

Grado en Ingeniería Informática
2020-2021

Trabajo Fin de Grado

Diseño adversarial para la industria del videojuego

Alejandro Valverde Mahou

Tutor

Alejandro Cervantes Rovira

Leganés, Julio de 2021



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

RESUMEN

El presente Trabajo de Fin de Grado propone la utilización de técnicas adversariales para la generación de imágenes. Concretamente se propone el uso de algoritmos basados en redes de neuronas para generar imágenes que puedan ser utilizadas en distintos videojuegos. Este trabajo se centra en un único conjunto de imágenes. Se decide hacer uso de imágenes de Pokémon, ya que es un conjunto de datos con una cantidad reducida y muy variada de ejemplos. Por tanto, se hace especial hincapié en técnicas aplicables a los modelos generativos para que sean capaces de crear buenos resultados con el menor número de ejemplos posibles. Para llevar a cabo la generación de imágenes se hace uso de algoritmos basados en redes de neuronas pertenecientes a la familia de las Generative Adversarial Networks (GAN).

Las fases principales de este Trabajo de Fin de Grado son la obtención y preparación de los datos, el desarrollo e implementación de modelos y técnicas adversariales para la creación de imágenes, y por último una evaluación de los resultados obtenidos haciendo uso de métricas numéricas y encuestas al público general con el fin de determinar el modelo capaz de generar imágenes de mejor calidad.

Los resultados obtenidos no guarden un gran parecido con el conjunto de imágenes originales, y por tanto no se pueda considerar un éxito en ese aspecto. Sin embargo, a pesar de sus limitaciones, con un mayor desarrollo, estas técnicas podrían llegar a ser utilizadas en la industria del videojuego con el objetivo de reducir costes de diseño y aumentar la cantidad de contenido.

Palabras clave: Inteligencia Artificial; Redes de Neuronas Artificiales; Generación de Imágenes; Redes Generativas Adversariales; Videojuegos; Pokémon

ÍNDICE GENERAL

1. INTRODUCCIÓN.	1
1.1. Objetivo	1
1.2. Motivación	1
1.3. Metodología	2
1.4. Trabajos relacionados	3
2. ESTADO DEL ARTE.	9
2.1. Inteligencia artificial y aprendizaje automático	9
2.2. Redes de neuronas artificiales.	10
2.2.1. Capas densas	11
2.2.2. Capas convolucionales.	11
2.2.3. Capas convolucionales transpuestas	12
2.3. <i>Autoencoders</i>	12
2.3.1. VAE	14
2.4. GAN.	15
2.4.1. DCGAN.	17
2.4.2. AEGAN.	17
2.4.3. Pix2Pix	20
2.5. Técnicas de Mejora	22
2.5.1. <i>Differentiable Augmentation</i>	22
2.5.2. <i>Progressive Growing</i>	23
2.6. Evaluación de modelos generativos adversariales	24
2.6.1. Inspección manual	25
2.6.2. <i>Inception Score</i>	25
2.6.3. <i>Frechet Inception Distance</i>	26
3. ESTUDIO Y OBTENCIÓN DE LOS DATOS	29
3.1. ¿Qué son los Pokémon?	29
3.2. Obtención de las imágenes de los Pokémon	30

3.3. Análisis de las imágenes.	32
3.3.1. Diferenciación de los Pokémon por tipo	32
3.3.2. Diferenciación de los Pokémon por generación	33
4. EXPERIMENTACIÓN Y RESULTADOS	35
4.1. DCGAN sobre <i>sprites</i>	35
4.1.1. DCGAN básico.	35
4.1.2. DCGAN con <i>Differentiable Augmentation</i>	36
4.1.3. DCGAN con <i>Differentiable Augmentation</i> y <i>Dropout</i>	37
4.1.4. DCGAN con <i>Differentiable Augmentation</i> y <i>Autoencoders</i>	38
4.1.5. Comparativa de modelos	39
4.2. DCGAN sobre iconos	41
4.2.1. DCGAN con <i>Differentiable Augmentation</i>	42
4.2.2. DCGAN con <i>Differentiable Augmentation</i> y <i>Autoencoders</i>	42
4.3. Pix2Pix de iconos a <i>sprites</i>	43
4.4. <i>Autoencoder</i> de iconos a <i>sprites</i>	45
4.5. AEGAN sobre <i>sprites</i>	45
4.6. Planteamientos descartados	46
4.6.1. Uso de información adicional a la hora de generar imágenes	47
4.7. Técnicas Avanzadas	50
4.7.1. AEGAN.	50
4.7.2. VAE	51
4.7.3. Progressive Growing.	51
5. EVALUACIÓN	53
5.1. FID	53
5.2. Encuesta informal	55
6. CONCLUSIONES Y TRABAJOS FUTUROS	69
6.1. Conclusiones Profesionales	69
6.2. Conclusiones Personales.	70
6.3. Trabajos Futuros	70

7. GESTIÓN DEL TRABAJO	73
7.1. Recursos utilizados.	73
7.1.1. Recursos <i>Hardware</i>	73
7.1.2. Recursos <i>Software</i>	73
7.2. Planificación	74
7.2.1. Tarea 1: Obtención de Conocimientos	74
7.2.2. Tarea 2: Recopilación de Datos	74
7.2.3. Tarea 3: Desarrollo de las Redes Generativas Adversariales	74
7.2.4. Tarea 4: Comparativa de los Modelos	75
7.2.5. Tarea 5: Escritura de la Memoria	76
7.2.6. Tarea 6: Reuniones con el Tutor y Preparación de la Presentación	76
7.2.7. Diagrama de Gantt	76
7.3. Marco regulador	77
7.3.1. Reglamento General de Protección de Datos	77
7.3.2. Ley de Propiedad Intelectual	79
7.3.3. Propiedad Intelectual de los Resultados	79
7.4. Entorno socioeconómico	79
7.4.1. Presupuesto	79
7.4.2. Impacto socioeconómico	82
BIBLIOGRAFÍA	83
8. ANEXO	
8.1. Abstract	
8.2. Introduction.	
8.2.1. Objectives.	
8.2.2. Motivation	
8.3. What are Pokémon?	
8.4. Results	
8.4.1. DCGAN.	
8.4.2. DCGAN with Pix2Pix	
8.5. Evaluation.	

8.6. Conclusions and Future Work.	
8.6.1. Professional Conclusions	
8.6.2. Personal Conclusions	
8.6.3. Future Work	

ÍNDICE DE FIGURAS

1.1	Algunos <i>sprites</i> de Pokémon de aspectos diferentes	2
1.2	Desarrollo <i>Incremental</i> aplicado en el trabajo	3
1.3	Esquema de la aplicación de las dos redes neuronales en el modelo <i>Face-to-Parameter</i> [2]	4
1.4	Resultados del modelo pokeGAN de <i>moxiegushi</i> [3]	5
1.5	Resultados del modelo PokeGAN de Conor Lazarou[6]	6
1.6	Imagen original (izquierda) y representación en caracteres (derecha) del Pokémon <i>Bulbasaur</i> ([9] minuto 5:54)	7
1.7	Resultados del acercamiento a través de GPT-2 de Matthew Rayfield[9]	7
2.1	Funcionamiento de una neurona artificial (elaboración propia)	10
2.2	Aplicación de una convolución sobre una imagen de 3x3 con un filtro de 2x2 (elaboración propia)	11
2.3	<i>Autoencoders</i> con diferentes tamaños del vector latente. Dimensión del vector, de arriba a abajo, 64, 1024 y 8192 (elaboración propia)	13
2.4	<i>Autoencoders</i> para la eliminación del ruido en la imagen original(arriba) y relleno de huecos(abajo) (elaboración propia)	13
2.5	<i>Autoencoders</i> donde el decodificador toma la media de los vectores latentes de dos imágenes distintas (elaboración propia)	14
2.6	VAE aplicado sobre el conjunto de datos MNIST con su correspondiente espacio latente[12]	16
2.7	Estructura básica de ejemplo de DCGAN[16]	17
2.8	Habitaciones generadas a través de un DCGAN[15]	18
2.9	Arquitectura de la AEGAN[17]	19
2.10	Ejemplos de problemas que se pueden resolver haciendo uso de una arquitectura Pix2Pix[18]	20
2.11	Arquitectura U-Net simplificada[18]	21
2.12	Esquema simplificado del funcionamiento del Discriminador en la arquitectura Pix2Pix[18]	21
2.13	Aplicación del <i>Differentiable Augmentation</i> en el Generador y el Discriminador[20]	22

2.14	Proceso de realización del <i>Differentiable Augmentation</i> sobre dos imágenes de ejemplo[20]	23
2.15	Proceso de entrenamiento haciendo uso de <i>Progressive Growing</i> [21]	24
2.16	Proceso de incorporación de nuevas capas realizando un suavizado en la técnica <i>Progressive Growing</i> [21]	24
2.17	Ejemplos de distintas imágenes clasificadas con <i>Inception Classifier</i> [25].	25
2.18	Distribución de las etiquetas ideal (izquierda) y distribución marginal ideal (derecha)[25].	26
2.19	Distintas evaluaciones del FID sobre imágenes con alteraciones[26].	27
3.1	Familia evolutiva de <i>Shinx</i> . De izquierda a derecha, <i>Shinx</i> , <i>Luxio</i> y <i>Luxray</i>	29
3.2	Icono (izquierda) y <i>sprite</i> (derecha) del Pokémon <i>Bulbasaur</i>	31
3.3	Distintos Pokémon cuya apariencia está fuertemente inspirada por su tipo. De arriba a abajo, Pokémon de tipo fuego, planta y agua	32
3.4	Distintos Pokémon cuya apariencia no tiene relación con su tipo. De arriba a abajo, Pokémon que parecen de un tipo, pero son de otro; Pokémon que requieren de contexto sobre que representa para entender su tipo	33
3.5	Última evolución de cada Pokémon inicial de todas las generaciones, empezando por la primera (arriba) y acabando en la octava (abajo)	34
4.1	Arquitectura del Generador (izquierda) y Discriminador (derecha) para la generación de <i>sprites</i>	35
4.2	Resultados obtenidos con el DCGAN básico	36
4.3	Resultados obtenidos con el DCGAN básico utilizando <i>Differentiable Augmentation</i>	37
4.4	Resultados obtenidos con el DCGAN básico utilizando <i>Differentiable Augmentation</i> y <i>Dropout</i>	38
4.5	Resultados del entrenamiento previo usando <i>Autoencoders</i>	39
4.6	Resultados obtenidos con el DCGAN básico utilizando <i>Differentiable Augmentation</i> e inicialización de pesos con <i>Autoencoders</i>	39
4.7	Resultados de las distintas arquitecturas que hacen uso de una DCGAN	40
4.8	DCGAN con <i>Differentiable Augmentation</i> y preentrenamiento usando <i>Autoencoders</i> con un tiempo de entrenamiento más largo	41
4.9	Arquitectura del Generador (izquierda) y Discriminador (derecha) para la generación de iconos	41
4.10	Resultados obtenidos con DCGAN y <i>Differentiable Augmentation</i>	42

4.11	Resultados obtenidos con DCGAN, <i>Differentiable Augmentation</i> e inicialización de pesos con <i>Autoencoders</i>	43
4.12	Arquitectura del Generador (izquierda) y Discriminador (derecha) para la transformación de iconos a <i>sprites</i>	43
4.13	Resultados obtenidos con la arquitectura Pix2Pix	44
4.14	Resultados obtenidos con <i>Autoencoder</i> para transformar de icono a <i>sprite</i>	45
4.15	Resultados del AEGAN para la reconstrucción y generado de imágenes de Pokémon	46
4.16	Interpolado de las imágenes reconstruidas por AEGAN	47
4.17	Distribución del número de Pokémon por tipo	48
4.18	Porcentaje de Pokémon de cada clase correctamente clasificados según su tipo	48
4.19	Esquema de la arquitectura propuesta para generar Pokémon con información adicional	49
4.20	Esquema de la arquitectura propuesta para generar Pokémon generando primero Pokémon en escala de grises, y posteriormente dotarles de color	49
4.21	Porcentaje de Pokémon correctamente clasificados según su generación	50
5.1	FID de los modelos que generan <i>sprites</i> representados como gráfico de barras	55
5.2	Respuestas de la pregunta 1 de la encuesta.	57
5.3	Imágenes del modelo 1 (<i>DCGAN</i>) y las respuestas de los usuarios	58
5.4	Imágenes del modelo 2 (<i>DCGAN + DF</i>) y las respuestas de los usuarios	59
5.5	Imágenes del modelo 3 (<i>DCGAN + DF + Dropout</i>) y las respuestas de los usuarios	60
5.6	Imágenes del modelo 4 (<i>DCGAN + DF + Autoencoder</i>) y las respuestas de los usuarios	61
5.7	Imágenes del modelo 5 (<i>Pix2Pix</i>) y las respuestas de los usuarios	62
5.8	Imágenes del modelo 6 (<i>Pix2Pix-Gen</i>) y las respuestas de los usuarios	63
5.9	Imágenes del modelo 7 (<i>Autoencoder</i>) y las respuestas de los usuarios	64
5.10	Imágenes del modelo 8 (<i>Autoencoder-Gen</i>) y las respuestas de los usuarios	65
5.11	Resultados de la encuesta respecto a los modelos de iconos	66
5.12	Imágenes del modelo 1 de iconos (<i>DCGAN + Autoencoder</i>)	66
5.13	Imágenes del modelo 2 de iconos (<i>DCGAN</i>)	67

6.1	Arquitectura final propuesta	70
7.1	Diagrama de Gantt del trabajo	78
A.1	Some <i>sprites</i> of Pokémon of different appearances	
A.2	Evolutionary family of <i>Shinx</i> . From left to right, <i>Shinx</i> , <i>Luxio</i> and <i>Luxray</i>	
A.3	Pre-training results using <i>Autoencoders</i>	
A.4	Results obtained with the basic DCGAN using <i>Differentiable Augmentation</i> and initialization of weights with <i>Autoencoders</i>	
A.5	Results obtained with DCGAN, <i>Differentiable Augmentation</i> and initialization of weights with <i>Autoencoders</i>	
A.6	Results obtained with the Pix2Pix architecture	
A.7	Final model architecture. (In spanish, from left to right <i>Latent Vector</i> , <i>Low resolution generated image</i> , <i>High Resolution generated image</i>	

ÍNDICE DE TABLAS

5.1	FID de los modelos que generan <i>sprites</i>	54
5.2	FID de los modelos que generan iconos	55
7.1	Tarea 1: Obtención de Conocimientos	74
7.2	Tarea 2: Recopilación de Datos	75
7.3	Tarea 3: Desarrollo de las Redes Generativas Adversariales	75
7.4	Tarea 4: Comparativa de los Modelos	76
7.5	Tarea 5: Escritura de la Memoria	76
7.6	Tarea 6: Reuniones con el Tutor y Preparación de la Presentación	77
7.7	Fases del trabajo con su duración	77
7.8	Costes de Personal del trabajo	80
7.9	Costes de <i>Hardware</i>	80
7.10	Costes de <i>Software</i>	81
7.11	Costes Totales	81
A.1	Models FID	

1. INTRODUCCIÓN

1.1. Objetivo

La generación de imágenes con técnicas adversariales está cobrando mucha relevancia y se está popularizando en gran medida. Esto ha llevado a una explosión de diferentes trabajos, nuevas arquitecturas de red y numerosas aplicaciones que hacen uso de estas técnicas.

Este trabajo plantea hacer uso de estrategias adversariales a través de redes neuronales con el objetivo de crear nuevas imágenes utilizables en la industria del videojuego. Concretamente, se va a intentar generar Pokémon nuevos, parecidos a los originales, pero distintos a cualquiera que ya exista.

A pesar de que existen distintas formas de generar imágenes, este trabajo se centrará exclusivamente en acercamientos que se basen en redes de neuronas artificiales con un entrenamiento adversarial, o lo que es lo mismo, modelos compuestos por dos o más redes de neuronales con objetivos enfrentados.

Por otro lado, este trabajo pretende servir como base de una línea de investigación orientada a la resolución del problema de generación automática de imágenes para videojuegos. Por lo tanto, se tiene también el objetivo de plantear propuestas para obtener arquitecturas de red competitivas con las mejores disponibles hasta el momento, como trabajo futuro de investigación.

1.2. Motivación

La motivación principal de este trabajo es acercar las técnicas adversariales basadas en redes de neuronas a la industria del videojuego, demostrando su utilidad, y los beneficios que ofrece. De poderse aplicar correctamente permitiría a distintos desarrolladores de videojuegos conseguir una gran cantidad de imágenes similares al conjunto original en un tiempo reducido, abaratando costes en creación de imágenes.

Debido a que en la actualidad no existe ninguna aportación que resuelva el problema de forma totalmente satisfactoria, en el ámbito de este proyecto se considerará suficiente desarrollar modelos cuyos resultados sean capaces de confundir a personas que no estén totalmente familiarizadas con el dominio del problema.

Otro motivo por el que se plantea utilizar el conjunto de datos de Pokémon es por su gran variabilidad y diferencia, como se puede apreciar en la Figura 1.1. Esto supone una dificultad añadida para los modelos basados en redes de neuronas, cuya tarea es realizar

generalizaciones sobre los datos, que, si son demasiado diferentes, resulta más costoso.

Hay que tener en cuenta que las técnicas comunes en generación automática de imágenes suelen utilizar como base de entrenamiento colecciones muy extensas de datos. Es el caso, por ejemplo, de las que usan imágenes de rostros humanos. En este caso, existe una limitación importante en cuanto a la cantidad de imágenes originales disponibles, que además son muy desiguales entre sí. Es importante ser capaces de evaluar cómo funcionan las arquitecturas establecidas en los trabajos de investigación en estas condiciones, y si requieren modificaciones o técnicas de aumento de datos adicionales para proporcionar resultados satisfactorios.



Figura 1.1. Algunos *sprites* de Pokémon de aspectos diferentes

El análisis y estudio de datos se realiza en mayor detalle en la sección 3 Estudio y obtención de los datos.

A título personal, se desea realizar este trabajo para profundizar en los conceptos sobre redes de neuronas planteados en la carrera, así como aprender nuevas tecnologías utilizadas en el mundo real. Además, dado el interés y conocimiento del autor sobre Pokémon, se podrá hacer uso de información adicional para construir los distintos modelos de redes.

1.3. Metodología

A la hora de desarrollar este trabajo se ha seguido un proceso cíclico con el objetivo de mejorar lo máximo posible los resultados obtenidos. El proceso que se ha seguido ha sido, inicialmente, realizar una investigación previa sobre el tema o tecnología que se quiere implementar. Después se ha seguido con la implementación y experimentación. Por último, se realiza una evaluación y comparación con el resto de acercamientos ya implementados previamente.

Seguir estos pasos de forma cíclica ha permitido que la solución inicial haya ido mejorándose a lo largo del desarrollo.

Si la tecnología implementada generaba resultados positivos, se han mantenido y se han seguido desarrollando. En caso contrario se han descartado. Estos planteamientos descartados se explican en mayor detalle en la sección 4.6 Planteamientos descartados. Este proceso se aprecia de forma visual en el esquema de la Figura 1.2. Por tanto, la metodología usada en este Trabajo de Fin de Grado es una modificación del modelo 'Incremental'.

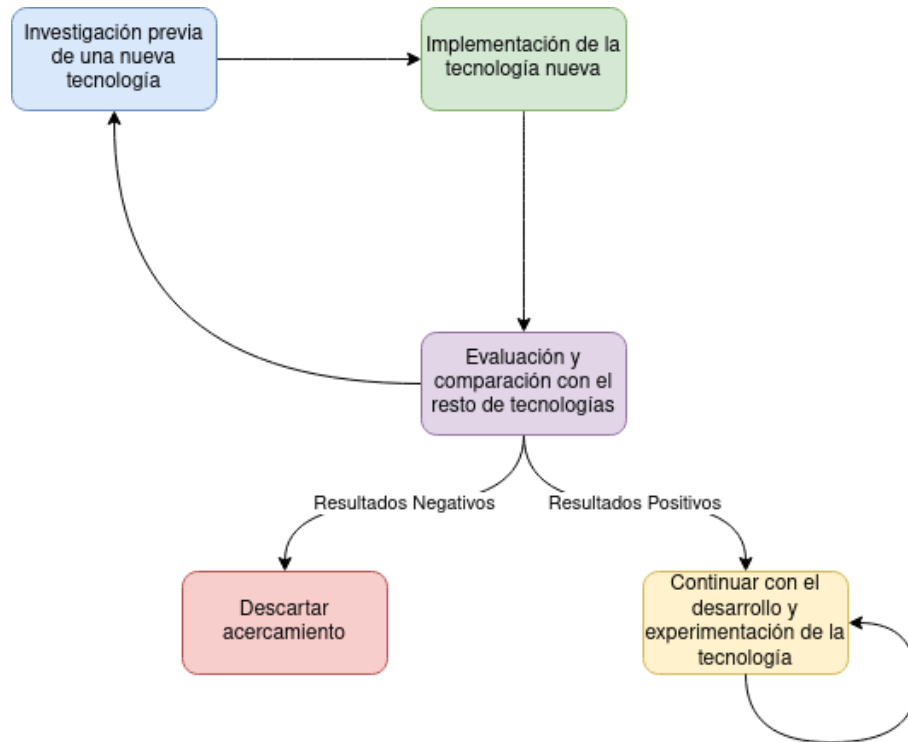


Figura 1.2. Desarrollo *Incremental* aplicado en el trabajo

1.4. Trabajos relacionados

Con el objetivo de introducir un contexto, y presentar mejor el problema a tratar, en este apartado se muestran ejemplos que enfocan la cuestión a tratar haciendo uso de acercamientos y técnicas distintas basadas en redes de neuronas artificiales.

El uso de redes de neuronas para generar gráficos dentro de la industria del videojuego no es un tema popular actualmente; es posible que esto se deba a la necesidad de estas redes de tener una gran cantidad de imágenes de referencia para poder crear imágenes nuevas. Poco a poco van apareciendo nuevos avances y formas de utilizar estas tecnologías para aplicarse a este contexto concreto. Uno de los trabajos más notables es el realizado por *NVidia*[1] que ha sido capaz de crear entornos de conducción interactivos realizando una transformación desde el videojuego *GTA V* a videos fotorealistas, utilizando como motor de renderización el modelo de redes de neuronas.

Otro trabajo que está basado en redes de neuronas adversariales para crear recursos utilizables en videojuegos es *Face-to-Parameter Translation for Game Character Auto-*

Creation[2], que permite crear personajes para videojuegos *RPG* (*Role-Playing Games*) a partir de imágenes de personas reales. Para ello, hace uso de una pareja de redes de neuronas: Imitador (G) y un Extractor de características (F). El primero se encarga de, dadas una serie de características de la imagen, crear un modelo tal y como lo haría un motor de un videojuego. Es decir, dada una serie de parámetros, renderiza un personaje completo. Las características son creadas por el Extractor de características, y además compara las características que ha generado el Imitador frente a las características que obtiene de la imagen original.

De esta forma, ambas redes entran en un juego adversarial donde el Imitador intenta generar las imágenes de la forma más fiel a las características que recibe como entrada, y el Extractor de Características intenta incrementar la dificultad, y por tanto el detalle de los mapas de características para conseguir una mayor diferencia entre la imagen real y la generada. En la Figura 1.3 se puede ver un esquema de este flujo de datos. Después de un entrenamiento, los autores consiguen generar resultados con muy buena calidad y relación entre el modelo virtual generado, y la imagen de referencia original.

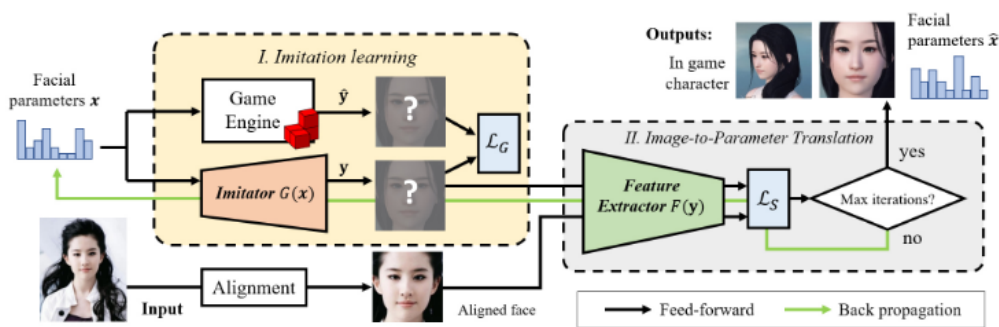


Figura 1.3. Esquema de la aplicación de las dos redes neuronales en el modelo *Face-to-Parameter*[2]

Como se ha mencionado en el apartado 1.2 Motivación, a fecha de este trabajo, no se ha desarrollado un modelo que genere correctamente imágenes de Pokémon nuevos que pasen como Pokémon reales.

Dado que la creación de Pokémon usando técnicas basadas en redes de neuronas es un tema de interés para un nicho muy reducido de investigadores y desarrolladores, no hay excesivos trabajos que traten este tema concreto. Los intentos que consiguen resultados más destacables dentro de este campo específico son:

- **pokeGAN de moxiegushi**[3]. Este modelo es el que mejores resultados ha obtenido a la hora de generar Pokémon realistas haciendo uso de una arquitectura básica, pero aún está muy lejos de llegar al punto que pueda engañar al ojo humano.

La Figura 1.4 muestra los resultados que el usuario de GitHub *moxiegushi* ha conseguido con su modelo. A pesar de que algunas imágenes, según su autor, no tienen

demasiado sentido, consigue un par de logros muy relevantes. Las imágenes cuentan con bordes muy marcados, y los colores dentro de la mayoría de imágenes es coherente.

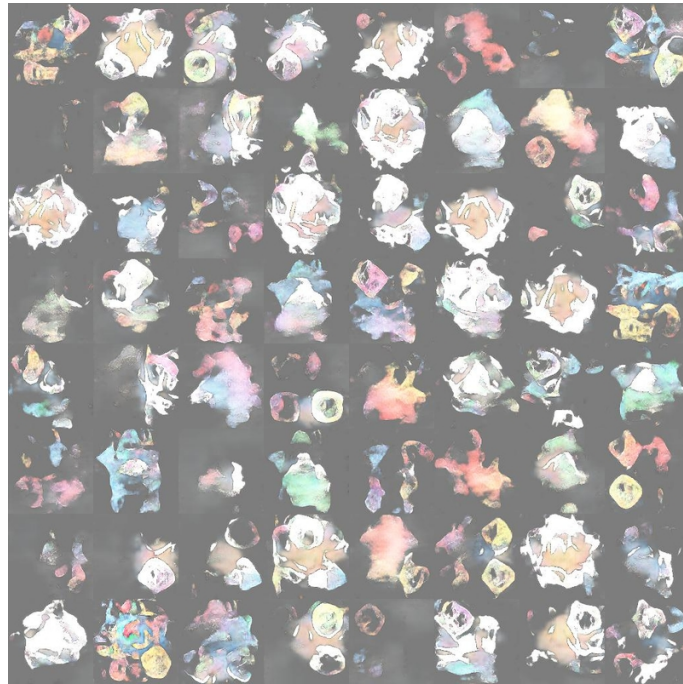


Figura 1.4. Resultados del modelo pokeGAN de *moxiegushi*[3]

El acercamiento que plantea este autor es una red generativa adversarial del tipo WGAN, aunque sugiere que una DCGAN generaría resultados similares. Trabaja con imágenes cuadradas de 256x256 píxeles, que es un tamaño relativamente alto para este tipo de modelos, pero consigue resultados aceptables. Este trabajo sirve como ejemplo para demostrar la complejidad que supone este proyecto en concreto, dado que es difícil conseguir resultados no solo que sean capaces de engañar al ojo humano, sino que ni siquiera consigan formas y colores que tengan sentido.

En este trabajo no se hace uso de fotografías reales como ejemplo, sino de dibujos, que tienen menor nivel de detalle, pero también son más abstractos y diferentes unos de otros.

Con este acercamiento hay varios trabajos similares (DCGAN sobre imágenes de 64x64[4]; DCGAN y VAE sobre un conjunto más amplio de imágenes[5] entre otros trabajos similares), aunque sin duda el que ha conseguido mejores resultados usando este acercamiento es el realizado por el usuario *moxiegushi*[3].

- **PokeGAN de Conor Lazarou**[6][7]. El autor de este trabajo opta por un acercamiento diferente al anterior, usando un modelo más sofisticado, junto a varias mejoras sobre el modelo original.

El modelo que decide usar es AEGAN, que se explicará en más detalle en la sección

4.5 AEGAN sobre *sprites*. Este modelo permite generar mejores resultados, aunque requiere una programación más compleja y un mayor tiempo de entrenamiento.

Dado que trabaja sobre *sprites* de los juegos originales, las imágenes son de una resolución relativamente baja (96x96 píxeles) y tienen una paleta de colores reducida. Por este motivo, con el objetivo de conseguir un mejor resultado y homogeneidad en el color, realiza un cambio en la última capa del generador, de forma que en lugar de usar los 3 canales que se usan típicamente (RGB), usa 16 canales para cada píxel, y luego en lugar de usarlos todos, usa uno exclusivamente, y este hace referencia a un color concreto de una lista de 16 colores. De esta forma, consigue resultados que usan colores muy similares dentro de la misma imagen.

Además, el autor decide añadir en las capas del Discriminador ruido, afirmando que esto hace que el modelo genere mejores resultados, porque reduce los entrenamientos inestables. Por último, también realiza un aumento de datos básico, y así es como genera las imágenes que se pueden ver en la Figura 1.5. Es interesante como el modelo ha aprendido no solo a tener consistencia en los colores, sino que además se pueden diferenciar ciertas partes reconocibles, como piernas, brazos e incluso ojos en algún caso.

