
Aplicación móvil de identificación de objetos para reciclaje



TRABAJO DE FIN DE GRADO

Celia Castaños Bornaechea

Grado en Desarrollo de Videojuegos

Facultad de Informática

Universidad Complutense de Madrid

Junio 2021

Documento maquetado con TEXIS v.1.0.

Este documento está preparado para ser imprimido a doble cara.

Aplicación móvil de identificación de objetos para reciclaje

Proyecto de fin de grado

Dirigida por: Pedro Pablo Gómez Martín

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

Junio 2021

Copyright © Celia Castaños Bornaechea

Índice

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Herramientas utilizadas	2
1.4. Plan de trabajo	3
2. Estado del arte	5
2.1. Aplicaciones de reciclaje	5
2.2. Generación sintética de imágenes	7
2.3. Redes neuronales	8
2.4. Identificación de objetos e imágenes	11
3. Generación de imágenes	15
3.1. Introducción	15
3.2. Unity	16
3.3. Aplicación	18
3.3.1. Idea	18
3.3.2. Planteamiento	19
3.3.3. Desarrollo	20
3.3.4. Modelos 3D	24
3.3.5. Capturas	25
4. Red Neuronal	27
4.1. Introducción	27
4.2. TensorFlow y TensorFlow Lite	28
4.3. Entrenamiento	29
4.4. Resultados	31
5. Aplicación de identificación de objetos	35
5.1. Introducción	35
5.2. Tensorflow Lite	36
5.3. Identificación de objetos	36

5.4. Pruebas	38
6. Conclusiones y trabajo futuro	41
6.1. Conclusiones	41
6.2. Trabajo futuro	41
Bibliografía	43
Índice alfabético	48

Índice de figuras

2.1. Estructura de una red neuronal artificial.	9
4.1. Organización de las carpetas con las imágenes separadas.	31
4.2. Variación de la precisión de la red neuronal a lo largo del entrenamiento de esta.	32
4.3. Variación de la precisión de la red neuronal durante las pruebas de esta.	33
4.4. Variación de la precisión de la red neuronal según la proporción de imágenes reales y sintéticas.	34
5.1. Comparación con una lata de atún	39
5.2. Comparación con una botella de agua.	40
5.3. Resultados del modelo 70-30 con una lata de bálsamo labial. .	40

Índice de Tablas

Capítulo 1

Introducción

1.1. Motivación

La generación de residuos está ligada al modelo de desarrollo actual de la sociedad y constituye uno de los principales problemas ambientales a los que se enfrenta el mundo[33]. Los residuos se pueden clasificar en dos tipos: los producidos por la actividad industrial, llamados residuos industriales, y los generados por la propia actividad humana, denominados residuos urbanos.

Bien es cierto que gran parte de la contaminación y emisiones de CO₂ provienen de grandes empresas, por ejemplo en 2018 el 25 % de las emisiones en España fueron generadas por solamente diez compañías; pero este TFG se centrará en los residuos urbanos, que es en aquello sobre lo que la población puede tomar responsabilidad y poner de su parte. El problema de los residuos sólidos urbanos viene del incremento de utilización de envases sin retorno en los últimos años. Estos embalajes pueden ser de diferentes materiales como celulosa, vidrio, plástico o mixtos (papel plastificado, telas plastificadas, etc.) lo que complica su tratamiento, puesto que se debe llevar a cabo una selección y separación previa [11].

Una mala gestión de los residuos puede provocar impactos medioambientales irreversibles. A día de hoy ya se pueden observar muchos de estos efectos que parecía que vendrían en el futuro. Podemos destacar entre ellos el incremento de las temperaturas en todo el país e incluso la del Mediterráneo, además del incremento del nivel de este mismo. También la dilatación del verano unos 9 días por década, que da lugar a que actualmente contemos con 5 semanas más que a comienzos de los años ochenta. Otros efectos han sido la desaparición de más de la mitad de los glaciares en España y los cambios en la distribución, comportamientos y alimentación de la biodiversidad, entre otros factores [2].

Pero la gestión adecuada de residuos es algo al alcance de la mano de cualquier ciudadano o ciudadana de a pie.

Mezclar materiales no sólo es contaminante porque dificulta la recuperación

ción y reciclaje de estos; sino que además el proceso de separación de residuos también es costoso y contaminante, y no siempre tiene resultados demasiado buenos. Muchas veces los materiales recuperados, al haberse juntado con otros, son de baja calidad y con un alto nivel de partículas no reciclables. Por eso, para facilitar este proceso encontramos contenedores especiales para los distintos residuos.

El separar los desechos de manera adecuada es una gran aportación al cuidado del medioambiente, pero como se ha comentado anteriormente, hay una gran cantidad de materiales y residuos diferentes y en ocasiones puede resultar difícil y confuso cómo deben separarse. Debido a esta dificultad surge la motivación de realizar este Trabajo de Fin de Grado. Para esto se ha querido plantear y desarrollar una aplicación de identificación de objetos que utilizando la cámara de un teléfono móvil indique la manera adecuada de desechar el residuo identificado. De esta forma, la información se encontraría de manera cómodamente accesible para una gran parte de la población, fomentando así el reciclaje.

1.2. Objetivos

Este proyecto tiene como objetivo desarrollar dos aplicaciones distintas:

La primera es una aplicación de identificación de objetos orientada al reciclaje para dispositivos móviles. A través de ella los usuarios pueden identificar objetos con el fin de solventar de manera fácil y rápida las dudas sobre cómo desechar correctamente los diferentes residuos. La aplicación dará información sobre el material y cuál es la manera adecuada de reciclarlo.

Para llevar a cabo esto es necesario entrenar una red neuronal, lo cual requiere cientos de imágenes. Con el fin de facilitar el proceso de obtención de estas, surge la segunda aplicación a desarrollar. El objetivo de esta es tener una alternativa que evite el proceso tedioso y lento que puede resultar la obtención de las imágenes. Se trata, por lo tanto, de una aplicación para ordenador de generación de imágenes sintéticas a partir de modelos tridimensionales. Dichos modelos podrán ser propios del usuario o bien se podrán aprovechar los que vengan por defecto.

1.3. Herramientas utilizadas

Para comenzar, se ha utilizado Android Studio para la creación de la aplicación de identificación de objetos. Para entrenar y obtener el modelo necesario para su correcto funcionamiento se ha creado un script en Python utilizando las librerías de Numpy y Tensorflow; como editor se ha utilizado PyScripter. Toda esta primera parte se ha basado en los ejemplos disponibles sobre Tensorflow Lite que se pueden encontrar en el blog de Tensorflow.

Para desarrollar la aplicación de generación de imágenes se ha utilizado el motor de videojuegos Unity. Los modelos que se utilizan en esta aplicación han sido conseguidos desde varios orígenes tales como Unity Asset Store, Free3D, CGTrader y 3DModelHaven.

Por último, se ha usado GitHub como plataforma para el control de versiones y TexMaker para el desarrollo de la memoria a partir de la plantilla Tesis.

Todo el proceso se ha realizado desde un dispositivo Windows y para las pruebas en teléfono móvil se ha usado uno con sistema operativo Android.

1.4. Plan de trabajo

El trabajo se divide en tres partes: la generación de imágenes, el entrenamiento de la red neuronal y el desarrollo de la aplicación de identificación de objetos.

La primera parte se trata de una aplicación que cargue modelos 3D, separados por material, y realice numerosas imágenes a cada uno cambiándoles la posición, la rotación y el fondo para obtener diversidad en las imágenes. Estas imágenes deberán ser guardadas separadas por el material al que corresponden, igual que lo estaban los modelos al cargarlos.

Con las imágenes generadas del paso anterior tiene lugar el entrenamiento de la red neuronal, la segunda parte del proyecto. Para llevar a cabo esto se deberá investigar sobre los distintos tipos de redes neuronales y elegir la opción más adecuada para que el resultado, el modelo entrenado, sea utilizado desde una aplicación móvil.

Por último, como se ha mencionado antes, queda el desarrollo de la aplicación para dispositivos móviles que haciendo uso de la cámara y el modelo entrenado, identifique el material del objeto al que se está enfocando y después indique al usuario cómo se debe reciclar dicho material.

Se considera que la parte principal del proyecto es el conseguir trasladar los resultados de la red neuronal a una aplicación móvil, puesto que es algo sobre lo que no se tiene experiencia previa este es el punto por el que se va a comenzar a investigar y a desarrollar el proyecto. Una vez se haya establecido la red neuronal y la exportación del modelo entrenado, se procederá a desarrollar la aplicación móvil que utilice el modelo generado.

Por último, una vez terminado lo que se considera la parte principal del proyecto, se pasará a desarrollar la aplicación de generación de imágenes. Para ello se investigarán las opciones que ya puedan existir o que puedan facilitar el trabajo, así como la forma de introducir modelos al ejecutar la aplicación.

Capítulo 2

Estado del arte

2.1. Aplicaciones de reciclaje

Debido a los distintos materiales que se encuentran en los residuos del día a día de la mayoría de las personas y la cantidad de estos que se generan, ha surgido una necesidad de ayuda para reciclar correctamente todos estos residuos.

Esta ayuda ha llegado, sobre todo, en forma de aplicación móvil. La generalización del uso de los dispositivos móviles en la población favorece a que esta sea la mejor forma de encontrar esta ayuda.

Cada nación cuenta con un sistema diferente de gestión de residuos y reciclaje, debido a la cantidad de opciones que hay en este trabajo se enfocará al territorio nacional. En España, en aplicaciones de reciclaje que identifiquen objetos contamos con dos. La primera es “AIRE Asistente Inteligente de Reciclaje”¹, disponible tanto para iOS, Android y en el navegador, es una aplicación publicada por “ecoembes”, empresa encargada del reciclaje de los residuos de los contenedores amarillo y azul en España. Se trata de un asistente virtual con el que se tienen una conversación de chat, el usuario escribe el material o el objeto que desea reciclar y el *bot* le responde explicando la forma idónea de desecharlo. También se le pueden enviar imágenes y audios para que identifique el objeto. Otra es “ReciclaYa”², aplicación desarrollada por Carrefour. En este caso la identificación se lleva a cabo mediante el código de barras del ticket de la compra que se haya hecho, según qué productos se hayan adquirido indica cómo reciclar cada uno sus envases. Esta funcionalidad está sólo disponible para algunas de las marcas principales, ya que son las propias empresas las que brindan esa información. Algunas de estas empresas sobre las que se disponen los datos son Carrefour, Coca Cola, Nestlé, Central Lechera Asturiana, Mahou San Miguel, PepsiCo, El Pozo o Pescanova, entre muchas otras.

