Capítulo 3

Generación de Imágenes

Resumen: Para evitar tener que hacer o descargar miles de fotos para ser utilizadas para el entrenamiento de la red neuronal se tomó la decisión de tratar de hacer un generador de imágenes a partir de modelos 3D.

3.1. Introducción

Como el objetivo del proyecto es reconocer e identificar distintos deshechos, su material principal y cómo se deben reciclar adecuadamente, es necesario llevar a cabo el entrenamiento de una red neuronal para conseguirlo. Para dicho entrenamiento son necesarias cientos de fotografías para cada material diferente que se quiera introducir.

Conseguir tantas imágenes distintas y claras supone un trabajo costoso y lento; sin tener en cuenta el almacenaje de estas. Contamos con varias opciones para conseguir las imágenes. Una de ellas, y quizá la más obvia, es descargándolas de la red. Esta opción cuenta con un par de problemas a tener en cuenta, el primero es lo tedioso que resulta este procedimiento. Buscar, seleccionar y descargar una a una cientos de imágenes resulta muy lento. El otro problema que tiene esta opción es que además hay que tener en cuenta los derechos de autor y uso de cada una de ellas.

Otra forma es la de llevar a cabo las fotografías personalmente, y en base a esta opción surgió la idea de llevar a cabo un vídeo de objetos de ese material. A partir de este vídeo se planteó que se podría procesar y separar los frames usándolos como imágenes para el entrenamiento. Para esta opción también se encuentran varios problemas. Una manera de hacerlo sería grabar todos los objetos a la vez; esto podría tener como resultado que no hubiera contenido suficiente para el entrenamiento o que los objetos no aparecieran suficientemente claros. Otra manera de llevarlo a cabo es grabando los distintos objetos individualmente, en este caso habría que supervisar el descarte

de los frames intermedios, aquellos que tendrían lugar en la transición entre un objeto y el siguiente.

Para estas opciones comentadas encontramos un problema importante, y es que habría que poseer todo aquello con lo que se quiere entrenar, o en su defecto hacer las capturas en un establecimiento en el que encontrar estos objetos. Pero en este caso hay que tener en cuenta la legislación para llevar esto a cabo. Además, con estas opciones existe el riesgo de perder imágenes y, en el caso de que ocurriera, sería necesario volver a pasar por todo el proceso para volver a tener suficiente material. Para evitar todos estos inconvenientes se plantea la posibilidad de llevar a cabo una aplicación con la que automatizar este proceso.

Actualmente se va viendo una gran mejoría en los gráficos de contenidos digitales audiovisuales, ya sean películas, videojuegos, videoclips, anuncios, etc. Esta mejora llega hasta tal punto que a veces puede resultar difícil distinguir entre realidad y CGI (Computer-generated imagery). Teniendo esto en cuenta y las oportunidades que aporta la automatización no se ha querido desaprovechar la oportunidad de enfocarlo desde un punto más cercano a este. Esta es la opción que se eligió para realizar las imágenes de entrenamiento de la red neuronal. De esta forma el objetivo final de este aparta do consiste en llevar a cabo una aplicación en la que se vayan cargando distintos modelos de objetos y residuos; a cada objeto, tras ser cargado, se le harán numerosas capturas en distintas posiciones y rotaciones que serán posteriormente utilizadas para el entrenamiento. Gracias a esta aplicación se contará con el número de capturas que se quieran de cada objeto y cada material, pudiendo aumentar esta cantidad siempre que se desee o necesite.

3.2. Unity

{**TODO TODO TODO:** Explicar cómo funciona Unity}

Unity[1] ha sido una de las principales tecnologías que se han utilizado para llevar este proyecto a cabo. En concreto para realiza la generación sintética de imágenes. Unity es una herramienta de creación de videojuegos desarrollada por Unity Technologies. Pero esta herramienta no se limita al ámbito de los videojuegos exclusivamente. Su uso está mucho más extendido por la cantidad de funcionalidades que ofrece de manera intuitiva y sencilla. Por esto es utilizado en contenido cinematográfico, transporte y producción e incluso en arquitectura, ingeniería y construcción.

Se tenía muy claro que para llevar esto a cabo se quería utilizar algún programa que ya ofreciera muchas de las funcionalidades necesarias desarrolladas. Con esto se quiso evitar el tener que crear desde cero escenas, una cámara y la iluminación. Puesto que esta parte del proyecto se basa en tra-

[1] `https://unity.com/es`

tar de facilitar y automatizar la obtención y generación de datasets para el entrenamiento de redes neuronales, se decidió seguir ese ideal aprovechando aquello que nos facilita el proceso y que está a nuestro alcance, evitándose así el desarrollo desde cero de estas funcionalidades más complejas.