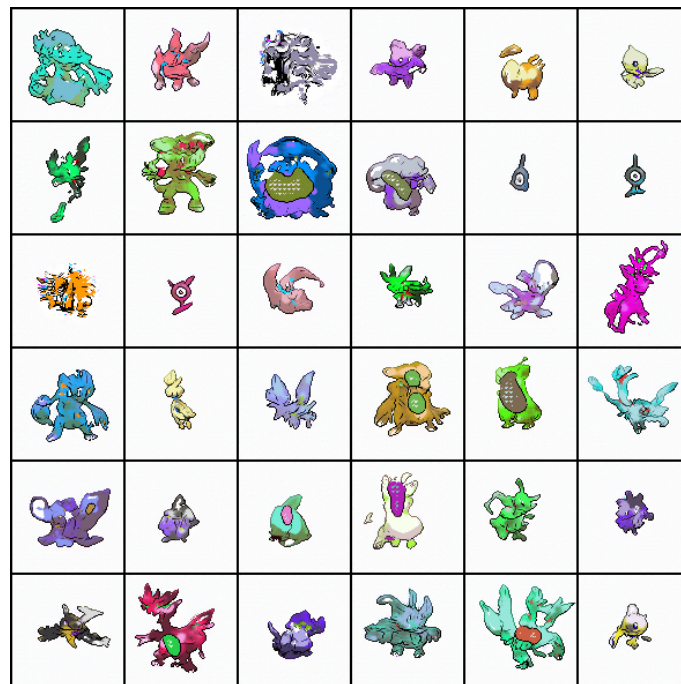


Figura 1.5. Resultados del modelo PokeGAN de Conor Lazarou[6]

Sin duda este acercamiento es mucho más refinado y complejo que el acercamiento anterior, pero, por ejemplo, la estrategia usada para la selección de colores no es aplicable en imágenes de mayor resolución o que tengan una gama de colores más amplia.

- **pokemon-gpt-2 de Matthew Rayfield[8][9]**. Este acercamiento es radicalmente

distinto al resto, ya que no hace uso de estrategias adversariales, sino que utiliza la arquitectura GPT-2[10], que es una arquitectura que a pesar de que tiene usos en la generación de imágenes, se usa principalmente en el tratamiento de textos.

El acercamiento que propone este autor es, sobre las imágenes de baja resolución, realizar una transformación a caracteres de texto, tal y como se muestra en la Figura 1.6.

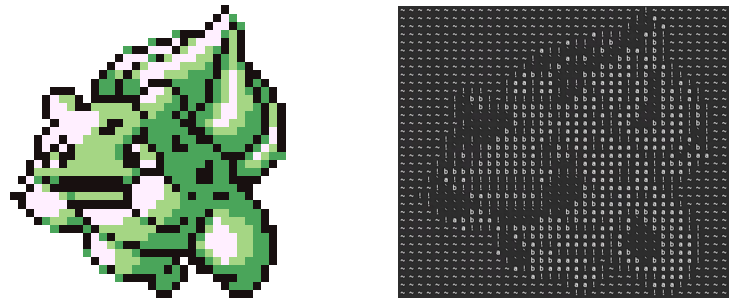


Figura 1.6. Imagen original (izquierda) y representación en caracteres (derecha) del Pokémon *Bulbasaur* ([9] minuto 5:54)

Una vez ha generado estas imágenes que se representan con caracteres de texto, se entrena un modelo usando la arquitectura GPT-2, y los resultados son de una calidad muy buena, como se puede apreciar en la Figura 1.7. A pesar de que a través de este acercamiento no se vean tantas partes diferenciadas en las imágenes como en el acercamiento anterior, los colores y las formas tienen una buena definición, y parecen tener motivos similares a los que podrían tener los Pokémon reales. A pesar de que no consiga los mejores resultados, es destacable el ingenio que supone utilizar una arquitectura que tiene como objetivo generar o analizar texto para realizar generación de imágenes.



Figura 1.7. Resultados del acercamiento a través de GPT-2 de Matthew Rayfield[9]

El principal problema que tiene este acercamiento es que no se puede aplicar sobre

imágenes de alta resolución, ni con demasiados colores, dado que el modelo está limitado por el tamaño de una línea, y el número de caracteres existentes. El autor trabaja con imágenes de 64x64 para conseguir sus resultados.

2. ESTADO DEL ARTE

2.1. Inteligencia artificial y aprendizaje automático

La inteligencia artificial es una rama de la informática que se encarga de crear programas y algoritmos que se comportan con cierta inteligencia. Consiste en hacer que los ordenadores se comporten con una inteligencia que se suele asociar con los humanos, para resolver distintos problemas. El término fue acuñado en el año 1956 por Jhon McCarthy, para describir "la ciencia e ingeniería de crear máquinas inteligentes"[11].

El concepto de inteligencia artificial abarca un dominio muy grande, y recoge una gran variedad de algoritmos y técnicas diferentes. Por otro lado, el aprendizaje automático (*machine learning*) es una subcategoría de la inteligencia artificial que se centra en aquellos algoritmos que llevan a cabo un aprendizaje a través de ejemplos, con el fin de mejorar su comportamiento. El aprendizaje automático se puede entender como mejorar una tarea T , respecto a una medida de rendimiento P basándose en la experiencia E .

El tipo de tareas que permite resolver el aprendizaje automático son aprendizaje por refuerzo, aprendizaje supervisado y aprendizaje no supervisado.

- **Aprendizaje por refuerzo:** Consiste en un sistema con estados, donde se pueden realizar distintas acciones que generan cierto refuerzo o castigo. El objetivo es encontrar una política que maximice el refuerzo obtenido realizando la mejor acción en cada estado, o en su defecto, encontrando la acción que minimice el castigo.
- **Aprendizaje supervisado:** Este tipo de aprendizaje intenta modelar la relación y dependencias entre la entrada de un modelo y su salida. Por tanto, los datos utilizados en el entrenamiento deben estar compuestos por una entrada y una salida deseada. Si el problema que se está tratando tiene una salida numérica, se trata de un problema de regresión. Por otro lado, si el problema tiene como salida deseada una o varias etiquetas, se trata de un problema de clasificación.
- **Aprendizaje no supervisado:** Es utilizado principalmente para resolver problemas de agrupamiento, que intenta determinar cuánto se parecen entre sí los distintos ejemplos. En este caso los datos están compuestos únicamente por una entrada, y es el algoritmo o modelo el que tiene que determinar el parecido entre los ejemplos dados.

2.2. Redes de neuronas artificiales

Las redes de neuronas son una de las técnicas de aprendizaje automático más populares actualmente, gracias principalmente al aprendizaje profundo (*deep learning*) que ha sufrido una explosión de popularidad en los últimos años, debido a los resultados espectaculares que se han conseguido con ellas. Las redes de neuronas artificiales aparecieron en 1943 de la mano de Warren McCulloch y Walter Pitts, con las células de McCulloch-Pitts, las precursoras de las neuronas artificiales modernas.

Las redes de neuronas están compuestas, tal y como indica su nombre, por neuronas artificiales y están diseñadas para simular el comportamiento del cerebro humano. Dentro de las redes de neuronas, existen una ingente cantidad de arquitecturas diferentes, y no todas cumplen los mismos propósitos.

Cada neurona sigue el mismo comportamiento, y es al unirse cuando son capaces de llevar a cabo tareas complejas. El funcionamiento de una neurona se define en la Figura 2.1. Cada neurona tiene una serie de entradas (x_i), que se multiplican por sus correspondientes pesos (w_i), y se suman. A esta suma se añade un umbral (b) que siempre toma el valor de 1, por tanto, su valor depende de su peso (w_0). Por último, se aplica a la suma total una función de activación (F) que suele ser no lineal, y se genera la salida de la neurona (y).

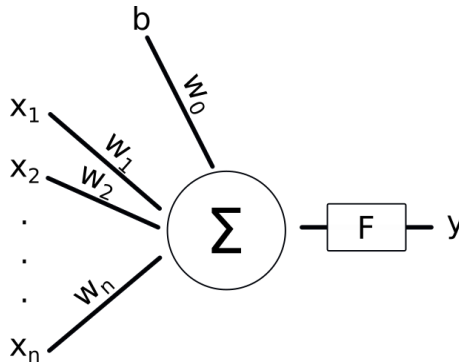


Figura 2.1. Funcionamiento de una neurona artificial (elaboración propia)

Por tanto, se puede definir la salida de una neurona según la Ecuación 2.1.

$$y = F(w_0 + \sum_{i=1}^n x_i * w_i) \quad (2.1)$$

El entrenamiento de estas neuronas se produce modificando iterativamente los pesos en función del error cometido, hasta que convergen cuando el error es mínimo. Este entrenamiento sencillo solo se puede aplicar cuando hay una única capa de neuronas. En caso contrario, dado que las neuronas no están en contacto con el error directamente, es necesario aplicar el algoritmo de retropropagación.

Las neuronas se estructuran en capas, para obtener comportamientos más complejos. A continuación se definen algunas de las capas de neuronas más relevantes en este trabajo.

2.2.1. Capas densas

Las capas densas son aquellas donde cada neurona recibe como entrada todas las neuronas de la capa anterior, y a su vez está conectada con todas las neuronas de la capa siguiente. Estas neuronas deben tener una función de activación no lineal, para conseguir, al combinarse, un comportamiento más complejo. Es el tipo de capa de neuronas más básico, y está presente en casi todas las arquitecturas de redes de neuronas.

2.2.2. Capas convolucionales

Las capas convolucionales son, tal como indica el nombre, capas que aplican convoluciones sobre sus entradas. Una convolución no es más que la aplicación de un filtro o *kernel* a una entrada concreta. Un filtro es una matriz que va recorriendo la entrada extrayendo sus características. Dado que este trabajo está centrado en el tratamiento de imágenes, todas las convoluciones serán bidimensionales. Aplicar un filtro a una imagen permite obtener características de la misma. En las capas convolucionales lo que se intenta es cambiar el valor de los filtros para conseguir los resultados deseados. En la Figura 2.2 se muestra la aplicación de un filtro de tamaño 2x2 sobre una imagen de tamaño 3x3, resultando en una imagen de 2x2. Normalmente, en cada capa de convolución, se aplican varios *kernel* diferentes, con el objetivo de conseguir más información de la imagen en cada paso.

Imagen

Filtro

Resultado

1	2	3
4	5	6
7	8	9

1	0
2	5

34	42
58	66

1	2	3
4	5	6
7	8	9

=1*1 + 2*0 + 4*2 + 5*5 = 34

1	2	3
4	5	6
7	8	9

= 42

1	2	3
4	5	6
7	8	9

= 58

1	2	3
4	5	6
7	8	9

= 66

Figura 2.2. Aplicación de una convolución sobre una imagen de 3x3 con un filtro de 2x2 (elaboración propia)

Las capas convolucionales son especialmente útiles para el procesado de imágenes, ya que son capaces de obtener características de las mismas. Estas capas convolucionales se suelen combinar con capas de *pooling*, que no son realmente neuronas, sino que aplican al conjunto de entrada algún tipo de función resumen, como seleccionar el máximo o la media.

2.2.3. Capas convolucionales transpuestas

Las capas convolucionales transpuestas nacen de la necesidad de tener una operación inversa a las capas de convolución. Así como las capas de convolución transforman la información de las imágenes de alto nivel en una serie de características de las mismas, las capas de convolución transpuesta transforman información de bajo nivel (que pueden ser características o simplemente ruido) a imágenes.

Este tipo de capas son necesarias especialmente en modelos generativos, que tienen la necesidad de crear imágenes a partir de información de un nivel más bajo. El funcionamiento de estas capas es muy sencillo. Cada una de las entradas de la imagen se multiplica por el *kernel* y se introduce en la matriz resultante, de forma que si se usa un *kernel* de tamaño 2x2, para cada píxel en la imagen de entrada, aparecerán 4 píxeles en la imagen de salida, por lo que si se aplica sobre toda la imagen original, el resultado será una imagen con el doble de tamaño.

Al igual que en las capas convolucionales, el entrenamiento consiste en encontrar la configuración óptima de cada uno de los *kernel* que se aplican.

2.3. Autoencoders

Los *Autoencoders* son un tipo de redes de neuronas no supervisadas cuyo objetivo es aprender representaciones o codificaciones de un conjunto de datos. Normalmente esto se hace con el objetivo de reducir la dimensionalidad, aunque también puede utilizarse para eliminar ruido de imágenes, detectar datos anómalos, o reconstruir partes borrosas de una imagen.

La arquitectura de un *Autoencoder* consiste en un codificador y un decodificador, donde el codificador intenta reducir la dimensionalidad del conjunto original, y el decodificador intenta hacer la acción inversa, partiendo de la codificación, recuperar la imagen original. Por eso se considera que pertenece al aprendizaje no supervisado, porque no hace uso de etiquetas, sino que, sin control externo, se entrena para recuperar la imagen original. Por tanto, siendo $E()$ el codificador, $D()$ el decodificador, x la imagen original, z el vector latente, o codificación y $\tilde{x}$ la imagen reconstruida por el decodificador, en la Ecuación 2.2 y la Ecuación 2.3 se muestran las transformaciones realizadas.

$$E(x) = z \quad (2.2)$$

$$D(z) = \tilde{x} \quad (2.3)$$

Y el entrenamiento intenta reducir la diferencia entre x y $\tilde{x}$.

El tamaño del vector latente tiene cierta importancia. Cuanto más pequeño sea este vector, menos información podrá almacenar, y por tanto peor reconstrucción se podrá hacer. Cuanto mayor sea, mejor calidad tendrá la reconstrucción, pero más espacio ocupará la compresión. En la Figura 2.3 se puede ver el efecto que tiene sobre las imágenes originales un tamaño del vector latente diferente.

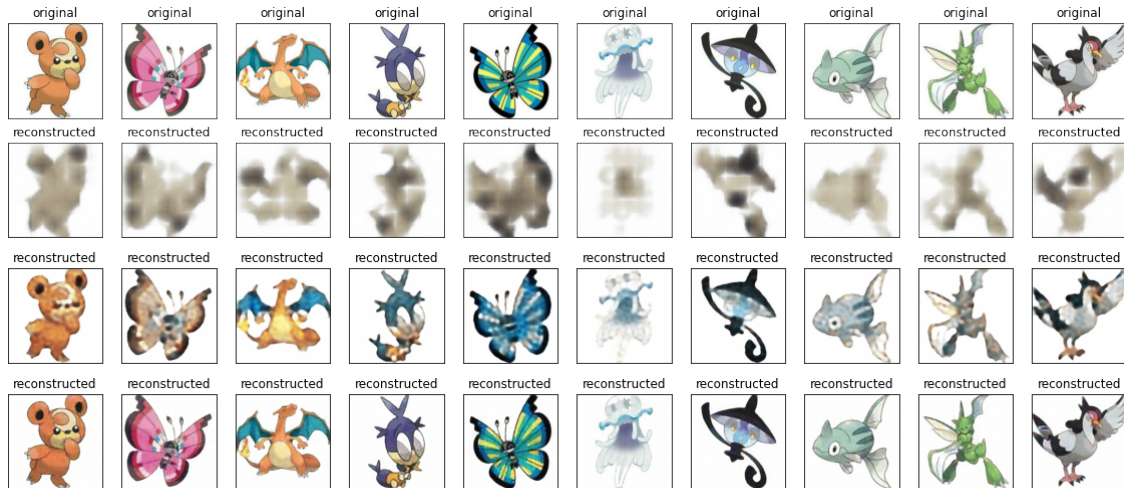


Figura 2.3. *Autoencoders* con diferentes tamaños del vector latente. Dimensión del vector, de arriba a abajo, 64, 1024 y 8192 (elaboración propia)

Como se ha mencionado, esta arquitectura, aparte de reducir la dimensionalidad de un conjunto de datos, puede realizar otras tareas, como eliminación de ruido, o reparación de imágenes. En la Figura 2.4 se muestra una implementación sencilla de estas dos técnicas.

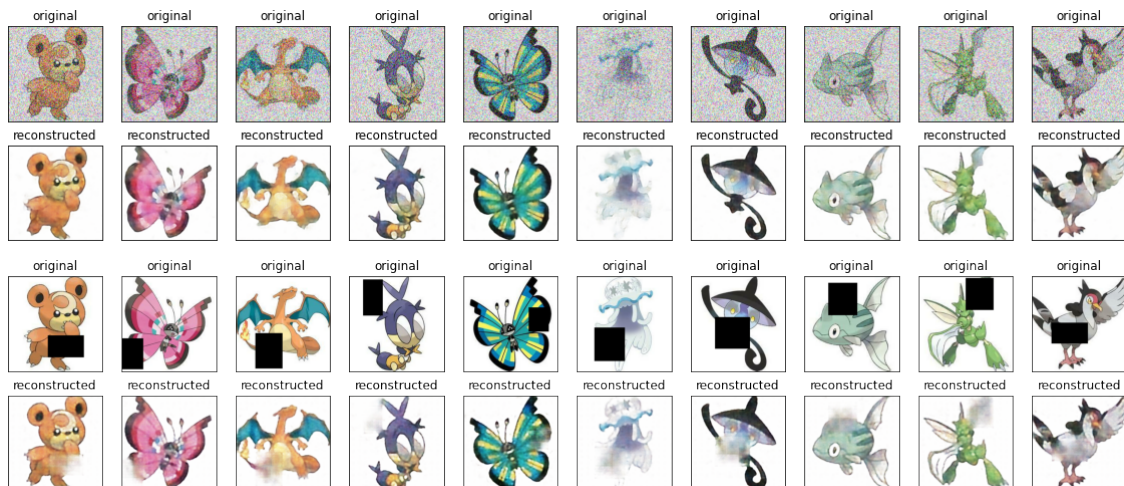


Figura 2.4. *Autoencoders* para la eliminación del ruido en la imagen original(arriba) y relleno de huecos(abajo) (elaboración propia)

2.3.1. VAE

Los VAE (*Variational Autoencoders*) son un tipo de *Autoencoder* que hace uso de datos probabilísticos para crear imágenes nuevas parecidas al conjunto de datos original. En lugar de usar un vector latente para comprimir la información, usa parámetros de distribución de probabilidad, como puede ser la media o la varianza. De esta forma, se crea un espacio continuo y estructurado, que es especialmente útil para la generación de imágenes[12].

La idea principal es usar un espacio para los vectores latentes continuo, de forma que si se obtiene el vector latente de a y de b y se hace una media, idealmente se tendría que encontrar un valor intermedio entre ambos. Esto no ocurre con los *Autoencoders* convencionales, ya que las muestras se agrupan en espacios latentes inconexos, y si se intenta realizar el cálculo del punto intermedio entre dos muestras, el resultado es, tal y como se puede ver en la Figura 2.5, un valor que no es realista, sino la superposición de las dos entradas.

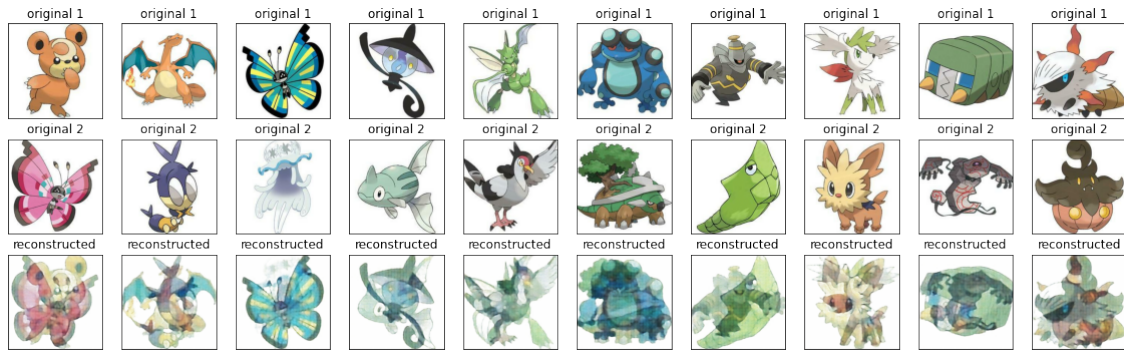


Figura 2.5. *Autoencoders* donde el decodificador toma la media de los vectores latentes de dos imágenes distintas (elaboración propia)

Estos resultados deficientes se obtienen porque, en la mayoría de casos, las representaciones de los espacios latentes no son continuas. Si el espacio de representación es discontinuo, y se intenta generar una imagen tomando un espacio intermedio, el decodificador producirá una salida no realista, porque no tiene información suficiente sobre esta región del espacio latente, ya que durante su entrenamiento nunca ha visto nada parecido.

Los VAE intentan resolver este problema construyendo un espacio de vectores latentes continuo. Para conseguirlo, en lugar de usar una representación intermedia fija de un tamaño concreto, usan dos vectores distintos donde cada uno representa una cosa: un vector para las medias $\vec{\mu} = (\mu_1, \dots, \mu_n)$ y un vector para las desviaciones estándar $\vec{\sigma} = (\sigma_1, \dots, \sigma_n)$, que unidas forman un vector de variables aleatorias descritas a través de una distribución normal ($N(\mu_1, \sigma_1), \dots, N(\mu_n, \sigma_n)$).

De esta forma, el vector latente que representa cada muestra es variable, y depende de su media y su varianza, por tanto no puede aprender que un solo punto en el espacio

representa un ejemplo, sino que tiene que aprender que cada ejemplo está representado por una distribución aleatoria, y tendrá que aplicar criterios de probabilidad para reconstruirla.

A lo largo del espacio, el decodificador irá asociando áreas completas a un mismo ejemplo, y no un punto aislado, y ligeras variaciones sobre este vector supondrá un cambio suavizado e interpolado.

La función de coste que se utiliza en este tipo de redes se denomina divergencia KL, que en lugar de medir la distancia entre dos puntos concretos, mide la diferencia entre distribuciones de probabilidad, de forma que se comparan la distribución normal generada respecto a la real que se quiere aproximar. Dado que la función de distribución real es desconocida, se usa una distribución centrada en 0, de forma que KL viene dado por la Ecuación 2.4.

$$KL = \sigma^2 + \mu^2 - \log \sigma - 1 \quad (2.4)$$

Pero también si no se considera nada más, no será capaz de diferenciar los diferentes ejemplos, y se tratarán todos por igual. Por eso es necesario además utilizar la función original de los *Autoencoders*. Estas dos funciones de coste se pueden apreciar en la Ecuación 2.5.

$$VAE_{loss} = KL + d(x, \tilde{x}) \quad (2.5)$$

donde d representa una función que mide la distancia entre la imagen original x y la imagen reconstruida $\tilde{x}$.

Gracias a esta función de coste, se facilita la continuidad del espacio de representación, y permite generar resultados como los que se pueden ver en la Figura 2.6 [12].

2.4. GAN

Las GAN[13] (*Generative Adversarial Network*) hace referencia a una familia de arquitecturas de redes de neuronas que son capaces de generar datos de distintos tipos donde dos o más redes compiten sobre funciones de coste opuestas para conseguir los mejores resultados posibles. Este tipo de arquitecturas suelen usarse principalmente en la generación y tratamiento de imágenes, siendo algunas de sus implementaciones más conocidas, crear caras de gente que no existe, convertir imágenes en blanco y negro a color o generar imágenes a partir de texto.

Las GAN están compuestas por una red generador G y una red discriminador D . Estas dos redes están enfrentadas, donde G intenta crear una serie de imágenes falsas, y D tiene que determinar si cada imagen es real o falsa, de forma que ambas redes entran en un juego donde D cada vez es capaz de determinar con mayor precisión si una imagen

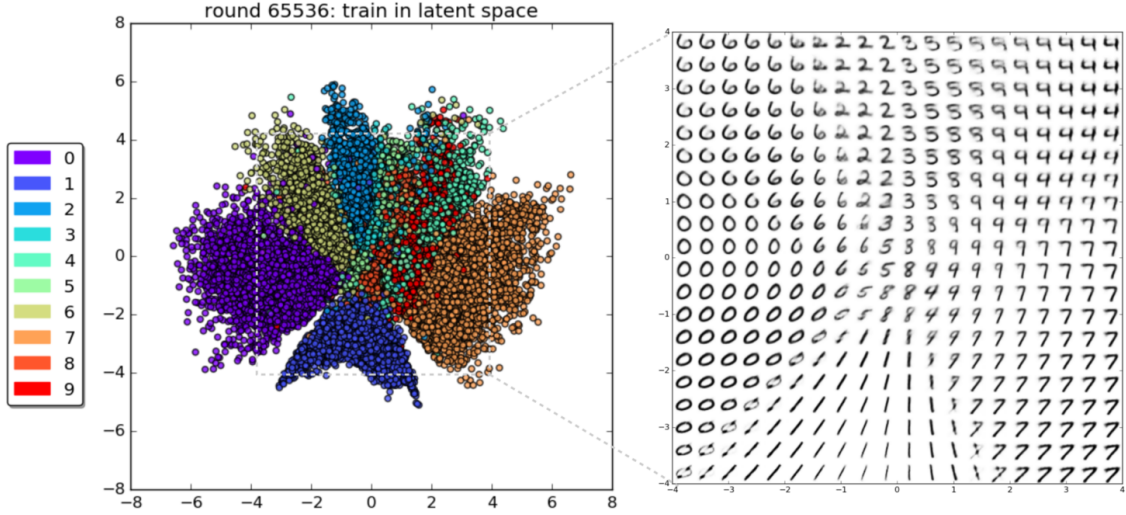


Figura 2.6. VAE aplicado sobre el conjunto de datos MNIST con su correspondiente espacio latente[12]

es real, o ha sido generada, y G por su parte cada vez es capaz de engañar con mayor efectividad a D . De esta forma, G cada vez crea imágenes con un mayor parecido a las imágenes originales, sin tener nunca acceso al conjunto de datos original.

Este tipo de arquitectura entran dentro de la subcategoría de aprendizaje no supervisado, ya que los datos no tienen una etiqueta asociada, pero a su vez, durante el proceso de aprendizaje, hace uso de una función de coste supervisada, en D , para determinar si una imagen es real o generada.

Para generar imágenes, G parte de un vector latente z , compuesto por números generados aleatoriamente, y a partir de ellos se crean las imágenes generadas $G(z)$. Después, estas imágenes generadas se pasan a D , junto a imágenes reales x . Se entrena para que D minimice la probabilidad de equivocarse al categorizar las imágenes, y G se entrena para minimizar el número de veces que no ha sido capaz de engañar a D , por tanto, la función de coste de D se encuentra en la Ecuación 2.6 y la función de coste de G se describe en la Ecuación 2.7

$$L_D = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \quad (2.6)$$

$$L_G = \mathbb{E}_{z \sim p(z)}[\log(D(G(z)))] \quad (2.7)$$

Por lo que, G y D están enzarzados en un juego de suma 0, o de minimax[14], donde la mejora de uno se traduce directamente en el empeoramiento del opuesto.

A lo largo de los años se han ido desarrollando variaciones y mejoras sobre esta arquitectura inicial. En este trabajo se hace uso de algunas de estas modificaciones con el objetivo de encontrar la mejor solución al problema planteado.

2.4.1. DCGAN

Las DCGAN[15] (*Deep Convolutional Generative Adversarial Network*) son la evolución natural de las GAN, pues plantea usar redes convolucionales para el tratamiento de imágenes, que proporcionan una mejora significativa de los resultados frente a las GAN convencionales, que hacen uso de capas densas.

La arquitectura de este tipo de redes, al igual que las GAN convencionales, están compuestas por un Generador (G) y un Discriminador (D). D recibe como entrada tanto imágenes reales como generadas por G , y por tanto está compuesto por una serie de capas convolucionales que extraen las características de cada imagen y una capa densa final con una única neurona encargada de determinar si es real la imagen o no.

Por su lado, G está compuesta por una serie de capas convolucionales transpuestas que, partiendo de un vector latente z , son capaces de generar imágenes nuevas.

En la Figura 2.7 se muestra un ejemplo básico y esquemático de este tipo de arquitecturas.

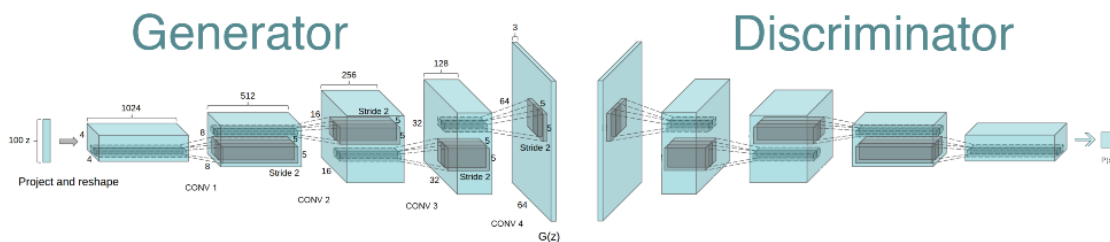


Figura 2.7. Estructura básica de ejemplo de DCGAN[16]

Este cambio en la arquitectura supone una inmensa mejora frente al modelo general y permite ser utilizada para multitud de aplicaciones diferentes, al mismo tiempo que permite el desarrollo de varias modificaciones y cambios que continúan mejorando esta técnica partiendo de esta arquitectura.

En el *paper* original, la aplicación sobre la que muestran las capacidades de la red es la generación de imágenes tomando como ejemplo un conjunto de datos que consiste en imágenes de distintas habitaciones, como se puede apreciar en la Figura 2.8.

2.4.2. AEGAN

La AEGAN[17] (*AutoEncoding Generative Adversarial Network*) es una red generativa más compleja, pues hace uso de una combinación entre GAN y *Autoencoder*. Fue propuesta por *Conor Lazarou* en 2020, y el propio autor utilizó su arquitectura para realizar una experimentación similar al trabajo que se plantea[7].

