# 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**

# 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(Sánchez y Castro, 2007). 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 $CO_2$ 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 (Fonfría, Sans y de Pablo Ribas, 1989).

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 (Alfonso, Estévez, Lobo, Diéguez, Prieto, Santamarta y Álvaro Gaerter, 2016).

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 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 platilla Texis.

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. Con la generalización del uso de los dispositivos móviles en la población favorece a que esta sea la mejor forma de encontrar esta ayuda.

En España, para empezar, existe la aplicación de "AIRE Asistente Inteligente de Reciclaje" disponible tanto para iOS[1] como Android[2] aplicación publicada por "ecoembes", empresa encargada del reciclaje de los residuos de los contenedores amarillo y azul en España.

Carrefour, por ejemplo, también cuenta con una aplicación de ayuda para reciclaje[3], esta a partir del código de barras del ticket de la compra que se haya hecho indica cómo reciclar los envases de los productos de algunas marcas.

En el lado internacional se encuentran aplicaciones como RecycleRight, Brisbane Bin and Recycling, Grow Recycling o Recycle Coach, aunque distintas entre ellas todas estas aplicaciones cuentan con un factor de explicación y aprendizaje sobre reciclaje para el usuario.

## 2.2. Redes neuronales

Redes neuronales más usadas. En qué se usan. Aplicaciones que las utilicen.

[1] `https://apps.apple.com/es/app/air-e/id1442582214`

[2] `https://play.google.com/store/apps/details?id=com.ecoembes.aire&hl=en`

[3] `https://www.reciclaya.app/es/como-funciona`

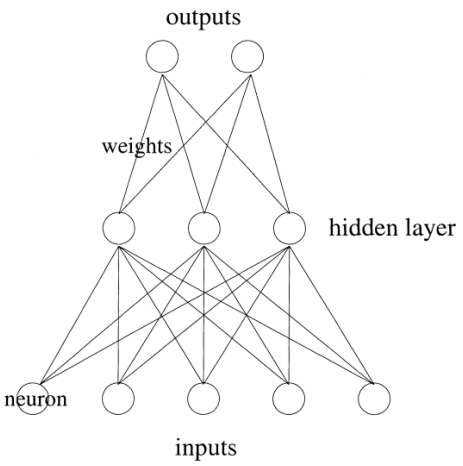


Figura 2.1: Estructura de una red neuronal artificial.

Las redes neuronales artificiales vienen inspiradas por la funcionalidad sofisticada del cerebro humano donde miles de millones de neuronas se encuentran interconectadas y procesan información de manera paralela. Una red neuronal artificial consta de una capa de entrada de neuronas, una o varias capas ocultas de neuronas y una capa final de neuronas de salida. En la figura2.1 se muestra la estructura habitual de una red neuronal artificial. Se muestra mediante líneas la conexión de las neuronas. Cada una de estas conexiones se asocia con un valor numérico llamado peso. Wang (2003);

## 2.3. Generación de imágenes

Esta parte, de generación de imágenes, se encuentra menos explorada que las demás del capítulo. Aún así encontramos numerosos acercamientos desde distintas áreas de investigación.

En este ámbito se encuentran las Redes Generativas Adversativas, que se relaciona con el tratamiento de imágenes. Este modelo consta de dos redes neuronales denominadas generador y discriminador, y 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 discriminador distingue entre los datos reales que se le proporcionan y los generados por la red anterior. Esto fue propuesto en 2014 en el artículo "Generative Adversarial Networks" [4].

[4] `https://arxiv.org/abs/1406.2661`

## 2.4. Identificación de objetos e imágenes

Google Lens, Google Photos, Microsoft Office ->ejemplos más conocidos.

