Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Generación automática de texturas para videojuegos a partir de una descripción textual

Automatic videogame texture generation from a text description

Markus Schüller Perdigón

La Laguna, *8* de *julio* de 2022

D. **Rafael Arnay del Arco**, con N.I.F. 78569591G profesor Contratado Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **José Demetrio Piñeiro Vera**, con N.I.F. 43774048B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Generación automática de texturas para videojuegos a partir de una descripción textual”

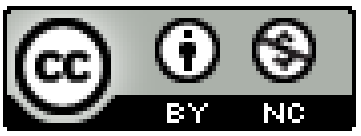
Ha sido realizada bajo su dirección por D. **Markus Schüller Perdigón**, con N.I.F. 43486444E.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de julio de 2022

Agradecimientos

A mi madre,
por supuesto.

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento-NoComercial 4.0 Internacional.

Resumen

El objetivo de este trabajo ha sido entrenar un modelo de Inteligencia Artificial para que, a partir de una descripción textual en lenguaje natural, sea capaz de generar imágenes que representen texturas para poder ser usadas en videojuegos (Text-to-image generation). Con el propósito de que el modelo produzca resultados acordes a la descripción que se introdujera, y dadas las limitaciones de tiempo y hardware de las que se dispone, se terminó por elegir que estas imágenes a generar serían skins del videojuego Minecraft, puesto que existe un gran número de ellas disponibles en internet y su poca resolución permitiría entrenar más rápidamente al modelo.

Palabras clave: Inteligencia Artificial, Text-to-image, texturas, skins, videojuegos, Minecraft.

Abstract

The objective of this project is to train an Artificial Intelligence model that, from a text description in natural language, is capable of generating images representing videogame textures (Text-to-image generation). With the purpose of yielding results that adjust to the input description, and given the hardware and time limitations, it was decided that these images to be generated will be skins of the videogame Minecraft, since there is a great number of them available on the internet and their low resolution would allow for a much faster training of the model.

Keywords: Artificial Intelligence, Text-to-Image, textures, skins, videogames, Minecraft.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción.

Para este trabajo se ha hecho uso de técnicas de inteligencia artificial que generan imágenes acordes a una entrada de texto dada, por lo que en el presente documento se describe su funcionamiento y su desarrollo. Este tipo de modelos de inteligencia artificial han mejorado enormemente en los últimos años, por lo que con este trabajo se pretenden conseguir resultados significativos con un problema más acotado.

En los últimos años, ha habido un gran avance en el campo de la Inteligencia Artificial, concretamente en el Aprendizaje Automático y principalmente debido al gran desarrollo de las redes neuronales. Con ello, muchos campos han progresado enormemente, entre ellos los más relacionados con este trabajo: el campo del Lenguaje Natural y el de la generación de imágenes. La herramienta de IA a diseñar consiste en:

1. Recibir un texto de entrada en lenguaje natural.
2. Reconocer el significado aproximado de ese texto.
3. Producir una imagen acorde a la descripción proporcionada.

El objetivo de este tipo de IAs, comúnmente conocidas como modelos *Text-To-Image* (**Figura 1.1**), es poder reproducir imágenes originales de alta fidelidad en base a lo que el usuario desee, de manera que éste pueda utilizar ese contenido con diversos propósitos. Por ejemplo, un diseñador podría describir un producto y obtener buenas ideas de cómo realizar el diseño gracias a la imagen generada por esa IA.

Dadas las limitaciones de tiempo y *hardware* de las que se disponen, se ha propuesto generar *skins* del videojuego *Minecraft*. Este es un videojuego de construcción del género *sandbox* que se caracteriza por su estilo artístico basado en cubos en *pixel art* donde los jugadores pueden personalizar su personaje jugable cambiando su textura, o como se le conoce comúnmente, su *skin* (**Figura 1.2**). Estas son imágenes desplegadas de un modelo tridimensional que, debido al estilo *pixel-art* del videojuego, tienen una baja resolución, por lo que resultan muy convenientes para entrenar rápidamente a una red neuronal.

En este documento comenzaremos viendo cuáles son las bases de la tecnología utilizada en los antecedentes, explicando cada uno de sus componentes y el estado del arte. Continuaremos con más detalles centrados en este trabajo específico con el capítulo dedicado a la metodología, en el que se describe concretamente el desarrollo del proyecto. Por último, terminaremos analizando los resultados en el capítulo dedicado a ello, además de presentar al final las conclusiones que se han obtenido con la realización de este trabajo.

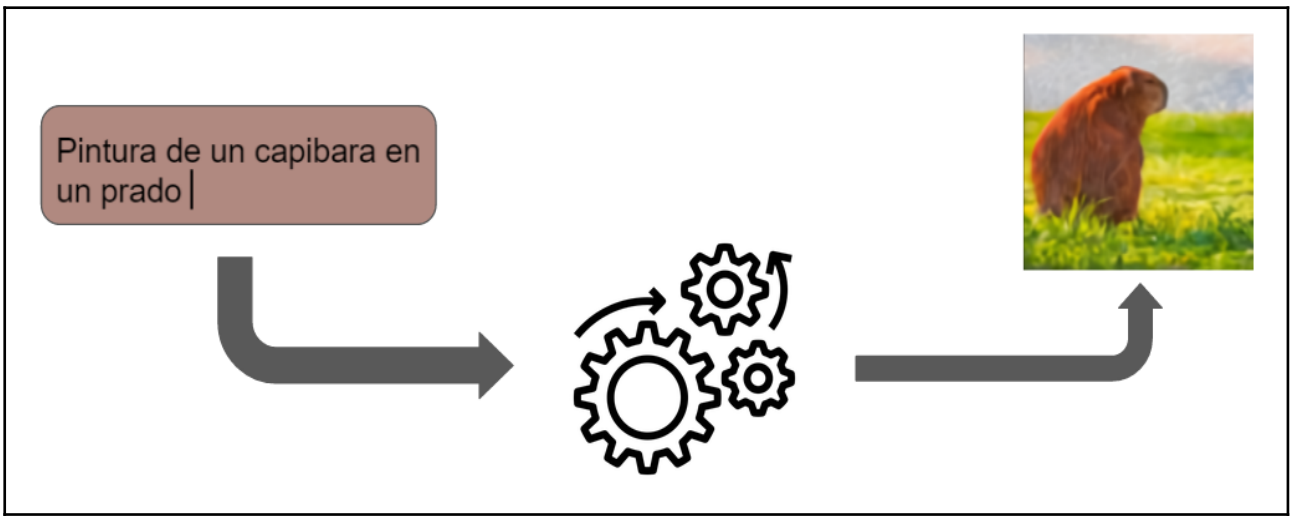


Figura 1.1: Esquema conceptual de una IA *Text-To-Image* genérica.

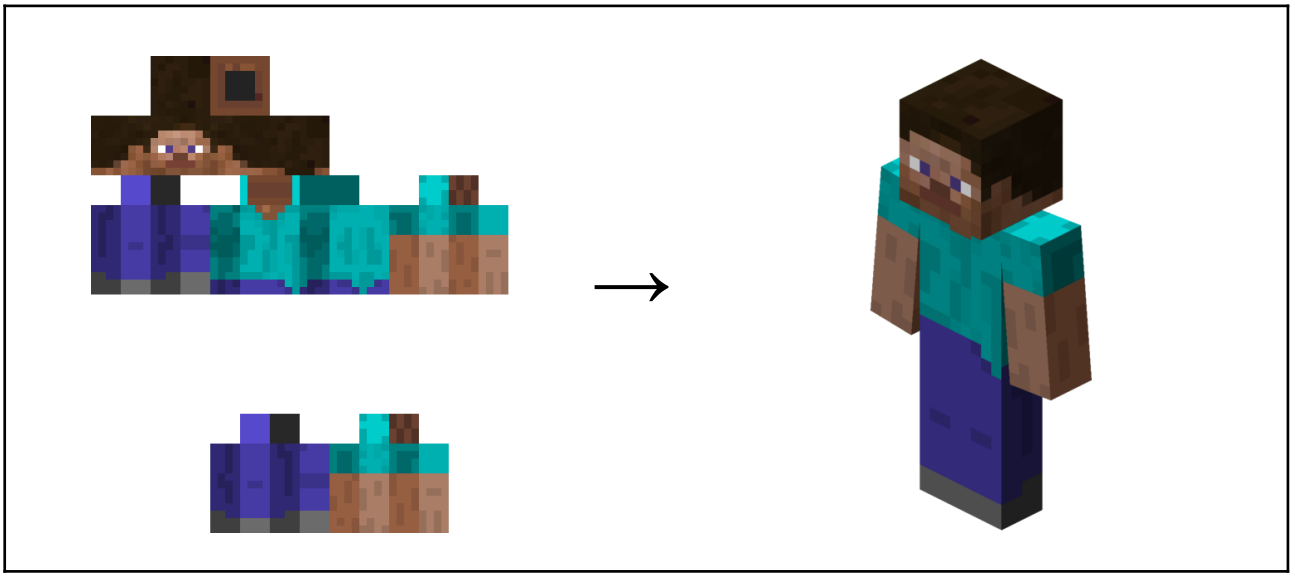


Figura 1.2: Ejemplo de una *skin* aplicada a un modelo de jugador de *Minecraft*.

1.1 Objetivos.

Para este trabajo se han planteado tres grandes objetivos principales, los cuales serán descritos en las secciones a continuación.

1.1.1 Construcción de un buen conjunto de datos.

Dado que se trata de un proyecto de Aprendizaje Automático, la construcción de un *dataset*, o conjunto de datos, es esencial para el entrenamiento de las redes neuronales. Por ello, obtener o construir un *dataset* consistente, variado y lo suficientemente grande será una de las primeras y más importantes tareas en este trabajo.

Como las imágenes a generar son *skins* de *Minecraft*, y el modelo tratará de generarlas a partir de descripciones textuales, el *dataset* ha de tener pares de imágenes y archivos de texto en los que se describa a la imagen asociada.

1.1.2 Conseguir un modelo *Text-To-Image* para *skins*.

El siguiente objetivo principal consiste en la obtención y el entrenamiento de una Inteligencia Artificial con el *dataset* previamente construido. Una vez entrenado, el modelo debería ser capaz de generar imágenes que se puedan utilizar como *skins* de *Minecraft*, y además en éstas deberían verse reflejadas las características descritas en un texto que se haya dado como entrada.

Por tanto, al entrenar la red neuronal, ésta debería llegar a entender la estructura que siguen las *skins* y asociando ciertas palabras a colores, partes del cuerpo específicas u objetos.

1.1.3 Proveer una aplicación gráfica.

Por último, aunque no sea estrictamente necesario, sería ideal desarrollar una aplicación gráfica para que el usuario pueda ver claramente los resultados generados, ya que como hemos visto en la introducción, las imágenes fuente de las *skins* son de un modelado tridimensional desplegado en dos dimensiones, por lo que para el usuario no sería tan fácil de visualizar.

Esta aplicación gráfica podría mostrar directamente al modelo en tres dimensiones con la *skin* aplicada, pudiendo rotar o moverla para así examinarla detenidamente, o seleccionar entre varios de los resultados que genere el modelo.

Capítulo 2 Antecedentes.

En el capítulo anterior se ha introducido la tecnología de Inteligencia Artificial a utilizar en este trabajo, las IAs *Text-To-Image*. A continuación, veremos más detalladamente el funcionamiento de este tipo de técnicas y cada uno de sus componentes, en especial sobre la que se ha basado este trabajo: **DALL-E** [1], un modelo *Text-to-Image* desarrollado por OpenAI que ha revolucionado el campo de la generación de imágenes.

2.1 Redes neuronales artificiales.

Como la mayoría de Inteligencias Artificiales más poderosas que se usan hoy en día, para generar este tipo de contenidos se usan **Redes Neuronales**. Son un modelo computacional que consiste en un conjunto de nodos, llamados neuronas, conectados entre sí para transmitirse señales y normalmente organizados en una o más capas. La información de entrada atraviesa la red neuronal por medio de las neuronas de entrada, tras lo cual se somete a diversas operaciones y acaba produciendo unos valores de salida en las neuronas de salida. Dado que este tipo de modelos aprenden y se forman a sí mismos, en lugar de ser programados de forma explícita, han demostrado ser excelentes en el campo del aprendizaje automático.

Gracias a los avances en hardware recientes, las redes neuronales se han conseguido desarrollar mucho últimamente ya que se aprovechan de la paralelización de procesamiento, una tarea ideal para tarjetas gráficas, las cuales han aumentado su potencia computacional especialmente en los últimos años.

Esto ha propiciado una nueva revolución para la Inteligencia Artificial, la cual ha abierto enormemente las puertas para investigaciones, entre otras cosas, de nuevas arquitecturas de redes neuronales, las cuales se pueden diseñar de manera que sean óptimas para determinado tipo de tareas. A continuación veremos a qué nos referimos con “Arquitectura” de una red neuronal y cuál ha sido la tendencia de su uso en los campos de la generación de imágenes.

2.1.1 Arquitectura.

Cuando nos referimos a la arquitectura de una red neuronal, estamos haciendo alusión a la manera en la que se interconectan las diferentes neuronas, a cómo se organizan las capas e incluso a cómo se conectan unas redes neuronales con otras o con otros elementos en modelos más complejos.