Existen varias herramientas que se podrían haber utilizado en lugar de Unity para llevar a cabo esta parte del proyecto. Unreal Engine [2] es otro motor de videojuegos cuyo uso también se extiende por numerosas industrias diferentes. Blender [3] es otro programa en el que se podría haber llevado esto a cabo. A pesar de que es una herramienta de creación de gráficos tridimensionales Blender permite añadir funcionalidades, denominados add-ons. Programados en Python podría haber sido otro acercamiento a la generación de imágenes.

Puesto que sobre estos programas no se tiene tanto conocimiento y experiencia previa como con Unity, se decidió que iba a ser más costoso de lo necesario suponiendo que los resultados acabarían siendo similares pero con la posibilidad de que fueran peores al ser algo desconocido.

Para este proyecto, Unity aporta las funcionalidades más necesarias, tales como la cámara, el sistema de iluminación o las escenas. Además, hay que contar con la amplia documentación con la que cuenta y los muchos foros donde los usuarios comparten sus problemas, soluciones, experiencias y descubrimientos sobre esta herramienta.

A pesar de las facilidades que nos brida Unity queda todo el trabajo de la aplicación por hacer. El funcionamiento básico de esta consiste en ir cargando los modelos 3D guardados, uno a uno. En cada frame el objeto que está cargado en la escena cambia su posición y rotación; además, también cambia el fondo.

3.3. Aplicación

Los objetivos de la aplicación de generación de imágenes fueron sufriendo cambios debido a las limitaciones y problemas que se fueron encontrando. Finalmente esta

3.3.1. Idea

Inicialmente se quiso que este apartado del trabajo no solo fuera útil para este proyecto, sino que también pudiera ser distribuido y utilizado por distintos usuarios. Esto se traduce a que se querría llevar a cabo una aplicación para PC con la cual se pudiera generar imágenes a partir de modelos tridimensionales. Estos modelos podrían ser los que la propia aplicación ofreciera, serían aquellos que se han utilizado en este proyecto, o bien podrían ser unos

[2] `https://www.unrealengine.com/en-US/`
[3] `https://www.blender.org/`

modelos 3D porpios que el usuario tuviera y sobre los que quisiera crear un dataset de imágenes. Esto permitiría generar más variedad de imágenes y materiales para entrenar la red neuronal, escalando y ampliando la aplicación de identificación de objetos haciéndola así más completa. Además, así no se limitaba la aplicación, y todo el trabajo que conlleva, a tener una única finalidad y unas pocas interacciones con él; sino que permitiría ser usado para este proyecto y a la vez para muchos otros, facilitando así el proceso de estos.

Desafortunadamente esta idea tuvo que ser descartada debido a la dificultad de importación de modelos. {**TODO TODO TODO:** referenciar subsecciones futuras} Tras invertir tiempo investigando sobre cómo importar modelos desde ejecución, los resultados dejaron mucho que desear. No sólo fue la ausencia de información sobre este tema, sino además la complicación de introducir modelos al propio editor de Unity lo que provocó la decisión de abandonar este objetivo.

La introducción de modelos 3D es algo que no suele utilizarse en videojuegos y Unity no cuenta con una manera fácil de llevarlo a cabo. Además de este problema, otro factor importante para el fracaso de esta idea fueron los numerosos problemas que surgieron al introducir modelos simplemente al editor de Unity. Viendo que esto ya consumía mucho tiempo y esfuerzo se decidió descartar la aplicación para más usuarios en la que poder introducir nuevos modelos.

3.3.2. Planteamiento

Existe un paquete para Unity llamado Perception[4] el cual genera datos etiquetados a partir de modelos 3D. Esto resulta de lo más interesante debido a la similitud que tiene con este apartado del proyecto. Unity Perception proporciona herramientas para generar datasets de gran tamaño utilizables en el entrenamiento y la validación en proyectos de visión computacional.

Para investigar y aprender sobre esta herramienta se ha elegido tomar el proyecto SynthDet[5] como referencia. SynthDet es un proyecto de detección de objetos orientado a los productos más habituales de los supermercados. Para ello, también se basan en entrenar un modelo de detección de objetos a partir de un dataset sintético. La idea, el proceso y las conclusiones están publicadas en el blog oficial de Unity divididas en tres partes [6][7][8].

