generador tampoco aprenda de forma correcta. La Figura 28 muestra las imágenes que produce el generador en esta situación.

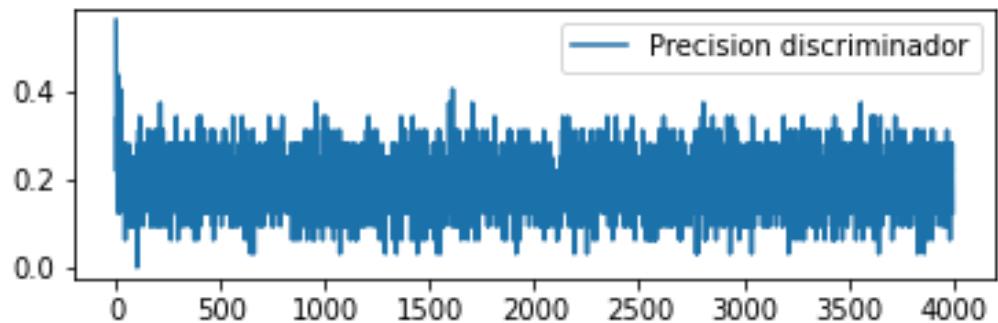


Figura 27, Precisión del Discriminador Cuando la Red No Converge

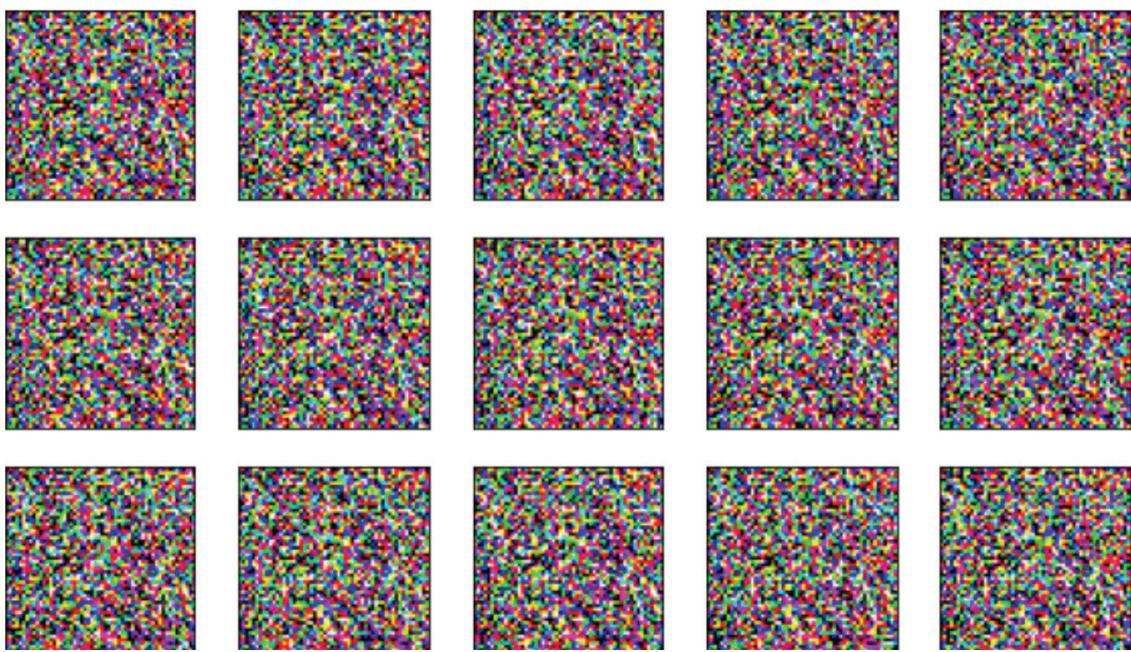


Figura 28, Resultados Obtenidos Cuando la Red No Converge

Se ha usado *binary-cross entropy* como función de pérdida, debido a que el problema que intenta resolver el discriminador es de clasificación entre dos posibles categorías. El tamaño de *batch* es de 64 porque si es mayor, la red tarda más en llegar a los mismos resultados que con 64 y si es menor, los resultados no producen imágenes de flores que parezcan reales.

La tasa de aprendizaje elegida es de 0,1 porque es el valor más alto con el que la red converge y, por lo tanto, el entrenamiento es más rápido.

5 Resultados

5.1 Análisis del entrenamiento

Para analizar los resultados obtenidos y ver si la red es capaz de entrenar de una manera correcta, hay que fijarse en tres variables: la pérdida del generador, del discriminador y la precisión del discriminador. La función de pérdida es una medida que indica la diferencia entre los valores obtenidos y los deseados. La precisión es una medida que indica cuántas imágenes han sido clasificadas correctamente. En las Figuras 29 y 30 se pueden ver los gráficos de estas tres variables a lo largo de las primeras 4000 épocas. Durante las 250 primeras épocas, se puede observar que la perdida, tanto del discriminador como del generador decrecen de manera considerable. También se puede ver que desde las 500 hasta las 750 épocas, se vuelve inestable el entrenamiento aumentando la pérdida de las dos redes, pero después se vuelve a estabilizar con una tendencia a decrecer. Por lo general, los valores del generador (media de 1,7028) son más altos que los del discriminador (media de 0,6481) y se mueve por un rango de valores más amplio (varianza 12,1907 frente a 0,4454).