Una de las arquitecturas más simples se trata de un **perceptrón**, que consiste en una capa de neuronas de entrada a las que se les somete a una única operación para producir una salida por medio de la función de activación (**Figura 2.1**).

El siguiente paso en complejidad viene dado por el **perceptrón multicapa**, que tiene una capa de entrada constituida por neuronas que introducen los patrones de entrada en la red y en las que no se produce procesamiento. Las capas ocultas están formadas por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas

salidas pasan a neuronas de capas posteriores. En su capa de salida se encuentran las neuronas cuyos valores de salida se corresponden con las salidas de toda la red (**Figura 2.2**).

Estos son los diseños más básicos de arquitecturas de redes neuronales y pueden servir de base para comprender y diseñar otras arquitecturas más complejas, las cuales se pueden usar en el campo de la generación de imágenes y del lenguaje natural, así como en muchos otros campos.

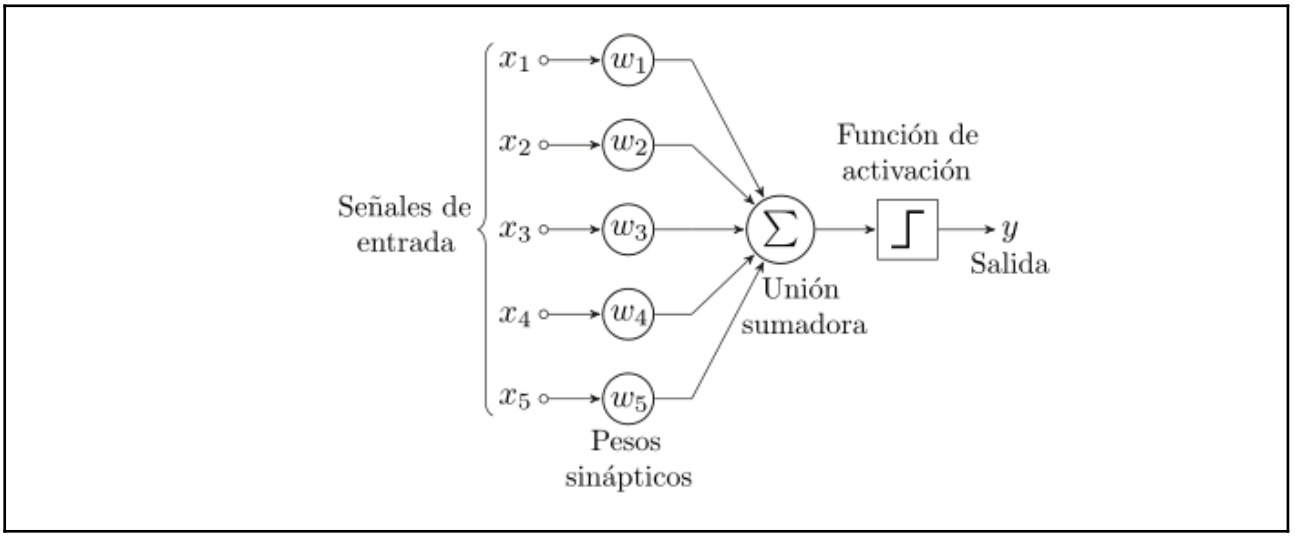


Figura 2.1: Arquitectura de un perceptrón simple.

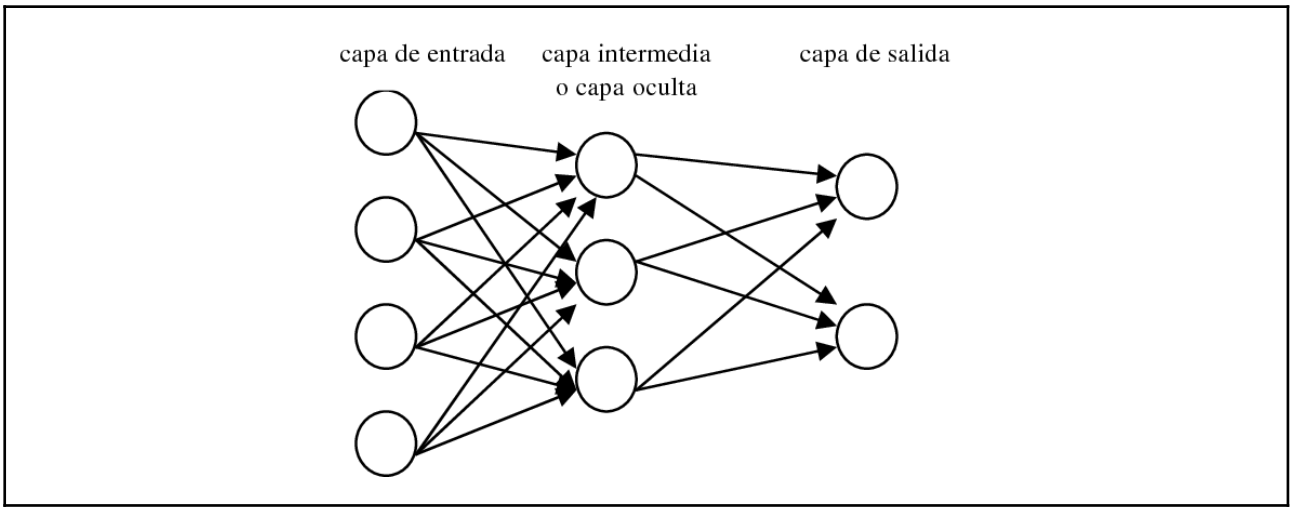


Figura 2.2: Arquitectura de un perceptrón multicapa.

2.2 Procesamiento del Lenguaje Natural.

Se trata de un campo de la Inteligencia Artificial que se ocupa de investigar la manera de comunicar las máquinas con las personas mediante el uso de lenguas naturales. Existen diferentes modelos que tratan diversos objetivos en este campo: generación de texto, reconocimiento de voz, traducción automática, responder coherentemente a preguntas hechas por personas… Sin embargo, para este trabajo nos centraremos en el análisis semántico del lenguaje, ya que es lo que se usa para reconocer el texto del usuario y que se genere contenido a partir de éste.

Muchos de estos modelos sientan sus bases en los conceptos de **tokenización** y en el de **word embedding**. Por un lado, la tokenización consiste en separar las piezas de texto en unidades más pequeñas conocidas como tokens. Estos tokens bien pueden ser caracteres, palabras enteras o sub-palabras siendo éste último la manera más común de tokenización de los últimos años.

Por otro lado, el word embedding es una técnica del Procesamiento del Lenguaje Natural que consiste, básicamente, en asignar un vector a cada palabra. Este vector guarda información semántica, lo que permite que pueda ser asociado o disociado a otros vectores (palabras) según distintos contextos gramaticales. De esta manera podemos representar matemáticamente a las palabras en un espacio n-dimensional de modo que una manzana esté geométricamente más “cerca” de un plátano y más “alejada” de, por ejemplo, un libro.

Tradicionalmente, para tratar con problemas de Procesamiento del Lenguaje Natural se han usado **Redes Neuronales Recurrentes**, que consisten en ir alimentando a una red neuronal secuencialmente las palabras de un texto: el output de la primera palabra se introduce como input para la segunda palabra y así sucesivamente (**Figura 2.3**). Sin embargo, este tipo de redes tienen un problema a la hora de procesar lenguaje natural: a medida que avanza el texto, los pesos de las primeras palabras se van disminuyendo, lo que se traduce en una falta de memoria y no puede brindar resultados muy fiables en textos muy largos. Para solucionar esto, se han introducido los llamados mecanismos de atención, que básicamente funcionan haciendo que una red neuronal asigne un vector de valores, llamado vector de atención, para cada palabra de un texto, que representará la importancia que le da el modelo al resto de palabras cuando está leyendo la palabra dada como input y las relacionará sin importar lo lejos que se encuentren estas palabras.

Es en estos mecanismos en los que se basan enormemente los **transformers** (**Figura 2.4**), un nuevo tipo de red neuronal que nació para sustituir a las redes neuronales recurrentes y ha proporcionado grandes avances en el campo del procesamiento del lenguaje natural.

La creación de los *transformers* en 2017 ha permitido el desarrollo de modelos de lenguaje mucho más poderosos, fiables y realistas debido principalmente a los mecanismos en los que se basan mencionados anteriormente, entre otros. También son usados en modelos que relacionan texto con imágenes, en los que nos centraremos y veremos con mayor profundidad posteriormente. Uno de los modelos más prominentes del Lenguaje Natural y basado en *transformers* es **GPT-3** [2], desarrollado por OpenAI y uno de los componentes que dan forma a DALL-E, el modelo en el que se centra este trabajo.

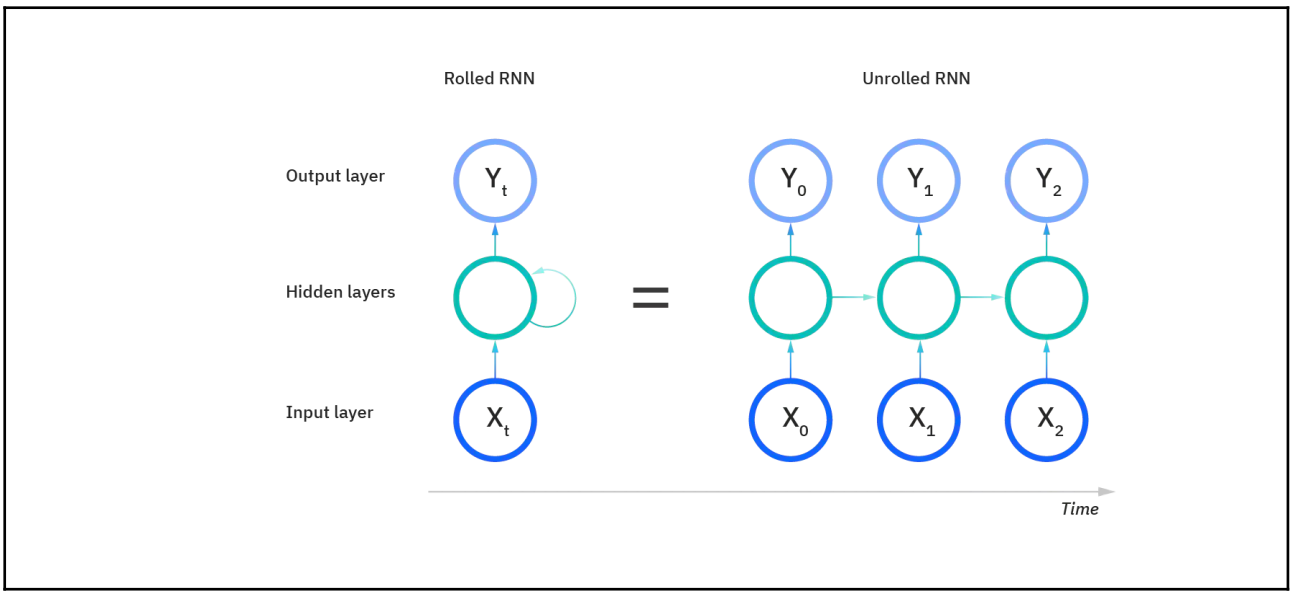


Figura 2.3: Arquitectura de una Red Neuronal Recurrente.

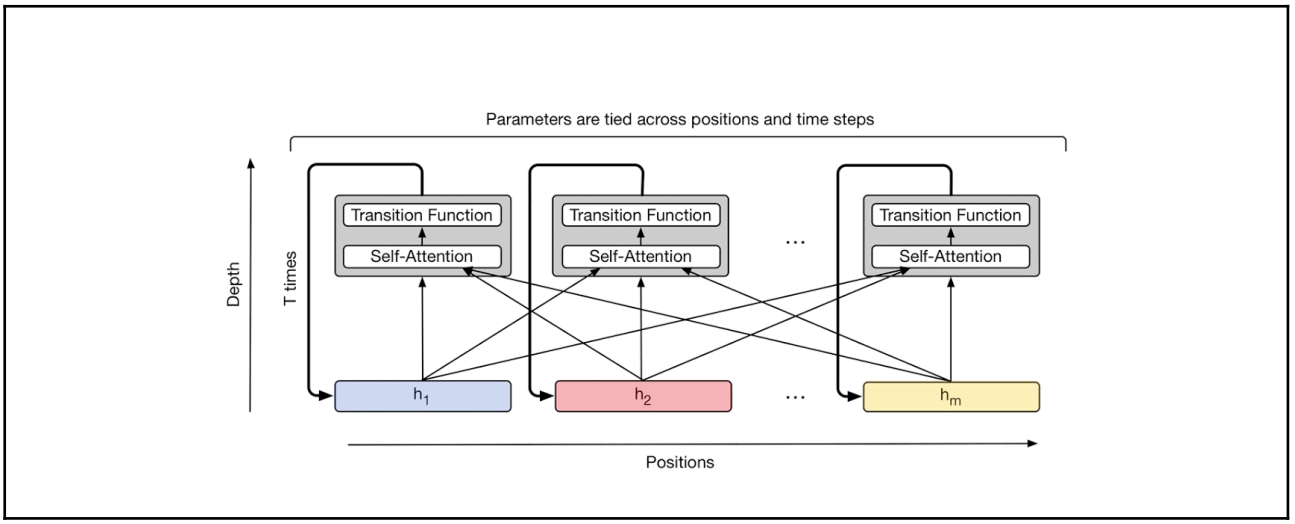


Figura 2.4: Arquitectura de un *Transformer*, donde se puede apreciar los mecanismos de atención.

