industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Diseño de un sistema inteligente móvil para facilitar procesos fenotípicos de Agricultura de Precisión.

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **María Inmaculada Ros Sánchez**

Director: Nieves Pavón Pulido

Codirector: Juan Antonio López Riquelme

Cartagena, 29 de junio de 2020

« Je me demande si les étoiles sont éclairées afin que chacun puisse un jour retrouver la sienne. »

Aux quatre petits princes.

Contenido

I

Capítulo 1

Introducción, motivación y objetivos.

En este primer capítulo, se exponen tanto la introducción como la motivación y objetivos tenidos en cuenta para la realización de este trabajo.

Capítulo 1. Introducción, motivación y objetivos.

Existen evidencias de que se vive actualmente un cambio en el sector agrícola y, por consiguiente, su modernización. Todos los sectores llevan años en continua mejora e industrialización, digitalizando así sus métodos y herramientas. Sectores como el de la automoción, la industria alimentaria, o la medicina, hacen uso de tecnologías de la información y comunicación (TIC) con el principal objetivo de mejorar sus resultados, ahorrar costes, o para incrementar la calidad de sus productos. Sin embargo, el sector agrícola es el que más se resiste a estos cambios y no ha sido hasta hace unos años, cuando las nuevas tecnologías, o agricultura de precisión, han comenzado a instalarse en los campos y cultivos de los agricultores y empresas agrícolas.

Este industrialismo global adaptado a las nuevas tecnologías está cada día más presente, y no resulta extraño ya el uso de maquinaria o herramientas cada vez más modernas y digitalizadas en todo tipo de sectores. Actualmente se observan enormes cambios tecnológicos, como es el desarrollo de la electrónica y de nuevas tecnologías de la información y comunicación.

Pero, ¿qué es exactamente Agricultura de Precisión (AP)? Como descripción general, diríamos que es aquella que hace uso de las Tecnologías de la Información y de las Comunicaciones (TICs), que gestiona a través de ellas los cultivos y terrenos agrícolas para así obtener unos datos agrónomos con el fin de realizar un posterior análisis de éstos. Estos datos nos permitirán conocer más a fondo la situación exacta del terreno, ayudando de este modo al agricultor a optimizar y mejorar su cosecha, ya sea de pequeño o gran tamaño. La agricultura de precisión permite así, a través de dichas tecnologías, la correcta dosificación de insumos (fertilizantes, agua…) sacando un gran rendimiento de la cosecha y a su vez, contribuyendo a una sostenibilidad del terreno mucho más adecuada, según las necesidades del mismo.

Referente a lo anteriormente dicho, la AP nos permite conocer la cantidad exacta de fertilizantes o agua, entre otros, que nuestro terreno necesita. Incluso es posible conocer qué sección de la cosecha se encuentra en situación de riesgo para así poder tratarla de manera individualizada, proporcionando dosis de insumos justas y concretas.

Esto favorece a otro punto muy importante hoy en día, como es la agricultura sostenible, favoreciendo así la reducción de insumos contaminantes para el suelo y el medio

ambiente. Lo que obtenemos con este tipo de agricultura tan precisa y exacta, es un ahorro notable en los costes y una mejora en cuanto a equilibrio medioambiental.

A lo largo de este primer capítulo, se resume brevemente el contenido de este proyecto, así como la motivación y objetivos tenidos en cuenta a la hora de elaborarlo.

1.1. Resumen.

Este proyecto consiste en la creación de una aplicación móvil Android, la cual será integrada y utilizada junto a un sistema mayor y más complejo en el contexto de AP. Enfocada actualmente al análisis de producto frutal en limoneros, esta aplicación se diseña con el fin de ofrecer al agricultor una herramienta de bajo coste para complementar a un sistema de mayor complejidad a la hora de analizar la producción, así como también acercaría el concepto de agricultura de precisión a cualquier usuario. Su uso sería complementario a otras tecnologías de AP más potentes.