La precisión del discriminador tiene una tendencia decreciente y su rango de valores se va acotando (media de 0,6625 y varianza de 0,0141). A partir de las 3500 épocas, el entrenamiento ya es estable y los resultados obtenidos mejoran de una forma menos notable.

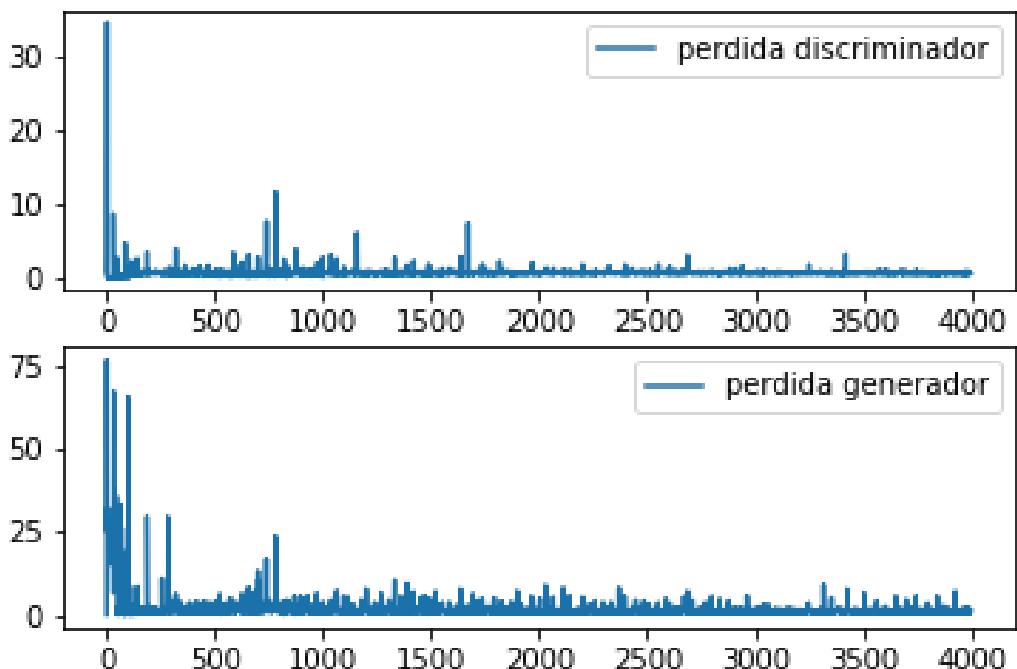


Figura 29, Pérdida del Discriminador y Generador

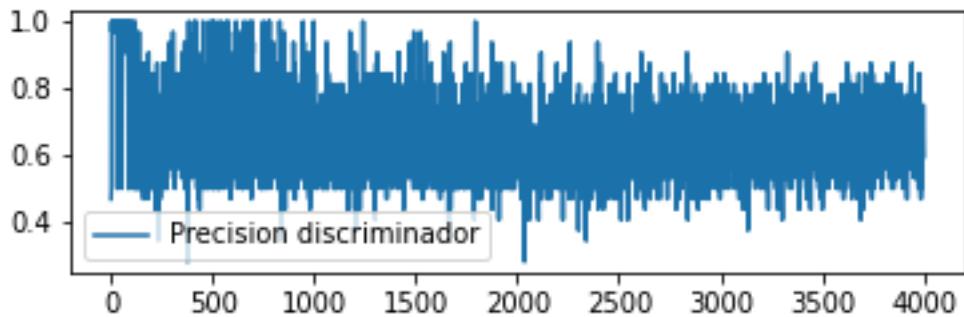


Figura 30, Precisión del Discriminador

5.2 Análisis de las imágenes generadas

A continuación, se muestra numéricamente, cada 200 épocas, cómo van evolucionando estos tres valores durante las primeras 4000 épocas en una ejecución típica. Las Figuras 31 a 51 muestran las imágenes de flores producidas cada 200 épocas.