2.3 Generación de imágenes.

Ahora veremos otro campo dentro de la Inteligencia Artificial, el de la generación de imágenes. Este campo es muy amplio y diverso, por lo que nos centraremos en el modelo usado en este trabajo: los **Variational AutoEncoders (VAEs)**, que son modelos de aprendizaje que mezclan las redes neuronales con distribuciones de probabilidad. Además, la diferencia que introducen las VAEs respecto a otros modelos de generación es que, en lugar de generar datos al azar, permite también influir la dirección en la que explorar las variaciones posibles.

Mediante redes neuronales, un VAE (**Figura 2.5**) se compone de:

- Un **encoder** (codificador), que transforma sus entradas en una representación intermedia, normalmente de dimensión muy inferior a la entrada, con el fin de obligarle a aprender una compresión eficiente que extraiga las propiedades principales de los datos de entrada.

- Un **decoder** (decodificador), cuya entrada es la salida del encoder para recuperar, en la medida de lo posible, la entrada original.
- Una **función de pérdida**, que mide cuánto se parece una salida de la red a la entrada.

El entrenamiento de un VAE consiste en entrenar simultáneamente al *encoder* y al *decoder* con el fin de acoplar el comportamiento de una a la otra, usando la función de pérdida como director del entrenamiento de manera que el error entre la entrada y la salida del *autoencoder* se vaya minimizando.

A lo largo de este entrenamiento, se irá definiendo lo que se llama el **espacio latente**, que es el mecanismo donde se codifica la información relevante de los datos de entrada. En este espacio tendríamos diferentes áreas dedicadas a las representaciones intermedias que genera el *encoder* donde, por ejemplo, habría un área que representa a los perros, otra para los gatos u otra para la vegetación, dependiendo de qué datos hayan sido alimentados al *autoencoder*. De este modo, generar nuevos datos con un *autoencoder* entrenado consistiría en escoger un punto aleatorio de este espacio latente y pasárselo al *decoder*, el cual tratará de generar una imagen que concuerde con la representación intermedia escogida.

Sin embargo, lo que diferencia a un VAE de otros *autoencoders* es que el espacio latente se interpreta como una distribución Gaussiana de probabilidad, ya que las representaciones intermedias vienen dadas en este caso por un vector de medias y otro vector de desviaciones típicas (**Figura 2.6)**. Gracias a esto, el *decoder* puede asociar áreas completas a ligeras variantes de la misma salida, en lugar de puntos aislados como ocurría en los *autoencoders* clásicos, y así conseguir un espacio latente mucho más suave e interpolado que es capaz de producir salidas que comparten propiedades de entradas diversas ([3], [4]).

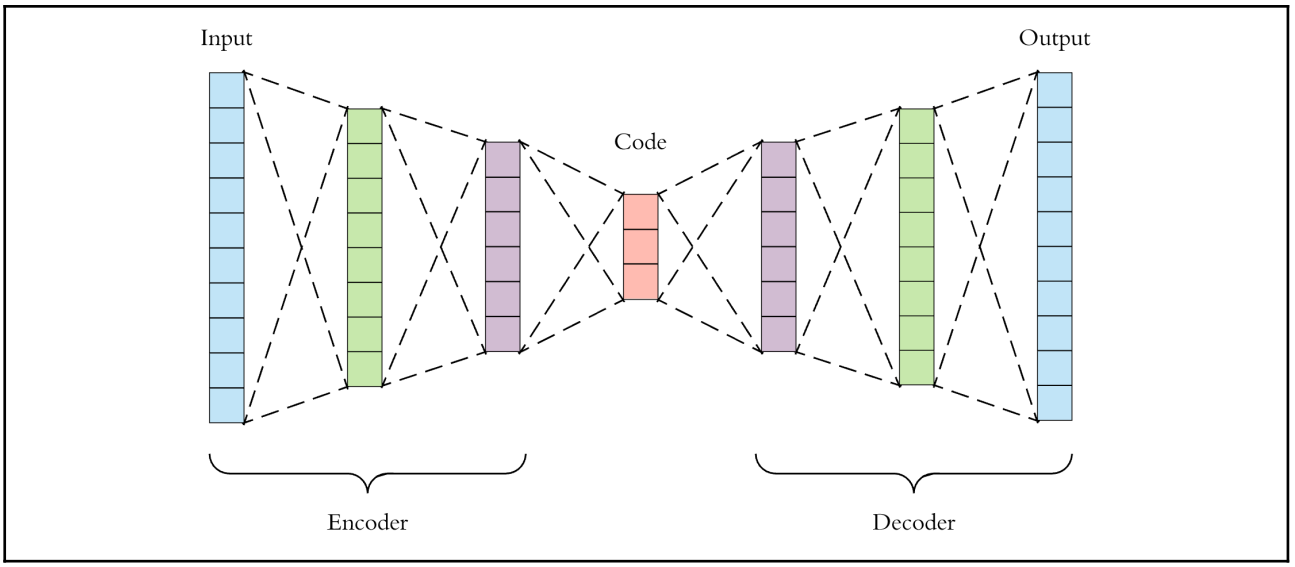


Figura 2.5: Estructura general de un *autoencoder.* Del *Blog de Fernando Sancho Caparrini* [4].

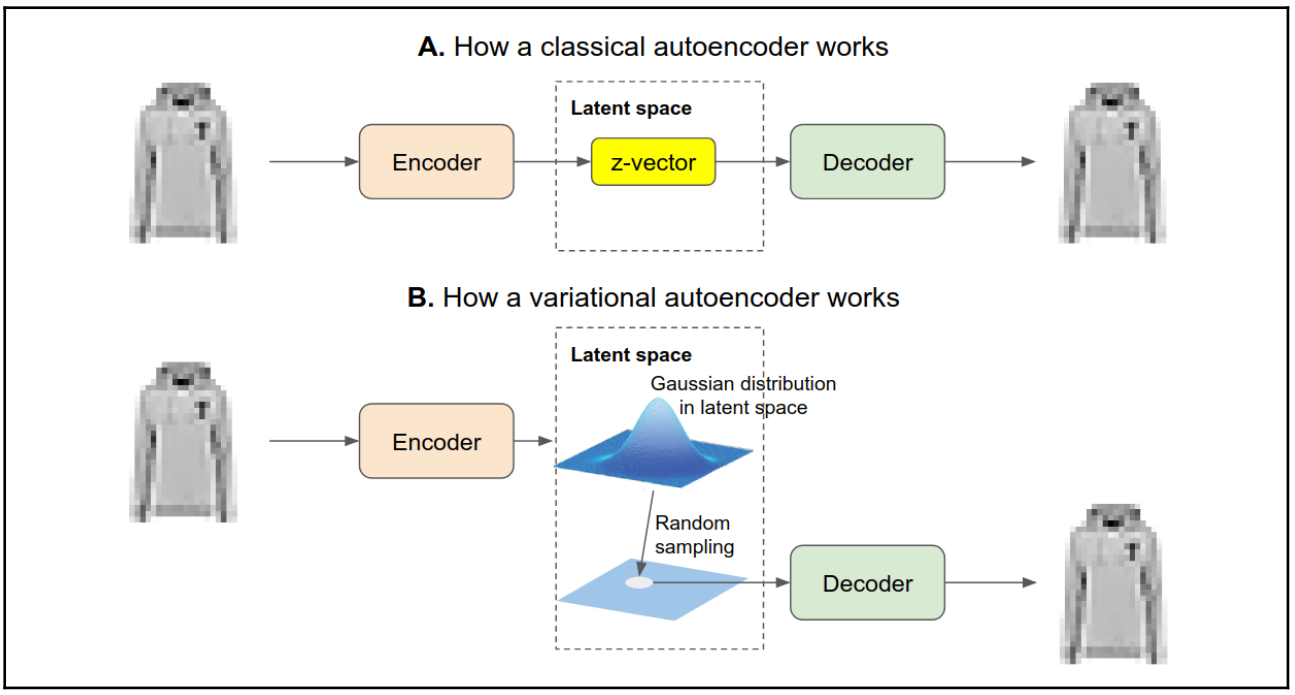


Figura 2.6: Comparación entre un *autoencoder* clásico y un VAE. Del *Blog de Fernando Sancho Caparrini* [4].

2.4 DALL-E.

A continuación veremos finalmente el modelo en el que se ha basado principalmente este trabajo: DALL-E. Se trata de una Inteligencia Artificial *Text-to-Image* basada en *transformers* desarrollada por OpenAI, por lo que sus componentes están fuertemente relacionados con su otro modelo de lenguaje, GPT-3. En el momento de su publicación, supuso una revolución para el campo de la generación de imágenes, produciendo resultados altamente realistas y consistentes con la descripción de entrada (**Figura 2.7**).

Uno de los componentes más importantes de DALL-E es su VAE (**Figura 2.8**), ya que es realmente quien se va a ocupar de generar las imágenes. Este VAE tiene la particularidad de tener un **codebook**, que es básicamente una lista de vectores asociados a un índice específico. Se usa para que, al obtener una representación intermedia en el espacio latente como salida del *encoder*, ésta se compara con cada uno de los vectores del *codebook* y se escoge la más similar, tras lo cual se puede alimentar como entrada al decoder para obtener una imagen acorde a lo que representan los vectores escogidos.

Más específicamente, la imagen de entrada se divide en una rejilla de 32x32 regiones, a la que el VAE tiene que encontrar un ***token* visual** (es decir, un vector del *codebook*) para cada una de estas regiones. De esta forma, estos *tokens* servirían como descripciones de la imagen a bajo nivel y al estar divididas en regiones es fácil codificar la información relativa a la posición de estas descripciones en la imagen.

Por otro lado, tenemos también la parte del *transformer* de DALL-E el cual, en primer lugar, se encargará de codificar en ***tokens* textuales** las descripciones en lenguaje natural, que vendrán definidas por un **vocabulario** (similar al *codebook*, pero para *tokens* de texto en este caso). A continuación, el trabajo del *transformer* será convertir estos *tokens* textuales en *tokens* visuales, de manera que se puedan alimentar al *decoder* del VAE y éste pueda generar una imagen que se corresponda con la descripción inicial [5].

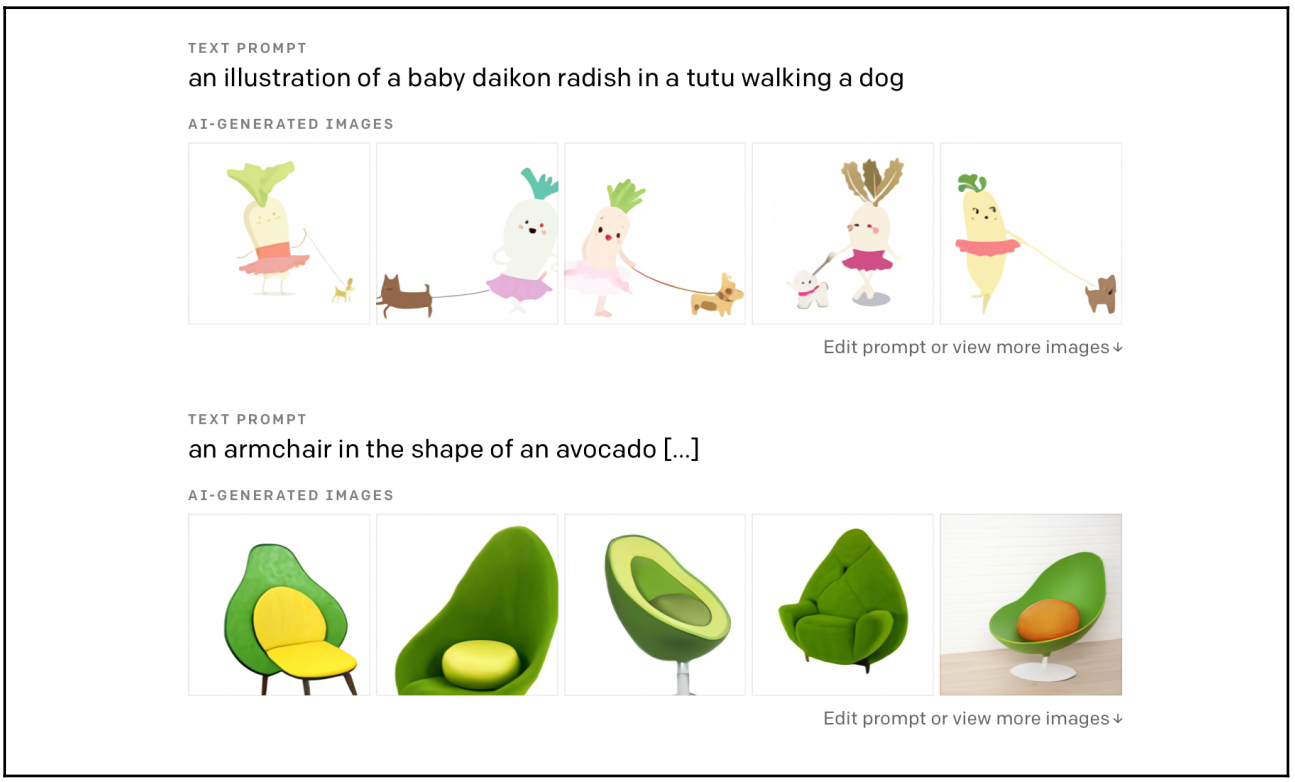


Figura 2.7: Ejemplos de resultados de DALL-E. Del blog *DALL·E: Creating Images from Text* [6].