El funcionamiento de esta arquitectura, aunque pueda parecer muy complejo, se basa



Figura 2.8. Habitaciones generadas a través de un DCGAN[15]

en conceptos sencillos ya definidos en este trabajo.

Uno de los problemas que poseen las GAN convencionales es que, dado que usan transformaciones inyectivas que se pueden expresar como $G : Z \rightarrow X$, no son capaces de realizar la transformación inversa. Esto supone un problema al usar ciertos conjuntos de datos, ya que el discriminador acaba realizando una clasificación perfecta, o cuasi perfecta, de forma que no devuelve información relevante al generador, que acaba colapsando sin poder converger.

La AEGAN intenta resolver este problema utilizando un *Autoencoder* y una GAN. La idea principal reside en aprender una transformación biyectiva que se puede expresar como $G : Z \rightarrow X$ y $E : X \rightarrow G$. En la Figura 2.9 se muestra en detalle la arquitectura de la AEGAN.

Dada la complejidad de la arquitectura presentada en la Figura 2.9, a continuación se procede a realizar una explicación detallada de la misma.

En primer lugar es necesario definir las 4 redes que forman parte de esta arquitectura, y están representadas en la Figura 2.9 con cuadrados blancos

- G es el Generador, encargado de crear imágenes. Recibe como entrada vectores latentes y devuelve imágenes generadas.
- E es el Codificador, encargado de crear vectores latentes. Recibe como entrada las imágenes y devuelve vectores latentes generados.
- D_z es el Discriminador de vectores latentes, encargado de determinar si un vector latente es real o generado.

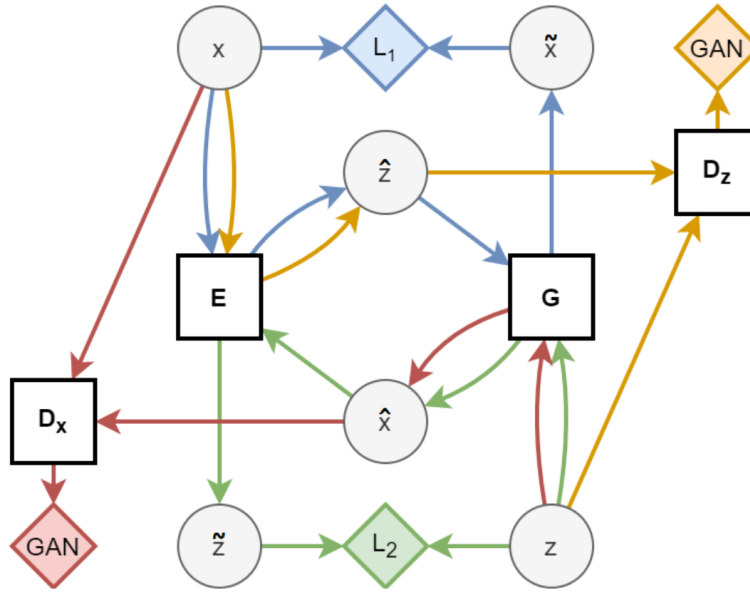


Figura 2.9. Arquitectura de la AEGAN[17]

- D_x es el Discriminador de imágenes, encargado de determinar si una imagen es real o generada.

G y E se combinan de diferentes formas con distintos objetivos que se explican a continuación. Los círculos blancos representan estas combinaciones:

- x son las imágenes reales del conjunto de datos originales.
- z son los vectores latentes generados a través de un proceso de randomización.
- $\hat{x}$ son las imágenes generadas por G a partir de z ($G(z) = \hat{x}$).
- $\hat{z}$ son los vectores latentes generados por E a partir de x ($E(x) = \hat{z}$).
- $\tilde{x}$ son las imágenes generadas por G a partir de $\hat{z}$ ($G(E(x)) = \tilde{x}$)
- $\tilde{z}$ son los vectores latentes generados por E a partir de $\hat{x}$ ($E(G(z)) = \tilde{z}$)

Por último, los cuadrados rotados de colores representan las distintas funciones de coste que se aplican a las redes.

- L_1 representa la capacidad del modelo de reconstruir imágenes usando E y G . Consiste en comparar x y $G(E(x))$. La idea con esta función de coste es realizar la función biyectiva. Se quiere que E codifique las imágenes de tal forma que G pueda realizar la transformación inversa de la mejor forma posible.
- L_2 representa la capacidad del modelo de reconstruir vectores latentes usando G y E . Al igual que L_1 , sirve para asegurar la biyectividad del modelo.

- *GAN*. Las dos GAN representan lo mismo, pero con diferentes entradas. La GAN que está junto a D_x recibe como entrada x y $G(z)$ y es la encargada de discriminar cuáles son reales y cuáles generadas, mientras que el que está junto a D_z hace lo propio, pero con z y $E(x)$.

Los resultados de este modelo son bastante notables y suponen una mejora significativa en el conjunto de datos empleados por el autor. Además, de manera indirecta, este modelo consigue una propiedad que es muy interesante en los modelos generativos, y es la capacidad de interpolación. Al igual que en los VAE, ya mencionados previamente en este trabajo, AEGAN trabaja con un espacio de vectores latentes continuos, por lo que es capaz de interpolar no solo las imágenes generadas, sino también las imágenes de entrenamiento.

2.4.3. Pix2Pix

Pix2Pix[18] (*Pixel to Pixel*) es una arquitectura que hace uso de un acercamiento adversarial a la hora de transformar una imagen a otra. Al igual que los *Autoencoders*, permite la transformación de una imagen a otra, pero el uso de un entrenamiento adversarial permite una mayor capacidad de abstracción y resolución de problemas más complejos.

Mientras que los *Autoencoders* se plantean generalmente para problemas de eliminación de ruido o reconstrucción de algunas partes ocultas de la imagen, los Pix2Pix tienen aplicaciones más complejas. Algunas de las presentadas en el *paper* original son transformar desde etiquetas de colores a una escena de una carretera, transformar de una vista aérea a un mapa, transformar una imagen en blanco y negro a color o transformar una imagen diurna a una imagen nocturna, entre otras aplicaciones. Esto se puede apreciar en la Figura 2.10

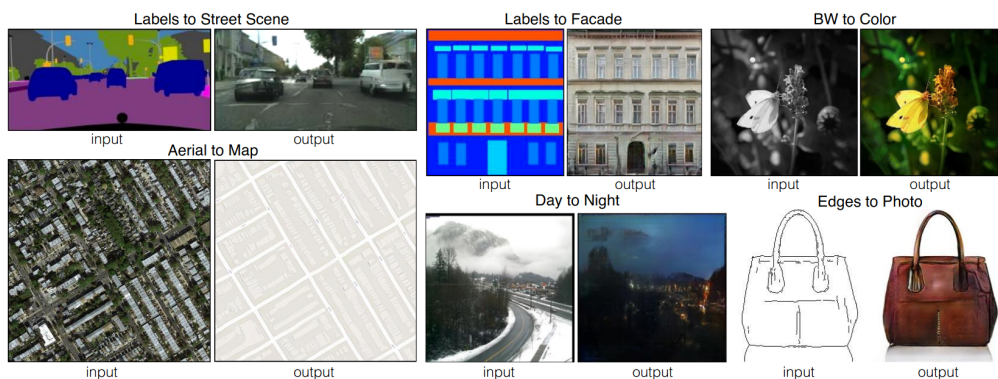


Figura 2.10. Ejemplos de problemas que se pueden resolver haciendo uso de una arquitectura Pix2Pix[18]

Al igual que los modelos generativos adversariales ya presentados en este trabajo, Pix2Pix está compuesto por un Generador G y un Discriminador D .

En este caso G recibe como entrada una imagen, y tiene que devolver una imagen diferente. Este Generador debe estar compuesto por un codificador y un decodificador, al igual que los *Autoencoders*, pero dado que muchas transformaciones requieren de un nivel de detalle alto, no se puede usar una arquitectura convencional porque, al pasar por el cuello de botella entre ambas partes, se perdería mucha información. Para resolver este problema los autores plantean el uso de una arquitectura denominada U-Net[19]. Esto se debe a que U-Net, además de estar formado por un Codificador y un Decodificador, las capas del mismo tamaño del Codificador y el Decodificador están conectadas, lo cual aporta información de la forma y estructura de la imagen original. En la Figura 2.11 se puede apreciar con más detalle esta arquitectura.

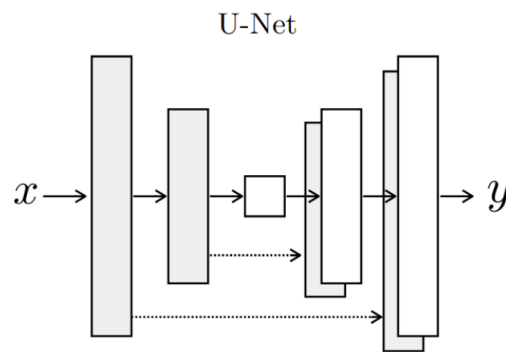


Figura 2.11. Arquitectura U-Net simplificada[18]

Por otro lado D recibe como entrada parejas de imágenes entrada-salida. De esta forma no solo aprende a determinar si una imagen es real o generada, sino que aprende a relacionar ambas imágenes para determinar si la imagen se corresponde con su entrada. Por tanto, si una imagen consigue un nivel de realismo muy alto, pero no se corresponde a su entrada, D será capaz de determinar que es una imagen generada. Este esquema se presenta en la Figura 2.12.

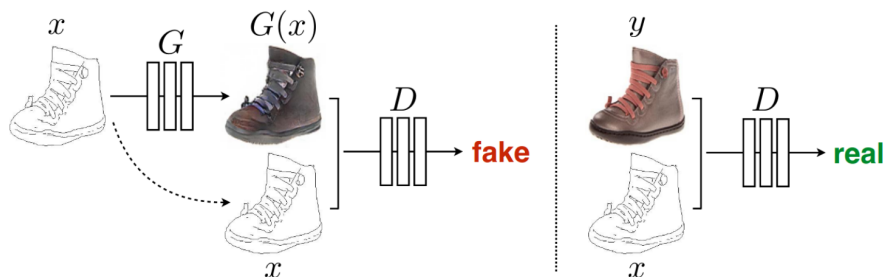


Figura 2.12. Esquema simplificado del funcionamiento del Discriminador en la arquitectura Pix2Pix[18]

Las conclusiones a las que llegan los autores originales con esta arquitectura es que el acercamiento adversarial a la transformación de imagen a imagen, especialmente si se

requiere mantener la forma detallada de la imagen, puede resolverse con la arquitectura Pix2Pix.

2.5. Técnicas de Mejora

En este apartado se explican dos técnicas distintas que no requieren de un cambio en la arquitectura de los modelos, sino en la forma de entrenar los mismos.

2.5.1. Differentiable Augmentation

Differentiable Augmentation[20] es una técnica de aumento de datos especialmente desarrollada para funcionar en arquitecturas GAN. El aumento de datos es una técnica que, aunque es aplicable sobre cualquier problema en el que haya redes neuronales artificiales, es especialmente útil cuando el conjunto de datos es reducido, o no variado.

Si se posee un conjunto reducido de datos, en el caso de las GAN, suele ocurrir que el Discriminador acaba memorizando con mucha facilidad el conjunto de datos original, de forma que es muy complicado engañarlo, y los modelos acaban colapsando.

Sin embargo, en este tipo de arquitecturas no es trivial realizar un aumento de datos. Si se aplica un aumento de datos sobre el conjunto de imágenes originales, el modelo aprenderá a generar imágenes con esas transformaciones, de forma que el resultado no sería el deseado.

La solución que plantean los autores es aplicar el aumento de datos sobre las imágenes reales y las imágenes generadas, de forma que esta transformación se aplique tanto en los resultados de la función de coste del Generador como en la del Discriminador. De esta forma, y tomando $T(x)$ como el aumento de datos sobre las imágenes x , la función de coste modificada del Discriminador se expresa en la Ecuación 2.8 y la del Generador en la Ecuación 2.9.

$$L_D = \mathbb{E}_{x \sim p_{data}(x)} [\log D(T(x))] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(T(G(z))))] \quad (2.8)$$

$$L_G = \mathbb{E}_{z \sim p(z)} [\log(D(T(G(z))))] \quad (2.9)$$

y se pueden entender visualmente en la Figura 2.13.

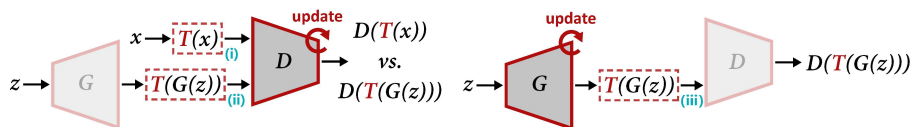


Figura 2.13. Aplicación del *Differentiable Augmentation* en el Generador y el Discriminador[20]

Realizar este aumento de datos permite a los autores conseguir resultados de una mayor calidad, especialmente cuando se realiza sobre un conjunto reducido de imágenes. En el *paper* llegan incluso a aplicar sobre distintas arquitecturas GAN conjuntos de datos de tan solo 100 imágenes con resultados de una calidad muy alta.

Las transformaciones que sugieren los autores son modificaciones aleatorias sobre el color, translaciones aleatorias y recortes aleatorios sobre las imágenes, en ese orden concreto. Esta aleatoriedad se calcula para cada imagen en cada ciclo, de forma que las probabilidades que tiene el Discriminador de ver la misma imagen repetida se reducen drásticamente, evitando así que sea capaz de memorizar las imágenes concretas, y por tanto los resultados tienden a mejorar. En la Figura 2.14 se puede ver con un ejemplo la aplicación de estas modificaciones sobre las imágenes.

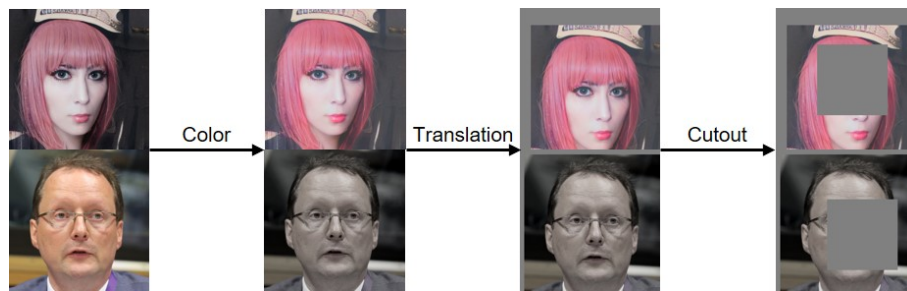


Figura 2.14. Proceso de realización del *Differentiable Augmentation* sobre dos imágenes de ejemplo[20]

2.5.2. *Progressive Growing*

Progressive Growing[21] es un método de entrenamiento de las GAN que plantea, tal como su nombre indica, realizar un entrenamiento donde el tamaño de la imagen resultante va aumentando de tamaño de forma progresiva.

La idea principal de este método de entrenamiento reside en que las capas de los modelos vayan aprendiendo la distribución de los datos secuencialmente. Es decir, el entrenamiento comienza con imágenes de baja resolución, y acaba con las imágenes originales. Esto hace que los modelos aprendan primero la estructura general de los datos, y según se añaden nuevas capas, sea capaz de definir cada vez más detalle. Este proceso se puede apreciar de forma visual en la Figura 2.15.

Este método de entrenamiento, además de suponer una mejora sobre la calidad de los resultados, reduce el tiempo de entrenamiento necesario, dado que la mayoría de iteraciones se realizan sobre las redes que trabajan con imágenes de menor resolución, y por tanto es necesario un menor número de cálculos.

Al realizar el proceso de incorporación de nuevas capas a un modelo, si no se realiza correctamente, puede hacer que, dado que la nueva capa no tiene ningún conocimiento,

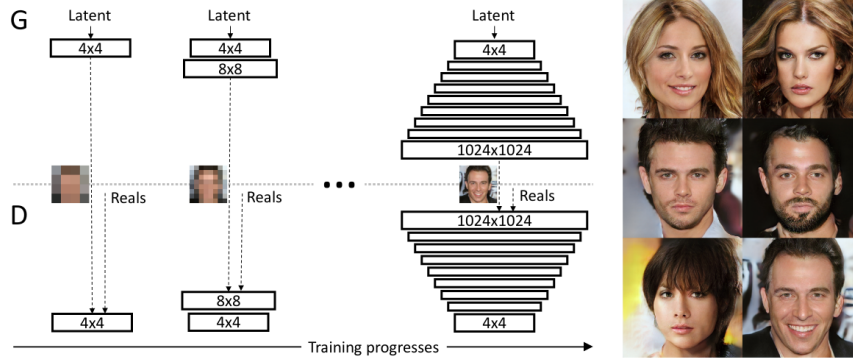


Figura 2.15. Proceso de entrenamiento haciendo uso de *Progressive Growing*[21]

el modelo colapse y no sea capaz de seguir aprendiendo, o tenga que volver a aprender desde 0 al insertar esta nueva capa. Por este motivo es necesario realizar la incorporación de forma suavizada. En la Figura 2.16 se puede apreciar este proceso donde α comienza con un valor de 0 y acaba con un valor de 1, creciendo en incrementos constantes. De esta forma, la red que trabajaba con imágenes de 16×16 pasa a trabajar con imágenes 32×32 .

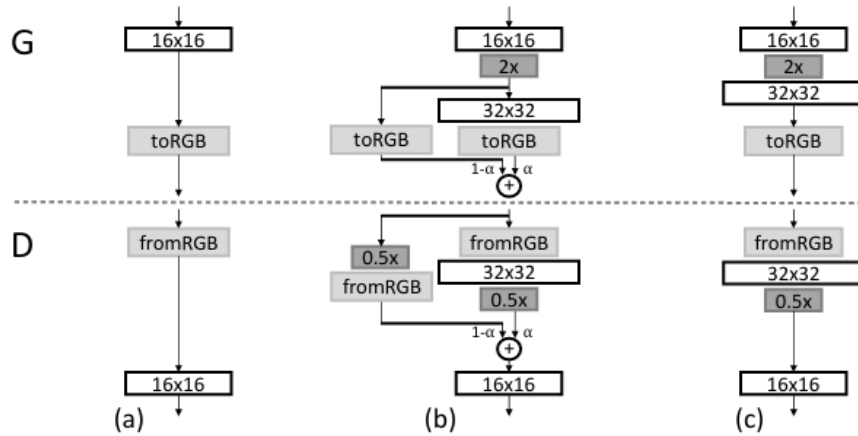


Figura 2.16. Proceso de incorporación de nuevas capas realizando un suavizado en la técnica *Progressive Growing*[21]

2.6. Evaluación de modelos generativos adversariales

La evaluación de modelos adversariales no es un proceso trivial, dado que su función de coste no determina la efectividad de los modelos o la calidad de los resultados. A día de hoy todavía no hay una medida única que se utilice para determinar cómo de bueno es un modelo.

Para que una medida sea útil, ha de cumplir ciertos requisitos[22]:

1. Tiene que ser capaz de determinar imágenes de mayor calidad frente a imágenes de una calidad más baja.

2. Tiene que ser capaz de favorecer los modelos que generen mayor variedad de imágenes.
3. Tiene que ser capaz de tolerar ligeras modificaciones sobre los datos.
4. Tiene que ser un proceso que no suponga una carga computacional excesivamente grande

Hay una gran cantidad de métricas que cumplen estos requisitos. En este apartado se explican solo algunas de las medidas más usadas.

2.6.1. Inspección manual

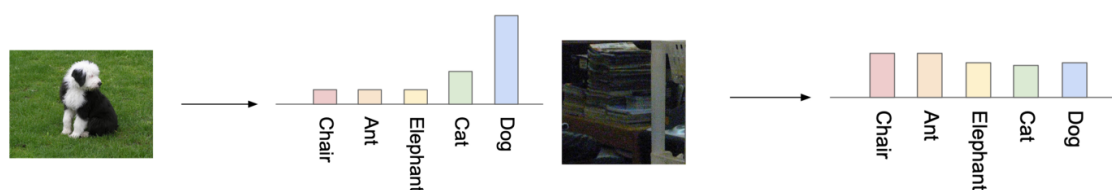
Es la medida más sencilla, pues consiste en la observación por parte de uno o varios usuarios para determinar la calidad del modelo. Los humanos tienen dificultad a la hora de evaluar con un número el valor de un modelo, y puede llegar a ser muy subjetivo, pero son extremadamente buenos a la hora de determinar si un modelo es mejor que otro. Es decir, esta tarea es especialmente útil cuando se tienen que comparar varios modelos, aunque no aporta un gran valor científico debido a su subjetividad.

2.6.2. Inception Score

El IS(*Inception Score*)[23] es una métrica muy popular a la hora de evaluar modelos generativos adversariales. Según los autores, el IS está correlacionado con la evaluación humana sobre la calidad de una imagen.

Esta medida evalúa tanto la variedad de imágenes como la calidad de las mismas, por lo que es una medida aceptable a la hora de determinar si un modelo es bueno o no.

El IS, toma su nombre del *Inception Classifier*[24], que es una red capaz de clasificar imágenes. Si la imagen tiene un objeto fácilmente reconocible por la red, su valor sobre una clase será alto, frente al resto de clases, que será más bajo. Si no hay un objeto reconocible, el valor sobre todas las clases será similar. En la Figura 2.17 se puede ver una representación gráfica de estos dos ejemplos.



(a) Clasificación de una imagen de un perro usando el *Inception Classifier* (b) Clasificación de una imagen de varios objetos usando el *Inception Classifier*

Figura 2.17. Ejemplos de distintas imágenes clasificadas con *Inception Classifier*[25].

Esta clasificación no se aplica sobre cada imagen por separado, sino que se aplica sobre un conjunto grande de imágenes, y se suman los valores obtenidos de cada etiqueta. Si la suma despunta sobre una sola clase, eso quiere decir que el conjunto de imágenes se ajusta a una única distribución, es decir, que todas las imágenes son muy parecidas. Por tanto, un modelo ideal sería aquel en el que cada imagen tiene un valor alto sobre una única clase, pero al sumarse todas las imágenes, que todas las clases estén igualadas. Por consiguiente, se requieren dos distribuciones diferentes, la distribución de las etiquetas de cada imagen y la distribución marginal de un conjunto de imágenes. En la Figura 2.18 se puede ver la situación ideal, donde se busca que cada imagen sea claramente distinguible, y que todas las imágenes tengan variedad.



Figura 2.18. Distribución de las etiquetas ideal (izquierda) y distribución marginal ideal (derecha)[25].

Esta técnica de evaluación es muy útil, pero tiene un defecto principal. Dado que se basa en la capacidad de reconocer objetos de una red ya entrenada, si intenta reconocer un objeto para el que no está entrenado, es decir, algún objeto que no pertenezca a una de sus clases, no devolverá un valor concreto de una clase, y por tanto su valor decrecerá. La única forma de resolver el problema es entrenando una red clasificadora que posea entre sus clases los mismos tipos de imágenes que se quieran generar.

Otro problema, que no se puede resolver usando una red apropiada para el problema que se quiere evaluar, es el hecho de que esta métrica no compara las muestras reales con las generadas. Es decir, no realiza ningún tipo de comparación respecto a los dos conjuntos de imágenes, y por tanto el valor que genera es demasiado arbitrario, y es necesario compararlo con los valores obtenidos respecto a las imágenes reales.

2.6.3. *Frechet Inception Distance*

El FID (*Frechet Inception Distance*)[26] es una mejora del IS. Esta métrica que calcula la distancia entre dos vectores de características, entre imágenes reales e imágenes generadas. Así resuelve uno de los principales problemas de IS.

Para conseguir los vectores de características hace uso del *Inception Classifier*, pero eliminándole la última capa, de forma que no se usa la clase que determina, sino un vector de características. Después se comparan distintas medidas estadísticas (media y varianza) del vector de características obtenido de las imágenes reales y del obtenido de las imágenes generadas.

Dado que se trata de una diferencia, el valor óptimo es 0, y cuanto mayor sea el valor, peores son los resultados. En la Figura 2.19 se puede apreciar como distintas modificaciones sobre imágenes originales hacen que el valor del FID aumenten.

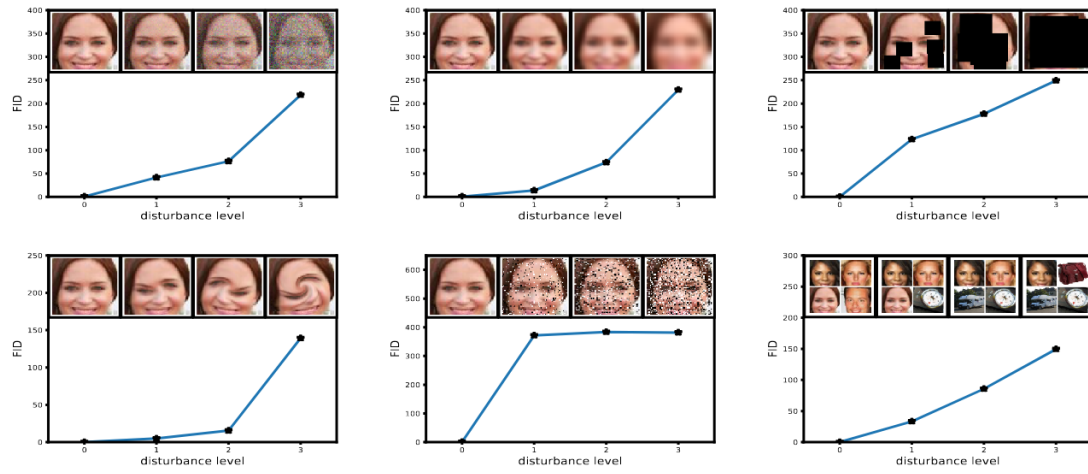


Figura 2.19. Distintas evaluaciones del FID sobre imágenes con alteraciones[26].

3. ESTUDIO Y OBTENCIÓN DE LOS DATOS

3.1. ¿Qué son los Pokémon?

Pokémon[27](ポケモン) es una franquicia de medios de entretenimiento mundialmente conocida. Su principal atractivo son las criaturas que dan nombre a la marca, los Pokémon. Estas criaturas nacieron en 1996 de la mano de *Satoshi Tajiri* en los videojuegos *Pokémon Rojo* y *Pokémon Azul*, creados por la compañía *Game Freak*[28]. En estos videojuegos, el jugador tiene que emprender un viaje como entrenador Pokémon. Su meta es cumplir dos objetivos principalmente. Capturar todas las especies de Pokémon diferentes para así conseguir recopilar toda la información sobre los mismos, y a su vez, entrenarlos para que consigan más poder y les permita derrotar a otros entrenadores Pokémon hasta convertirse en el mejor entrenador Pokémon de todo el juego.

Desde sus inicios en 1996, Pokémon no ha dejado de ganar popularidad en todo el mundo, entre otras cosas, gracias a que se han seguido creando criaturas nuevas cada pocos años, que se incorporaban, junto a las ya existentes, en los videojuegos, series de televisión, mangas y demás productos de la compañía.

Estos Pokémon, diseñados por *Ken Sugimori*, suelen estar basados en animales u objetos del mundo real y suelen ser muy reconocibles, siendo radicalmente distintos unos de otros.

Cada uno de los Pokémon tiene varios aspectos y características que los hacen únicos. Como es lógico, cada Pokémon tiene su propio nombre y aspecto. Además, pueden tener una o varias evoluciones, que son transformación que puede alcanzar el Pokémon si se cumplen unas condiciones concretas. A su vez, cada Pokémon pertenece a uno o dos tipos, que, normalmente, se pueden adivinar por la forma y color del Pokémon. Existen actualmente un total de 18 tipos. En la Figura 3.1 se puede apreciar la familia evolutiva del Pokémon *Shinx*, donde las tres evoluciones son de tipo eléctrico.



Figura 3.1. Familia evolutiva de *Shinx*. De izquierda a derecha, *Shinx*, *Luxio* y *Luxray*

Además de los tipos, otra forma de clasificar a los Pokémon es según su generación. La generación de un Pokémon representa simplemente cuando fueron creados. Siguiendo el ejemplo anterior, la familia evolutiva de *Shinx* completa pertenece a la cuarta generación, o lo que es lo mismo, aparecieron por primera vez en los juegos *Pokémon Diamante* y *Pokémon Perla* en el año 2007. Esto es una característica importante a tener en cuenta, ya que los diseños de los Pokémon de cada generación son distintos, y siguen estilos artísticos diferentes. Este tema se desarrolla en mayor profundidad en la sección 3.3 Análisis de las imágenes.

Hay numerosas formas diferentes de clasificar Pokémon. Por ejemplo, se puede clasificar por poder, forma, color e incluso rareza, pero para este proyecto se utilizará exclusivamente el tipo y la generación, ya que son las que visualmente tienen una mayor importancia.

3.2. Obtención de las imágenes de los Pokémon

Actualmente existen 898 Pokémon, divididos en ocho generaciones diferentes, aunque hay determinados Pokémon que tienen varias formas. Para este trabajo se van a usar exclusivamente dos tipos de imágenes:

- **sprites**: Se han denominado así las imágenes de alta resolución de los Pokémon, es decir, las que representan más fielmente a un Pokémon.

El término *sprite* se usa incorrectamente en este caso, ya que la definición de un *sprite* es una imagen que forma parte de un videojuego, lo que no ocurre con estas imágenes. Aún así, se decide usar este término, para que sea más sencillo determinar de qué tipo de imágenes se está hablando.

- **iconos**: Se decide usar la palabra icono para representar las imágenes de baja resolución de los Pokémon.