Época 1: pérdida discriminador: 2,7464911937713623; pérdida generador: 76,8624267578125; precisión discriminador: 0,5. Las imágenes de flores generadas en esta iteración se muestran en la Figura 31



Figura 31, Flores Producidas en la Iteración 1

Época 199: pérdida discriminador: 0,57367; pérdida generador: 0,21056; precisión discriminador: 0,71875. Las imágenes de flores generadas en esta iteración se muestran en la Figura 32

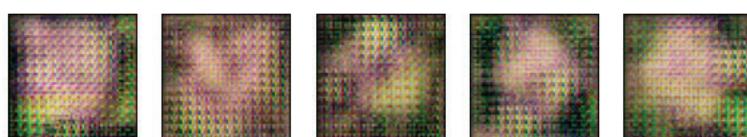


Figura 32, Flores Producidas en la Iteración 199

Época 399: pérdida discriminador: 1,03817; pérdida generador: 0,78529; precisión discriminador: 0,53125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 33

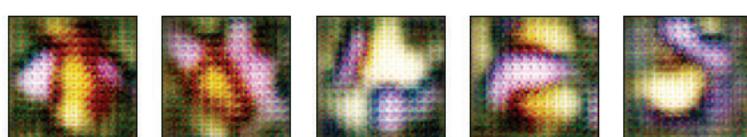


Figura 33, Flores Producidas en la Iteración 399

Época 599: pérdida discriminador: 0,72493; pérdida generador: 1,22249; precisión discriminador: 0,78125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 34

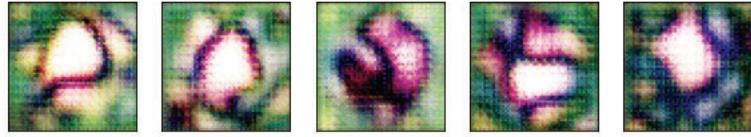


Figura 34, Flores Producidas en la Iteración 599

Época 799: pérdida discriminador: 0,32753; pérdida generador: 3,17622; precisión discriminador: 0,875. Las imágenes de flores generadas en esta iteración se muestran en la Figura 35



Figura 35, Flores Producidas en la Iteración 799

Época 999: pérdida discriminador: 2,74486; pérdida generador: 3,86952; precisión discriminador: 0,5. Las imágenes de flores generadas en esta iteración se muestran en la Figura 36



Figura 36, Flores Producidas en la Iteración 999

Época 1199: pérdida discriminador: 0,60457; pérdida generador: 0,94807; precisión discriminador: 0,625. Las imágenes de flores generadas en esta iteración se muestran en la Figura 37



Figura 37, Flores Producidas en la Iteración 1199

Época 1399: pérdida discriminador: 0,50702; pérdida generador: 1,60565; precisión discriminador: 0,75. Las imágenes de flores generadas en esta iteración se muestran en la Figura 38

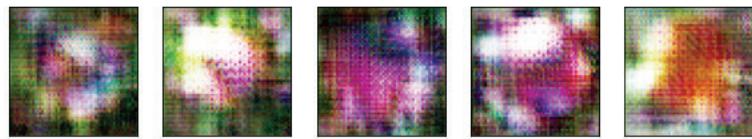


Figura 38, Flores Producidas en la Iteración 1399

Época 1599: pérdida discriminador: 0,65188; pérdida generador: 0,53386; precisión discriminador: 0,59375. Las imágenes de flores generadas en esta iteración se muestran en la Figura 39

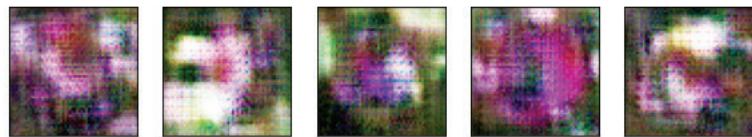


Figura 39, Flores Producidas en la Iteración 1599

Época 1799: pérdida discriminador: 0,57795; pérdida generador: 1,19919; precisión discriminador: 0,75. Las imágenes de flores generadas en esta iteración se muestran en la Figura 40

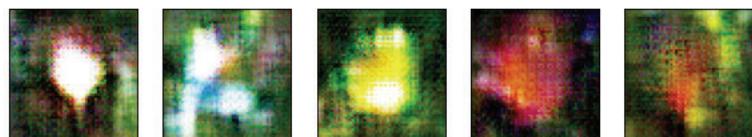


Figura 40, Flores Producidas en la Iteración 1799

Época 1999: pérdida discriminador: 0,48308; pérdida generador: 1,05591; precisión discriminador: 0,75. Las imágenes de flores generadas en esta iteración se muestran en la Figura 41.



Figura 41, Flores Producidas en la Iteración 1999

Época 2199: pérdida discriminador: 0,69797; pérdida generador: 0,75700; precisión discriminador: 0,53125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 42.



Figura 42, Flores Producidas en la Iteración 2199

Época 2399: pérdida discriminador: 0,80954; pérdida generador: 1,51143; precisión discriminador: 0,5. Las imágenes de flores generadas en esta iteración se muestran en la Figura 43.

