Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

---

CómicGAN:

Generación de ilustraciones con redes GAN de crecimiento progresivo

Doble Grado en Ingeniería del Software
y
Tecnologías para la Sociedad de la Información

Autor: Guillermo Iglesias Hernández
Director: Edgar Talavera Muñoz

Abril 2021

Agradecimientos

A Edgar, por ser un espejo donde mirarme, gracias por haberme enseñado desinteresadamente todo lo que sé, si esto ha sido posible es porque me has apoyado y confiado en mí.

A mis padres, mi hermano y mis abuelos, ellos son los que se han esforzado para que yo pueda hacer lo que quiero, espero no decepcionaros nunca.

A todos los compañeros de la carrera, en especial a Gonzalo, Yoel y Guille, con los que he compartido tantos momentos y a los que le pertenece cierta parte de todo lo que he hecho. A Diego y Raúl por todos los momentos juntos.

A toda mi gente del "parterre". Sois la parte más fundamental de mi vida, no se que haría sin vosotros. Os quiero mucho.

En general estoy agradecido a cualquier persona a la que haya querido en mi vida, sé que no soy nada sin la gente que me rodea. Gracias por estar ahí.

Resumen / Abstract

Resumen

El presente trabajo de fin de grado muestra la implementación de redes generativas adversarias (GANs) para generar ilustraciones completamente nuevas, haciendo uso de imágenes de cómic que previamente han sido estudiadas, normalizadas y filtradas. En la memoria se reflejará la evolución de la investigación, que toma como punto de partida los resultados realizados en un anterior TFG en el cual se plantean los primeros pasos a seguir para obtener un *dataset* válido.

Se presentan el conjunto de pasos para obtener el resultado final: la metodología de trabajo utilizada, la configuración de trabajo en remoto haciendo uso de *Google Colab*, la búsqueda y estudio de arquitecturas y la evolución y mejora de las distintas redes hasta logar los resultados finales.

Para llevar a cabo la generación de ilustraciones de cómic se implementan un conjunto de modelos de forma progresiva, llevando una evolución desde modelos más simples a más complejos y actuales, con el fin de asimilar mejor los conocimientos necesarios. De esta manera finalmente se propone el uso de redes generativas adversarias de crecimiento progresivo o ProGAN. Mediante la utilización de la arquitectura ProGAN se consiguen mejorar los resultados en comparación con el uso de métodos tradicionales llegando a obtener imágenes de gran similitud a las originales, manteniendo la originalidad evitando así la copia directa de imágenes del *dataset*.

A fin de validar y demostrar que los resultados obtenidos son replicables, se ha realizado una comparación entre dos conjuntos de imágenes diferentes. A pesar de que ambos *datasets* cuentan con el mismo estilo de dibujo, presentan diferencias significativas en su composición. Por una parte se muestran resultados para imágenes de personajes con el cuerpo entero y posteriormente haciendo uso de ilustraciones de caras de personajes en primer plano.

Abstract

This final degree work shows the implementation of generative adversarial networks (GANs) to generate completely new illustrations, making use of comic images that have been previously studied, normalized and filtered. The report will reflect the evolution of the research, which takes as a starting point the results of a previous research in which the first steps to follow in order to obtain a valid *dataset* are proposed.

The set of steps to obtain the final result are presented: the work methodology used, the remote work configuration using Google Colab, the search and study of architectures and the evolution and improvement of the different networks to achieve the final results.

To carry out the generation of comic illustrations, a set of models are implemented progressively, taking an evolution from simpler to more complex and actual models, in order to better assimilate the necessary knowledge. In this way, the use of progressively growing generative adversarial networks or ProGANs is finally proposed. By using the ProGAN architecture, it is possible to improve the results compared to the use of traditional methods, obtaining images of great similarity to the original ones, maintaining the originality and avoiding the direct copy of images from the dataset.

In order to validate and demonstrate that the results obtained are replicable, a comparison between two different sets of images has been performed. Although both *datasets* have the same drawing style, they present significant differences in their composition. On the one hand, results are shown for images of characters with the whole body and later using illustrations of character faces in the foreground.

Índice de contenidos

Índice de figuras

Introducción

1.1. Trasfondo, ¿cómo hemos llegado hasta aquí?

Las investigaciones científicas dentro del área de la inteligencia artificial es uno de los campos más fructíferos e interesantes de los últimos años en el mundo de la informática. Los avances logrados en esta materia impulsan a pasos agigantados su desarrollo, logrando resultados nunca antes vistos, que hacen replantearse a la humanidad los límites de la tecnología.

La idea de realizar una investigación en el campo tiene especial interés, pues supone lograr resultados relacionados directamente con la vanguardia tecnológica.

Todo esto despierta un interés por realizar una investigación profunda en un proyecto de *Deep Learning*, con el que aprender y afianzar las bases para una futura linea de investigación en una materia que es, en la actualidad, una de las más relevantes en la ciencia.

1.1.1. Investigación en el área de la Inteligencia Artificial

El paradigma actual de la investigación científica se sustenta en la coexistencia e interacción de diferentes disciplinas que, gracias a trabajar conjuntamente, consiguen progresos en sus diferentes campos. Cuando hoy en día se desarrollan proyectos de investigación, en ellos participan profesionales de distintas áreas para lograr un objetivo común.

En este contexto, la inteligencia artificial de hoy en día forma parte de todo tipo de proyectos de diferentes áreas, gracias a su gran capacidad de adaptación y eficiencia. Sin embargo, no son solo el resto de disciplinas las que se benefician de la inteligencia artificial, puesto que muchos de los grandes hitos logrados en los últimos años por la inteligencia artificial son frutos de proyectos relacionados con otras ramas del conocimiento.

Este ecosistema en el que las diferentes disciplinas científicas hacen uso unas de otras es idóneo para el desarrollo de proyectos relacionados con el campo de la inteligencia artificial, como el *Deep Learning*[1]. Cualquier avance que se desarrolle en el campo es susceptible en un futuro de ser reutilizado en otro problema completamente diferente. Esta característica propia del *Deep Learning* propicia el surgimiento de proyectos que, aunque por su propia finalidad no sean especialmente interesantes, sean muy importantes en el sector de la investigación pues, si se consiguen mejoras en los sistemas de aprendizaje automático, dichas mejoras pueden ser fácilmente trasladadas a otros problemas nuevos, logrando mejorar sus resultados anteriores.

[1] El *Deep Learning* es la disciplina de la inteligencia artificial que crea sistemas capaces de aprender automáticamente a resolver problemas complejos. La principal diferencia entre *Deep Learning* y *Machine Learning* es que el *Deep Learning* consigue resolver problemas más complejos gracias a la mayor dimensión y profundidad de sus redes neuronales.

Es por esto que en la actualidad el *Deep Learning* se postula como una de las técnicas científicas que mejores resultados da y más versátil es. No es difícil encontrar algún tipo de aplicación de aprendizaje automático en proyectos de ramas de conocimiento como la química[1], biología[2], sociología[3], lingüística[4] o física[5] entre otras. Los resultados de este tipo de investigaciones impulsan el desarrollo en inteligencia artificial.

La figura 1 muestra los resultados del proyecto *Alphafold*, desarrollado por la empresa *DeepMind*, cuyo propósito es el de predecir el plegamiento de proteínas basándose en los aminoácidos de su estructura. La primera y segunda fila de la imagen corresponden con una representación visual de la distancia entre los aminoácidos de la proteína siendo la primera fila la representación real de la distancia entre cadenas de aminoácidos y la segunda la predicha por la inteligencia artificial, la tercera fila es la representación tridimensional de dichos resultados siendo la figura en verde la real y la azul la predicha. Los resultados de esta inteligencia artificial son especialmente importantes ya que logran mejores resultados que otro tipo de aplicaciones tradicionales:

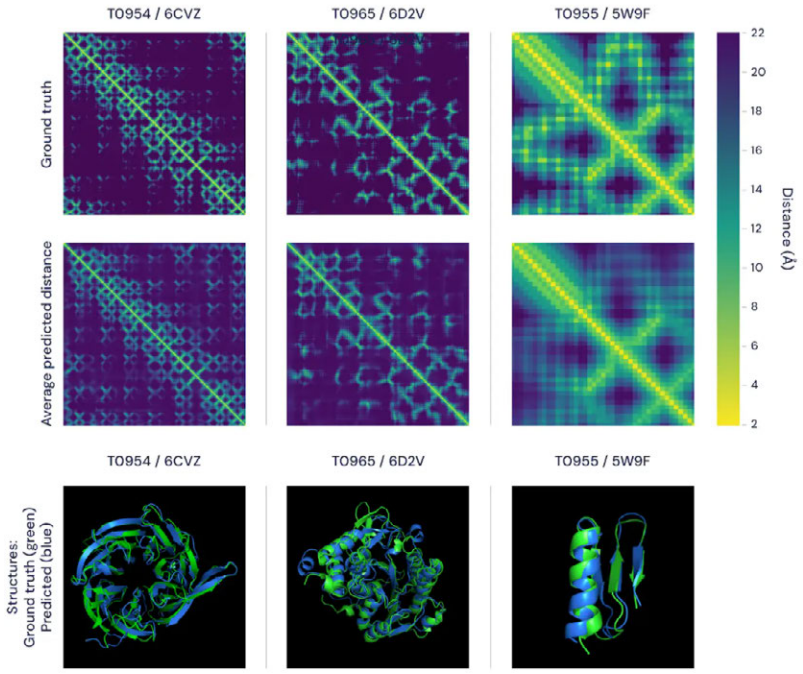


Figura 1: Resultados de la inteligencia artificial *Alphafold* cuyo propósito es predecir el plegamiento de proteínas.

Esta situación en la que se encuentra el aprendizaje automático hoy en día, hace que los proyectos que surjan relacionados con la investigación en el campo del *Deep Learning* tengan una mayor profundidad e importancia de la que una persona no entendida en la materia pueda entender. Como ya se ha mencionado anteriormente, cualquier avance que se realice en la materia, por muy pequeño que pueda parecer, propicia un impulso del cual se pueden beneficiar a su vez otras investigaciones, aunque sus objetivos finales sean completamente opuestos.

1.1.2. ¿Cómo se logran los avances en *Deep Learning*?

El *Deep Learning* es un concepto relativamente reciente, ha surgido mediante la evolución del campo de la inteligencia artificial que ha llevado a la obtención de grandes resultados. Durante los últimos años está sufriendo una gran cantidad de avances que hacen que, lo que antes parecía imposible lograr, ahora es factible. No podemos olvidar que la investigación en *Deep Learning* se lleva realizando muy pocos años en comparación con otras disciplinas científicas y, por lo tanto, los avances en la materia se realizan de manera muy rápida.

Los proyectos que se realizan hoy en día en este campo hace escasos años eran impensables, debido a la gran velocidad de desarrollo en el aprendizaje automático de la última década. Arquitecturas como las redes generadoras adversarias o GAN[2] han sido postuladas recientemente en el año 2014[6], propiciando así el surgimiento de nuevos proyectos anteriormente inviables por el estado de la técnica de ese momento. En la figura 2 se puede observar cómo, con los avances realizados en la última década, se han logrado perfeccionar tareas como la generación de caras humanas:

Figura 2: Evolución de los resultados en los últimos años ante el problema de generar imágenes de caras humanas con el uso de inteligencia artificial.

[2]GAN: Generative Adversarial Networks

Otro de los factores a considerar, propios del aprendizaje automático, es que es muy difícil para los investigadores comprender el funcionamiento exacto de las diferentes redes que se crean por su gran complejidad. Debido a esto, uno de los mayores problemas con los que tienen que lidiar los investigadores hoy en día, es la inestabilidad y poco control sobre las redes que se crean[3]. Este problema se intenta solucionar hoy en día intentando crear nuevas arquitecturas, capaces de minimizar la inestabilidad y variabilidad en los resultados a la hora de crear y entrenar las redes neuronales.

Dicho esto, el estado actual de las distintas investigaciones enfocadas al campo del *Deep Learning*, sugieren nuevas y variadas arquitecturas y modelos de entrenamiento que consigan minimizar o incluso eliminar la variabilidad entre diferentes entrenamientos.

1.1.3. Origen y finalidad de esta investigación

El desarrollo del presente Trabajo de Fin de Grado (TFG) surge como la continuación del trabajo final de grado realizado anteriormente titulado *Procesamiento automático de ilustraciones: Clasificación multi-etiqueta de cómics con Deep Learning*[7], en el cual se sentaban las bases de la investigación presente. El objetivo final de la investigación es desarrollar redes neuronales capaces de crear un cómic a través del uso de *Deep Learning*.

Durante el desarrollo del primer TFG se realizó una labor de investigación, por la cual se obtuvo un dataset de imágenes lo suficientemente amplio como para realizar diferentes entrenamientos con él. Para estudiarlo, familiarizarse en profundidad con su contenido y poder hacer un buen uso de él se estudió su composición, se normalizó, se optimizó, limpió de ruido su contenido, y por último se realizó una aproximación a un trabajo de *Deep Learning* usando como entrada dicho dataset para la clasificación de elementos en una imagen. Un resultado de dicho trabajo se puede observar en la figura 3. Con este trabajo se constató que las bases para el desarrollo del presente trabajo estaban bien asentadas y que el dataset obtenido se podía reutilizar en otro tipo de aplicaciones.

[3] El principal problema de la inestabilidad viene cuando, al realizar el proceso de entrenamiento de las redes, es imposible controlar aspectos como la probabilidad de éxito. A esto se suma que para diferentes entrenamientos de una misma red con igual configuración los resultados obtenidos con cada uno de ellos sean cualitativamente muy diferentes.

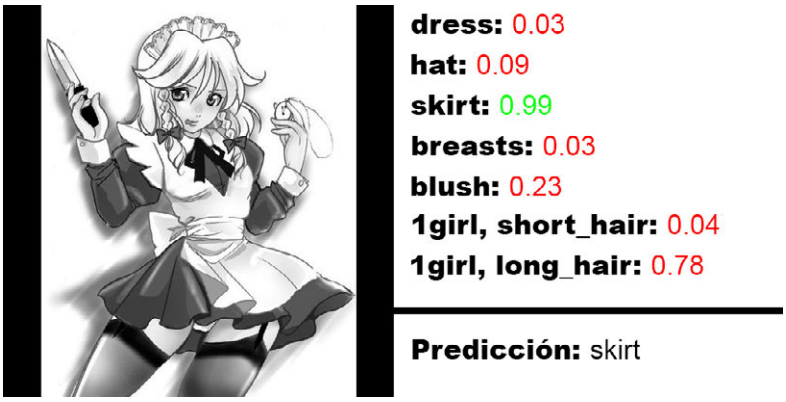


Figura 3: Resultados del desarrollo del anterior TFG[7] para la clasificación de 7 elementos en una imagen.

Una vez realizado el trabajo previo, el presente TFG se centra la continuación de la investigación, intentando replicar las imágenes del dataset a través del uso de redes GAN, que permitan crear imágenes de estilo de cómic con inteligencia artificial, continuando con la investigación y acercándose al objetivo final de la replicación total de un cómic con el uso de *Deep Learning*.

El desarrollo del presente TFG toma como bases la investigación científica, y por lo tanto no se centrará únicamente en el desarrollo de redes GAN para la creación de imágenes, sino que una parte muy grande de los esfuerzos realizados se basan en la búsqueda y estudio de nuevas arquitecturas capaces de adaptarse a nuestro problema. Haciendo uso así de modelos de entrenamiento novedosos cuya aplicación pueda dar valor extra a la investigación, puesto que gracias a ellos se obtendrán mejores resultados.

1.2. Motivaciones

Debido a que la realización del doble grado ha despertado una inquietud por ampliar los conocimientos adquiridos, la investigación plantea un reto mayor en el cual poder poner a prueba todo lo aprendido y aumentar mis conocimientos, poniendo a prueba mis capacidades como profesional. Creo que actualmente estamos en una posición privilegiada históricamente en el sector de la informática, y más aún en el campo de la inteligencia artificial por los motivos que se han comentado en la sección 1.1.1.

La investigación es para mí un ecosistema en el que poder desarrollarme completamente como profesional, ahondando profundamente en materias que siempre me han interesado y pudiendo desarrollar proyectos a la vez que adquiero conocimientos que aportan gran valor a mi carrera. Por ello, la posibilidad de embarcarme en un proyecto enfocado a la investigación, es la causa principal por la que se eligió el proyecto presente.

A la hora de elegir un proyecto, me parecía más interesante empezar desde cero teniendo que lidiar con problemas inesperados y aprendiendo en el camino cómo es trabajar en una investigación directamente, y no con un proyecto simulado el cual eliminaría gran cantidad de problemas.

Este proyecto en general nació en el año 2019, y el presente TFG corresponde con la continuación del trabajo realizado en el TFG titulado *Procesamiento automático de ilustraciones: Clasificación multi-etiqueta de cómics con Deep Learning*[7], el principal motivo del trabajo presente es, en concreto, continuar con la labor de investigación, siguiendo con el proyecto y avanzar en él tal y como desde un primer momento se había proyectado.

1.3. Objetivos

Debido a la ambición y la naturaleza de este proyecto, se han ido planteando diferentes hitos que cumplir según se avanzaba en la realización del mismo. Por ello, se han dividido los objetivos planteados para el trabajo en dos tipos diferentes.

1.3.1. Objetivos principales

Los objetivos principales del presente trabajo son:

- **Investigar y desarrollar redes GAN *vanilla*[4]:** Para lograr crear imágenes de cómic y replicar las imágenes presentes en nuestro dataset será necesario la creación y entrenamiento de redes generativas adversarias. Estas redes tomarán como datos de entrada para los entrenamientos las imágenes reales de nuestra base de datos, e intentarán aprender las características de estas para crear nuevas imágenes no existentes anteriormente. Estas redes sentarán las bases, para posteriormente intentar ser optimizadas y mejoradas.
- **Buscar arquitecturas para mejorar los resultados:** Una vez se hayan probado los modelos más simples y observado los diferentes resultados obtenidos, se buscarán nuevas arquitecturas que se adapten lo mejor posible a nuestro problema.
- **Reutilizar los conocimientos previos adquiridos:** Para realizar el presente trabajo será necesario usar los resultados ya adquiridos, que sientan las bases sobre las cuales proseguir con la investigación. Conseguir integrar estos resultados con nuestras futuras investigaciones sirve para no desperdiciar todo el trabajo realizado y usarlo para aumentar la posibilidad de éxito.

[4]Vanilla es el término comúnmente utilizado en la literatura para definir la forma más sencilla y sin modificar de algo. En el caso de las redes GAN corresponde con la arquitectura más sencilla y simple.

1.3.2. Objetivos secundarios

Los objetivos secundarios del presente trabajo son:

- **Seguir una metodología propia de la investigación:** El marco en el que se desarrolla el trabajo es el de la investigación, y por lo tanto, se considera especialmente interesante que a la hora de desarrollar los distintos pasos, estos se realicen con plena consciencia y con la intención de asemejarse lo máximo posible a una investigación científica. Al hacer esto se conseguirán adquirir unas destrezas, que en un futuro puedan ser útiles a la hora de desarrollar investigaciones.
- **Familiarizarse con entornos de desarrollo en remoto:** Durante la investigación anterior se vio el problema de la falta de recursos computacionales a la hora de realizar entrenamientos con redes neuronales. Para paliar este problema, se buscaron entornos de desarrollo en remoto. Para poder desarrollar de manera completa la actividad investigadora será necesario seguir usando dichos entornos y conseguir un mayor conocimiento en cómo estos funcionan, sus ventajas y desventajas.
- **Adquirir un conocimiento profundo de redes generativas adversarias:** Debido a la poca carga lectiva durante toda la carrera en temas relacionados con la inteligencia artificial, las redes GAN ni siquiera son nombradas. Para poder hacer uso de ellas, primero es necesario tener conocimientos en la materia para no limitarse a reutilizar modelos ya creados y probados. Este conocimiento servirá para ver con un ojo crítico todas las posibilidades de arquitecturas y escoger la que más se pueda adaptar a las necesidades de la investigación, adecuando la solución al problema en cuestión. Cabe destacar que hay muchos de los aspectos que no son exclusivos de las redes GAN, sino que son comunes a cualquier tipo de red neuronal y por lo tanto se necesitan conocimientos en el campo del *Deep Learning* en general.

Estado del arte

2.1. *Deep Learning*

Antes de comenzar a estudiar estructuras complejas de aprendizaje automático es necesario hacer un breve repaso sobre los fundamentos y teoría básica de este área. Se considera beneficioso realizar una primera toma de contacto con el área del *Deep Learning* en la que se estudie las características principales de este área para posteriormente ir especializando los conocimientos.

Con ello se sientan unas bases sobre las que posteriormente se pueden estudiar estructuras avanzadas de *Deep Learning*.

2.1.1. Fundamentos del *Machine Learning*

El aprendizaje automático es el conjunto de técnicas y algoritmos utilizados en inteligencia artificial para realizar tareas complejas de manera automatizada. Su funcionamiento se basa en estructuras matemáticas, capaces de autorregularse, que aprenden a través de observar ejemplos del problema que quieren tratar. Dicho de otra manera, el aprendizaje automático funciona gracias a redes que aprenden a solucionar un problema concreto, a través del aprendizaje del mismo tras exponerse al problema y aprender de él.

Estas redes fundamentadas en estructuras matemáticas son las llamadas redes neuronales y están estructuradas en un sistema de capas, en el que cada capa se conecta con una anterior y posterior habiendo una capa de entrada en la que se recibe el problema y una de salida que se encarga de la solución del mismo. La figura 4 muestra un esquema general de la estructura típica de las capas de una red neuronal:

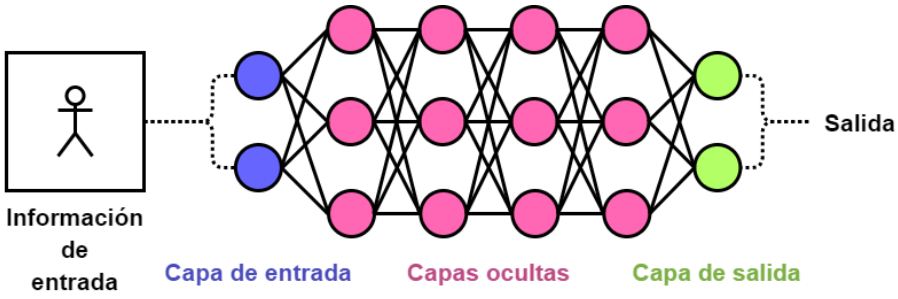


Figura 4: Estructura general de capas de una red neuronal.

Como se puede observar en la imagen, las redes neuronales se componen de capas constituidas a su vez por las llamadas neuronas. Dichas neuronas son, de manera simplificada, funciones matemáticas con parámetros que cambian llamados pesos. Cada neurona tiene un peso que sirve para definir su salida, al cambiar ese peso cambia el comportamiento de la salida de la neurona.

Las neuronas se usan para procesar la información y, poco a poco, conformar la salida de la red. Por su parte el cambio de sus pesos es el que configura las redes para tratar cada problema. Dicho de otro modo, las capas de neuronas transmiten la información y generan una salida para la red, por su parte al regular su comportamiento se puede hacer que una red aprenda a resolver un problema concreto, configurando los pesos de cada neurona.

2.1.2. Diferencias entre *Machine Learning* y *Deep Learning*

Una vez comprendido el fundamento general de una red de neuronas, hay que atender de nuevo a la estructura de capas de las redes neuronales. Como se ha explicado, cada neurona individual se encarga de procesar cierta parte de la información original y, es al procesarse por todas las neuronas, cuando se consiguen extraer características del problema a solucionar.

Como se ha explicado anteriormente, cada una de las capas se conecta con la anterior y la siguiente a ella, más concretamente las neuronas de dicha capa son las que se conectan hacia un lado y al otro, pero en una misma capa las neuronas no se conectan unas con otras. De forma general se puede decir que cada capa se encarga de procesar la totalidad de la información que le llega y generar una nueva información de salida. Es este proceso de transmisión de información entre capas el que va definiendo la salida buscada en las redes, cada capa se encarga de extraer ciertas características de la información que recibe y transmitírsela a la siguiente capa.

El *Machine Learning* tiene su origen y sus bases teóricas en la biología. El origen de esta disciplina de la inteligencia artificial es la imitación del funcionamiento del cerebro de los animales para intentar replicar su comportamiento.

En trabajos de neurobiología como el titulado *Receptive fields of single neurones in the cat's striate cortex*[8] se expone que las primeras capas neuronales de la visión de un gato son las encargadas de procesar información a bajo nivel[5]como líneas y, a medida que se suceden las capas, estas se encargan de procesar elementos de mayor nivel como formas más complejas.

La teoría propuesta en estos trabajos enfocados a la neurobiología se ha visto confirmada a través de investigaciones como la titulada *Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks*[9] publicada en el año 2011, que pone de manifiesto las características que identifica cada capa de distintas redes neuronales. En la figura 5 se puede observar uno de los resultados de dicha investigación:

[5]La información de una imagen a bajo nivel son aquellos detalles más pequeños como bordes o texturas mientras que la información de alto nivel son elementos como ojos, caracteres o un animal completo. A medida que aumenta el nivel de la información se hace referencia a elementos más grandes compositivamente y más complejos.

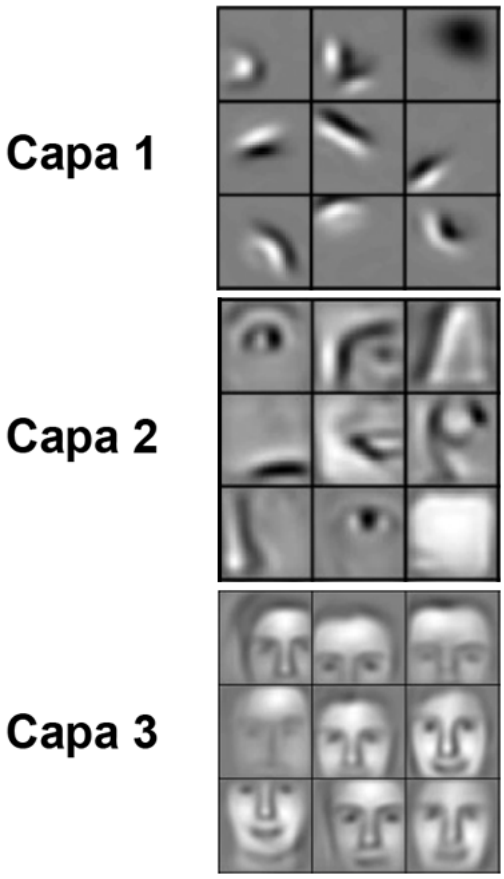


Figura 5: Ejemplo de características aprendidas por las capas de una red neuronal para imágenes de caras humanas.

Aquí es donde se puede hacer la diferencia entre *Machine Learning* y *Deep Learning*, una vez observado el comportamiento de la información a través de las capas, se puede concluir que a mayor número de capas, más compleja es la información que se puede tratar. Las redes de *Deep Learning* cuentan con un número elevado de capas, lo que hace que puedan tratar problemas más complicados y, por eso, son redes con las que, en general, se obtienen mejores resultados. Como contrapartida a mayor número de capas mayor tamaño de red, lo que ralentiza el proceso de aprendizaje.

2.1.3. Cuando el machine learning da paso al *Deep Learning*

Como se ha explicado anteriormente, las bases del *Deep Learning* son las mismas en las que se sustenta el *Machine Learning*. La diferencia básica entre ambas disciplinas es la capacidad de las redes de solucionar problemas más o menos complejos debido a la complejidad de su estructura.

Las redes que se consideran como *Deep Learning* son las que cuentan con un número elevado de capas, estas capas proporcionan a la red la capacidad de aprender un mayor número de características del problema que se está tratando. En general, al tener redes más grandes y más profundas, se consiguen mejores resultados y poder tratar problemas que con redes menores no se podría.

Hay muchos tipos de arquitecturas diferentes de *Deep Learning*, cada una de ellas está especializada en un aspecto y se usa para tratar cierto tipo de problemas. Todas estas nuevas arquitecturas y sus variaciones tienen su origen en investigaciones en las que se proponen nuevos modelos de redes neuronales para tratar o mejorar los resultados de cierto problema. Una de estas arquitecturas dedicada a la generación de información es la conocida como *Adversarial Neural Networks* o GAN.

2.2. Redes generativas adversarias (GAN)

Las redes generativas adversarias o *Adversarial Neural Networks* es una arquitectura de redes neuronales, capaces de generar información nueva con las mismas estadísticas que la información con la que se realiza el entrenamiento. La arquitectura se compone de dos redes separadas, llamadas red generadora y red discriminadora, que realizan un entrenamiento en forma de juego de suma cero[6]. Dicho entrenamiento se basa en que una red se encarga de engañar a la otra intentando crear información falsa para que la otra red la identifique como real.