[4] https://github.com/Unity-Technologies/com.unity.perception

[5] https://github.com/Unity-Technologies/SynthDet

[6] Parte 1: https://blogs.unity3d.com/2020/05/01/synthetic-data-simulating-myriad-possibilities-to-train-robust-machine-learning-models/

[7] Parte 2: https://blogs.unity3d.com/2020/06/10/use-unitys-computer-vision-tools-to-generate-and-analyze-synthetic-data-at-scale-to-train-your-

[8] Parte 3: https://blogs.unity3d.com/2020/09/17/training-a-performant-object-detection-ml-model-on-synthetic-data-using-unity-computer-vision-t

A pesar de todos los factores similares que cuenta este proyecto con SynthDet finalmente se decidió seguir otro camino y no utilizar Perception. Aunque Perception ofrece una forma de generar imágenes ya desarrollada y que para muchos proyectos resulta muy útil, se le han encontrado ciertas pegas por las que se ha decidido finalmente prescindir de dicho paquete y desarrollar la aplicación personalmente.

El primer punto, y el más decisivo, es que Perception nos brinda la oportunidad de generar las imágenes para la aplicación de identificación de objetos; en cambio, para el desarrollo de la aplicación de generación de imágenes no se ha encontrado manera viable para llevarlo a cabo. Las pegas surgen con cómo se podría incorporar este paquete a una aplicación propia sin perder funcionalidades. Debido a lo complejo que es este paquete sería necesario estudiar y comprender toda la lógica que lo constituye hasta el más bajo nivel. Por ello, aunque desarrollar una aplicación propia tendría como resultado algo más sencillo, se decidió que merecía la pena ya que se reduce el riesgo de no poder terminarla, como podría ocurrir de otra forma debido a la complejidad de Perception.

Un último factor que se tuvo en cuenta fue que ya se había decidido y desarrollado parte de la aplicación móvil de identificación de objetos previamente. Como se explicará en el capítulo {**TODO TODO TODO:** capítulo de la identificación}, esta aplicación se ha desarrollado basándose en el ejemplo disponible de TensorFlow Lite sobre clasificación de imágenes; por lo tanto la aplicación se centra e identifica un único objeto, dando así la mayor información sobre este. Perception por el contrario está planteado para generar datasets en los que cada imagen cuenta con varios objetos a identificar en ella.

Con todo lo mencionado anteriormente se tomó la decisión de desarrollar una aplicación sencilla que generara imágenes de un único objeto en cada una, que pudiera ser publicada y que los usuarios pudieran crear sus datasets introduciendo nuevos modelos y materiales, ampliando así los objetivos reconocibles por la aplicación.

3.3.3. Desarrollo

Como se ha comentado en la subsección anterior, finalmente se decidió llevar a cabo la aplicación desde cero.

Para empezar, se desarrolló un Game Manager encargado de la dinámica de la aplicación. Esta es la entidad que maneja la carga de objetos, la separación por materiales, el final de la aplicación y que tiene las rutas a los archivos necesarios.

Al iniciar la aplicación el GM guarda todas las rutas de los recursos y crea la carpeta donde se guardarán las capturas tomadas por la aplicación. Una vez haya hecho esto recorre las carpetas de los distintos tipos de materiales.

Son recorridas en orden alfabético. Cada vez que se de paso a un nuevo material se creará, si no existe, una subcarpeta nueva con el nombre del material en la carpeta destino para las imágenes generadas. Aparte del GM hay otras entidades encargadas del resto de características de la aplicación.

Para utilizar los modelos en la aplicación cada uno de ellos tendrá un "prefab" que será instanciado en la escena. Los "prefabs" son objetos reutilizables, y creados con una serie de características dentro de la vista proyecto, que serán instanciados en el videojuego cada vez que se estime oportuno y tantas veces como sea necesario[9].

La entidad encargada de los modelos es la que carga e instancia los prefabs de estos. Cuando se llama al método de cargar un nuevo material desde el GM esta entidad accede a la subcarpeta dentro de "Prefabs" del material correspondiente, este se indica como un parámetro del método. Si la carpeta de ese material existe, se cargan en un array todos los recursos en ella casteándolos a GameObjects. Además, lleva la cuenta de los objetos instanciados y cuántos quedan de ese tipo y si alguno de los objetos no se ha cargado correctamente pasa al siguiente. Cada vez que sea necesario un nuevo objeto esta entidad lo instancia y le añade el componente para la gestión de su posición.