¹<https://www.ecoembes.com/proyectos-destacados/chatbot-aire/>

²<https://www.reciclaya.app/es/como-funciona>

Además de aplicaciones de identificación de residuos, también se encuentran numerosas aplicaciones enfocadas más a la ayuda y el aprendizaje sobre reciclaje, algunos ejemplos son “Residus” de la Generalitat de Catalunya³ en colaboración con “ecomenbes” y “ecovidirio”. Esta aplicación cuenta con una base de datos de residuos con su respectivo contenedor adecuado y el proceso que sigue una vez es desecharlo en este. Cuenta con dos maneras de informarme sobre cada desecho. La primera es por contenedores, se selecciona el contenedor sobre el que se quiere tener más información, sobre este sale un listado de todos los residuos disponibles en la base de datos reciclables en él, y finalmente se selecciona uno de ellos. La segunda manera es introducir en el buscador el residuo. Cuando se tiene uno seleccionado, en ambas opciones, se explica el proceso que sufre este desecho desde el contenedor, pasando por la plantas de selección y reciclaje, y en qué se transforma finalmente. Además de informar sobre la forma adecuada de deshacerse de cada residuo, también cuenta con un buscador de puntos limpios a partir de la localización del dispositivo o mediante un buscador.

Otro ejemplo es “Recicla y suma” de Pensumo⁴. Es una herramienta pensada para incentivar el reciclaje animando al ciudadano a que se acostumbre a separar en casa los residuos que genera para luego acudir a los contenedores y reciclarlos. La función principal es la de generar el hábito en el ciudadano, para que, animado por el incentivo económico, acabe interiorizando la necesidad de reciclar y aprenda a depositar correctamente los residuos. Los usuarios sacan fotos en el momento que depositan los residuos en el contenedor correcto y reciben dinero por la acción.

Alejándose un poco del reciclaje como tal encontramos “The Planet App”⁵ por Clean Planet Ventures, SL. Es una aplicación de concienciación, aprendizaje y ayuda sobre la huella de carbono que genera cada uno. Permite al usuario calcular su huella de carbono y conocer las emisiones que genera separadas por categorías, además de comparar sus emisiones con las de distintos grupos poblacionales y otros usuarios y configurar un plan de sostenibilidad personalizado adquiriendo así hábitos y realizando acciones que reduzcan las emisiones que genera.

En el lado internacional existen muchas más aplicaciones, como “My Little Plastic Footprint”⁶, de Plastic Soup Foundation. Esta es muy similar a la aplicación anterior, pero en vez de ser respecto a la huella de carbono es sobre el plástico que genera el usuario. En este caso, la aplicación te indica tu PMI (índice de masa plástica), cuando se tiene esto se comienza la denominada “dieta de plástico” que consiste en ir adquiriendo como hábito alternativas al plástico que se utiliza normalmente. A medida que se van incorporan-

³<https://play.google.com/store/apps/details?id=cat.gencat.mobi.residus>

⁴<https://reciclaysuma.com/>

⁵<https://theplanetapp.com/>

⁶<https://mylittleplasticfootprint.org/>

do los consejos de la aplicación al perfil del usuario, el PMI de este se irá reduciendo.

“Grow Recycling”⁷, de Gro Play Digital, es un juego educativo para niños utilizado en colegios y guarderías en Estados Unidos y Europa para impartir educación sobre la sostenibilidad. El juego consiste en alimentar a distintos cubos de basura con los residuos correspondientes para que niños y niñas puedan aprender a cuidar del planeta. Otra aplicación educativa de reciclaje es “Recycle Coach”⁸, de Municipal Media Inc., con ella se puede aprender sobre el proceso de reciclaje, los diferentes materiales y el proceso de depósito y recogida de residuos en tu zona. Además, cuenta con *tests*, artículos y actividades para aprender apropiadamente sobre la disposición de residuos.

2.2. Generación sintética de imágenes

Respecto a generación de imágenes sintéticas se encuentran distintos acercamientos. Por un lado, están las redes generativas adversativas, relacionadas con el tratamiento de imágenes. Este modelo consta de dos redes neuronales denominadas generador y discriminador, respectivamente. El objetivo es generar datos similares a los que se han usado para el entrenamiento. La red generador, como su nombre indica, es la encargada de generar datos del tipo de los del entrenamiento. Por otro lado, la red discriminadora distingue entre los datos reales que se le proporcionan y los generados por la red anterior [14].

Acercamientos más similares al que se propone en este trabajo, son sobre imágenes generadas a partir de entornos virtuales. Por ejemplo, SYNTHIA [31] es un *dataset* de imágenes fotorrealistas generadas a partir de los frames obtenidos de la redenización de un mundo virtual formado a partir de modelos tridimensionales, además de las imágenes cuenta con las anotaciones semánticas a nivel de *pixel* que etiquetan los distintos objetos presentes en la escena, distinguiendo hasta 13 clases distintas. Este *dataset* está orientado al entrenamiento de coches autónomos, sistemas de conducción asistida y proyectos relacionados, como complemento para diferentes *datasets* reales (Camvid, KITTI, U-LabelMe, CBCL).

Otro ejemplo similar, es el planteamiento de generar imágenes efectivas para el entrenamiento de detección de objetos partiendo de un conjunto pequeño de imágenes reales y el modelo tridimensional del objeto a identificar [?]. Los parámetros pueden reutilizarse para generar un número ilimitado de imágenes de entrenamiento del objeto de interés en poses del modelo tridimensional arbitrarias. Por cada imagen real se computan cinco parámetros de la pose, tres de orientación y dos de traslación que permite colocar el modelo 3D en la posición deseada. Una vez se obtiene la imagen, para hacerla lo

⁷<https://www.groplay.com/apps/grow-recycling/>

⁸<https://recyclecoach.com/>

más similar a la imagen real, pasa por un postprocesamiento que involucra el añadir desenfoque, para simular la lejanía y el movimiento del objeto; ruido, que imita el ruido que añade la cámara al tomar la fotografía; y propiedades del material del objeto. Esto permite que las imágenes generadas sintéticamente sean más similares a las reales, obteniendo un *dataset* sintético muy similar a uno real.

Un proyecto basado en el anterior, es la detección de objetos en contextos industriales [7]. En este proyecto se parte del modelo renderizado de una pieza utilizada en el entorno industrial con el que se genera un *dataset*. Como en el caso anterior, para añadir más diversidad e identificar las piezas en un mayor número de situaciones, generan las imágenes con diferentes fondos creados a partir de una selección de imágenes de localizaciones reales; alterando la iluminación y añadiendo sobras de distintas formas e intensidades; y creando imágenes desenfocadas o en movimiento.

Perception [37] es un paquete de Unity en desarrollo que proporciona herramientas para generar *datasets* sintéticos a gran escala. Está planteado para tareas de aprendizaje automático, tales como detección de objetos, segmentación semántica y más. Este *dataset* está en formato por frames, estos son capturados utilizando sensores simulados y es lo que se utiliza para el entrenamiento y la validación del modelo. Perception, además de permitir al usuario que genere sus propios datos para la tarea que requiera, ofrece una cantidad de etiquetas comunes ya generadas para facilitar la generación de datos sintéticos. Además de la generación de *datasets* sintéticos, incluye herramientas para generar *keypoints*, poses y animaciones aleatorias. Este paquete puede utilizarse para diversidad de proyectos, desde simulación de coches autónomos o *smart cities*⁹, pasando por demostraciones de recoger y colocar objetos mediante un brazo robótico¹⁰, hasta detención de objetos utilizando únicamente imágenes sintéticas para el entrenamiento.

Este último es el proyecto SynthDet [19], utiliza Perception para crear el *dataset* utilizado en el entrenamiento de la red neuronal. Generan imágenes de productos habituales de supermercados con el fin de estudiar la viabilidad de los supermercados sin cajas registradoras. Este objetivo se basa en el supermercado Amazon Go [29], creado por Amazon en 2017. Se entrará más en profundidad sobre este proyecto y sus distintas características en la sección 2.4

2.3. Redes neuronales

Las redes neuronales artificiales surgen inspiradas por la funcionalidad sofisticada del cerebro humano, donde miles de millones de neuronas se encuentran interconectadas y procesan información de manera paralela. Tratan

⁹<https://github.com/Unity-Technologies/Unity-Simulation-Smart-Camera-Outdoor>

¹⁰<https://github.com/Unity-Technologies/Robotics-Object-Pose-Estimation>

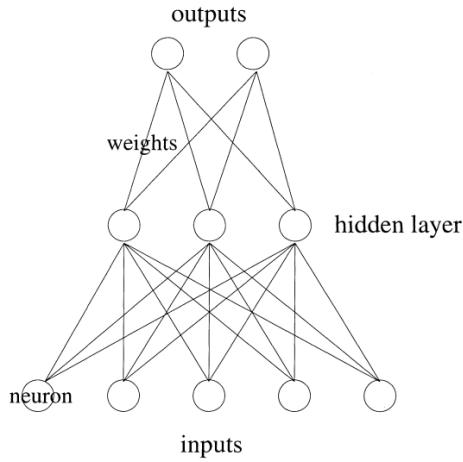


Figura 2.1: Estructura de una red neuronal artificial.

de emular el comportamiento del cerebro humano caracterizado por el aprendizaje a través de la experiencia y la extracción de conocimiento a partir de un conjunto de datos [10], además de simular las características básicas de las redes neuronales biológicas; el procesamiento paralelo, la memoria distribuida y adaptabilidad. Esto permite a las redes artificiales aprender y generalizar a partir de un conjunto de datos de relación matemática desconocida [16]. De esta forma, adquieren, almacenan y utilizan conocimientos basados en la experiencia en lugar de ser programados de manera explícita [41].