La figura 6 muestra un esquema general de la composición y estructura de las distintas redes de una red GAN:

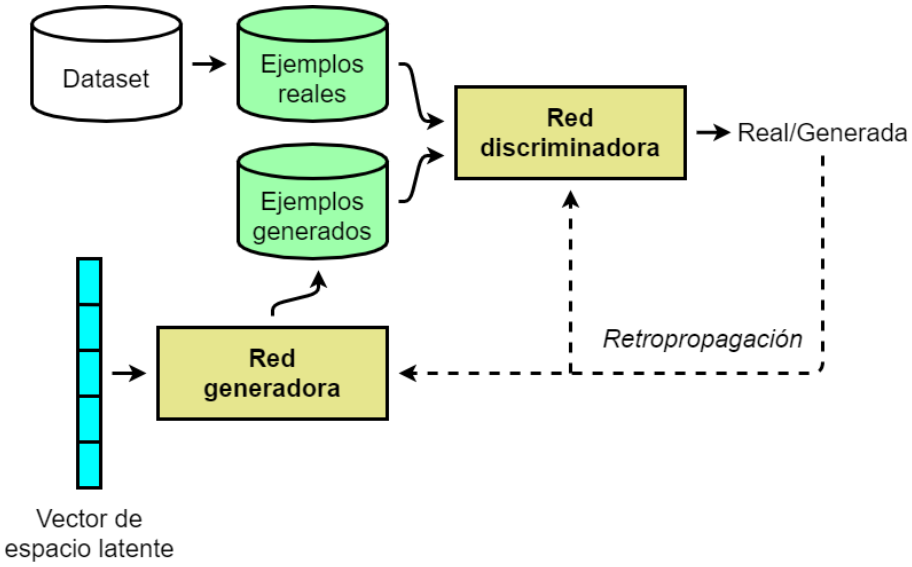


Figura 6: Esquema de estructura general de una arquitectura GAN.

[6]El juego de suma cero es aquella representación matemática en los diferentes actores tienen pérdidas o ganancias de modo que la suma total de pérdidas y ganancias entre todos los actores es cero. Dicho de otra manera las pérdidas o ganancias que sufre un participante se equilibran con la del resto para que siempre haya el mismo valor.

2.2.1. Composición básica de una GAN

Como se ha mencionado anteriormente, la estructura más básica de una arquitectura GAN se compone por dos redes:

- **Red generadora:** La función de esta red es crear nueva información para que se asemeje lo más posible a la información de la realidad. Su entrada es un vector formado por un número variable (normalmente 100) de números decimales entre 0 y 1, llamado *vector de espacio latente*. Dicho conjunto de números cumple la función de servir como entrada de la red. Por cada *batch*[7] del entrenamiento se cambia el valor del espacio latente, de forma que se introduce variabilidad a la entrada de la red permitiéndole aprender diferentes características de la información que se quiere replicar. De esta forma cada uno de los números representará una característica de la información tras completar el entrenamiento de la red. La estructura de la red es variable dependiendo del problema que se quiera solucionar y su composición[8]. Como salida de la red se intentará reconstruir la información que se pretende generar. La figura 7 recoge un esquema de la estructura a alto nivel definida para una red generadora general.

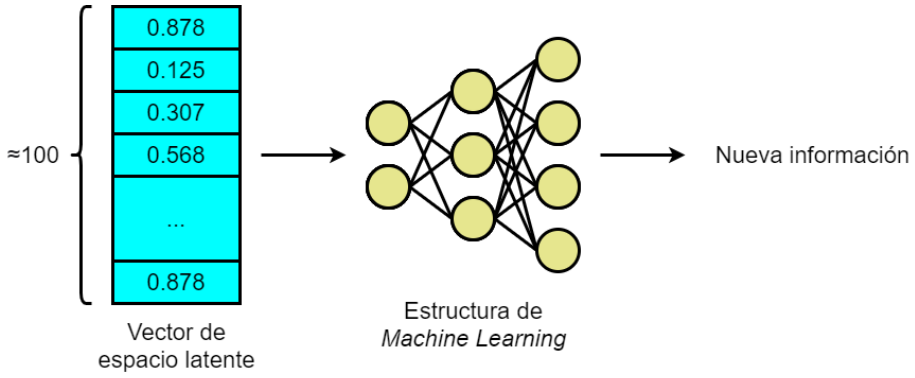


Figura 7: Esquema de estructura general de la red generadora.

[7] Un *batch* es cada una de las iteraciones con las que se realiza el entrenamiento de una red neuronal, a grandes rasgos en un *batch* se le introduce a la red un conjunto de información de entrada y se evalúa el resultado de salida de la red para ajustar los pesos de cada neurona, dependiendo de la validez de la salida.

[8] La composición de una red neuronal se compone del conjunto de hiperparámetros y estructura que se le quiera dar a una red neuronal. Esta se compone entre otros del número de capas que la componen, las funciones de activación, la función de pérdidas o el *learning rate*.

- **Red discriminadora:** Esta red es la encargada de evaluar los resultados obtenidos como salida de la red generadora, para que posteriormente esta evaluación sirva para actualizar los pesos de esta. Para actualizar los pesos de la red generadora la red discriminadora realiza una evaluación sobre la información generada por la red generadora, esta información se evalúa y se refuerza positivamente a la generadora si ha sido clasificada como información real. Por otra parte se refuerza negativamente si ha sido clasificada como información falsa. Durante este proceso de retroalimentación los pesos de la red discriminadora se congelan para evitar la inestabilidad en el entrenamiento. Por su parte la red discriminadora actualiza sus pesos recibiendo información generada por la red generadora (información falsa) e información del *dataset* (información real), su función es diferenciar qué información es real y cuál ha sido generada por inteligencia artificial. La figura 8 recoge un esquema de la estructura a alto nivel definida para una red generadora general.

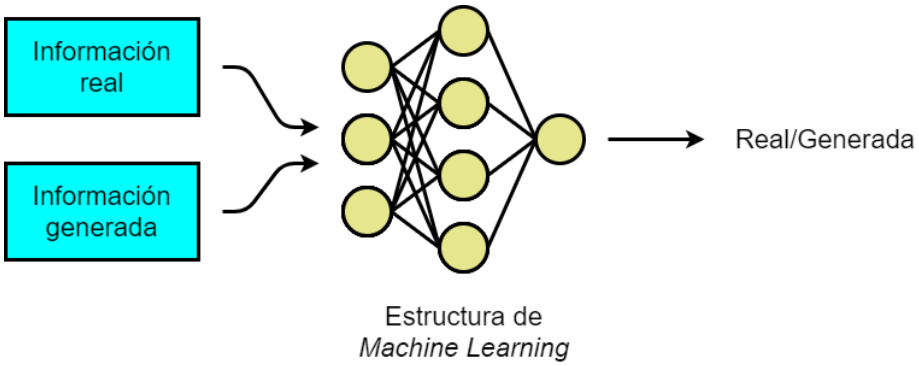


Figura 8: Esquema de estructura general de la red discriminadora.

2.2.2. Estructura básica de un entrenamiento de una GAN

Para entrenar las dos redes definidas anteriormente hay que definir una estructura compleja con el fin de poder actualizar los pesos de ambas redes al mismo tiempo.

Debido a que la red generadora tiene como salida una información que no puede ser directamente evaluada porque es completamente nueva, hay que hacer uso de la red discriminadora para actualizar los pesos de esta red. Para evaluar la salida de la red generadora se toma esta información y se le introduce como entrada a la red discriminadora.

Esta red puede obtener como salidas posibles un 0 (información falsa) o un 1 (información verdadera), si se consigue "engañar" a la red y se obtiene como salida un 1 esto se toma como un resultado favorable, ya que la información ha sido clasificada como verdadera y por lo tanto se parece a la información que se quiere replicar. Si el resultado de la clasificación ha sido un 0, se toma como resultado malo porque la red discriminadora ha sabido localizar la información generada artificialmente como falsa.

El entrenamiento de la red discriminadora se realiza de forma separada, teniendo como información de entrada información real del dataset e información falsa generada por la red generadora[9]. La red discriminadora se encargará de clasificar esa información como falsa o real y se actualizarán sus pesos dependiendo de si ha acertado o no.

La figura 9 muestra un esquema de cómo se realiza el entrenamiento de una estructura GAN, como se puede observar la red discriminadora se usa junto a la generadora para actualizar los pesos de esta mientras que, por su parte, la red discriminadora entrena usando como información la del *dataset* y la generada por la red generadora.

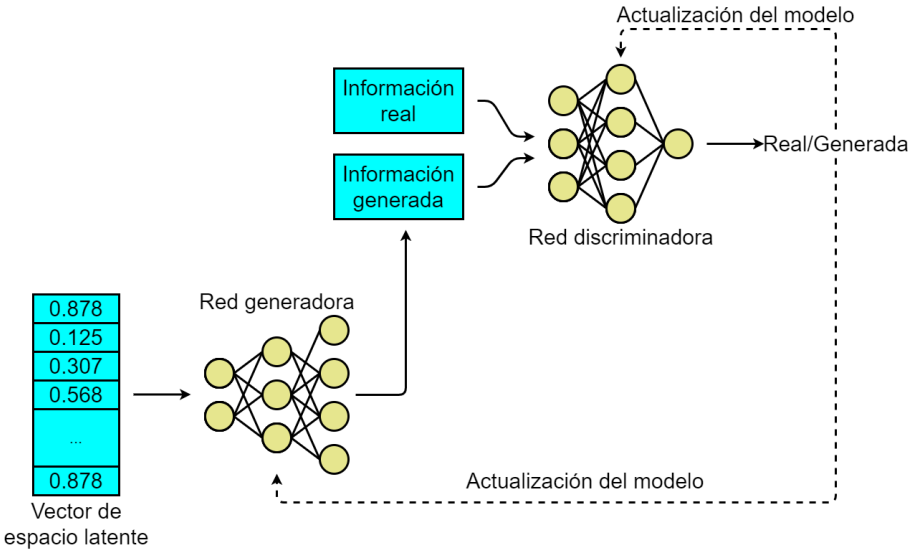


Figura 9: Esquema del entrenamiento de una estructura GAN.

[9]Esta información de la red generadora se crea con el único propósito de entrenar la red discriminadora y no está relacionada con la información generada en el entrenamiento de la red generadora.

Este juego de engañar y descubrir entre la red generadora y discriminadora es el que hace avanzar el entrenamiento, pues si la red discriminadora no diferencia bien la información real de la generada artificialmente la red generadora no recibirá una retroalimentación que le sirva para poder progresar correctamente. Por otro lado si la red discriminadora diferencia siempre qué información es real y cuál no se produce un estado en el que la red generadora es incapaz de engañar a la discriminadora y por lo tanto nunca recibe una retroalimentación positiva de sus creaciones. De esta manera le es imposible avanzar puesto que nada de lo que es capaz de generar se identifica como real.

Esta complejidad que se puede ver en la estructura de los entrenamientos de las redes GAN tiene como resultado una gran inestabilidad en las redes, ya que como se ha definido para ir consiguiendo cada vez mejores resultados hay que mantener un equilibrio entre las redes que componen la arquitectura y dicho equilibrio no es fácil de encontrar.

2.3. Posibles problemas de las redes GAN

Debido a la complejidad de las redes generativas por su estructura y modelo de entrenamiento, como se ha explicado en la sección 2.2, estas pueden sufrir problemas que no son tan comunes en estructuras de *machine learning* tradicionales. Durante esta sección se hará un repaso de los tipos de problemas más comunes que pueden ser relevantes para nuestra investigación.

2.3.1. Red discriminadora demasiado certera

Debido a la estructura con la que se entrena una red GAN, se puede dar el caso en el que la red discriminadora quede sobreentrenada, lo cual puede llevar a problemas como anteriormente se ha descrito en la sección 2.2.2.

Una red discriminadora con una tasa de error demasiado baja podrá diferenciar casi siempre qué información es verdadera y cuál ha sido generada artificialmente. Esto provoca que la red generadora siempre tenga una retroalimentación negativa y actualice sus pesos en consecuencia. En un principio esta actualización es completamente normal, pero si se mantienen en el tiempo la red generadora será incapaz de generar información que engañe a la discriminadora y, en última instancia, intentará sin éxito mejorar quedando estancada o empeorando sus resultados en el peor de los casos.

En un primer vistazo, parece una posibilidad muy remota que esta situación llegase a suceder, ya que de un momento a otro es muy difícil que la red discriminadora aumente considerablemente su tasa de acierto y la red generadora sea incapaz de seguir mejorando. Pese a esta intuición, cabe destacar que el problema de clasificación al que se enfrenta la red discriminadora es mucho más sencillo que el de creación de la red generadora.

Por la propia naturaleza del problema es más difícil generar una información nueva que diferenciar entre una real y una falsa. Además durante las primeras fases del entrenamiento la información generada artificialmente y la real serán completamente distintas, ya que la red generadora no estará apenas entrenada y durante sus primeras iteraciones sus creaciones serán prácticamente ruido, esto hace aún más sencilla la tarea de diferenciar dichas creaciones de la información real.

Otra característica de este problema es, que si la red discriminadora comienza a tener muy buenos resultados en sus clasificaciones, se actualizará en consecuencia y se volverá mejor clasificando. Estas clasificaciones al ser mejores generarán que la red aún aprenda más rápido y así creando una sucesión de mejoras que hace que la red discriminadora se desmarque completamente de la generadora, agravando el problema descrito anteriormente.

2.3.2. Alta inestabilidad en el entrenamiento

La interacción entre las redes neuronales que forman parte de una arquitectura GAN tiene como principal problema una gran inestabilidad. Esta inestabilidad se manifiesta de modo que a la hora de entrenar las redes que componen nuestros modelos, estas son muy difíciles de coordinar para que avancen simultáneamente.

Cuando se realiza un entrenamiento y se producen intervalos de mucha inestabilidad, se puede observar cómo las pérdidas[10] de las redes comienzan a tener valores muy extremos, ya sean por ser muy elevados o muy bajos. Esto provoca una reacción en cadena, por la cual la inestabilidad provoca más inestabilidad y hace que se prolongue en el tiempo este problema. Esto es porque si una red cambia muy rápidamente la otra tiene que adaptarse a ese cambio brusco.

Este periodo puede prolongarse indefinidamente, impidiendo la mejora de las redes que se pretenden entrenar, o incluso revirtiendo el entrenamiento realizado. La principal razón de esto es que estos cambios tan rápidos en las redes hacen que sus pesos cambien bruscamente y esto no siempre supone una mejora, al contrario. Lo que se busca es una mejora progresiva y controlada, ya que de modo contrario es impredecible el resultado del entrenamiento y, lo más probable es que el resultado sea malo.

[10] La función de pérdida se utiliza en los entrenamientos de redes neuronales para indicar la tasa de error de las mismas, dicho error se traduce en un cambio en los pesos de las redes entrenadas a través del cálculo del gradiente que minimice el valor de dicho error. En otras palabras el valor de la función de pérdida o *loss function* indica el buen o mal desempeño de una red neuronal para cada iteración de un entrenamiento.

2.3.3. El problema de la parada

Tradicionalmente los entrenamientos en *machine learning* se realizan hasta que la tasa de error de la red llega a un límite inferior concreto. Pese a que este modelo tenga algunos problemas como el del *overfitting*[11], el punto en el cual dejar de entrenar es claro y se define previamente. En una situación ideal el error de una red neuronal tradicional seguiría una función monótona decreciente, ya que a medida que entrena su tasa de acierto es mayor y su pérdida menor.

Interpretar el error y la pérdida en una arquitectura GAN es más complejo que en una estructura tradicional de *machine learning*, ya que hay varias redes interactuando conjuntamente. Cada una de las redes que forma parte de nuestro sistema tiene su propia función de pérdida, pero como las redes interactúan unas con otras las funciones de pérdida respectivas estarán influenciadas por el resto de redes en cierta medida. Por ejemplo, en la pérdida de la red generadora hay que tener en cuenta que dicha pérdida depende de la certeza del discriminador, en otras palabras, que la pérdida sea muy pequeña no significa que la información generada sea muy parecida a la real, sino que la información generada es muy difícilmente diferenciada de la real por parte el discriminador.

Para decidir el punto en el que dejar de entrenar las redes no hay una metodología concreta, por lo tanto lo más habitual es ir observando cómo mejora la calidad de la información generada manualmente y decidir de manera subjetiva el punto en el cual terminar el entrenamiento. Lo más habitual es ir guardando los distintos estados de las redes a medida que se realizan los entrenamientos, de modo que se seleccionan los que se consideren más adecuados.

2.4. *Gradient Explosion* y *Gradient Vanishing*

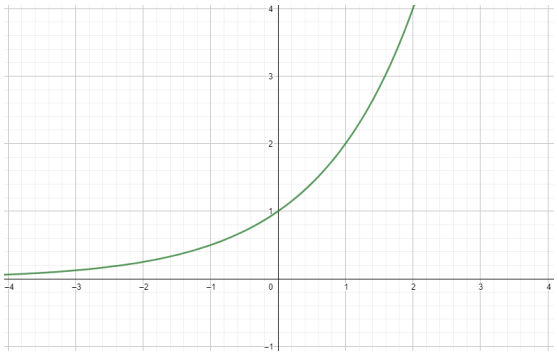
Este problema afecta a cualquier tipo de red neuronal. Al realizar un entrenamiento con una red se calcula el error de una iteración, a dicho error le corresponde un gradiente que indicará el valor a cambiar en cada uno de los pesos de los parámetros de la red. Esta actualización de pesos se realiza capa a capa en el proceso que se conoce como retropropagación o *Backpropagation*. Esta nueva configuración de pesos busca corregir el error producido en la iteración del entrenamiento en cuestión y puede ser más o menos brusca en los pesos dependiendo del error.

[11] El *overfitting* se produce cuando una red neuronal es entrenada hasta cierto punto en el que deja de generalizar el problema y busca soluciones específicas para los casos con los que realiza su entrenamiento, es decir, la red se aprende de memoria los casos con los que ha sido entrenada. Este problema provoca que la red neuronal mejore sus resultados durante el entrenamiento pero no sea capaz de obtener buenos resultados con otros datos.

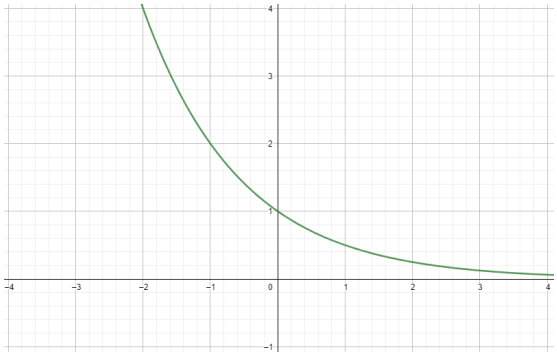
Dicho esto, al calcular el gradiente para una red neuronal de n capas se calculan n derivadas que se aplican de manera multiplicativa entre capas para aplicar el algoritmo de retropropagación. Si el valor de dichas derivadas es muy grande (en concreto mayor que uno), se producirá un crecimiento exponencial del valor del gradiente, pudiendo llevar a cambios en los pesos de la red muy grandes y pudiendo llegar a producir el llamado *Gradient Explosion*. Por otro lado si el valor de dichas derivadas es pequeño (menor que uno), puede llegar a suceder el *Gradient Vanishing* al producirse un decrecimiento exponencial muy elevado.

Para entender más en profundidad la explicación matemática de este problema sólo hay que atender a las gráficas de las funciones exponenciales, donde el valor de la base es menor o mayor que uno como se puede observar en la función 1 y en la figura 10:

$$f(x) = a^x \tag{1}$$



(a) Gráfica de la función exponencial para un valor de $a>1$.



(b) Gráfica de la función exponencial para un valor de $a<1$.

Figura 10: Gráficas de crecimiento y decrecimiento exponencial.

Como se puede observar, a medida que el número de capas es mayor (valor x) se produce un crecimiento o decrecimiento más pronunciado, aumentando la posibilidad de producirse los problemas relacionados con el gradiente descritos anteriormente.

En este punto se deben diferenciar los casos de *Gradient Explosion* y *Gradient Vanishing* y estudiar las causas y las consecuencias de cada caso en los entrenamientos de nuestras redes:

- **Gradient Explosion:** La acumulación de valores de error muy grandes provoca cambios bruscos en el comportamiento de la red que se está entrenando. Estos cambios bruscos pueden dar lugar a que la red tenga un comportamiento muy inestable, la inestabilidad en las redes puede llevar a que la red empeore su comportamiento como se ha indicado en la sección 2.3.2.

  A la hora de identificar si se está produciendo *Gradient Explosion* en la red que se está entrenando hay que fijarse en los valores de las pérdidas. Las pérdidas cuando se produce este problema toman valores muy elevados con cambios muy bruscos entre iteraciones, también pueden tener como valor NaN[12].

  La figura 11 muestra un ejemplo en el que se produce *Gradient Explosion* en torno al *batch* 10.000 para un entrenamiento de redes GAN de la presente investigación. Se puede observar que a partir de ese punto las redes son incapaces de continuar aprendiendo:

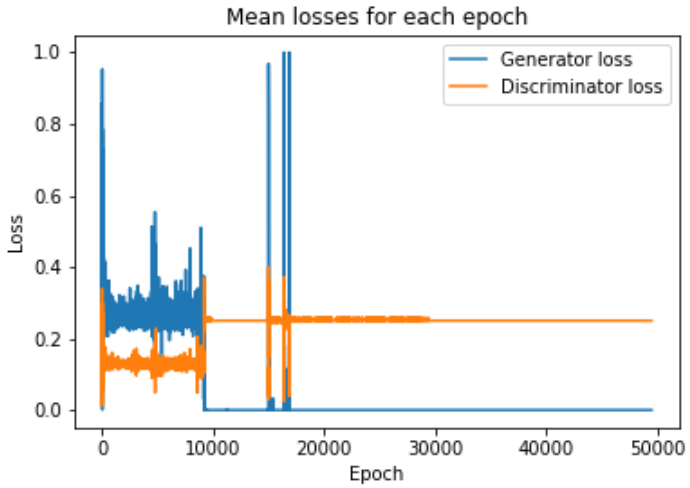


Figura 11: Ejemplo de un entrenamiento con *Gradient Explosion*.

[12]NaN (Not a Number) se usa en diferentes lenguajes de programación para expresar un valor incalculable bien por ser un número imaginario, demasiado grande o una indeterminación entre otros.

- **Gradient Vanishing:** Cuando los cambios de pesos en los parámetros de una red se realizan de manera cada vez menor se produce este error. Si en una red los cambios son cada vez menores se produce una desaceleración en el aprendizaje de la misma, pudiendo llegar a quedar parada completamente e impidiendo su mejora en el tiempo. Cuando esto sucede la red deja de entrenar y por lo tanto deja de tener sentido seguir con el entrenamiento.

  La característica principal para identificar este problema es un cambio cada vez menor en las pérdidas de la red que se entrena, hasta el punto que puede llegar a suceder que siempre sea el mismo. A un nivel más bajo se pueden observar gradientes mayores en las capas más cercanas a la salida de la red mientras que las capas más cercanas a la entrada de la red permanecen con los mismos pesos. Esto se corresponde con la gráfica de la función exponencial que se ha descrito anteriormente en la que para mayor valor de x (capa más cercana a la entrada) se produce un valor menor.

  La figura 12 muestra un ejemplo en el que se produce *Gradient Vanishing* en torno al *batch* 5.000 para un entrenamiento de redes GAN de la presente investigación. Se puede observar que los gradientes de las redes se minimizan haciendo que las redes se queden estancadas en un error del que no pueden salir:

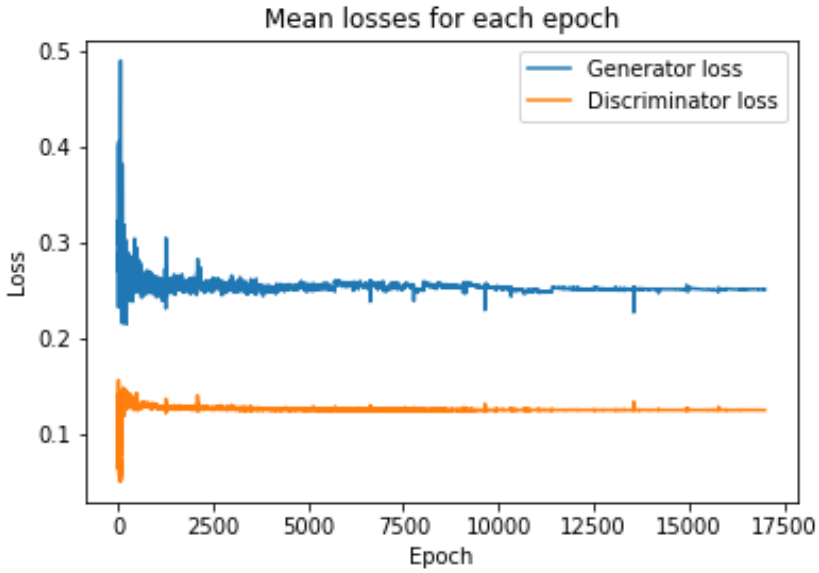


Figura 12: Ejemplo de un entrenamiento con *Gradient Vanishing*.

La situación más normal ante la sucesión de cualquiera de los dos problemas es que la red a entrenar quede inservible, ya que una vez sucede un problema como los descritos es muy difícil que la red vuelva a un estado normal de entrenamiento, pues se produce una cascada en la que el problema en cuestión se retroalimenta a sí mismo y cada vez se empeora.

2.5. Composición de los *batches*

A la hora de realizar un entrenamiento en una red neuronal hay que hacer uso de la información presente en nuestro *dataset*. Esta información se puede escoger de maneras muy diversas dando lugar a diferentes esquemas de entrenamiento.

De manera simplificada, para entrenar una red hay que hacer uso de casos que sirvan de ejemplo para que la red aprenda las diferentes características de la información que está tratando. Estos casos son los que forman parte del dataset de información real. Esta información se usa como entrada para el entrenamiento de la red, pero hay que definir una manera concreta de qué casos y cómo escogerlos del *dataset* para hacer uso de ellos:

- **Selección individual:** La forma más sencilla de escoger la información es una a una, de manera iterativa recorriendo así todo el *dataset*. Una vez se acabe con todos los casos reales se da por finalizado el *epoch*[13] y se vuelve a comenzar. Este modelo de entrenamiento es el más básico y tiene como inconvenientes la poca eficiencia en el entrenamiento y la posibilidad de que la red aprenda los casos y se produzca *overfitting*.
- **Selección individual aleatoria o estocástica:** En este caso la selección de los casos con los que se entrena a la red se realiza de manera aleatoria uno por uno. Cada vez que se vaya a realizar una iteración del entrenamiento se escoge un caso aleatorio del *dataset* y se hace uso de él. Una vez se escoge un caso del *dataset* este no queda excluido de ser escogido inmediatamente después, esto provoca que una misma información pueda ser utilizada varias veces mientras que otra no sea nunca utilizada. Al ser aleatorio el proceso de elección se evita la posibilidad de que la red comience a aprenderse todos los casos ya que se está metiendo cierta variabilidad extra a la información de entrenamiento. Esto por ejemplo evita que la red comience a hacer la media de todas las informaciones para minimizar el error. En definitiva, se introduce cierta varianza a la selección que ayuda al entrenamiento a ser más aleatorio y parecerse más a la realidad al ser un entorno menos predecible; sin embargo al igual que en el método anterior los entrenamientos con este modelo no son eficientes. Una consecuencia directa de este método de uso es que desaparece el concepto de *epoch* al no recorrerse el *dataset* de manera consistente y equitativa.

[13] Se define *epoch* como el conjunto de *batches* con los que se entrena la red haciendo uso todos los casos del *dataset*. Por lo tanto un *epoch* es el entrenamiento de la red con todos los casos reales una vez. Un entrenamiento suele estar compuesto de varios *epochs*.

- **Agrupación por *batches*:** Para aumentar la eficiencia de los entrenamientos, los casos con los que se realizan los algoritmos aprendizaje automático se agrupan y se calculan simultáneamente. De esta forma al realizar el proceso de *Backpropagation* se actualizan los pesos calculando la media de todos los casos con los que se ha realizado la iteración. Gracias a esto se evita realizar múltiples veces la retropropagación que es el algoritmo más pesado computacionalmente, agrupando los resultados y aumentando la eficiencia de la red. A simple vista se puede ver que al realizar la actualización de los pesos de manera agrupada se pierde cierta precisión, ya que se agrupan los resultados y los gradientes individuales pueden quedar difuminados en el resultado final. Dicho de otra manera no es lo mismo hacer varias actualizaciones de manera seguida que todas a la vez porque en el primer caso al realizar las últimas actualizaciones se hará uso de los cambios de las primeras. A la hora de realizar los cálculos conjuntos de todos los casos se realizan operaciones matriciales para las cuales se hace uso, si es posible, de tarjetas gráficas, estas tarjetas están especialmente diseñadas[14] para realizar operaciones matriciales en las conocidas como operaciones MIMO (Multiple Input-Multiple Output). El aumento de eficiencia que se consigue haciendo uso de este tipo de tarjetas acelera drásticamente la velocidad del entrenamiento. La eficiencia ganada con este proceso hace que sea el más utilizado en entrenamientos de *machine learning* ya que se hace uso de todo el potencial que pueden ofrecer los ordenadores actuales. Igual que en el primer caso la poca variabilidad en los casos de entrenamiento puede favorecer la aparición de *overfitting* en la red que se entrena.

- **Agrupación de *batches* estocásticos:** Es el proceso de entrenamiento más especializado y se basa en realizar *batches* de varios casos de manera aleatoria. Para formar un *batch* se escogen varias informaciones del dataset de manera aleatoria, igual que en el caso de selección individual estocástica los casos no son excluidos de futuras elecciones pudiendo aparecer varias veces seguidas. En este caso esto tiene como fortaleza que los distintos *batches* no comparten los mismos casos, aumentando aún más la variabilidad, ya que es prácticamente imposible que dos *batches* sean iguales. De esta forma se aprovecha al completo la eficiencia del uso de la tarjeta gráfica, al mismo tiempo que se aumenta la variabilidad en los casos de entrenamiento evitando el *overfitting*. Este modelo es el que proporciona los mejores resultados de manera general, ya que aprovecha la eficiencia de las tarjetas gráficas en tareas de inteligencia artificial al mismo tiempo que minimiza los posibles inconvenientes de agrupar información.