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

Conseguir tantas imágenes, distintas y claras, supone un trabajo costoso y lento; sin tener en cuenta el almacenaje de estas. Contamos con varias opciones para conseguir las imágenes. Una de ellas, y quizá la más obvia, es descargándolas de la red. Esta opción cuenta con un par de problemas a tener en cuenta. El primero es lo tedioso que resulta este procedimiento; buscar, seleccionar y descargar una a una cientos de imágenes resulta muy lento. El otro problema, 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 llevarlo a cabo es grabando los distintos objetos individualmente; en este caso habría

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

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

Actualmente se va viendo una gran mejoría en los gráficos de contenidos digitales audiovisuales, ya sean películas, videojuegos, videoclips, anuncios, etc. Esta mejora llega hasta tal punto que a veces puede resultar difícil distinguir entre realidad y CGI (Computer-generated imagery). Teniendo esto en cuenta y las oportunidades que aporta la automatización, no se ha querido desaprovechar la oportunidad de enfocarlo desde un punto más cercano a este. Esta es la opción que se eligió 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 desee o necesite.

## 3.2. Unity

Unity[1] ha sido una de las principales tecnologías que se han utilizado para llevar a cabo este proyecto de final de grado, en concreto para realiza la generación sintética de imágenes. Unity es una herramienta de creación de videojuegos desarrollada por Unity Technologies. Pero esta herramienta no se limita 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 Engine[5] es otro motor de videojuegos cuyo uso también se extiende por numerosas industrias diferentes. Blender[6] 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 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

[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/`

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 {**TODO TODO TODO:** referenciar capítulo 5 AplicacionIdentificación} 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. {**TODO TODO TODO:** referenciar subsecciones futuras} Tras invertir tiempo investigando sobre cómo importar modelos desde ejecución, los resultados dejaron mucho que desear. No sólo fue la ausencia de información sobre este tema, sino además la complicación de introducir modelos al propio editor de Unity lo que provocó la decisión de abandonar este objetivo.

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

### 3.3.2. Planteamiento

Existe un paquete para Unity llamado Perception[7] el cual genera datos etiquetados a partir de modelos 3D. Esto resulta de lo más interesante debido a la similitud que tiene con este apartado del proyecto. Unity Perception proporciona herramientas para generar datasets de gran tamaño utilizables en el entrenamiento y la validación en proyectos de visión computacional.

Para investigar y aprender sobre esta herramienta se ha elegido tomar el proyecto SynthDet[8] 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 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 {**TODO TODO TODO:** referencia capítulo de la identificación} , esta aplicación se ha desarrollado basándose en el ejemplo disponible de TensorFlow Lite sobre clasificación de imágenes; por lo tanto la aplicación se centra e identifica un único objeto,

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

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

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

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

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

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 necesario[12].

La entidad encargada de los modelos es la que carga e instancia los prefabs de estos. Cuando se llama al método de cargar un nuevo material desde el GM, esta entidad accede a la subcarpeta dentro de "Prefabs" del material correspondiente, este se indica como un parámetro del método. Si la carpeta de ese material existe, se cargan en un array todos los recursos en ella casteándolos a GameObjects. Además, lleva la cuenta de los objetos instanciados y cuántos quedan de ese tipo. 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

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

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.

[13] `https://docs.unity3d.com/Manual/LoadingResourcesatRuntime.html`

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 explicado[15]. 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 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 {**TODO TODO TODO:** añadir referencia al capítulo del entrenamiento} . Los tres

[14] `https://docs.unity3d.com/ScriptReference/ScreenCapture.html`
[15] `https://www.youtube.com/watch?v=lT-SRLKUe5k`

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 hay experimentado y probado formas de realizarlo [16][17]. Una de las páginas que se referenciaban para desarrollar esta funcionalidad con .fbx fue la documentación de Autodesk [18]. Otras opciones que existen son dos paquetes, de pago, disponibles para Unity que lleven a cabo esta funcionalidad: ObjReader[19] (sólo para archivos .obj) y TriLib2[20].

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.