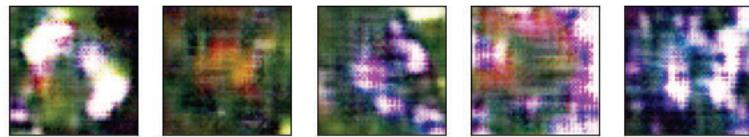


Figura 43, Flores Producidas en la Iteración 2399

Época 2599: pérdida discriminador: 0,57756; pérdida generador: 0,79269; precisión discriminador: 0,6875. Las imágenes de flores generadas en esta iteración se muestran en la Figura 44.



Figura 44, Flores Producidas en la Iteración 2599

Época 2799: pérdida discriminador: 1,07249; pérdida generador: 5,59351; precisión discriminador: 0,46875. Las imágenes de flores generadas en esta iteración se muestran en la Figura 45.



Figura 45, Flores Producidas en la Iteración 2799

Época 2999: pérdida discriminador: 0,62175; pérdida generador: 0,90918; precisión discriminador: 0,53125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 46.



Figura 46, Flores Producidas en la Iteración 2999

Época 3199: pérdida discriminador: 0,59133; pérdida generador: 0,66381; precisión discriminador: 0,625. Las imágenes de flores generadas en esta iteración se muestran en la Figura 47.



Figura 47, Flores Producidas en la Iteración 3199

Época 3399: pérdida discriminador: 0,70290; pérdida generador: 1,05350; precisión discriminador: 0,53125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 48.



Figura 48, Flores Producidas en la Iteración 3399

Época 3599: pérdida discriminador: 0,57937; pérdida generador: 0,86412; precisión discriminador: 0,75. Las imágenes de flores generadas en esta iteración se muestran en la Figura 49.



Figura 49, Flores Producidas en la Iteración 3599

Época 3799: pérdida discriminador: 0,56889; pérdida generador: 2,09392; precisión discriminador: 0,78125. Las imágenes de flores generadas en esta iteración se muestran en la Figura 50.



Figura 50, Flores Producidas en la Iteración 3799

Época 3999: pérdida discriminador: 0,58937; pérdida generador: 1,93634; precisión discriminador: 0,625. Las imágenes de flores generadas en esta iteración se muestran en la Figura 51.



Figura 51, Flores Producidas en la Iteración 3999

A partir de la época 4000, el proceso de entrenamiento se vuelve mas lento y las mejoras se vuelven casi imperceptibles. Las Figuras 52, 53 y 54 muestran las imágenes producidas en las épocas 6000, 8000 y 16000 respectivamente.



Figura 52, Imágenes Producidas en la Iteración 6000



Figura 53, Imágenes Producidas en la Iteración 8000



Figura 54, 16000 Iteraciones

A partir de la ejecución de entrenamiento típica presentada para la GAN propuesta, se puede observar que ya desde las primeras iteraciones, la red aprende que los bordes de las imágenes son verdes y el centro de colores. Alrededor de las 3000 épocas, las imágenes se empiezan a parecer más a los resultados finales, pero siguen siendo borrosas. De las épocas 4000 a 6000, el único cambio es el aumento en la nitidez de la imagen. A partir de ahí, no hay diferencias apreciables visualmente en los resultados obtenidos durante las iteraciones 6000, 8000 y 16000.

La Figura 55 muestra la salida que se produce cada 200 épocas, manteniendo la misma entrada durante 8000 épocas. Se observa que, en las primeras épocas, se define la forma de la flor y el color y, a partir de ahí, se va mejorando la nitidez, hasta que se llega un punto, cerca de las 6000 épocas, en las que no hay cambios notables.