Se usa este nombre porque las imágenes que se han decidido usar son aquellas que dentro de los videojuegos aparecen como iconos en diferentes pantallas.

En la Figura 3.2 se puede ver la diferencia entre los dos tipos de imágenes descritas previamente.

Para obtener estas imágenes se ha hecho uso de una de las páginas web en español dedicada a Pokémon, *Wikidex*[29]. Es la mayor enciclopedia Pokémon en español, que posee páginas especializadas para cada uno de los Pokémon existentes.

Para recopilar todas las imágenes, inicialmente se decidió hacer una recopilación manual, pero cuando se vio el enorme esfuerzo que eso supondría, se pasó a usar técnicas de *web scraping* con revisado manual para obtener todas las imágenes.



Figura 3.2. Icono (izquierda) y *sprite* (derecha) del Pokémon *Bulbasaur*

Una vez se han recopilado todas las imágenes de los 898 Pokémon, teniendo en cuenta formas alternativas, se obtiene un total de 1132 *sprites* y 1132 iconos. Cuando se tienen todas estas imágenes, se ha realizado una transformación sobre las mismas.

Las imágenes de Pokémon, por desgracia, no están estandarizadas para un tamaño concreto, ni a una posición concreta dentro de la imagen. Por este motivo, hay imágenes que no son cuadradas, o que tienen más borde alrededor de la imagen. Con el objetivo de facilitar el trabajo de las redes de neuronas, se intenta realizar una estandarización. Para ello, se siguen los siguientes pasos:

1. **Eliminar los espacios de los bordes**, de forma que el Pokémon esté tocando los cuatro márgenes en las imágenes.
2. **Añadir márgenes para que las imágenes sean cuadradas**. Esto se hace para las imágenes que son rectangulares. Se calcula en qué dimensión son más largas, y se añaden píxeles vacíos en la otra dirección hasta que se alcanza una imagen cuadrada.
3. **Se transforma de .png a .jpg**. Esto se hace con el objetivo de reducir el número de canales (de RGBA a RGB), y por tanto, la cantidad de información que tendrán que tratar los modelos de redes de neuronas. Simplemente se ponen en blanco aquellos píxeles que son transparentes en las imágenes originales. Esta transformación se realiza exclusivamente sobre los *sprites*, ya que se considera que los iconos, al ser de tamaño más reducido, no requieren de una mayor simplificación.
4. **Se escalan las imágenes**, con el objetivo de que tengan todas el mismo tamaño, ya que si no, no podrían darse como entrada a los modelos de redes neuronales. El tamaño que se usa para los *sprites* es 256x256 y el tamaño de los iconos es 32x32.

Por último, y para poder contar con un mayor número de imágenes, se almacena tanto la imagen procesada, como la imagen procesada con un volteo en el eje vertical, haciendo un total de 2264 instancias de cada tipo de imagen.

3.3. Análisis de las imágenes

En este apartado se va a realizar un análisis más intensivo sobre las imágenes.

3.3.1. Diferenciación de los Pokémon por tipo

Como se ha mencionado previamente existen un total de 18 tipos Pokémon diferentes. Cada uno de estos tipos representa un estado elemental en el mundo Pokémon y son atributos que se asocian a cada Pokémon y a cada uno de los ataques que pueden realizar. Cada uno de los tipos tiene una serie de ventajas y debilidades frente a otros, convirtiendo los juegos en una especie de piedra-papel-tijera con muchas más variables y complicaciones.

Cada Pokémon puede pertenecer a uno o dos tipos elementales. Este factor no solo afecta a sus características dentro del juego, sino también a su apariencia. Por ejemplo, los Pokémon que pertenecen al tipo *planta* suelen tener un color más verdoso, y sus formas son semejantes a plantas del mundo real, mientras que los Pokémon de tipo *fuego* suelen ser rojos o naranjas, con formas de distintos animales.

La existencia de esta clasificación propia dentro de los datos puede ser relevante a la hora de generar nuevos Pokémon. En la Figura 3.3 se puede apreciar Pokémon de varios tipos, y como su apariencia está fuertemente correlacionada con su tipo.



Figura 3.3. Distintos Pokémon cuya apariencia está fuertemente inspirada por su tipo. De arriba a abajo, Pokémon de tipo fuego, planta y agua

Por desgracia, este tipo de acercamientos tiene una serie de fallos. El primero es que hay muchos Pokémon cuyo tipo no es fácilmente identificable por su apariencia, y el

segundo que muchas de las formas de identificar el tipo del Pokémon son conociendo atributos de la vida real. En la Figura 3.4 se muestran una serie de Pokémon difíciles de clasificar por algún motivo, ya sea porque su esquema de colores no cuadra, o porque es necesario tener el conocimiento de a qué se parece para relacionar el concepto con el tipo al que pertenece. Dado que las redes no tienen este conocimiento previo del mundo real, acercamientos que usen el tipo del Pokémon, puede que no funcionen correctamente.



Figura 3.4. Distintos Pokémon cuya apariencia no tiene relación con su tipo. De arriba a abajo, Pokémon que parecen de un tipo, pero son de otro; Pokémon que requieren de contexto sobre que representa para entender su tipo

3.3.2. Diferenciación de los Pokémon por generación

Es innegable que el estilo artístico de Pokémon ha ido cambiando desde su creación hasta día de hoy. Es normal, dada la longevidad de la saga, y la cantidad de distintos diseñadores y desarrolladores que trabajan en la empresa. Este cambio se puede apreciar claramente con el paso de las generaciones Pokémon.

El término de generación se usa dentro de este contexto para representar el momento en el que aparecen nuevas especies de Pokémon. Esta aparición de nuevos Pokémon es representada a través de un juego nuevo, aunque no todos los juegos conllevan una aparición de nuevos Pokémon, y por tanto, no se consideran nuevas generaciones.

A día de hoy existen 8 generaciones. Cada generación supone cambios en diseño, aparición de nuevas modas en el diseño y cambios generales en el mismo. Algunos de estos cambios fueron gracias a las mejoras en el *hardware* donde se estaba desarrollando. Por ejemplo, en las primeras generaciones se fue añadiendo progresivamente más color a los diseños porque las consolas permitían el uso de más color. De la misma forma, según

se han ido sucediendo las generaciones, los diseños han ido apareciendo un mayor grado de detalle en los diseños, pues los dispositivos permitían representar ese detalle.

Para visualizar este cambio en los diseños, en la Figura 3.5 se muestra la evolución final de cada Pokémon inicial de cada generación. Se puede apreciar una progresión tanto en el aumento de detalles como en el uso de más colores.



Figura 3.5. Última evolución de cada Pokémon inicial de todas las generaciones, empezando por la primera (arriba) y acabando en la octava (abajo)

4. EXPERIMENTACIÓN Y RESULTADOS

En este apartado se explican los diferentes acercamientos y cómo se han implementado las distintas técnicas planteadas para resolver el problema concreto de generar Pokémon nuevos.

4.1. DCGAN sobre *sprites*

El primer acercamiento que se plantea es crear un modelo de DCGAN para generar directamente imágenes de Pokémon nuevos.

4.1.1. DCGAN básico

El acercamiento básico consiste en una pareja simple de Generador y Discriminador. Tanto la arquitectura del entrenamiento como el proceso de entrenamiento está muy influenciado por los tutoriales que ofrece la documentación de la librería *Tensorflow*[30]. La arquitectura usada se puede ver la Figura 4.1.

Cada uno de los bloques del Generador están compuestos por una capa de *convolución transpuesta*, seguida por una capa de *batch normalization* y por último una capa que aplica la función *leakyReLU* a la salida.

Los bloques usados en el Discriminador son iguales que el generador, pero con capas *convolucionales* convencionales y sin capas de *batch normalization*.

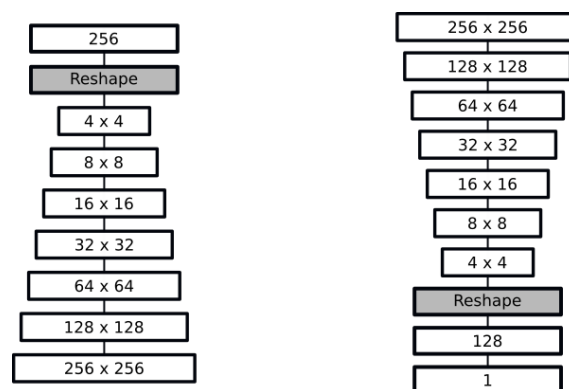


Figura 4.1. Arquitectura del Generador (izquierda) y Discriminador (derecha) para la generación de *sprites*

Este primer acercamiento se plantea como un caso base, y como punto de partida. Los resultados obtenidos, lejos de ser ideales, muestran ciertas formas claras, aunque los colores no tienen una separación clara, y aparte de la silueta, no se reconoce ninguna

parte del cuerpo clara. Los bordes no terminan de definirse del todo. En general, es un primer acercamiento simple, que muestra resultados esperados, y todavía está lejos de poder llegar a engañar al ojo humano. En la Figura 4.2 se pueden apreciar algunos de los Pokémon generados por este acercamiento.

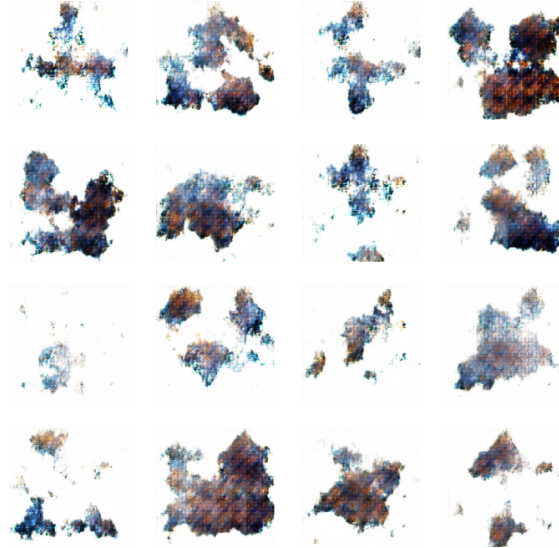


Figura 4.2. Resultados obtenidos con el DCGAN básico

Uno de los principales motivos por el que el modelo no genera mejores resultados se debe a que el Discriminador realiza un aprendizaje muy rápido, y el Generador no puede seguirle el ritmo, y se bloquea en resultados malos, y no es capaz de mejorar.

Es importante remarcar que el entrenamiento ha sido muy reducido, con 1000 ciclos de entrenamiento, que ha tardado alrededor de 1 hora. Es probable que si se dejara un tiempo de entrenamiento mayor, los resultados mejorarían. Con el objetivo de comparar bajo las mismas condiciones los diferentes acercamientos se van a mantener estas condiciones para todas las pruebas realizadas.

4.1.2. DCGAN con *Differentiable Augmentation*

La primera mejora que se plantea es utilizar *Differentiable Augmentation* para mejorar el rendimiento del DCGAN. *Differentiable Augmentation* es un tipo de *Data Augmentation* que se puede aplicar sobre DCGAN. Dado que es una técnica que se aplica sobre los DCGAN para mejorar los resultados generados, no es necesario modificar la arquitectura, y se sigue usando la descrita en la Figura 4.1.

Con el objetivo de reducir el tiempo de entrenamiento, se entrena con tan solo 160 imágenes por ciclo, que son elegidas aleatoriamente en cada ciclo. El entrenamiento también se realiza con 1000 ciclos, y el tiempo de entrenamiento es ligeramente superior a una hora y media. Se puede ver que tan solo implementando el *Differentiable Augmentation* ya supone un incremento notable en el tiempo de entrenamiento.

Los resultados obtenidos también son notablemente mejores. Se puede apreciar una mayor diferenciación de bordes, que separan distintas partes de la imagen. Los colores también son más consistentes en la imagen, aunque sigue realizando mezclas inconexas de los mismos. El modelo también ha aprendido a seguir unos patrones recurrentes, que se pueden apreciar en distintas imágenes, e incluso dentro de una misma imagen. Esto no es un buen comportamiento, ya que este tipo de patrones no se encuentra en las imágenes de entrenamiento, y se nota que es muy artificial y poco orgánico. En la Figura 4.3 se pueden apreciar los resultados generados por este acercamiento.

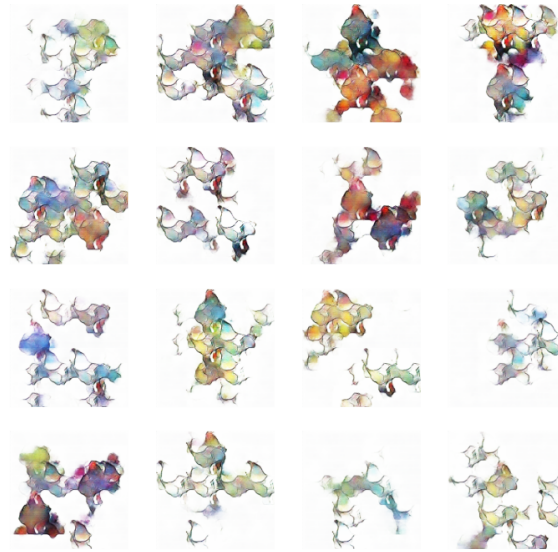


Figura 4.3. Resultados obtenidos con el DCGAN básico utilizando *Differentiable Augmentation*

A pesar de las notables mejoras respecto al modelo anterior, todavía se encuentra lejos de ser capaz de engañar a un humano a la hora de determinar si una imagen es real o generada. Para solventar este problema sería imperativo dejar al modelo más ciclos de entrenamiento. Aun así, se van a probar nuevas modificaciones sobre este modelo para intentar encontrar soluciones que generen mejores resultados, antes de realizar un entrenamiento largo y costoso.

4.1.3. DCGAN con *Differentiable Augmentation* y *Dropout*

Como los resultados que genera el modelo usando la técnica de *Differentiable Augmentation* mejoran los resultados originales, se decide continuar con esta técnica. En este caso se decide añadir capas de *Dropout* a los modelos, para reducir el sobreajuste de los modelos.

Este *Dropout* se aplica tanto en el Generador como en el Discriminador. En este caso este acercamiento genera unos resultados objetivamente peores incluso que el caso base. Esto puede deberse a diferentes factores, principalmente a que los modelos no consigan suficiente información en cada una de sus capas de forma que los resultados son subópti-

mos.

Este entrenamiento se realiza en las mismas condiciones que el anterior. Se utilizan 160 imágenes por ciclo durante 1000 ciclos. El tiempo que lleva este entrenamiento está cerca de las 2 horas.

Los resultados obtenidos pierden la diferenciación de los bordes, mezcla colores de forma errónea, y además repite constantemente un patrón en todas las imágenes que genera. En general, es un acercamiento no satisfactorio, y por tanto se descarta usar esta técnica en futuros entrenamientos. Los resultados generados con esta configuración se pueden apreciar en la Figura 4.4.

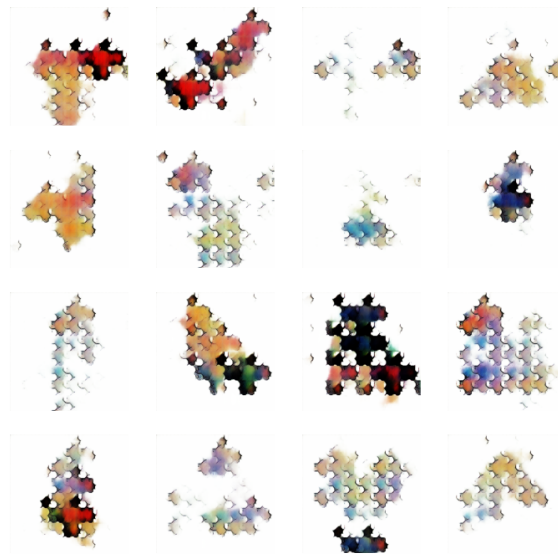


Figura 4.4. Resultados obtenidos con el DCGAN básico utilizando *Differentiable Augmentation* y *Dropout*

4.1.4. DCGAN con *Differentiable Augmentation* y *Autoencoders*

En este acercamiento se plantea usar *Autoencoders* para inicializar los pesos de las redes con el objetivo de que las capas del modelo posean cierto conocimiento previo sobre el problema, y después, tras el entrenamiento, puedan generar mejores resultados.

Estos *Autoencoders* utilizan un número muy reducido de ciclos, pues tampoco se desea que las redes sobreentrenen demasiado. Los resultados de este preentrenamiento se pueden apreciar en la Figura 4.5. Estos resultados no son perfectos, pero son suficientes como para que se puedan utilizar. Las capas de codificación del *Autoencoder* se transforman en el Discriminador, y las capas de decodificación en el Generador.

Los resultados obtenidos con esta técnica son los mejores obtenidos hasta el momento. Aun así, no son suficientemente buenos. El uso de colores está más definido, aunque sigue mezclándolos cuando no debe. Los bordes están más definidos y delimitados, pero

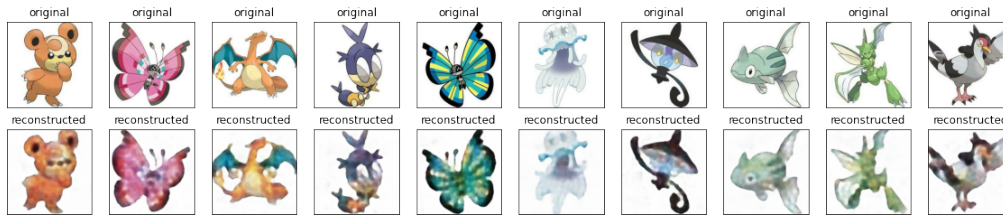


Figura 4.5. Resultados del entrenamiento previo usando *Autoencoders*

las formas no son capaces de imitar partes del cuerpo de forma diferenciada o partes reconocibles. Es un buen avance, y una mejora de los resultados, pero no es suficientemente bueno. Los resultados se pueden apreciar en la Figura 4.6.



Figura 4.6. Resultados obtenidos con el DCGAN básico utilizando *Differentiable Augmentation* e inicialización de pesos con *Autoencoders*

Al igual que los dos entrenamientos anteriores, este se realiza sobre 160 imágenes que se eligen aleatoriamente en cada ciclo durante 1000 ciclos. El proceso entero, incluyendo el preentrenamiento lleva alrededor de 1 hora y media.

4.1.5. Comparativa de modelos

En este apartado se procede a realizar una comparación de forma visual de los modelos que funcionan a través de una DCGAN sobre los *sprites*.

Realizando una inspección rápida y visual a los resultados mostrados en la Figura 4.7 se puede afirmar que el mejor de estos resultados es el modelo que hace uso de *Differentiable Augmentation* y *Autoencoders*. Por ese motivo se realiza un entrenamiento más intensivo usando esta arquitectura concreta. Los resultados pueden apreciarse en la Figura 4.8. Este entrenamiento hace uso de todas las imágenes durante los 1000 ciclos del entrenamiento.



Figura 4.7. Resultados de las distintas arquitecturas que hacen uso de una DCGAN

Este modelo, aunque mejora respecto al anterior, sigue son ser suficientemente bueno como para poder engañar al ojo humano. Aunque el uso de colores ha mejorado, y los bordes están un poco más definidos, sigue sin ser posible reconocer partes del cuerpo concretas o estructuras que tengan sentido orgánico.

Se podría realizar un entrenamiento más largo, pero por el momento se va a dejar así, para centrarse en otros acercamientos.

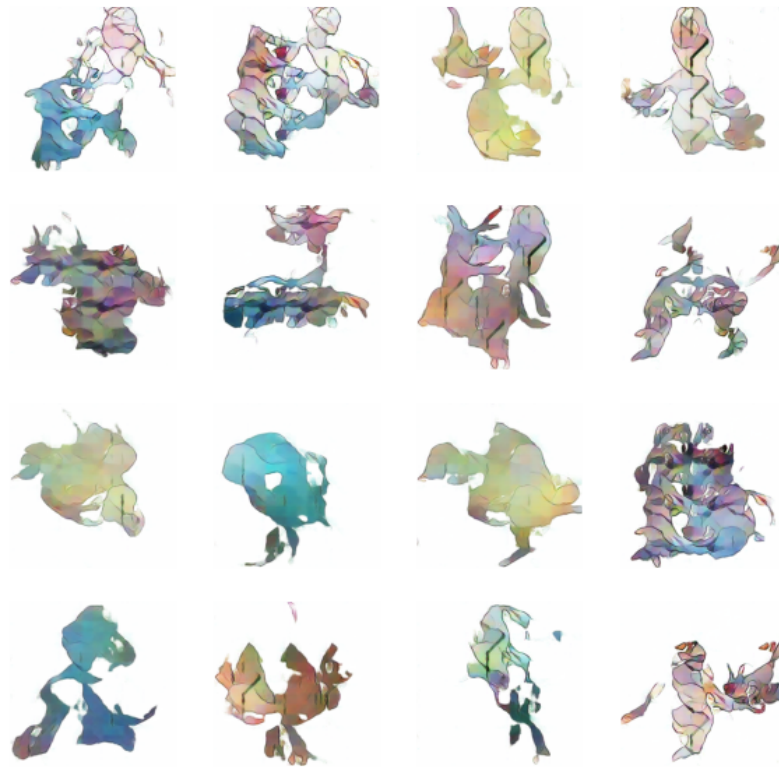


Figura 4.8. DCGAN con *Differentiable Augmentation* y preentrenamiento usando *Autoencoders* con un tiempo de entrenamiento más largo

4.2. DCGAN sobre iconos

Este apartado forma parte de un segundo acercamiento que consiste en generar imágenes reducidas o de baja resolución, y luego transformarlos en imágenes de alta resolución. Esta primera parte consiste en generar imágenes de baja resolución.

La arquitectura que se plantea en este apartado es similar a la usada en el DCGAN sobre los *Sprites*, pero con un menor número de capas, dado que los iconos son de menor tamaño. La Figura 4.9 muestra esta arquitectura reducida.



Figura 4.9. Arquitectura del Generador (izquierda) y Discriminador (derecha) para la generación de iconos

4.2.1. DCGAN con *Differentiable Augmentation*

Dado que se ha demostrado previamente que el uso del *Differentiable Augmentation* genera mejores resultados sobre los DCGAN, en este caso se utiliza directamente. Los resultados son, en proporción, bastante mejores que los resultados en alta resolución, lo cual tiene sentido, ya que los modelos tienen que aprender a simular imágenes de una resolución mucho menor.

Los bordes están mucho mejor definidos. Los colores son más consistentes y las formas, aunque no consiguen definir partes reconocibles, se podría considerar un buen avance, siempre y cuando sea posible realizar la transformación de baja a alta resolución. Los resultados se pueden observar en la Figura 4.10.

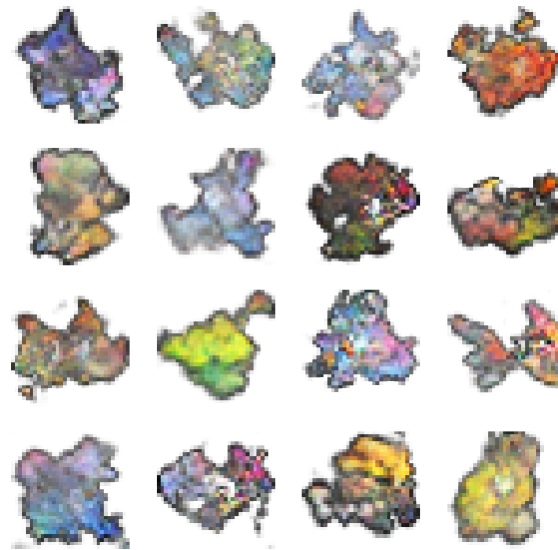


Figura 4.10. Resultados obtenidos con DCGAN y *Differentiable Augmentation*

Este entrenamiento se realiza sobre todas las imágenes de entrenamiento por ciclo, durante 1000 ciclos, y este entrenamiento tarda alrededor de 2 horas.

4.2.2. DCGAN con *Differentiable Augmentation* y *Autoencoders*

Siguiendo el mismo pensamiento que en el apartado anterior, dado que el uso de *Autoencoders* supone una mejora sobre los modelos de alta resolución. Por tanto se aplica nuevamente. Los resultados no son claramente mejores, y para determinar si es mejor aplicar los *Autoencoders* será necesario aplicar métricas más objetivas para determinar si es mejor o no, y no usar exclusivamente la exploración visual.

El entrenamiento, teniendo en cuenta la inicialización de pesos, tarda nuevamente alrededor de 2 horas.



Figura 4.11. Resultados obtenidos con DCGAN, *Differentiable Augmentation* e inicialización de pesos con *Autoencoders*

4.3. Pix2Pix de iconos a *sprites*

Una vez se tienen los iconos creados, es necesario crear una red que sea capaz de transformar estos iconos en *sprites*. El primer acercamiento que se plantea para conseguir este objetivo es una arquitectura Pix2Pix.

Se plantea una arquitectura básica de Pix2Pix, con un generador G que sigue la arquitectura U-Net y un Discriminador D con una arquitectura convolucional básica. En la Figura 4.12 se puede apreciar con más detalle esta arquitectura.

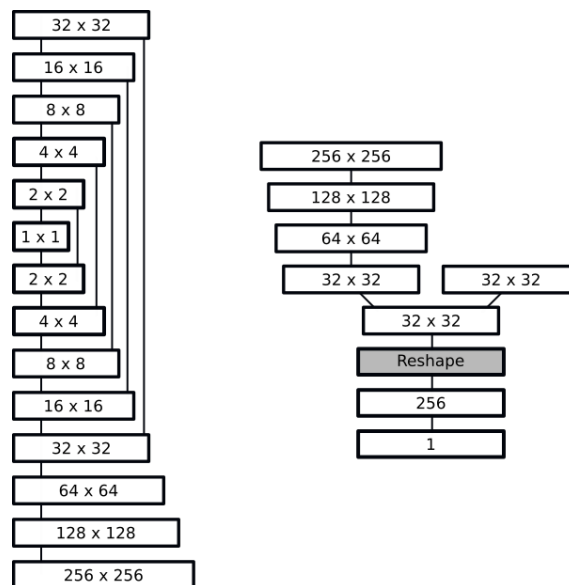


Figura 4.12. Arquitectura del Generador (izquierda) y Discriminador (derecha) para la transformación de iconos a *sprites*

Los resultados obtenidos con esta arquitectura son bastante deficientes. El problema principal por el cual se obtienen estos resultados tan malos es porque no se está pidiendo un pequeño cambio sobre la imagen original, sino que se intenta hacer una transformación completa de la imagen original para, partiendo de un icono, conseguir un *sprite*, que no solo tiene mayor resolución, sino que la mayoría de veces tiene una pose distinta, e incluso colores diferentes. Los resultados se pueden observar en la Figura 4.13, donde se puede apreciar que los resultados ni poseen los colores adecuados, ni la silueta ni la forma. Además se puede observar un patrón recurrente que genera la red, y no debería generar. En general, es una solución que a pesar de estar intentando recrear Pokémon que ya existen, tiene peor calidad que la solución basada en DCGAN exclusivamente.

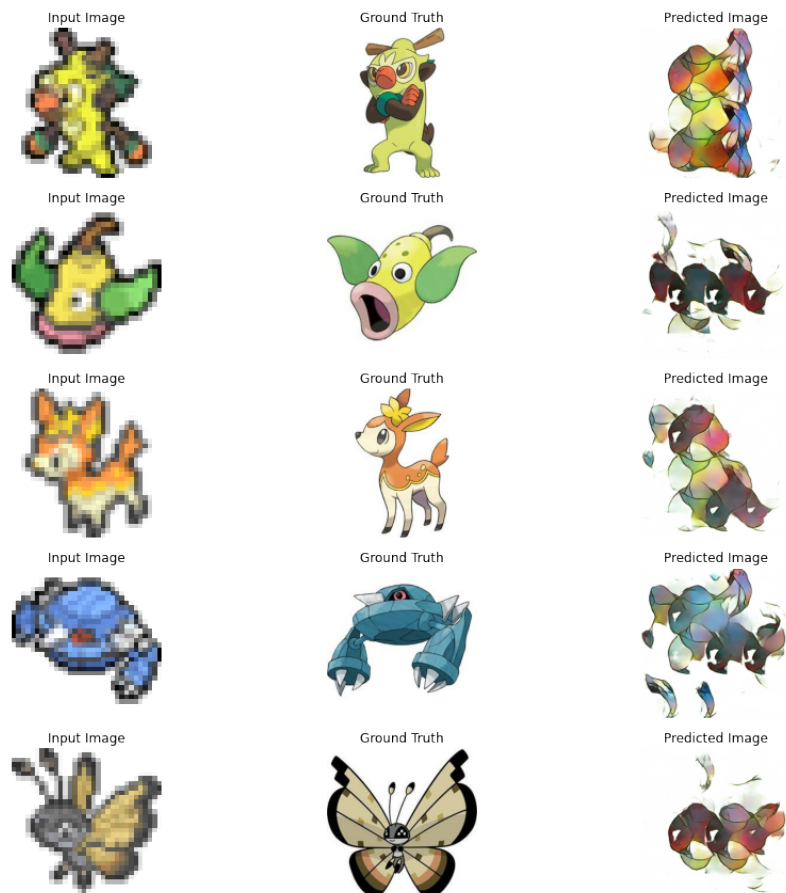


Figura 4.13. Resultados obtenidos con la arquitectura Pix2Pix

Para intentar arreglar estos fallos iniciales, se intenta aplicar un aumento de datos, pero los resultados no mejoran significativamente. Por tanto este acercamiento se considera un fracaso en términos de resultados, aunque si es útil a la hora de entender el comportamiento de las redes cuando se les plantea un problema complejo con una cantidad muy reducida de datos.