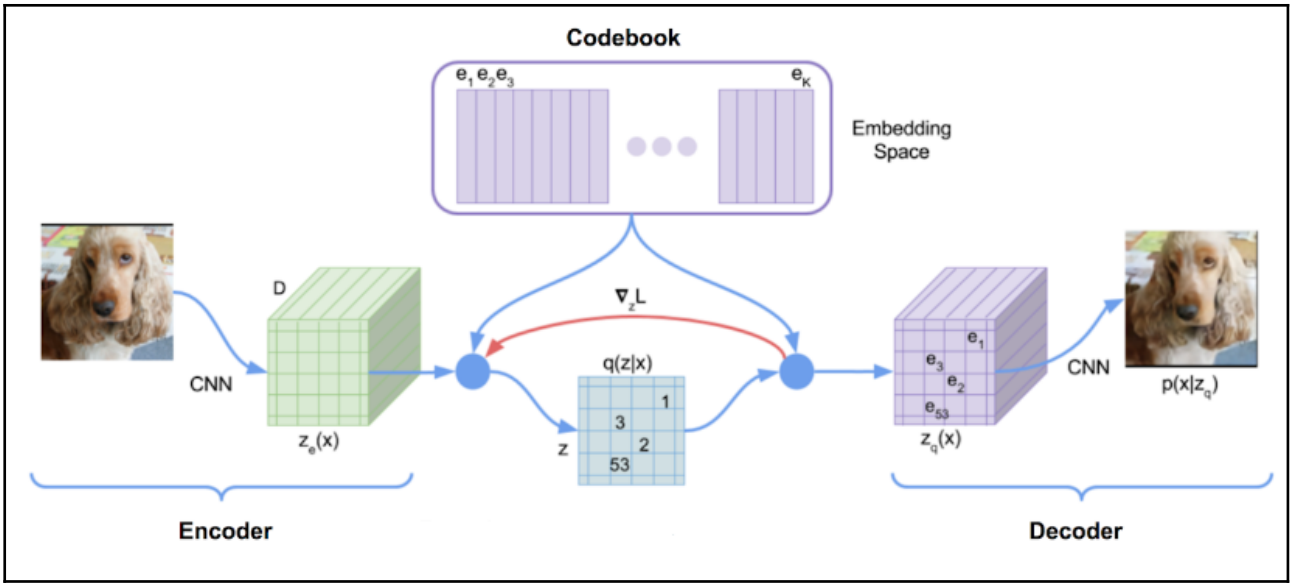


Figura 2.8: Arquitectura del VAE usado por DALL-E. De *Zero-Shot Text-to-Image Generation* [1].

2.5 Estado del arte.

Desde la publicación de DALL-E en enero de 2021, la investigación no ha cesado y se han publicado nuevos modelos que lo han superado nuevamente, como por ejemplo DALL-E 2 [7] como ampliación del mismo, capaz de generar imágenes de mayor resolución y con bastante más realismo, o IMAGEN [8], el competidor directo de DALL-E desarrollado por Google, cuyos resultados ofrecen mayor calidad incluso que DALL-E 2

(**Figura 2.9**, **Figura 2.10**).

Sin embargo, por motivos de seguridad, ninguno de estos modelos ha estado completamente disponible para uso público. Es por este motivo que muchos investigadores independientes han estado tratando de replicar las IAs mencionadas, desarrollando y publicando el código necesario para entrenar uno mismo un modelo de estas características, o al menos una aproximación de estos. A fecha de redacción de este documento, se ha terminado de desarrollar la implementación de DALL-E 1 en Pytorch y sigue en desarrollo DALLE-2 e IMAGEN, razón por la cual en este trabajo se ha optado por utilizar DALL-E 1.

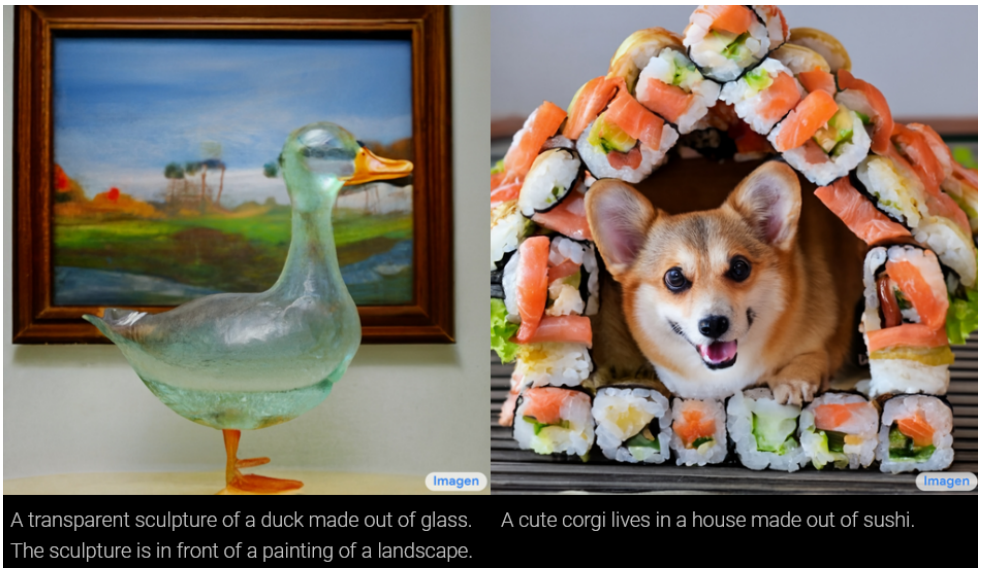


Figura 2.9: Resultados de IMAGEN. Del blog *Imagen: Text-to-Image Diffusion Models* [9].

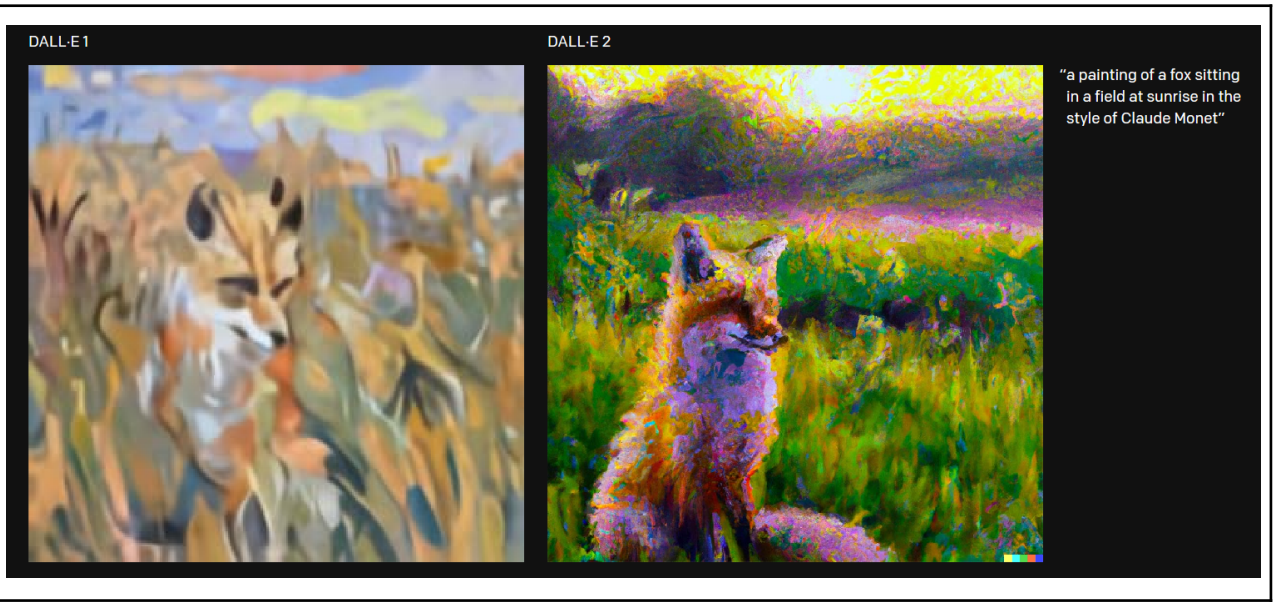


Figura 2.10: Comparativa entre DALL-E y DALL-E 2. Del blog *DALL-E 2* [10].

Capítulo 3 Metodología.

3.1 Tecnología.

La principal pieza de tecnología utilizada en este trabajo es **dalle-pytorch** [11], una replicación de DALL-E 1 en Pytorch cuyo código está abierto para poder entrenar desde cero un modelo con una estructura lo más parecida posible al modelo original de OpenAI.

Dado que este modelo está implementado en Python con Pytorch, la mayor parte del código adicional estará escrito también en este lenguaje, concretamente en la versión 3.8.5. En estos programas de Python se han utilizado diversas librerías, con las más relevantes enumeradas a continuación:

- **Pytorch** [12]: Proporciona funcionalidades para la implementación de redes neuronales, sirviendo de base para la implementación de *dalle-pytorch*.
- **OpenCV** [13]: Se trata de una librería de visión artificial, aunque se ha utilizado únicamente para pequeñas operaciones con imágenes, concretamente para el post-procesado de las imágenes generadas.
- **Beautiful Soup** [14]: Permite extraer información de contenido en formato HTML, por lo que se ha usado como *web scraper* para obtener las imágenes y descripciones del *dataset*.

Por otro lado, también se ha utilizado JavaScript, HTML y CSS para la implementación de la aplicación web gráfica para visualizar las *skins* en modelados 3D. Para esto ha sido de gran utilidad **skinview3d** [15], un módulo de JS que proporciona funciones para crear fácilmente un visualizador de *skins* de *Minecraft*, y **Bootstrap**, un framework de CSS para implementar rápidamente buenos estilos visuales en la página web.

3.2 *Skins.*

Como se ha introducido en el primer capítulo de este documento, las *skins* [16] que se generarán son imágenes de 64 por 64 píxeles (en el formato actual) del modelo del jugador de forma desplegada, donde se mapean cada una de las partes de su cuerpo.

Cada una de las partes del cuerpo debe estar definida en una posición específica, las cuales se pueden consultar en la **Figura 3.1**. Además, las áreas marcadas como “Second Layer” (Segunda Capa) y “Helmet” (Casco) hacen referencia a una segunda capa que se aplica sobre el modelo del jugador, de manera que quede superpuesta, dotando al modelo de cierta profundidad extra y que se pueda imitar, por ejemplo, prendas de ropa o diversos peinados. Estas partes de segunda capa son opcionales y, por tanto, los píxeles que las comprenden pueden ser completamente transparentes o completamente opacos.

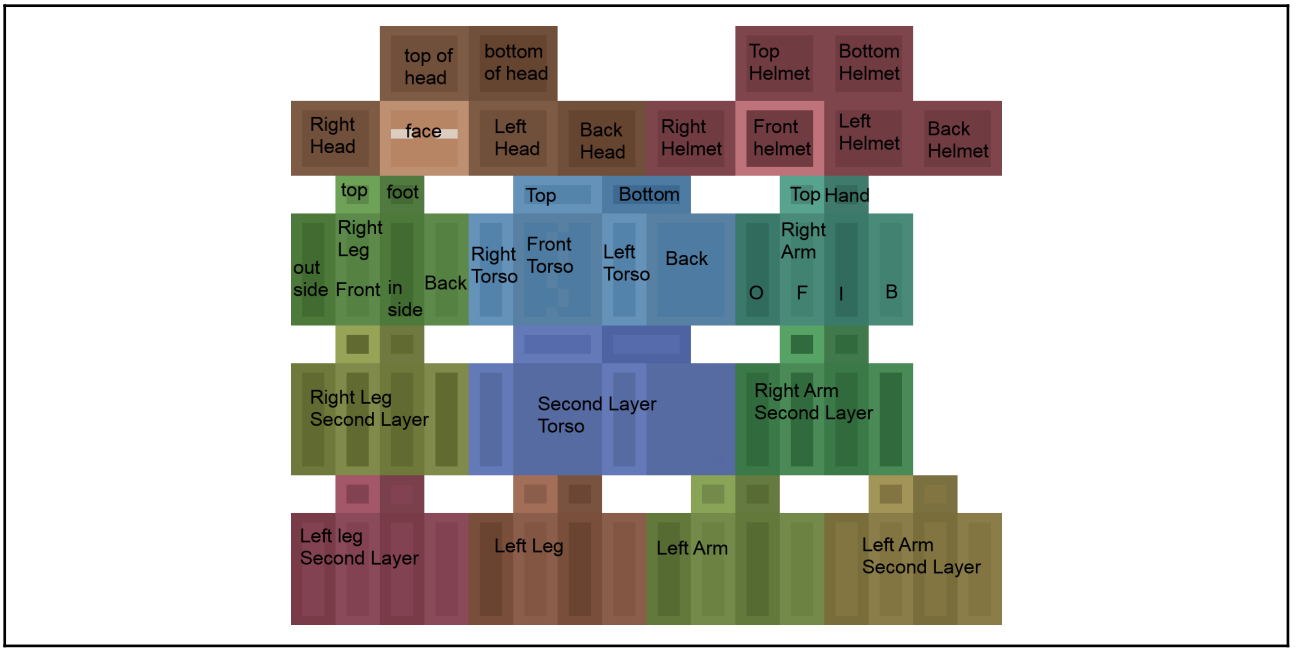


Figura 3.1: Mapa de partes del cuerpo de una *skin*.