Una red neuronal artificial consta de una capa de entrada de neuronas, una o varias capas ocultas y una capa final de salida. En la figura 2.1 se muestra la estructura habitual de una red neuronal artificial.

Las neuronas están conectadas entre ellas mediante líneas, cada una de estas conexiones se asocia con un valor numérico llamado peso[38]. Durante el aprendizaje de la red se modifica el valor de los pesos asociados a las neuronas con el fin de que generar una salida a partir de unos datos de entrada. Se considera que el aprendizaje ha terminado cuando los valores de los pesos permanecen estables.

Aunque existen muchas definiciones para las redes neuronales, una de las más extendidas establece que una red neuronal artificial se define como un grafo dirigido, donde los nodos representan las neuronas y las aristas la sinapsis. Aquellos nodos que no cuentan con conexiones entrantes son consideradas las neuronas de entrada, y los que no tiene conexiones salientes las de salida. Por lo tanto, los nodos con conexiones entrantes y salientes corresponden a las neuronas ocultas [10, 13, 16, 26]

Se pueden seleccionar varios puntos de inicio de las redes neuronales

artificiales. Algunos comienzan hablando de cómo Ramón y Cajal en 1888 demuestra que el sistema nervioso está compuesto por una red de células individuales, las neuronas, ampliamente interconectadas entre sí y su funcionamiento [22]. Otros nombran a Alan Turing como pionero al estudiar el cerebro como una forma de ver el mundo de la computación. Independientemente, en todos los casos se toma a Warren McCulloch (neurobiólogo) y Walter Pitts (estadístico) como los primeros teóricos que concibieron fundamentos de la computación neuronal en el artículo “*A logical calculus of Ideas Imminent in Nervous Activity*” en 1943 [25]. Posteriormente, en 1956, tuvo lugar el congreso de Dartmouth, comúnmente considerado el nacimiento de la inteligencia artificial. Un año después Frank Rosenblatt comenzó el desarrollo de “Perceptron”.

Perceptron es considerada la red neuronal más antigua, siendo un sistema clasificador de patrones. Este modelo era capaz de reconocer patrones similares a los del entrenamiento pero que no había visto antes. Sin embargo, contaba con ciertas limitaciones, la más importante fue que no era capaz de resolver la función lógica OR exclusivo¹¹ ni clasificar clases no separables linealmente. Esto fue demostrado matemáticamente en los años 60 por Minsky y Papert, lo que desencadenó un fuerte desinterés en la computación neuronal.

No fue hasta principios de los años 80 cuando se consiguió devolver el interés en este campo con la publicación del artículo “*Hopfield Model o Cross-bar Associative Network*” de John Hopfield y el redescubrimiento de David Rumelhart y G. Hinton del algoritmo de aprendizaje de propagación hacia atrás (*backpropagation*) planteado por Paul Werbos en 1974 [6, 22, 24].

Existen dos tipos de redes neuronales principales, las redes convolucionales y las recurrentes. Las primeras se desarrollaron para el reconocimiento de imágenes y son útiles para distinguir elementos esenciales en una entrada de la red compleja [28]. Una particularidad de estas redes es la operación de convolución¹² que se realiza en las primeras capas utilizando como filtro, esto permite que tenga muchas aplicaciones en el tratamiento de imágenes [23]. Por otro lado, las redes neuronales recurrentes son aquellas en las que la salida de una neurona vuelve en forma de entrada, por lo que puede haber ciclos en las conexiones entre las neuronas de cada capa. Cuentan con la peculiaridad de que no sólo aprenden de los datos sino también de secuencias de estos. Todo esto las convierte en buenas candidatas para trabajar con datos con dependencias estructurales en una dimensión, como el texto, o el audio [28, 15].

También se lleva a cabo una distinción en las redes neuronales según el

¹¹XOR u OR exclusiva es una puerta lógica cuya salida es verdadera si una, y sólo una, de las entradas es verdadera. Si ambas son falsas o ambas son verdaderas la salida es falsa.

¹²Una convolución es un operador matemático que transforma dos funciones f y g en una tercera que representa la magnitud en la que se superponen f y una versión trasladada e invertida de g .

paradigma de aprendizaje que siguen, existen dos tipos principales, las redes neuronales supervisadas y las no supervisadas. Las redes neuronales supervisadas se caracterizan por la presencia de un agente externo que controla el entrenamiento, este determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada se modifican los pesos de las conexiones con el fin de conseguir una salida aproximada a la esperada. En cambio, las redes con aprendizaje no supervisado no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta [24].

Algunas de las Redes Neuronales más expandidas actualmente son AlexNet, VGGNet, MobileNet e InceptionNet.

AlexNet [21] es una red ampliamente utilizada que se dió a conocer en la competición de ImageNet, donde consiguió el primer puesto con una tasa de error del 15.3 % (2012). Su estructura se divide en dos GPUs y fue la primera red neuronal en utilizar ReLu¹³ como función de activación¹⁴ en vez de la función sigmoide¹⁵. Este cambio mejoró notablemente la velocidad de entrenamiento de las redes de aprendizaje profundo [8, 18, 40].

La segunda red destacada es VGGNet, esta también saltó a la fama gracias a ImageNet en 2014. La diferencia más destacable con AlexNet es la gran cantidad de capas convolucionales más que tiene. Además de una arquitectura más profunda presenta también otros nuevos conceptos [35, 40].

InceptionNet, desarrollada por Google, fue de las primeras redes que no se basó en añadir más capas convolucionales para mejorar la red, sino que cambió la estructura de estas usando filtros de varios tamaños. El objetivo era tener filtros de diferentes tamaños para un mismo objeto. Esto provocó que la red fuera más “ancha” en vez de más profunda [36].

La última red es MobileNet, esta también fue desarrollada por Google. Surgió con la intención de adaptar InceptionNet a dispositivos móviles. Su objetivo fue reducir el número de computaciones y parámetros pero manteniendo unos resultados similares [34, 17].

2.4. Identificación de objetos e imágenes

La detección e identificación de objetos es un campo de la visión artificial y el procesamiento de imágenes. El objetivo de la visión computacional es desarrollar algoritmos que reciban una imagen de entrada y produzcan una

¹³ReLu transforma los valores de entrada anulando los negativos y manteniendo los positivos.

¹⁴la función de activación en redes neuronales devuelve una salida que es generada por una o varias entradas. Cada capa de la red cuenta con una función de activación.

¹⁵La función sigmoide transforma los valores de entrada a una escala entre 0 y 1.

interpretación describiendo los objetos presentes, en qué posición y la relación espacial tridimensional de los objetos presentes. A pesar de que actualmente ya es posible detectar y reconocer objetos en conjuntos de miles de imágenes complejas, todavía no se cuenta con un paradigma aceptado y dominante con el que la mayoría de investigadores trabajen [4].

La detección, seguimiento y reconocimiento de objetos en imágenes es uno de los problemas claves en visión computacional [9]. Normalmente, en cada imagen sólo aparece una pequeña cantidad de los objetos posibles, pero estos pueden encontrarse en una gran cantidad de posiciones y escalas que se deben tener en cuenta [5].

Esto es algo utilizado en diversos ámbitos, desde objetos, personas y caras, hasta campos mucho más específicos. Un ejemplo es VOCUS (Visual Object detection with a CompUtational attention System) [12], un sistema capaz de seleccionar automáticamente secciones de interés en imágenes y detectar objetos específicos. Cuenta con dos modos de trabajo, un primer modo de exploración en el que no se especifica un objetivo y en el que se buscan zonas de interés. Son consideradas zonas de interés aquellas en las que haya grandes contrastes o características únicas (por ejemplo una oveja negra en un rebaño de ovejas blancas). El segundo modo es de búsqueda con un objetivo definido, en este modo se utiliza información previamente aprendida sobre el objetivo para seleccionar las computaciones más destacadas respecto a ello.

Otro ejemplo es el reconocimiento de matrículas de los coches presentes en una imagen [27]. En este caso se presenta una doble identificación, ya que primero debe localizarse la matrícula en la imagen y posteriormente sobre esta se tiene que llevar a cabo la segmentación e identificación de los caracteres que la componen.

En el apartado 2.2 se mencionaron varios proyectos de generación de *datasets* sintéticos. Todos ellos, además, tienen también como objetivo la identificación de objetos en imágenes.

SynthDet [19] trata de identificar los distintos productos contenidos en carritos de la compra con el objetivo de facturar automáticamente las adquisiciones de los clientes en supermercados. Está basado en Amazon Go[29], supermercado de la empresa Amazon, ubicado en Seattle (Estados Unidos). El modelo de Amazon Go se basa en un sistema de cámaras y sensores distribuidos por el establecimiento que analizan los movimientos de los clientes. Estos, además de obtener la información necesaria para realizar los cobros, también extraen información sobre las pautas de comportamiento de los consumidores [30]. SynthDet propone realizar la misma tarea de identificación y facturación de productos, pero entrenando el modelo a partir de imágenes sintéticas.

El proyecto sobre detección de drones [32] y el de identificación de piezas industriales [7], cuentan con un trasfondo similar. La generación del *dataset*

surge por la necesidad de entrenar el modelo para finalmente llevar a cabo una identificación de objetos.

Pero la inclusión y el desarrollo de la detección y la identificación de objetos se limita al entorno de la investigación; sino que en la vida cotidiana también se percibe un crecimiento de este ámbito. Actualmente el uso de la detección e identificación de objetos se está expandiendo por más aplicaciones, una de las más conocidas y extendidas es Google Lens¹⁶, la cual tiene diversas funcionalidades dentro del mundo de la detección de objetos. Esta aplicación es una de las más completas en este dominio, a partir de una imagen o de la cámara del dispositivo escanea y traduce textos; identifica objetos y busca en Google otros similares; reconoce lugares y edificios, ofreciendo información sobre ellos; detecta animales y plantas; y registra operaciones matemáticas sobre las que ofrece explicaciones y resultados de la web.