[14] Las tarjetas gráficas cuentan con un número muy elevado de núcleos que realizan operaciones muy sencillas. A diferencia de los microprocesadores tradicionales, el mayor número de operaciones menos especializadas permite a las tarjetas gráficas un mayor rendimiento a la hora de hacer cálculos con matrices de gran tamaño.

Metodología

3.1. Trabajo previo

Antes de comenzar a estudiar las nuevas estructuras de *Deep Learning* con las que se va a trabajar, hay que hacer un repaso del trabajo realizado en el anterior TFG.

En el anterior trabajo se obtuvo un *dataset* de imágenes que será el que se usará para los entrenamientos de las redes creadas. Dicho *dataset* cuenta con unas 300.000 imágenes, clasificadas con etiquetas dependiendo de su contenido. Además, en el anterior trabajo se hicieron modificaciones sobre las imágenes para obtener una versión de dichas imágenes en blanco y negro y con una menor resolución[15]. Dichos *datasets* generados a partir del original pueden ser reutilizados en un futuro, así como generar nuevos siguiendo el mismo método que en el trabajo anterior.

A parte del *dataset*, en el anterior trabajo se desarrollaron redes clasificadoras para la clasificación de n elementos[16], las cuales podemos reutilizar aprovechando su estructura general para la creación de nuevas redes. Por ejemplo, para crear una red discriminadora, se puede reutilizar la estructura de una red clasificadora modificando la última capa de la red y sacando una única salida.

Cabe destacar que para el tratamiento de las imágenes se hace una normalización de su información, pasándola a un rango [0, 1] ya que las redes neuronales trabajan mejor con números dentro de estos límites[17]. Esta normalización se mantiene para el desarrollo de la nueva investigación.

En general, todo el trabajo anterior sienta unas bases donde poder empezar con el nuevo trabajo. Los conocimientos adquiridos con la anterior investigación sirven para poder desarrollar estructuras de *Deep Learning* con facilidad y los resultados anteriores pueden ser reutilizados para comenzar con el nuevo desarrollo. Aunque lo más probable es que en algún punto se necesiten de nuevas creaciones y haya que descartar las anteriores.

3.2. Estudio de la composición de las imágenes

Para recapitular y hacer un estudio sobre los posibles problemas que puedan surgir en esta investigación, hay que hacer un breve análisis de cómo están compuestas las imágenes del *dataset* del que se hará uso. Si se estudian las imágenes que se tienen se pueden detectar elementos comunes en ellas que más adelante puedan manifestarse de manera positiva o negativa.

[15] Durante los estudios realizados en el anterior trabajo se decidió disminuir la dimensionalidad de la información que se usaba en las redes para disminuir la carga de estas. Con esto se consiguió una mayor eficiencia y velocidad en los entrenamientos además de mejores resultados.

[16] Se hicieron clasificaciones de 3, 5 y 7 elementos.

[17] En la anterior investigación se justificó los motivos de este tipo de normalización y se investigaron las maneras más óptimas de realizar la normalización. Las redes neuronales trabajan mejor con valores cercanos al 0 ya sea en el rango [-1, 1] o como en este caso [0, 1].

Este paso no parece importante, pero supone familiarizarse con el problema que se quiere enfrentar y esto dará un mayor conocimiento que, en última instancia, puede hacer que se detecten particularidades que resulten cruciales para la investigación.

Las redes que se desarrollen aprenderán las características más comunes de las imágenes del *dataset*, ya que de esta manera estarán minimizando el error a la hora de realizar sus entrenamientos. Para poder guiar las redes que se creen hacia unos resultados u otros primero hay que conocer bien cuáles son las posibilidades y qué tipo de imágenes son las que componen el *dataset* del que se hará uso.

En el anterior trabajo se realizó un estudio en gran profundidad sobre este aspecto, como resumen los aspectos más relevantes del *dataset* son:

- **Distribución de etiquetas:** En el anterior trabajo se hizo un estudio en profundidad sobre la frecuencia de aparición de las etiquetas o *tags* en el *dataset*. La figura 13 muestra la distribución de las etiquetas más populares:

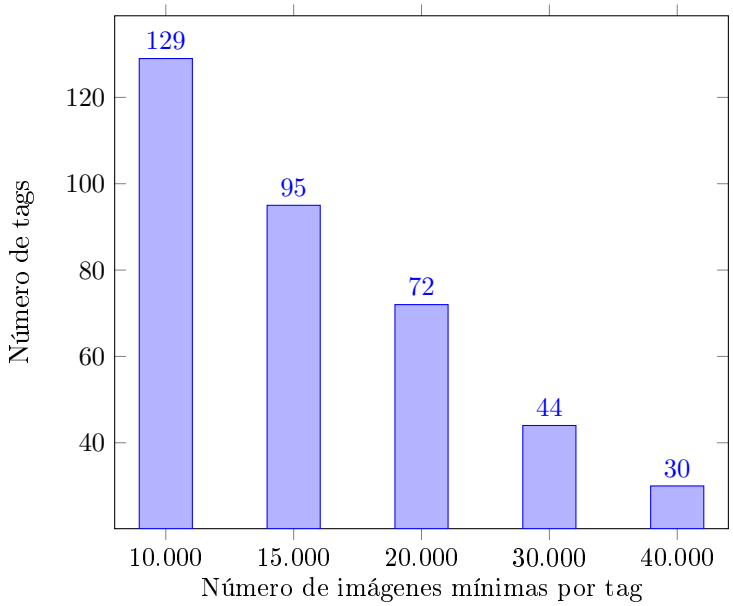


Figura 13: Resultado de la contabilización del número de *tags* con muchas imágenes.

Este esquema da una imagen general de cómo se distribuyen las etiquetas y cuántas etiquetas hay con cuántas imágenes. Esto puede ser útil en un futuro a la hora de realizar filtrado de imágenes dependiendo de sus elementos.

- **Imágenes más comunes:** Hay que atender a qué clases de etiquetas hay en el *dataset*. Para ello en el trabajo anterior se hizo una clasificación de las 100 etiquetas más comunes, un resumen de dicho resultado se puede ver en la figura 14:

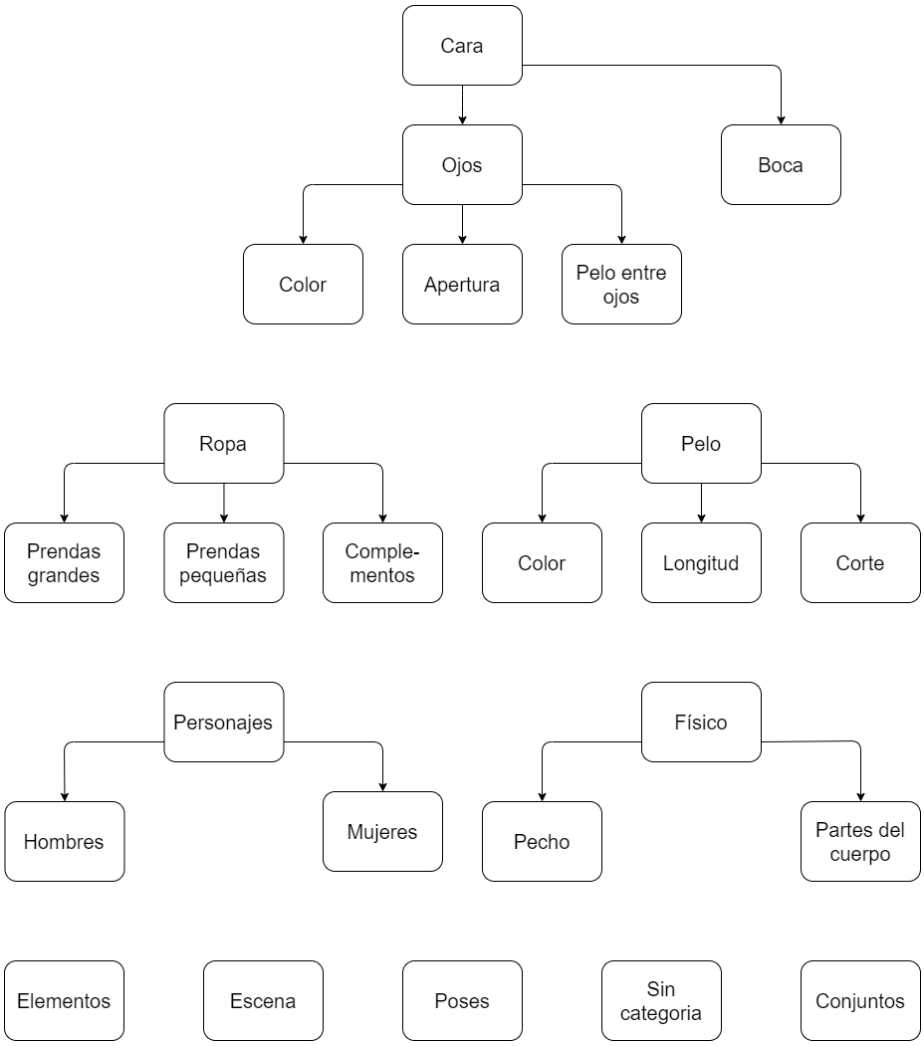


Figura 14: Esquema de los tipos de etiquetas del *dataset*.

En general, se observa que la mayoría de imágenes del *dataset* son de un personaje mayoritariamente femenino en solitario. Sobre esta simplificación hay muchas variaciones en su posición, vestimenta, fondo, complementos, cara o pelo entre otros. Para nuestro cometido se buscará, por lo tanto, recrear estas imágenes más comunes. En la figura 15 se puede ver un ejemplo tipo de imagen que se pretende recrear:

Figura 15: Imagen con una composición de elementos frecuente en el *dataset*.

- **Bandas negras:** Debido a las transformaciones que sufrieron las imágenes originales del *dataset*[18] aparecen bandas negras para ajustar todas las imágenes a un mismo tamaño. Estas bandas tienen como cometido hacer que todas las imágenes tuviesen la misma forma cuadrada, por lo tanto dependiendo de la imagen original las bandas pueden aparecer en forma horizontal, vertical o un recuadro negro. Esta transformación se estudió en el anterior trabajo y es especialmente relevante para la investigación que se va a realizar. La presencia de bandas negras en las imágenes hará que, a la hora de replicar las imágenes con las redes GAN que se desarrollen, estas repliquen también dichas bandas, esto puede tener consecuencias que se tienen que tener en cuenta. Un ejemplo de imágenes con los recuadros mencionados se puede observar en la figura 16:

[18] El origen del *dataset* es una página web llamada *Danbooru*[10] que recopila dibujos originales creados por usuarios y subidos a la página *danbooru.donmai.us*. A la hora de hacer la recopilación de imágenes, estas fueron normalizadas para que todas tuviesen las mismas dimensiones.

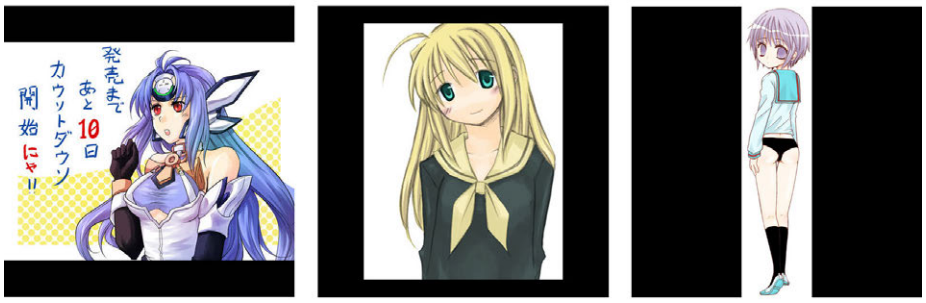


Figura 16: Imágenes con bandas y recuadros negros.

3.3. Entorno de desarrollo en remoto

Durante la realización del anterior trabajo fin de grado se observó la dificultad de realizar trabajo con los medios disponibles. La baja potencia del equipo con el se cuenta hace que se tengan que buscar nuevas alternativas que ofrezcan mayor potencia para realizar los diferentes entrenamientos de las redes.

Una de las carencias esenciales a las que nos enfrentamos es la imposibilidad de acceder a una tarjeta gráfica potente, específicamente preparada para cálculos matriciales que proporcione una capacidad de cálculo suficiente. Este elemento es esencial para desarrollar redes neuronales, pues su capacidad de realizar operaciones matriciales hace que los entrenamientos sean más óptimos en comparación con el uso de otros medios como trabajar con los microprocesadores tradicionales.

Todo esto deriva en la búsqueda de entornos que proporcionen medios especializados en operaciones relacionadas con la inteligencia artificial. Una de estas posibilidades es la plataforma de *Google* conocida como *Google Colaboratory* o *Google Colab*.

3.3.1. Google Colaboratory

La plataforma *Google Colab* permite la ejecución de código de *Python*[19] de manera remota haciendo uso de recursos en la nube[20]. Esta plataforma permite a los usuarios conectarse a un entorno de desarrollo basado en celdas en el cual poder ejecutar código haciendo uso de distintos recursos ofrecidos.

[19] *Python* es un lenguaje de programación interpretado orientado a objetos lanzado en el año 1991 el cual se caracteriza por la gran cantidad de librerías de todo tipo, para el propósito de el presente trabajo son especialmente interesantes las relacionadas con el *Machine Learning* como *Keras*, *Tensorflow* o *Pytorch* entre otras.

[20] El término de la nube hace referencia a recursos informáticos los cuales se ofrecen de manera remota de modo que el usuario es ignorante del *hardware* y su mantenimiento.

Una de las ventajas de desarrollar código en esta plataforma es que a la hora de ejecutarlo se ofrecen como recursos memorias RAM de gran capacidad y, sobre todo, de tarjetas gráficas de gran potencia especialmente preparadas para operaciones matriciales relacionadas con el *Machine Learning*. A través del uso de estas tarjetas gráficas se consiguen acelerar enormemente los procesos relacionados con operaciones matriciales, que son las bases sobre las que se sustentan las redes neuronales.

Para dar un ejemplo de cómo afecta el hardware en la velocidad de, en este caso, la operación de convolución que más adelante se usará en el proyecto, existe un cuaderno[21] de *Google Colab* llamado *Tensorflow with GPU*[11] el cual compara los tiempos de ejecución de operaciones convolucionales usando CPU y GPU. En este caso se realiza el test para convoluciones de 32 filtros con un tamaño de núcleo de (7,7). Para disminuir la varianza se realizan 10 operaciones por cada ejecución. La figura 17 muestra el código utilizado para realizar las pruebas, utilizando el parámetro image_size para variar el tamaño de las imágenes.

```
%tensorflow_version 2.x
import tensorflow as tf
import timeit

image_size=250

device_name = tf.test.gpu_device_name()
if device_name != '/device:GPU:0':
  print(
      '\n\nThis error most likely means that this notebook is not '
      'configured to use a GPU.  Change this in Notebook Settings via the '
      'command palette (cmd/ctrl-shift-P) or the Edit menu.\n\n')
  raise SystemError('GPU device not found')

def cpu():
  with tf.device('/cpu:0'):
    random_image_cpu = tf.random.normal((100, image_size, image_size, 3))
    net_cpu = tf.keras.layers.Conv2D(32, 7)(random_image_cpu)
    return tf.math.reduce_sum(net_cpu)

def gpu():
  with tf.device('/device:GPU:0'):
    random_image_gpu = tf.random.normal((100, image_size, image_size, 3))
    net_gpu = tf.keras.layers.Conv2D(32, 7)(random_image_gpu)
    return tf.math.reduce_sum(net_gpu)

# We run each op once to warm up; see: https://stackoverflow.com/a/45067900
cpu()
gpu()
```

Figura 17: Código del test de velocidad de GPU y CPU.

[21] Los cuadernos de *Google Colab* son los archivos de código ejecutables en la plataforma.

La figura 18 muestra una gráfica con los resultados de ejecutar el código de la prueba descrita, contrastando los resultados entre ejecuciones en CPU y en GPU:

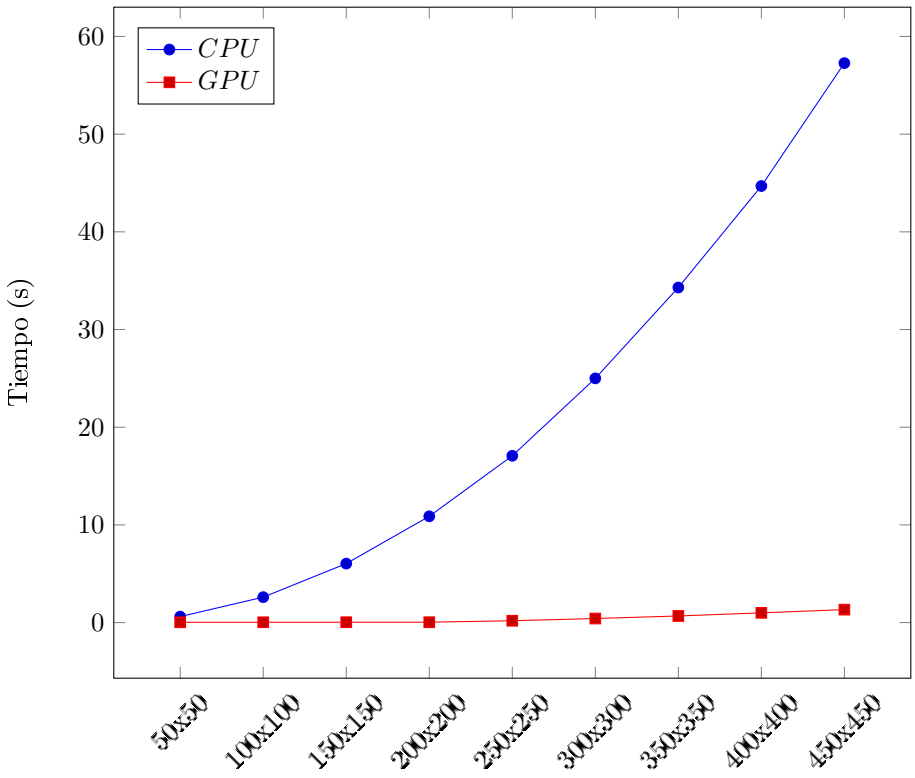


Figura 18: Evolución de los tiempos de ejecución para distintos tamaños de imagen en píxeles.

Como se puede ver, la diferencia entre uso de tarjeta gráfica o no para realizar operaciones relacionadas con el *Machine Learning* es notable y lleva a poder realizar entrenamientos de manera más rápida.

Cabe destacar que el tamaño de la memoria RAM también es relevante para el uso que haremos de ella. Esta memoria es la encargada de guardar el estado y la estructura de las redes que se crean. Estas redes pueden llegar a tener millones de parámetros y por lo tanto el espacio que ocupan en memoria es elevado[22]. Por su parte, la información con la que se realizan los entrenamientos ha de ser cargada para poder entrenar con ella, esta carga se almacena también en la memoria RAM.

[22] Al realizar los entrenamientos en el anterior trabajo se llegaron a ocupar varios gigabytes sólo con la estructura de una red neuronal.

3.3.2. Arquitectura de entorno de trabajo

Ahora que se ha escogido y estudiado el entorno en el que se desarrollarán los distintos modelos, hay que definir una metodología para conectar el entorno con el *dataset* y los resultados de los entrenamientos. Para el trabajo que se va a realizar se puede considerar que hay tres partes implicadas:

- **Dataset:** Las imágenes del *dataset* así como sus metadatos están almacenados en el disco duro de un equipo. Las imágenes están organizadas en carpetas y tienen formato .jpg y los archivos de metadatos tienen formato .json.
- **Entorno de desarrollo:** Como se ha explicado anteriormente, el entorno de desarrollo y ejecución de los diferentes entrenamientos está en la nube a través del portal *Google Colaboratory*.
- **Resultados:** Los resultados de los entrenamientos ejecutados en *Google Colab* pueden ser mostrados en el propio portal, pero si se quieren almacenar permanentemente la plataforma permite conectar el entorno de desarrollo con *Google Drive*[23].

Estos elementos se relacionan entre sí haciendo uso unos de otros para poder realizar las tareas de *Machine Learning*. Por su parte el *dataset* y sus metadatos han de ser utilizados por los modelos creados en *Google Colab* y sus resultados tienen que poder volver a recuperarse en un equipo local.

Para interconectar todos los agentes hay que solucionar dos problemas, el primero de ellos es cómo almacenar los resultados que se generan de los entrenamientos para ser posteriormente guardados y estudiados. Como se ha señalado anteriormente este problema tiene una fácil solución pues desde *Google Colaboratory* se puede conectar la entrada/salida de ficheros con *Google Drive*. Para ello simplemente hace falta iniciar sesión en una cuenta de *Google* y facilitar los permisos pertinentes a *Google Colaboratory*. Una vez almacenados los resultados en *Google Drive* simplemente se pueden descargar si se considera necesario y almacenar en cualquier disco duro.

El segundo problema a tratar es cómo conectar el *dataset* para que este pueda ser usado por *Google Colaboratory*. Para ello la opción más directa es subir los distintos archivos a una cuenta de *Google Drive* para que posteriormente puedan ser accedidos al conectarse a dicho almacenamiento, sin embargo esto es imposible pues al contar con un *dataset* que tiene un número muy elevado de archivos (más de 300.000) la operación de subir los archivos a una cuenta de *Google Drive* es eterna. Pese a que los archivos no ocupen mucho espacio, al ser un número muy elevado se ralentiza el proceso de manera drástica.

[23] *Google Drive* es un servicio de almacenamiento de archivos en la nube ofrecido por *Google*

La solución que se encuentra para esto es crear un repositorio de *Github*[12] en el que almacenar las imágenes y metadatos del *dataset*. Posteriormente desde el entorno de ejecución remoto se clona dicho repositorio para que pueda ser utilizado por las redes que se creen. El proceso de almacenar y descargar un número muy elevado de archivos está optimizado en *Git* permitiendo realizarse en un tiempo asumible.

Para recapitular y resumir el proceso de conexión entre las distintas partes del sistema la figura 19 muestra un esquema de cómo esta organizado la arquitectura de los distintos componentes:

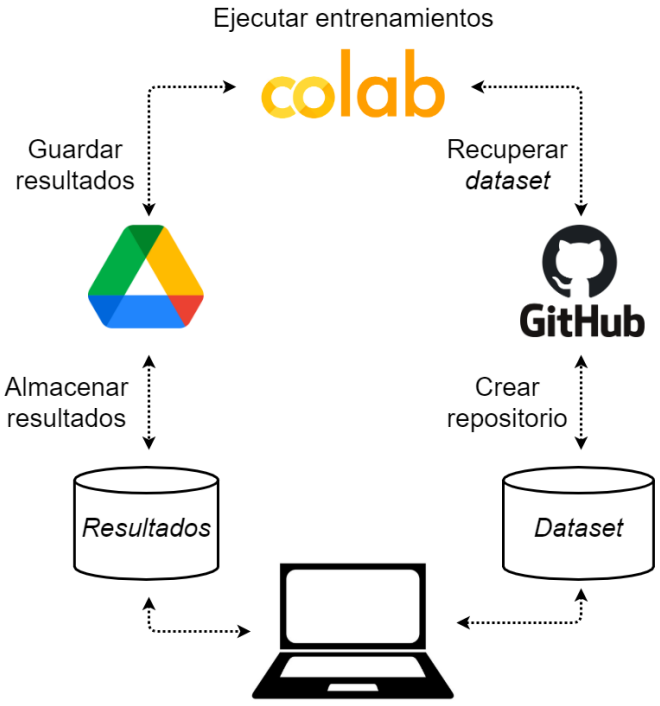


Figura 19: Arquitectura del entorno de trabajo.

3.3.3. Estudio de la estructura de las imágenes

Para comprender la complejidad del problema que se hace frente primero hay que comprender cómo se componen las imágenes que se quieren replicar. Los elementos y composición de las imágenes que están presentes en el *dataset* son los que se usan por las estructuras que se diseñen para crear nuevas imágenes fijándose en ellas. Es decir, las imágenes de la base de datos son las que hay que imitar, la dificultad del problema de esta imitación está directamente relacionada con la complejidad de las imágenes que se intentan copiar.

Uno de los resultados de este tipo de aplicaciones de inteligencia artificial más interesante de los últimos años es la creación de imágenes de caras humanas con investigaciones como la desarrollada por un laboratorio de investigación de la empresa *NVIDIA* llamada *A style-based generator architecture for generative adversarial networks*[13]. Este tipo de aplicaciones de estructuras de aprendizaje automático aplicadas a la replicación de imágenes son en las que se basa el trabajo actual.

Sin embargo antes de comenzar con la presente investigación, hay que estudiar las diferencias que hay con este tipo de trabajos. El factor más importante es la diferencia compositiva existente entre las imágenes que se quieren replicar, en el caso de las imágenes de caras hay que destacar que los elementos presentes en ellas siempre son los mismos dando lugar a poca varianza y un entorno más estable. Sin embargo, en el caso de la investigación que se quiere llevar a cabo las composiciones entre distintas imágenes pueden variar de manera drástica, dando lugar a un problema mucho más complejo. Esto se puede observar en que en el caso de las imágenes de caras los elementos como ojos o boca ocupan siempre lugares relativamente cercanos mientras que en el problema de imágenes de cómic los elementos pueden variar tanto de forma como de posición, color o estructura.

Esta complejidad en las imágenes del *dataset* que se quiere utilizar generará una mayor dificultad a las redes creadas de comprender y replicar las características de las imágenes y, en definitiva, hace que el problema sea muy complicado. Por ello los resultados que se pretenden obtener tienen que estar en sintonía con el problema y por lo tanto no se pretende obtener un nivel de detalle muy elevado.

Para observar la complejidad de la que se habla en las imágenes del *dataset* la figura 20 muestra algunas imágenes del *dataset* con estructuras complejas:

Figura 20: Imágenes del *dataset*.

Como se puede ver las distintas composiciones hacen muy complicada la labor de extraer características comunes de las imágenes. En un futuro se diseñarán mecanismos para paliar el problema como por ejemplo el filtrado de imágenes haciendo uso de los metadatos del *dataset*.

3.4. Estructura de la nueva investigación

Para desarrollar las redes GAN con las que se obtengan los resultados buscados, hay que partir de una estructura inicial simple e ir adaptándola a los diferentes inconvenientes que se presenten. De este modo se comenzará con la composición más sencilla que se pueda crear y, poco a poco, se irá desarrollando dicha estructura para mejorar los resultados.

Este proceso iterativo asegura que los diferentes pasos que se vayan dando sean lo más certeros posibles. Hay que recordar que las redes de *Deep Learning* son muy inestables y más aún si se tratan de redes GAN. Así se irá desarrollando el modelo hasta conseguir los mejores resultados posibles intentando solucionar los problemas que se vayan presentando.

Esta metodología que se seguirá dará como resultado un conjunto de redes del cual puede ser trazada su evolución a lo largo del tiempo. Además que quedarán justificados todos los cambios que se vayan realizando.

A la hora de realizar la investigación se seguirán tres enfoques distintos: empezando por la manera más sencilla de redes GAN que serían redes formadas por capas densas, posteriormente redes convolucionales para dotar a las redes de una mayor complejidad intentando avanzar la investigación y por último se seguirá la metodología de entreno llamada *Progressive Growing of GANs* en la que se intentarán conseguir los mejores resultados posibles.

Cada una de estas etapas de la investigación buscan llevar una continuidad entre los distintos entrenos, pasando de una metodología a otra de manera suave teniendo un mayor control. Perfectamente se podría empezar aplicando la técnica más compleja y que a priori tendría mejores resultados que es la ProGAN, al ser la más avanzada de las tres, pero al ser una técnica más compleja es más fácil que falle en algún punto.

El objetivo es empezar con una arquitectura de redes sencilla, desarrollarla hasta minimizar los fallos que esta pueda tener y en ese punto evolucionarla y escalarla. De este modo se consiguen minimizar los errores y, si repentinamente sucede algún mal funcionamiento, se controla la causa de este.

Se realizarán pues una gran cantidad de entrenos usando cada técnica probando distintos tipos de configuraciones y estructuras que busquen ir mejorando las redes e ir solucionando los problemas que surjan.

3.5. Aproximación a las redes GAN densas

Las primeras redes servirán como toma de contacto con la resolución del problema. Sin embargo no se espera que con estas redes se obtengan unos buenos resultados. La filosofía general de esta primera aproximación lleva a diseñar unas redes sencillas que puedan ser controladas y estudiadas fácilmente. Si se usa una estructura de red simple, se consigue tener un mayor control sobre los diferentes parámetros de esta. Por una parte esta aproximación proporciona una mayor sencillez a la hora de crear, pero se está limitando la capacidad de las redes de conseguir imitar las imágenes del *dataset*.

Dicho de otra forma, se pretende realizar una primera iteración de entrenos que sirvan de toma de contacto para ver cómo unas redes sencillas responden al problema, qué problemas pueden surgir en un futuro y cómo se diferencia el problema a tratar de otros de los que existe una amplia bibliografía.

3.5.1. Configuración del entreno

A la hora de realizar el entrenamiento, se decide usar el método de agrupación de *batches* estocásticos ya que es el que mejor resultados proporciona en un principio. Sin embargo, y para hacer más sencillo el proceso, las agrupaciones de imágenes que forman un *batch* se realiza previamente y la selección aleatoria se realiza entre los distintos *batches* posibles. Como tamaño de *batch* se usarán inicialmente *batches* de 128 imágenes. Este número de imágenes por *batch* podrá ser modificado por ejemplo reduciéndolo si las redes tardan demasiado en realizar el entrenamiento para un *batch*.