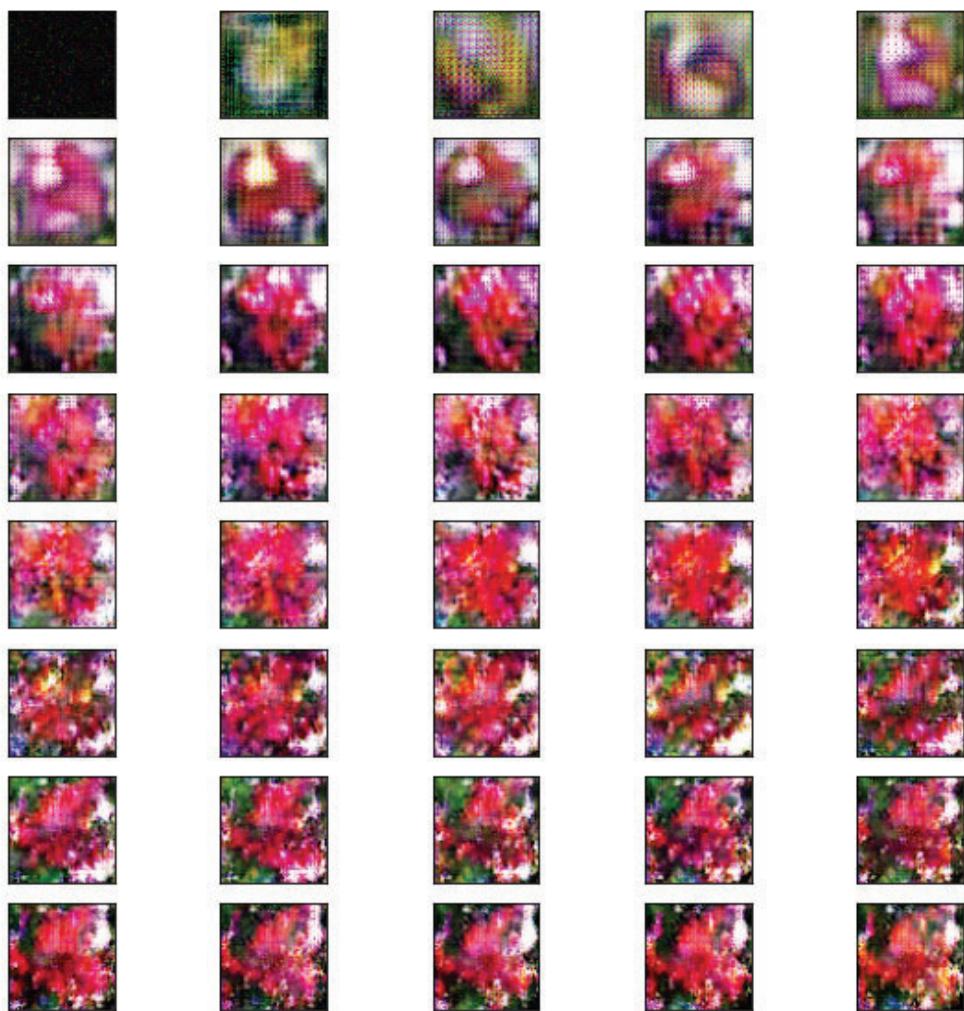


Figura 55, *Misma Entrada Durante el Entrenamiento*

5.3 Prueba de clasificación mediante Google Lens

Para comprobar la semejanza de los resultados obtenidos con imágenes de flores reales, se ha hecho uso de una aplicación móvil de Google llamada Google Lens. Esta aplicación tiene la opción de realizar una búsqueda por imágenes y como resultado, se obtiene una clasificación de la imagen. Si no fuera posible etiquetar la imagen de entrada, se obtienen imágenes similares.

Para este experimento, se ha utilizado la aplicación para Android versión 1.14.220323019. Para realizar la prueba, se han usado las flores producidas en la Figura 52. En la Figura 56, se puede ver una captura de pantalla de esta prueba. Arriba aparece la imagen de la DCGAN y abajo la categoría a la que pertenece, en este caso, una planta *Rododendro* o, más comúnmente, *azalea*.