Otra aplicación de Google con identificación de objetos es Google Fotos¹⁷, una aplicación de intercambio y almacenamiento de fotografía y vídeo. Puede utilizarse en conjunto con Google Lens para tener todas las funcionalidades mencionadas anteriormente sobre las imágenes almacenadas en la aplicación, pero además Google Fotos cuenta con su propia funcionalidad de identificación de texto, objetos y personas. En la aplicación pueden realizarse búsquedas que devuelven una selección de imágenes y vídeos en los que aparece la persona, objeto o texto buscado, además de funciones con posturas y acciones.

Otras aplicaciones similares, aunque algo más rústicas, son las recogidas en el paquete de Microsoft Office. En este caso la identificación se presenta como una funcionalidad de accesibilidad para personas con discapacidad visual. Se trata del texto alternativo, disponible para imágenes, formas, vídeos, gráficos y tablas, entre otros. En algunas de sus aplicaciones Microsoft va un paso más allá y genera el texto alternativo de las imágenes automáticamente, identificando personas, objetos y situaciones.

¹⁶<https://lens.google/intl/es-419/>

¹⁷<https://www.google.com/intl/es/photos/about/>

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 crear un generador de imágenes a partir de modelos 3D.

3.1. Introducción

Como el objetivo del proyecto es reconocer e identificar distintos hechos, 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 quizás 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, 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. 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 podrían procesar y separar los frames, usándolos después como imágenes para el entrenamiento. Para esta opción también se encuentran varios problemas. Una posibilidad para hacerlo sería grabar todos los objetos a la vez, pero esto podría tener como resultado que no hubiera contenido suficiente para el entrenamiento o que los objetos no aparecieran suficientemente claros. Otra manera de llevárselo 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 finalmente para realizar las imágenes de entrenamiento de la red neuronal. De esta forma el objetivo final de este apartado consiste en llevar a cabo una aplicación en la que se vayan cargando distintos modelos de objetos y residuos a los que se les harán numerosas capturas, en cada una aparecerán en distintas posiciones y rotaciones. Finalmente estas imágenes generadas serán posteriormente utilizadas para el entrenamiento. Gracias a esta aplicación se contará con el número de capturas que se quieran de cada objeto y material, pudiendo aumentar esta cantidad siempre que se deseé o necesite.

3.2. Unity

Unity¹ ha sido una de las principales tecnologías que se han utilizado para llevar a cabo este proyecto de final de grado, en concreto para realizar la generación sintética de imágenes. Unity es una herramienta de creación de videojuegos desarrollada por Unity Technologies. Pero esta herramienta no se limita solamente 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.

Unity permite la creación de escenas y objetos de manera muy sencilla. Además, la interacción con ellos no sólo se puede realizar a través de código, sino también mediante componentes visuales. utilizando Unity pueden desarrollarse juegos y aplicaciones para un gran abanico de plataformas; desde

¹<https://unity.com/es>

PC, Mac y Linux, pasando por Android e iOS, hasta consolas de videojuegos como PlayStation4, Xbox One, Nintendo Switch o Google Stadia.

Unity además cuenta con una clase base, `MonoBehaviour`², de la que prácticamente todos los scripts de Unity derivan. Esta puede ser activada o desactivada desde el editor de Unity según se requiera. `MonoBehaviour` ofrece numerosos métodos y mensajes, explicados en la documentación, que facilitan el desarrollo de aplicaciones en esta plataforma; los ejemplos más utilizados son las funciones `Start()`³ y `Update()`⁴. La primera se llama cuando el script está habilitado antes de que se llame al `Update()` por primera vez. El `Update()`, en cambio, es llamado en cada frame si `MonoBehaviour` está activo.

Se tenía muy claro que para llevar esto a cabo se quería utilizar algún programa que ya ofreciera muchas de las funcionalidades necesarias ya desarrolladas. Con esto se quiso evitar el tener que crear desde cero escenas, la cámara y la iluminación. Puesto que esta parte del proyecto se basa en tratar 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 Engine⁵ es otro motor de videojuegos cuyo uso también se extiende por numerosas industrias diferentes. Blender⁶ 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; con lo que se podría haber tratado de crear 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 comentadas anteriormente. Además, cuenta con una amplia documentación 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

²<https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>

³<https://docs.unity3d.com/ScriptReference/MonoBehaviour.Start.html>

⁴<https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html>

⁵<https://www.unrealengine.com/en-US/>

⁶<https://www.blender.org/>

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 durante el desarrollo debido a las limitaciones y problemas que se fueron encontrando. En esta sección del capítulo se va a hablar desde cómo se planteó la aplicación en un principio, hasta cuál fue el resultado; pasando por todas las decisiones que se tuvieron que ir tomando.

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 quería llevar a cabo una aplicación para PC con la cual se pudieran generar imágenes a partir de modelos tridimensionales. Estos modelos podrían ser los que la propia aplicación ofreciera, aquellos que se han utilizado en este proyecto, o bien podrían ser unos modelos 3D propios 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 desarrollo de estos.

Desafortunadamente esta idea tuvo que ser descartada debido a las dificultades que se sufrieron con la importación de los modelos. 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 Perception⁷ 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 SynthDet⁸ 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 ^{9 10 11}.

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. Además, así se tendría conocimiento

⁷<https://github.com/Unity-Technologies/com.unity.perception>

⁸<https://github.com/Unity-Technologies/SynthDet>

⁹Parte 1: <https://blogs.unity3d.com/2020/05/01/synthetic-data-simulating-myriad-possibilities-to-train-robust-machine-learning-models/>

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

¹¹Parte 3: <https://blogs.unity3d.com/2020/09/17/training-a-performant-object-detection-ml-model-on-synthetic-data-using-unity-computer-vision-tools/>

y control sobre todas las características de la aplicación.

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 , 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 la aplicación cada vez que se estime oportuno y tantas veces como sea necesario¹².

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

¹²Esto 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/>

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. Si alguno de los objetos no se ha cargado correctamente lo salta y 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 de coordenadas, 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 en este caso se utiliza el generado para la posición en el eje Z; puesto en positivo y negativo.

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 eje X en el que es visible 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 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.

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 no supuso mucho problema de desarrollar, sí hubo que revisarla poco después. Las pruebas que se iban realizando según se iban añadiendo y mejorando 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”¹³. Una vez identificado el origen del problema este se solucionó sin grandes esfuerzos, simplemente cambiando de lugar la carpeta contenedora de los prefabs quedó solventado.

Llegado este punto ya solo quedaba generar las imágenes y Unity cuenta con una clase enfocada a esto, la clase “ScreenCapture”¹⁴. A pesar de lo esperanzador y sencillo que se presentaba esta parte acabó no siendo así. Utilizando la clase que ofrece Unity tan solo había 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 vio 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 explicado¹⁵. 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.

Llegados a este punto del desarrollo se decidió que era momento de ampliar la cantidad de modelos y materiales para asegurarse del correcto funcionamiento y poder empezar a realizar pruebas útiles para la red neuronal. Esta parte se esperaba que, a pesar de ser lenta por el tiempo que hay que dedicarle, fuera sencilla. La parte más larga se suponía que sería la búsqueda de modelos tridimensionales útiles y semirrealistas; por otro lado la carga de los modelos en Unity se esperaba que fuera sencilla, ya que ya se había realizado

¹³<https://docs.unity3d.com/Manual>LoadingResourcesatRuntime.html>

¹⁴<https://docs.unity3d.com/ScriptReference/ScreenCapture.html>

¹⁵<https://www.youtube.com/watch?v=1T-SRLKUe5k>

en proyectos anteriores. Desafortunadamente, esto no fue del todo así; debido a que Unity está en constante crecimiento y actualización, la forma de introducir nuevos modelos 3D con sus materiales y texturas correspondientes ha sufrido cambios y ya no resulta tan sencillo como antaño. Pero la dificultad de introducir estos recursos no sólo viene de parte de Unity, la exportación de los modelos es otro factor importante; si esto no se ha llevado a cabo adecuadamente también genera problemas. En muchos casos el modelo sí se importaba adecuadamente pero ni las texturas ni los materiales se cargaban correctamente con el modelo, lo que lo hacía inservible.

Debido a todos estos problemas, la cantidad de modelos encontrados frente a los que podían realmente utilizarse era muy pequeña, para poder continuar con el desarrollo se decidió hacer pruebas simplemente con los tres tipos de materiales que se habían probado antes de llevar a cabo esta aplicación. Sobre esto se hablará en más profundidad en el capítulo 4 . Los tres materiales seleccionados fueron plástico, vidrio y metal.

Viendo las dificultades que se encontraron con la importación de los modelos se decidió prescindir de la funcionalidad para que los usuarios introdujeran sus modelos. A pesar de esta decisión, se exploró cómo se podría desarrollar esto como trabajo futuro, como resultado se encontró que Unity no proporciona ningún soporte para llevarlo a cabo, pero hay desarrolladores que han experimentado y probado formas de realizarlo¹⁶¹⁷. Una de las páginas que se referenciaban para desarrollar esta funcionalidad con .fbx fue la documentación de Autodesk¹⁸. Otras opciones que existen son dos paquetes, de pago, disponibles para Unity que lleven a cabo esta funcionalidad: ObjReader¹⁹ (sólo para archivos .obj) y TriLib2²⁰.

Con esta decisión tomada ya sólo quedaba un último detalle por desarrollar: el fondo de las imágenes. Para el fondo se eligieron un número amplio de imágenes para que así hubiera suficiente diversidad. Se compone de tres planos de los cuales para cada captura se muestran entre uno y tres de ellos. Se les asigna una rotación, en los ejes X e Y, y una imagen a cada uno. Todo esto es establecido de manera aleatoria. Tras asignar todas estas variantes, se realiza la captura.