De manera adicional a un sistema mayor de AP, será posible introducir el uso de la aplicación móvil para un completo análisis de la producción agrícola. Se ha desarrollado previamente un sistema capaz de determinar la cantidad de limones existentes en una imagen mediante técnicas de *Machine Learning* y *Deep Learning*. Gracias a las técnicas de aprendizaje profundo se consigue un entrenamiento con una red neuronal, la cual determinará tras analizar los píxeles de la imagen, en qué zonas hay limones y en cuáles no. Esto nos dará como resultado una estimación visual de la cantidad de limones que tendrá nuestra cosecha.

Por tanto, el principal objetivo será adaptar el uso de este sistema de aprendizaje automático a una aplicación móvil, la cual podrá ser utilizada a pie de campo en cualquier momento por el agricultor.

Por otra parte, haciendo referencia a la parte de desconocimiento por parte de un amplio sector agrónomo, cabe destacar que no es fácil para agricultores con cosechas de pequeño tamaño el uso de los servicios actuales de AP, ya sea debido a su precio o al desconocimiento de esta área. Con esta herramienta, se cubre la parte de rechazo por elevado coste y se intenta, además, un acercamiento a esta área, ya sea para darla a conocer al usuario o para introducirle en el mundo de la agricultura de precisión y el uso de las tecnologías de la información y comunicación, así como del *IoT*, o Internet de las Cosas (*Internet of Things*).

A lo largo de este informe se tratarán de manera detallada diferentes aspectos referentes a la AP, así como de la aplicación desarrollada.

1.2. Introducción.

En una gran cosecha, conocer los problemas y defectos localizados de manera exacta puede suponer un problema a la hora de una buena identificación. Esto conlleva a una difícil corrección a la hora de emplear el tratamiento adecuado en la medida justa, provocando en la gran mayoría de ocasiones, un uso desequilibrado de insumos.

El control de este tipo de problemas, así como el control de la producción estimada o el análisis detallado de una gran o pequeña cosecha, es posible gracias a la agricultura de precisión y sus variadas técnicas.

La AP no sólo nos permite controlar de manera regulada la cantidad correcta de insumos que se le proporcionan a nuestra cosecha, sino también, la posibilidad de aumentar la producción gracias a la correcta aplicación de tratamientos, agua, etc. Por otro lado, y dentro del control de insumos, también tenemos el control de malas hierbas mediante agroquímicos.

El uso desmesurado o descontrolado de éstos puede provocar la resistencia de estas malas hierbas a este tipo de productos, lo cual ocasiona el uso de productos químicos más caros, más potentes y más perjudiciales para el medio ambiente. Dentro de estos tratamientos, uno de los más comunes es el de pulverizar de manera prácticamente homogénea el terreno, con el fin de eliminar las máximas malas hierbas posibles, sin importar el alto coste extra que supone el aplicar un tratamiento tan desigual o desproporcionado en el cultivo. Las elevadas dosis de agroquímicos y la resistencia que presentan algunas hierbas a éstos, produce un problema global a la hora de tratar los cultivos.

Se requiere de un conocimiento muy exacto y concreto para una correcta aplicación de las dosis, por lo que el uso desproporcionado o desigual de herbicidas no solo genera un gasto extra (pérdida económica), sino que también hablamos de un daño medioambiental (contaminación de suelo y, por consiguiente, y debido a las filtraciones, la contaminación de las aguas subterráneas).

Es por todo ello, entre otras muchas razones, el motivo por el cual surge la necesidad de un cambio en la agricultura. Es aquí donde entra la agricultura de precisión.

Gracias a las nuevas y mejoradas tecnologías, podemos tener al alcance de la mano estudios y análisis detallados de la cosecha, facilitando así la correcta aplicación de insumos. Por consiguiente, la puesta en práctica de la agricultura de precisión no solo mejora las cosechas y reduce los costes, sino que supone un gran avance en cuanto a la contaminación ambiental, intentando reducir hasta lo mínimo el uso de herbicidas.

Con el enfoque dado anteriormente, las ventajas de la AP frente a la agricultura tradicional son realmente claras. Hablamos no solo de una mejora de la producción, sino también de la posibilidad de usar los insumos de forma cada vez más y más controlada, aplicando dosis mucho más reducidas y controladas.