3.3 *Dataset.*

Una de las primeras fases del desarrollo consistió en la búsqueda de un buen conjunto de imágenes de *skins* de *Minecraft*, las cuales tendrían que tener asociadas una descripción de las mismas en un archivo de texto. Tras no conseguir encontrar *datasets* ya construidos que cumplieran con estas características, se comenzó a escribir un *script* de *web scraping* con *BeautifulSoup* en Python para construir de forma automática un *dataset* propio, sacado en este caso de la página web MinecraftSkins.net (https://www.minecraftskins.net/). Esta página es conveniente ya que, a pesar de que no tenga tanto volumen de imágenes como otras, sí que la gran mayoría vienen acompañadas de una descripción textual útil de la imagen.

En concreto, se construyeron dos versiones del mismo dataset, ambos con un total de 2042 imágenes: uno de pares de imágenes y sus descripciones para el entrenamiento de DALL-E, y otro solamente de imágenes pero estructuradas en diferentes categorías y subcategorías para el entrenamiento del VAE. Estas categorías fueron obtenidas automáticamente desde la página web, ya que en ella ya se clasifican las *skins* en esas 7 categorías principales, a las que se añadieron manualmente subcategorías para que el VAE pudiera distinguir mejor distintas temáticas.

Como se puede observar en la **tabla 5.1**, realmente el *dataset* construido no está distribuido equitativamente, teniendo mucha más presencia las *skins* relacionadas con personas (tanto en la categoría “Personas” como en las subcategorías de “Gente” en otras categorías). Esto tendrá repercusiones a la hora de analizar los resultados obtenidos, los cuales veremos más adelante.

Categoría	Subcategoría	Tamaño
Fantasía	Criaturas	113
	Dioses	7

Categoría	Subcategoría	Tamaño
	Magos	24
	Gente	54
	Realeza	10
	Ciencia ficción	36
	Superhéroes	9
	Guerreros	53
	Total	**306**
Minecraft	Animales	14
	Creepers	33
	Ender	11
	Slimes	9
	Aldeanos	7
	Zombies	18
	Otros monstruos	21
	Total	**113**
Videojuegos	Antigüedad	48
	Futurista	43
	Personas	92
	Criaturas	71
	Total	**254**
Películas	Antigüedad	25
	Futurista	33
	Personas	49
	Criaturas	30
	Total	**137**
TV	Antigüedad	9
	Futurista	7

Categoría	Subcategoría	Tamaño
	Personas	53
	Criaturas	44
	Total	**113**
Personas	Mujeres	430
	Hombres	435
	Celebridades	42
	Total	**907**
Otros	Animales	64
	Personajes	80
	Objetos	34
	Superhéroes	34
	Total	**212**
Total	-	2042

Tabla 3.1: Distribución de las *skins* en categorías y subcategorías.

3.4 Pruebas iniciales y ajustes.

El entrenamiento de *dalle-pytorch* consta de dos partes, puesto que tiene dos grandes componentes: el VAE y DALL-E, el *transformer*. Primeramente hay que entrenar al VAE con el *dataset* separado en categorías descrito anteriormente, para luego añadirlo entrenado a DALL-E y que se pueda entrenar con descripciones de texto.

Antes de entrenar al modelo completamente, se probó con diversas configuraciones durante poco tiempo de entrenamiento para comprobar que el camino tomado era el correcto. En primer lugar, se probó a usar un VAE pre-entrenado, aunque dio pésimos resultados debido a que las imágenes con las que había sido entrenado eran completamente diferentes (**Figura 3.2**).

Después de esto, se comenzaron a hacer pruebas entrenando tanto al VAE como a DALL-E, aunque también surgieron diversos problemas. Uno de ellos fue que **no todas las imágenes del *dataset* tenían el mismo formato**, ya que algunas eran algo antiguas y seguían un formato de 64x32 píxeles con una distribución de las partes del cuerpo distintas, por lo que a las redes neuronales les costaría algo más comprender la estructura de las *skins* correctamente. Esto llegó a solucionarse con un pequeño *script* que aplicaba una transformación a todas las imágenes que siguieran el antiguo formato, modificándolas para que tuvieran dimensiones de 64x64 y cambiando de lugar las partes del cuerpo para adaptarlas al nuevo formato (**Figura 3.3**).

Otro de los problemas encontrados estuvo relacionado con la **transparencia** y es que, como hemos mencionado anteriormente, muchos de los píxeles de una *skin* pueden

ser transparentes y de hecho, muchos otros lo son obligatoriamente. El problema viene con que el modelo *dalle-pytorch* no estaba preparado para trabajar con imágenes con el canal *Alpha*, el que proporciona la transparencia a imágenes RGBA, sino que se entrena y genera únicamente imágenes en RGB. Tras tratar de solucionar esto por cuenta propia, se estableció contacto con el desarrollador de *dalle-pytorch* a través de GitHub y, con su ayuda, conjuntamente actualizamos el repositorio para añadir soporte a imágenes con transparencia.

Además, en estas primeras pruebas pudimos observar que muchas veces lo generado no se ajustaba con el texto de entrada, por lo que se decidió que el *dataset* debería ser ampliado en cuanto a sus descripciones textuales. Para automatizar esto en cierta medida, en primer lugar se añadieron descripciones generales a cada categoría y subcategoría del *dataset* para el VAE, para luego con ayuda de un *script* añadir estas descripciones al otro *dataset* a cada *skin* que se encontrara en la categoría correspondiente. Tras esto, se añadieron descripciones manualmente a cerca de la mitad de las *skins* del *dataset*, las cuales se centraban en describir físicamente a la *skin*, haciendo énfasis en los colores y las partes del cuerpo.

Figura 3.2: Resultados al generar una *skin* con un VAE pre-entrenado.

Figura 3.3: Comparativa entre una *skin* con el antiguo formato (izq.) y su transformación en el nuevo formato.

3.5 Entrenamiento.

Finalmente, con los mencionados problemas solucionados, se pudo dejar largo tiempo a las redes neuronales entrenando. Como se mencionó anteriormente, lo primero es entrenar al VAE, que fue entrenado usando los *scripts* proporcionados por el repositorio de *dalle-pytorch*, con ciertos ajustes para adaptarse a imágenes de 64x64 con transparencia. Este entrenamiento consistió en 200 etapas (unas 51.000 iteraciones en total), durando cerca de 3 horas con el *hardware* descrito en la **tabla 3.2**.

A continuación llegó el momento de entrenar directamente a DALL-E, esta vez durante 500 etapas (unas 134.000 iteraciones). Este entrenamiento es más costoso en recursos que el anterior, por lo que su duración ha rondado las 24 horas aproximadamente.

Una vez terminado el entrenamiento, disponemos de un archivo llamado *dalle.pt* (además de otro *vae.pt* para el VAE) el cual contiene todo el modelo ya entrenado y podemos utilizarlo para generar imágenes. Con este fin, se implementó el fichero *generate.py*, basado en gran medida en el fichero del mismo nombre que provee el repositorio de *dalle-pytorch*, que encapsula en una función la generación de *skins*, recibiendo como parámetros la descripción textual de entrada y la ruta a *dalle.pt*.

Para tener una noción sobre cómo se realiza el entrenamiento a través de la librería de *dalle-pytorch*, podemos fijarnos en la **figura 3.4** que muestra un ejemplo simplificado (tan solo sería una iteración) del código necesario para entrenar tanto al VAE como a DALL-E.

```
import torch
from dalle_pytorch import DiscreteVAE

vae = DiscreteVAE(
    image_size = 64,
    num_layers = 3,           # Número de capas del VAE
    num_tokens = 8192,        # Número de tokens visuales
    codebook_dim = 512,       # Dimensión del codebook
    hidden_dim = 64,          # Dimensión oculta del VAE
    num_resnet_blocks = 1,    # Número de bloques de ResNet
    temperature = 0.9,        # Temperatura Gumbel Softmax
    straight_through = False, # Método de paso directo para el Gumbel
Softmax
)

# Ruido como ejemplo para la entrada
images = torch.randn(4, 3, 256, 256)

# Entrenamiento del VAE
loss = vae(images, return_loss = True)
loss.backward()
```

```
import torch
from dalle_pytorch import DiscreteVAE, DALLE

vae = DiscreteVAE(
    image_size = 256,
    num_layers = 3,
    num_tokens = 8192,
    codebook_dim = 1024,
    hidden_dim = 64,
    num_resnet_blocks = 1,
    temperature = 0.9
)

dalle = DALLE(
    dim = 1024,
    vae = vae,                  # Mismo VAE que el que se entrenó.
    num_text_tokens = 10000,    # Tamaño del vocabulario de texto.
    text_seq_len = 256,         # Longitud (máxima) de la secuencia de
texto.
    depth = 12,                 # Profundidad de DALLE.
    heads = 16,                 # Cabezas de Atención.
    dim_head = 64,              # Dimensiones de cada cabezal.
    attn_dropout = 0.1,         # Dropout de la atención.
    ff_dropout = 0.1            # Dropout de feed-forward.
)

# Ruido como ejemplo para la entrada
text = torch.randint(0, 10000, (4, 256))
images = torch.randn(4, 3, 256, 256)

# Entrenamiento del DALL-E
loss = dalle(text, images, return_loss = True)
loss.backward()

# Tras entrenar durante un largo periodo de tiempo, se puede generar una
imagen:
images = dalle.generate_images(text)
```

Figura 3.4: Código para entrenar al VAE (arriba) y para entrenar a DALL-E (abajo).

Componente	Especificaciones

Tarjeta Gráfica	NVIDIA RTX 2060 6GB
Procesador	AMD Ryzen 5 3600 3,2GHz
Memoria	16 GB 3,2GHz

Tabla 3.2: Lista de componentes relevantes usados en el entrenamiento.

3.6 Post-procesado.

Como se puede entrever con el resultado expuesto en el apartado anterior, aún con soporte para transparencias, las imágenes generadas no se ajustaban del todo al formato requerido. En las *skins*, sus píxeles pueden ser o bien completamente transparentes o bien completamente opacos (es decir, píxeles con el canal *Alpha* a 0 o a 255, sin ningún valor entre medio); sin embargo, las imágenes que se generaban contenían píxeles con valores intermedios de transparencia. Además, algunas veces se dibujaban píxeles opacos en regiones donde deberían haber píxeles transparentes. Para solucionar esto, se implementó un *script* de **post-procesado**, el cual se lleva a cabo siguiendo los siguientes pasos para cada *skin*:

1. Despejar las zonas que deben estar completamente transparentes.
2. Eliminar toda transparencia del cuerpo (primera capa), puesto que éste es siempre opaco.
3. Establecer un umbral a partir del cual, si el valor en el canal *Alpha* cada pixel de la segunda capa de la *skin* lo supera, se aplica el valor máximo (255). Si no lo supera, se establece el mínimo (0).

Estas operaciones se han implementado en el fichero *postprocess.py*, en el cual se definen una función para cada uno de los pasos mencionados y una función principal que, dada la ruta al directorio de *skins* a post-procesar, llama a las demás funciones para aplicarles el post-procesado a cada *skin*.

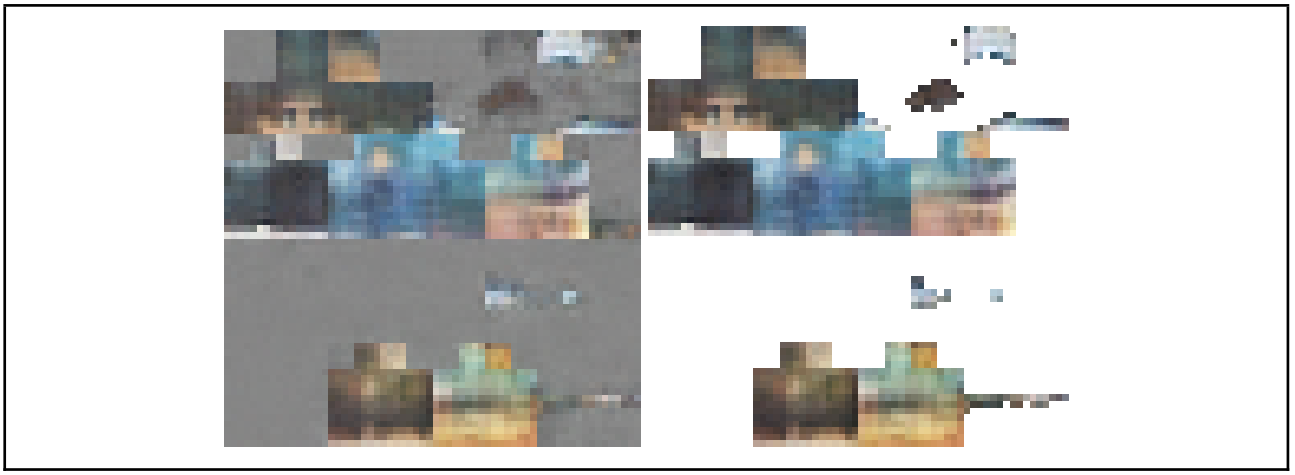


Figura 3.5: Comparación con una *skin* antes y después de aplicar post-procesado.