¹⁶<https://forum.unity.com/threads/load-3d-model-at-runtime.122720/>

¹⁷<https://theslidefactory.com/loading-3d-models-from-the-web-at-runtime-in-unity/>

¹⁸<http://docs.autodesk.com/FBX/2013/ENU/FBX-SDK-Documentation/>

¹⁹<https://starscenesoftware.com/objreader.html#ObjReader>

²⁰<https://assetstore.unity.com/packages/tools/modeling/trilib-2-model-loading-package-157548#description>

3.3.4. Modelos 3D

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

Como se ha comentado, para generar imágenes sintéticas mediante esta aplicación, se parte de modelos tridimensionales. Para este proyecto todos los modelos seleccionados y utilizados son de libre uso conseguidos desde varias páginas dedicadas a esto, principalmente CGTrader²¹, Free3D²² y la propia Asset Store de Unity²³. En esta última opción no se encontraron demasiados recursos para el desarrollo, pero fueron los que se utilizaron en el inicio del proceso; posteriormente la cantidad de modelos se fue aumentando visitando las páginas mencionadas. El formato principal de que se ha tratado de conseguir para los modelos es FBX.

FBX es un formato de archivo propiedad de Autodesk, permite que estos recursos tridimensionales tengan la mínima pérdida de datos entre los distintos softwares de edición 3D²⁴. 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.

Como se comentó previamente, parecía que la carga de los modelos en Unity sería un mero trámite sin mucha complicación, pero finalmente no fue así. Se llegó a la conclusión de que el problema a veces era que la carga de los materiales y texturas no se hacía correctamente o bien que el modelo no estaba bien exportado y fallaba al importarlo en Unity. Se hicieron numerosas pruebas tratando de abrirlos desde distintos editores para intentar comprender qué ocurría con las texturas. Durante esta exploración se encontró que algunos de estos modelos incluso presentaban problemas mayores. Un ejemplo fue el modelo de una botella de plástico cuyas caras internas se renderizaran al revés, en vez de ver las caras externas .

Con algunos modelos la decepción venía en el último momento, después de conseguir que se importaran correctamente con sus materiales y texturas, el resultado de esto era muy distinto a lo que se esperaba, teniendo que ser desecharo .

Puesto que se habían conseguido muchos modelos pero muy pocos funcionaban directamente en Unity, se estudió si había alguna manera de salvar y utilizar estos modelos. En la mayoría de los casos no

²¹<https://www.cgtrader.com/>

²²<https://free3d.com/es/>

²³<https://assetstore.unity.com/>

²⁴<https://www.autodesk.com/products/fbx/overview#intro>

pudo ser, pero afortunadamente hubo unos pocos que con los que sí. Estos modelos se importaron primero en Blender, ya que es la única herramienta de modelado 3D con la que se tiene experiencia, y se volvieron a exportar los modelos a FBX pero cambiando la configuración. En algunos casos se aprovechó que el mismo recurso estaba en varios formatos y se partió de otro que no fuera FBX.

Como resultado final no se tiene una gran base de datos de modelos ni se ha podido generar un dataset amplio como se esperaba para el entrenamiento de la red neuronal. Pero lo conseguido permite generar lo suficiente como para probar y estudiar los errores y mejoras del proyecto.

3.3.5. Capturas

Como se explicó en el apartado del desarrollo, para realizar capturas se programó un script que generara las imágenes a partir de la cámara de la escena de Unity.

Para llevarlo a cabo se utilizó como referencia un tutorial de Code Monkey²⁵. Para generar las imágenes se tienen dos métodos principales. El primero establece que se quiere guardar una captura, el tamaño de esta, el directorio donde se guardará y si es en formato PNG o JPG.

El otro método, se encarga de la lógica de generar la imagen, para ello se ha utilizado el método de MonoBehaviour “OnPostRender()”²⁶. Este método se llama siempre después de que la cámara renderice la escena y ejecuta el código que se haya introducido. Para asegurarse de que no generaba imágenes en frames que no se requería, el código de este método se regula mediante un booleano. Todas las variables que este método necesita son asignadas en el método mencionado anteriormente; con esa información se permite que el código de generar capturas se ejecute y como resultado guarde la captura del tamaño y formato establecido, en el directorio indicado.

Para no perder información, el tamaño de las capturas es el mismo que la renderización de la cámara. Como se comentó previamente, las imágenes generadas se guardan dentro de una misma carpeta pero separadas en subdirectorios según a qué material corresponde el objeto de la imagen.

Unity permite guardar las imágenes en dos formatos distintos, JPG²⁷ y PNG²⁸; para este proyecto se decidió usar PNG. Esta elección

²⁵ <https://www.youtube.com/watch?v=1T-SRLKUe5k>

²⁶ <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnPostRender.html>

²⁷ JPG: <https://docs.unity3d.com/ScriptReference/ImageConversion.EncodeToJPG.html>

²⁸ PNG: <https://docs.unity3d.com/ScriptReference/ImageConversion.html>

se tomó por querer evitar la pérdida de información y que fueran lo mejor posible para el entrenamiento. Posteriormente se llegó a la conclusión de que con el formato JPG los resultados habrían sido similares pero ocupando menos espacio en el disco. En cualquier caso, el cambio de formato se hace de manera muy sencilla, es una casilla que se activa y desactiva en los componentes del GameManager desde el editor de Unity.

También, en los componentes del GameManager, se puede elegir la cantidad de imágenes que se quieren tomar de cada objeto. Este valor está puesto por defecto a 10 pero puede ampliarse y reducirse todo lo que se desee, siempre y cuando el valor sea un número positivo y entero.

Capítulo 4

Red Neuronal

Resumen: En este capítulo explican las características de la red neuronal, necesaria para llevar a cabo la identificación de objetos. Tensorflow Lite ha sido el framework seleccionado y a partir del cual se ha desarrollado el script de entrenamiento. Para elegir la mejor relación entre imágenes sintéticas y reales que utilizar para la aplicación, se ha realizado una investigación probando con diversas proporciones y comparando los resultados.

4.1. Introducción

Para llevar a cabo la identificación de objetos necesaria para el funcionamiento de la aplicación de este trabajo, es necesario utilizar técnicas y herramientas de aprendizaje automático.

El aprendizaje automático es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial. Tiene como objetivo desarrollar técnicas que permitan que las computadoras aprendan¹. Para aplicar esto a los distintos campos en los que es aprovechado, se trabaja con redes neuronales. Las redes neuronales artificiales simulan el funcionamiento de las biológicas, cuentan con un conjunto de unidades denominadas neuronas conectadas entre sí las cuales se transmiten señales².

Esas redes neuronales deben entrenarse a partir de un conjunto de datos amplio para conseguir el comportamiento buscado. Tras finalizar el entrenamiento la red obtiene un valor que indica la precisión con la que puede detectar aquello para lo que se le ha entrenado.

Como conjunto de datos para el entrenamiento de la red para este proyecto se utilizan las imágenes generadas en el capítulo anterior

¹https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

²https://es.wikipedia.org/wiki/Red_neuronal_artificial

(capítulo 3). Buscando el funcionamiento óptimo de la aplicación se probarán distintas configuraciones mezclando imágenes reales con sintéticas.

El desarrollo, a nivel de programación, de la red neuronal se lleva a cabo partiendo de los ejemplos disponibles de TensorFlow Lite, para facilitar la realización de la lógica necesaria para generar y entrenar la red neuronal.

4.2. TensorFlow y TensorFlow Lite

El equipo de Google Brain comenzó a investigar en 2011 el uso de redes neuronales a gran escala para utilizarlo en los productos de la empresa. Como resultado desarrollaron DistBelief, su primer sistema de entrenamiento e inferencia escalable. A partir de este, basándose en la experiencia y en un entendimiento más completo de las necesidades y requerimientos del sistema ideal, desarrollaron TensorFlow³. Este utiliza modelos de flujo de datos y los *mapea* en una gran variedad de diferentes plataformas, desde dispositivos móviles, como máquinas sencillas de una o varias GPUs, hasta sistemas a gran escala de cientos de máquinas especializadas con miles de GPUs. La API de TensorFlow y las implementaciones de referencia fueron lanzadas como un paquete de código abierto bajo una licencia de Apache 2.0 en noviembre de 2015; [1].

Como el nombre de TensorFlow indica, las operaciones son llevadas a cabo por redes neuronales en *arrays* multidimensionales de datos, mejor conocidos como tensores. Puesto que la estructura son grafos de flujo de datos, en estos, los nodos corresponden a las operaciones matemáticas, como sumas, multiplicaciones, factorización y demás; en cambio, los bordes corresponden a los tensores. [20].

Como se ha comentado, TensorFlow fue desarrollado para ser ejecutado en sistemas muy diversos, incluidos dispositivos móviles. Este fue llamado TensorFlow Mobile y permitió a los desarrolladores para móvil crear aplicaciones interactivas sin los retrasos que generaban los cálculos computacionales de aprendizaje automático.

A pesar de las optimizaciones para mejorar la actuación de los modelos y que el mínimo *hardware* requerido era bastante accesible; seguía existiendo cuello de botella en la velocidad de cálculo computacional por la baja latencia de los dispositivos móviles. Por ejemplo, un dispositivo móvil cuyo *hardware* era capaz de ejecutar 10 GFLOPS⁴

³<https://www.tensorflow.org/>

⁴*Giga floating point operations per second*, proveniente de FLOPS, operaciones de coma flotante por segundo.

estaba limitado a ejecutar un modelo de 5 GFLOPS a 2 4FPS⁵, lo que provocaba que las aplicaciones no funcionaran como era esperado [3].

TensorFlow Lite es la evolución de TensorFlow Mobile, algunas de las optimizaciones que incluye son el uso de *frameworks* como la API de redes neuronales de Android y redes neuronales optimizadas para móvil como MobileNets [17] y SqueezeNet[18].

TensorFlow Lite permite ejecutar modelos de aprendizaje profundo en dispositivos móviles. Estos son entrenados en una computadora y posteriormente trasladados al dispositivo sin necesidad de utilizar un servidor. Utiliza MobileNet, la cual está diseñada y optimizada para imágenes en móviles, incluyendo detección y clasificación de objetos, detección de caras y reconocimiento de lugares. Los modelos de TensorFlow Lite generados en el entrenamiento tienen como extensión de archivo *.tflite* el cual tiene formato FlatBuffer. Tensorflow Lite no se limita a los modelos que han sido directamente creados con esta extensión, sino que en la documentación de TensorFlow se encuentra un conversor que toma un modelo en otro formato y lo convierte a *.tflite*⁶.