Esto permite la aplicación de insumos solo en las áreas en las que se requiere y donde esta intervención tendrá un futuro beneficio en el área económica. También como se comentaba anteriormente, el suministro reducido de insumos proporcionará una ventaja medioambiental notable, siendo éste un tema de actualidad y que presenta gran interés de manera global.

La AP, transformada a una metodología de trabajo, permite a los productores ser menos dependientes de los insumos externos a la explotación.

La puesta en marcha de un sistema de AP supone el uso de tecnologías y sistemas los cuales podemos agrupar en cuatro grupos distintos:

1) Sistemas de Posicionamiento Global (GPS y DGPS), nos permitirán conocer la posición y localización exacta de los equipos agrícolas.
2) Sensores, materiales diseñados para detectar información y proporcionar numerosos datos sobre la parcela analizada.
3) Sistemas de Información Geográfica (GIS), diseñados para capturar, almacenar, manipular y analizar las informaciones recibidas geográficamente.
4) Tecnologías con posibilidad de manejar de forma automática los equipos agrícolas.

Este último grupo haría referencia a los sistemas de visión artificial aplicados a la AP, usados para llevar adelante la navegación automatizada de los vehículos. Podemos clasificar estos sistemas en directos e indirectos.

- Sistemas directos: nos permiten trabajar en tiempo real. No se requiere de ningún operario para supervisión siendo la actuación inmediata tras la toma de imágenes.
- Sistemas indirectos: el procesamiento es posterior a la captura de los datos, ya sea mediante imágenes o videos.

Estos últimos presentan un gran crecimiento con el sistema de drones dentro de la agricultura. Pueden ser controlados de manera remota por un usuario y también de forma automática programada mediante ordenadores.

Como se comentaba anteriormente, hoy en día está cada vez más presente el uso de tecnologías de visión artificial aplicadas a la agricultura con el fin de identificar y localizar problemas de manera más específica y concreta. Esto también facilita un ahorro en costes y una mejora en la productividad. La agricultura de precisión es una realidad cada día más presente en los campos y cultivos, aunque sigue presentando cierto rechazo a la hora de dar el primer paso hacia el cambio. Existe aún un gran desconocimiento acerca de esta área, de sus ventajas y su aplicación, siendo así uno de los principales puntos a superar a la hora de integrar las nuevas tecnologías agrícolas en el día a día de muchos agricultores.

Otro de los inconvenientes referente al rechazo de la AP, es el elevado coste que puede presentar en un primer momento el contrato de estos servicios.

Es por todo ello, que con este proyecto se pretende, por una parte, llegar de manera más fácil al consumidor y usuario agrícola facilitándole una herramienta como es una aplicación móvil Android, para que desde su dispositivo móvil pueda experimentar un contacto asequible con las tecnologías de la AP.

Por otra parte, no solo se pretende dar a conocer la AP y acercarla al agricultor de manera fácil, sino que principalmente se propone como uso complementario a otros sistemas y tecnologías más complejos. Complementando esta herramienta con otro tipo de tecnologías de AP, se podrían obtener mejores resultados, así como conocer mejor la producción futura. Es por ello que, gracias al desarrollo de la aplicación móvil, donde se integrarán las técnicas desarrolladas previamente de entrenamiento de red neuronal, se podrá realizar una identificación visual de la producción aproximada de limones existentes en la imagen tomada.

Esta imagen será la analizada por el modelo, previamente entrenado para un correcto procesamiento, y como resultado obtendremos esa misma imagen con los píxeles que correspondan a un limón en color rojo.

La imagen final se compondrá en conjunto de todos aquellos píxeles originales y modificados en color rojo, formando la imagen final con el resultado de la producción aproximada de limones en la imagen tomada por nuestro dispositivo móvil.

1.3. Motivación y objetivos.

1.3.1. Motivación.

Actualmente existen millones y millones de dispositivos, todos dependientes del IoT: toman información y no hacen nada con ella. Esta información es enviada a la nube, tratada en centros de datos donde se procesa, y posteriormente, se obtienen conclusiones o se activan ciertos eventos. Este comportamiento pasivo es aquello que busca cambiar lo que llaman *Edge Computing* (*Ilustración 1*).