3.7 Aplicación gráfica.

Con el fin de poder visualizar claramente los resultados que se obtengan con este modelo, y una vez terminado el entrenamiento y el desarrollo troncal de este proyecto, se

comenzó con la implementación de una pequeña aplicación web que terminaría de dar forma al trabajo realizado.

Para visualizar las *skins* correctamente en un modelo tridimensional, se ha optado por utilizar ***skinview3d*** y se empezó a escribir el código HTML, CSS y JavaScript que daría soporte a una sencilla página web que consistiría en un encabezado donde se muestra la descripción textual que hayamos pasado como entrada a DALL-E, un canvas principal donde se muestra la *skin* que hayamos seleccionado y donde podemos rotarla o hacer zoom para examinarla, y diversas imágenes debajo, una por cada *skin* generada por DALL-E a las que podremos hacer click para posicionarla en el canvas principal (**Figura 3.6**).

Para poder ver esta página web en un navegador correctamente, fue necesario añadir un *script* que ejecutaría un servidor local. Este *script* llamado *app.py* fue implementado en Python, para así poder llamarlo fácilmente junto con los otros programas de este trabajo.

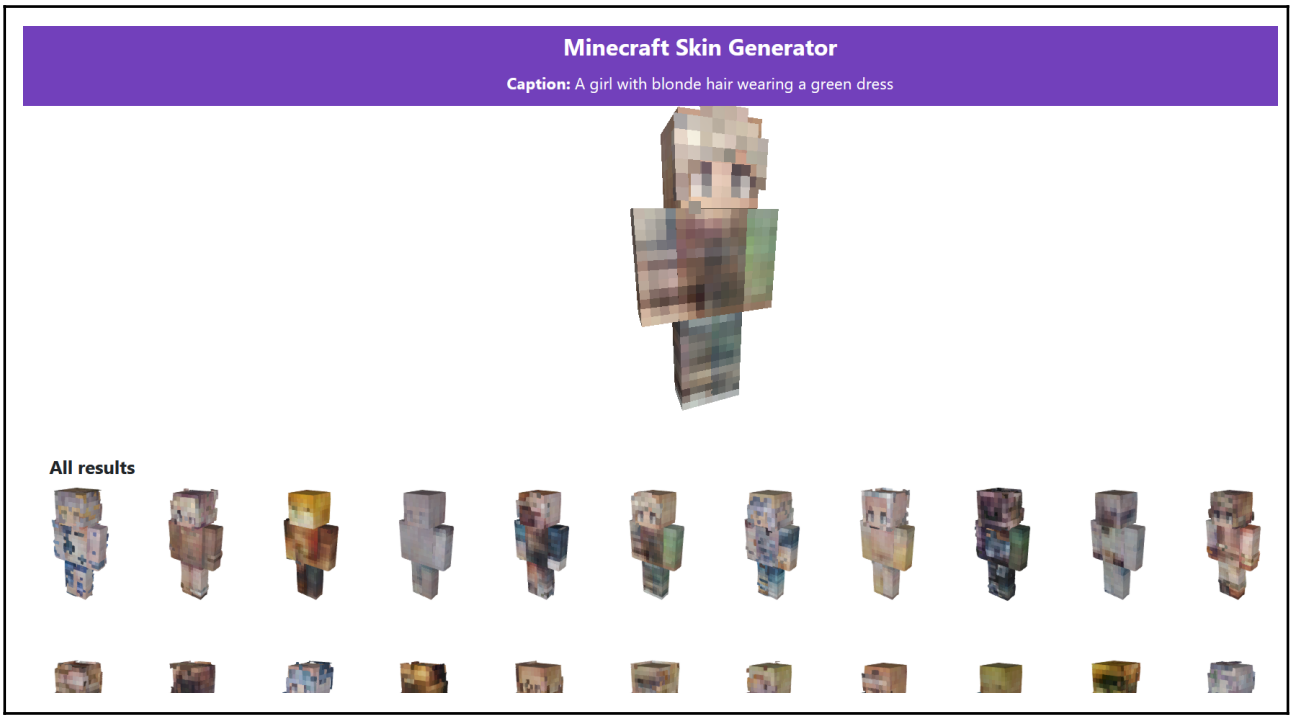


Figura 3.6: Vista de la aplicación web.

3.8 Ajustes finales.

Finalmente, se terminaron haciendo retoques en el código para integrar todas las diferentes partes de este trabajo: la generación de imágenes, el post-procesado y la aplicación gráfica. Para esto, se escribió un fichero *main* en Python, el cual se puede usar de la siguiente manera:

```
usage: main.py [-h] [--dalle DALLE] [--no_viewer] --text TEXT

Generate and postprocess skins.

optional arguments:
```

```
-h, --help    show this help message and exit
--dalle DALLE  The path to dalle.pt
--no_viewer  Doesn't open the viewer app.
--text TEXT  The text prompt
```

Este programa se encargaría, en primer lugar, de llamar a la función que genera imágenes en *generate.py*, pasándole como entrada el texto que se haya especificado en el argumento *–text*. Esta función crearía un nuevo directorio de las 64 imágenes generadas, junto con un fichero *caption.txt* que contiene el texto de entrada. Después, se llama a la función de *postprocess.py* para así corregir todas las imágenes que se encuentran en el directorio generado. Ya por último, copiamos estas *skins* a una carpeta dentro del módulo de la aplicación gráfica y ejecutamos el servidor local con *app.py*, tras lo cual se abrirá una ventana del navegador con el visualizador de *skins*, mostrando todas las que acaban de ser generadas.

Todo el código desarrollado se ha organizado en un repositorio en GitHub (https://github.com/alu0101130507/Minecraft-Skin-Generator), el cual contiene tanto los programas que se han mencionado (incluyendo los *scripts* de preprocesado, postprocesado, generación de imágenes…), como los *datasets* utilizados y el código de la aplicación gráfica. Por otro lado, debido al límite de almacenamiento de GitHub, se ha dispuesto un enlace a Google Drive con los modelos entrenados del VAE y DALL-E.

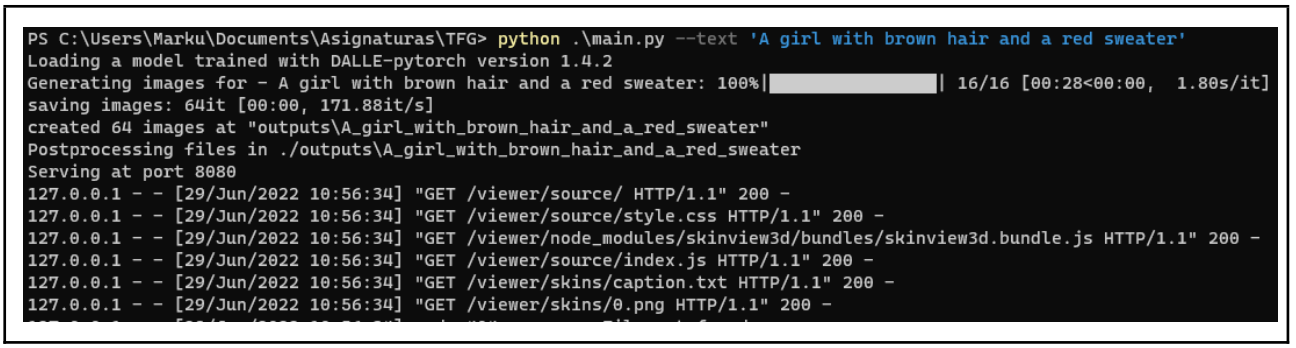


Figura 3.7: Ejemplo de ejecución del programa principal.

Capítulo 4 Resultados.

Dada la naturaleza de este trabajo, los resultados serán analizados cualitativamente, juzgando tanto el realismo de las imágenes generadas como su adecuación a las descripciones asociadas. Como veremos, existen cierto tipo de entradas de texto que producen mejores resultados que otros, por lo que detallaremos exactamente cuáles son estos, por qué sucede y cuáles son los patrones visuales que mejor ha aprendido a reproducir la red neuronal. En primer lugar, examinaremos la evolución de los resultados a lo largo del tiempo de entrenamiento, tanto para el VAE por un lado como para DALL-E al completo por otro.

4.1 Evolución.

En esta sección revisaremos la evolución de las imágenes que se generan a lo largo del paso de las iteraciones de entrenamiento del VAE y de DALL-E. Hay que tener en cuenta que los resultados mostrados en esta sección son su versión pura tras salir de sus respectivas redes neuronales, por lo que no se le ha aplicado ninguna operación de post-procesado ni se muestran aplicadas a un modelo tridimensional debido a que no siempre cumplen con el formato requerido.

4.1.1 VAE.

Como hemos visto previamente, los VAEs se entrenan tratando de reconstruir la imagen que se le pasa al codificador, por lo que veremos las comparaciones entre las entradas y las salidas a lo largo de las diferentes iteraciones.

Fijándonos en los resultados de las primeras iteraciones en la **figura 4.1**, vemos que la red aún no está suficientemente entrenada y por tanto las reconstrucciones carecen de forma y colores. Es a partir de cerca de las 500 iteraciones cuando comienzan a formarse las estructuras de las *skins*, aunque siguen sin tener colores bien definidos. Tras miles de iteraciones se empiezan a ver buenas reconstrucciones de colores, así como de ciertos patrones específicos, como caras, pelo y ojos. Por último tenemos en esta figura los resultados de las últimas iteraciones, en los que se puede ver que sigue teniendo problema para reproducir detalles y colores específicos, aunque lo hace relativamente bien con ciertos patrones, como caras, pelo y ropa.

En conclusión, los resultados que se obtienen con este VAE producen mejores resultados cuando la imagen a generar se trata de una persona con patrones visuales comunes, como ojos, pelo o ciertas piezas de ropa. Esto era esperable, ya que como se vio en la descripción del *dataset*, este contiene en su mayor parte *skins* de personas y, por tanto, es en lo que más se ha entrenado a la red neuronal.

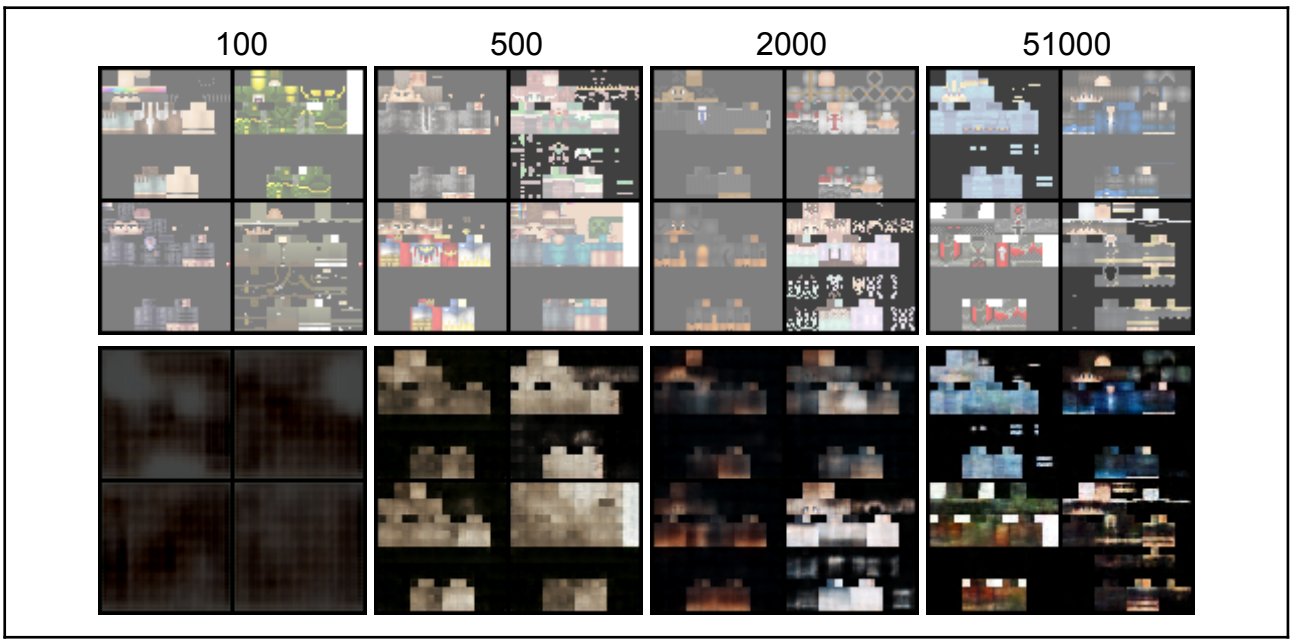


Figura 4.1: Imágenes de entrada (arriba) y reconstruidas (abajo) a lo largo de diferentes iteraciones de entrenamiento del VAE.

4.1.2 DALL-E.

A continuación veremos los resultados obtenidos a lo largo del entrenamiento de DALL-E, el cual consistía en alimentar una descripción de texto al *transformer* para predecir los *tokens* visuales de la imagen, los cuales se pasan al *decoder* entrenado del VAE para generar la imagen final. Por esto, veremos en la **figura 4.2** diferentes iteraciones con el texto de entrada y la imagen que se genera.