Las arquitecturas Pix2Pix, especialmente son utilizadas normalmente para transformar imágenes que tienen características en común entre ellas. Dado que este no es el caso entre las imágenes que se quiere tratar, puede que este acercamiento no sea el más acertado.

4.4. Autoencoder de iconos a *sprites*

Dado que el acercamiento usando un Pix2Pix ha fallado, se plantea realizar la misma con un acercamiento que no hace uso de redes adversariales, un *Autoencoder*. Se plantea una arquitectura básica con una función de coste por diferencia entre la imagen original y la recreada, pero los resultados vuelven a ser negativos. Estos resultados se pueden apreciar en la Figura 4.14, y se puede observar que no son capaces de generar formas definidas, ni bordes marcados, aunque los colores si están más o menos bien hechos.

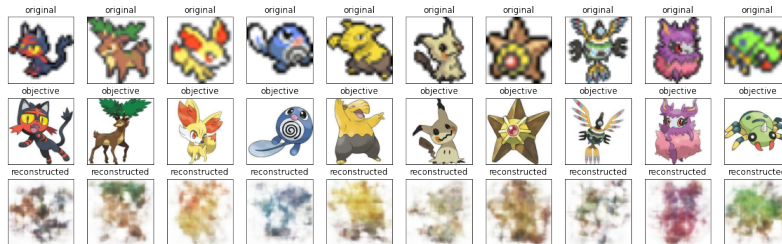


Figura 4.14. Resultados obtenidos con *Autoencoder* para transformar de icono a *sprite*

Este acercamiento, aparte de no ser adversarial, no tiene la potencia suficiente para realizar una transformación tan costosa como es transformar de icono a *sprite*.

Dado que los dos acercamientos planteados que intentan aumentar la calidad y detalle de las imágenes han fallado, se decide abandonar este acercamiento, y continuar con generar directamente la imagen en formato *sprite*.

4.5. AEGAN sobre *sprites*

Este acercamiento se basa en la arquitectura AEGAN, que mezcla *Autoencoders* con DCGAN[17].

Dada la complejidad de este acercamiento, se decide usar el código que ofrece el creador de la arquitectura, en lugar de intentar replicarlo. Esto va en detrimento de una personalización más exhaustiva del modelo, pero reduce mucho el tiempo de desarrollo y análisis.

De este acercamiento es necesario comentar su capacidad de reconstrucción de las imágenes originales, y su capacidad de generar imágenes nuevas.

Dado que este acercamiento no aporta demasiado al trabajo, por ser obra de otro autor, y no haberse realizado ninguna modificación grande, unido a los largos tiempos de entrenamiento que requiere esta arquitectura, la experimentación que se ha realizado no ha sido muy extensa, y siempre ha acabado, de una forma u otra, con el colapso del modelo. En la Figura 4.15 se pueden apreciar lado a lado la capacidad de reconstrucción del modelo junto a su capacidad de generación. Por un lado, este modelo realiza una recons-

trucción suficientemente buena, siendo capaz de representar la silueta y color principal de la imagen con bastante precisión, pero, por desgracia, el modelo ha colapsado a la hora de generar Pokémon nuevos y solo es capaz de generar 1.

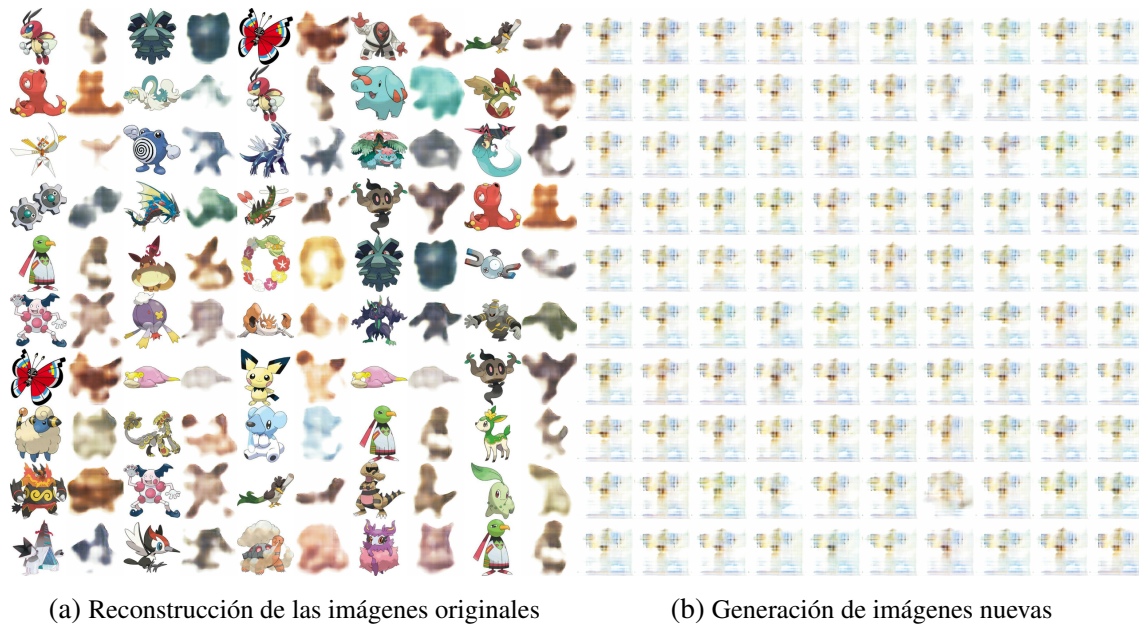


Figura 4.15. Resultados del AEGAN para la reconstrucción y generado de imágenes de Pokémon

Otra característica especialmente útil de la arquitectura AEGAN es la capacidad de realizar interpolaciones no solo sobre las imágenes generadas, sino sobre el conjunto de imágenes originales. Dado que el modelo ha colapsado en la generación de imágenes, no aporta demasiado mostrar su interpolado, ya que todas las imágenes serían iguales. Pero si es interesante mostrar el interpolado de la reconstrucción de imágenes. En la Figura 4.16 se puede ver la interpolación, y los resultados son bastante buenos. Realiza la interpolación correctamente, dentro de la reconstrucción que realiza.

El entrenamiento que se ha realizado con esta arquitectura ha sido reducido. Los parámetros que se usaron en el entrenamiento son: 1000 ciclos, 16384 para la dimensión del vector latente y 32 de tamaño el *batch*.

4.6. Planteamientos descartados

En este apartado del trabajo se presentan los planteamientos que se han descartado por distintos motivos, con el objetivo de demostrar el esfuerzo y dedicación realizado así como enseñar algunos acercamientos que por cualquier motivo no se han podido llevar a cabo y pueden ser de utilidad a quién quiera continuar este trabajo.

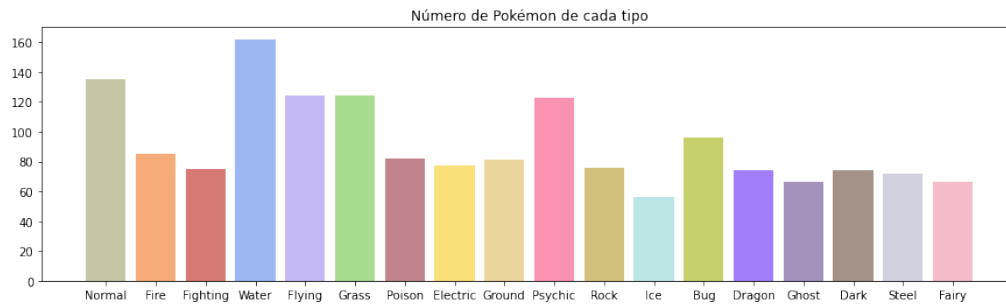


Figura 4.17. Distribución del número de Pokémon por tipo

rea similar, con conclusiones parecidas a las obtenidas. Los resultados, aunque no son ideales, dado que se está realizando una clasificación multiclase con 18 clases distintas, obtener resultados por encima del 20 % de acierto se consideraría una clasificación suficientemente buena, aunque muy lejos de valores deseados, más si clasificar Pokémon no es el objetivo final, sino un paso más a la hora de generarlos. En la Figura 4.18 se puede apreciar como los resultados obtenidos no siempre superan el 20 %.

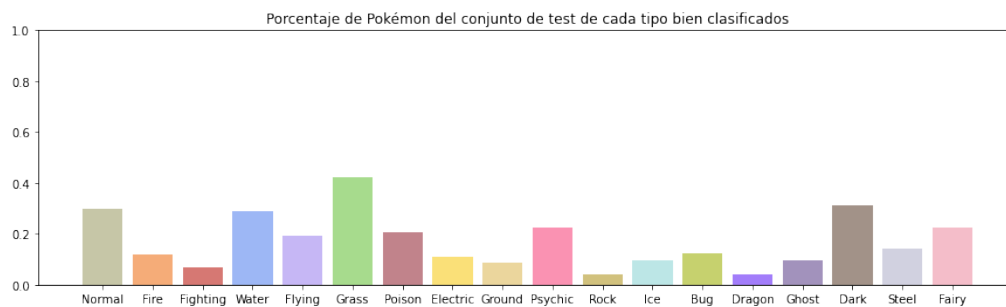


Figura 4.18. Porcentaje de Pokémon de cada clase correctamente clasificados según su tipo

Debido a estos resultados se descarta usar el tipo del Pokémon como información adicional a la hora de crear Pokémon nuevos, ya que no aporta suficiente información, y se considera que no haría que los modelos fueran más coherentes.

Los acercamientos que se plantearon que requerían del uso del tipo del Pokémon eran variados, y se explican a continuación con el propósito de que, si alguien tiene deseo de expandir este trabajo, pueda tenerlos en cuenta a la hora de desarrollar una mejor solución:

- **Generar Pokémon con información adicional:** Este es el acercamiento más sencillo y consiste en otorgar al generador una entrada compuesta por un vector latente generado aleatoriamente y un tipo elegido aleatoriamente, y el discriminador no solo debería determinar si es una imagen real o generada, sino que debería determinar el tipo del Pokémon. Lo que se buscaría es que el generador cree Pokémon realistas, y además acordes a un tipo concreto.

Una arquitectura que se podría plantear es un generador que recibe el vector latente y el tipo del Pokémon que debe generar, un discriminador que determine si la

imagen es real o generada, y un clasificador que determine el tipo del Pokémon. La función de coste del generador tendría que tener en cuenta si engaña al discriminador y si coincide en tipos con el clasificador. En la Figura 4.19 se puede observar un esquema de esta arquitectura.

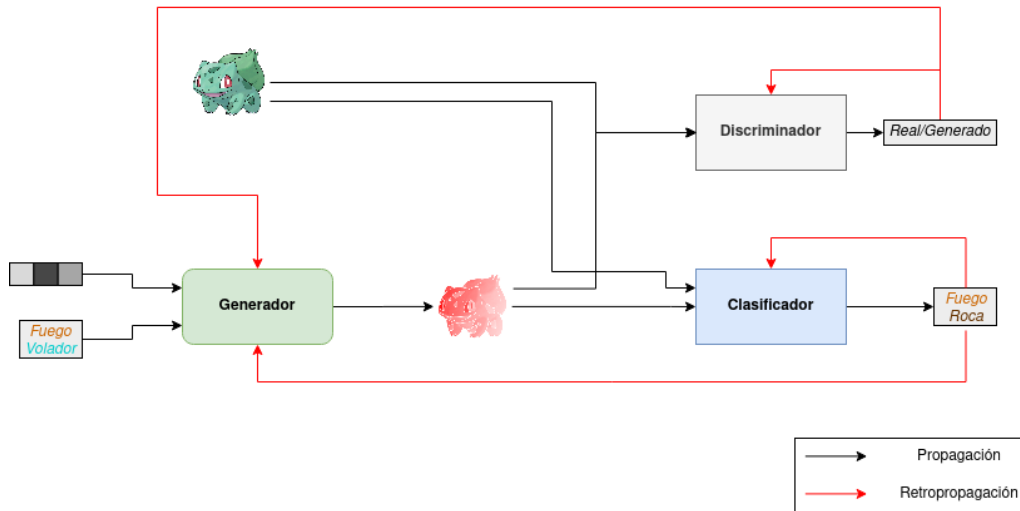


Figura 4.19. Esquema de la arquitectura propuesta para generar Pokémon con información adicional

- **Generar Pokémon en blanco y negro y darle color:** Otra propuesta que podría ser de utilidad es generar Pokémon en escala de grises, pasarle por un clasificador de tipos, y colorear el Pokémon en función del tipo clasificado. En esta arquitectura forman parte tres modelos diferentes. Un modelo GAN, que crea Pokémon en escala de grises, un clasificador que determina el tipo del Pokémon, y una arquitectura Pix2Pix que coloree el Pokémon en función del tipo. En la Figura 4.20 se puede observar un esquema de esta arquitectura.

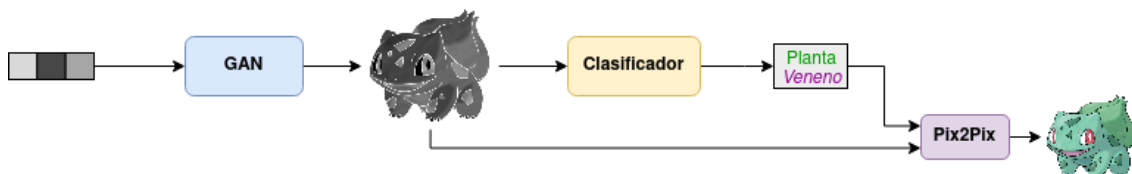


Figura 4.20. Esquema de la arquitectura propuesta para generar Pokémon generando primero Pokémon en escala de grises, y posteriormente dotarles de color

Otra forma de aportar información adicional a los modelos es informando de la generación a la que pertenecen. La justificación de este acercamiento nace de la evolución de los diseños a lo largo de la saga, que ha ido progresando de diseños más sencillos y con menos colores a diseños con gran cantidad de detalle y numerosos colores. Por tanto se plantea realizar el mismo proceso intentando clasificar los Pokémon por generación para evaluar si realmente es información útil para los modelos. Sin embargo, a pesar de que se

obtienen mejores resultados en la clasificación, no son lo suficientemente buenos como para justificar el desarrollo y evaluación de un modelo que use esta información, debido al reducido tiempo que se tiene para el desarrollo de este trabajo. De todas formas, este acercamiento podría llegar a ser de interés, y se plantea como trabajo a futuro para buscar una mejora sobre los modelos actuales. En la Figura 4.21 se aprecia el porcentaje de Pokémon correctamente clasificados del conjunto de test al crear una red que clasifique los Pokémon por generación.

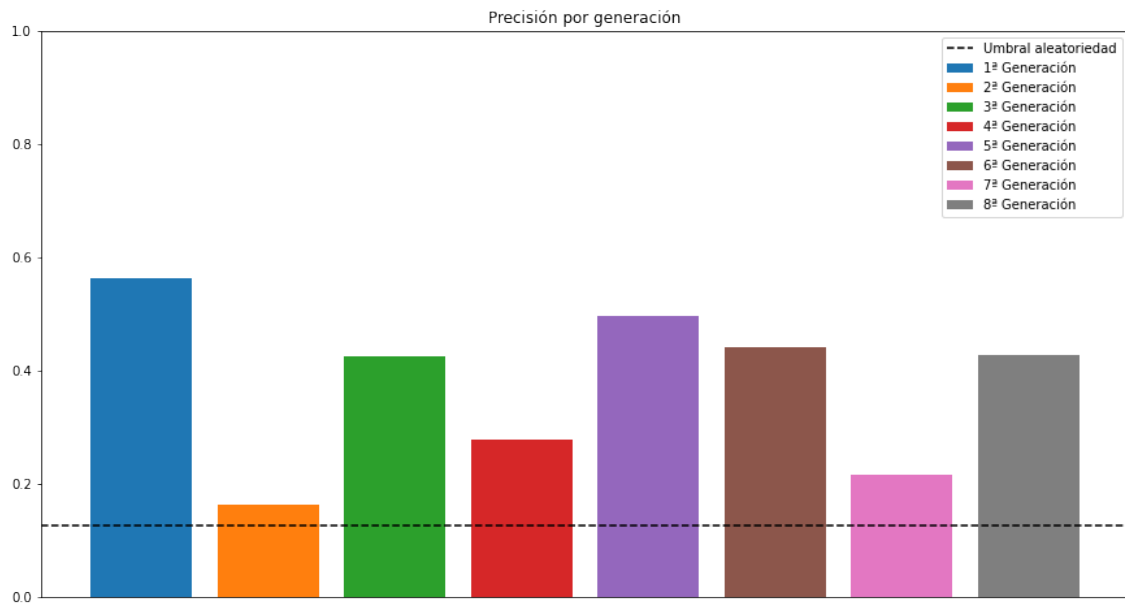


Figura 4.21. Porcentaje de Pokémon correctamente clasificados según su generación

4.7. Técnicas Avanzadas

Por falta de tiempo y conocimientos del autor, hay una serie de técnicas que se podrían haber resuelto con mayor eficacia el problema planteado, pero que no ha sido posible, y se plantean como trabajos futuros para mejorar el modelo actual.

4.7.1. AEGAN

Como se ha explicado en el apartado 4.5 AEGAN sobre *sprites*, en este trabajo se ha utilizado la implementación original, sin realizar ninguna modificación. Esto se debe a la complejidad de este acercamiento, que cuenta con 4 redes interconectadas, con sus correspondientes funciones de coste, y que sigue un entrenamiento muy concreto.

El no haber desarrollado la arquitectura supone que no ha podido optimizarse para los parámetros deseados. Las imágenes han tenido que ser escaladas al tamaño que acepta el modelo, y no se han podido incorporar técnicas que se han demostrado útiles para este problema concreto como es el *Differentiable Augmentation*.

Sería interesante desarrollar esta arquitectura para trabajos futuros, dado que el propio autor de la arquitectura ha demostrado que es útil para resolver este problema en su artículo “*I Generated Thousands of New Pokemon using AI*”[6], aunque es importante remarcar que el conjunto de imágenes que utiliza como ejemplo no son las mismas que se utilizan en este trabajo (son de menor resolución).

4.7.2. VAE

Esta técnica no ha sido aplicada por distintos motivos. El primero es que no es capaz de imágenes nuevas, solo puede realizar interpolaciones entre las imágenes originales, y el segundo es que no es considerado un acercamiento adversarial, y por tanto no entra dentro de los objetivos de este trabajo.

Sin embargo es necesario destacar que este tipo de acercamientos, a pesar de que no pueda generar imágenes nuevas, si podría crear Pokémon nunca vistos, que sean una mezcla o fusión de Pokémon ya existentes. Al igual que el acercamiento anterior, sería interesante plantearlo para trabajos futuros.

4.7.3. Progressive Growing

El motivo por el cual no se aplica esta técnica, que es muy probable que sea capaz de mejorar los resultados, y es aplicable a la mayoría de acercamientos propuestos en este trabajo, se debe a que, a pesar de que según los autores supone una reducción en el tiempo de entrenamiento[21], supone un coste de memoria mucho más elevado, y no se pudo llevar a cabo con las máquinas disponibles.

Sin duda habría sido de gran interés para el trabajo haber podido utilizar este método de entrenamiento, y se propone que se utilicen en trabajos futuros para demostrar su utilidad.

Esta técnica es utilizada en una de las redes más populares y que mejores resultados genera actualmente: *StyleGAN*[32], que es capaz gracias a esta técnica de aprender en distintos puntos de entrenamiento el estilo de lo que se desea generar. *StyleGAN* utiliza como punto de partida *Progressive Growing*, pero lo lleva un paso más allá, permitiendo generar con más precisión y detalle las imágenes que desea generar.

Al igual que *Progressive Growing*, sería interesante utilizar la arquitectura *StyleGAN* para la generación de Pokémon.

5. EVALUACIÓN

La evaluación de modelos generativos no es un proceso trivial. Esto se debe a que no existe una métrica objetiva que determine la eficacia con la que un modelo generativo es capaz de crear imágenes nuevas. Es una consecuencia de que la mayoría de modelos adversariales pertenecen al aprendizaje no supervisado, y su función de coste no representa cómo de bueno es el modelo, sino cómo de cerca está de converger. En la mayoría de acercamientos, la función de coste de la red Generador determina en qué medida ha sido capaz de engañar a la red Discriminador, mientras que la función de coste de la red Discriminador indica cuál es su capacidad de clasificar correctamente muestras como reales o generadas.

Existen una serie de métricas que se utilizan con frecuencia a la hora de evaluar este tipo de arquitecturas. En este caso se plantea usar el FID como medida más pseudoobjetiva. Adicionalmente, se plantea una encuesta informal a distintos usuarios.

5.1. FID

El FID es una de las técnicas más extendidas a la hora de evaluar redes generativas. Esta técnica hace uso de una red preentrenada denominada *InceptionV3*[24]. Los pesos de la red han sido entrenados con el *dataset ImageNet*[33], y sobre 1000 clases distintas. Esto se hace con el objetivo de simular una interpretación de la percepción visual humana.

El FID se calcula sobre dos conjuntos de imágenes, y lo que determina es la distancia que existe entre los valores estadísticos de media y varianza en los vectores latentes obtenidos en la penúltima capa de la red *InceptionV3*. Dado que lo que se mide es una distancia entre dos conjuntos de imágenes, cuanto menor es esta diferencia, más parecidos son los dos conjuntos de imágenes.

Sin embargo, al estar trabajando con imágenes que representan Pokémon, algo que no ha visto en ningún momento la red, y además tienen formas extrañas y gran variabilidad en las formas y colores, y no siempre representan cosas identificables de forma sencilla, puede resultar en que esta forma de evaluar los modelos no sea perfecta y tenga ciertos fallos. De todas formas, funciona suficientemente bien como para que se pueda utilizar para realizar una puntuación sobre los distintos modelos generativos planteados. Además, dado que todos los modelos están evaluados con la misma métrica, todos están en igualdad de condiciones.

Primero es necesario calcular cuál es el valor objetivo al que tienen que aspirar los modelos. Para ello se calcula el valor del FID entre dos grupos de imágenes distintas pertenecientes al conjunto de datos original. Como se puede apreciar en la Tabla 5.1, el

valor obtenido sobre dos conjuntos de *sprites* es de **5,4**. Por tanto, cuanto más cercano el valor del FID del resto de modelos a este número, mejor será el modelo, supuestamente. La idea sería encontrar un modelo con un valor tan bajo como el obtenido al comparar los dos conjuntos de datos. Si se consiguiera este hipotético modelo, este sería capaz de generar imágenes de Pokémon de una calidad muy alta.

En la Tabla 5.1 se muestran los valores de FID obtenidos con cada modelo planteado. Tanto el que consta de Pix2Pix como el que está formado por un *Autoencoder*, dado que toman como entrada un icono, se evalúan tanto con iconos que forman parte del conjunto original, como con iconos que han sido generados por otra red distinta.

Los valores obtenidos para cada modelo que genera *sprites* se pueden apreciar en la Tabla 5.1, o de forma más visual, en la Figura 5.1. Se puede ver que los peores resultados son los basados en *Autoencoders*, con valores por encima de 100. Esto coincide con los valores encontrados con la inspección manual.

Después se encuentra tanto el acercamiento inicial, como el que hace uso de capas de *Dropout*, lo cual, nuevamente, coincide con los resultados encontrados con la exploración manual.

Por último, y con mejores resultados de FID, se encuentran los modelos de DCGAN con *Differentiable Augmentation*, y su modificación que hace uso de *Autoencoders*, lo cual, coincide con lo esperado. Lo que sí es sorprendente son los resultados obtenidos con el acercamiento Pix2Pix, que habían sido descartados en la fase de desarrollo por generar imágenes que a primera vista parecían peores que los acercamientos ya mostrados.

Tabla 5.1. FID de los modelos que generan *sprites*

Nombre del modelo	FID
<i>Caso base entre imágenes originales</i>	5,400
<i>DCGAN básico</i>	94,238
<i>DCGAN con Differentiable Augmentation</i>	57,988
<i>DCGAN con Differentiable Augmentation y Dropout</i>	82,327
<i>DCGAN con Differentiable Augmentation y Autoencoders</i>	47,032
<i>Pix2Pix con iconos originales</i>	43,680
<i>Pix2Pix con iconos generados</i>	39,373
<i>Autoencoder con iconos originales</i>	107,187
<i>Autoencoder con iconos generados</i>	108,090

Por tanto, dado que se encuentra una diferencia entre la exploración de los resultados por parte del autor y los valores obtenidos usando el FID. Con el objetivo de decidir qué modelo genera mejores resultados, se hará uso de la encuesta que se realiza a distintos usuarios.

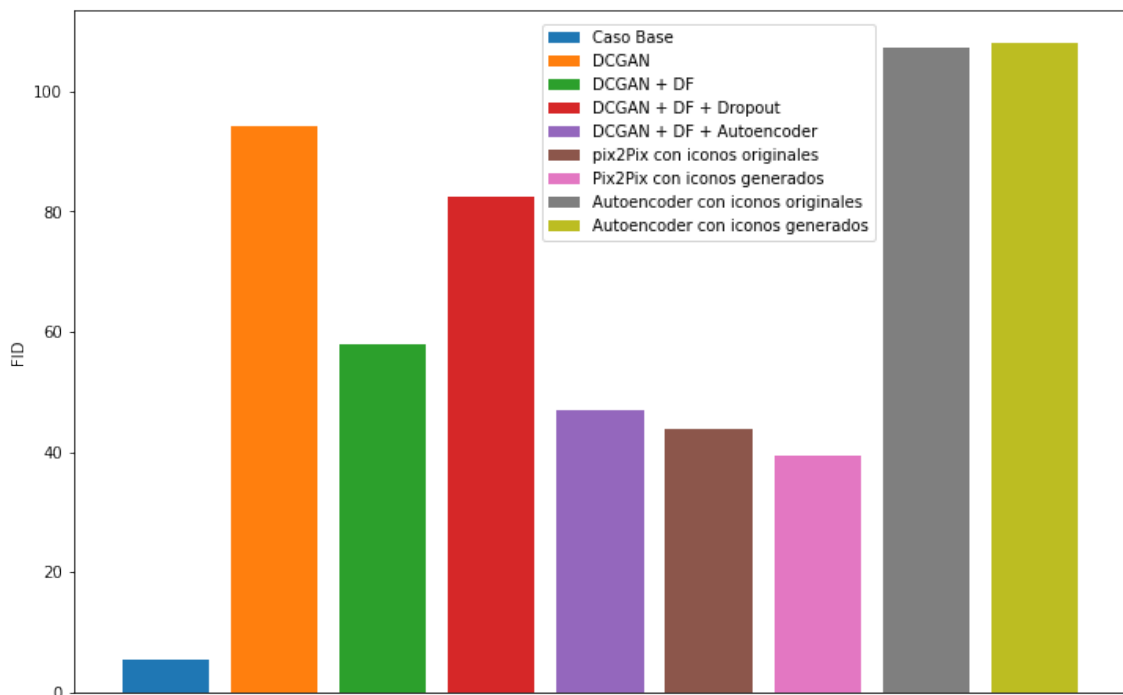


Figura 5.1. FID de los modelos que generan *sprites* representados como gráfico de barras

Respecto a los modelos que generan iconos, solo se tienen dos modelos distintos, que al observar los resultados, no se supo determinar cuál era mejor. En la Tabla 5.2 se muestran los FID de estos dos modelos, y muestran que el mejor acercamiento es el que hace uso de *Autoencoders*, pero por una diferencia mínima. De todas formas, antes de concluir que este modelo es mejor, se usará la encuesta para confirmar este valor.

Tabla 5.2. FID de los modelos que generan iconos

Nombre del modelo	FID
<i>Caso base entre imágenes originales</i>	0,522
<i>DCGAN con Differentiable Augmentation y Autoencoders</i>	1,284
<i>DCGAN con Differentiable Augmentation</i>	1,299

5.2. Encuesta informal

Esta encuesta se utiliza como medida adicional y se espera que coincida con los resultados obtenidos usando el FID. En lugar de pedir a los usuarios que valoren a través de un sistema de puntuación arbitrario, o utilicen un porcentaje de parecido respecto a los valores reales, se les pide que ordenen en función de cuánto se parecen respecto al conjunto de imágenes originales.

Esta decisión respecto a la estructura de la encuesta se realiza debido a que los seres humanos son relativamente malos puntuando objetiva y arbitrariamente una serie de

muestras, pero tienen una gran capacidad para realizar ordenaciones en una serie de elementos.

La encuesta está formada por 3 preguntas, y está disponible para su visionado en <https://forms.gle/WjHbzahCcoLYeCfT6> (Ya no acepta más respuestas).

1. *¿Qué nivel de familiaridad tienes con los Pokémon?*

Esta pregunta se realiza con la excusa de que un usuario solo es capaz de determinar qué se parece más a un Pokémon, si tiene al menos nociones básicas de qué es un Pokémon.

2. *Ordena los modelos según se parezcan más a los Pokémon reales, siendo la columna 1 lo más parecido y la columna 8 lo menos.*

Antes de hacer esta pregunta a los usuarios, se les muestra un conjunto de imágenes de referencia seguido por un conjunto de imágenes generadas por cada uno de los modelos.

La respuesta es una matriz cuadrada de selección múltiple donde solo se acepta una solución única por columna y por fila.

3. *Indica cuál de los 2 modelos se parece más al modelo original.*

Esta pregunta es similar a la anterior, pero en este caso se pregunta sobre los modelos que generan iconos.

Igual que en la pregunta previa, se muestran primero, imágenes del conjunto de datos original, y después un conjunto de imágenes por cada modelo.

Dado que en este caso solo hay dos modelos, la pregunta consiste en decidir cuál se parece más en una pregunta de elección múltiple con 2 opciones.

Se consiguen un total de **67** respuestas. A continuación se muestra los resultados de cada una de las preguntas.