Fue el framework elegido puesto que cumplía con todas las necesidades que requería el proyecto para el entrenamiento de la red neuronal. Con una amplia documentación y numerosos ejemplos de uso facilitando el trabajo. Además, otro factor importante es el fácil traspaso del resultado a una aplicación móvil sobre la que se hablará en profundidad en el capítulo 5, y el uso del modelo en ella sin necesidad de servidores intermedios como se contará. Asimismo la agilidad del trabajo al tratarse de plataformas en las que se tiene experiencia.

4.3. Entrenamiento

La fase de entrenamiento del proyecto está basada en las recomendaciones para generar modelos de TensorFlow Lite, ya que el modelo que se va a utilizar para la identificación es el de este formato. Para llevar a cabo el entrenamiento de la red neuronal se ha realizado un *script* que se encargue de la lógica de esto. Este *script* está basado en el ejemplo de Tensorflow Lite y puede usarse desde Google Collab, donde está disponible para entrenar los modelos que se deseé⁷. Este utiliza la librería Task de TensorFlow Lite, la cual contiene herramientas para que los desarrolladores creen experiencias de aprendizaje automático con Tensorflow Lite, tales como un clasificador de imágenes,

⁵*frames* por segundo, del inglés *frames per second*.

⁶<https://www.tensorflow.org/lite/convert?hl=es-419>

⁷<https://colab.research.google.com/drive/1sqBewUnvdAT00-yblj55EBFb2sM24XHR?hl=es-419>

detector de objetos o clasificador de lenguaje natural, entre otras. También la biblioteca Model Maker, la cual simplifica el proceso de adaptación y conversión de un modelo de red neuronal de TensorFlow para aplicaciones de aprendizaje automático. Y por último, MobileNetV2[34], que utiliza convolución separable en profundidad haciendo la red más eficiente.

Para este proyecto se desarrolló el *script* en el dispositivo para así evitar tener que cargar tantas imágenes a la plataforma de Google Collab y descargar el modelo constantemente para utilizarlo. Basándose en el que proporciona TensorFlow de ejemplo, el archivo resulta bastante similar pero con unos cambios personalizados. El ejemplo está planteado para cargar todas las imágenes y después separarlas en entrenamiento y *test*, cada grupo tendrá el porcentaje de imágenes que se le establezca. Por ejemplo, por defecto se reparten en el 90 % de las imágenes para entrenamiento y el 10 % restante para validación. Pero esta no es la funcionalidad requerida para este proyecto. Puesto que el entrenamiento se va a realizar a partir de imágenes generadas y posteriormente se va a utilizar sobre objetos reales, para comprobar la viabilidad del modelo es necesario que las imágenes de *test* sean imágenes reales. Por lo tanto, en este caso, es necesario cargar dos carpetas diferentes para los dos grupos. Puesto que estas siguen teniendo que estar separadas por el material al que pertenecen(plástico, metal, vidrio, etc.) dentro de las dos carpetas para entrenamiento y *test* se encuentran las imágenes separadas en subcarpetas según esta distribución. Para mayor claridad, la estructura de carpetas queda como se representa en la figura 4.1. En esta separación se ha mantenido la proporción aproximada de 90% de imágenes de entrenamiento y 10% de validación para todos los materiales.

El entrenamiento se divide en varios *epochs*. Los *epochs* son el número de veces que el algoritmo recorre todos los datos. Estos datos se dividen en *batches*, y en cada uno de estos se recoge una pequeña parte de los datos. Cada *batch* se recorre en una iteración por lo que cada *epoch* tiene tantas iteraciones como faltan para recorrer al completo el conjunto de datos. Esto resulta en que en cada *epoch* el número de iteraciones es el resultado de dividir la cantidad de datos entre el tamaño de *batch*, [39]. Por ejemplo, si se tienen 2400 imágenes y el tamaño de *batch* es 32, en cada *epoch* habría 75 iteraciones. Para estos valores se han mantenido los de por defecto de TensorFlow ya que se ha considerado que eran adecuados; siendo el número de *epochs* 5 y el tamaño de *batch* 32.

Con todo lo mencionado se genera el modelo entrenado con extensión *.tflite*, además de un documento de texto plano con las etiquetas de los distintos materiales. Estos dos archivos son lo que se han

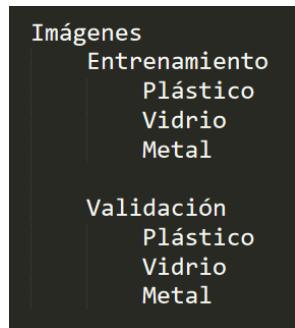


Figura 4.1: Organización de las carpetas con las imágenes separadas.

de trasladar a la aplicación móvil para ser utilizados.

4.4. Resultados

Con las imágenes sintéticas generadas y el *script* de entrenamiento de la red neuronal programado, se pasa a investigar con qué porcentaje de imágenes sintéticas se obtiene un entrenamiento óptimo de la red que permita el correcto funcionamiento de la aplicación.

Esta investigación se ha llevado a cabo con las opciones para el prototipado de la aplicación debido a las dificultades mencionadas en la sección 3.3.4 con la obtención de modelos tridimensionales. Esto significa que se ha limitado el entrenamiento a utilizar imágenes reales para dos de los tres materiales seleccionados (vidrio y plástico). En cambio, para el material restante (metal) llevar a cabo las pruebas y la comparación de precisión de la red mezclando imágenes reales y las propias generadas. , Se ha seleccionado este material debido a que es del que más modelos tridimensionales se tienen y por lo tanto es para el que más imágenes pueden generarse. El resto de imágenes utilizadas se han obtenido de datasets abiertos al uso público de Kaggle⁸.

Para la comparación se ha comenzando con sólo imágenes reales y se han ido aumentando paulatinamente en un 10% las imágenes generadas hasta contar con un *dataset* de sólo imágenes sintéticas. La figura 4.2 corresponde al crecimiento de la precisión de la red durante el entrenamiento; se encuentra representado para los distintos porcentajes de imágenes generadas y reales. En todos los casos crece de manera similar sin haber grandes diferencias según la proporción de imágenes sintéticas.

En la figura 4.3, por el contrario, se observa cómo varía la precisión

⁸<https://www.kaggle.com/>

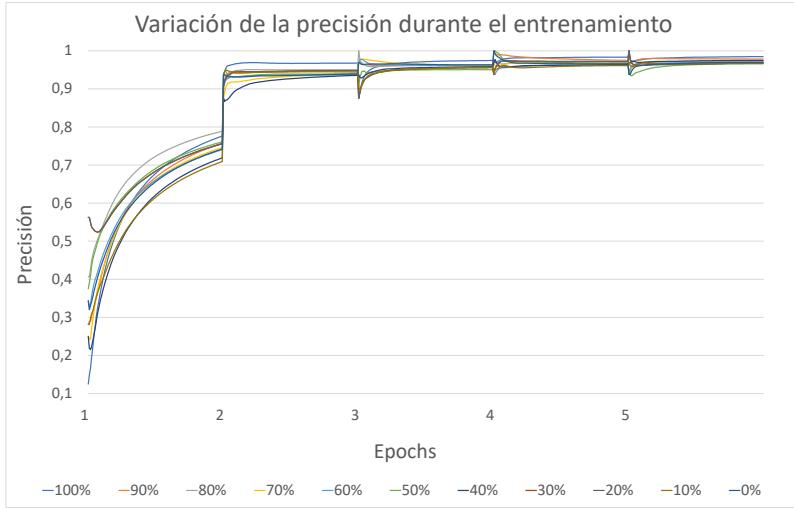


Figura 4.2: Variación de la precisión de la red neuronal a lo largo del entrenamiento de esta.

al probarse el modelo con los datos de *test*, aquí se sacan las primeras conclusiones sobre la red. Se observa que la mayoría de las opciones se mantienen casi todo el proceso en una precisión mayor del 90%. Se pueden destacar los datasets con los porcentajes de 0% y 10% de imágenes reales cuya precisión se queda muy por debajo de la del resto. La figura 4.4 muestra el valor final de la precisión en los distintos casos. A partir de esta última figura pueden sacarse las conclusiones de con qué porcentaje se obtendrían los mejores resultados.

Los dos *datasets* que cuentan con la mayor cantidad de imágenes generadas serían los menos recomendados para utilizar debido a su baja precisión. En cambio, a partir del 80% de imágenes generadas se observa que la precisión siempre está por encima del 90%, aunque nunca llega a superar el 95%.

Con los resultados obtenidos se considera como mejor opción utilizar un 70% de imágenes generadas para el *dataset* final. Esta opción es de las que mayor precisión tienen y a la vez permite tener un *dataset* que funciona adecuadamente formado principalmente por imágenes sintéticas, lo que facilita la obtención de este.

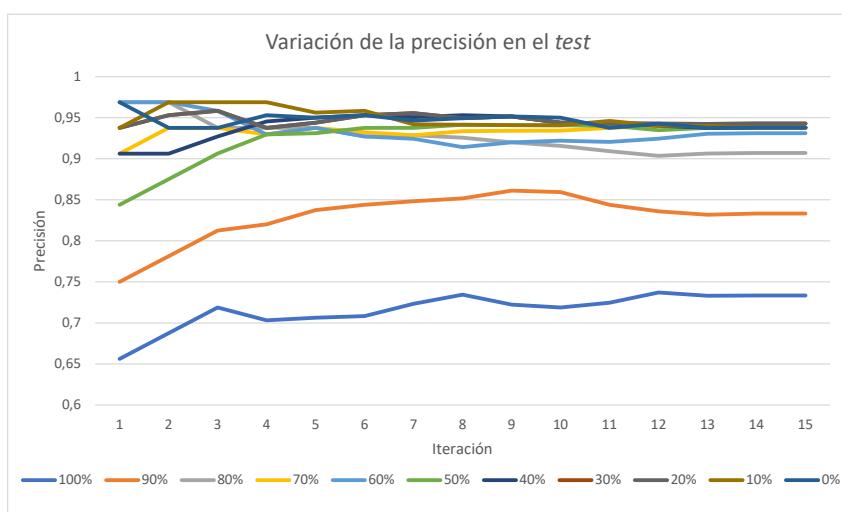


Figura 4.3: Variación de la precisión de la red neuronal durante las pruebas de esta.