El objeto instanciado cambia de posición y rotación a unas aleatorias en cada frame para que sean distintas en cada captura. Para la posición se establecen unos límites entre los que se genera un valor aleatorio para cada eje. Para el eje vertical, el eje Y, se establece un valor propio en negativo para el mínimo y positivo para el máximo. Para el eje de la Z, la profundidad, se establece de manera similar, en este caso no se utiliza el mismo valor en simetría respecto al eje cero, sino que se tiene un valor máximo establecido por parámetro y el mínimo estará posicionado a dos unidades respecto a la cámara. Por último, la posición en el eje X, en horizontal, se genera también de manera aleatoria pero el rango en vez de ser a partir de un valor pasado por parámetro puesto en positivo y negativo, en este caso en cambio para ese valor se utilizará el generado para la posición en el eje Z.

Se decidió generar la posición en el eje X de esta manera puesto que según cuál sea la distancia del modelo respecto a la cámara el rango en el que el objeto será visible en el eje X es directamente proporcional. De esta manera no se limita la posición a unos valores cerrados, sino que permite la posibilidad de generar muchas más posiciones viables. En el eje Y no se decidió realizar de esta misma manera ya que se observó tras hacer numerosas pruebas que el que este rango estuviera relacionado con la posición Z traía más problemas que ventajas, en la mayoría de los casos el rango acababa siendo demasiado amplio y se decidió que no merecía la pena ralentizar la aplicación con una operación cuando al final no generaba buenos resultados.

[9] Esto es una cita pero no las entiendo muy bien así que por ahora lo pongo así: `https://academiaandroid.com/que-son-los-prefab-en-unity-3d/`

Una vez se ha generado la nueva posición aleatoria se comprueba que está dentro del campo de visualización de la cámara, en el caso de que no sea así se vuelve a generar una nueva posición aleatoria. Cuando la posición sea adecuada entonces se establece la rotación aleatoria.

A pesar de que toda esta parte que ha sido explicada no supuso mucho problema de desarrollar sí hubo que revisarla poco después. Las pruebas que se iban realizando según se avanzaba, añadía y mejoraban las distintas funcionalidades indicaban que todo funcionaba correctamente y no parecía haber ningún problema; pero esto no era del todo correcto. Estas pruebas iniciales solamente se estaban haciendo desde el editor y, un poco más adelante, al generar la primera build de la aplicación, se descubrió que nada de lo desarrollado funcionaba. A pesar de lo desesperanzador que resulta esto, afortunadamente fue en un momento muy temprano del desarrollo y se pudo solucionar sin provocar grandes retrasos.

Para identificar dónde estaba el error se probaron las distintas partes ya desarrolladas de la aplicación de manera independiente, con este proceso se encontró que era la carga de objetos lo que traía problemas y no actuaba como se esperaba. Tras investigar sobre el asunto se descubrió que en la build sólo se cargan los recursos que se encuentran en la carpeta Resources"[10]. Una vez identificado el origen del problema este se solucionó sin grandes esfuerzos, simplemente cambiando de lugar la carpeta contenedora de los prefabs queda solventado.

Llegado este punto ya solo quedaba generar las imágenes y Unity cuenta con una clase enfocada a esto, la clase "ScreenCapture"[11]. A pesar de lo esperanzador y sencillo que se presentaba esta parte teniendo que añadir una mera línea de código, aprendiendo de los errores pasados y tratando de reducir los futuros este cambio se probó inmediatamente, descubriendo que desde el editor no había ningún problema pero al generar el ejecutable no se obtenía el resultado esperado.

Investigando al respecto se descubrió que esta clase suele generar problemas habitualmente, aunque no hay nada respaldado por Unity. En vista que lo que proporcionaba Unity no iba a ser útil se buscaron otras optativas decidiendo, finalmente, crear una clase propia para la captura de imágenes siguiendo un tutorial bien explicado[12]. Las capturas se irán guardando en la carpeta destino dentro de una subcarpeta nombrada como el material correspondiente.

De cada modelo se generan tantas imágenes como se le haya indicado al GameManager, y se genera una cada frame. Una vez se hayan capturado todos los modelos de todos los materiales tantas veces como se haya establecido la aplicación termina y se cierra, tanto desde el editor como el ejecutable.