Como función de pérdida de las redes se usará la pérdida *binary crossentropy* y como optimizador el *Adam* con un *learning rate* de 0.0002 y de *beta* 0.5 ya que son parámetros estándar que han sido eficientes para la resolución de problemas similares[24]. Este punto de partida sienta una base para más adelante poder ir modificando estos valores, por ejemplo disminuyendo el *learning rate* para evitar la inestabilidad de las redes ya que, con un *learning rate* menor se consigue que las redes modifiquen sus pesos de manera más leve.

3.5.2. Filtrado de imágenes del *dataset*

Antes de usar el *dataset* que se tiene por completo hay que tomar ciertas decisiones que faciliten la tarea. Gracias a esto se pretende simplificar la distribución de las imágenes de las que se hará uso.

[24] Como referencia se usa el repositorio de *Github* llamado Keras-GAN[14].

Se deciden usar imágenes en blanco y negro en vez de imágenes con los tres canales RGB. Como ya se observó en la anterior investigación la reducción de las dimensiones de los canales de color implica que la red tenga un menor tamaño y las imágenes menos información, por lo tanto sea más sencillo que las redes aprendan las características de la imagen a replicar. Como con el uso de imágenes en blanco y negro se obtuvieron buenos resultados en la anterior investigación se considera óptimo usarlo como punto de partida.

Además se decide cambiar la dimensionalidad de las imágenes que se están usando del *dataset*. Para ello se decide usar imágenes de 48x48 píxeles que es la dimensión de las imágenes que se utilizó en el clasificador del anterior TFG[25]. Además de reducir la dimensión de las imágenes, se decide cribar qué imágenes del dataset se van a usar para evitar la aparición de imágenes que no sean válidas para el propósito de la investigación.[26]. Para ello, se escogen sólo las imágenes que cuenten con la etiqueta *1girl* o *solo*.

A la hora de normalizar las imágenes del *dataset* se realiza en el rango [-1, 1] en vez del rango [0,256]. De esta manera la información de entrada sea igual que la de salida en la red generadora ya que la activación de esta es mediante la función *tanh*. Este rango de normalización es el más utilizado en redes GAN ya que en general las estructuras de *machine learning* trabajan mejor dentro de valores entre -1 y 1 siendo los rangos más habituales [0,1] y [-1,1].

3.5.3. Equilibrar entreno entre generador y discriminador

Uno de los principales problemas que puede surgir a la hora de tratar con redes generativas adversarias es la inestabilidad de los entrenamientos. Como se trata con varias redes al mismo tiempo que interaccionan entre sí, es muy difícil conseguir que las distintas redes converjan conjuntamente para encontrar la solución óptima.

Antes de probar a crear estructuras de redes complejas, como se pretende mantener unas arquitecturas en las redes sencillas, se decide probar con otro tipo de mecanismos que, en vez de cambiar las redes en sí, controlen el entrenamiento.

[25] Con imágenes de 48x48 píxeles se consiguieron buenos resultados a la hora de realizar clasificaciones. Este tamaño es lo suficientemente pequeño como para que las redes creadas no estén sobredimensionadas mientras que no se pierde información de las imágenes por falta de resolución.

[26] Durante el estudio del *dataset* en el TFG anterior se observaron ciertas imágenes que eran inservibles para el uso que se iba a hacer de ellas. Dichos conjuntos de imágenes se estudiaron en profundidad y se vio que la forma más sencilla de no usarlas es mediante el filtrado por las etiquetas que tienen las imágenes.

Para evitar que la red discriminadora se vuelva demasiado certera, se debe controlar su estado de entrenamiento e impedir que se sobreentrene y llegue a un punto en el que la diferencia entre las redes sea tan grande que se imposibilite cualquier avance. Esto sucede porque el problema al que tiene que hacer frente esta red es mucho más sencillo que el de la red generadora y por lo tanto normalmente tiende a especializarse más rápidamente que dicha red obteniendo mejores resultados e impidiendo que la generadora mejore.

Diferencia de *batches* de entreno: Una de las soluciones más comunes para evitar este problema es realizar un número mayor de *batches* de la red generadora que de la red discriminadora. De esta forma se impulsará el desarrollo del generador intentando evitar que se quede atrás y sus producciones sean fácilmente identificables por la red discriminadora. Se decide, por lo tanto, utilizar este mecanismo realizando 2 *batches* de entrenamiento del generador por cada *batch* de entrenamiento del discriminador.

Uno de los mayores inconvenientes que tiene la diferencia de *batches* es que no se adapta al estado de desincronización que pueda haber entre las redes, al ser un mecanismo estático puede provocar que sea insuficiente si la red discriminadora se especializa demasiado, mientras que en otro punto en el que la red discriminadora no distinga correctamente imposibilite el avance. Durante las distintas pruebas que se realicen se probará a añadir el desfase y prescindir de él cuando, bien las redes avancen al mismo ritmo, bien cuando haya mucha inestabilidad entre ellas.

Etiquetado de imágenes generadas como reales: Se decide complementar con otro mecanismo que evite de forma dinámica que el discriminador se sobreentrene. Para ello, se decide intentar engañar al discriminador haciéndole creer que imágenes generadas son reales del *dataset*.

Esto se basa en que a la hora de entrenar el discriminador este se entrena con imágenes generadas igual que siempre pero, al entrenar con las imágenes reales, se cambian algunas de ellas por imágenes generadas por la red generadora. De esta forma se incluyen ente las imágenes etiquetadas como reales algunas que no lo son y se fuerza a la red discriminadora a que identifique algunas de las imágenes generadas como reales cuando en realidad no lo son. Este engaño consigue igualar los estados de entrenamiento de las dos redes pues se engaña a la red discriminadora para que no sea demasiado eficaz con las clasificaciones que realiza.

La figura 21 muestra un esquema en el que se puede observar cómo se componen los *batches* de entrenamiento de la red discriminadora antes y después de usar el nuevo mecanismo:

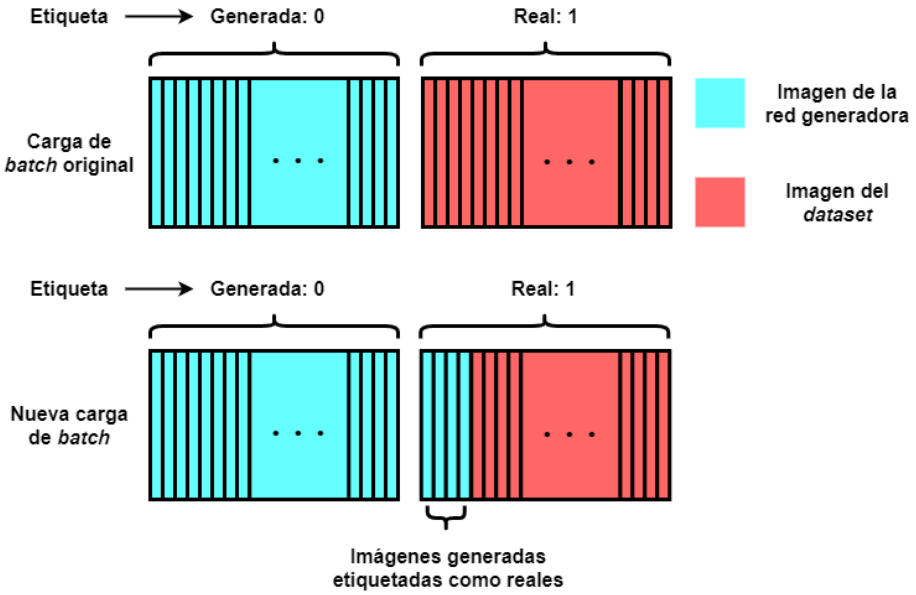


Figura 21: Esquema de la composición de los *batches* antes y después del nuevo mecanismo de engaño de la discriminadora.

En este punto hay que definir cuántas imágenes generadas se introducirán en un *batch* para engañar a la red. Para ello debemos definir qué porcentaje de imágenes generadas habrá dentro de la mitad del *batch* etiquetada como imágenes reales. Este porcentaje puede ser definido de manera estática pero como se busca un mecanismo que se adapte a cada estado del entrenamiento se decide calcular de manera dinámica dependiendo del desempeño de la red discriminadora en su tarea de clasificar imágenes.

El cálculo de este porcentaje se basará en la precisión de la red discriminadora durante el entrenamiento de la misma. Al mismo tiempo se definirá un porcentaje máximo de imágenes generadas que se tratan como reales, en este caso un 10 % como máximo. En la primera iteración del entrenamiento se usará como porcentaje el máximo y posteriormente se irá calculando dinámicamente. Para este cálculo se quiere que el porcentaje de imágenes vaya desde el 0 %, si la precisión de la red es menor o igual al 50 % (la red no distingue imágenes reales de generadas) hasta el porcentaje máximo, si la red tiene una precisión del 100 % (la red distingue todas las imágenes).

La fórmula que se utilizará se puede observar en la ecuación 2

$$Perc_{Fake} = (Acc - 0{,}5)/0{,}5 * Perc_{Max} \quad (2)$$

donde $Perc_{Fake}$ es el porcentaje de imágenes calculado, Acc es la precisión de la red discriminadora para esa iteración del entrenamiento y $Perc_{Max}$ es el porcentaje máximo definido previamente.

Capas de *dropout* Otra de las técnicas que se pueden usar a la hora de controlar la red discriminadora es la aplicación de capas de *dropout*[27] entre sus capas densas. Al hacer uso de *dropout* se intenta evitar que la red discriminadora se especialice demasiado, e incluso introduciendo una probabilidad elevada (como del 50 %) se dificulta su aprendizaje directamente.

3.5.4. Métricas del entreno

Para tener una visión de la evolución de los distintos entrenos que se realicen se guarda diferente tipo de información del entreno y desempeño de las redes. Esta información sirve para evaluar si existe algún tipo de problema en los entrenamientos que se realicen. El primer paso para evolucionar y mejorar las redes que se diseñen es saber cómo funcionan estas.

Muestra de imágenes generadas: Como es lógico a medida que sucedan cada uno los entrenos que se decidan hacer se irán guardando una serie de imágenes de las que genera la red discriminadora. Para ello, simplemente basta con coger la red en un punto del entrenamiento y hacer una predicción para cierto vector de dimensión latente aleatorio. Para examinar más imágenes de una vez se mostrarán 25 imágenes por cada muestra de resultados. La figura 22 muestra un ejemplo de muestra de imágenes en la que se muestran 25 imágenes del *dataset*:

[27] El *dropout* es una técnica que se basa en que ciertas neuronas son elegidas aleatoriamente durante un entrenamiento y se bloquea su cambio de peso para dicho entrenamiento. Con esto tradicionalmente se intenta que existan ciertas neuronas que se sobre entrenen y se especialicen demasiado, dicho de otra forma, se fomenta que la red generalice y se previene el *overfitting*.

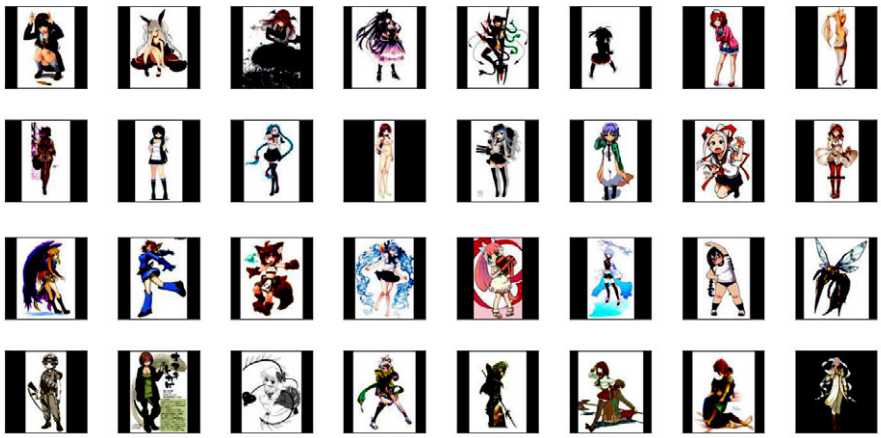
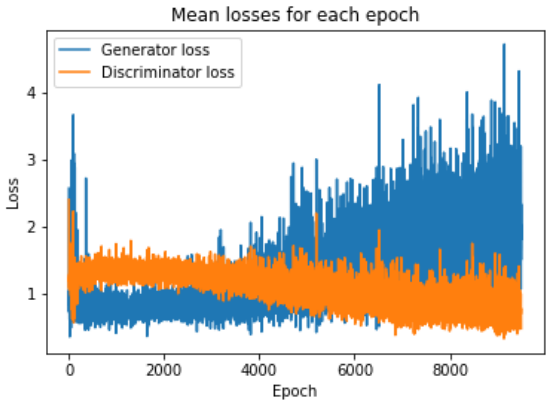
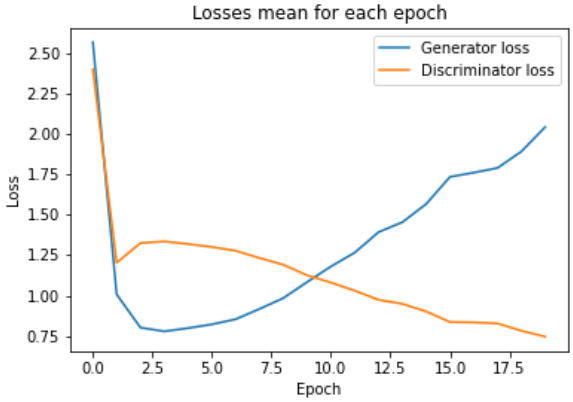


Figura 22: Ejemplo de muestra de imágenes.

Gráficos de pérdidas de las redes: De cada iteración de un *batch* se obtienen las pérdidas para la red generadora y discriminadora. Para tener una visión más clara que estos datos en texto se decide realizar una representación gráfica de cómo evolucionan las pérdidas de ambas redes. Se deciden dividir las gráficas en dos tipos: una para los valores de cada *batch* de entrenamiento que será más detallada pero para entrenos largos será menos legible, y una gráfica con los datos medios de cada *epoch* en la que se podrá observar de manera más clara la evolución de los datos a medida de un entrenamiento. La figura 23 muestra unas gráficas de pérdidas de ejemplo:



(a) Gráfica de pérdidas de un entrenamiento de 10.000 *batches*.



(b) Gráfica de pérdidas de un entrenamiento de 18 *epochs*.

Figura 23: Gráficas de crecimiento y decrecimiento exponencial.

Matriz de confusión para las redes discriminadoras: Además de eso se decide obtener la matriz de confusión[28] de la red discriminadora a lo largo del entrenamiento y así poder comprobar el comportamiento de la red discriminadora más en profundidad. Si existiese algún tipo de anomalía en las predicciones podríamos observarla gracias a la matriz de confusión.

Además, para facilitar la legibilidad se duplicará la matriz de confusión ge-

[28] Una matriz de confusión contiene los resultados de distintas clasificaciones de una red neuronal. Cada fila corresponde con una etiqueta de las reales y cada columna con una etiqueta predicha, los valores de cada elemento de la matriz son el número de predicciones realizadas para ese par de etiqueta real/predicha. Una red clasificadora perfecta siempre tendría una matriz de confusión en la que todos sus elementos se encuentren en posiciones donde las etiquetas real y predicha coincidan.

nerando dos matrices de modo que una muestre los datos en crudo y la otra los datos porcentuales respecto a las predicciones.

La figura 24 muestra un ejemplo de matrices de confusión con 2 posibles clasificaciones como en la investigación presente. En ella se puede observar cómo la mayoría de las clasificaciones son acertadas, agrupándose en la diagonal principal, mientras que hay 8 predicciones erróneas:

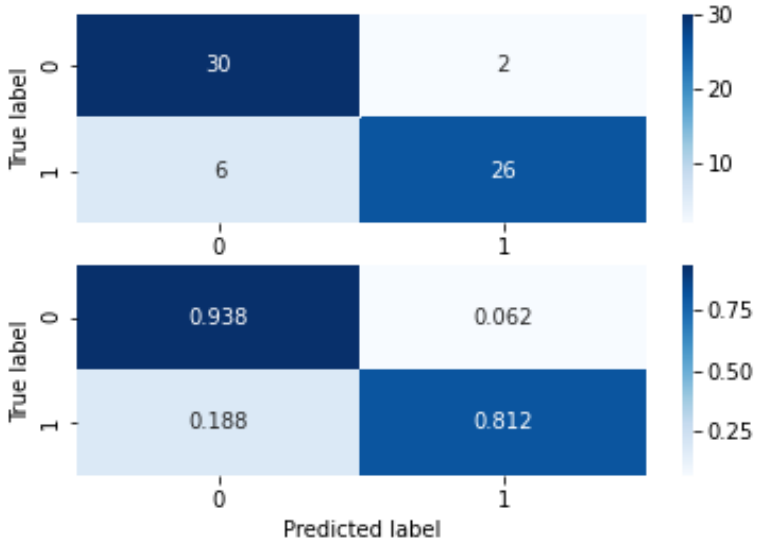


Figura 24: Ejemplo de matrices de confusión.

3.5.5. Estructura general de las redes discriminadoras

Las redes de este conjunto de entrenamientos se basan en la simplicidad de su estructura, el aspecto más importante de ellas es cómo responden al problema. Lo que se quiere observar durante estos entrenamientos es ver si ciertos hiperparámetros como las funciones de activación, el tamaño de *batch*, el tipo de optimizador y su *learning rate* o el tamaño de las capas interaccionan entre sí.

En general, la red discriminadora consta de un conjunto de capas densas de manera sucesiva, el número de neuronas de estas capas depende del entrenamiento en concreto. El número de neuronas de una capa se define siendo la mitad de neuronas de la capa anterior. Esto se realiza para ir guiando a la red a obtener una única clasificación correspondiente a si la imagen de entrada es real o generada.

Como función de activación de las capas ocultas se usa la *LeakyReLu* y para

la última capa que se usa una sigmoidal.

Para poder procesar la imagen con capas densas la imagen de entrada tiene que ser transformada en un vector unidimensional usando una capa de *Flatten*.

Para ciertos entrenamientos se usarán capas de *Dropout* por los motivos indicados en la sección 3.5.3. La probabilidad de *dropout* se define al 50 %.

La figura 25 muestra cómo se compone de manera general una red discriminadora de este conjunto de entrenamientos:

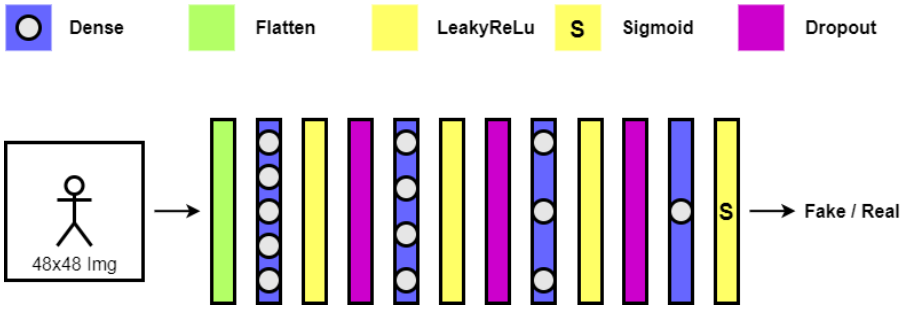


Figura 25: Estructura general de red discriminadora densa.

Esta será la estructura que se seguirá para los entrenamientos. Ciertos parámetros como el número de capas, la inclusión de capas de *Dropout* o el número de neuronas de las capas densas pueden sufrir modificaciones que servirán para comprobar cómo varía el funcionamiento de las redes al aplicarse.

3.5.6. Estructura general de las redes generadoras

Para intentar dotar de una mayor potencia a la red generadora que a la discriminadora se usarán capas convolucionales. Con ello se pretende que la red generadora esté más especializada que la discriminadora intentando equilibrar ambos entrenamientos.

Las capas convolucionales se configuran de tal manera que cada capa que se sucede disminuye el número de filtros y aumenta el tamaño de la imagen haciendo uso de capas de *Upsampling2D*. De esta manera se irá consiguiendo mayor resolución y menor número de filtros; en la última capa se formará una única imagen gracias a un filtro[29]. Por su parte el tamaño del núcleo será uno de los parámetros que se tomen en consideración para hacer pruebas sobre él, ya que hay estudios[15] que indican que este parámetro puede evitar el *overfitting*.

Como función de activación se utilizará la *ReLu* y, después de cada activación, se realizará una normalización de la información haciendo uso de la capa de *Batch Normalization*. Por último la activación de la capa de salida será una tangencial.

[29] El filtro de la capa de salida se encarga de agrupar los resultados de las capas anteriores y formar una imagen con un canal correspondiente al rango de blanco/negro.

Para pasar del vector de dimensión latente a comenzar a tratar imágenes se usará una capa densa que creará la información que posteriormente gracias a una capa de *Reshape* forme la primera imagen de la red.

La figura 26 muestra cómo se compone de manera general una red generadora de este conjunto de entrenamientos:

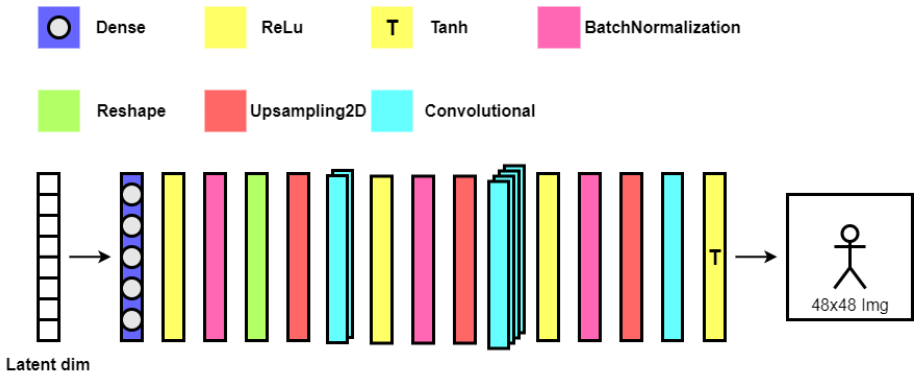


Figura 26: Estructura general de red generadora convolucional.

3.5.7. Evolución de los entrenamientos

Una vez definido cómo se realizarán los entrenamientos para esta primera aproximación se diseñan y entrenan varias redes. En cada entreno se intentan introducir mejoras en la arquitectura para así ir evolucionando los resultados y paliando los errores que van surgiendo.

La figura 27 muestra cómo evolucionan los entrenamientos que se realizan donde verticalmente y hacia abajo se produce la progresión:

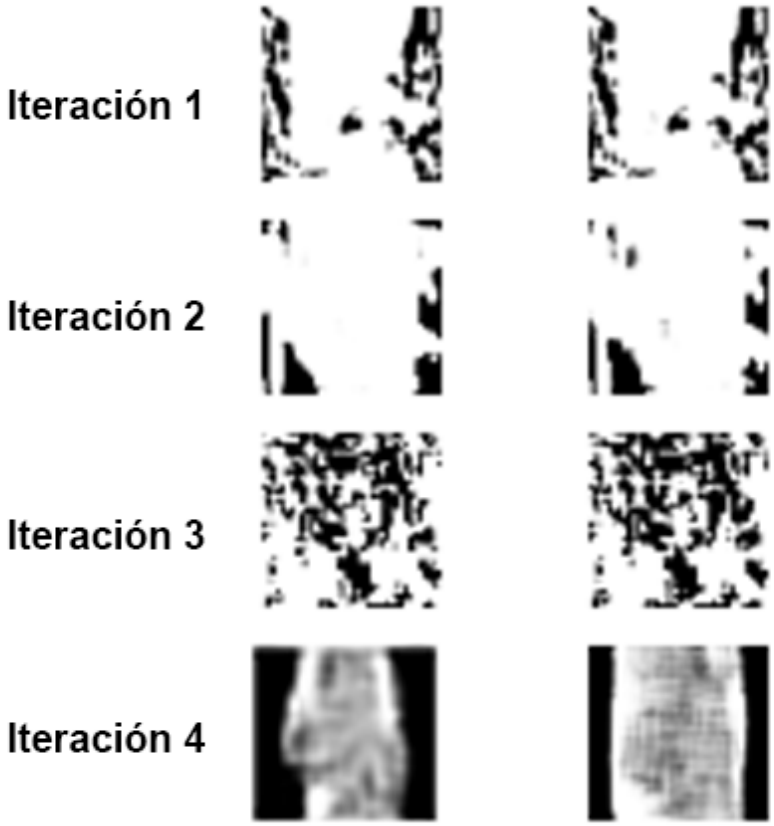


Figura 27: Evolución de los entrenamientos realizados.

Como se puede observar, los primeros resultados obtenidos constaban de formas aleatoriamente distribuidas, mientras que a medida que se hacen ajustes en la arquitectura se consiguen formar las bandas negras presentes en casi todas las imágenes e incluso se observan agrupaciones en mitad de la imagen en lo que podría ser una persona. Estos resultados constatan que las redes son capaces de tratar el problema e intentar replicar el *dataset*, pese a tener una calidad muy mala no se considera un mal resultado.

También queda demostrado que la poca potencia de las redes limita su avance pues las formas están muy difuminadas. Para proseguir con el avance de la investigación hay que hacer cambios profundos en las arquitecturas y evolucionar los modelos para llegar un poco más allá.

3.6. De la GAN *vanilla* a la CGAN

El siguiente paso para evolucionar y escalar los resultados es la inclusión de una arquitectura de redes convolucionales tanto para la discriminadora como para la generadora. Con ello se consigue dotar a nuestras redes de mecanismos más potentes y complejos para ir escalando poco a poco la estructura de las redes. Al mismo tiempo, se decide reducir aún más la dimensionalidad de las imágenes de entrada y estudiar si las redes tienen mayor facilidad a la hora de identificar características de las imágenes. Una vez conseguidos resultados con una menor resolución se pretende comenzar a escalar las redes para estudiar cómo reaccionan a mayores tamaños de imagen.

Para aumentar la potencia de la red se decide tomar como base un ejemplo de problema similar en el que se hayan obtenido buenos resultados previamente. Para ello estos cambios se hacen tomando como referencia el repositorio de *Github* titulado *Fashion-MNIST-GAN-Keras*[16] ya que dicho repositorio cubre un problema de generación de imágenes con una estructura y propósito similar al del presente trabajo.

3.6.1. Cambios en las estructuras de las redes

En este punto se busca tener unas estructuras más complejas que proporcionen un marco más complejo y especializado en la tarea de tratar imágenes bidimensionales. Se decide pasar a usar capas convolucionales en la red discriminadora, de esta forma la red discriminadora se intenta que esté más especializada en la tarea al mismo tiempo que se dota de mayor capacidad a la red generadora para no descompensar esta mejora.

Estos cambios se realizan con el propósito de escalar las redes intentando conseguir una mayor potencia que, en última instancia, logre unos mejores resultados.

3.6.2. Reducción de tamaño de las imágenes

Se decide estudiar un cambio en el tamaño de las imágenes que se usan, pasando de ser de 48x48 píxeles a 28x28 píxeles. Esta reducción tiene como objetivo simplificar el problema ya que al reducir el tamaño de la información se reduce. Antes de hacer efectivo este cambio, se deben observar qué cambios sufren las imágenes del dataset ya que la reducción puede ser demasiado grande y perderse información en el camino. Para ello observamos la figura 28 que muestra varias imágenes en las dos resoluciones que se quieren observar:

Figura 28: Imágenes antes (izquierda) y después (derecha) de reducir su resolución.

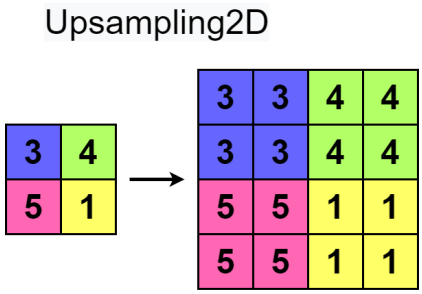
Se puede observar que las imágenes conservan su estructura general, en la reducción se han perdido gran cantidad de detalles, sin embargo la estructura y formas generales de los dibujos se mantienen. Si se quisiesen obtener imágenes de alta calidad esta resolución sería insuficiente, pero para el objetivo actual se quieren obtener imágenes que representen formas reconocibles y para ello esta resolución es suficiente. Más adelante si se obtuviesen buenos resultados simplemente sería necesario escalar las redes para que estas puedan procesar las nuevas resoluciones.

El plan a seguir es intentar obtener resultados para imágenes de la menor resolución e ir escalando las redes para ver cómo cambian dichos resultados cuando la resolución aumenta, comprobando si las redes son capaces de asimilar o no los nuevos detalles de imágenes más grandes. Para esto se usarán imágenes de 28x28 píxeles y, cuando se obtengan buenos resultados para dicha resolución, se pasará a usar imágenes de 48x48 píxeles, asímismo se pasarán a usar imágenes de 64x64 píxeles cuando sea conveniente.