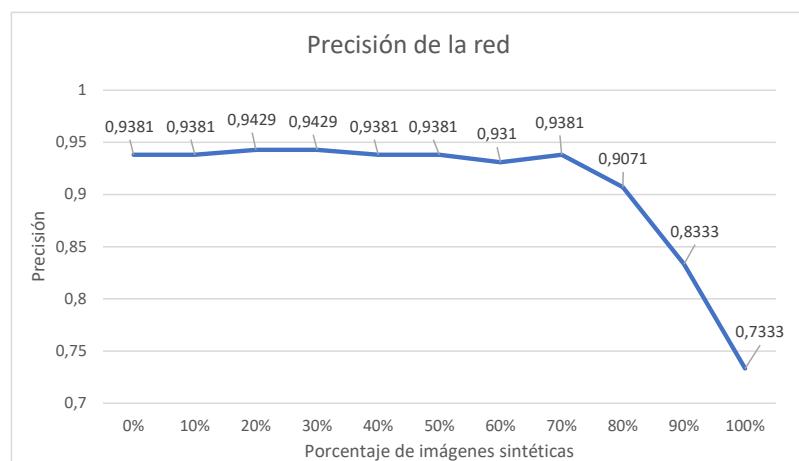


Figura 4.4: Variación de la precisión de la red neuronal según la proporción de imágenes reales y sintéticas.

Capítulo 5

Aplicación de identificación de objetos

Resumen: En este capítulo se trata el desarrollo de la aplicación móvil. Esta aplicación tiene como objetivo identificar y diferenciar diversos objetos y residuos que una vez identificado se explica al usuario cuál es la forma adecuada de desecharlos y reciclarlos. La aplicación es para dispositivos con sistema operativo Android y está llevada a cabo en Android Studio con Java y la librería de TensorFlow Lite para introducir la identificación.

5.1. Introducción

Como se ha comentado, la aplicación de identificación de objetos para reciclaje es el objetivo principal de este trabajo de fin de grado. Se trata de una aplicación para dispositivos móviles la cual, utilizando la cámara del dispositivo en el que está instalada, identifica el material del objeto al que está enfocando el usuario. En la interfaz aparecen las tres opciones con más probabilidad, acompañadas del porcentaje de confianza respecto a esa opción. Independientemente de cuántas etiquetas de materiales distintas hay, siempre se muestran las tres con mayor confianza. La aplicación está constantemente interpretando lo que recibe desde la cámara, así que las etiquetas y sus respectivos porcentajes se actualizan ininterrumpidamente sin necesidad de estar tomando fotografías para cada elemento.

El desarrollo se ha centrado en dispositivos con sistema operativo Android puesto que el acceso a las otras opciones posibles, iOS y Raspberry, resultaba inviable.

5.2. Tensorflow Lite

Como se presentó en la sección 4.2, para el desarrollo del proyecto se han empleado las herramientas que brinda TensorFlow Lite, framework desarrollado por Google para aprendizaje profundo en dispositivos móviles. Además de facilitar el entrenamiento de redes neuronales como se explicó en el capítulo anterior, también simplifica la importación de los modelos generados y su uso en distintas plataformas. Los modelos de TensorFlow Lite pueden manejarse desde dispositivos Android, iOS y Raspberry, y para el desarrollo de una aplicación para cada una de estas opciones pueden utilizarse Android Studio, Swift y Objective-C; y Python, respectivamente.

Para el correcto funcionamiento, en Andorid, de la aplicación con el modelo entrenado son necesarias dos librerías. La primera es la librería de tareas de TensorFlow Lite, que cuenta con un conjunto de bibliotecas específicas de tareas potentes y fáciles de usar por los desarrolladores para crear experiencias de aprendizaje automático. Esta funciona varias plataformas y es compatible con Java y C++. La segunda librería necesaria es la de compatibilidad, esta facilita la integración de modelos en la aplicación. Proporciona API de alto nivel que ayuda a transformar los datos de entrada en el formato requerido por el modelo, además de interpretar su salida.

5.3. Identificación de objetos

La aplicación desarrollada está basada en el ejemplo que proporciona TensorFlow Lite sobre reconocimiento de flores¹. El funcionamiento es similar pero se ha adaptado de identificar flores a diferenciar distintos materiales. Como se comentó en la sección 4.3, cuando se explicaba el entrenamiento de la red neuronal, como prototipado se ha trabajado simplemente con tres materiales (metal, vidrio y plástico). Para el correcto funcionamiento de la aplicación en Android es importante que haya al menos tres objetivos a identificar; esto es debido a que la librería de TensorFlow Lite está planteada de tal forma que muestra las tres opciones sobre las que tiene más confianza de qué se trata. Si se utilizan menos de tres no funciona.

Para utilizar TensorFlow Lite en primer lugar se han de incluir las librerías para poder utilizar sus funcionalidades. Una vez hecho esto se importa el intérprete, este carga y ejecuta el modelo dándole una serie de datos de entrada. Tras ejecutarlo escribe los resultados por pantalla.

¹<https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/?hl=es-419#0>

La aplicación hace uso de la cámara del dispositivo, lee *frames* desde esta y los convierte a imágenes, estos son los datos que se utilizan como entrada para el modelo. Dichos datos de entrada se reciben desde la cámara en formato *bitmap*, matrices donde cada casilla tiene asignado un color y que en conjunto forman una imagen. Ese *bitmap* es convertido a *byte buffer*, denominado *IMG data*, lo que lo hace legible para el modelo entrenado. Además utilizar Byte buffers como entrada permite que la API de Java sea más rápida². Estos datos se cargan en el intérprete de TensorFlow Lite y finalmente se obtienen los valores del resultados. Estos valores que se obtienen son índices respecto a las etiquetas y la probabilidad de que esa imagen corresponda a esa etiqueta, y se devuelven en un array correspondiendo cada posición de este a cada una de las etiquetas disponibles.³ Finalmente, se seleccionan las tres etiquetas con mayor probabilidad que son mostradas por la interfaz; por esto se comentaba previamente que es importante tener al menos tres objetivos de identificación para el correcto funcionamiento de la aplicación.

Respecto a la interfaz de usuario de la aplicación, se ha mantenido semejante a la proporcionada en el ejemplo. En la parte superior aparece el logo de Tensorflow, en la parte central se muestra lo que se está apuntando con la cámara y en la parte inferior se tiene un panel. En este panel es donde se encuentran las etiquetas sobre el objeto identificado y las probabilidades correspondientes a cada una. Este, además, se despliega hacia arriba y mostrando varias características del proceso. Estas son la resolución con la que se recogen las imágenes, la rotación del dispositivo y el tiempo de inferencia, que es el tiempo que tarda en detectar de qué objeto se trata. Además de estas características, cuenta con dos opciones configurables por el usuario que permiten la mejora de la aplicación. La primera es la elección de cuántos hilos utilice, estos permiten acelerar la ejecución de los operadores. Sin embargo, el aumento de hilos utilizados hará que se utilicen más recursos y potencia. La otra opción es elegir entre utilizar la CPU o la GPU. La GPU es uno de los aceleradores que TensorFlow Lite puede aprovechar. Si la GPU del dispositivo es compatible con las especificaciones de TensorFlow Lite se utilizará esta, si no, será la CPU con cuatro subprocesos. Según el dispositivo y la opción que se utilice la velocidad variará⁴.

²https://www.tensorflow.org/lite/performance/best_practices?authuser=1

³https://www.youtube.com/watch?v=JnhW5tQ_7Vo

⁴<https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android/?hl=es-419#7>

5.4. Pruebas

Con todo lo necesario desarrollado, se llevan a cabo varias pruebas para comprobar el funcionamiento de la aplicación de identificación. Para tener suficientes datos con los que comparar el funcionamiento se escogen cuatro de los modelos entrenados con los distintos *datasets*. Estos son los dos modelos extremos, correspondiendo a los compuestos al completo por imágenes sintéticas y por imágenes reales, respectivamente; el modelo óptimo, con el 70% de imágenes generadas; y, por último, el simétrico del anterior, con el 30% de imágenes sintéticas.

Para las comparaciones se han cogido diversos objetos y se ha estudiado qué etiqueta cuenta con el mayor valor de confianza para cada uno. También se tiene en cuenta aquellos casos en que las dos primeras etiquetas tienen porcentajes muy similares y por lo tanto van cambiando entre primera y segunda posición.

El objeto de estudio principal han sido los objetos metálicos, ya que son aquellos que han sido entrenados con imágenes generadas. En la figura 5.1 se pueden observar los resultados en los distintos casos con una lata de atún. A excepción del modelo entrenado por completo con imágenes sintéticas, los demás detectan que se trata de una lata con conveniente confianza. Algo similar ocurre al poner como objetivo la tapa metálica de un bote.

Con un tercer objeto metálico diferente los resultados cambian en el modelo con el 70% de imágenes sintéticas. El resto se mantienen prácticamente igual, variando levemente la confianza, pero manteniendo la misma etiqueta que en los otros casos. En cambio, con el modelo mencionado se observa que el porcentaje queda equilibrado entre las tres etiquetas como se exemplifica en la figura 5.3.

Una observación importante que se descubre, es la dificultad, en todas las opciones, para distinguir los objetos de vidrio respecto a los de plástico. Para tres objetos de vidrio diferentes todas los identifican como plástico con más de un 60% de confianza. Esto no genera mucha preocupación puesto que como seres humanos a veces también es costoso diferenciar plástico translúcido de vidrio a simple vista.

Un problema importante que se ha observado es la variabilidad del resultado dependiendo de la iluminación. Como se puede observar en la figura 5.2 para el mismo objeto y con el mismo modelo, en este caso el generado a partir de solamente imágenes reales, se obtienen resultados diferentes al cambiarlo de posición.