[10] `https://docs.unity3d.com/Manual/LoadingResourcesatRuntime.html`
[11] `https://docs.unity3d.com/ScriptReference/ScreenCapture.html`
[12] `https://www.youtube.com/watch?v=lT-SRLKUe5k`

{**TODO TODO TODO:** Modelos modelos en resources colisión FOV importación problemas carga destrucción organización posición y rotación Fondos rotación aleatoria material aleatorio composición aleatoria Capturas cantidad formato tamaño script - por qué uno propio? guardado distinción}

3.3.4. Modelos 3D

Esta sección está orientada a explicar el proceso de conseguimiento y uso de los modelos tridimensionales

Para llevar a acabo esta aplicación de generación de imágenes son necesarios modelos 3D de los que partir para la generación de estas capturas. Para este proyecto todos los modelos seleccionados y utilizados son de libre uso conseguidos desde varias páginas dedicadas a esto.

Los primeros acercamientos se realizaron con modelos importados desde la Unity Asset Store[13]. De inicio no hubo mucho éxito encontrado modelos realistas de uso libre en esta plataforma para la aplicación, así que para ampliar la cantidad de modelos y materiales posteriormente se tuvo que explorar en otras muchas páginas. Para los primeros acercamientos con el pack de latas[14] encontrado en la Asset Store de Unity fue suficiente para inciar el desarrollo y las pruebas de la aplicación. Este pack contiene los archivos FBX, texturas y materiales de 16 latas diferentes, conjunto a sus respectivos prefabs; con estos recursos se pudo comenzar el desarrollo.

La nueva posición y rotación serán aleatorias, controlando siempre que no queden fuera de los límites de la cámara. Una vez situado el objeto y generado el fondo se realiza una captura de la pantalla. Cuando se hayan recorrido todos los modelos de todas las carpetas de materiales la aplicación terminará su ejecución.

El formato elegido para los modelos es FBX. FBX es un tipo de formato de archivo, propiedad de Autodesk. Se decidió utilizar este formato ya que es uno de los más extendidos, se encuentran bastantes recursos sin problema y su carga en Unity suele ser sencilla. Además, ha sido un formato utilizado para modelos de proyectos anteriores por lo que trabajar con ello generaba más tranquilidad que con algo desconocido.

Con lo comentado previamente parecía que la carga de los modelos en Unity sería un mero trámite sin mucha complicación. Desafortunadamente, esto no es así. Puesto que Unity continúa actualizándose y mejorando funcionalidades, esta es una de las que ha sido editada. La carga de los modelos se hacía correctamente. El problema que surgió es que ni las texturas ni los materiales se cargan correctamente con el modelo.

Para llevar esto a cabo los modelos exportados en FBX tienen que serlo de una manera especial.

[13] `https://assetstore.unity.com/`

[14] `https://assetstore.unity.com/packages/3d/props/free-cans-opened-pack-145723`

Esto provocó que muchos de los modelos que se tienen han tenido que ser importados primero a blender

Se hicieron muchas pruebas tratando de abrirlos desde distintos editores para tratar de comprender qué ocurría con las texturas. Algunos de estos modelos incluso presentaban problemas mayores, por ejemplo que las caras se renderizaran al revés. En este caso concreto se trataba de una botella de la cual se podían ver las caras internas en vez de las externas {**TODO TODO TODO:** mejorarlo y poner imágenes}

La aplicación irá recorriendo todas las carpetas de materiales La generación de imágenes se lleva a cabo a partir de modelos 3D. Los modelos están agrupados en carpetas según su material. La aplicación recorre todas las carpetas de materiales disponibles Todos los modelos se cargan al iniciar la aplicación, se guarda cuántos materiales dis Estos modelos son de distintos objetos y materiales y están ordenados y separados por esto último. De cada uno de los objetos se toman por defecto diez capturas y en cada una tendrán una posición y rotación diferente, siendo esta y aleatoria.

3.3.5. Capturas

{**TODO TODO TODO:** Las capturas se guardan como .jpg, formato elegido porque era el más **sencillo** de utilizar y ningún otro formato aportaba nada que lo hiciera mejor para la generación de imágenes. Las imágenes se exportan con un tamaño de **XX** x **YY**.

Se les pone como nombre el del modelo capturado, añadiendo la fecha y hora en la que se realiza la captura.

La captura se toma una vez el frame ha sido renderizado tutorial: `https://www.youtube.com/watch?v=lT-SRLKUe5k` }