Cabe destacar la cifra de dispositivos *IoT* en 2020 que *Ericssonn* y *Cisco* estimaron en 2010 y 2011 respectivamente. [1].

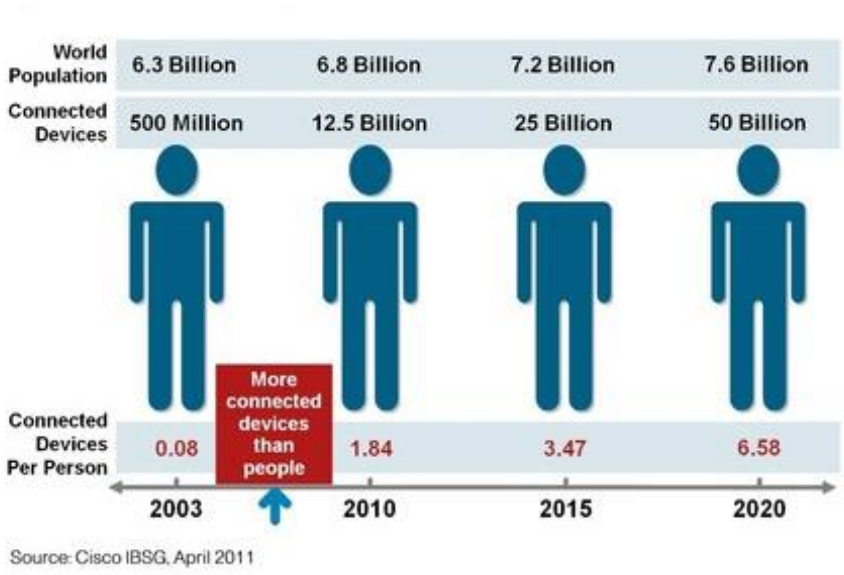


Ilustración 1. Dispositivos IoT.

La idea final de esta tendencia cada día más popular se podría resumir de la siguiente manera: aprovechar los recursos en cada caso para optimizar la capacidad de proceso y la transferencia de datos. [1]

Este Trabajo Fin de Grado desarrollado en el marco de la AP, tiene su origen en el proyecto de investigación "Diseño, desarrollo y validación de un sistema inteligente de toma de decisiones en el manejo del agua de riego en agricultura (AgroSmartGIS)" y surge con el objetivo de experimentar sobre las técnicas de *Deep Learning*, la integración de un modelo entrenado en una app móvil desarrollada desde cero, y la implementación de esta herramienta a pie de campo por cualquier usuario. A partir de una red neuronal de *Deep Learning* entrenada para la detección de limones en árboles limoneros, se adapta el modelo pre-entrenado de esta

red neuronal a una aplicación diseñada en Android. Basándome así en dicho proyecto y gracias al desarrollo del modelo de predicción, se obtiene este mismo modelo en versión *tflite* para adaptarlo posteriormente a una aplicación móvil Android.

Con este proyecto se brinda la oportunidad de un sencillo uso de una herramienta basada en la Inteligencia Artificial para aquellos usuarios que deseen usarla en plantaciones agrícolas, en nuestro caso, de árboles limoneros. A su vez, será una herramienta que podrá complementar a otras técnicas más potentes, obteniendo en conjunto unos mejores resultados.

1.3.2. Objetivos.

La AP sigue siendo una tecnología algo desconocida por muchos agricultores. Para otros, es algo que ya forma parte de su día a día como una parte del proceso de producción. Para ambos casos se fija el objetivo principal de este proyecto: proporcionar una aplicación móvil accesible a cualquier usuario que desee tanto complementar otros sistemas ya contratados de AP, o, por otro lado, dar un paso hacia el descubrimiento de esta nueva área. Puesto que estas nuevas tecnologías no están al alcance de todo el mundo, y no todos los usuarios agrícolas se atreven a lanzarse en grandes proyectos de agricultura de precisión para sus terrenos, se pretende dar también a conocer las herramientas de Inteligencia Artificial mostradas con orientación sencilla de uso. También, como se comentaba antes, se enfoca al uso de esta aplicación como complemento de otras tecnologías para así obtener mejores resultados en la producción final.