### 3.3.4. Modelos 3D

{**TODO TODO TODO:** teniendo en cuenta que podrían existir complicaciones con la obtención de los modelos se consiguieron datasets de imágenes de objetos para poder hacer pruebas de prototipado. Teniendo en cuenta los materiales que se pudieron obtener (plástico, vidrio y latas) se continuó en esa línea en la obtención de los modelos para tener un dataset pequeño, y poco variado, pero completo con el que poder hacer pruebas sin problema. Como se comentará en el capítulo 5 {**TODO TODO TODO:** referencia a capitulo 5} para el correcto funcionamiento de la aplicación son necesarios al menos tres objetivos a identificar diferentes. } 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 CGTrader[21], Free3D[22] y la propia Asset Store de Unity[23]. En esta última opción no se encontraron

[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

[21] https://www.cgtrader.com/

[22] https://free3d.com/es/

[23] https://assetstore.unity.com/

demasiados recursos para el desarrollo, pero fueron los que se utilizaron en el inicio del proceso; posteriormente la cantidad de modelos se fue aumentado 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 3D[24]. 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 {**TODO TODO TODO:** poner imágenes}.

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 desechado {**TODO TODO TODO:** añadir imágenes. Lata cocacola polaca}.

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 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.

[24] `https://www.autodesk.com/products/fbx/overview#intro`

### 3.3.5. Capturas

{**TODO TODO TODO:** cómo se nombran las capturas y que habría que revisar para los objetos grandes y pequeños.}

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 Monkey [25]. 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, JPG[27] y PNG [28]; para este proyecto se decidió usar PNG. Esta elección 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 se un número positivo y entero.

[25] `https://www.youtube.com/watch?v=lT-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.EncodeToPNG.html`

# Capítulo 4

# Red Neuronal

**Resumen:**

## 4.1. 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 TensorFlow. 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 GPU, 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; puede encontrarse todo en su página web[1] (**?**).

Como el nombre de TesorFlow 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. (Karim, 2018).

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

[1] `https://www.tensorflow.org/`

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 GFLOPS[2] estaba limitado a ejecutar un modelo de 5 GFLOPS a 2 FPS[3], lo que provocaba que la aplicación no funcionara como era esperado (Alsing, 2018).

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 (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto y Adam, 2017) y SqueezeNet(Iandola, Han, Moskewicz, Ashraf, Dally y Keutzer, 2016). 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*[4].

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.

## 4.2. 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

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

[3] *frames* por segundo, del inglés *frames per second*.

[4] `https://www.tensorflow.org/lite/convert?hl=es-419`

modelos que se desee[5]. Este utiliza la librería Task de TensorFlow Lite, la cual contiene herramientas para que los desarrolladores creen experiencias de AA con Tensorflow Lite tales como un clasificador de imágenes, detector de objetos o clasificador de lenguaje naturar, 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 AA. Y por último, MobileNetV2, que utiliza convolución separable en profundidad haciendo la red más eficiente.

Para este proyecto se prefirió tener disponible el *script* en el dispositivo y así evitar tener que cargar tantas imágenes a la plataforma y, además, el 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* tendrán que estar 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 *batchs*, en cada uno de estos se recoge una pequeña parte de los datos, y cada *batch* se recorre en una iteración. Cada *epoch* tiene varias iteraciones, tantas como 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*. Por ejemplo, si se tienen 2400 imágenes y el tamaño de *batch* es 32, en cada *epotch* 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 dis-

[5] `https://colab.research.google.com/drive/1sqBewUnvdAT0O-yblj55EBFb2sM24XHR?hl=es-419`

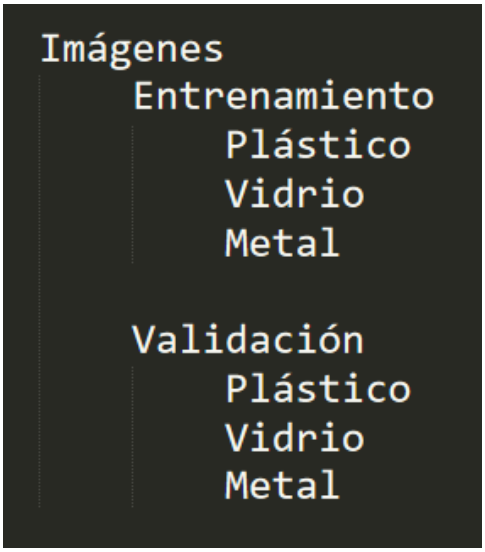


Figura 4.1: Organización de las carpetas con las imágenes separadas.

tintos materiales. Estos dos archivos son lo que se han de trasladar a la aplicación móvil para ser utilizados.