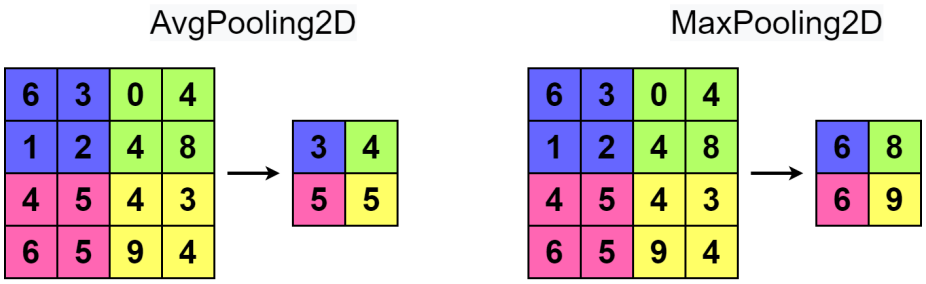
3.6.3. Cambio de dimensionalidad haciendo uso de los *strides*

Para realizar los cambios de dimensión entre las capas convolucionales dentro de una misma red anteriormente se hacía uso de las capas de *Upsampling2D*. Se decide cambiar este método por la reducción o aumento de dimensionalidad haciendo uso de los *strides* de las convoluciones.

La capa de *Upsampling2D* para doblar las dimensiones de una matriz y sus equivalentes *AveragePooling2D* o *MaxPooling2D* para reducir a la mitad dichas dimensiones trabajan de manera estática. En el caso del aumento de tamaño, la capa se encarga de duplicar el valor de cada pixel en las dos dimensiones mientras que para reducir se decide el valor de cada pixel por la media o el máximo de los cuatro píxeles adyacentes. La figura 29 muestra el funcionamiento de las capas descritas:



(a) Esquema de funcionamiento de la capa de *Upsampling2D*.



(b) Esquema de funcionamiento de las capas de *AveragePooling2D* y *MaxPooling2D*.

Figura 29: Esquema de funcionamiento de las capas de cambio de dimensión.

Estos mecanismos son ampliamente usados y dan buenos resultados pero se quiere dotar a las redes de mecanismos más complejos. En este sentido el cambio de dimensionalidad de una imagen se puede realizar junto a la propia convolución, creando imágenes de mayor o menor resolución al mismo tiempo que se realiza una convolución. Este mecanismo es interesante puesto que se permite a la red aprender cómo debe escalar o reducir la imagen. Para ello se modifica el parámetro *strides* a (2,2) haciendo que cada convolución duplique o divida a la mitad el tamaño de la imagen.

Esta manera de cambiar la dimensión de imágenes en redes GAN fue propuesta en el año 2015 por Jost Tobias Springenberg en al *paper* titulado *Striving for Simplicity: The All Convolutional Net*[17] donde se estudian las ventajas de sustituir capas de *MaxPooling* por convoluciones con *strides* incrementados.

3.6.4. Estructura general de las redes discriminadoras

La inclusión de capas convolucionales en estas redes hace que haya que modificar drásticamente la estructura de las redes discriminadoras que se diseñen. Durante los entrenamientos que se realicen con esta metodología se usarán capas convolucionales que serán las encargadas de realizar la labor de aprendizaje sustituyendo a las anteriores capas densas.

La estructura general de las nuevas capas convolucionales sigue el comportamiento inverso al de una red generadora: a medida que se sucedan las capas estas contarán con un mayor número de filtros y las imágenes que se traten serán cada vez de menor dimensión. Una vez se consiga una imagen del tamaño pequeño deseado se introducirá una capa de *Flatten* para crear un vector de dicha información y se clasificará con una capa densa de una neurona. La clasificación buscada será la que decida si la imagen de entrada era real o generada por el generador.

En cuanto a las funciones de activación se usará la *LeakyReLu* para activar las capas convolucionales Para la última capa densa encargada de hacer la clasificación se usará, igual que en la sección 3.5, la activación sigmoidal.

Del mismo modo se hará una normalización del *batch* después de cada activación y se mantendrán las capas de *Dropout* al 50 % de probabilidad pudiendo ser eliminadas para hacer pruebas.

El número de capas dependerá del tamaño de la imagen con la que se trate. Para imágenes más grandes se usarán más capas que para imágenes más pequeñas ya que será necesario procesar más la información de las mismas. En concreto se usarán dos capas convolucionales para imágenes de 28x28, tres para imágenes de 48x48 y cuatro para imágenes de 64x64. El número de filtros será uno de los parámetros con los que se juegue para probar distintas configuraciones optimizando la red y variará desde 32 hasta 128 filtros.

La figura 30 muestra cómo se compone de manera general una red discriminadora de este conjunto de entrenamientos:

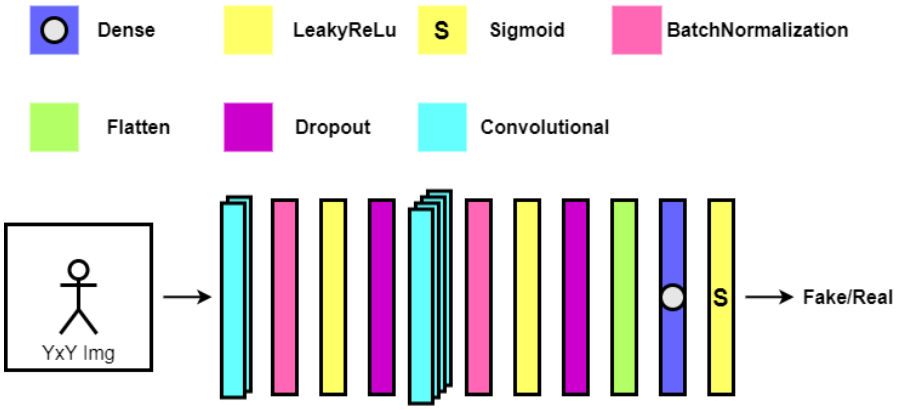


Figura 30: Estructura general de red discriminadora convolucional.

3.6.5. Estructura general de las redes generadoras

En general estas redes sufren pocos cambios respecto a la sección 3.5.6, el único cambio notable es la eliminación de las capas de *Upsampling* por la utilización de *strides* de (2,2).

Las funciones de activación se mantienen siendo la *ReLu* para las capas ocultas de la red y la tangencial o *Tanh* para las capas de salida. Por su parte se mantienen las capas de *BatchNormalization*.

Las convoluciones disminuyen a la mitad su número de filtros a medida que se suceden las capas y aumenta el tamaño de las imágenes con las que tratan. Asimismo el número de capas convolucionales es de tres para imágenes de 28x28 y 48x48, y cuatro para imágenes de 64x64.

La figura 31 muestra cómo se compone de manera general una red generadora de este conjunto de entrenamientos:

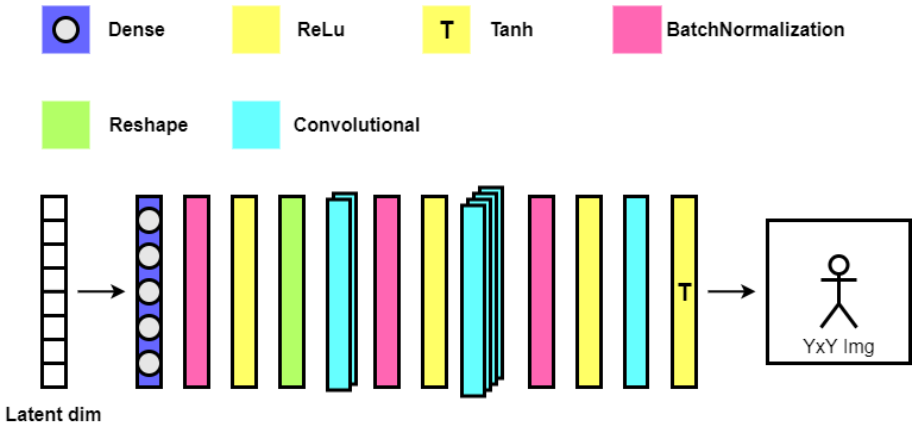
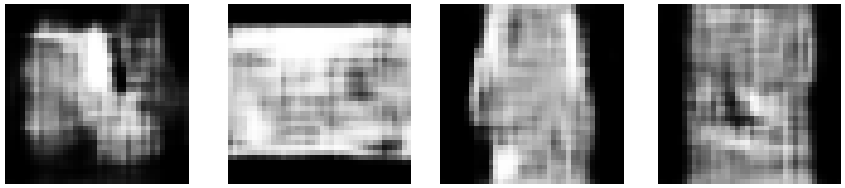


Figura 31: Estructura general de red generadora convolucional.

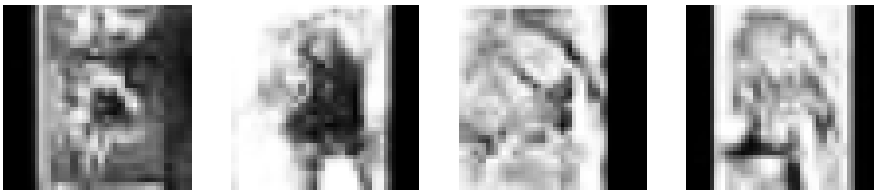
3.6.6. Evolución de los entrenamientos

El conjunto de entrenamientos que se realizan siguiendo la arquitectura descrita toma como bases los resultados de la anterior metodología. Por ello se considera interesante comparar los resultados obtenidos con los anteriores y observar si se han conseguido mejorar las producciones de las redes.

La figura 32 contiene las producciones de los anteriores entrenamientos y las nuevas para poder ser comparadas directamente:



(a) Producciones de la GAN *vanilla* para los entrenamientos de la sección 3.5.



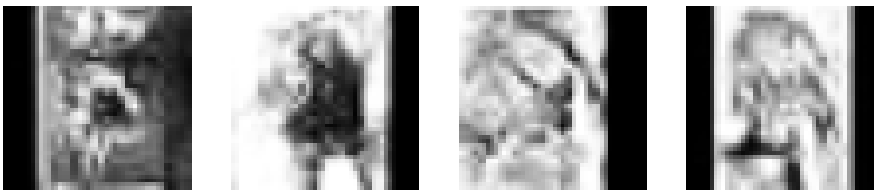
(b) Producciones de la CGAN.

Figura 32: Producciones de la GAN *vanilla* y la CGAN.

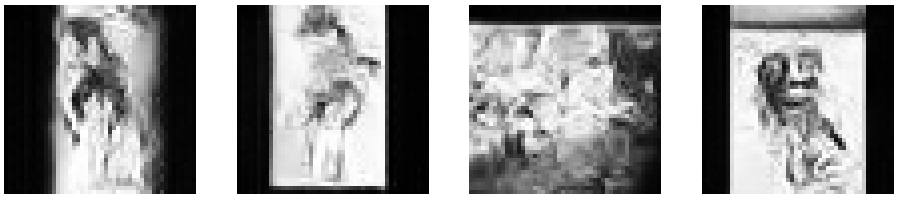
Se puede observar que la calidad de las imágenes generadas ha mejorado considerablemente, llegando a imitar formas humanas en las que se pueden identificar cabeza, piernas y cuerpo como en la segunda producción de la figura 32b. Esta mejora constata el buen funcionamiento de las nuevas redes, produciendo imágenes de mayor calidad gracias al uso de capas convolucionales.

Una vez observada la mejora producida en las redes se decide escalar estas para comenzar a generar imágenes de mayor resolución y estudiar esta evolución.

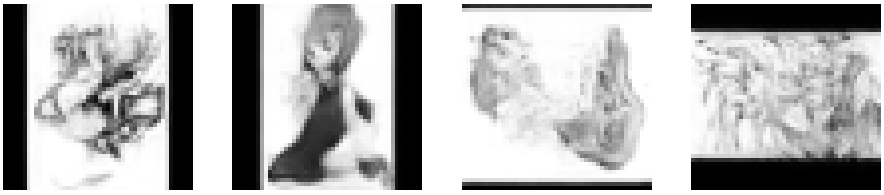
La figura 33 muestra la evolución de las imágenes generadas al aumentar la resolución de las mismas:



(a) Producciones de la CGAN para una resolución de 28x28.



(b) Producciones de la CGAN para una resolución de 48x48.



(c) Producciones de la CGAN para una resolución de 64x64.

Figura 33: Evolución de los entrenamientos realizados.

La evolución de las imágenes muestra como a mayor resolución se consiguen generar imágenes más detalladas. Las producciones de la figura 33c son un ejemplo de los buenos resultados que se pueden lograr con el uso de estas redes, en especial la segunda imagen que muestra claramente a una persona arrodillada mostrando sus brazos, piernas, pelo e incluso ojos.

Uno de los problemas que se pueden analizar es la composición de las imágenes. Como se puede ver la mayoría de ellas cuentan con una amalgama de formas que, a primera vista, parecen tener sentido pero si se atiende a los detalles se observa que no tienen una forma clara. Esto se puede identificar como que las redes son capaces de imitar el entorno y composición general de las imágenes del *dataset*, pero no detalles más concretos. Por así decirlo las redes son capaces de engañarse a sí mismas minimizando su error al imitar los aspectos más generales de una imagen, sin embargo no son capaces llegar a los detalles.

Este problema confirma que las redes están aprendiendo, pero pone de manifiesto que estas para minimizar su error se centran en el entorno de las imágenes como el fondo, las formas de cuerpos o elementos, el paisaje, etc. Esto provoca que las calidad se vea limitada pues las redes no son capaces de llegar más allá.

La valoración final de este conjunto de entrenamientos es que se ha conseguido evolucionar las redes anteriores para conseguir resultados, sin embargo esta evolución llega a un límite insuficiente para los propósitos de la investigación. Por todo esto se decide seguir mejorando las redes para intentar solucionar los problemas presentes.

3.7. Progressive Growing Of GAN

El siguiente paso en la investigación consiste en poner en práctica la metodología de entrenamiento llamada Progressive Growing Of GANs o ProGAN que fue propuesta en el año 2018 por Tero Karras en el *paper* titulado *Progressive Growing of GANs for Improved Quality, Stability, and Variation*[13] y que se basa en la sucesión de entrenamientos progresivos para la generación de imágenes. En dicho artículo se propone la nueva metodología de entrenamiento y se ponen en evidencia los buenos resultados que esta tiene para problemas de generación de imágenes.

La técnica consiste en realizar redes GAN que progresivamente vayan creciendo entrenamiento tras entrenamiento y, aprovechando los pesos guardados del entrenamiento anterior, escalar la red para obtener imágenes cada vez de mayor resolución. Es decir se realizan una serie de entrenamientos empezando por uno que consta de redes que generen una imagen a muy baja resolución, estos modelos se evolucionan añadiendo nuevas capas para aumentar la resolución de las imágenes con las que se entrena. De esta manera la red crece teniendo como referencia el entrenamiento anterior.

En la figura 34 se puede ver la evolución de las imágenes generadas con una ProGAN:

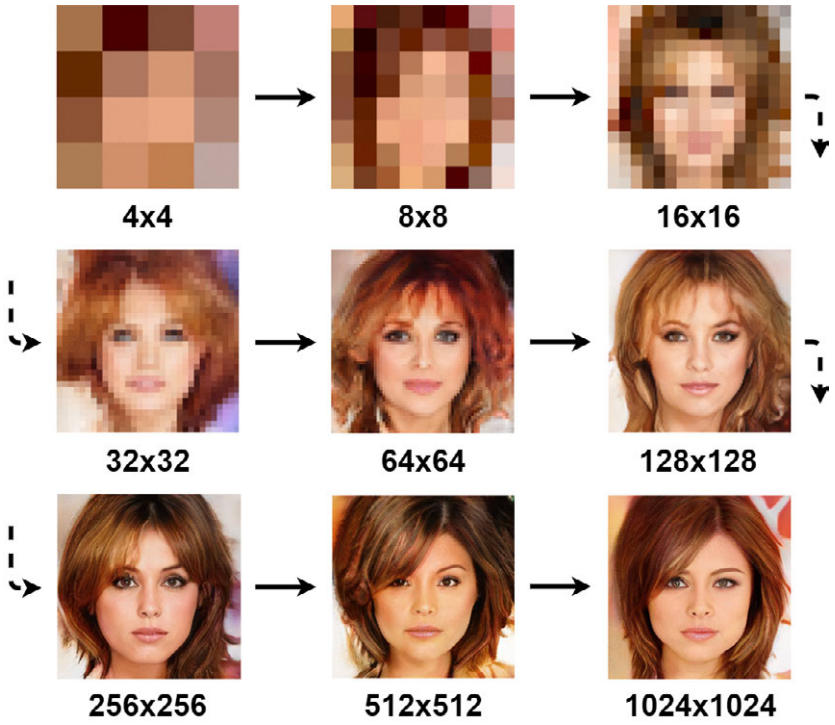


Figura 34: Imágenes creadas en los distintos entrenamientos de una ProGAN.

En la figura 35 se puede ver un esquema general de cómo se aplica esta técnica añadiendo nuevas capas progresivamente a una red tras cada entrenamiento consiguiendo así escalar sus resultados:

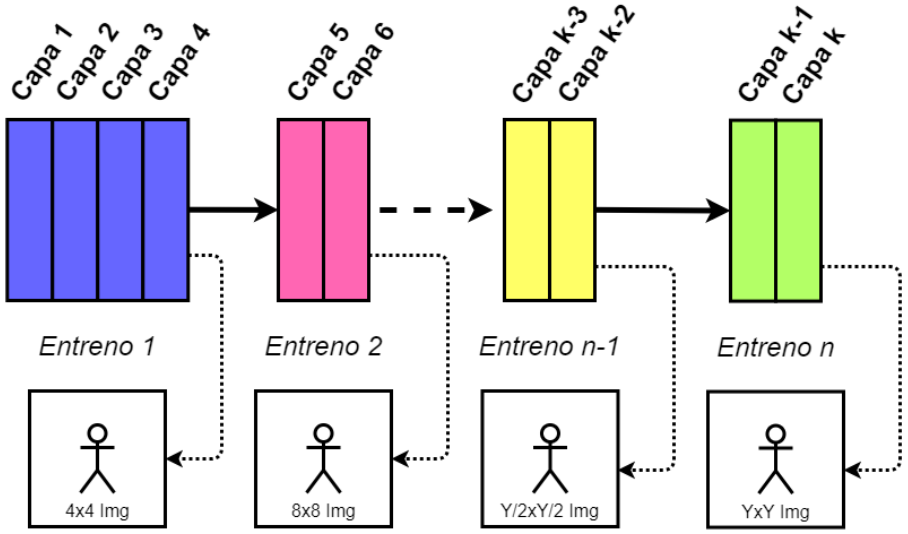


Figura 35: Esquema de los distintos entrenamientos de una ProGAN.

Para explicar la metodología de entrenamiento se hará un sumario con los distintos pasos que se realizan para una única red genérica, más adelante se detallarán las particularidades que puedan surgir. Con esto podemos dividir los pasos que esta sigue en los siguientes:

1. **Entrenar la red:** La primera red que se vaya a entrenar marca el mínimo tamaño de imagen. Este tamaño será con el que se comiencen la serie de entrenamientos y, más adelante, irá evolucionando para lograr mayores tamaños. El entrenamiento en cuestión se realiza con normalidad sin ninguna característica especial.

2. **Guardar los pesos:** El estado de entrenamiento de la red entrenada ha de ser guardado para su posterior reutilización. Este guardado se realiza de la misma manera que para un entrenamiento normal. Una red se define por su estructura y pesos y por lo tanto con esos datos se vuelve a reconstruir la red.

3. **Cargar la red anterior:** Una vez terminado el primer entrenamiento, hay que combinar los resultados de este con el próximo que se pretenda realizar. Para ello es necesario cargar de nuevo la red del primer entrenamiento con el propósito de modificarla y volver a entrenar con la nueva estructura. Dicho esto, se carga la estructura y pesos de la primera red y se comienza su modificación.

4. **Eliminar las capas innecesarias:** Para escalar la red que se ha entrenado lo primero es eliminar las capas que marcan el tamaño final de las imágenes, ya que estas capas limitan el crecimiento. En la red discriminadora estas capas son las primeras mientras que en la red generadora son las últimas. Estas capas son las que están conectadas directamente con las imágenes del *dataset* que se vayan a utilizar. En el presente trabajo estas capas que son eliminadas son las capas convolucionales que se relacionan directamente con la información de entrenamiento bien generándola o discriminándola. Cuando se eliminan estas capas cualquier capa relacionada con esta también queda eliminada como capas de *dropout* o de normalización. Al eliminar estas capas se le está dando la libertad a la red de crecer añadiendo nuevas capas que traten imágenes de mayor tamaño. Más adelante en la secciones 3.7.1 y 3.7.2 se explicará detalladamente el por qué de la eliminación de estas capas. En la figura 36 podemos ver un caso práctico de eliminación en el que la última capa de una red generadora es eliminada porque es la encargada de generar la imagen de salida de la red:

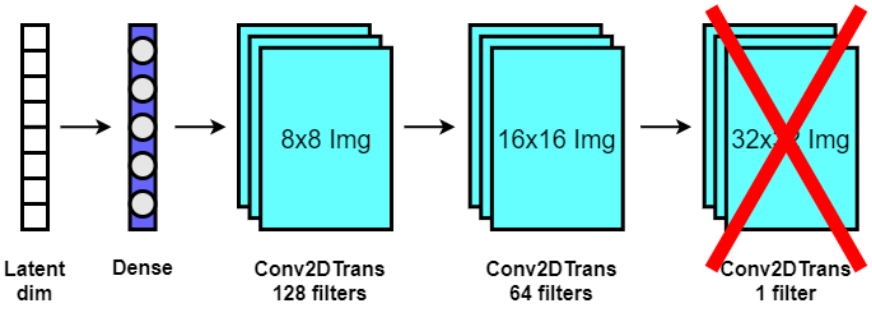


Figura 36: Red generadora en la que su última capa convolucional de un filtro es eliminada.

5. **Añadir las nuevas capas:** Para crear la nueva red hay que aprovechar las capas eliminadas en el paso anterior y, en su posición, colocar las nuevas capas que están destinadas a aumentar el tamaño de las imágenes que se generen. Las nuevas capas ocupan el lugar que ha quedado libre por la eliminación y consiguen recoger la información de las capas que se quedan libres y aprovecharla para tratar imágenes de mayor tamaño.

La figura 37 muestra cómo es reconstruida la red del anterior ejemplo para poder obtener imágenes con mayor resolución:

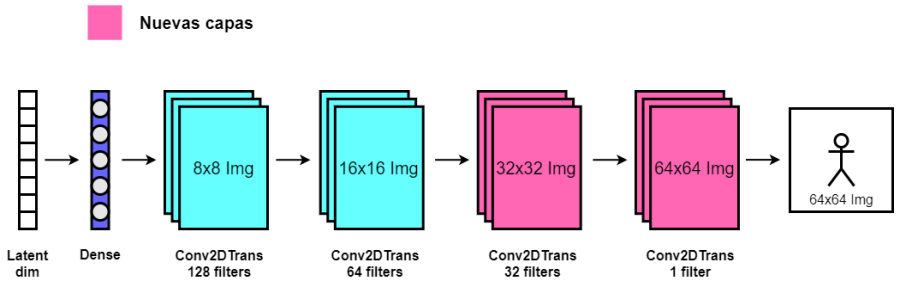


Figura 37: Red generadora en la que aumenta el tamaño de sus generaciones respecto al anterior entrenamiento.

6. **Entrenar la red de nuevo:** El nuevo modelo se vuelve a entrenar con las nuevas capas incluidas. Las capas pertenecientes al anterior entrenamiento no se congelan[30] para que de esta manera siga su entrenamiento pudiendo asimilar nuevas características de las imágenes y adaptarse a las nuevas capas introducidas. El *paper* donde se introduce este modelo[13] especifica que en cada entrenamiento se entrenan todas las capas para que el modelo se pueda adaptar al crecimiento progresivo de la red.

   A partir de este punto el proceso se repite durante tantos entrenamientos como sea necesario para obtener la resolución deseada.

[30] Congelar una capa significa impedir que sus pesos varíen, es decir, parar su entrenamiento. Muchas metodologías de entrenamiento congelan las capas de un modelo pre-entrenado y realizan los nuevos entrenamientos sólo en las nuevas capas, sin embargo esto no sucede así en una ProGAN.

Para recapitular sobre el esquema de entrenamiento de las ProGAN la figura 38 muestra un breve resumen de los principales pasos en los que se basa la metodología:

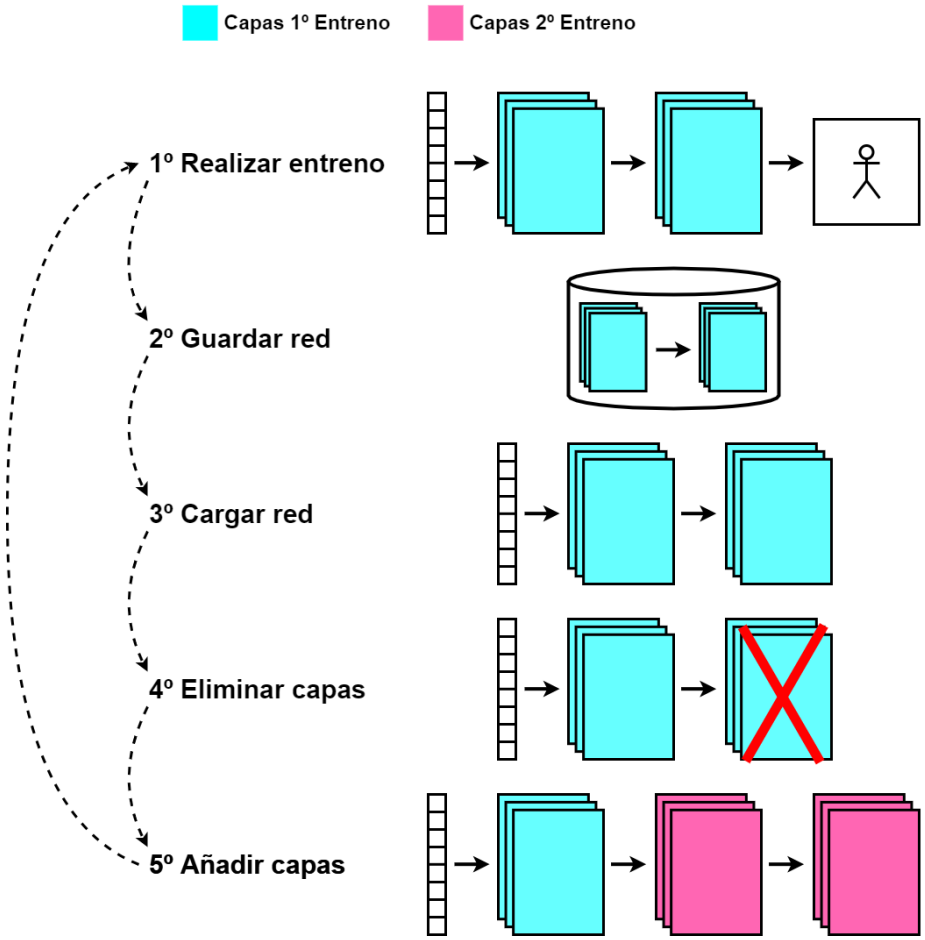


Figura 38: Esquema resumen del modelo de entrenamiento de una ProGAN.

3.7.1. Escalado de la red discriminadora

Una vez conocida la forma general de entrenamiento de una arquitectura ProGAN hay que atender a los pequeños detalles que afectan a cada una de las redes que se van a diseñar y cómo hacer que estas encadenen entrenamientos uno detrás de otro.

La red discriminadora tiene que adaptarse a tamaños de imágenes de entrada que de un entrenamiento a otro cada vez son mayores. Para ello las capas convolucionales que se vayan incluyendo a la red después de cada entrenamiento tienen que ser de mayor tamaño. Al incluir capas que soporten imágenes de mayor tamaño se van escalando los resultados tras cada entrenamiento pero dicha inclusión de capas no se puede hacer de manera directa.

En este punto es donde entra en juego la eliminación de capas. Para ello se va a ver una red de ejemplo en el que por simplificar sólo se incluyen las capas convolucionales. La figura 39 muestra la estructura de la red discriminadora de ejemplo para imágenes de 8x8:

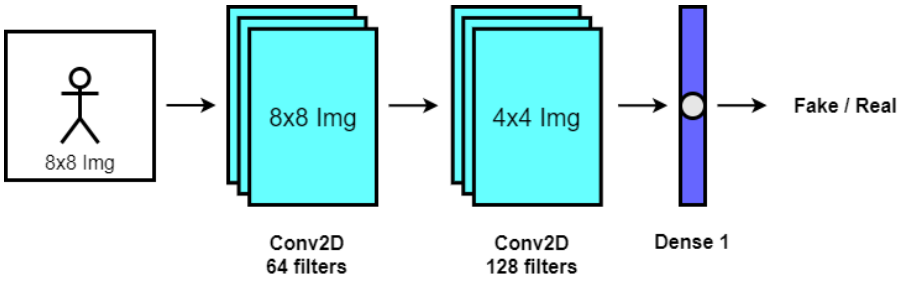


Figura 39: Ejemplo de estructura de una red discriminadora.

Como se puede observar la red realiza una reducción de dimensionalidad de 8x8 a 4x4 y después de esa información hace la clasificación de imagen real o generada. Si se quisiese escalar la red directamente insertando nuevas capas convolucionales no sería posible por un fallo en las dimensiones de entrada de la red entrenada.

Si se atiende a la entrada de la primera capa de la red, esta espera una dimensionalidad de 4x4x1 que es la imagen con la que se realiza su entrenamiento; sin embargo si se le cambia dicha imagen por una capa convolucional con un número de filtros n siendo n>1 las dimensiones serían incompatibles. Al haber entrenado la red con una entrada de 4x4x1 su composición de entrada estará adaptada a un único canal de imagen pero si las nuevas capas que se introducen a la red tienen un número de filtros distinto de 1 se produce una incompatibilidad.