La aplicación hará ver al usuario el gran avance que supondría adaptar la tecnología a sus cosechas, proporcionando datos aproximados visuales sobre la producción existente.

El desarrollo de la aplicación está enfocada a un posterior uso simple, con interfaz gráfica sencilla para captar la atención del usuario.

El objetivo principal de este proyecto en aspectos técnicos, será, una vez transformado el modelo de *Tensor Flow* entrenado con un código *Python*, a modelo *Tensor Flow Lite* (para uso en dispositivos móviles), su posterior adaptación a la aplicación desarrollada en Android Studio, integrando correctamente el modelo.

Dicho modelo tiene como objetivo la toma de una imagen y su posterior procesamiento para obtener la imagen final con el resultado de la evaluación con el modelo. En primer lugar, se tomará la imagen con el teléfono móvil. Esta pasará a ser procesada

previamente, antes de ser enviada al modelo. Su procesamiento consistirá en obtener de dicha imagen, de tamaño 4608×2592, un total de 641520 imágenes de 32x32. La imagen será subdividida en sub-imágenes de tamaño 32x32x3 (RGB), analizando, dentro de cada una de estas sub-imágenes, cada uno de sus píxeles. Con ello, obtendremos una valoración de 1 o 0, que será guardada en un *array* con el resto de valores. Así, el modelo analizará dicho *array* con valores 1 y 0, siendo 1 la detección de un limón y 0 lo contrario. Como resultado del análisis, la imagen aparecerá con los limones detectados pintados en color rojo.

El presente trabajo tiene como finalidad crear, analizar y exponer una aplicación de uso fácil para el agricultor, la cual le permitiría analizar en breves instantes y a pie de campo una ligera estimación de su producción frutal. En este caso, para el proyecto se hace uso de un modelo previamente entrenado y facilitado para su integración en la aplicación móvil, el cual detecta los limones existentes en el árbol de la imagen tomada. Esto conseguiría un gran acercamiento de las técnicas de AP a aquellos agricultores independientes, o pequeñas empresas, los cuales no tengan la disponibilidad de contratar servicios altamente profesionales. Por otra parte, y como objetivo principal, se fijará como complemento a un sistema mayor y más complejo de AP. Con la aplicación se trabajará de manera conjunta con el resto del sistema, ofreciendo así un análisis más completo y detallad

1.4. Descripción resumida de capítulos.

En el Capítulo 1, en el cual nos encontramos, se explica de manera detallada la introducción a este proyecto. Pasando por los distintos puntos de este capítulo, encontramos el resumen, introducción, motivación y objetivos del proyecto.

En el Capítulo 2, nos centraremos en el "estado del arte". En este capítulo encontramos una descripción de lo que actualmente significa la AP y de qué manera está presente hoy en día. A su vez, se explican las técnicas existentes de la Agricultura de Precisión (2.1). Por otra parte, se explican conceptos como el *Deep Learning* o *Machine Learning* y de qué forma están totalmente relacionados con la Agricultura de Precisión (2.2). También se exponen las diferencias entre los sistemas de *Deep Learning*, *Cloud Computing* o *Edge Computing* (2.3), así como las carencias y barreras a las que se enfrentan hoy en día los sistemas actuales, los propósitos o desafíos para la agricultura de precisión en España y, en general, una vista sobre las oportunidades y facilidades que los usuarios agrícolas tienen a la hora de implantar la agricultura de precisión en sus cosechas (2.4).

El Capítulo 3 (diseño del sistema), basado principalmente en las técnicas usadas para el diseño del proyecto, pasando por una breve introducción (3.1), y detalles sobre la arquitectura tanto de *Hardware* (3.2) como de *Software* (3.3). A su vez, encontramos en el apartado 3.4 (tecnología usada) una descripción detallada acerca de la aplicación Android desarrollada. Este apartado, está dividido en tres sub-apartados destinados a explicar lo más detallado posible el uso de Android Studio (3.4.1), TensorFlow y TensorFlow Lite (3.4.2). Siguiendo dentro del capítulo 3, encontraremos también los métodos usados durante el desarrollo de la aplicación, como son el modelo de TensorFlow usado (3.5.1) y cómo convertir correctamente el modelo TensorFlow a TensorFlow Lite (3.5.2). Por último, se presenta un esquema final explicativo del diseño, estructura y funcionamiento de la aplicación.