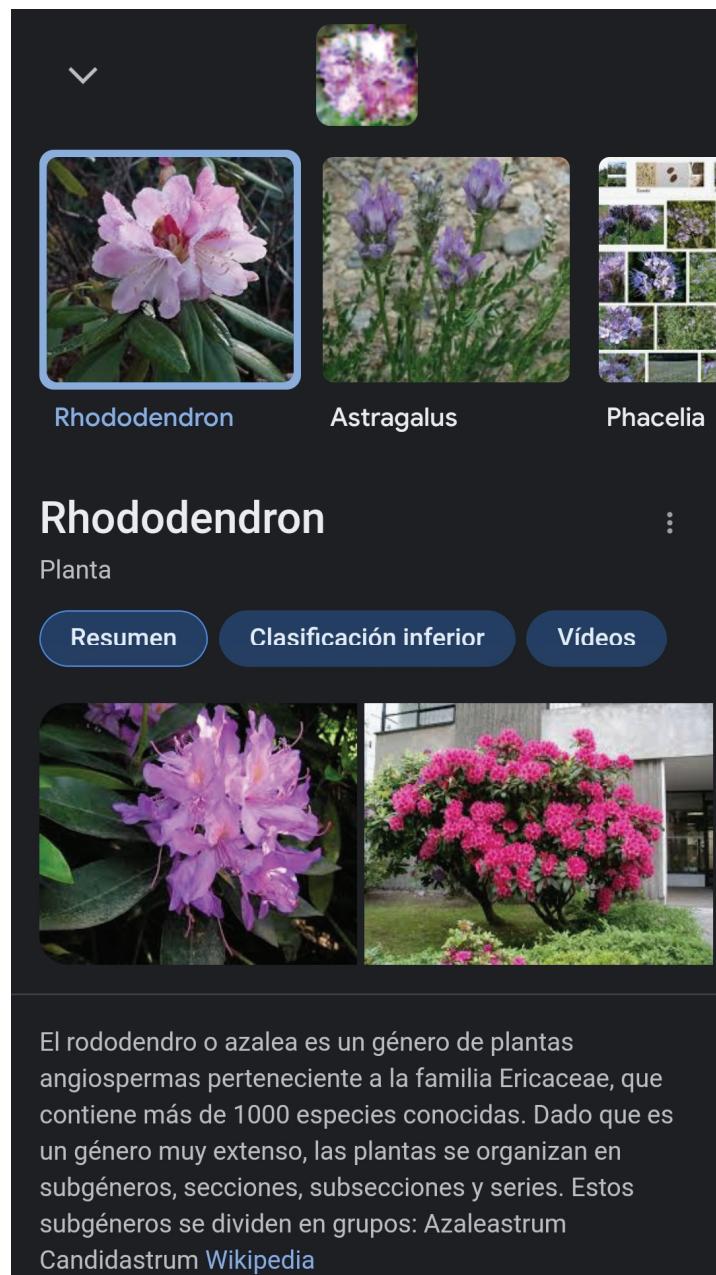


Figura 56, Imagen Falsa Etiquetada Correctamente

En algunas de las imágenes producidas por la red de neuronas generadora, Google Lens ha producido un error en la clasificación, no siendo capaz de clasificar las imágenes o clasificándolas como cuadros, pero cuadros de flores. Un ejemplo se puede ver en la figura 57.

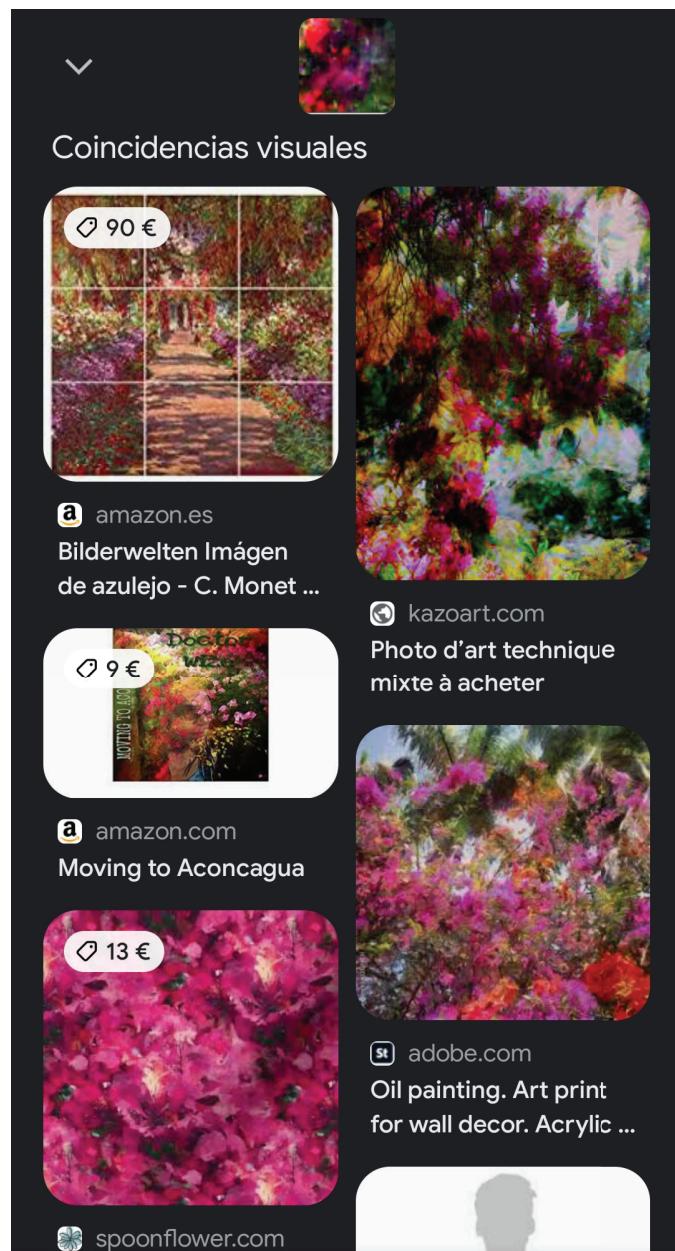


Figura 57, Imagen Generada con Coincidencias de Cuadros de Flores

Por último, está la categoría de imágenes clasificadas de forma incorrecta, donde las imágenes relacionadas producidas por Google Lens no son flores. Un ejemplo se puede observar en la Figura 58.