1. Familiaridad con los Pokémon

Tal y como se puede ver en la Figura 5.2, casi la totalidad de los encuestados tienen cierto conocimiento sobre los diseños de la saga Pokémon. A pesar de que haya ciertos usuarios que no están del todo familiarizados con los diseños, en principio no se plantea descartarlos. En caso de que haya algún resultado dudoso, el desempate se hará en función de su nivel de familiaridad con la saga.

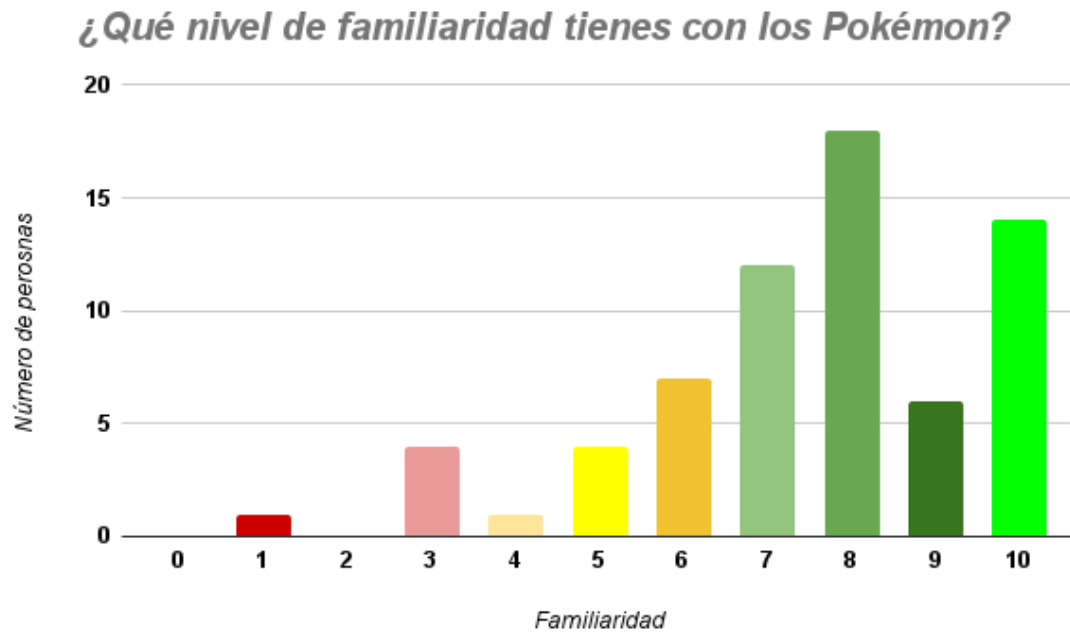
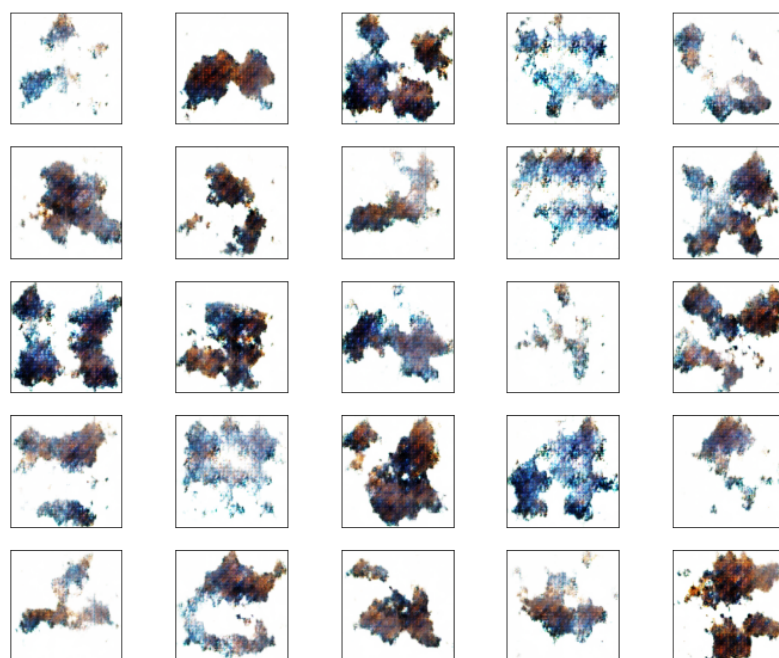


Figura 5.2. Respuestas de la pregunta 1 de la encuesta.

2. Ordenación de los modelos

En esta pregunta, se van a analizar las respuestas de cada modelo de forma independiente, y luego en conjunto, para decidir qué modelo es el que mejor resultados genera, utilizando la información de las encuestas junto al cálculo del FID.



Modelo 1 (DCGAN)

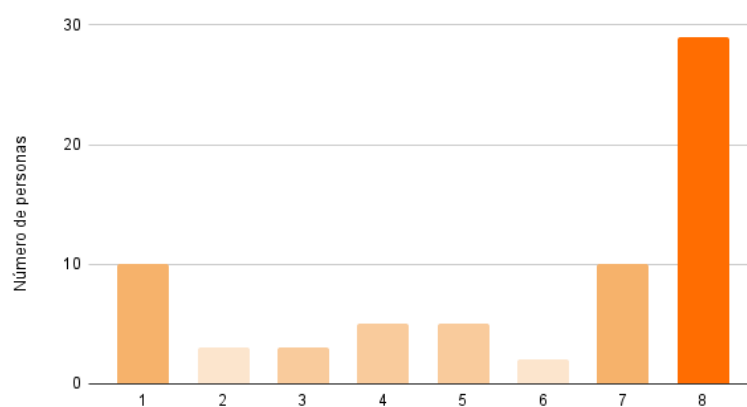
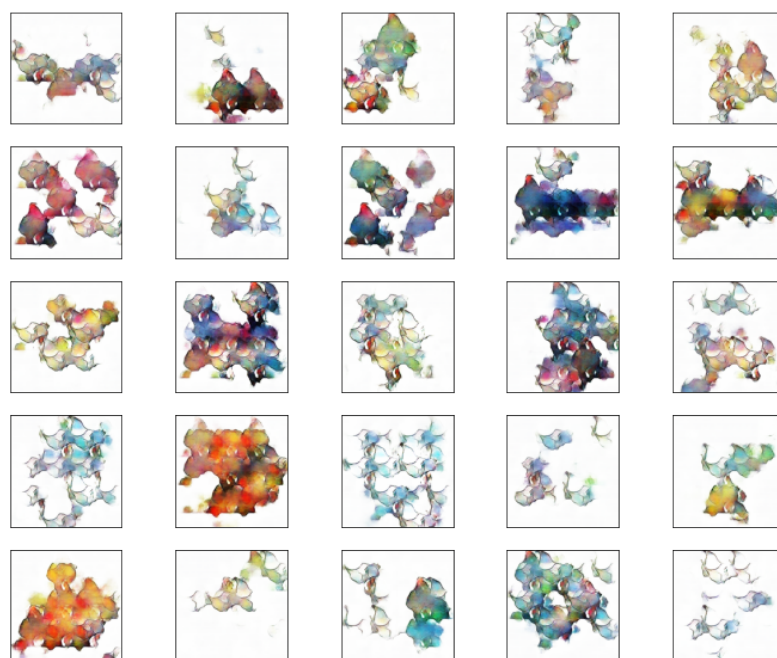


Figura 5.3. Imágenes del modelo 1 (*DCGAN*) y las respuestas de los usuarios

Las imágenes presentadas a los usuarios para el primero modelo (*DCGAN*) se pueden apreciar en la Figura 5.3, junto a los resultados de los usuarios. Los usuarios sitúan este modelo como el peor de todos, con una diferencia bastante marcada. Su FID es de 94,238, así que tiene sentido que lo sitúen como uno de los peores, pero no el peor tal y como muestra la encuesta. De todas formas, esto confirma que este modelo no es suficientemente bueno, y por tanto se puede descartar como solución óptima.



Modelo 2 (DCGAN + DF)

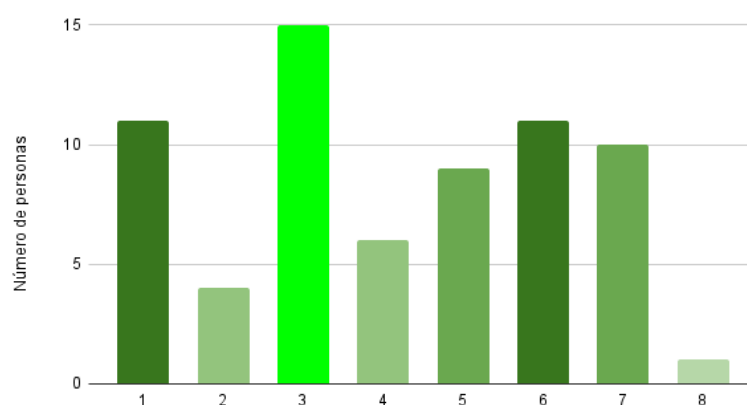
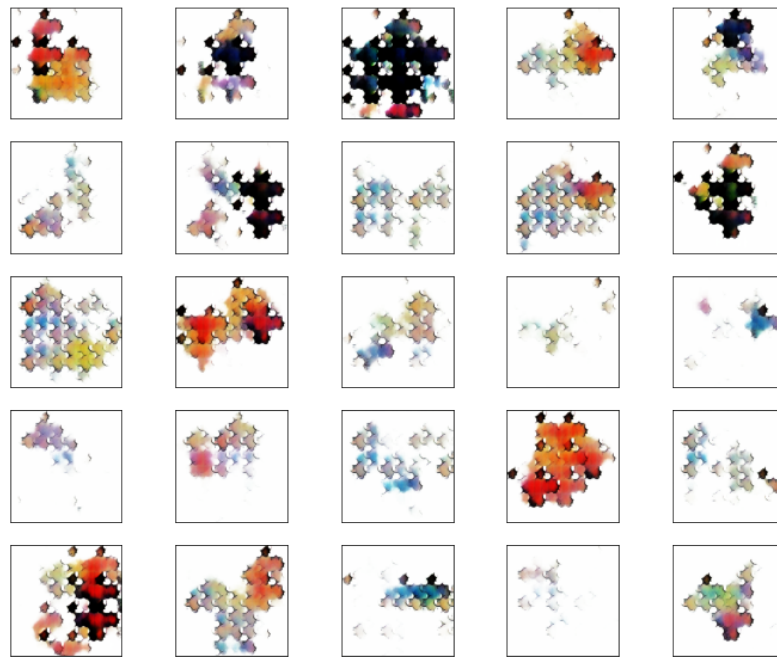


Figura 5.4. Imágenes del modelo 2 (*DCGAN + DF*) y las respuestas de los usuarios

Se pueden observar las imágenes mostradas a los usuarios del segundo modelo (*DCGAN + DF*), así como las respuestas obtenidas en la Figura 5.4. En este caso los resultados no son tan claros. El FID obtenido del modelo es de 57.988, bastante mejor que el modelo anterior. Esto se refleja en que las votaciones lo sitúan como el tercer mejor modelo, aunque también tiene una gran cantidad de votos para la primera posición, o la sexta. Asimismo hay muchos votos que lo sitúan como quinto o séptimo, por tanto, a pesar de que tenga muchos votos para ser el tercero también hay muchos que lo sitúan bastante peor.



Modelo 3 (DCGAN +DF + Autoencoder)

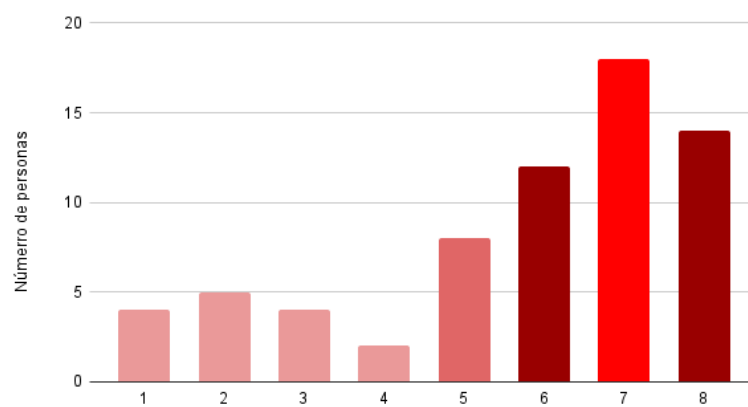
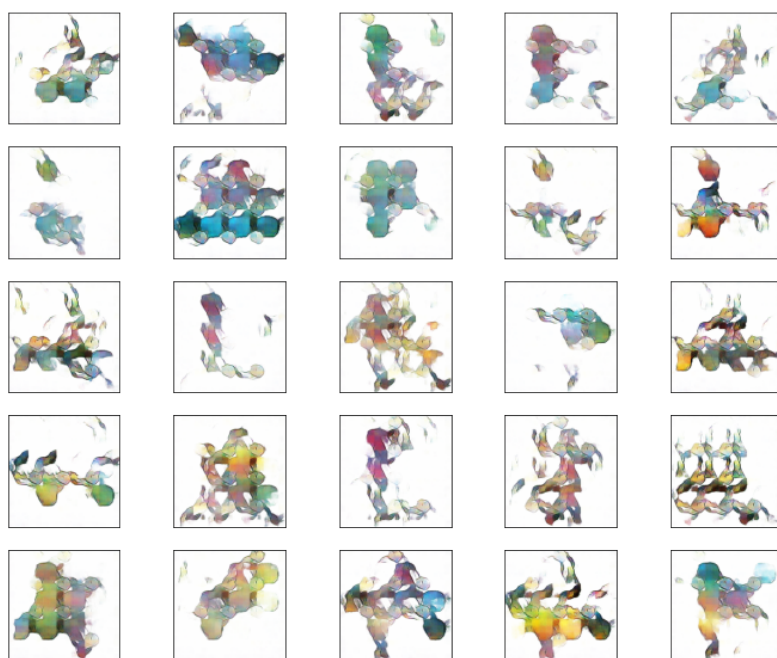


Figura 5.5. Imágenes del modelo 3 (*DCGAN + DF + Dropout*) y las respuestas de los usuarios

Los resultados respecto al tercer modelo (*DCGAN + DF + Dropout*), que se pueden apreciar en la Figura 5.5, junto a las imágenes correspondientes, son los esperados. Su FID es de 82,327, y la encuesta lo sitúa en los últimos puestos, en el séptimo puesto, pero con votos tanto como octavo como sexto.



Modelo 4 (DCGAN + DF + Autoencoders)

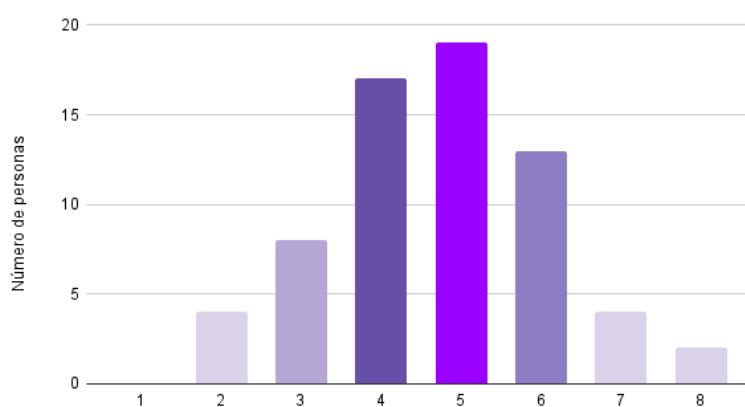


Figura 5.6. Imágenes del modelo 4 (*DCGAN + DF + Autoencoder*) y las respuestas de los usuarios

Los resultados obtenidos para el modelo 4 (*DCGAN + DF + Autoencoder*) son un poco inesperados. Este modelo es considerado por el autor como el que mejor resultados genera, y su FID es de 47,032, pero la encuesta lo sitúa como el quinto mejor modelo, aunque con bastantes votos también entre el cuarto y el sexto. Por tanto, a pesar de que este modelo parecía que era uno de los mejores, en la práctica no alcanza los valores deseados.

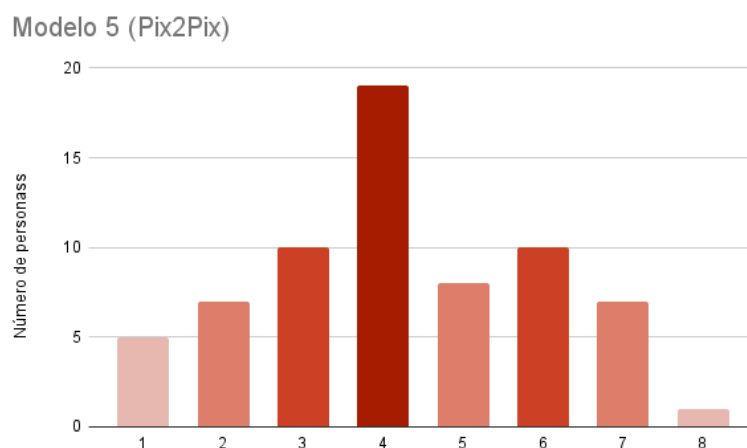
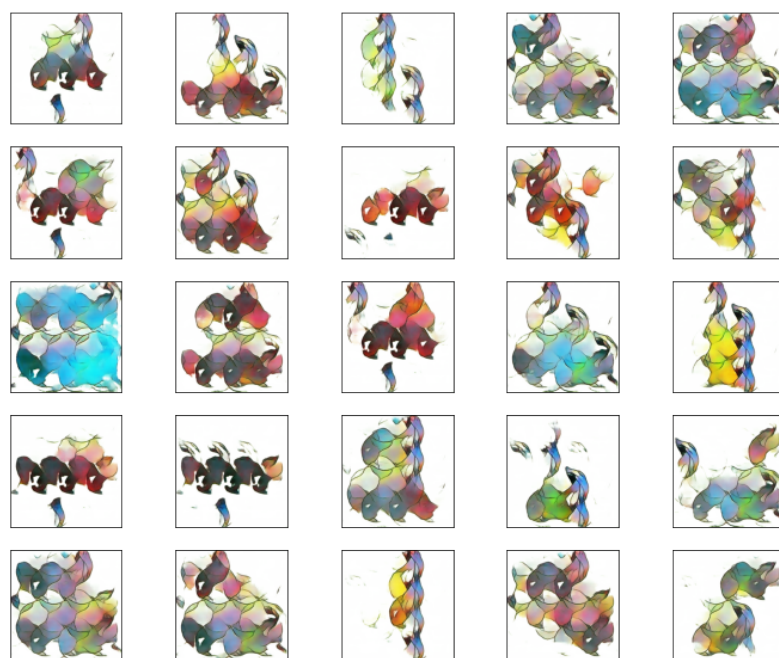
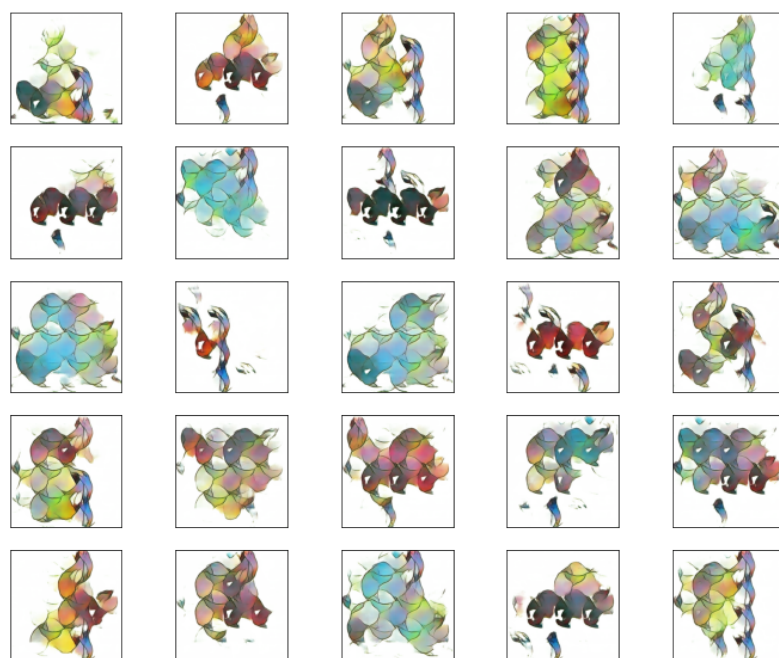


Figura 5.7. Imágenes del modelo 5 (*Pix2Pix*) y las respuestas de los usuarios

Los resultados del quinto modelo (*Pix2Pix*) lo sitúan mejor que al modelo anterior, dándole a este modelo la cuarta posición, tal y como se puede apreciar en la Figura 5.7. Esto supone el desempate que se buscaba entre el modelo 4 y el modelo 5, proclamando a este último como vencedor. Lo curioso es que ninguno de los dos (modelo 4 y 5) es interpretado por los usuarios como el más parecido a las imágenes originales, difiriendo por tanto con el autor. Este análisis por parte de los usuarios coincide con el cálculo del FID, que para este modelo es de 43,680 (menor que el modelo anterior, que tiene 47,032).



Modelo 6 (Pix2Pix-Gen)

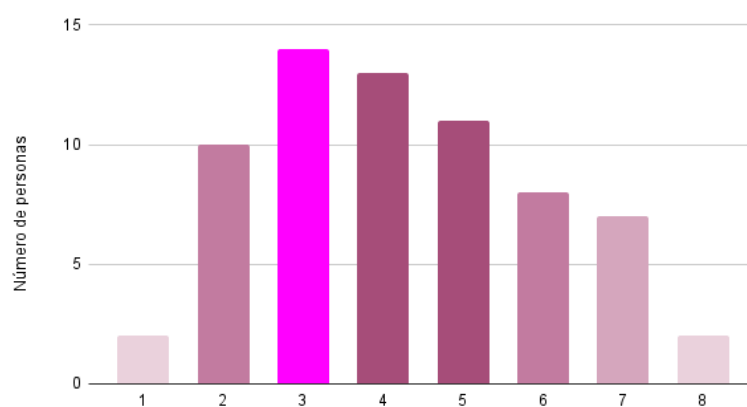


Figura 5.8. Imágenes del modelo 6 (*Pix2Pix-Gen*) y las respuestas de los usuarios

En este caso, al sexto modelo (*Pix2Pix-Gen*) lo sitúa la encuesta, como se puede ver en la Figura 5.8, en la tercera posición, por encima del modelo 5 (*Pix2Pix*), coincidiendo nuevamente con lo obtenido en los cálculos del FID, cuyo valor en este modelo es de 39,373, reafirmando que este modelo es mejor que los modelos basados en DCGAN.

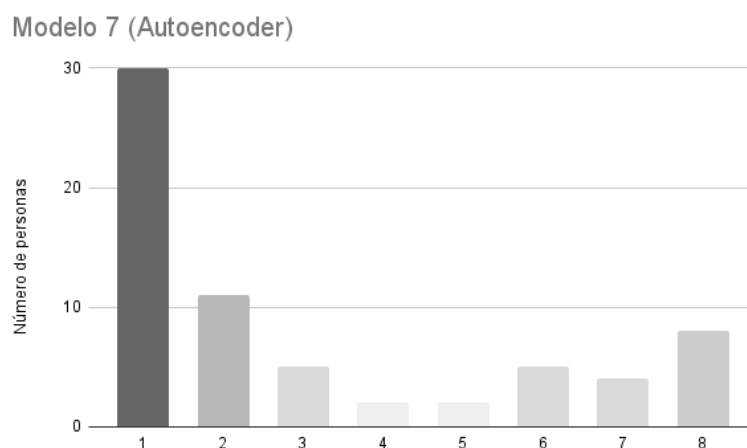
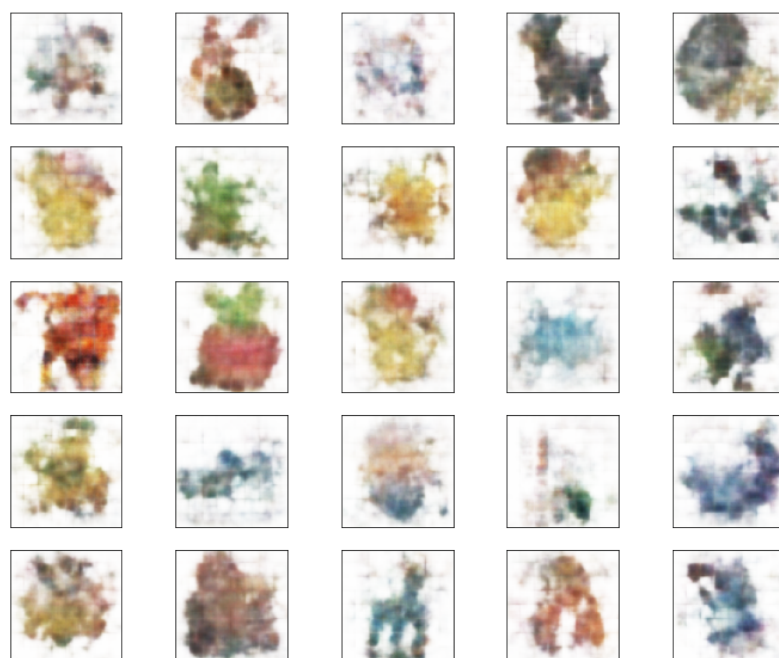


Figura 5.9. Imágenes del modelo 7 (*Autoencoder*) y las respuestas de los usuarios

Por último se encuentran los modelos basados en *Autoencoders*. En la Figura 5.9 se puede apreciar que en las encuestas se pone el séptimo modelo (*Autoencoder*) como el mejor, a pesar de que tanto en la exploración visual como en el cálculo del FID, que es 107,187, sitúan estos modelos como los peores. Esto se debe a cómo está formulada la pregunta. En la Figura 5.9 se puede ver que los resultados mostrados a los usuarios son muy parecidos a Pokémon originales, pero con una distorsión sobre los mismos, y dado que a los usuarios se les pide que reconozcan cuál se parece más, optan por este modelo.

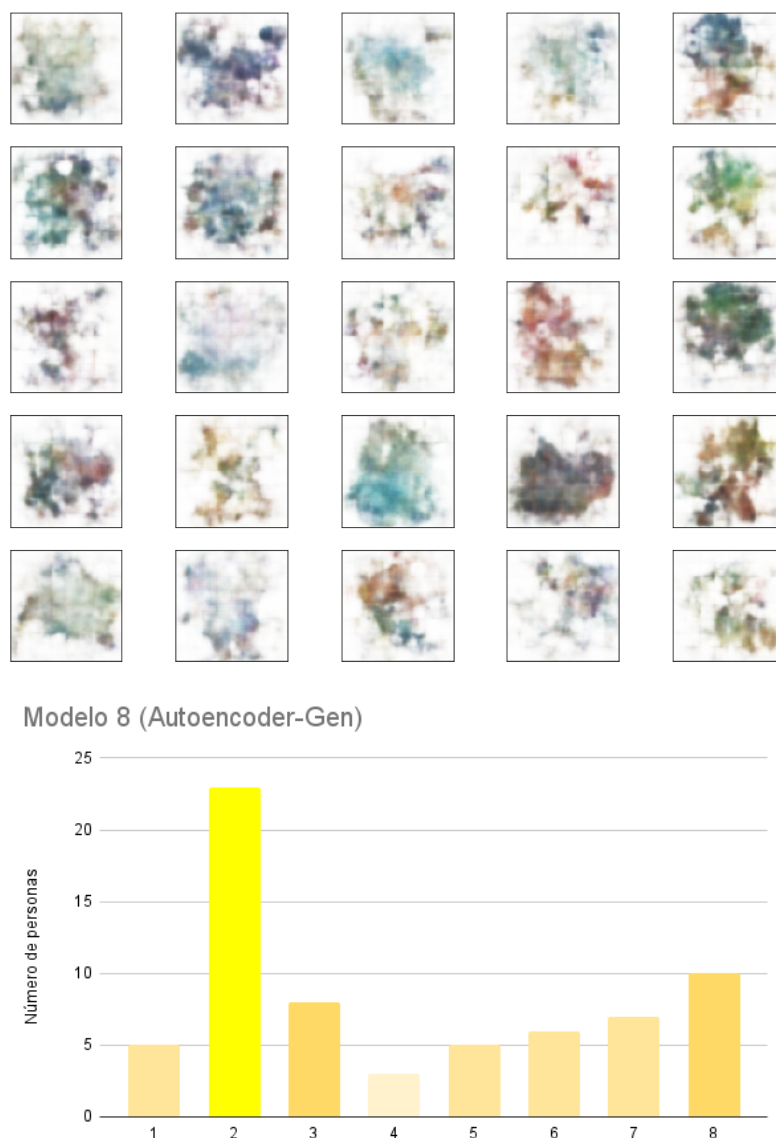


Figura 5.10. Imágenes del modelo 8 (*Autoencoder-Gen*) y las respuestas de los usuarios

El octavo y último modelo (*Autoencoder-Gen*) le pasa lo mismo que al modelo anterior. A pesar de que su FID es 108,090, y en la exploración inicial, se descarta por generar imágenes de una calidad muy baja, los resultados de la encuesta, que se pueden apreciar junto a las imágenes a las que tenían acceso los usuarios en la Figura 5.10, lo ponen como el segundo mejor modelo. La explicación que se encuentra para entender esta disparidad es la misma que para el caso anterior. Los resultados son parecidos a los Pokémon originales, pero difuminados en cierta medida. De todas formas, al observar estos resultados en la encuesta se replantean la utilidad de este tipo de modelos para solucionar este problema concreto.

3. Decisión del mejor modelo sobre iconos

En este caso, se pide decidir qué modelo es más parecido, y la decisión, como se puede observar en la Figura 5.11, se decanta en gran medida por el modelo 1

(DCGAN + *Autoencoder*), coincidiendo nuevamente con los cálculos del FID, que indican que el modelo 1 es ligeramente mejor que el 2.