En las primeras iteraciones vemos que ya la red neuronal tiene capacidades generativas, aunque la imagen no siempre cuadre con la descripción. Esto se debe a que, como el VAE ya está entrenado para producir imágenes, el modelo es capaz de generar ciertas formas y patrones, y lo que tiene que aprender en este entrenamiento es a relacionar los *tokens* visuales con el texto.

A las 72000 iteraciones (cerca de la mitad del entrenamiento), podemos ver que las imágenes comienzan a cuadrar más consistentemente con las descripciones, aunque estas carezcan de detalle y funcionen casi exclusivamente con personas. Conforme avanza el entrenamiento, las imágenes van ganando más detalle y se van ajustando más a lo descrito en el texto, a pesar de que estos resultados se presenten casi solamente en aquellas descripciones textuales que sobre personas. De nuevo, esta carencia probablemente se deba a la gran presencia de personajes humanos en el *dataset*, los cuales además tienen una serie de características comunes fácilmente reconocibles: ojos, boca, pelo, ropa, etc.

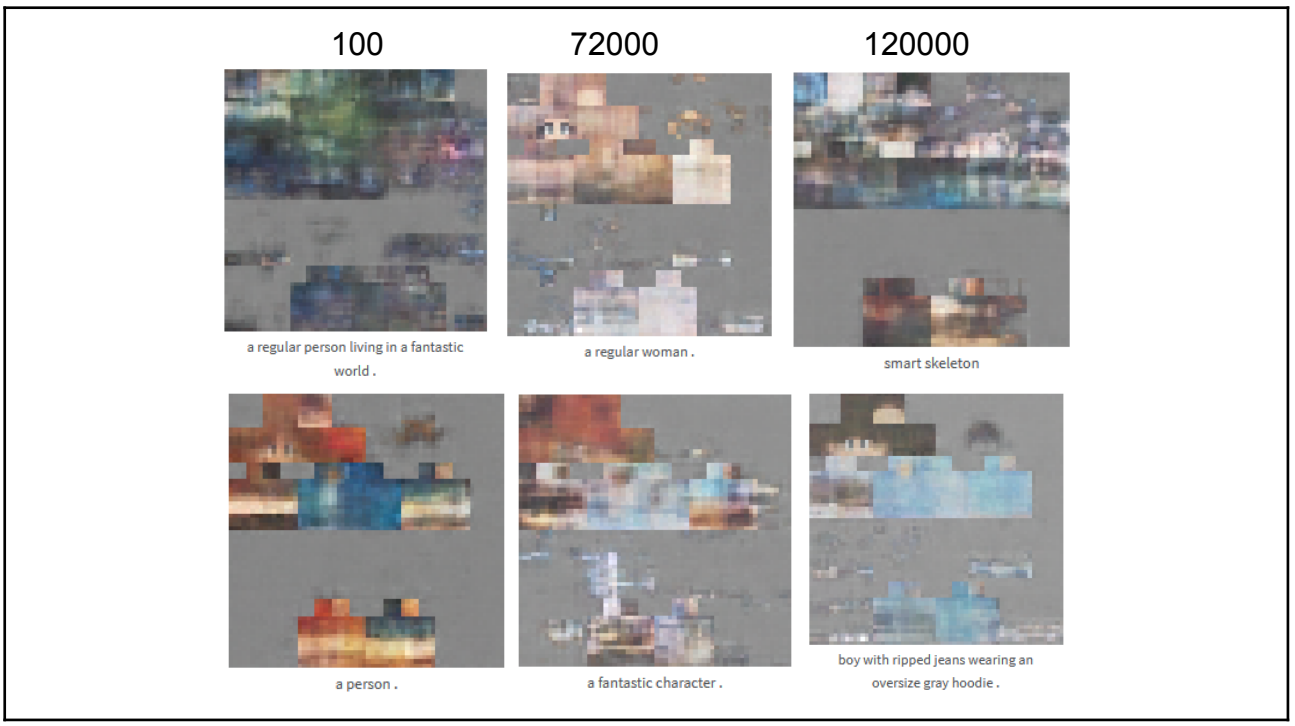


Figura 4.2: Varios ejemplos de salidas de DALL-E a lo largo de diferentes iteraciones de entrenamiento.

4.2 Patrones aprendidos.

A pesar de que hayan ciertas entradas a las que este modelo de DALL-E no pueda producir buenas imágenes, sí que existen ciertos patrones que ha ido aprendiendo a reconocer y generar, algunos de los cuales serán enumerados a continuación. A diferencia de con la anterior sección, aquí veremos las *skins* una vez pasadas por el post-procesado y en vista tridimensional, para así poder apreciar más claramente sus características.

4.2.1 Género.

Como hemos ido explicando, el principal ámbito en el que funciona este modelo de manera óptima es en el de la generación de *skins* de personas, pudiendo distinguir claramente entre géneros especificando en la entrada de texto las palabras “girl”, “woman” o “boy”, “man”, entre otras (**Figura 4.3**).

Como podemos ver, muchas de las imágenes tienen ciertos artefactos que estropean visualmente a la *skin*, pero aún así se pueden distinguir bien las diferencias entre mujeres y hombres, apreciables principalmente por el pelo y la forma de los ojos, de forma similar a los ejemplos encontrados en la base de datos. Además, podemos ver diferentes atisbos a prendas de ropa (trajes, camisetas y zapatos, especialmente).

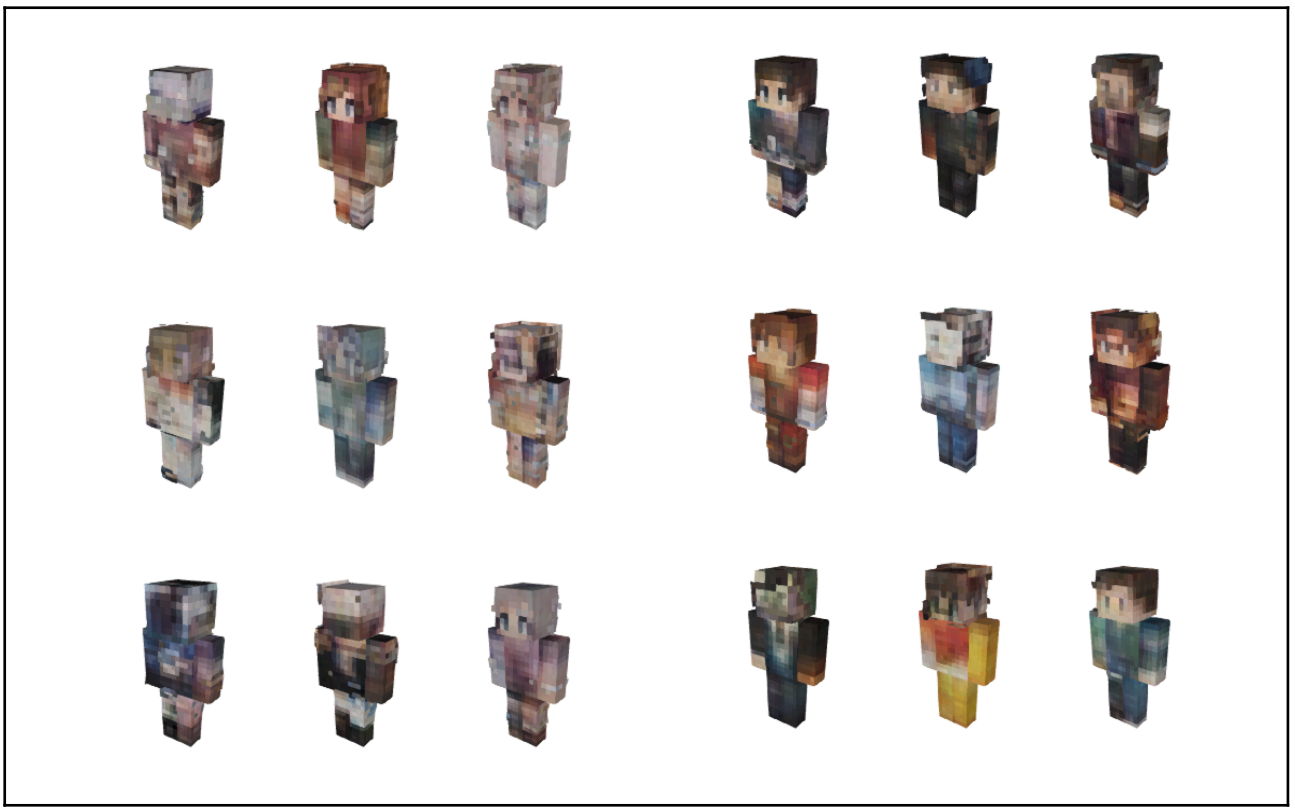


Figura 4.3: Resultados de la entrada “A girl” a la izquierda y “A boy” a la derecha.

4.2.2 Pelo.

Otra de las características reconocidas por el modelo es el pelo, pudiendo solicitar distintos colores de forma relativamente consistente. En la **figura 4.4** podemos observar dos ejemplos de una ejecución para diferentes peinados. Aunque no ocurre siempre, se puede ver una clara tendencia a respetar los colores de los peinados especificados.

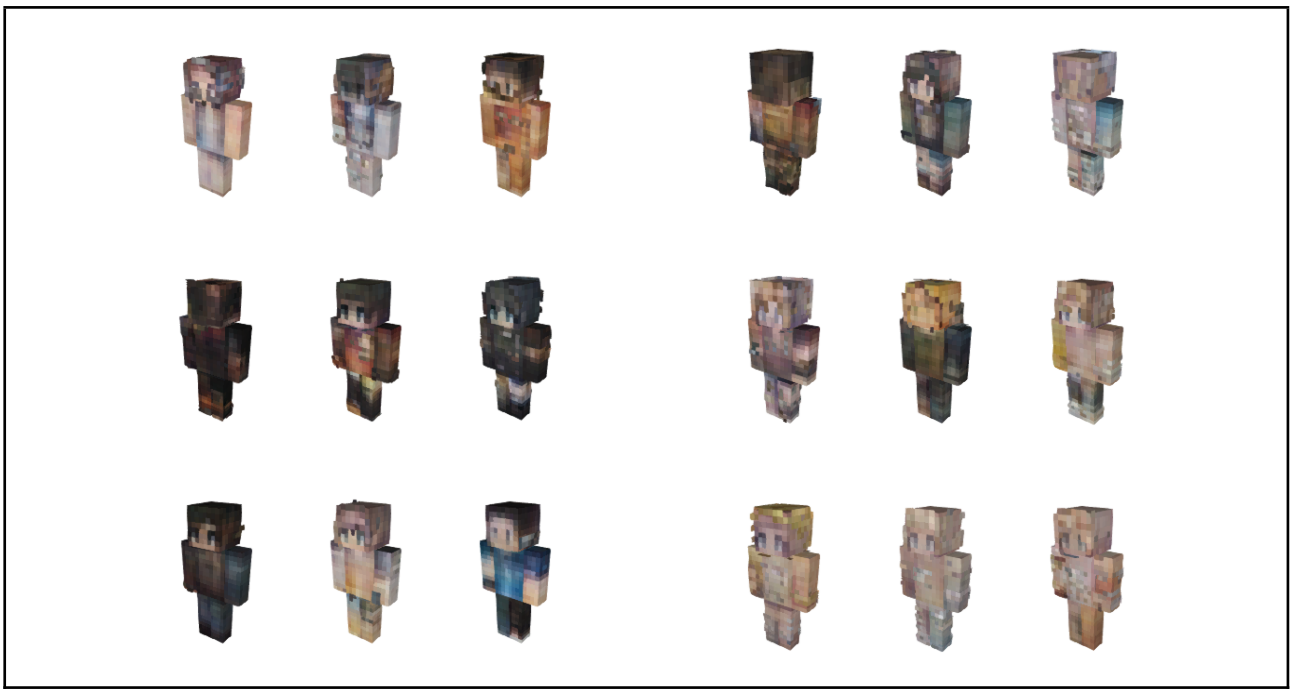


Figura 4.4: Resultados de la entrada “A girl with black hair” a la izquierda y “A girl with blonde hair” a la derecha.

4.2.3 Ropa y colores.

También es posible reproducir determinadas piezas de ropa e incluso especificar sus colores. En este caso, es menos común encontrar ejemplos que satisfagan lo que se establece en las entradas de texto, aunque dado que con cada ejecución se genera un gran número de imágenes, es relativamente fácil dar con *skins* que se ajusten a lo que uno busca.

En la **figura 4.5** podemos observar varios ejemplos de buenos resultados, aunque hay que tener en cuenta que en la ejecución para esas entradas de texto habían muchas otras *skins* que no se ajustaron a la entrada. Además, podemos ver que no siempre se ajustan literalmente a la entrada, en muchas ocasiones cambiando el color que se haya especificado.

Sin embargo, vemos que la red neuronal por lo general ha empezado a aprender este tipo de patrones, consiguiendo acertar varios colores como el blanco y el negro en estos ejemplos, e incluso acercándose al rojo, aunque más bien parezca naranja. También ha aprendido a situar algunos accesorios, como los que se muestran en la figura: brazaletes, corbatas, capuchas o cinturones.

Figura 4.5: Ejemplos de *skins* que se ajustan a una entrada de texto que especifica su ropa.

4.2.4 No humanos.

Como hemos visto en los ejemplos anteriores, el dominio principal de este modelo es la generación de *skins* de personas. Por ello, para comparar, ahora veremos algunos ejemplos en los que se soliciten imágenes de criaturas no humanas, o incluso de humanos que no muestren la cara o el pelo, llevando algún tipo de armadura que los tape,

por ejemplo.