Para solucionar este problema hay que introducir unas capas que hagan de *proxy* entre el antiguo y nuevo modelo. En este caso, se introduce una nueva capa al principio de la red. Esta capa ha de tener el mismo número de filtros que la capa por la que será sustituida en el nuevo entrenamiento y la misma dimensionalidad de salida, al mismo tiempo tiene que tener como entrada las imágenes del *dataset* del entrenamiento y por lo tanto no puede reducir dimensionalidad.

La figura 40 muestra la inclusión de la nueva capa en el modelo, para una mayor claridad y realizar un seguimiento exhaustivo en cada capa se especifica su dimensionalidad de entrada o salida:

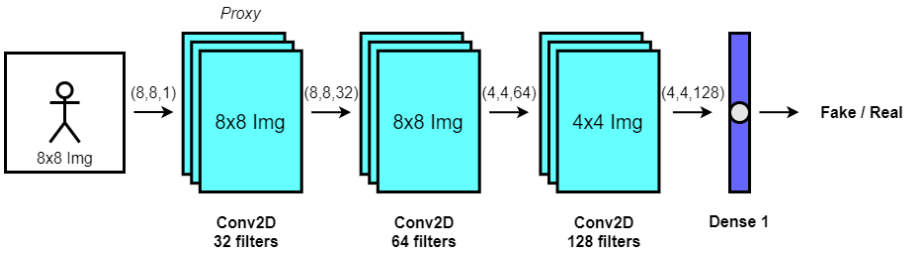


Figura 40: Introducción de capa *proxy* a la red discriminadora.

La nueva capa de *proxy* se elimina como anteriormente se ha indicado en el siguiente entrenamiento. De esta forma la dimensionalidad de salida de las nuevas capas y de entrada de las antiguas coincide haciendo posible el nuevo entrenamiento. En la figura 41 se muestra cómo se elimina la capa de *proxy* y se introducen nuevas capas al modelo permitiendo su entrenamiento de nuevo. Cabe destacar que el nuevo modelo tiene su propia capa de *proxy* para enlazar el entrenamiento actual con el siguiente:

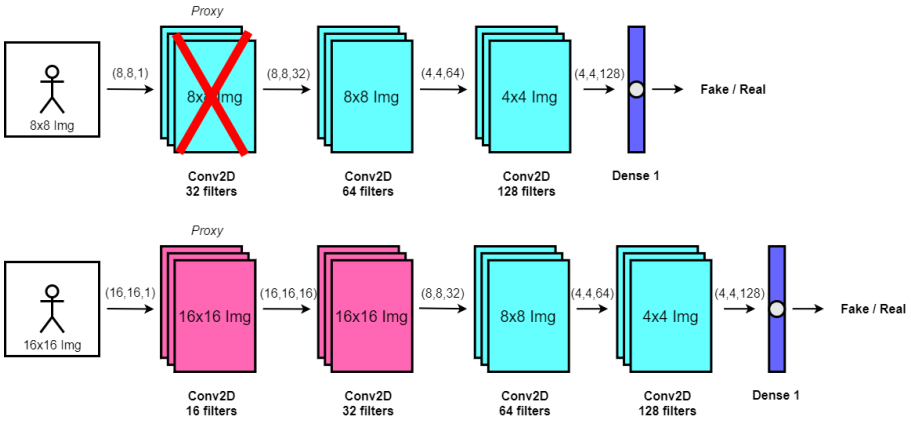


Figura 41: Combinación de entrenamientos añadiendo al antiguo modelo nuevas capas.

3.7.2. Escalado de la red generadora

De la misma forma que la red discriminadora necesita especial atención a la hora de ser diseñada para que la sucesión de entrenamientos no genere problemas, la red generadora tiene sus propias características. El crecimiento de la red generadora se realiza en las últimas capas de la red para así ir generando imágenes de mayor resolución a medida que se suceden los distintos entrenamientos. Estas últimas capas son las encargadas de que cada vez se generen imágenes de mayor tamaño.

Para observar la forma general de la red generadora y cómo afrontar el problema de dimensionalidad se observa la figura 42 que muestra un ejemplo de una red generadora simplificada:

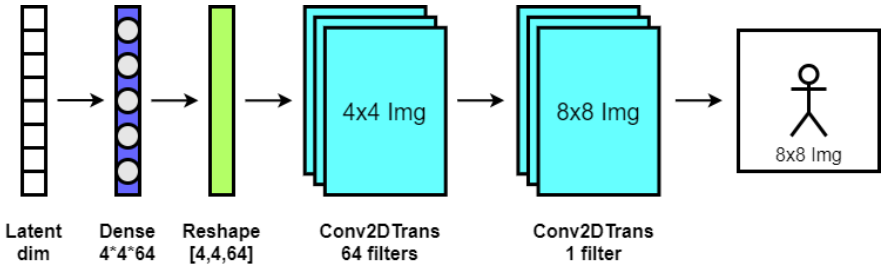


Figura 42: Ejemplo de estructura de una red generadora.

En concreto según se han diseñado las redes de la presente investigación la capa de salida de la red generadora consta de una capa convolucional de 1 filtro. Esta capa es la encargada de aunar la información de las capas anteriores para generar una única imagen de salida en la red. Esta capa servirá además como *proxy* para encadenar los diferentes entrenamientos.

Por lo tanto, y habiendo definido qué capa actúa como proxy, el esquema de eliminación y sustitución de capas para la red generadora del ejemplo se puede observar en la figura 43:

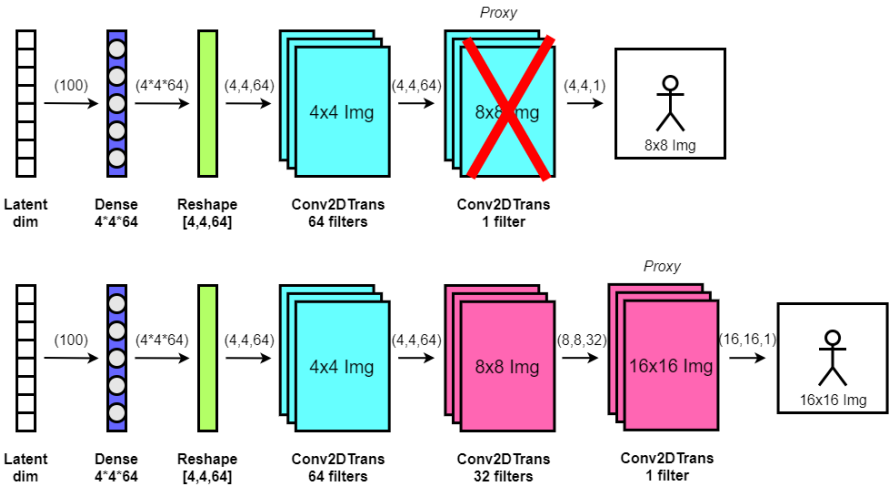


Figura 43: Combinación de entrenamientos añadiendo al antiguo modelo nuevas capas.

3.7.3. Nuevas capas

La arquitectura ProGAN propone la inclusión de nuevas capas diseñadas para que funcionen en la nueva arquitectura. Estas capas se diseñan bien para sustituir a capas que se han usado en los anteriores entrenamientos o bien para hacer nuevas funciones.

Las nuevas capas que se proponen en el *paper*[13] en el que se propone esta metodología son las siguientes:

- ***Pixel Normalization* o *Pixelnorm*:** Como sustitución a las capas de *BatchNormalization* usadas en otras arquitecturas se propone la capa denominada *Pixel Normalization*. Su labor es normalizar los valores de los píxeles de una matriz a una longitud fija para evitar que la magnitud de las señales se dispare. Esta normalización se basa en la propuesta por Alex Krizhevsky en el año 2012 en el *paper* titulado *ImageNet Classification with Deep Convolutional Neural Networks*[18] y su fórmula se describe en la ecuación 3

$$b_{x,y} = \frac{a_{x,y}}{\sqrt{\frac{1}{N}\sum_{j=0}^{N-1}(a_{x,y}^{j})^2 + \epsilon}} \tag{3}$$

  donde $\epsilon = ^{10-8}$ N es el número de canales y $a_{x,y}$ y $b_{x,y}$ son el valor de entrada y normalizado de cada pixel respectivamente.

- ***Minibatch Standard Deviation*:** Esta capa se encarga de generar muestras con una menor variación que la del *dataset*. Se usa únicamente en las redes discriminadoras. Su uso ayuda a distinguir a las redes de qué *batches* son reales de cuáles no propiciando que la red generadora produzca más variedad de imágenes evitando, entre otros, el colapso modal[31].

  El cálculo que realiza esta capa es la desviación estándar de los píxeles de un mapa a través del *batch*. Dicho valor lo concatena como un canal extra.

3.7.4. *Fade in* entre entrenamientos

Además de los mecanismos ya mencionados la metodología introduce como mecanismo de combinación de entrenamientos el denominado *Fade in*. El objetivo de este es que cada vez que se vayan a entrenar nuevas capas en un modelo estas sean gradualmente incorporadas evitando así un cambio brusco respecto al modelo anterior.

El mecanismo se basa en coger la salida del entreno anterior, hacer un *Upsampling2D* de esta y combinarla con la salida de las nuevas capas. Esta combinación se hará de manera que durante los primeros *batches* tengan más importancia las producciones de los modelos ya entrenados y gradualmente se vaya desvaneciendo esta mezcla hasta que las producciones vengan sólo de las nuevas capas.

Para implementarlo basta con obtener como salidas de la red la generada por el modelo preentrenado y aplicarle el *Upsampling2D*, por otra parte obtener la salida generada por las nuevas capas. Una vez con ambas salidas hay que generar una nueva salida multiplicando cada matriz de salida por un valor del rango [0,1], el valor de esta multiplicación viene dado por un parámetro denominado α.

[31] El colapso modal es un problema que sucede cuando las redes generadoras encuentran un valor en el que minimizan su error a cambio de producir siempre el mismo tipo de imágenes. Esto provoca que sus producciones sean siempre iguales con muy ligera variación.

La salida de las capas pre-entrenadas se multiplica por (1-α) mientras que la salida de las capas nuevas se multiplica por α. Estas dos salidas son sumadas para formar la nueva imagen. De esta forma se consigue formar una nueva imagen en la que la imagen generada por las nuevas capas supone un α % de la misma. A medida que se suceden los *batches* se aumenta el valor de α hasta llegar a 1, punto en el que la salida es la obtenida por las nuevas capas.

La figura 44 muestra un esquema de cómo se estructura el mecanismo de *Fade in* en una red generadora:

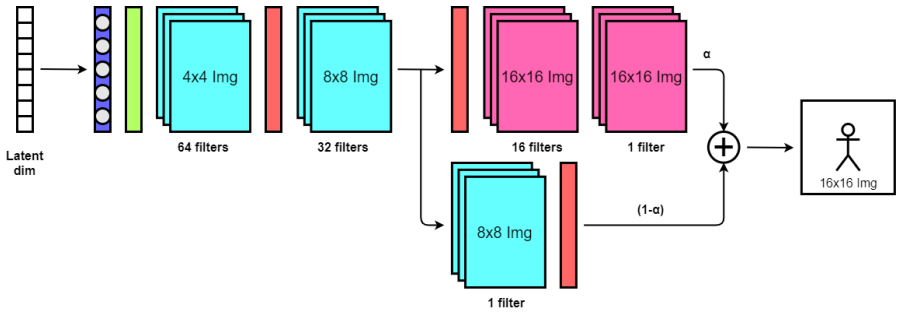


Figura 44: Esquema de disposición de capas en una red generadora con *Fade in*.

Gracias a esto se consigue que, a medida que se desarrolla el entrenamiento, se haga poco a poco una combinación de los resultados del anterior entrenamiento que forman una imagen de buena calidad con la salida de las nuevas capas que aún están sin entrenar y por lo tanto generan imágenes peores. La filosofía es usar la red preentrenada para guiar a la nueva poco a poco disminuyendo la variabilidad en el entreno previniendo el surgimiento de problemas como el *Gradient Explosion*.

El valor durante un entrenamiento de α es susceptible de sufrir gran cantidad de cambios durante las pruebas pero, por definir un punto de partida, se establece que α valga un 20 % al principio del entrenamiento y vaya aumentando a razón de 0.008 % en cada *batch*. De esta forma tomarán 10.000 *batches* hasta llegar a que el 100 % de la imagen provenga de las nuevas capas. Estos valores pueden variar, por ejemplo para entrenamientos más cortos se puede definir el paso a 0.016 y que el proceso dure 5.000 *batches*.

3.7.5. Filtro más exhaustivo del *dataset*

Para intentar simplificar el problema se decide intentar normalizar aún más las imágenes que se utilizan del *dataset*. Para ello se deciden aplicar distintos tipos de filtros que tienen por objetivo que las imágenes sean más parecidas compositivamente entre sí.

Elección de etiquetas Hasta ahora las etiquetas de las imágenes de los entrenamientos son *1girl* y *solo* pero si se quiere simplificar las imágenes a replicar hay que hacer un mayor filtrado. Para ello se decide añadir al filtrado las siguientes etiquetas:

- ***White_background***: Para evitar que haya mucha diferencia en los entornos de las distintas imágenes y favorecer que las redes se centren en el personaje dibujado, se incluye esta etiqueta al filtro. Si todas las imágenes tienen un fondo blanco es más difícil que las redes creadas se centren en elementos del entorno y con ello se consigue centrar el foco en el personaje dibujado.
- ***Full_body***: Esta etiqueta hace que las imágenes que se generen tengan elementos como brazos o piernas que sean fácilmente identificables y sirvan para estudiar el rendimiento de las redes.

Con este nuevo filtrado se consigue reducir la variabilidad entre distintas imágenes y con ello simplificar el problema a tratar.

Filtrado de barras negras Para evitar que las bandas negras tengan demasiada importancia a vista de las redes se filtran las imágenes con bandas negras. Para ello se deciden eliminar de las imágenes que se usan en los entrenamientos las imágenes con bandas horizontales. La elección en concreto de bandas horizontales es debido a que hay menos imágenes con dichas bandas y por lo tanto se hace un filtrado de menos imágenes.

Como consecuencia de la nueva elección de imágenes se pasan a usar 3.400 imágenes del *dataset* original para los entrenamientos. Esta gran reducción puede provocar otros problemas pero se considera que 3.400 es un número suficiente para los entrenamientos que se realizarán.

3.7.6. Uso de imágenes a color

Hasta ahora se habían usado para los entrenamientos imágenes con un único canal de color correspondiente al blanco y negro. En este punto de la investigación en el que se han conseguido resultados con una cierta calidad se considera interesante pasar a usar imágenes a color. A la hora de obtener dichas imágenes no hay ningún problema pues el *dataset* original está en RGB.

Este cambio propiciará que, a la hora de crearse las imágenes además de mostrar su forma se puedan identificar sus colores. Esto es especialmente interesante, pues por los colores se puede analizar si un elemento es una cosa u otra. Por ejemplo se puede identificar más fácilmente el vestido de un personaje si este es azul o rojo que si se muestra en blanco y negro. En pocas palabras gracias al color se puede identificar más fácilmente las intenciones de las redes.

Como contrapartida, la incorporación de tres canales de color implica que las imágenes contengan más información, como se ha indicado anteriormente y en el anterior TFG[7] se estudió. Esto genera que las redes tengan una mayor dimensión, pues necesitan de más parámetros para tratar la información. Pese a esto se considera beneficioso el cambio y, si fuese necesario, en un futuro se podría volver a usar las imágenes en blanco y negro.

En cuanto a la estructura de las redes esta no cambia prácticamente nada para adaptarse a las nuevas imágenes. El único cambio se produce en las capas convoluciones, donde en las generadoras pasan de uno (blanco y negro) a tres (RGB) filtros en la última capa que genera la imagen y el discriminador cambia su dimensionalidad de entrada.

3.7.7. Estructura general de las redes discriminadoras

Como se ha indicado anteriormente, al aplicar esta metodología de entrenamiento será necesario diseñar, crear y entrenar varias redes para cada entrenamiento que se quiera realizar. Con cada par de redes, generadora y discriminadora, creadas se tratarán imágenes de cierta resolución. Una vez entrenadas dichas redes se pasarán a redes que procesen imágenes del doble de tamaño, reutilizando las estructuras entrenadas de las primeras. Por así decirlo, para llegar a la generación de imágenes de cierto tamaño hay que hacer varios pasos previos entrenando redes de menor dimensión.

En general la decisión de usar unas u otras capas viene dada por la arquitectura de las ProGAN. Por ello capas que antes se usaban pasan a ser eliminadas o sustituidas por otras.

La activación de las capas ocultas continúa siendo la *LeakyReLu* y para la capa de salida la función sigmoidal, igual que en las secciones 3.5 y 3.6.

Para realizar los cambios de dimensionalidad se usarán capas de *AveragePooling2D* o *AvgPooling2D*.

Cabe destacar la inclusión de la nueva capa de *MinibatchStdev* antes de la última convolución para generar muestras con una menor variación.

En cuanto a las capas convolucionales, el número de filtros será inicialmente de 256 para tratar con imágenes de 16x16, 8x8 y 4x4 píxeles y, desde ahí se reducirá a la mitad con cada duplicación de tamaño, quedándose en un valor fijo de 64 filtros a partir de imágenes de 64x64 píxeles.

La figura 45 muestra cómo se componen de manera general las redes discriminadoras de un conjunto de entrenamientos que va desde imágenes de 8x8 hasta 32x32 píxeles:

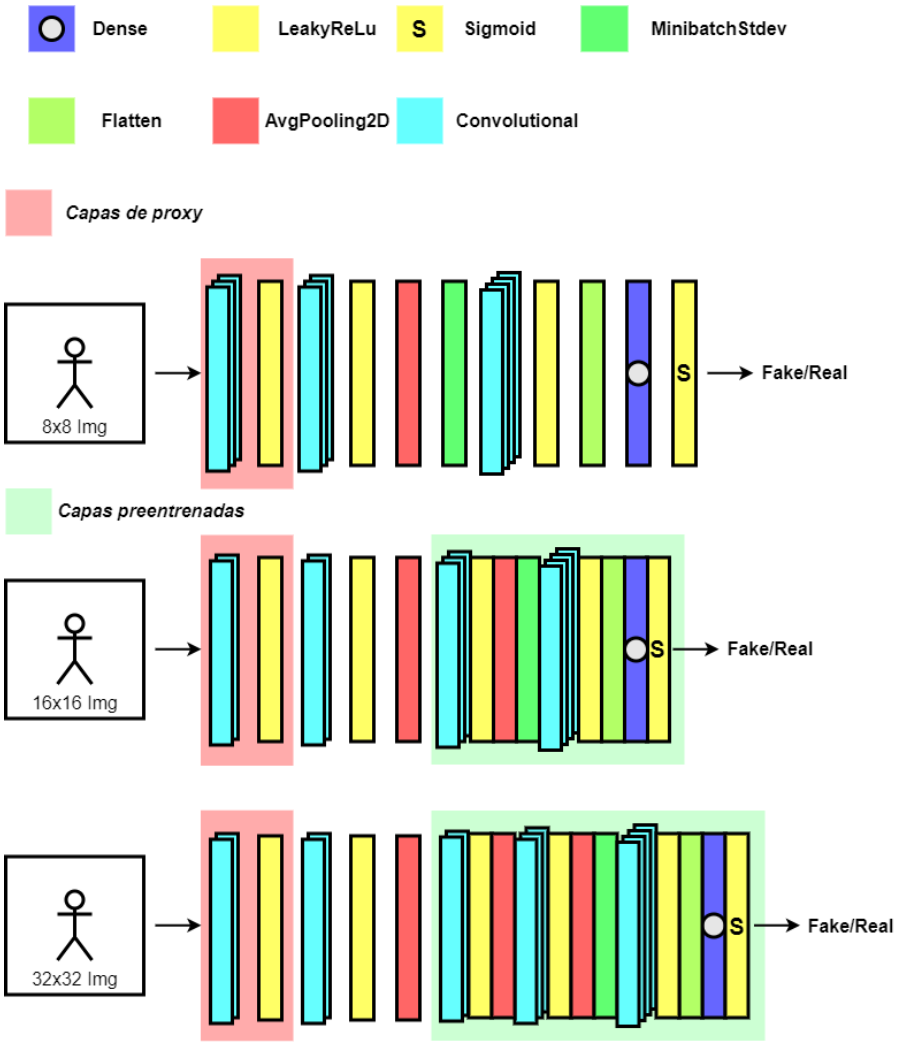


Figura 45: Estructura general de las redes discriminadoras de un conjunto de entrenamientos desde 8x8 hasta 32x32 píxeles.

El esquema muestra la forma general que tienen las redes de escalar para conseguir procesar imágenes del doble de dimensión. Para las pruebas que se realicen si se quisiesen escalar las redes sólo habría que seguir el esquema de crecimiento.

3.7.8. Estructura general de las redes generadoras

La estructura de crecimiento de las redes generadoras sigue la misma filosofía que para las redes discriminadoras. La diferencia más esencial entre ambas redes es que a las redes generadoras se le aplica el mecanismo de *Fade in* entre entrenamientos[32].

Como funciones de activación de las capas ocultas de las redes se usa la *LeakyReLu* igual que en las redes discriminadoras. Para la capa de salida se mantiene la función tangencial como en las secciones 3.5 y 3.6.

Para realizar los cambios de dimensionalidad se usarán capas de *Upsampling2D*.

El número de filtros de las capas convolucionales se estructura de la misma manera que para las redes discriminadoras, usando 256 para tratar con imágenes de 16x16, 8x8 y 4x4 píxeles y reduciendo a la mitad a medida que las imágenes aumentan hasta un tope de 64 filtros a partir de imágenes de 64x64 píxeles.

[32] En el *paper* en el que se propone la metodología se aplican capas de *Fade in* tanto para las redes generadoras como para discriminadoras. Como este mecanismo intenta combinar entrenamientos de manera estable, es más eficiente en las redes generadoras, que de manera natural son menos estables. Sin embargo se decide no utilizar para las redes discriminadoras por simplicidad y porque se considera que el esfuerzo de aplicar dichas capas no merece la pena. Si fuese necesario en un futuro se podría estudiar la aplicación de estas capas a las redes discriminadoras.

La figura 45 muestra cómo se componen de manera general las redes generadoras de un conjunto de entrenamientos que va desde imágenes de 8x8 hasta 32x32 píxeles:

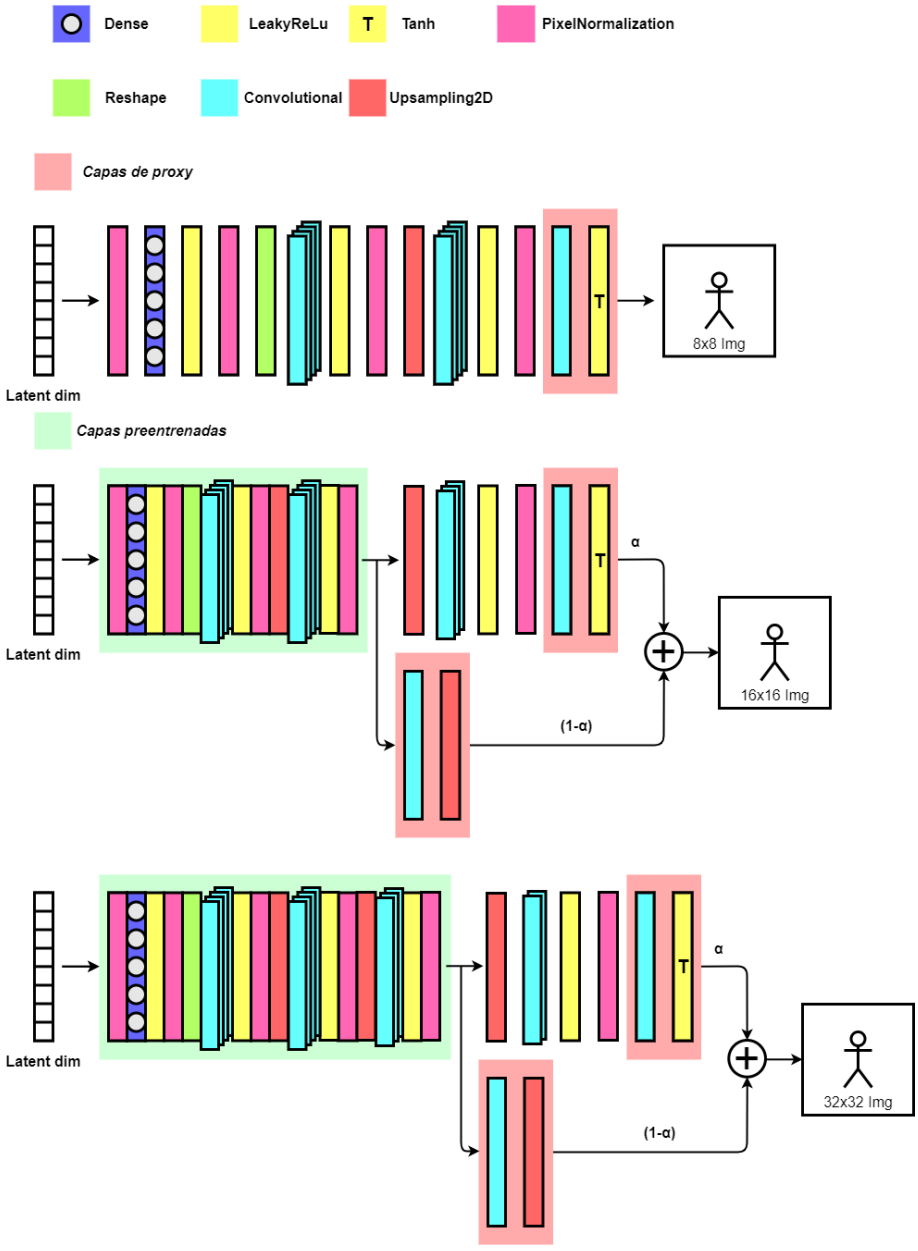


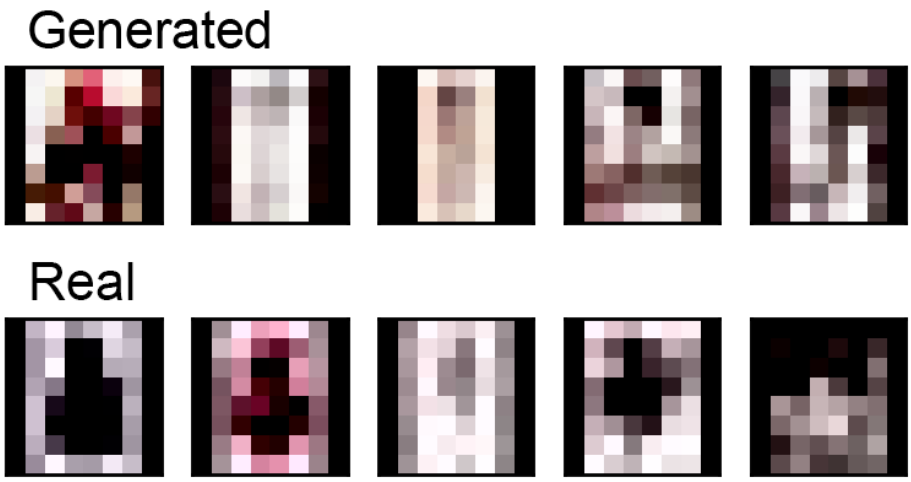
Figura 46: Estructura general de las redes generadoras de un conjunto de entrenamientos desde 8x8 hasta 32x32 píxeles.

De la igual modo que para las redes discriminadoras, las imágenes que se tratan pueden seguir escalando siguiendo la misma estructura de crecimiento que la presentada.

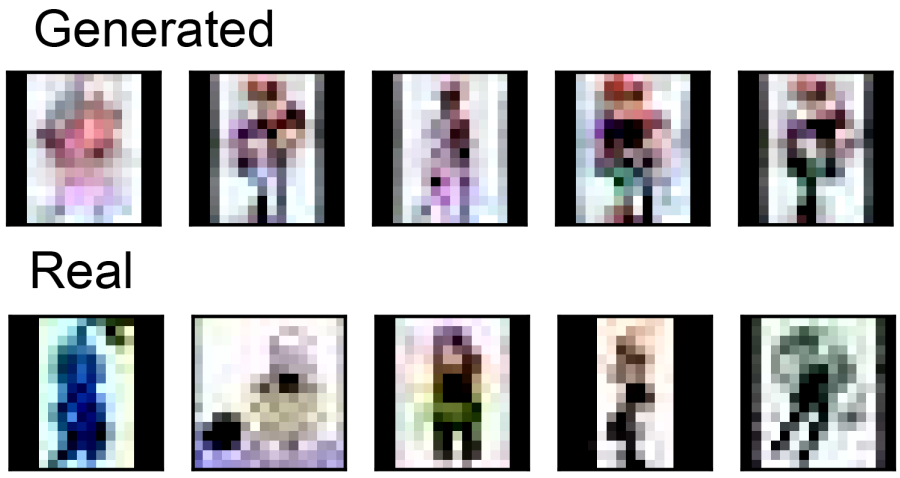
3.7.9. Evolución de los entrenamientos

Se realizan entrenamientos para imágenes a color del *dataset* con el filtrado descrito en la sección 3.7.5. El objetivo es realizar un conjunto de entrenamientos que vayan desde imágenes de 8x8 hasta 128x128 píxeles. Por lo tanto se realizarán una serie de 5 entrenamientos.

La figura 47 muestra la evolución de las imágenes producidas comparadas con imágenes reales del *dataset* para los entrenamientos de menor resolución donde la primera fila de cada conjunto de imágenes pertenece a las generadas por las redes y la segunda fila contiene imágenes del *dataset*:



(a) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 8x8.