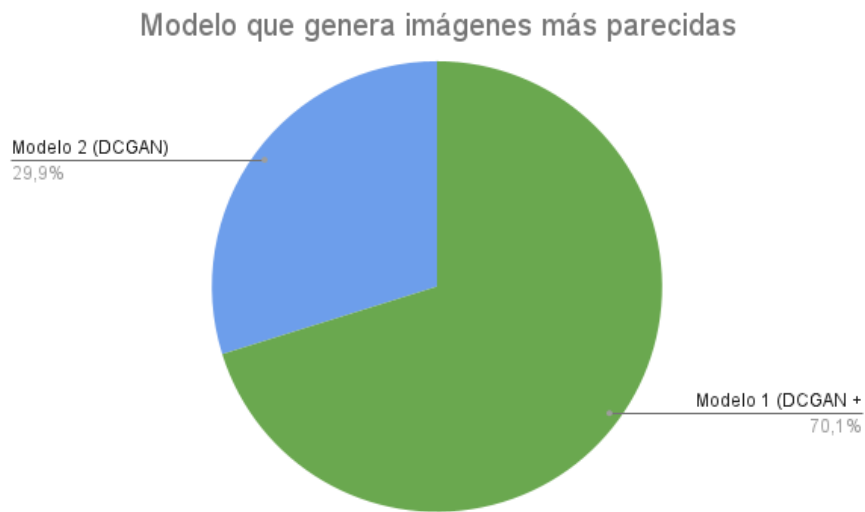


Figura 5.11. Resultados de la encuesta respecto a los modelos de iconos

Las imágenes que se muestran a los usuarios de estos modelos son las mostradas en las Figuras 5.12 y 5.13. Ambas imágenes son muy parecidas, lo cuál es sorprendente que las respuestas se hayan decantado en tal medida sobre uno de los dos modelos, lo que reafirma los valores obtenidos con el FID, que es de 1,284 para el modelo 1, y de 1,299 para el modelo 2.



Figura 5.12. Imágenes del modelo 1 de iconos (*DCGAN + Autoencoder*)

Como conclusión de la evaluación, los modelos que crean iconos son mucho más acertados que los que crean *sprites* generando imágenes similares a las originales.

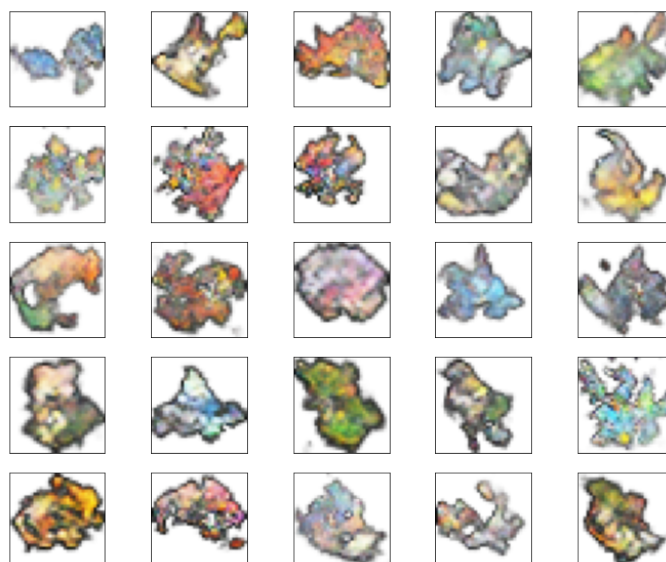


Figura 5.13. Imágenes del modelo 2 de iconos (*DCGAN*)

El mejor modelo para la generación de iconos está formado por el modelo 1, que está compuesto por un DCGAN, cuyos pesos han sido preentrenados haciendo uso de un *Autoencoder*. Estos modelos, aunque sí muestran bastante parecido a las imágenes originales, no cuentan con partes reconocibles, como podría ser piernas, brazos, ojos o bocas. A pesar de ello, tanto los colores como las formas son variadas y reconocibles, teniendo también bordes claramente definidos. A pesar de ello, su FID está todavía lejos del caso base, lo que indica que sigue habiendo cierto margen de mejora.

Respecto a los modelos que generan *sprites*, existe cierto conflicto. Según el FID, el mejor modelo es el 6, que consiste en un Pix2Pix que genera Pokémon a través de iconos generados por otros modelos. Si no se tienen en cuenta los acercamientos por *Autoencoders*, la encuesta coincide con este resultado. El autor opina que el mejor modelo es el 4, que se compone por un DCGAN con *Differentiable Augmentation* y preentrenamiento de los pesos con *Autoencoders*, pero, dado que tanto los valores obtenidos con el cálculo del FID como la encuesta coinciden en el modelo 6, es el que se trata como mejor modelo.

Este acercamiento, dado que fue descartado en la fase de desarrollo, no tiene un entrenamiento demasiado intensivo, ni se han intentado aplicar distintas técnicas sobre el mismo. Aun así, se presenta como modelo final a la hora de generar *sprites* de Pokémon.

Esta evaluación ha ayudado a confirmar una serie de ideas: el FID, a pesar de utilizar una red de neuronas artificiales que no ha sido entrenada con Pokémon, ha sido capaz de obtener resultados muy similares a los obtenidos por usuarios reales, demostrando así su eficacia y utilidad a la hora de evaluar modelos generativos. Además, ha ayudado a evitar un posible sesgo de confirmación por parte del autor, que de no haber realizado esta evaluación más extendida, habría elegido un modelo que genera peores resultados.

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1. Conclusiones Profesionales

En este apartado se planteará si los objetivos propuestos para el trabajo se han cumplido satisfactoriamente, o si por el contrario, no han podido cumplirse, y por qué motivo.

El objetivo principal de este trabajo es mostrar la utilidad que tienen los modelos adversariales en la industria del videojuego, que son capaces de generar una gran cantidad de imágenes de una calidad muy similar a las imágenes que toma como referencia. Su problema principal es que requieren de una gran cantidad de imágenes para conseguir resultados positivos.

Por tanto, otro objetivo de este trabajo era conseguir resultados positivos haciendo uso de un conjunto reducido y dispar de imágenes de referencia. En este caso, el conjunto de datos que se decide usar es el de los Pokémon.

El objetivo de generar Pokémon nuevos nunca vistos, pero que guarden parecido a los Pokémon originales no se ha conseguido. Incluso los mejores modelos de *sprites* son muy diferentes al conjunto de datos original, y tienen numerosas carencias. No tienen partes reconocibles, los colores se mezclan de forma incorrecta y, por lo general, no tienen la apariencia de un Pokémon. Esta es una de las consecuencias y mayores desventajas de las técnicas utilizadas. Dado que los modelos encargados de crear las imágenes no tienen acceso directo al conjunto de datos original, sino que tienen que inferir su conocimiento a través de otra red. Esto, aunque pueda parecer una desventaja, es lo que otorga la potencia a este tipo de acercamientos, pero requieren de un entrenamiento muy intensivo, y es muy fácil que el modelo no consiga converger.

En este trabajo se propone el uso de diferentes técnicas para mejorar este entrenamiento, y permitir que pueda hacerse con pocas imágenes. A pesar de ello, el objetivo no se ha cumplido completamente. Aunque el modelo de generación de *sprites* no haya conseguido muy buenos resultados, los modelos de generación de iconos sí que ha conseguido generar imágenes muy parecidas al conjunto de imágenes original. Esto se debe a varios factores. El primero y más sencillo es que los iconos son de mucha menos resolución que los *sprites*, y por tanto los modelos tienen que generar menos píxeles, lo que hace que tengan menos pesos que optimizar. El segundo, pero que también es muy importante, es que los iconos originales cuentan con una paleta de colores mucho más reducida, y no cuentan ni con sombras ni con poses complicadas que supongan una percepción de la profundidad.

Finalmente, el modelo que se propone como final consiste en dos modelos concatena-

dos. El modelo de iconos basado en DCGAN con un preentrenamiento de sus pesos con *Autoencoders*, seguido por una arquitectura Pix2Pix que transforma iconos en *sprites*. En la Figura 6.1 se puede apreciar gráficamente esta arquitectura.

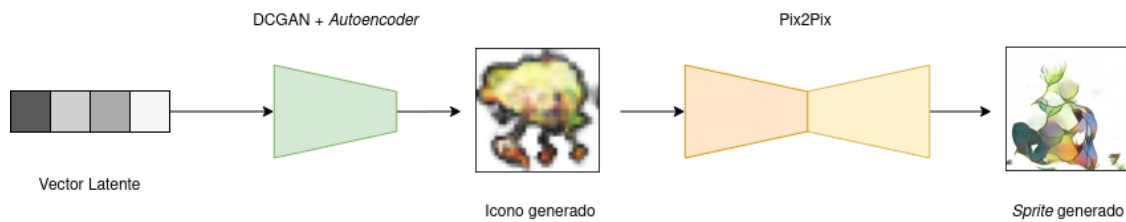


Figura 6.1. Arquitectura final propuesta

A pesar de que el objetivo principal de conseguir imágenes capaces de engañar al ojo humano no se haya conseguido, sí se ha realizado un progreso significativo desde el caso base al modelo final.

Los acercamientos adversariales son muy potentes, pero tienen una serie de problemas, la mayoría discutidos en este trabajo. El principal, y que se ha intentado paliar en este trabajo es la gran cantidad de imágenes para su entrenamiento, pero no es la única. Cada problema requiere un gran esfuerzo y especialización para solucionarse. No existe una red genérica que sea buena para todos los problemas. Esto hace que se tenga que invertir mucho tiempo y recursos para conseguir buenos resultados.

6.2. Conclusiones Personales

A nivel personal, este trabajo ha supuesto un gran esfuerzo de desarrollo e investigación de tecnologías que se pudieran aplicar al problema concreto, lo cual ha servido para expandir los conocimientos sobre algoritmos de redes de neuronas que se explican en la carrera de Ingeniería Informática. Además ha servido para desarrollar habilidades de análisis y lectura de *papers* técnicos de investigación.

A pesar de que los resultados finales del trabajo no sean demasiado vistosos, se cree que estas técnicas podrían aplicarse con resultados muy favorables, especialmente si se dispone de una potencia computacional alta, y acceso a un gran número de imágenes de referencia. Un ejemplo donde estas técnicas podrían ser muy útiles es en la generación de texturas para videojuegos, en la generación de distintos enemigos o la creación de *sprites* de objetos inertes.

6.3. Trabajos Futuros

Existen innumerables formas de continuar este trabajo. En primera instancia, sería necesario perfeccionar los resultados obtenidos. Para ello, tal y como se explica en la sección 4.6 Planteamientos descartados, se podrían aplicar distintas técnicas que se han intentado

en este trabajo, pero no ha sido posible llevar a cabo. También existen numerosas técnicas que podrían ser aplicables. Por ejemplo, se podría utilizar la arquitectura *StyleGAN*[32], que es una de las más interesantes actualmente y que, idealmente, permitiría resolver uno de los mayores problemas que tienen los resultados actuales, y es que no son capaces de crear partes concretas como pueden ser brazos o piernas, y tampoco son capaces de dotar a la imagen de detalles como podría ser ojos o boca.

Además, si esta técnica se mejora y perfecciona, de forma que genere Pokémon que puedan guardar similitud con los Pokémon reales, se podría plantear una aplicación web para que cualquier usuario interesado pudiera utilizar el modelo para generar distintos Pokémon.

Fuera del problema concreto de crear Pokémon, estas técnicas se pueden aplicar para la generación de imágenes 2D para videojuegos (y no solo para videojuegos). También se podría extrapolar a la generación de modelos tridimensionales, con un aumento de complejidad bastante elevado.

Concretamente, dentro de la industria del videojuego, estas técnicas se podrían aplicar sobre la generación de texturas. Las texturas utilizadas en videojuegos, especialmente si tienen una apariencia realista, son muy costosas de realizar. Es necesario tener el material que se quiere representar, usar una cámara concreta y fotografiarla con condiciones de luz óptimas, para no generar sombras no deseadas. Otra forma de conseguirlas es teniendo artistas capaces de crear estas texturas fotorealistas, pero requiere de tiempo y esfuerzo. Lo más común es comprar las texturas de páginas especializadas que cuentan con multitud de texturas. Este coste y esfuerzo se podría reducir si se hiciera uso de un modelo basado en GAN, encargado de generar estas texturas.

Otro posible uso de esta tecnología dentro de la industria del videojuego podría ser para generar *sprites* de enemigos para juegos de *RPG* por turnos. Estos juegos suelen tener una gran cantidad de enemigos, de diseño muy variado. Normalmente, los diseñadores son los encargados de crearlos. Por desgracia, la mayoría de veces no tienen tiempo suficiente para crear tantos como el juego necesita, y suelen hacerse uso de una técnica muy sencilla que consiste en usar el mismo diseño pero con colores distintos, para representar un enemigo diferente. Haciendo uso de una arquitectura Pix2Pix se podría entrenar para que, por ejemplo, creara enemigos con un diseño similar a la entrada recibida.

Y usando un acercamiento más creativo, se podrían diseñar animaciones o patrones de ataque, por ejemplo, usando una técnica de GAN cuyo objetivo sea generar código que se pueda transformar en estas animaciones o ataques, que sean parecidas a los ejemplos de entrenamiento.

Las posibilidades son prácticamente infinitas, y si se utilizan correctamente, estas técnicas podrían revolucionar la industria del videojuego, dotándola de mucho más contenido, con un coste mucho más reducido.

7. GESTIÓN DEL TRABAJO

7.1. Recursos utilizados

Para llevar a cabo este trabajo, se hace uso de una serie de recursos *hardware* y *software*. En esta sección se exponen estos recursos que han permitido desarrollar los modelos, llevar a cabo su experimentación, y su correspondiente evaluación.

7.1.1. Recursos *Hardware*

- Portátil *Lenovo V110 15ISK*
 - Memoria RAM: 3,6 GiB
 - Procesador: *Intel® Core™ i5-6200U CPU @ 2.30GHz × 4*
 - Tarjeta Gráfica: *Mesa Intel® HD Graphics 520 (SKL GT2)*
- Ordenador Sobremesa
 - Memoria RAM: 15,5 GiB
 - Procesador: *Intel® Core™ i5-7400 CPU @ 3.00GHz × 4*
 - Tarjeta Gráfica: *GeForce GTX 1060 6GB*
- Servidor Remoto
 - Memoria RAM: 29,8 GiB
 - Procesador: *Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz*
 - Tarjeta Gráfica: *2 x GeForce GTX 1080 8 GB*

7.1.2. Recursos *Software*

- Entorno de desarrollo: *Anaconda*
- Módulos: *matplotlib.pyplot, numpy, Tensorflow y Keras, scipy, pandas.*
- Aplicaciones: *Jupyter Notebook, Overleaf, Inkscape, Draw.io, Gimp.*
- Lenguajes de Programación: *Python3, Bash.*

7.2. Planificación

En este apartado se detallan los pasos que se han seguido para llevar a cabo el desarrollo de este trabajo, así como un diagrama de Gantt que resume y muestra de forma gráfica estos pasos.

7.2.1. Tarea 1: Obtención de Conocimientos

En este sub apartado se engloban las tareas de investigación sobre cualquier técnica o implementación necesaria para el desarrollo del trabajo. También se incluyen la realización de tutoriales o guías necesarias para ayudar a comprender o implementar cualquier tecnología necesaria. En la Tabla 7.1 se hace un desglose de estas tareas.

Tabla 7.1. Tarea 1: Obtención de Conocimientos

Tarea	Fecha Inicio	Fecha Final	Horas
Tutorial básico para aplicar GAN sobre Pokémon	01/09/20	06/09/20	5
Investigar sobre diferentes formas de generar imágenes	20/09/20	25/09/20	15
Crear un clasificador de Pokémon en función del tipo	26/09/20	06/10/20	8
Investigar medidas de evaluación de las GAN	12/10/20	18/10/20	10
Investigar técnicas de <i>subsampling</i> en GAN	18/01/21	20/01/21	3
Investigar <i>Progressive Growing</i>	08/02/21	09/02/21	4
Obtención de Conocimientos	01/09/20	09/02/21	45

7.2.2. Tarea 2: Recopilación de Datos

Dentro de este apartado se incluyen las tareas relacionadas con la obtención y etiquetado de las imágenes de Pokémon utilizadas en el proyecto. En la Tabla 7.1 se muestran las tareas que forman este paso.

7.2.3. Tarea 3: Desarrollo de las Redes Generativas Adversariales

Este paso consiste en realizar el desarrollo de los distintos modelos que se plantean en este trabajo. En la Tabla 7.3 se muestran todos los modelos que se intentan plantear.

Tabla 7.2. Tarea 2: Recopilación de Datos

Tarea	Fecha Inicio	Fecha Final	Horas
Obtención manual de los <i>sprites</i> de todos los Pokémon	10/09/20	25/09/20	18
Obtención manual de los iconos de todos los Pokémon	04/11/20	09/11/20	8
Creación de una biblioteca para el tratamiento de imágenes	12/09/20	18/10/20	7
Obtención automática de <i>sprites</i> e iconos de todos los Pokémon	21/12/20	27/12/20	8
Preprocesado de todas las imágenes	28/12/20	03/01/21	10
Recopilación de Datos	10/09/20	03/01/21	51

Tabla 7.3. Tarea 3: Desarrollo de las Redes Generativas Adversariales

Tarea	Fecha Inicio	Fecha Final	Horas
Creación del GAN inicial con las imágenes recopiladas	07/10/20	08/10/20	6
Creación del GAN con la técnica de <i>Differentiable Augmentation</i>	16/10/20	18/10/20	10
Creación del GAN con los Pokémon en blanco y negro	19/10/20	21/10/20	9
Creación del Pix2Pix para colorear imágenes	30/10/20	31/10/20	7
Creación del GAN con los iconos	20/11/20	21/11/20	5
Creación del Pix2Pix para convertir los iconos en <i>sprites</i>	16/11/20	26/11/20	9
Pruebas y experimentos con distintas arquitecturas y formas de GAN	18/01/21	28/01/21	12
Creación del GAN con <i>Progressive Growing</i>	09/02/21	12/02/21	12
Pruebas y experimentos con distintas arquitecturas y formas de GAN de iconos	22/01/21	27/01/21	11
Pruebas y experimentos con distintas arquitecturas y formas de Pix2Pix	10/02/21	17/02/21	10
Desarrollo de las Redes Generativas	07/10/20	17/02/21	91

7.2.4. Tarea 4: Comparativa de los Modelos

Dentro de este apartado se incluye la exploración de los datos, la evaluación de los modelos y la realización de una encuesta para usuarios que ayuda en la evaluación de los modelos. En la Tabla 7.4 se puede ver un despliegue de estas tareas.

Tabla 7.4. Tarea 4: Comparativa de los Modelos

Tarea	Fecha Inicio	Fecha Final	Horas
Exploración de los resultados obtenidos	20/02/21	21/02/21	6
Desarrollo del código del FID	17/01/21	20/01/21	7
Evaluación de los modelos usando el FID	08/06/21	09/06/21	5
Desarrollo del cuestionario de evaluación	09/06/21	09/06/21	5
Recopilación y exposición de los resultados del cuestionario	13/06/21	15/06/21	10
Comparativa de los Modelos	17/01/21	15/06/21	33

7.2.5. Tarea 5: Escritura de la Memoria

En este apartado se recoge el tiempo empleado para la escritura de la memoria del trabajo. La Tabla 7.5 muestra los tiempos necesarios para cada apartado de la memoria.

Tabla 7.5. Tarea 5: Escritura de la Memoria

Tarea	Fecha Inicio	Fecha Final	Horas
Escribir Introducción	22/12/20	30/01/21	18
Escribir Estado del Arte	20/01/21	12/03/21	20
Escribir Estudio y Obtención de los Datos	01/02/21	15/02/21	13
Escribir Experimentación y Resultados	25/02/21	05/05/21	28
Escribir Evaluación	07/05/21	17/06/21	12
Escribir Conclusiones y Trabajos Futuros	18/06/21	20/06/21	6
Escribir Gestión del Trabajo	01/06/21	10/06/21	16
Escritura de la Memoria	22/12/20	20/06/21	113

7.2.6. Tarea 6: Reuniones con el Tutor y Preparación de la Presentación

En este apartado se presentan la duración y día de las reuniones con el tutor del trabajo, Dr. Alejandro Cervantes, que se plantean como mensuales para realizar un seguimiento del trabajo.

Además, en este apartado se incluye el tiempo necesario para realizar la presentación de la defensa del trabajo. En la Tabla 7.6 se muestran todos estos valores.

7.2.7. Diagrama de Gantt

En este apartado se muestra la tabla resumen de todas las tareas (Tabla 7.7 y el diagrama de Gantt de las mismas (Figura 7.1).

Tabla 7.6. Tarea 6: Reuniones con el Tutor y Preparación de la Presentación

Tarea	Fecha Inicio	Fecha Final	Horas
Reunión Inicial	31/07/20	31/07/20	2
Reunión Septiembre	10/09/20	10/09/21	1
Reunión Octubre	08/10/20	08/10/20	1
Reunión Noviembre	13/11/20	13/11/20	1
Reunión Diciembre	-	-	0
Reunión Enero	13/01/21	13/01/21	1
Reunión Febrero	15/02/21	15/02/21	1
Reunión Marzo	16/03/21	16/03/21	1
Reunión Abril	-	-	0
Reunión Mayo	07/05/21	07/05/21	1
Reunión Junio	15/06/21	15/06/21	1
Preparación presentación	15/06/21	30/06/21	5
Reuniones con el Tutor y Preparación de la Presentación	31/07/20	30/06/21	15

Tabla 7.7. Fases del trabajo con su duración

Tarea	Fecha Inicio	Fecha Final	Horas
Obtención de Conocimientos	01/09/20	09/02/21	45
Recopilación de Datos	10/09/20	03/01/21	51
Desarrollo de las Redes Generativas	07/10/20	17/02/21	91
Comparativa de los Modelos	17/01/21	15/06/21	33
Escritura de la Memoria	22/12/20	20/06/21	113
Reuniones con el Tutor y Preparación de la Presentación	31/07/20	30/06/21	15
Tiempo total del trabajo	31/07/20	30/06/21	348

7.3. Marco regulador

En este apartado se hace referencia al marco legal y regulatorio en el cual se encuentra el trabajo. Se mencionarán y detallarán en que aspectos están relacionadas con los distintos apartados del trabajo. Además se tratarán las cuestiones relacionadas con la propiedad intelectual del trabajo desarrollado.

7.3.1. Reglamento General de Protección de Datos

La Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDPGDD)[34] es la ley que protege a las personas físicas respecto a la cir-

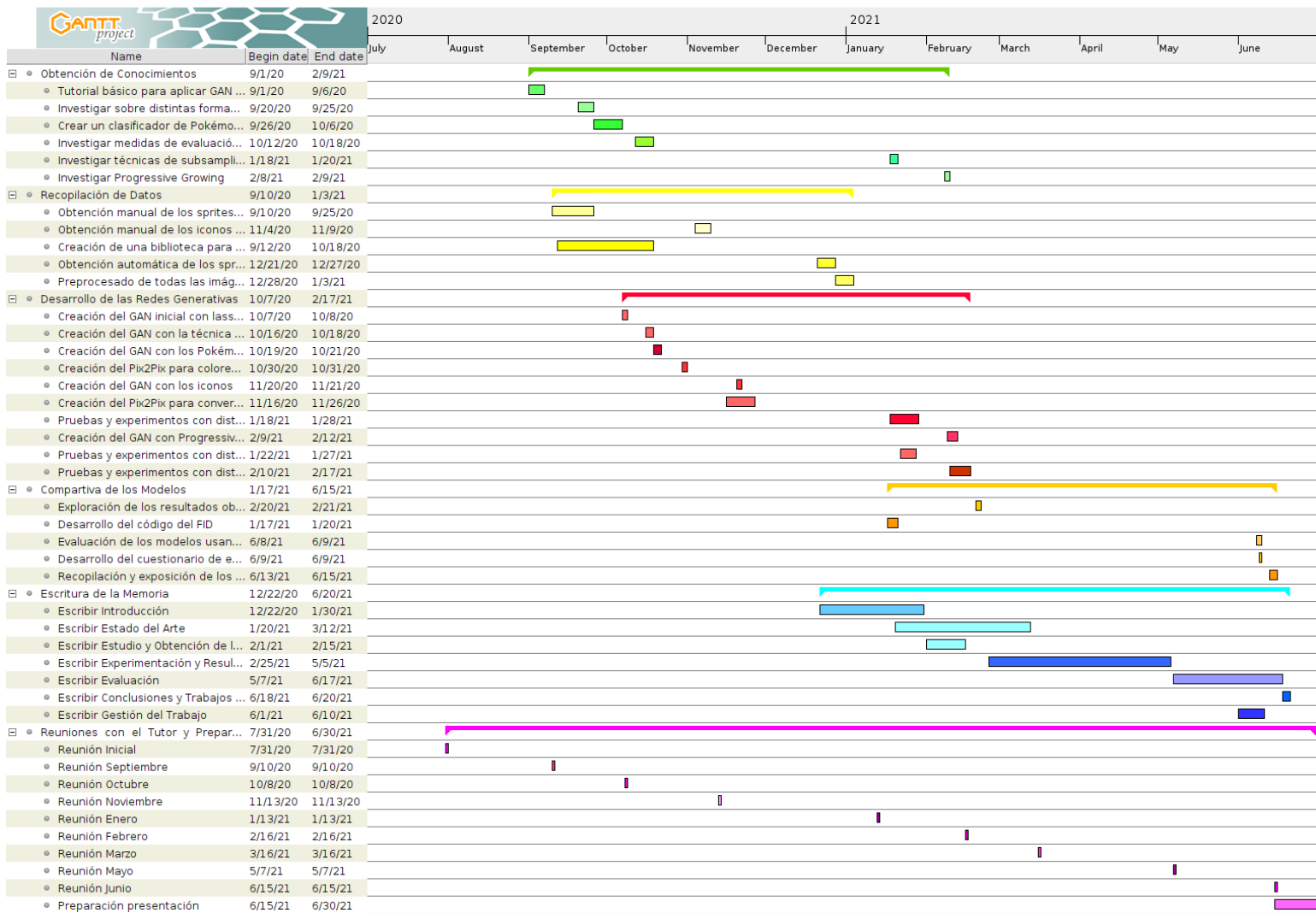


Figura 7.1. Diagrama de Gantt del trabajo

culación y utilización de su información y datos personales, como pueden ser imágenes, correos electrónicos o direcciones domiciliarias, o cualquier tipo de dato que pueda permitir la identificación de una persona.

El objetivo de esta ley es proteger la privacidad e intimidad de los usuarios y garantizar el uso apropiado y justificado de su información personal.

Dentro del contexto de este trabajo, esta ley es relevante dado que se realiza una encuesta para evaluar la calidad de los modelos desarrollados. En esta encuesta se menciona explícitamente *"Todos los datos recogidos en la misma son exclusivamente de uso estadístico y ningún dato personal será presentado en el trabajo."*. Ninguna información personal es almacenada ni recogida de los usuarios que rellenan la encuesta.

7.3.2. Ley de Propiedad Intelectual

La Ley de Propiedad Intelectual[35] indica que la propiedad intelectual de cualquier obra de cualquier índole pertenece exclusivamente al autor de la misma. Otorga al autor el derecho exclusivo sobre la explotación comercial de la obra.

Esta ley es relevante dentro del trabajo porque se hace uso de imágenes de Pokémon. Estas imágenes pertenecen a *Nintendo*, *The Pokémon Company*, *Nintendo Creatures* y *Game Freak*. Su uso es estrictamente no comercial, y con fines puramente académicos y educativos.

Dado que los resultados de los distintos modelos basados en redes de neuronas entrenadas con imágenes de Pokémon son imágenes que pueden simular o ser parecidas a Pokémon, se determina que estos resultados pueden ser considerados *Fanarts* sobre Pokémon, y por tanto se rigen por las reglas que establece *The Pokémon Company* sobre este tipo de obras[36].

7.3.3. Propiedad Intelectual de los Resultados

En este trabajo no se plantea ningún acercamiento innovador, ya que todos están basados en trabajos previos, sin aportar diferencias significativas. No requiere ni el desarrollo de nuevas patentes ni protecciones adicionales

7.4. Entorno socioeconómico

En este apartado se plantea analizar tanto el presupuesto necesitado para este proyecto, como el posible impacto socioeconómico que podría crear la estandarización y uso en gran escala de acercamientos adversariales en la creación de gráficos e imágenes para distintos videojuegos.

7.4.1. Presupuesto

En este apartado se muestra el presupuesto ficticio que se presenta como posible para este trabajo, dada su magnitud.

Se tienen en cuenta los costes directos, compuestos por costes de personal y costes de material, y los costes indirectos. También se tiene en cuenta costes sobre riesgo y beneficios, y por último se le aplica el IVA.

Costes de Personal

En este trabajo se pueden diferenciar dos roles. El rol de supervisor y jefe de proyecto, desempeñado por el tutor del trabajo Alejandro Cervantes, y el rol del desarrollador, desempeñado por el alumno Alejandro Valverde. En la Tabla 7.8 se realiza un desglose de los costes de cada rol.

Tabla 7.8. Costes de Personal del trabajo

Rol	Coste/Hora	Horas Totales	Coste Total
Supervisor y Jefe de Proyecto	15,70 €	30	471,00 €
Desarrollador	12,35 €	348	4297,80 €
Total			4768,80€

Costes de Material

En este apartado se describen tanto el coste de *hardware* como de *software* de los recursos utilizados en este trabajo.