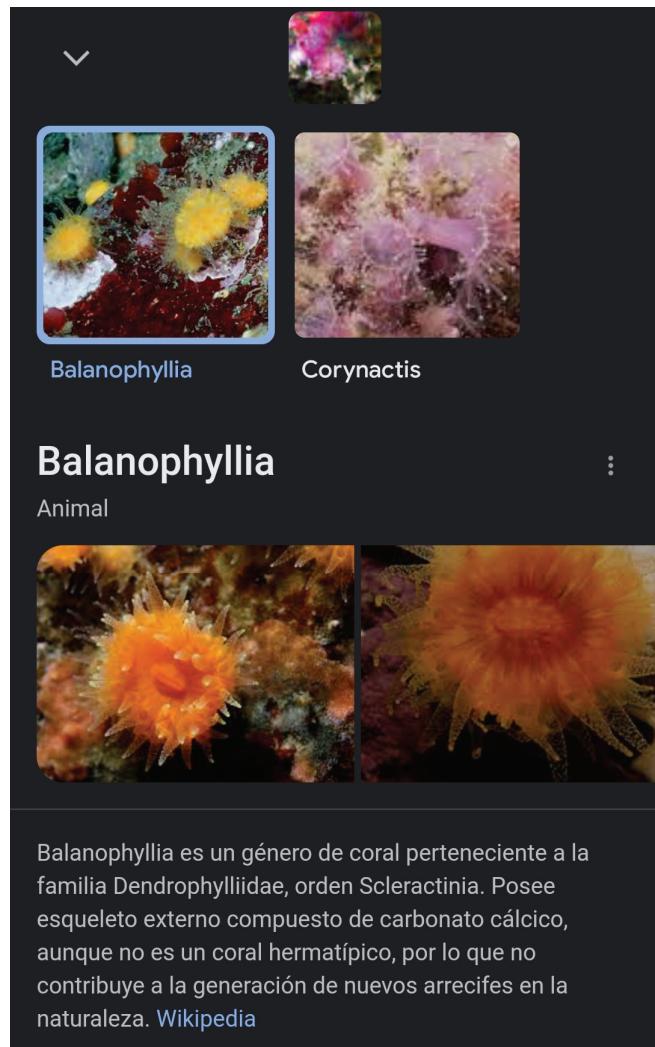


Figura 58, Imagen Generada Clasificada Erróneamente

De las 25 imágenes generadas y mostradas en la Figura 52, 15 (60%) han sido clasificadas correctamente, 7 (28%) están relacionadas con flores y 3 (12%) no tienen similitud con flores de acuerdo a la aplicación Google Lens.

6 Conclusiones y líneas futuras

6.1 Conclusiones

En este trabajo de fin de grado, se ha implementado un generador de imágenes de flores. Esto ha sido posible gracias a un modelo generador, conocido como red generativa antagónica convolucional, DCGAN, perteneciente al ámbito del aprendizaje automático profundo, dentro de la Inteligencia Artificial. Para ello, se ha realizado un estudio previo para la adquisición de conocimientos en TensorFlow y Keras, GAN y DCGAN. Posteriormente, se ha diseñado, implementado y evaluado la red obtenida como solución, implementada en el lenguaje de programación Python con Tensorflow y Keras y usando el entorno de desarrollo de Google Colab. El código se puede encontrar en el siguiente enlace(<https://colab.research.google.com/drive/14JkUuDpLm7sDnoj6HRDsI0Ik712ae8dr?usp=sharing>). Para obtener esta red solución, se ha realizado una búsqueda de los hiperparámetros óptimos para entrenar la red de una manera eficiente y obtener las imágenes de flores más parecidas a las del conjunto de datos original.

Los resultados obtenidos son favorables. El entrenamiento de la red ha sido capaz de converger y llegar a una situación de equilibrio. Las imágenes producidas por la DCGAN son los deseados, dado que un ser humano es capaz de distinguir que las imágenes generadas son de flores. También, al realizar una búsqueda mediante Google Lens con las imágenes generadas por la DCGAN, el buscador obtiene, en la mayoría de los casos, imágenes relacionadas con flores.

6.2 Líneas Futuras

La finalización de este trabajo abre la puerta al desarrollo de otros proyectos futuros. En primer lugar, el aumento de la resolución y nitidez de las imágenes producidas o la reducción de ruido de estas. Para ello sería necesario implementar una red generativa antagónica de súper-resolución (SRGAN), cuya entrada fuesen las imágenes de la DCGAN. El uso principal de este tipo de redes es el de aumentar la resolución de las imágenes de entrada, mejorando así también la nitidez.

Asimismo, resulta interesante modificar la DCGAN propuesta en este trabajo para que sea capaz de producir imágenes imitando diferentes estilos de pintura. Este tipo de redes se les conoce como CycleGAN. Este tipo de redes son similares a las GAN, pero la función del generador, en vez de ser generar las imágenes a partir de datos pertenecientes a una distribución de probabilidad, es cambiar el estilo a una imagen de entrada, por ejemplo, imitando estilos de pintura de artistas famosos.

Una posible aplicación de la red generativa convolucional implementada en este trabajo sería la construcción de un sistema inteligente capaz de realizar diseños de ramos de flores para diferentes eventos como bodas, aniversarios o funerales. Para ello, habría que investigar posibles métodos de combinar varios tipos de ramos y coronas de flores.

La utilización de autocodificadores variacionales (VAE) o un híbrido de estos modelos con las GAN, conocido como VAE-GAN, es una alternativa al objetivo de este trabajo en la generación de flores. Resulta interesante realizar un estudio comparativo del uso de los tres sistemas: GAN, VAE y VAE-GAN.