## 4.3. Resultados

Con las imágenes sintéticas generadas y el *script* de entrenamiento de la red neuronal programado, se dio pie a investigar con qué porcentaje de imágenes sintéticas se obtenía un entrenamiento óptimo de la red que permitiera el correcto funcionamiento de la aplicación posteriormente. 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) y con el restante (lata), y con el que más imágenes podían generarse, llevar a cabo las pruebas y comparación de precisión de la red mezclando imágenes reales y las propias generadas. Todas las imágenes utilizadas se han obtenido de datasets abiertos al uso público de Kaggle[6]

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 entrenamiento de sólo imágenes sintéticas. La figura {**TODO TODO TODO:** meter gráfica entrenamiento} corresponde al crecimiento de la precisión de la red durante el entrenamiento; se encuentra representado para los distintos porcentajes de imágenes generadas y reales.

[6] `https://www.kaggle.com/`

En todos los casos crece de manera similar sin haber grandes diferencias según la proporción de imágenes sintéticas. Por otro lado, en la figura {**TODO TODO TODO:** meter gráfica de test} se observa cómo varía la precisión al probarse el modelo con los datos de *test*. La figura {**TODO TODO TODO:** figura de la precisión final} 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.

Una vez desarrollada la aplicación de identificación de objetos se probaron cuatro de los modelos entrenados para corroborar el correcto o mal funcionamiento, respectivamente, de cada uno. Para ello se eligieron los modelos extremos con todas las imágenes reales o todas las imágenes sintéticas; el modelo intermedio, dividido en mitad y mitad; y el modelo con mejor precisión, repartido en 70 % imágenes reales y 30 % generadas.

# Capítulo 5

# Aplicación de Identificación de Objetos

**Resumen:** En este capítulo se habla del 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. Tensorflow Lite

TensorFlow Lite no sólo facilita el trabajo de entrenar la red neuronal, sino también el uso del modelo generado en la aplicación. Cuenta con dos bibliotecas distintas, la primera es la biblioteca de tareas[1] la cual contiene

Para este proyecto se decidió aprovechar el ejemplo que

## 5.2. Identificación de objetos

{**TODO TODO TODO:** Qué hace ->apuntas a un objeto saca con qué porcentaje considera que es cierto material.}

...

...

[1] `https://www.tensorflow.org/lite/inference_with_metadata/task_library/overview`

# Capítulo 6

# Conclusiones y trabajo futuro

## 6.1. Conclusiones

{**TODO TODO TODO:** Conclusiones de la generación de imágenes, si es útil si ahorra tiempo. ¿Encuesta sobre si la aplicación sería útil? Para escribir algo sobre que sí es algo que la gente considere necesario. Y qué opciones prefieren.}

## 6.2. Trabajo futuro

{**TODO TODO TODO:** Añadir más materiales, más objetos, hacer una interfaz propia, accesibilidad, modelos más realistas para el entrenamiento?, Mejor iluminación?}

# Bibliografía


Alfonso, C., Estévez, R. E., Lobo, J. M., Diéguez, B. L., Prieto, F., Santamarta, J. y Álvaro Gaerter. Emergencia climática en España. 2016.

Alsing, O. *Mobile Object Detection using TensorFlow Lite and Transfer Learning*. Proyecto Fin de Carrera, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

Fonfría, R., Sans, R. y de Pablo Ribas, J. *Ingeniería ambiental: contaminación y tratamientos*. Colección productiva. Marcombo, 1989. ISBN 9788426707420.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. y Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. y Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. 2016.

Karim, R. *TensorFlow: Powerful Predictive Analytics with TensorFlow*. Packt Publishing, Limited, 2018.

Sánchez, M. y Castro, J. *Gestión y Minimización de Residuos*. Fundación Confemetal, 2007. ISBN 9788496743342.

Wang, S.-C. *Artificial Neural Network*, páginas 81–100. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0377-4.