(b) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 16x16.

(c) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 32x32.

Figura 47: Evolución de los entrenamientos realizados.

Se puede observar que con estas resoluciones tan bajas no se pueden identificar elementos en las imágenes, tanto las originales como las generadas. Sin embargo se observa que la evolución se realiza correctamente ya que a grandes rasgos tanto las agrupaciones de las formas como los colores de las mismas son muy similares a los originales. Es decir, en este punto no se puede decir si las redes pueden tratar imágenes con una gran calidad, pero sí se puede observar que la evolución producida es estable y correcta, pasando de unas resoluciones a otras sin problemas.

Habiendo realizado este pequeño análisis se continua con la serie de entrenamientos que se pueden observar en la figura 48

(a) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 64x64.

(b) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 128x128.

Figura 48: Evolución de los entrenamientos realizados.

La serie de entrenamientos hace ver los buenos resultados de la aplicación de la metodología de entrenamiento ProGAN para el problema. Al comparar las imágenes generadas con las pertenecientes al *dataset* se puede observar claramente como son muy similares, llegando a una gran calidad las imágenes producidas por las redes que, en algún caso, son difícilmente diferenciables de las originales.

Las imágenes generadas por las redes logran con creces imitar a las del *dataset* llegando a imitar vestidos, sombreros, pelo e incluso pequeños detalles como zapatos como se puede observar en el cuarto dibujo de la figura 48b.

La evolución de las imágenes generadas va acorde con lo esperable, consiguiendo cada vez un mayor nivel de detalle que hace ver que la unión de entrenamientos se realiza correctamente. Por otra parte se puede observar cómo las redes son capaces de generar una gran diversidad en las formas de sus dibujos variando entre las posiciones, vestimentas, pelo, etc. de sus personajes. Esto denota que las nuevas capas incluidas logran su objetivo.

En cuanto a la inclusión de imágenes a color en vez de blanco y negro se considera un gran acierto. A través de los colores de las imágenes se pueden analizar más fácilmente e identificar sus detalles no sólo por su forma sino por su color. Esto da una visión más clara, por ejemplo, de la capacidad de las redes de innovar la forma de crear sus personajes.

En este punto la investigación realizada se da completamente por satisfecha, no obstante se considera interesante hacer pruebas para estudiar cómo se comportan las redes ante otro tipo de imágenes.

3.8. Pruebas de la ProGAN para otro tipo de dibujos

Para estudiar en mayor profundidad el rendimiento de la metodología ProGAN se decide buscar otro *dataset* y probar cómo responde a las nuevas imágenes. El objetivo en concreto es estudiar si para un conjunto de imágenes que tengan una distribución más sencilla se mejoran los resultados.

Para ello hay que encontrar un *dataset* que tenga imágenes sencillas y que pueda ser fácilmente exportado a las estructuras ya creadas. Cuando más parecido sea en estructura al *dataset* que ya se ha probado más sencillo será estudiar cómo reaccionan las redes al cambio.

Es interesante además que las imágenes que se traten sean dibujos y, si es posible, del mismo tipo que los del anterior *dataset*, es decir de estilo de cómic japones.

3.8.1. *Nagadomi's moeimouto dataset*

Dentro del *dataset* con el que se cuenta hay dos subconjuntos de imágenes. Uno llamado *Tagged Anime Illustrations*[33] que es el que se ha utilizado para los anteriores resultados y otro llamado *Nagadomi's moeimouto* que consta de caras de personajes.

Este *dataset* es ideal para el propósito actual, pues contiene imágenes similares a las anteriores y además estas tienen una estructura y composición más sencilla al ser sólo de caras. Esto es especialmente interesante, pues supone que la distribución de las imágenes sea más uniforme, lo que implica una mayor sencillez a la hora de imitar.

Por lo tanto se considera idónea la elección de este conjunto de imágenes para repetir el conjunto de entrenamientos y observar cómo responden las redes a dicho cambio.

3.8.2. Composición del *dataset*

En concreto el *dataset* se divide en carpetas, cada una de las carpetas pertenece a un personaje diferente y dentro de ella hay una cantidad de imágenes de dicho personaje. En un principio no se realizará un etiquetado de cada personaje sino que se usarán todas las imágenes sin ninguna distinción.

La figura 49 muestra algunas de las imágenes presentes en el *dataset*:

Figura 49: Imágenes del *dataset*.

[33] Por simplicidad también se denotará a este *dataset* como *dataset original* por ser el utilizado originalmente en la investigación.

3.8.3. Normalizado de imágenes

Las dimensiones de las imágenes en sí son variables, con un máximo de 160 píxeles en una de sus dimensiones. Algunas imágenes tienen una forma cuadrada pero otras tienen sus dos dimensiones desiguales por lo tanto se considera necesario hacer una conversión de las imágenes antes de tratar con ellas.

Para ello hay varias opciones, en el *dataset* original este proceso se había realizando añadiendo las ya mencionadas barras o recuadros negros. Este proceso suponía que apareciesen las bandas negras, lo que significa añadir elementos que pueden distraer a las redes neuronales de la distribución de las imágenes. Esto puede resultar problemático más aún si se considera que el color negro se codifica con valor 256 en todos sus canales; puede suceder que las redes se centren demasiado en replicar estas bandas cuando en realidad son el elemento más accesorio de las imágenes.

Se decide probar a redimensionar las imágenes de tal manera que se distorsionen para ocupar un espacio de 160x160 píxeles cada una. Con ello se consiguen igualar las dimensiones de todas las imágenes a costa de distorsionarlas. La magnitud de la distorsión puede resultar problemática pues, si esta es demasiado grande, puede provocar que las imágenes dejen de tener una forma adecuada para ser replicadas.

Por ello se decide estudiar las imágenes después de la distorsión para ver si estas han empeorado drásticamente su forma. La figura 50 muestra un conjunto de imágenes después de la transformación para que sean cuadradas:

Figura 50: Imágenes del *dataset* normalizadas.

Como se puede observar algunas imágenes han sufrido deformaciones, pero en ningún caso lo suficientemente marcadas como para suponer un problema. Se considera buena la normalización y se decide usar el nuevo conjunto de imágenes para realizar la nueva serie de entrenamientos.

3.8.4. Cambios en las redes

Para que se adapten al nuevo *dataset* hay que hacer unos leves ajustes en la estructura de las redes diseñadas.

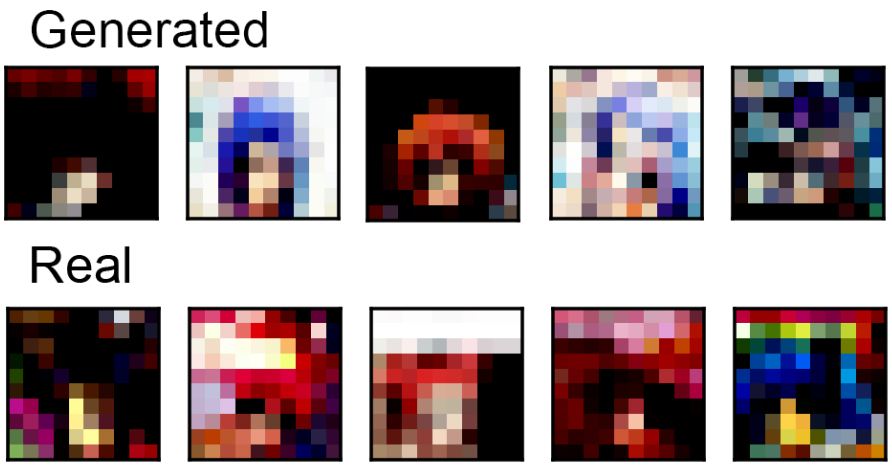
Las imágenes del nuevo *dataset* están codificadas en el rango [0,1] mientras que las del *dataset* original estaban en el rango [0,256]. Esto provoca que el proceso de normalización de sus valores para pasar al rango [-1,1].

Además de esto la sucesión de tamaños para las nuevas redes será 10x10, 20x20, 40x40, 80x80 y 160x160 píxeles correspondientemente con los valores anteriores que eran 8x8, 16x16, 32x32, 64x64 y 128x128 píxeles. El número de filtros de cada capa convolucional es correspondiente una por una entre cada dimensión.

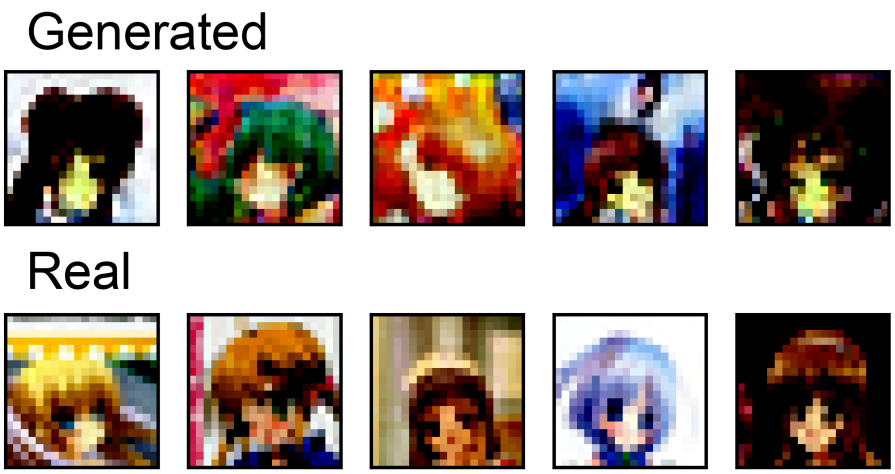
3.8.5. Evolución de los entrenamientos

La evolución de la serie de entrenamientos sigue la misma estructura que en la sección 3.7.9 yendo desde 10x10 hasta 160x160 píxeles.

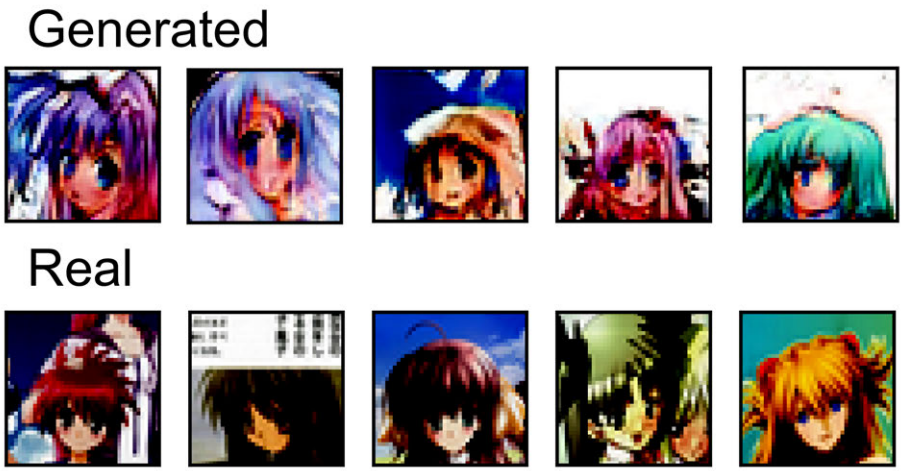
La figura 51 muestra la evolución de las imágenes producidas para los entrenamientos de menor resolución comparadas con imágenes reales del *dataset* usado:



(a) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 10x10.



(b) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 20x20.

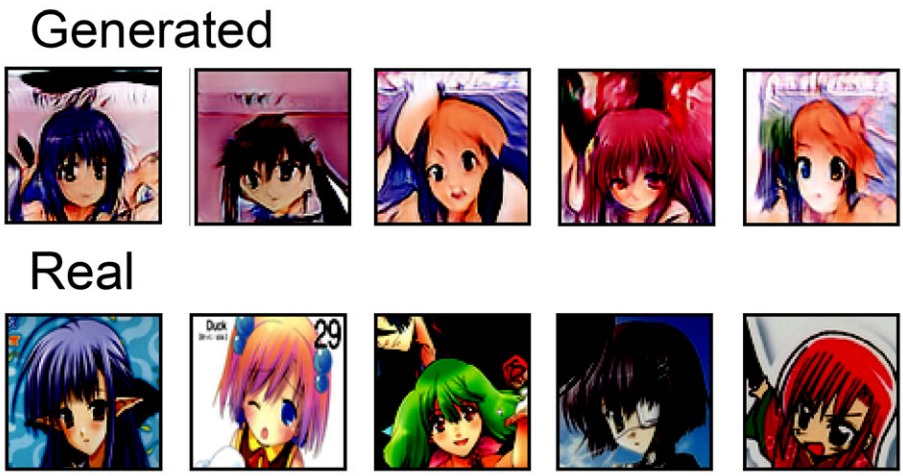


(c) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 40x40.

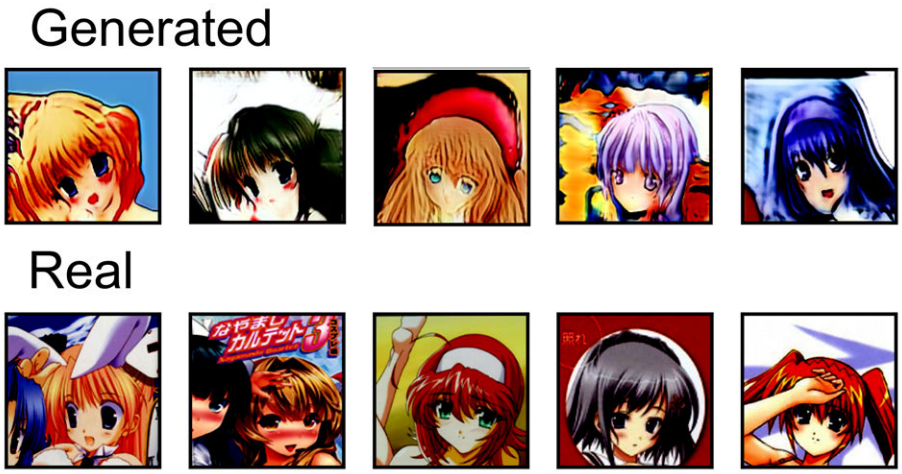
Figura 51: Evolución de los entrenamientos realizados.

Como se puede observar las imágenes producidas pese a tener una resolución baja muestran bastantes detalles, esto se debe a que sus composiciones son más simples que las del anterior *dataset*. Sólo con estos entrenamientos quedan patentes los buenos resultados que se consiguen con las redes creadas, además se puede observar una evolución correcta pasando de un entrenamiento al siguiente sin mayor problema.

Habiendo observado el buen desempeño de las redes se decide continuar con la serie de entrenamientos para observar que clase de imágenes son capaces de crear a mayores resoluciones. La evolución de los entrenamientos se puede observar en la figura 52



(a) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 80x80.



(b) Producciones de la ProGAN e imágenes del *dataset* para una resolución de 160x160.

Figura 52: Evolución de los entrenamientos realizados.

Al observar la evolución de los distintos entrenamientos se puede observar cómo se confirma la premisa propuesta para este experimento: con un *dataset* de imágenes más sencillas se consiguen resultados más legibles. La gran calidad de las imágenes confirma que las redes son completamente funcionales, generando imágenes prácticamente iguales a las del *dataset* del que provienen.

Se puede observar, por ejemplo en el cuarto dibujo de la figura 52a, que las imágenes son muy similares a las del *dataset*. Esta calidad tan alta supone la confirmación del éxito de la investigación presente.

Además cabe destacar la evolución general de los entrenamientos, en los que se puede observar cómo evolucionan las redes al mismo paso que va cambiando la resolución de las imágenes.

En este punto se considera por terminada la sección de desarrollo de la investigación. A partir de este punto sólo resta hacer una valoración crítica de los resultados obtenidos, tras la utilización de todas las metodología propuestas. Es necesario hacer una recopilación de todo lo logrado con el objetivo de analizar si se han logrado los objetivos de la sección 1.3.1 y poner en contraste cada metodología utilizada.

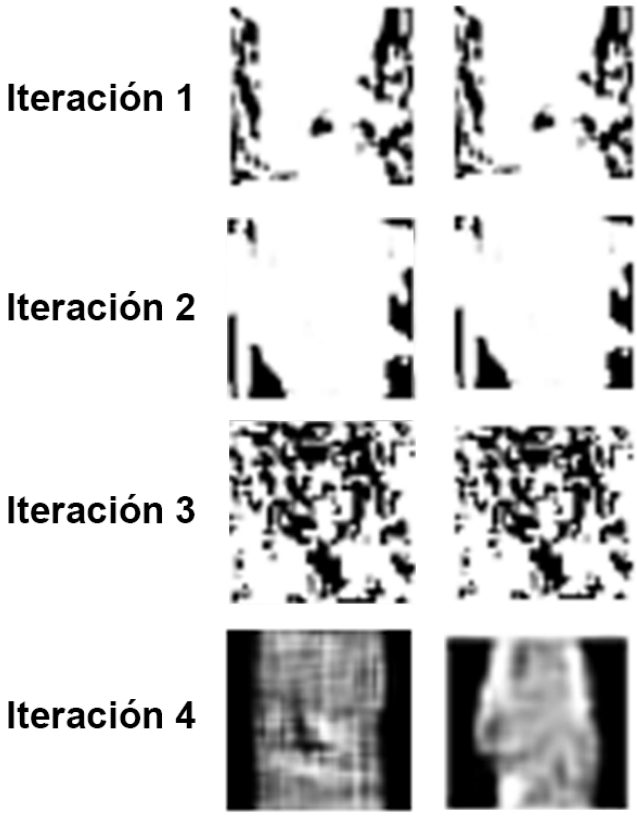
Resultados

4.1. Evolución de producciones entre distintos métodos

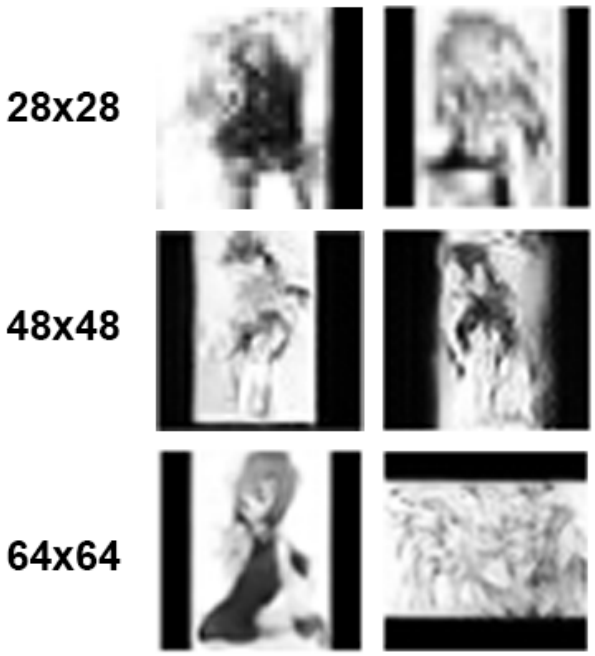
Esta sección se centrará en hacer un repaso de lo que se ha conseguido gracias a la investigación realizada. Se hará balance de todos los resultados y se compararán entre sí.

Lo primero que se considera esencial es hacer una recopilación de qué resultados se conseguían con el uso de cada una de las metodología aplicadas. Esto sirve para tener en un único punto una visión general del rendimiento de las redes para poder compararse directamente unas con otras. Es especialmente interesante agrupar los resultados para ver su evolución de manera conjunta.

La figura 53 muestra los resultados de las producciones de las secciones 3.5, 3.6, con modelos previos a la aplicación de la metodología ProGAN para imágenes en blanco y negro:



(a) Producciones de la sección 3.5.

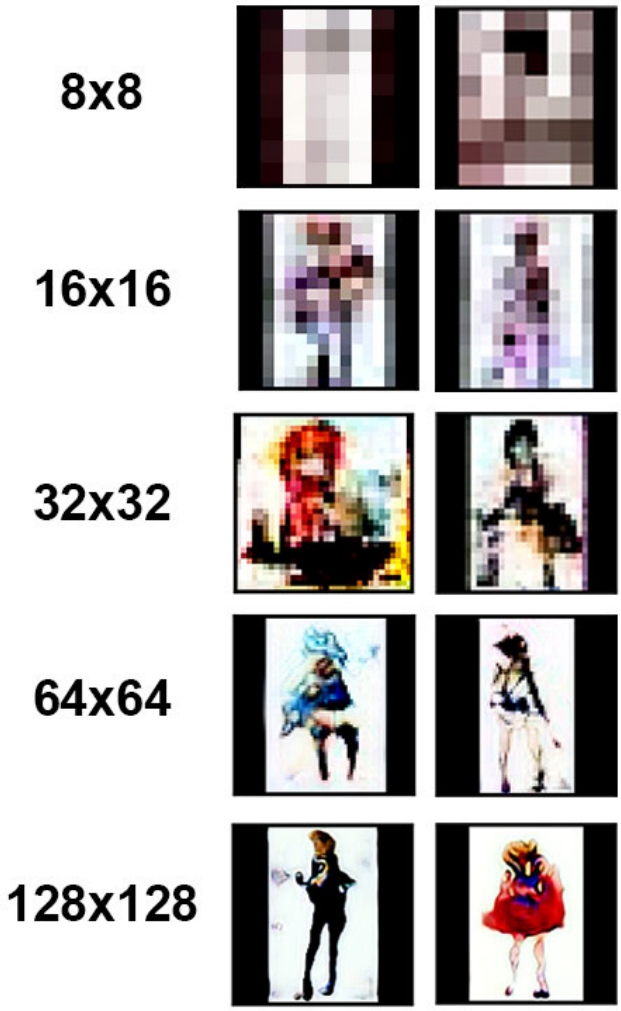


(b) Producciones de la sección 3.6.

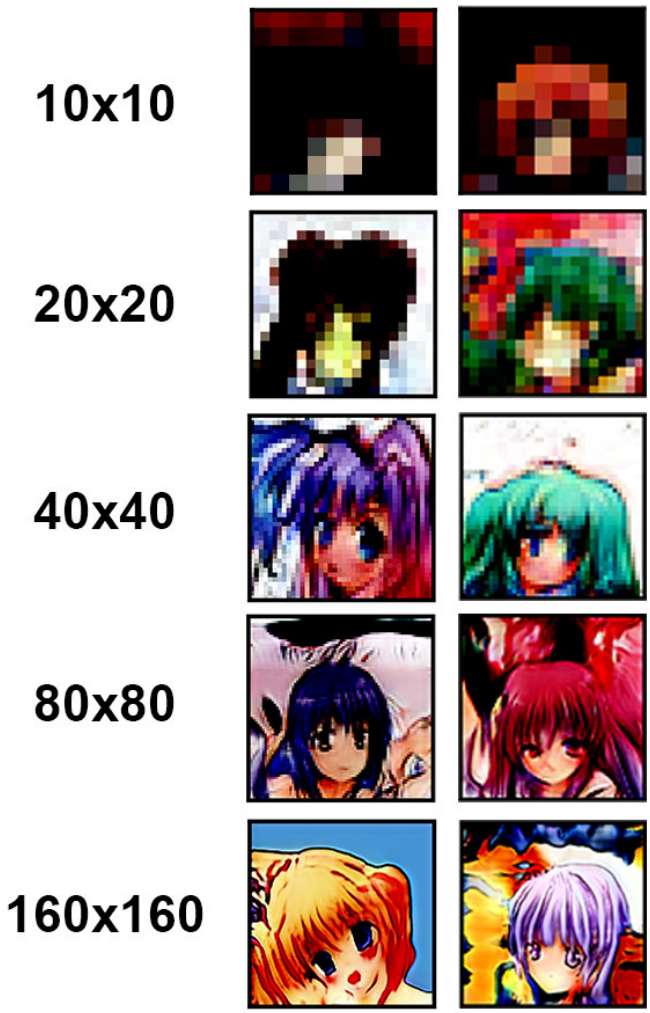
Figura 53: Evolución de los entrenamientos realizados con el uso de las distintas metodologías.

Como se puede observar durante este periodo se produjeron avances significativos, sin embargo se pudo observar que las redes carecían de mecanismos suficientemente potentes como para escalar adecuadamente los resultados. Por ello se decidió buscar otro tipo de metodología, la ProGAN, capaz de mejorar los resultados hasta el momento. Al mismo tiempo se decidió pasar a generar imágenes a color.

Las producciones resultado de la aplicación de la metodología ProGAN para dos *datasets* distintos se puede observar en la figura 54



(a) Producciones de la sección 3.7 para imágenes de cuerpo entero de personajes.



(b) Producciones de la sección 3.8 para imágenes de caras de personajes.

Figura 54: Evolución de los entrenamientos realizados con el uso de la metodología ProGAN.

En las figuras se puede observar cómo ha ido evolucionando la investigación para pasar de imágenes de poca calidad a ir cada vez definiendo mejores modelos hasta lograr una calidad muy alta de imitación. Además se observa que el paso de imágenes en blanco y negro a color fue una idea acertada logrando una mejoría en la claridad de las imágenes haciendo que sea más sencilla la labor de identificar los dibujos generados tal y como se había hipotetizado en la sección 3.7.6.

Esta imagen general es la que muestra la labor investigadora realizada y en ella se puede ver cómo han surtido efectos los distintos cambios propuestos para ir mejorando, poco a poco, pero de manera clara, concisa y eficiente las producciones de las redes.

Cuando se proponía en un principio seguir una metodología propia de la investigación precisamente se hacía alusión a esto, a que para conseguir unos buenos resultados no se puede empezar la casa por el tejado e intentar implementar la metodología más compleja. Este mecanismo puede funcionar pero no denota unas buenas prácticas, pues el objetivo es ir mejorando los modelos para ir escalando los resultados siendo en todo momento consciente de qué esta fallando.

Dicho esto los resultados de la investigación se observan sobre todo en la evolución y no tanto en los resultados *per se*, lo interesante es cómo se ha podido pasar de unas imágenes con formas prácticamente aleatorias a resultados consistentes y de alta calidad. Es esta evolución lo que muestra que la investigación ha sido un éxito pero que también ha sido llevada con cierta profesionalidad propia de una investigación de este tipo.

4.2. Recapitulación de la evolución de la investigación

Se considera necesario hacer una valoración crítica de los aspectos diferenciales de cada una de las metodologías de las que se han hecho uso. Ahora que se han concluido todos los entrenamientos es necesario comparar en qué se diferencia el uso de unas metodologías u otras y qué evolución ha supuesto la implementación de cada una de ellas.

Para ello hay que hacer un repaso de cada una de las metodologías ahora que ya se tienen resultados de la investigación completa. La figura 55 muestra un esquema de cómo a ido evolucionando y ramificándose a grandes rasgos la investigación:

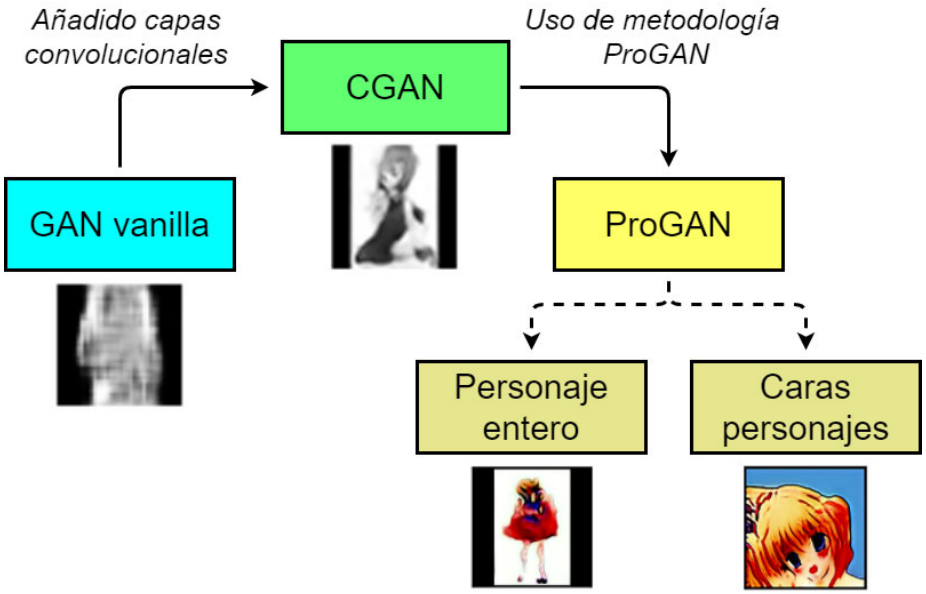


Figura 55: Esquema que muestra los pasos que ha seguido la investigación y los resultados obtenidos con cada uno de ellos.

Ahora se irán desglosando a modo de resumen los distintos pasos y motivos con los que seguir la investigación, de esta manera se tendrá una visión más clara de los distintos motivos que han llevado a los resultados finales.

4.2.1. GAN *vanilla*

Las redes que se diseñaron en este punto suponían la primera toma de contacto con las redes GAN. Esto significa que no había ningún conocimiento previo de cómo responderían las redes a los distintos parámetros que se definían. Pese a seguir arquitecturas presentes en bibliografía la aplicación de estas al *dataset* de dibujos tenía resultados inesperados.

Por todo eso se siguió una progresión en las distintas redes para ir mejorando su funcionamiento. Por ejemplo al principio las primeras redes tenían una inestabilidad exagerada, es por este motivo que se plantearon la introducción de ciertos mecanismos para evitar el sobreentreno de la red discriminadora.

Durante los primeros entrenamientos que se realizaron con esta metodología los valores de las pérdidas de las redes alcanzaban dimensiones muy altas y cambiantes, esto fue lo que llevó a plantear los cambios ya mencionados.