En la **figura 4.6** podemos ver que no se suelen conseguir buenos resultados solicitando *skins* que no sean personas. Sin embargo, dependiendo de qué entradas utilicemos podemos ver ciertos patrones que sí se ajustan con las descripciones textuales. Por ejemplo, en el caso de solicitar “A green zombie”, vemos intentos de representar un personaje con la piel verde en varias ocasiones, aunque carezca de muchos detalles. Por otro lado, también encontramos que con “A red creature” la mayoría de imágenes generadas se corresponden con un personaje rojo, aunque de nuevo no tienen demasiados detalles distinguibles.

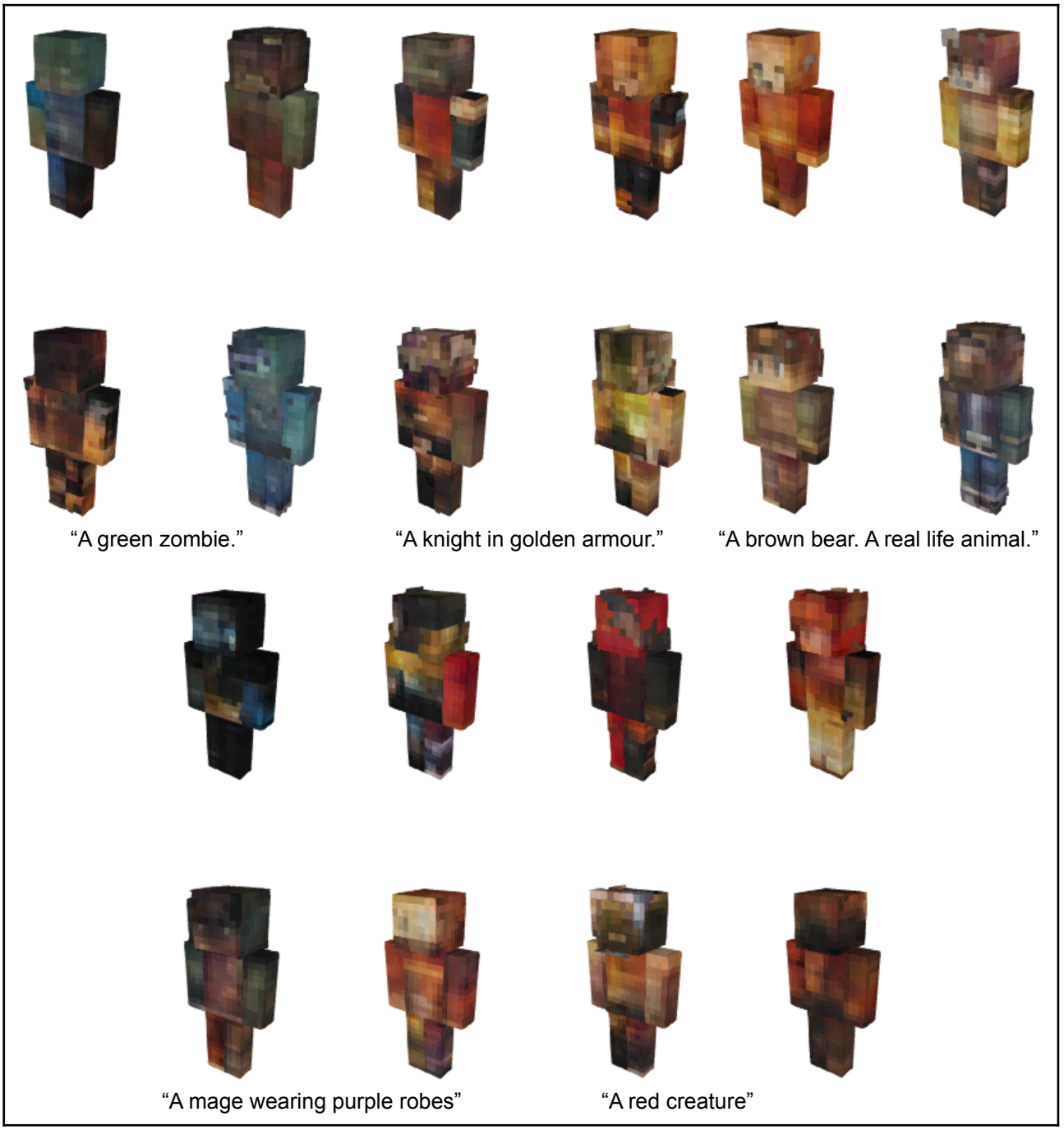


Figura 4.6: Intentos de reproducir personajes no humanos.

Capítulo 5 Conclusiones y líneas futuras

En este trabajo hemos visto el funcionamiento y los resultados de una tecnología que, aparte de ser visualmente impresionante, abre las puertas a muchas posibilidades en el futuro que nos espera. Aunque sea un inconveniente el hecho de que los modelos oficiales estén restringidos para su reentrenamiento, pudimos observar que, gracias a los esfuerzos por replicar estos modelos, también es posible conseguir resultados significativos al alcance de cualquier persona interesada en esta materia.

Desde 2021 se han publicado modelos generadores de imágenes cada vez más sorprendentes y es esperable que esta tendencia siga manteniéndose próximamente, por lo que sería interesante replicar este trabajo con IAs más avanzadas en un futuro, especialmente si se llegan a abrir los modelos oficiales. Además, resultaría de interés probar a utilizar diferentes conjuntos de datos, organizarlos de diferente manera o expandirlos, en especial con el *dataset* para el VAE, el cual, al no necesitar descripciones textuales, podría ser más fácil de expandir.

En conclusión, a pesar de que no se hayan podido conseguir resultados perfectos, este trabajo ha logrado poner de manifiesto el potencial que aguarda en los modelos *Text-to-Image*. Además, hemos visto que este modelo es capaz de brindar unos resultados favorables la mayoría de veces, en concreto con más consistencia cuando solicitamos personas con una descripción de su apariencia.

Por tanto, se ha tratado de un trabajo esclarecedor con respecto a este tipo de tecnologías, cuyos avances se han dado muy recientemente, ha servido como pretexto para desarrollar una aplicación que puede ser utilizada tanto como propósito lúdico como educativo y abre las puertas a más investigación en este campo de la Inteligencia Artificial y a más experimentación con el propósito concreto que se ha seguido en este caso.

Capítulo 6 Summary and Conclusions

In this project we have seen the results and the functioning of a piece of technology that, besides of it being visually impressive, opens the doors to many different possibilities about our future. Even though it's inconvenient that the official models are restricted for retraining, we could see that, thanks to the efforts made by people that were trying to copy these models, it's also possible to get significant results within reach to anyone who is interested in this subject.

Since 2021, more and more amazing imaging models have been published and it is expected that this trend will continue in the near future, so it would be interesting to replicate this work with more advanced AIs in the future, especially if the official models are opened. In addition, it would be of interest to try using different datasets, organizing them differently or expanding them, especially with the VAE dataset, which, not needing textual descriptions, could be easier to expand.

In conclusion, despite not being able to achieve perfect results, this project has succeeded in highlighting the potential that awaits in Text-to-Image models. Moreover, we have seen that this model is able to provide favorable results most of the time, in particular with more consistency when we request people with a description of their appearance.

Therefore, it has been an enlightening project respect to this type of technologies, whose advances have been made very recently, has served as a pretext to develop an application that can be used both as a playful and educational purpose and opens the door to further research in this field of Artificial Intelligence and more experimentation with the specific purpose that has been followed in this case.

Capítulo 7 Presupuesto.

Para la realización de este trabajo no ha sido necesaria la utilización de ninguna herramienta de pago, aunque sí que se han estado utilizando grandes cantidades de recursos energéticos para el entrenamiento de las redes neuronales, el cual se realizó localmente en un ordenador personal a lo largo de un total de 50 horas, considerando tanto el entrenamiento final como otros de prueba.

Por otro lado, también se ha de tener en cuenta el trabajo realizado en horas, el cual ha rondado las 300 horas, juntando tanto el trabajo en reuniones y seminarios como el trabajo autónomo.

Número de horas (h)	300
Coste por hora (€)	25
Total (€)	7500

Tabla 7.1: Tabla de presupuesto sobre el trabajo realizado.

Capítulo 8 Anexo: Código del post-procesado.

```
# postprocess.py
#
# Markus Schüller Perdigón
# 2022
#
# Postprocessing operations on the skins generated by DALL-E

import cv2, os, sys

def clearWhiteZones(skin):
  """
  Clears the zones that are supposed to be transparent from the skin.
  @param skin: The skin image to be cleared in opencv format.
  """
  whiteZones = [
    [[0,8], [0,8]],     # Top left corner
    [[0,8], [56,64]],   # Top right corner
    [[0,8], [24,40]],   # Top middle space
    [[16,20], [0,4]],   # Mid left space 1
    [[32,36], [0,4]],   # Mid left space 2
    [[48,52], [0,4]],   # Mid left space 3
    [[16,20], [52,56]], # Mid right space 1
    [[32,36], [52,56]], # Mid right space 2
    [[16,20], [12,20]], # Middle space 1
    [[16,20], [36,44]], # Middle space 2
    [[32,36], [12,20]], # Middle space 3
    [[32,36], [36,44]], # Middle space 4
```

```
    [[48,52], [12,20]], # Middle space 5
    [[48,52], [28,36]], # Middle space 6
    [[48,52], [44,52]], # Middle space 7
    [[16,48], [56,60]], # Big right space 1
    [[16,52], [60,64]]  # Big right space 2
  ]

  for zone in whiteZones:
    skin[zone[0][0]:zone[0][1], zone[1][0]:zone[1][1]] = [0, 0, 0, 0]

def removeBodyTransparency(skin):
  """
  Removes the transparency of the body (first layer) from the skin.
  @param skin: The skin image to be cleared in opencv format.
  """
  body = [
    [[0,8], [8,24]],    # Top and bottom head
    [[8,16], [0,32]],   # Head sides
    [[16,20], [4,12]],  # Top and bottom right leg
    [[16,20], [20,36]], # Top and bottom torso
    [[16,20], [44,52]], # Top and bottom right arm
    [[20,32], [0,56]],  # Right leg, torso and right arm sides
    [[48,52], [20,28]], # Top and bottom left leg
    [[48,52], [36,44]], # Top and bottom left arm
    [[52,64], [16,48]]  # Left leg, torso and left arm sides
  ]

  for zone in body:
    for x in range(zone[1][0], zone[1][1]):
      for y in range(zone[0][0], zone[0][1]):
        skin[y, x][3] = 255

def setBinaryTransparency(skin):
  """
  Sets the skin's second layer transparency to be either 0 or 255.
```

```
  @param skin: The skin image to be cleared in opencv format.
  """
  for x in range(0, 64):
    for y in range(0, 64):
      if skin[y, x][3] >= 200:
        skin[y, x][3] = 255
      else:
        skin[y, x][3] = 0


def postprocess(skin_dir):
  """
  Main function, calls the other functions to postprocess the skins in
the specified folder.
  @param skin_dir: The directory containing the skins to be
postprocessed (they will be saved there).
  """
  print("Postprocessing files in " + skin_dir)
  for file in os.listdir(skin_dir):
    if file.endswith('.png'):
      skin = cv2.imread(skin_dir + '/' + file, cv2.IMREAD_UNCHANGED)
      # Postprocessing operations
      clearWhiteZones(skin)
      removeBodyTransparency(skin)
      setBinaryTransparency(skin)
      # Save the skin
      cv2.imwrite(skin_dir + '/' + file, skin)
```

Bibliografía

[1] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M. & Sutskever, I.. (2021). Zero-Shot Text-to-Image Generation. *Proceedings of the 38th International Conference on Machine Learning*, in *Proceedings of Machine Learning Research* 139:8821-8831.

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.

[3] Rocca, J. (2019). *Understanding Variational Autoencoders (VAEs)*. Towards Data Science. https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

[4] Sancho, F. (2020). *Variational AutoEncoder.* Blog de Fernando Sancho Caparrini. http://www.cs.us.es/~fsancho/?e=232

[5] Kilcher, Y. (2021). *OpenAI DALL·E: Creating Images from Text (Blog Post Explained)* [Video]. Youtube. https://youtu.be/j4xgkjWlfL4

[6] Ramesh, A., Pavlov, M., Goh, G. & Gray, S. (2021). DALL-E: Creating Images from Text. *OpenAI*. https://openai.com/blog/dall-e/

[7] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*.

[8] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., ... & Norouzi, M. (2022). Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding. *arXiv preprint arXiv:2205.11487*.

[9] Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., ... & Norouzi, M. (2022). Imagen: Text-to-Image Diffusion Models. *Google Research*. https://imagen.research.google/

[10] Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). DALL-E 2. *OpenAI.* https://openai.com/dall-e-2/

[11] Wang, P. (2021). *DALL-E in Pytorch*. GitHub. https://github.com/lucidrains/DALLE-pytorch

[12] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32* (pp. 8024–8035).

Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[13] Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

[14] Richardson, L. (2007). *Beautiful soup documentation*. Crummy. https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[15] *skinview3d*. (2017). GitHub. https://github.com/bs-community/skinview3d

[16] *Skin*. (2009). Minecraft Wiki. https://minecraft.fandom.com/wiki/Skin