Como coste de *hardware*, se tiene en cuenta todos los dispositivos que han sido necesarios para llevar a cabo el trabajo. Esto se muestra con detalle en la Tabla 7.9.

Tabla 7.9. Costes de *Hardware*

Material	Precio	Meses	Vida Útil	Coste Total
Ordenador Sobremesa	846,35 €	10	6 años	117,55 €
Portátil Lenovo V110-15ISK	429,00 €	10	4 años	89,37 €
Monitor HP 24"	109,00 €	10	10 años	9,08 €
Monitor Samsung 24"	118,00 €	10	10 años	9,83 €
Ratón Baojiarong T20	17,99 €	10	6 años	2,50 €
Teclado Mecánico Newskill Serike	41,28 €	10	15 años	2,29 €
Total				230,62 €

El único coste de *software* que se tiene en cuenta en este trabajo es el alquiler ficticio de un ordenador remoto. Este alquiler no ha sido necesario, ya que el tutor ha podido hacer uso de los ordenadores disponibles en su departamento para llevar a cabo los entrenamientos que requerían de más potencia de cálculo.

Dado que el resto de tecnologías que se han usado, o eran gratuitas, o gracias a las licencias de la universidad, no ha sido necesario realizar ningún pago. Esto se puede ver con detalle en la Tabla 7.10.

*₁ Este coste es una aproximación del coste de alquilar los servicios de *Amazon Machine Learning*[37] realizando sobre ellos la transformación de dólares americanos a eu-

Tabla 7.10. Costes de *Software*

Servicio	Coste/Hora	Horas totales	Coste Total
Uso de ordenador remoto	0,35 €* ₁	30* ₂	10,50 €
<i>Google Workspace</i>	0,00 €	10	0,00 €
<i>Overleaf</i>	0,00 €	113	0,00 €
Total			10,50 €

ros el 10 de junio de 2021.

*₂ Tiempo aproximado de los entrenamientos que se realizaron en el ordenador remoto.

Coste Total

A los costes ya calculados, es necesario añadir un porcentaje adicional de costes indirectos, como puede ser el coste de la electricidad, o la tarifa del internet. En este caso se estima como un 15 % del coste total del trabajo.

Además de estos costes, se añade un 10 % de riesgos e imprevistos y un 15 % adicional que representa el margen de beneficios del trabajo. Por último, sobre el coste total de todos los costes se ha de añadir un 21 % de IVA.

En la Tabla 7.11 se detalla el coste de cada apartado desglosado así como el coste total del trabajo.

Tabla 7.11. Costes Totales

Concepto	Coste
Costes de Personal de Trabajo	4768,80 €
Costes de <i>Hardware</i>	230,62 €
Costes de <i>Software</i>	10,50 €
Costes Directos	5009,52 €
Costes Indirectos(15 %)	751,49 €
Riesgos e Imprevistos(10 %)	500,99 €
Beneficio(15 %)	751,49 €
Coste total sin IVA	7013,89 €
IVA	1472,92 €
Total	8486.81 €

7.4.2. Impacto socioeconómico

En este apartado se presenta el impacto socio-económico que tendría la posibilidad de usar técnicas adversariales para llevar a cabo la creación de gráficos utilizables en videojuegos.

Sobre la tarea de creación de Pokémon que se ha llevado a cabo en este trabajo, los resultados no podrían usarse dentro de un videojuego, dado que no tienen calidad suficiente, pero sí podrían usarse como fuente de inspiración a la hora de desarrollar Pokémon nuevos. De esta forma, aunque su impacto es pequeño, podría llegar a ser útil a la hora de hacer futuros juegos de la saga de Pokémon. Por tanto, el impacto directo de este trabajo no es demasiado grande, debido a la baja calidad de los resultados obtenidos, y que está centrado sobre un único conjunto de datos.

Si esta forma de generar imágenes tomando como referencia distintas imágenes de videojuegos se populariza y perfecciona, podría ser utilizada por empresas de desarrollo de videojuegos para crear imágenes o *sprites* que utilizar en sus obras. Esto podría abaratar el coste de generación de imágenes, aunque podría encarecerse el coste de los equipos, ya que si se quiere obtener buena calidad en las imágenes, hacen falta una gran cantidad de recursos computacionales. Se trata de realizar una creación artística automatizada, lo que requiere perfiles distintos dentro de los equipos de desarrollo.

Una limitación de esta técnica, que podría resultar en detrimento de las compañías más pequeñas o con acceso a menor cantidad de imágenes que tomar como referencia, es la gran cantidad de imágenes que requieren este tipo de redes para generar resultados utilizables. Si se perfeccionan las distintas técnicas aplicadas en este trabajo que permiten utilizar conjuntos reducidos y dispares de datos, esta limitación podría desaparecer, o al menos reducirse lo suficiente como para que los resultados fueran aceptables.

Este trabajo se plantea como un desarrollo de investigación, dentro del uso de tecnologías innovadoras. No se intenta crear un producto que genere un retorno económico inmediato, sino crear un marco de trabajo, o *framework*, donde se incluyan técnicas y metodologías que permitan la resolución de cualquier problema que requiera de creación de imágenes para la industria del videojuego.

Por lo tanto, se concluye que a pesar de que el impacto, tanto social como económico, de los modelos desarrollados en el trabajo no es demasiado elevado, su perfeccionamiento así como su ampliación a otros conjuntos de datos podría suponer un cambio en la forma de crear imágenes para videojuegos. Lo que generaría un impacto innovador en esta industria, que afectaría a nivel económico al coste de la generación de las imágenes y a nivel social con el perfil de trabajador necesario para crear y aplicar estas técnicas.

BIBLIOGRAFÍA

- [1] T.-C. Wang et al., *Video-to-Video Synthesis*, 2018. arXiv: [1808.06601 \[cs.CV\]](#).
- [2] T. Shi et al., *Face-to-Parameter Translation for Game Character Auto-Creation*, 2019. arXiv: [1909.01064 \[cs.CV\]](#).
- [3] moxiegushi, *pokeGAN*, mayo de 2018. [En línea]. Disponible en: <https://github.com/moxiegushi/pokeGAN>.
- [4] ghanashyamvatti, *PokeGAN*, oct. de 2020. [En línea]. Disponible en: <https://github.com/ghanashyamvatti/PokeGAN>.
- [5] arghyadeep99, *PokeGAN*, oct. de 2019. [En línea]. Disponible en: <https://github.com/arghyadeep99/PokeGAN>.
- [6] C. Lazarou, *I Generated Thousands of New Pokemon using AI*, nov. de 2020. [En línea]. Disponible en: <https://towardsdatascience.com/i-generated-thousands-of-new-pokemon-using-ai-f8f09dc6477e>.
- [7] ———, *PokeGAN*, nov. de 2020. [En línea]. Disponible en: <https://github.com/ConorLazarou/PokeGAN>.
- [8] M. Rayfield, *pokemon-gpt-2*, dic. de 2020. [En línea]. Disponible en: <https://github.com/ConorLazarou/PokeGAN>.
- [9] ———, (2020). “creating 3000 new POKEmon with AI (the wrong way) featuring GPT-2,” Youtube, [En línea]. Disponible en: <https://www.youtube.com/watch?v=Z9K3cwSL6uM>.
- [10] A. Radford et al., “Language Models are Unsupervised Multitask Learners,” 2019.
- [11] Wikipedia, *Inteligencia artificial — Wikipedia, La enciclopedia libre*, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Inteligencia_artificial&oldid=135481907%7D.
- [12] F. S. Caparrini, *Variational AutoEncoder*, mar. de 2020. [En línea]. Disponible en: <http://www.cs.us.es/~fsancho/?e=232#Ref1>.
- [13] I. J. Goodfellow et al., *Generative Adversarial Networks*, 2014. arXiv: [1406.2661 \[stat.ML\]](#).
- [14] Wikipedia, *Minimax — Wikipedia, La enciclopedia libre*, 2019. [En línea]. Disponible en: <https://es.wikipedia.org/w/index.php?title=Minimax&oldid=120831239%7D>.
- [15] A. Radford, L. Metz y S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, 2016. arXiv: [1511.06434 \[cs.LG\]](#).

- [16] GLUON, *Deep Convolutional Generative Adversarial Networks*, 2017. [En línea]. Disponible en: https://gluon.mxnet.io/chapter14_generative-adversarial-networks/dcgan.html.
- [17] C. Lazarou, *Autoencoding Generative Adversarial Networks*, 2020. arXiv: [2004.05472 \[cs.LG\]](#).
- [18] P. Isola, J.-Y. Zhu, T. Zhou y A. A. Efros, *Image-to-Image Translation with Conditional Adversarial Networks*, 2018. arXiv: [1611.07004 \[cs.CV\]](#).
- [19] O. Ronneberger, P. Fischer y T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. arXiv: [1505.04597 \[cs.CV\]](#).
- [20] S. Zhao, Z. Liu, J. Lin, J.-Y. Zhu y S. Han, “Differentiable Augmentation for Data-Efficient GAN Training,” en *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.
- [21] T. Karras, T. Aila, S. Laine y J. Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, 2018. arXiv: [1710.10196 \[cs.NE\]](#).
- [22] A. Borji, *Pros and Cons of GAN Evaluation Measures*, 2018. arXiv: [1802.03446 \[cs.CV\]](#).
- [23] T. Salimans et al., *Improved Techniques for Training GANs*, 2016. arXiv: [1606.03498 \[cs.LG\]](#).
- [24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens y Z. Wojna, *Rethinking the Inception Architecture for Computer Vision*, 2015. arXiv: [1512.00567 \[cs.CV\]](#).
- [25] D. Mack, *A simple explanation of the Inception Score*, mar. de 2019. [En línea]. Disponible en: <https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a>.
- [26] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler y S. Hochreiter, *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, 2018. arXiv: [1706.08500 \[cs.LG\]](#).
- [27] S. Tajiri, K. Sugimori, GameFreak y Nintendo. (1996). “Pokémon,” [En línea]. Disponible en: <https://www.pokemon.com/es/>.
- [28] (1989). “Game Freak,” [En línea]. Disponible en: <https://www.gamefreak.co.jp/>.
- [29] Htfcuddles. (2006). “Wikidex,” [En línea]. Disponible en: <https://www.wikidex.net/wiki/WikiDex>.
- [30] Tensorflow, *Deep Convolutional Generative Adversarial Network*, 2021. [En línea]. Disponible en: <https://www.tensorflow.org/tutorials/generative/dcgan>.
- [31] H. M. Soares. (2017). “Who is that Neural Network?” [En línea]. Disponible en: <https://jgeekstudies.org/2017/03/12/who-is-that-neural-network/>.

- [32] T. Karras, S. Laine y T. Aila, *A Style-Based Generator Architecture for Generative Adversarial Networks*, 2019. arXiv: [1812.04948](https://arxiv.org/abs/1812.04948) [cs.NE].
- [33] J. Deng et al., “ImageNet: A Large-Scale Hierarchical Image Database,” en *CVPR09*, 2009.
- [34] J. del Estado, *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*. <https://www.boe.es/buscar/doc.php?id=BOE-A-2018-16673>, 2018.
- [35] M. de Cultura, *Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia*. <https://www.boe.es/buscar/doc.php?id=BOE-A-1996-8930>, 1996.
- [36] T. P. Company, *Información legal*, <https://www.pokemon.com/es/legal/>, 2021.
- [37] A. W. Services, *Build a Machine Learning Model*, 2021. [En línea]. Disponible en: <https://aws.amazon.com/getting-started/projects/build-machine-learning-model/services-costs/>.
- [38] K. Sugimori. (2020). “Ken Sugimori Pokémon artwork. From Bulbagarden Archives,” [En línea]. Disponible en: <http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm>.
- [39] K. Sugimori, GameFreak y Nintendo. (1996). “Pokédex,” [En línea]. Disponible en: <https://www.pokemon.com/es/pokedex/>.
- [40] (1889). “Nintendo Company,” [En línea]. Disponible en: <https://www.nintendo.es/>.
- [41] T. Folkman. (2020). “How To Create Unique Pokémon Using GANs,” [En línea]. Disponible en: <https://towardsdatascience.com/how-to-create-unique-pokemon-using-gans-ea1cb6b6a5c2>.
- [42] S. Verma. (2019). “Multi-Label Image Classification with Neural Network | Keras,” [En línea]. Disponible en: <https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede>.
- [43] akua21, *Mini-Pokedex*, <https://github.com/akua21/Mini-Pokedex>, 2020.
- [44] Min-Ling Zhang y Zhi-Hua Zhou, “Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, n.º 10, pp. 1338-1351, 2006.
- [45] I. Kuznetsov. (2019). “Image Augmentation in Numpy. The spell is simple but quite unbreakable,” [En línea]. Disponible en: <https://medium.com/@schatty/image-augmentation-in-numpy-the-spell-is-simple-but-quite-unbreakable-e1af57bb50fd>.

- [46] J. Brownlee. (2019). “How to Implement the Inception Score (IS) for Evaluating GANs,” [En línea]. Disponible en: <https://machinelearningmastery.com/how-to-implement-the-inception-score-from-scratch-for-evaluating-generated-images/>.
- [47] ———, (2019). “How to Implement the Frechet Inception Distance (FID) for Evaluating GANs,” [En línea]. Disponible en: <https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/>.
- [48] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler y S. Hochreiter, *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*, 2018. arXiv: [1706.08500](https://arxiv.org/abs/1706.08500) [cs.LG].
- [49] A. Odena, V. Dumoulin y C. Olah, “Deconvolution and Checkerboard Artifacts,” *Distill*, 2016. doi: [10.23915/distill.000003](https://doi.org/10.23915/distill.000003). [En línea]. Disponible en: <http://distill.pub/2016/deconv-checkerboard>.

8. ANEXO

8.1. Abstract

This Bachelor Thesis proposes the use of adversarial techniques for image generation. Specifically, the use of algorithms based on artificial neural networks is proposed with the purpose of generating images that can be used in different video games. This work focuses on a single dataset. It is decided to make use of a dataset made up of Pokémon images, since it is a dataset with a small and diverse number of examples. Therefore, special emphasis is placed on techniques applicable to generative models so that they are able to create good results with the fewest possible examples. To carry out the generation of images, algorithms based on artificial neural networks belonging to the GAN family are used.

The main phases of this Bachelor Thesis are the obtaining and preparation of the dataset, the development and implementation of adversarial models and techniques for the creation of images, and, finally, an evaluation of the obtained results using both numerical metrics and surveys, in order to determine which model is capable of generating the images with the best quality.

The results obtained do not bear a great resemblance to the original set of images, and therefore cannot be considered a success in that regard. However, despite its limitations, with further development, these techniques could be used in the video game industry with the aim of reducing design costs and increasing the amount of content.

Keywords: Artificial Intelligence; Artificial Neural Networks; Image Generation; Generative Adversarial Networks; Video games; Pokémon

8.2. Introduction

8.2.1. Objectives

The generation of images with adversarial techniques is becoming very important and is becoming quite popular. This has led to an explosion of different works, new network architectures, and numerous applications that make use of these techniques.

This work proposes to make use of adversarial strategies through artificial neuronal networks in order to create new images usable in the video game industry. Specifically, it will try to generate new Pokémon, similar to the original ones, but different from any that already exist.

Although there are different ways of generating images, this work will focus exclusively on approaches that are based on artificial neural networks with adversarial training, or what is the same, models composed of two or more neural networks with opposing objectives.

Furthermore, an attempt is made to solve the problem by exclusively using a reduced and uneven dataset, with the aim of analyzing, evaluating and comparing the behavior of different architectures and approaches.

8.2.2. Motivation

The main motivation of this work is to bring adversarial techniques based on artificial neural networks to the video game industry, demonstrating its usefulness and the benefits it offers. If it could be applied correctly, it would allow different video game developers to obtain numerous images similar to the original set in a reduced time, lessening the costs in creating those images.

Besides, the work is presented as a study of the different adversarial models and how they behave on reduced and unequal sets of data. As an additional motivation, since there is no really positive result that solves the specific problem of generating new Pokémon, the challenge is to find a model good enough to fool a person who is not too familiar with the franchise.

Another reason for using the Pokémon dataset is because of its great variability and difference, as can be seen in Figure A.1. This is an added difficulty for models based on artificial neural networks, whose task is to make generalizations about the data, which, if they are too different, is more expensive and complex.

On a personal basis, it is desired to carry out this work to deepen the concepts on artificial neural seen in the degree, as well as to learn new technologies used in the real world. In addition, given the author's interest and knowledge about Pokémon, further



Figure A.1. Some *sprites* of Pokémon of different appearances

information may be used to build the different network models.

8.3. What are Pokémon?

Pokémon[27](ポケモン) is a world-renowned entertainment media franchise. Its main attraction is the creatures that give the brand its name, the Pokémon. These creatures were born in 1996 from the hand of *Satoshi Tajiri* in the *Pokémon Red* and *Pokémon Blue* video games, created by the company Game Freak[28].

These Pokémon, designed by *Ken Sugimori*, are usually based on animals or objects from the real world and are very recognizable, being radically different from each other. Each of the Pokémon has various aspects and characteristics that make them unique. Of course, each Pokémon has its own name and appearance. In addition, they can have one or more evolutions, which are transformations that the Pokémon can achieve if specific conditions are met. In turn, each Pokémon belongs to one or two types, which can usually be guessed by the shape and color of the Pokémon. There are currently a total of 18 types. In FigureA.2 you can see the evolutionary family of the Pokémon Shinx, where the three evolutions are of the electric type.



Figure A.2. Evolutionary family of *Shinx*. From left to right, *Shinx*, *Luxio* and *Luxray*

8.4. Results

To solve the problem that arises, different techniques are used based on adversarial approaches. The most notable are those that make use of a DCGAN, and those that combine both DCGAN and Pix2Pix.

8.4.1. DCGAN

In this case, a DCGAN is used, combined with two techniques that allow to improve the results obtained: Differentiable Augmentation and a pre-training of their weights using an Autoencoder.

Differentiable Augmentation is a type of Data Augmentation that can be applied on DCGAN. It allows to generate better results using a reduced set of images by applying different transformations on the input images. Also, since it is a training style, it does not require modification of the original architecture.

The pre-training of the weights using an Autoencoder tries that, before starting the adversarial training, both the generator and the discriminator have basic information that allows them to generate and discern the results so that images more similar to the original Pokémon are obtained.

The results of the pre-training with the Autoencoder can be seen in the Figure A.3. It can be seen that, despite not being a perfect, it does have a lot of resemblance to the original images.

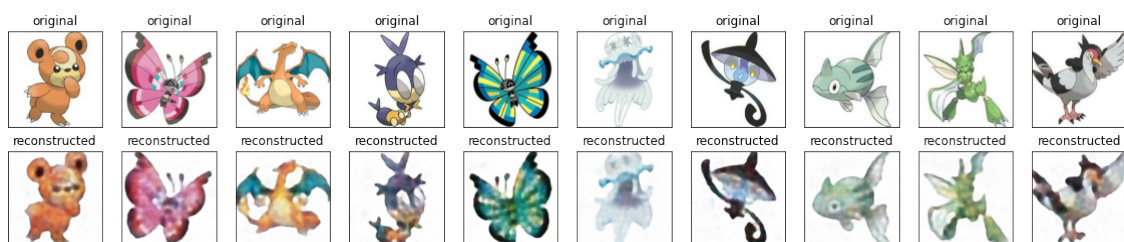


Figure A.3. Pre-training results using *Autoencoders*

The results obtained, despite not being good enough to fool the human eye, do have enough positive results. The colors, although in some cases they are mixed in an unwanted way, they have a lot of cohesion and sense. The edges are well defined. One cannot distinguish details or parts of the body. They are good results for a first approach, but they are still far from ideal. They can be seen in the Figure A.4.



Figure A.4. Results obtained with the basic DCGAN using *Differentiable Augmentation* and initialization of weights with *Autoencoders*

8.4.2. DCGAN with Pix2Pix

This approach consists of making use of a DCGAN architecture to generate low-resolution images, which are easier to generate, and which require a lower computational cost and a Pix2Pix architecture is then used to perform the transformation from low to high resolution to achieve the desired results.

Again, with DCGAN both Differentiable Augmentation and network pre-training with Autoencoders are applied. This allows to generate results very similar to the originals, as can be seen in the Figure A.5. However, although they bear a greater resemblance to the original images, they still fail to create recognizable body parts, and make disjointed mixtures of colors.

For the Pix2Pix model, a basic Pix2Pix architecture is proposed, with a generator that follows the U-Net architecture and a Discriminator with a basic convolutional architecture.



Figure A.5. Results obtained with DCGAN, *Differentiable Augmentation* and initialization of weights with *Autoencoders*

The results obtained with this architecture are quite poor. The main problem why these bad results are obtained is because it is not asking for a small change on the original image, but rather trying to make a complete transformation of the original image in order, starting from low resolution, to get an image with greater resolution that not only has this higher resolution, but most of the time it has a different pose, and even different colors. The results can be seen in Figure A.6, where it can be seen that they do not have the appropriate colors, or the silhouette or the shape. In addition, a recurring pattern can be observed that the network generates, and it should not generate.

Although the results are still far from desirable, some progress and utility can be seen in them.

8.5. Evaluation

The evaluation of generative models is not a trivial process. This is because there is no objective metric that determines how effectively a generative model is capable of creating new images. It is a consequence of the fact that most adversarial models belong to unsupervised learning, and their cost function does not represent how good the model is, but how close it is to converging. In most approaches, the cost function of the Generator network determines to what extent it has been able to deceive the Discriminator network, while the cost function of the Discriminator network indicates its ability to correctly classify samples as real or generated.

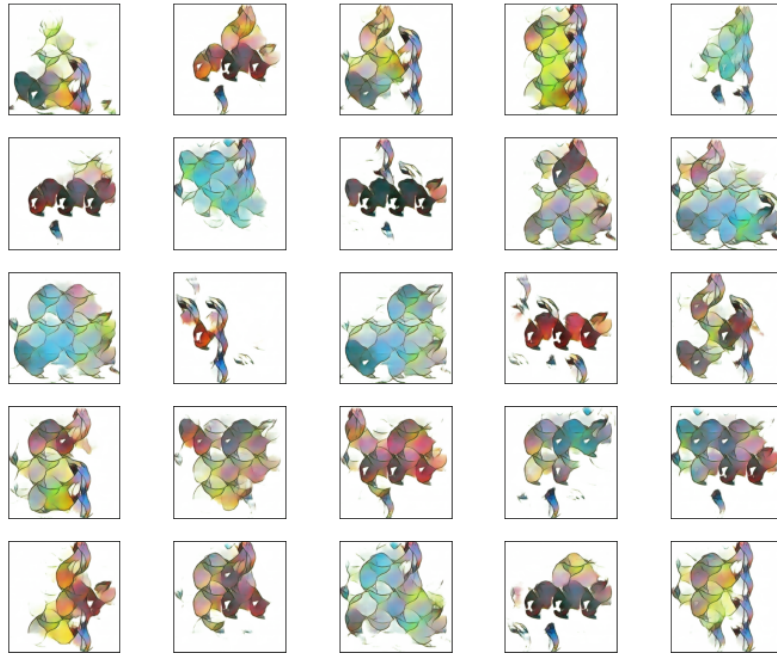


Figure A.6. Results obtained with the Pix2Pix architecture

There are a number of metrics that are frequently used when evaluating these types of architectures. In this case, it is proposed to use the FID as a more pseudo-objective measure. Additionally, an informal survey of different users is carried out.

Table A.1 shows the FID values obtained with each proposed model. Both the one consisting of Pix2Pix and the one consisting of an Autoencoder, since they take an icon as input, are evaluated both with icons that are part of the original set, and with icons that have been generated by a different network.

Table A.1. Models FID

Model Name	FID
<i>Base case between original images</i>	5,400
<i>Basic DCGAN</i>	94,238
<i>DCGAN with Differentiable Augmentation</i>	57,988
<i>DCGAN with Differentiable Augmentation and Dropout</i>	82,327
<i>DCGAN with Differentiable Augmentation and Autoencoders</i>	47,032
<i>Pix2Pix with original icons</i>	43,680
<i>Pix2Pix with generated icons</i>	39,373
<i>Autoencoder with original icons</i>	107,187
<i>Autoencoder with generated icons</i>	108,090

The survey is used as an additional measure and is expected to coincide with the results obtained using the FID. Instead of asking users to rate through an arbitrary scoring system, or use a percentage of similarity to the original images, they are asked to rank

based on how similar they are to the set of original images.

This decision regarding the structure of the survey is made because human beings are relatively poor at objectively and arbitrarily scoring a series of samples, but they have a great capacity to perform rankings on a series of elements.

As expected, the results of the FID and the survey coincide, proclaiming as the best model the one that makes use of a Pix2Pix architecture and receives low-resolution images generated by a DCGAN as inputs.

This demonstrates both the effectiveness of the FID metric to compare different generative models and that the results that are most similar to the original ones are achieved using adversarial approaches, instead of those that only uses Autoencoders.

8.6. Conclusions and Future Work

8.6.1. Professional Conclusions

The main objective of this work is to show the usefulness of adversarial models in the video game industry, which are capable of generating a large number of images of a quality very similar to the images taken as a reference. Their main problem is that they require a large number of images to achieve positive results.

Therefore, another objective of this work was to achieve positive results using a reduced and disparate set of reference images. In this case, the data set that is decided to use is that of the Pokémon.

In this work, the use of different techniques is proposed to improve training, and allow it to be carried out with few images. Despite this, the objective has not been fully met. Although the high-resolution images model has not achieved very good results, the low-resolution images models have managed to produce images that are very similar to the original dataset. This is due to several factors. The first and simplest is the difference in resolution, which means that the models have to generate fewer pixels, and therefore have fewer weights to optimize. The second, but which is also very important, is that the original icons have a much smaller color palette, and do not have shadows or complicated poses that imply a perception of depth.

Finally, the proposed model consists in turn of two concatenated models. The icon model based on DCGAN with a pre-training of its weights with Autoencoders, followed by a Pix2Pix architecture that increases the resolution of the images. This architecture can be seen graphically in Figure A.7.

Although the main objective of obtaining images capable of deceiving the human eye has not been achieved, significant progress has been made from the base case to the final

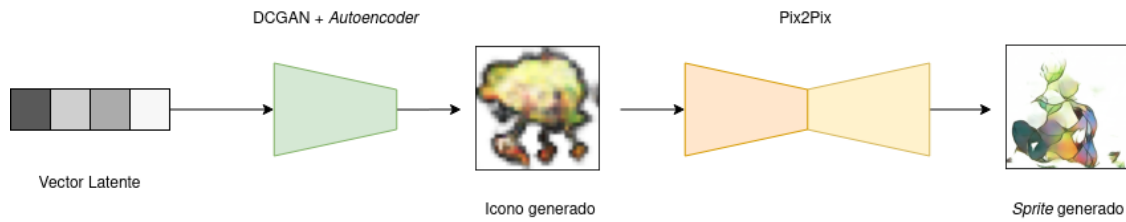


Figure A.7. Final model architecture. (In spanish, from left to right *Latent Vector*, *Low resolution generated image*, *High Resolution generated image*).

model

Adversarial approaches are very powerful, but they have a number of problems, most of which are discussed in this paper. The main one, and that has been tried to alleviate in this work, is the large number of images for their training, but it is not the only one. Each problem requires great effort and expertise to solve. There is no one generic network that is good for all problems. This means that a lot of time and resources have to be invested to achieve good results.

8.6.2. Personal Conclusions

On a personal level, this work has involved a great effort to develop and research technologies that could be applied to the specific problem, which has served to expand the knowledge about neuron network algorithms that are explained in the Computer Science and Engineering degree. It has also served to develop skills for analysis and reading of technical research papers.

8.6.3. Future Work

There are countless ways to continue this work. In the first instance, it would be necessary to refine the results obtained. For this, different techniques could be applied that may be useful in the specific problem. For example, the StyleGAN architecture could be used, which is one of the most interesting state of the art techniques and which, ideally, would allow to solve one of the biggest problems that current results have, and that is that they are not capable of creating specific parts such as arms. or legs, and they are not capable of giving the image details such as eyes or mouth.

In addition, if this technique is improved and perfected, so that it generates Pokémon that can be similar to real Pokémon, a web application could be proposed so that any interested user could use the model to generate different Pokémon.

Outside of the specific problem of creating Pokémon, these techniques can be applied for the generation of 2D images for video games (and not only for video games). It could also be extrapolated to the generation of three-dimensional models, with a fairly high

increase in complexity.

Specifically, within the video game industry, these techniques could be applied to the generation of textures. Textures used in video games, especially if they have a realistic appearance, are very expensive to make. It is necessary to have the material that you want to represent, use a specific camera and photograph it with optimal light conditions, so as not to generate unwanted shadows. Another way to achieve them is to have artists capable of creating these photorealistic textures, but it requires time and effort. The most common is to buy them in a specialized webpage that have multitude of textures. This cost and effort could be reduced if a model based on GAN was used, in charge of generating these textures.

Another possible use of this technology within the video game industry could be to generate enemy sprites for turn-based RPG games. These games usually have a large number of enemies, of very varied design. Typically, designers are responsible for creating them. Unfortunately, most of the time they do not have enough time to create as many as the game needs, and they usually use a very simple technique that consists of using the same design, but with different colors, to represent a different enemy. Using a Pix2Pix architecture, it could be trained to, for example, create enemies with a design similar to the input received.

And using a more creative approach, animations or attack patterns could be designed, for example, using a GAN technique whose objective is to generate code that can be transformed into these animations or attacks, which are similar to the training examples.

The possibilities are practically endless, and if used correctly, these techniques could revolutionize the video game industry, providing it with much more content, at a much lower cost.