Esta metodología sirvió por lo tanto para asentar unas bases con la mayor estabilidad posible. Esto se manifestó en las producciones cada vez de mayor calidad generadas por las redes. En cierto punto las producciones dejaban ver que los problemas surgidos habían sido solucionados, porque lo que antes parecían manchas sin ningún orden ahora tenían una forma que se asemejaba a las imágenes del *dataset*.

En este punto se consideró que, como se habían estabilizado los entrenamientos y las producciones comenzaban a tener sentido, se podía seguir con la evolución migrando a una metodología un poco más compleja que consiga mejores resultados.

4.2.2. GAN convolucional o CGAN

Antes de aplicar una metodología muy compleja se consideró que era un buen punto intermedio aplicar redes convolucionales tanto para la red discriminadora como para la generadora. Los entrenamientos realizados con la GAN *vanilla* pusieron de manifiesto que si las dos redes no mantenían una estructura similar se producía una inestabilidad que impedía a las redes funcionar adecuadamente. Con este paso se pretendía que las dos redes tuviesen una forma similar y que por lo tanto no se distanciasen mucho los entrenamientos.

Por eso se consideró que el primer paso era aplicar a ambas redes la misma estructura, en este caso además siendo convolucional para intentar especializar a las redes en la tarea.

Pese a que las redes tengan una estructura similar su propósito no es el mismo, pues la tarea de la red discriminadora es mucho más sencilla que la de la generadora. Para intentar equilibrar las redes se aplicaron ciertas técnicas como la aplicación de las capas de *dropout*.

Con esto se vio que las redes entrenaban correctamente pero aún sufrían de algún periodo de alta inestabilidad que de manera empírica mostraba que empeoraba las producciones generadas. Para solucionar esto se intentó simplificar el funcionamiento de los entrenamientos para tener un mayor control sobre ellos. Por ello se decidió eliminar la diferencia de *batches* de entrenamiento entre redes o, entre otros, el mecanismo por el que se hacían pasar imágenes generadas como reales.

El propósito de esto era simplificar la solución que se tenía y con ello poder apreciar mejor cómo funcionaban las redes. Además los problemas de alta inestabilidad y sobreentreno ya se habían solucionado previamente así que prescindir de estos mecanismos no era una mala idea ahora que ya habían cumplido su cometido. Si tras esto las redes volviesen a la alta inestabilidad se podrían volver a aplicar las medidas sin ningún tipo de esfuerzo.

Además se consideró que si se reducía aún más el tamaño de las imágenes con las que se trataba se podría simplificar el problema más y analizar más rápido y con una mayor probabilidad de acierto si las redes rendían correctamente.

Una vez obtenidos los resultados para imágenes de 28x28 píxeles se vio interesante estudiar cómo se formaban imágenes de mayor tamaño con la misma estructura general. Por ello se obtuvieron imágenes de 48x48 y 64x64 píxeles con unos buenos resultados.

La calidad de las producciones parecía en este punto verse limitada por la metodología utilizada y por ello se consideró pasar a usar una nueva.

4.2.3. *Progressive growing of* GANS o ProGAN

Antes de escoger esta metodología en concreto se realizó un estudio en profundidad de qué arquitecturas eran las utilizadas en la actualidad para resolver problemas del mismo tipo. Para ello se hizo una investigación del estado del arte de las redes GAN, sobre todo aquellas derivadas de la DCGAN[19].

De todas las redes estudiadas aquella propuesta por la ProGAN era la que más se ajustaba a lo que se buscaba, teniendo una estructura que se mantenía relativamente simple pero introduciendo complejos cambios a la manera que se entrenaba llegando a unos resultados mejores.

A parte de la implementación de la metodología ProGAN se pasó a usar imágenes a color para poder estudiar mejor las producciones que se fuesen realizando.

Durante los primeros entrenamientos realizados se pudo observar una de las grandes ventajas de las ProGAN, al comenzar con imágenes de muy baja resolución las redes son capaces de imitar las imágenes del *dataset* que se use. Esto hace que los primeros resultados que se obtengan sean muy buenos en cuanto a semejanza con el *dataset* y sirve para corroborar el buen funcionamiento de la red. Al ir aumentando la resolución se puede observar si en algún punto se produce una anomalía ya que se tiene constancia que los resultados anteriores han sido buenos.

Otra ventaja es que al estar los entrenamientos divididos se pueden repetir con mayor sencillez. A lo largo de la investigación han habido muchos puntos en los que un entrenamiento no era correcto por diversos motivos. Ante esto las ProGAN permitían cargar los resultados del entrenamiento inmediatamente anterior y volver a lanzar el nuevo con el problema solucionado. Esto produce una mayor eficiencia evitando la pérdida de tiempo en repeticiones de entrenamientos que ya funcionaban bien.

Con todo esto se hicieron una gran cantidad de pruebas para distintas funciones de pérdida, hiperparámetros distintos, estructuras diferentes, etc. y se llegó a la solución final expuesta en la sección 3.7. Tras haber hecho toda la labor de tuneo de las redes se consiguieron unos resultados excelentes que ponían fin a una investigación en la que se había logrado el objetivo siguiendo una investigación precisa, clara y consistente.

Como punto extra se decidió hacer pruebas para un *dataset* más complejo que corroborase los buenos resultados. La incorporación de imágenes de cara suponía el culmen de la investigación de la mejor manera posible, corroborando que se había producido una evolución en los resultados excelente y trazable.

Impacto social y medioambiental

5.1. Impacto social

Desde el punto de vista social hay que tener en cuenta lo íntimamente relacionada que está la visión de la mente humana con la inteligencia artificial. Típicamente se consideraba que los ordenadores podrían automatizar ciertas tareas, pero que en ningún momento podrían hacer tareas relacionadas con la creatividad, sentimientos y otros aspectos considerados únicos de la especie humana. Sin embargo los avances producidos en el campo de la inteligencia artificial durante los últimos años hacen que cambie la visión que se tiene del impacto que pueda tener la tecnología en nuestra forma de vida.

A mediados del siglo XX surgió la inteligencia artificial tal y como se conoce hoy en día, pese a ser un concepto mucho más antiguo, y durante sus primeros años la sociedad comenzó a tener en cuenta las posibilidades de los ordenadores en un futuro. Esta primera época de la inteligencia artificial se conoce como *Verano de la inteligencia artificial* y se caracteriza por un positivismo tecnológico por el cual las personas creían que las posibilidades de la informática podrían hacer que los ordenadores sustituyesen a las personas en un futuro no muy lejano.

Durante las décadas de los 70, 80 y principios de los 90 se produjo la época conocida como *Invierno de la inteligencia artificial* en la que, debido a las limitaciones de la época y el fracaso de proyectos muy importantes, se consideró que las creencias anteriores eran demasiado positivas y que la inteligencia artificial jamás podría llegar a los objetivos que se pensaba.

Durante los últimos años, sobre todo a partir de 2010, gracias a los grandes avances en el campo, ha surgido un repunte en el interés en proyectos relacionados con inteligencia artificial. Desde entonces se vuelve a considerar que la inteligencia artificial tiene grandes posibilidades y se producen grandes avances en la materia.

Estos periodos ponen de manifiesto que cuando más avances se producen es cuando la sociedad se interesa más por la materia y cuando más preocupan las consecuencias de intentar imitar la manera de pensar de los seres humanos. La investigación presente puede tener como impactos en la sociedad aquellos derivados de poner de manifiesto lo que se puede lograr con redes generativas.

En la presente investigación se muestra cómo se pueden replicar aspectos que hasta ahora se pensaban puramente humanos como la creatividad. Esto hace que los resultados de la investigación despierten en cierto sector de la población recelo a la hora de confiar en la tecnología. Cuando la sociedad observa un avance en lo que hacen los ordenadores se produce un cambio en su cosmovisión[34], lo que puede producir movimientos sociales en contra de la tecnología. En los últimos años se han podido observar cómo ciertos sectores sociales desconfían de avances como el 5G o creen que hay chips de control mental en las vacunas.

[34] La cosmovisión es la imágen que tiene una persona, sociedad o cultura de cómo se articula el mundo a su alrededor. Suele estar formada por valoraciones y percepciones de su entorno.

Pese a parecer que esto esta muy desconectado del presente TFG se considera que cualquier avance en una rama del conocimiento tan delicada como la que intenta replicar la inteligencia humana es susceptible de producir grandes cambios en la sociedad.

5.2. Impacto ambiental

El presente proyecto no supone ninguna transformación directa en el ecosistema y medioambiente, sin embargo los medios de los que hace uso han de ser tomados en cuenta.

Los medios más inmediatos de los que se hace uso son el equipo de trabajo necesario para realizar cualquier tarea relacionada con la informática, hay que tener en cuenta que para realizar uso de un equipo de trabajo este ha de ser construido con materiales, que en muchos casos son difícilmente reciclables y en otros su extracción es muy costosa medioambientalmente.

El caso más popular es el de la roca llamada coltán que se usa para la construcción de dispositivos tecnológicos. Su extracción provoca la deforestación de reservas naturales y ecosistemas. A parte de esto el 80 % del coltán proviene de la República Democrática del Congo, un país que cuenta con una guerra civil íntimamente relacionada con la presencia del coltán en su territorio. El gran valor de la roca hace que las luchas por su adquisición promuevan la guerra civil así como casos de esclavitud infantil en minas ilegales por todo su territorio.

Además de los materiales necesarios para trabajar en el proyecto hay que destacar el consumo energético, especialmente intenso en la inteligencia artificial por el alto uso de tarjetas gráficas. Para que funcionen las tarjetas gráficas estas han de ser alimentadas con energía que ha de ser extraída. En España la energía renovable supone un 43,6 % del total[20].

Cabe destacar el uso de *Google Colaboratory*, para que esta plataforma funcione existen grandes centros de cálculo con una gran cantidad de servidores que son los que realizan las operaciones de forma remota. La existencia de este tipo de instalaciones provoca durante los últimos años preocupaciones por su alto consumo energético. Por ejemplo durante los últimos años están surgiendo críticas a servidores dedicados a hacer gran cantidad de cálculos relacionados con *blockchain*.

Al mismo tiempo se están observando que este tipo de nuevas tecnologías impulsan el desarrollo y despliegue de energía renovables para usar su energía por ser esta más barata a la larga.

Por lo tanto se considera que el presente TFG no tiene ningún tipo de particularidad en cuanto a sus impactos, sin embargo se encuentra en un sector en el que existen ciertas preocupaciones que no se pueden quedar en olvido.

Líneas de investigación

6.1. Ramificaciones del proyecto

Pese haber finalizado la investigación actual y suponiendo esta una investigación completa y que ha logrado los objetivos propuestos surgen ciertos aspectos donde seguir trabajando. Estas líneas de investigación suponen aspectos paralelos del proyecto realizado.

6.2. Profundización el la metodología ProGAN

La línea de investigación más directa que surge es seguir utilizando la metodología ProGAN para tratar el problema. Por una parte se podrían proponer mejoras y estudiar cambios en su estructura, optimizando las redes y los distintos hiperparámetros del entrenamiento. Con eso se intentaría observar si son viables cambios en las redes y cómo estos pueden afectar al desempeño de las redes.

Por otra parte otra manera de ahondar en el comportamiento de las redes es probar distintos tipos de imágenes en los *datasets* que se usen para entrenamiento. Con ello se podría observar cómo reaccionan los modelos creados ante cambios en la composición de sus datos de entrada. No hay que olvidar que las redes generativas en general y las ProGAN en concreto han sido diseñadas para tratamiento de imágenes reales, por lo tanto sería interesante probar cómo tratan dicho tipo de imágenes.

Además de esto se podría probar a realizar entrenamientos más largos, por una parte haciendo más *batches* por cada entrenamiento pero al mismo tiempo tratando imágenes de mayor tamaño. Esto podría generar nuevos problemas pero es una línea de investigación totalmente viable.

6.3. Búsqueda de nuevas arquitecturas

Esta es la rama de las investigaciones que resulta más natural y la que se considera que puede obtener mejores resultados. Para escalar el problema hay que probar nuevos tipos de redes GAN para observar si se pueden obtener mejores resultados con el uso de estos.

Se considera que se podrían generar imágenes aún de mayor calidad, si se consigue una arquitectura adecuada se podrían generar imágenes de una gran calidad. Por ejemplo una de las arquitecturas que mejores resultados ha obtenido en los últimos años es la StyleGAN[21], esta supone una modificación en las redes que podría aportar mejores resultados a través de la inclusión de un espacio latente intermedio y ruido para todas las capas. Esta arquitectura además sigue utilizando el esquema de crecimiento de la ProGAN y supone su evolución, pues fue propuesta por los mismos científicos de NVIDIA *labs*.

Otra evolución de los modelos del presente trabajo es conseguir que las redes generasen imágenes de cómic condicionadas por ciertos datos de entrada. En concreto existen redes como la Pix2Pix[22] encargadas de hacer una traducción de imagen a imagen. Con este tipo de arquitecturas se podrían diseñar modelos capaces de, recibiendo como entrada un esquema del dibujo que se quiere, crear una ilustración completamente nueva con la forma indicada como entrada.

A parte de los dos tipos de arquitectura actualmente existen grandes avances en el campo del *Deep Learning* los cuales pueden ser aplicables a la investigación presente.

6.4. Generador de páginas enteras

Uno de los pasos a los que se dirige la investigación es a la creación de cómics enteros gracias al uso de inteligencia artificial. Esto consistiría en que, a través del uso de redes neuronales, se puedan crear páginas compuestas por viñetas que guarden una coherencia entre sí y cuenten una historia.

Esta inteligencia artificial sería capaz de ilustrar un cómic pero al mismo tiempo decidir un guión, una composición de páginas y un estilo propio. Los modelos encargados de realizar esta tarea serían evoluciones de los actuales y por lo tanto suponen una de las líneas de futuras investigaciones más interesantes.

Conclusiones

7.1. Conclusiones del proyecto

El desarrollo de la investigación ha sido excepcionalmente interesante, la metodología que se ha seguido ha producido resultados mejores de los que en un principio se esperaban y, al mismo tiempo, se ha ido avanzando aprendiendo por el camino.

El area del deep learning ha resultado especialmente motivadora y gracias al trabajo realizado en el desarrollo del TFG, se han sentado las bases para empezar una línea futura de investigación. En cuanto a mi desarrollo personal, este trabajo ha despertado mi interés en seguir orientado mi carrera profesional en torno a la investigación en este área.

Durante la investigación se han aprendido tantas cosas que resulta apasionante seguir con la misma para poder desarrollar los modelos aún más y seguir investigando en el campo del *Deep Learning*. Esto es muy interesante pues se considera que este campo de estudio ha sufrido unos grandes avances durante los últimos años llegando a ser una de las disciplinas científicas más importantes del presente. Esto supone un campo donde poder avanzar y encontrar financiación y posibilidades de futuros proyectos en los que poner en práctica todo lo aprendido durante el desarrollo de la investigación.

Por una parte al tener que investigar las distintas arquitecturas del presente se han ido conociendo como se van sucediendo los avances científicos, se han leído una gran cantidad de *papers* y se ha aprendido cómo aplicar los conocimientos del estado del arte para la presente investigación. Esto me forma personalmente como un profesional con futuro y con unas bases bien asentadas, capaz de estudiar las investigaciones ajenas así como tener la capacidad crítica de analizar y reproducir arquitecturas de vanguardia. Además la investigación supone una carta de presentación ideal para poder enseñar al resto de la comunidad lo que se es capaz de hacer.

Por lo tanto considero la investigación actual como una entrada idónea en el mundo investigador, en el cual considero que quiero desarrollar mi carrera profesional. Más aún después de haber realizado mi propia investigación a la cual veo un futuro brillante por el que quiero apostar y seguir investigando.

Apéndices

8.1. Apéndice A: GAN Vanilla

Listing 1: Creación del modelo discriminador

```
model = Sequential()

model.add(Flatten(input_shape=self.img_shape))
model.add(Dense(self.img_shape[0]*self.img_shape[1]))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(4098))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(2046))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(1024))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(512))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(256))
model.add(LeakyReLU(alpha=0.2))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()

img = Input(shape=self.img_shape)
validity = model(img)

return Model(img, validity)
```

Listing 2: Creación del modelo generador

```
model = Sequential()

model.add(Dense(256 * 6 * 6, activation="relu", input_dim
    =self.latent_dim))
model.add(BatchNormalization())
model.add(Reshape((6, 6, 256)))
model.add(UpSampling2D())
#Output shape: (6, 6, 256)

model.add(Conv2D(128, kernel_size=(3, 3), activation='
    relu', padding='same'))
model.add(BatchNormalization())
model.add(UpSampling2D())
#Output shape: (12, 12, 128)

model.add(Conv2D(64, kernel_size=(5, 5), activation='relu
    ', padding='same'))
model.add(BatchNormalization())
model.add(UpSampling2D())
#Output shape: (24, 24, 64)

model.add(Conv2D(self.channels, kernel_size=(5, 5),
    activation='tanh', padding='same'))
#Output shape: (48, 48, 1)

model.summary()

noise = Input(shape=(self.latent_dim,))
img = model(noise)

return Model(noise, img)
```

8.2. Apéndice B: CGAN

Listing 3: Creación del modelo discriminador

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(32, (5,5), strides=(2,2), padding
    ='same', input_shape=[self.img_rows, self.img_cols, self
    .channels]))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(64,(5,5), strides=(2,2), padding
    ='same'))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(128,(5,5), strides=(2,2), padding
    ='same'))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.5))

model.add(layers.Conv2D(128,(5,5), strides=(2,2), padding
    ='same'))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.5))

model.add(layers.Flatten())
model.add(layers.Dense(1))

return model
```

Listing 4: Creación del modelo generador

```
model = tf.keras.Sequential()
model.add(layers.Dense(8*8*128, use_bias=False,
    input_shape=(self.latent_dim,)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Reshape((8, 8, 128)))
assert model.output_shape == (None, 8, 8, 128) #None
    refers to the batch size

# Conv2DTranspose are used to upsample for generating an
    image from a seed which is random noise
model.add(layers.Conv2DTranspose(128, (5,5), strides
    =(2,2), padding='same', use_bias=False))
assert model.output_shape == (None, 16, 16, 128)
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Conv2DTranspose(128, (5,5), strides
    =(2,2), padding='same', use_bias=False))
assert model.output_shape == (None, 32, 32, 128)
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Conv2DTranspose(64, (5,5), strides=(2,2)
    , padding='same', use_bias=False))
assert model.output_shape == (None, 64, 64, 64)
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))

model.add(layers.Conv2DTranspose(1, (5,5), strides=(1,1),
     padding='same', use_bias=False, activation='tanh'))
assert model.output_shape == (None, self.img_rows, self.
    img_cols, self.channels)

return model
```

8.3. Apéndice C: ProGAN

Listing 5: Creación del modelo discriminador para imágenes de 10x10

```
in_image = layers.Input(shape=self.img_shape)

x = layers.Conv2D(256, (1,1), strides=(1,1), padding='
    same', kernel_initializer=self.init, kernel_constraint
    =self.const)(in_image)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D8x8_1', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky8x8_1')(x)
x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D8x8_2', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky8x8_2')(x)
x = layers.AveragePooling2D()(x)

x = MinibatchStdev()(x)
x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D4x4_1', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky4x4_1')(x)
x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D4x4_2', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky4x4_2')(x)

x = layers.Flatten()(x)
out_class = layers.Dense(1, activation='sigmoid')(x)

return keras.Model(in_image, out_class)
```

Listing 6: Creación del modelo generador para imágenes de 10x10

```
in_latent = layers.Input(shape=(self.latent_dim,))

x = PixelNormalization(name='pixelnormDense_1')(in_latent
    )
x = layers.Dense(5*5*256, kernel_initializer=self.init,
    kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leakydense_1')(x)
x = PixelNormalization(name='pixelnormDense_2')(x)
x = layers.Reshape((5, 5, 256))(x)

x = layers.Conv2DTranspose(256, (3,3), strides=(1,1),
    padding='same', name='conv2d4x4_1', kernel_initializer
    =self.init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky4x4_1')(x)
x = PixelNormalization(name='pixelnorm4x4_1')(x)

x = layers.Conv2DTranspose(256, (3,3), strides=(1,1),
    padding='same', name='conv2d4x4_2', kernel_initializer
    =self.init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky4x4_2')(x)
x = PixelNormalization(name='pixelnorm4x4_2')(x)

x = layers.UpSampling2D(interpolation='nearest')(x)
x = layers.Conv2DTranspose(256, (3,3), strides=(1,1),
    padding='same', name='conv2d8x8_1', kernel_initializer
    =self.init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky8x8_1')(x)
x = PixelNormalization(name='pixelnorm8x8_1')(x)

x = layers.Conv2DTranspose(256, (3,3), strides=(1,1),
    padding='same', name='conv2d8x8_2', kernel_initializer
    =self.init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky8x8_2')(x)
x = PixelNormalization(name='pixelnorm8x8_2')(x)

out_image = layers.Conv2DTranspose(self.channels,
    activation='tanh', kernel_size=(1,1), strides=(1,1),
    padding='same', kernel_initializer=self.init,
    kernel_constraint=self.const)(x)

return keras.Model(in_latent, out_image)
```

Listing 7: Creación del modelo discriminador para imágenes de 20x20

```
discriminator_pretrained = tf.keras.models.load_model('/
    content/drive/My Drive/Colab Notebooks/GAN/SavedModel/
    discriminator.h5', custom_objects={'MinibatchStdev':
    MinibatchStdev})
in_image = layers.Input(shape=self.img_shape)

x = layers.Conv2D(128, (1,1), strides=(1,1), padding='
    same', kernel_initializer=self.init, kernel_constraint
    =self.const)(in_image)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D16x16_1', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky16x16_1')(x)

x = layers.Conv2D(256, (3,3), strides=(1,1), padding='
    same', name='conv2D16x16_2', kernel_initializer=self.
    init, kernel_constraint=self.const)(x)
x = layers.LeakyReLU(alpha=0.2, name='leaky16x16_2')(x)
x = layers.AveragePooling2D(name='avgpooling16x16')(x)

for i in range(3,14):
layer = discriminator_pretrained.get_layer(index=i)
x = layer(x)

layer = discriminator_pretrained.get_layer(index=14)
out_class = layer(x)

return tf.keras.Model(in_image, out_class)
```

Listing 8: Creación del modelo generador para imágenes de 20x20

```
generator_pretrained = tf.keras.models.load_model('/
    content/drive/My Drive/Colab Notebooks/GAN/SavedModel/
    generator.h5', custom_objects={'PixelNormalization':
    PixelNormalization})
in_latent = layers.Input(shape=(self.latent_dim,))

layer = generator_pretrained.get_layer(index=1)
x = layer(in_latent)

for i in range(2,19):
layer = generator_pretrained.get_layer(index=i)
x = layer(x)

new_layers = layers.UpSampling2D(interpolation='nearest',
     name='upsampling_16x16_0')(x)
new_layers = layers.Conv2DTranspose(256, (3,3), strides
    =(1,1), padding='same', name='conv2d16x16_1',
    kernel_initializer=self.init, kernel_constraint=self.
    const)(new_layers)
new_layers = layers.LeakyReLU(alpha=0.2, name='
    leaky16x16_1')(new_layers)
new_layers = PixelNormalization(name='pixelnorm16x16_1')(
    new_layers)

new_layers = layers.Conv2DTranspose(256, (3,3), strides
    =(1,1), padding='same', name='conv2d16x16_2',
    kernel_initializer=self.init, kernel_constraint=self.
    const)(new_layers)
new_layers = layers.LeakyReLU(alpha=0.2, name='
    leaky16x16_2')(new_layers)
new_layers = PixelNormalization(name='pixelnorm16x16_2')(
    new_layers)
out_image_hres = layers.Conv2DTranspose(self.channels,
    (1,1), strides=(1,1), padding='same', name='
    conv2d16x16_3', kernel_initializer=self.init,
    kernel_constraint=self.const)(new_layers)

#bypass_layer = layers.Conv2DTranspose(self.channels,
    (1,1), strides=(1,1), padding='same', name='
    conv2d16x16_bypass', kernel_initializer=self.init,
    kernel_constraint=self.const)(x)
#out_image_lres = layers.UpSampling2D(interpolation='
    nearest', name='upsampling_16x16')(bypass_layer)
```

```
layer = generator_pretrained.get_layer(index=19)
x = layer(x)
out_image_lres = layers.UpSampling2D(interpolation='
    nearest', name='upsampling_16x16')(x)

return tf.keras.Model(in_latent, [out_image_lres,
    out_image_hres])
```

Listing 9: Train step para realizar el fade in

```
def train_step(self, images):
noise = tf.random.normal([self.batch_size, self.
    latent_dim])

with tf.GradientTape() as gen_tape, tf.GradientTape() as
    disc_tape:
# Get outputs of the generator
image_lowres, image_highres = self.generator(noise,
    training=True)
# Do the fade in of both images
image_lowres = image_lowres * (1-self.alpha)
image_highres = image_highres * self.alpha
generated_img = image_lowres + image_highres
if self.alpha < 1:
self.alpha += self.alpha_step

true_output = self.discriminator(images, training=True)
fake_output = self.discriminator(generated_img, training=
    True)

gen_loss = self.generator_loss(fake_output)
disc_loss = self.discriminator_loss(true_output,
    fake_output)

grad_generator = gen_tape.gradient(gen_loss, self.
    generator.trainable_variables)
grad_discriminator = disc_tape.gradient(disc_loss, self.
    discriminator.trainable_variables)

self.gen_optimizer.apply_gradients(zip(grad_generator,
    self.generator.trainable_variables))
self.dis_optimizer.apply_gradients(zip(grad_discriminator
    , self.discriminator.trainable_variables))

return gen_loss, disc_loss
```

Referencias

[1] KT Schütt y col. «Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions». En: *Nature communications* 10.1 (2019), págs. 1-10.

[2] Jianyi Yang y col. «Improved protein structure prediction using predicted inter-residue orientations». En: *bioRxiv* (2019). DOI: 10.1101/846279. eprint: https://www.biorxiv.org/content/early/2019/11/18/846279.full.pdf. URL: https://www.biorxiv.org/content/early/2019/11/18/846279.

[3] Jose R SAURA, Ana Reyes-Menéndez y Pedro PALOS-SANCHEZ. «Un Análisis de Sentimiento en Twitter con Machine Learning: Identificando el sentimiento sobre las ofertas de BlackFriday». En: *Revista Espacios* 39.42 (2018).

[4] Luciano Floridi y Massimo Chiriatti. «GPT-3: Its nature, scope, limits, and consequences». En: *Minds and Machines* 30.4 (2020), págs. 681-694.

[5] Alvaro Sanchez-Gonzalez y col. *Learning to Simulate Complex Physics with Graph Networks*. 2020. arXiv: 2002.09405 [cs.LG].

[6] Ian Goodfellow y col. «Generative adversarial nets». En: *Advances in neural information processing systems*. 2014, págs. 2672-2680.

[7] Guillermo Iglesias Hernández. «Procesamiento automático de ilustraciones: Clasificación multi-etiqueta de cómics con Deep Learning». Madrid, jun. de 2020. URL: http://oa.upm.es/64638/.

[8] David H Hubel y Torsten N Wiesel. «Receptive fields of single neurones in the cat's striate cortex». En: *The Journal of physiology* 148.3 (1959), págs. 574-591.

[9] Honglak Lee y col. «Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks». En: *Commun. ACM* 54 (oct. de 2011), págs. 95-103. DOI: 10.1145/2001269.2001295.

[10] *Danbooru2020: A Large-Scale Crowdsourced and Tagged Anime Illustration Dataset*. 2021. URL: https://www.gwern.net/Danbooru2020.

[11] *TensorFlow with GPU*. 2021. URL: https://colab.research.google.com/notebooks/gpu.ipynb (visitado 2021).

[12] Guillermo Iglesias Hernández. *Dataset*. 2020. URL: https://github.com/guillermoiglesiashernandez/Dataset.

[13] Tero Karras y col. «Progressive growing of gans for improved quality, stability, and variation». En: *arXiv preprint arXiv:1710.10196* (2017).

[14] Erik Linder-Norén. *Keras-GAN*. 2021. URL: https://github.com/eriklindernoren/Keras-GAN.

[15] Ş. Öztürk y col. «Convolution Kernel Size Effect on Convolutional Neural Network in Histopathological Image Processing Applications». En: *2018 International Symposium on Fundamentals of Electrical Engineering (ISFEE)*. 2018, págs. 1-5. DOI: 10.1109/ISFEE.2018.8742484.

[16] Anish Kumar Posim Reddy. *Fashion-MNIST-GAN-Keras*. 2019. URL: https://github.com/anishreddy3/Fashion-MNIST-GAN-Keras.

[17] Jost Tobias Springenberg y col. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].

[18] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. «Imagenet classification with deep convolutional neural networks». En: *Advances in neural information processing systems* 25 (2012), págs. 1097-1105.

[19] Alec Radford, Luke Metz y Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[20] *Las renovables alcanzan el 43,6 de la generación de energía eléctrica en 2020, su mayor cuota desde que existen registros*. 2020. URL: https://www.ree.es/es/sala-de-prensa/actualidad/nota-de-prensa/2020/12/las-renovables-alcanzan-el-43-6-por-ciento-de-la-generacion-de-2020-su-mayor-cuota-desde-existen-registros.

[21] Tero Karras, Samuli Laine y Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2019. arXiv: 1812.04948 [cs.NE].

[22] Phillip Isola y col. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].