7 Análisis de Impacto

Este capítulo recoge el impacto del proyecto y los resultados obtenidos en los siguientes ámbitos:

- Personal: Desde antes de iniciar el grado, ya tenía claro que me quería especializar en la Inteligencia Artificial. Mi interés y conocimientos sobre este campo, pero especialmente en aprendizaje automático, ha ido creciendo durante el desarrollo del TFG. Haber realizado un proyecto desde el principio hasta el final, en un tema tan vanguardista como las DCGAN, ha supuesto una gran oportunidad en mi aprendizaje, sobre todo por haber conseguido unos resultados satisfactorios.
- Empresarial y económico: El desarrollo y utilización de la DCGAN propuesta, podría ahorrar los costes asociados al diseño y generación de imágenes de flores en las industrias del cine, videojuegos y moda. Pero donde más destaca es en la creación de conjuntos de datos de flores. Esto podría hacer más económico el hecho de recolectar miles de fotografías de flores, teniendo en cuenta el ahorro de los fotógrafos y el procesado de las imágenes.
- Medioambiental: El entrenamiento de modelos de aprendizaje automático, produce un alto costo energético asociado al consumo de electricidad de los componentes usados. Especialmente de las unidades de procesamiento gráfico (GPU), que son donde se realizan la mayoría de los cálculos necesarios. En mi caso específico, al no realizar el entrenamiento el local si no en la nube (Google Colab), también hay que tener en cuenta el coste asociado a la transferencia de datos entre mi ordenador y el servidor donde es ejecutado.

Teniendo en cuenta los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030, se recomienda que, según el séptimo objetivo, “Garantizar el acceso a una energía asequible, segura, sostenible y moderna”, la energía usada provenga de fuentes renovables. También mi trabajo cumple con el noveno objetivo, “construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación”. Esto es debido a que las GAN fueron inventadas recientemente, en el año 2014, por lo que su utilización y análisis suponen un gran avance en la innovación del aprendizaje profundo [4].

- Cultural: Las imágenes de flores generadas pueden ser utilizadas como inspiración para diseñadores y artistas en la producción de nuevos diseños de flores.

8 Bibliografía

- [1] F. Escolano Ruiz, M.A Cazorla Quevedo, M. I. Alfonso Galipienso, O. Colomina Pardo, M. A Lozano Ortega, Inteligencia Artificial. Modelos, Técnicas y Áreas de Aplicación, (2003) pp3-8.
- [2] C. Janiesch, P, Zschech, K.Heinich, Machine Learning and Deep Learning (2020), arXiv:2104.05314
- [3] J. Beckett, What's a Generative Adversarial Network? Inventor Explains, 2017 [Interview]. Available: <https://blogs.nvidia.com/blog/2017/05/17/generative-adversarial-networks/>. Último acceso en 30/02/2022
- [4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S.Ozair, A.Courville, Y. Bengio, Generative Adversarial Nets, NIPS, 2014. arXiv:1406.2661
- [5] D. P. Kingma and M. Welling , Auto-Encoding Variational Bayes, 2013. arXiv:1312.6114
- [6] Maind, Sonali B., and Priyanka Wankar, Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication* 2.1 (2014). pp 96-100.
- [7] D. Manrique, From Artificial Cells to Deep Learning. An Evolutionary Story. Archivo Digital UPM, 2021. <https://oa.upm.es/66864/>
- [8] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, 2019
- [9] S. B. Maind, P.Wankar Research paper on basic of artificial neural network. International Journal on Rece nt and Innovation Trends in Computing and Communication, 2014, vol. 2, no 1, p. 96-100.
- [10] Hecht-Nielsen R., Counter-propagation networks. *J. Of Applied Optics* 26(3), 1987
- [11] H.Chanampe, S. Aciar, M. de la Vega, J.L.M Sotomayor, G. Carrascosa, A. Lorefice, Modelo de redes neuronales convolucionales profundas para la clasificación de lesiones en ecografías mamarias, XXI WICC, 2019, <http://sedici.unlp.edu.ar/handle/10915/77381>
- [12] F. Li, J. Johnson, S.Yeung, (2017) Lecture 11 | Detection and Segmentation, [Video]. Available: <https://www.youtube.com/watch?v=nDPWywWRIRo>. Último acceso en: 30/01/2022
- [13] MasterClass staff, What is CGI? Examples of Computer-Generated Imagery in Film, 2022 [Article]. Available: <https://www.masterclass.com/articles/what-is-cgi> . Último acceso en 30/02/2022

[14] Diederik P. Kingma and Max Welling, An Introduction to Variational Autoencoders, Foundations and Trends in Machine Learning, 2019.
arXiv:1906.02691

Este documento esta firmado por



Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
Fecha/Hora	Tue May 31 15:48:04 CEST 2022
Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
Numero de Serie	561
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)