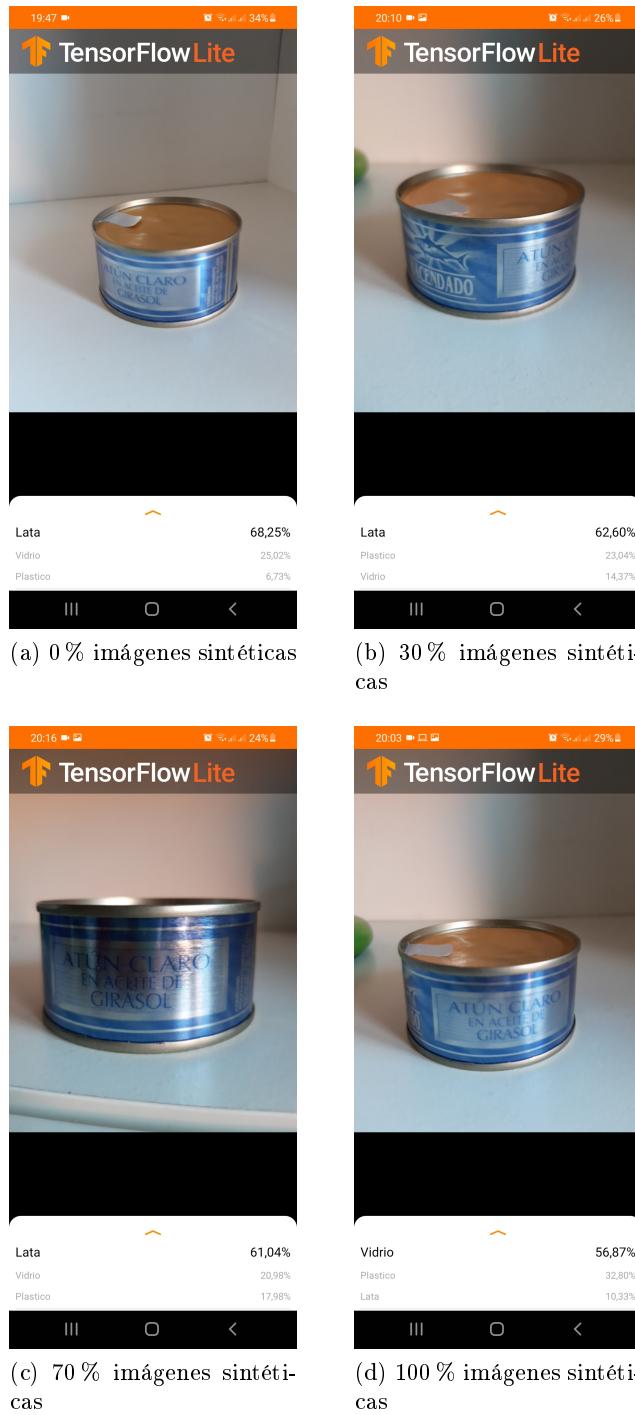


Figura 5.1: Comparación con una lata de atún

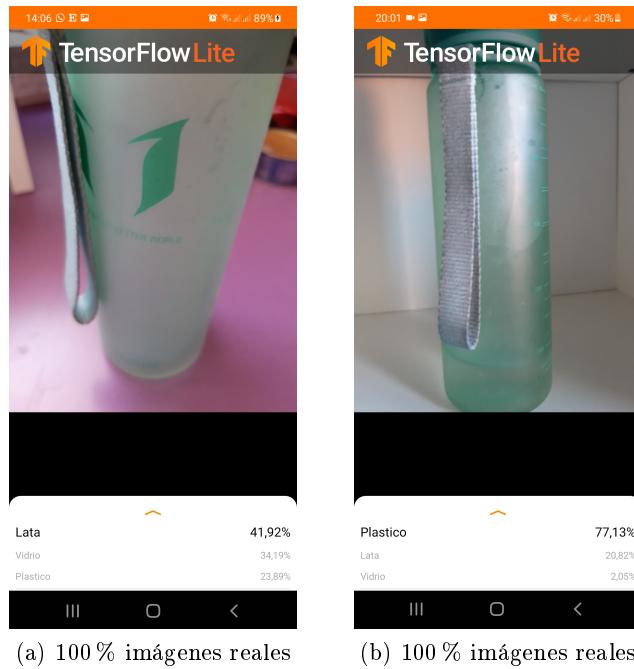


Figura 5.2: Comparación con una botella de agua.



Figura 5.3: Resultados del modelo 70-30 con una lata de bálsamo labial.

Capítulo 6

Conclusiones y trabajo futuro

6.1. Conclusiones

6.2. Trabajo futuro

Bibliografía

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] C. Alfonso, R. Estévez Estévez, J. M. Lobo, B. Lozano Diéguez, F. Prieto, J. Santamarta, and A. Gaerter. Emergencia climática en España. Diciembre 2016.
- [3] O. Alsing. Mobile object detection using tensorflow lite and transfer learning. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.
- [4] Y. Amit. *2D Object Detection and Recognition: Models, Algorithms, and Networks*. Mit Press. MIT Press, 2002.
- [5] Y. Amit, P. Felzenszwalb, and R. Girshick. *Object Detection*. Springer International Publishing, Cham, 2020.
- [6] X. Basogain Olabe. Redes neuronales artificiales y sus aplicaciones. *Dpto. Ingeniería de Sistemas y Automática, Escuela Superior de Ingeniería Bilbao. Open Course Ware.* [En línea] disponible en http://ocw.ehu.es/ensenanzas-tecnicas/redes-neuronales-artificiales-y-sus-aplicaciones/Course_listing. [Consultada 20-09-2012], 2008.
- [7] J. Cohen, C. F. Crispim-Junior, C. Grange-Faivre, and L. Tougne. CAD-based Learning for Egocentric Object

- Detection in Industrial Context. In *15th International Conference on Computer Vision Theory and Applications*, volume 5, Valletta, Malta, Feb. 2020. SCITEPRESS - Science and Technology Publications.
- [8] G. Cortina Fernández. Técnicas inteligentes para su integración en un vehículo autómata. Trabajo de Fin de Grado en Ingeniería del Software, Facultad de Informática UCM, Departamento de Ingeniería del Software e Inteligencia Artificial, Curso 2019/2020., 2020.
- [9] B. Cyganek. *Object Detection and Recognition in Digital Images: Theory and Practice*. Wiley, 2013.
- [10] R. Flórez López, J. M. Fernández, and J. M. Fernández Fernández. *Las Redes Neuronales Artificiales. Metodología y Análisis de Datos en Ciencias Sociales*. Netbiblo, 2008.
- [11] R. Fonfría, R. Sans, and J. de Pablo Ribas. *Ingeniería ambiental: contaminación y tratamientos*. Colección productiva. Marcombo, 1989.
- [12] S. Frintrop. *VOCUS: A visual attention system for object detection and goal-directed search*, volume 3899. Springer, 2006.
- [13] G. A. Gómez Rojas, J. C. Henao López, and H. Salazar Isaza. Entrenamiento de una red neuronal artificial usando el algoritmo simulated annealing. *Scientia Et Technica*, 2004.
- [14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative adversarial networks*, 2014.
- [15] G. Guridi Mateos et al. Modelos de redes neuronales recurrentes en clasificación de patentes. B.S. thesis, 2017.
- [16] J. R. Hilera and V. J. Martínez Hernando. *Redes neuronales artificiales: fundamentos, modelos y aplicaciones*. 01 1995.
- [17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

- [18] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size, 2016.
- [19] Y.-C. Jhang, A. Palmar, B. Li, S. Dhakad, S. K. Vishwakarma, J. Hogins, A. Crespi, C. Kerr, S. Chockalingam, C. Romero, A. Thaman, and S. Ganguly. Training a performant object detection ML model on synthetic data using Unity Perception tools, Sep 2020.
- [20] R. Karim. *TensorFlow: Powerful Predictive Analytics with TensorFlow*. Packt Publishing, Limited, 2018.
- [21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
- [22] P. Larranaga, I. Inza, and A. Moujahid. Tema 8. redes neuronales. *Redes Neuronales, U. del P. Vasco*, 12, 1997.
- [23] M. A. López Pacheco. Identificación de sistemas no lineales con redes neuronales convolucionales. *Cuidad de México: Centro de investigación y de estudios avanzados*, 2017.
- [24] D. J. Matich. Redes neuronales: Conceptos básicos y aplicaciones. *Universidad Tecnológica Nacional, México*, 41, 2001.
- [25] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 1943.
- [26] B. Müller, J. Reinhardt, and M. Strickland. *Neural Networks: An Introduction*. Physics of Neural Networks. Springer Berlin Heidelberg, 1995.
- [27] C. Parra Ramos and D. Regajo Rodríguez. Reconocimiento automático de matrículas. *Universidad Carlos III de Madrid*, 2006.
- [28] R. Pavón Benítez. Técnicas de deep learning para el reconocimiento de movimientos corporales. Trabajo de Fin de Grado en Ingeniería del Software, Facultad de Informática UCM, Departamento de Ingeniería de Software e Inteligencia Artificial, Curso 2019/2020, 2020.

- [29] A. Polacco and K. Backes. The amazon go concept: Implications, applications, and sustainability. *Journal of Business and Management*, 24(1), 2018.
- [30] A. Polacco and K. Backes. The amazon go concept: Implications, applications, and sustainability. *Journal of Business and Management*, 24(1), 2018.
- [31] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [32] A. Rozantsev, V. Lepetit, and P. Fua. On rendering synthetic images for training an object detector. *Computer Vision and Image Understanding*, 137, 11 2014.
- [33] M. Sánchez and J. Castro. *Gestión y Minimización de Residuos*. Fundación Confemetal, 2007.
- [34] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.
- [37] Unity Technologies. Unity Perception package, 2020.
- [38] S.-C. Wang. *Artificial Neural Network*. Springer US, Boston, MA, 2003.
- [39] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2019.
- [40] W. Yu and Y. Bai. Visualizing and comparing alexnet and vgg using deconvolutional layers. 2016.

- [41] J. Zurada. *Introduction to Artificial Neural Systems.* West, 1992.