El Capítulo 4, uno de los más importantes, será donde se explique el procesamiento de la imagen. Pasando por una breve introducción (4.1), la selección de la imagen a analizar (3.2), y el proceso del análisis (pre-procesado, procesado y reconstrucción final de la imagen, 4.3, 4.4, y 4.5 respectivamente).

En el Capítulo 5, correspondiente a los resultados obtenidos, se muestran imágenes de las pruebas realizadas a pie de campo con la aplicación móvil.

Por último, el capítulo 6. En él se comentan las conclusiones finales, así como algunas propuestas de mejora y trabajo futuro con la aplicación.

Capítulo 1.

II

Capítulo 2

Estado del arte.

En este segundo capítulo, se exponen el estado del arte, así como conceptos como el Deep Learning, Machine Learning, Cloud Computing y Edge computing..

Capítulo 2. Estado del arte.

La AP se concibió desde EEUU como un círculo que se retroalimentaba año a año y donde el único objetivo culminaba con la realización de dosis variables de insumos. Quiere decir esto, que se aplicó la idea de usar la tecnología de la información para, de manera adecuada, controlar los suelos y cultivos frente a los cambios naturales. Una herramienta clave en este sistema es el uso de GPS, gracias al cual podemos obtener información detallada de cada parcela, así como también medios que nos permitan la recopilación de grandes cantidades de datos, los cuales usaremos para estimar y analizar el campo analizado.

Frente a la agricultura tradicional, destacamos que gracias a la agricultura moderna se han incorporado herramientas tecnológicas en continua mejora, lo cual lleva a este tipo de agricultura a ser mucho más eficiente haciendo que ahorremos tiempo y dinero. Se logra así una mayor producción, calidad y reducción de insumos innecesarios gracias a un mayor control sobre la producción.

La agricultura forma parte de la historia de España. Desde el asentamiento de la producción industrial a mediados del siglo XX y desde la entrada a la Unión Europea en 1986, se empezó un camino hacia la modernización de la agricultura tradicional.

Por tanto, es gracias a la agricultura moderna que la eficiencia (en cuanto a calidad y cantidad se refiere) se ha visto directamente afectada positivamente, así como una mayor productividad con maquinaria que permite adaptarse a la demanda que proviene de los mercados, y por supuesto, un mayor control sobre la producción. Esto último, hace que las plagas sean controladas de manera mucho más fácil gracias a prácticas que se aplican directamente sobre la zona afectada. También se controlan mejor los sistemas de riego, los insumos, etc.

Las tecnologías relacionadas con la agricultura de precisión presentan actualmente una creciente aceptación por parte de las empresas agrícolas. Dadas las nuevas y mejores técnicas empleadas en este tipo de tecnologías, son muchas las empresas poseedoras de terrenos para producción agrícola las que se atreven a dar el paso hacia una nueva forma de mejorar la producción final. Empresas en España como Agrosap, SGS España, SmartRural, Agrodrone o Hemav, entre otras; ofrecen al cliente la posibilidad de servicios de agricultura de precisión, gestión de mapas de rendimiento, mapas de estado de cultivo, agricultura moderna planta a planta, digitalización de campos, optimización de costes, aumento de la producción, etc.

2.1. Agricultura de Precisión: técnicas y sistemas existentes.

Como ya se ha comentado antes, la AP se basa en el uso de técnicas basadas en las TICs para así determinar de manera exacta la cantidad de insumos, por ejemplo, en el momento adecuado, en la cantidad correcta y en el lugar exacto.

La AP presenta técnicas cada día más competentes dentro de un mundo donde la tecnología crece por momentos. Involucra así, el uso de sistemas de posicionamiento global (GPS) como base principal de su tecnología, así como otros medios electrónicos mediante los cuales, se obtienen los datos suficientes para su posterior tratamiento e informe final.

La AP tiene numerosos beneficios, entre los cuales destacan los siguientes:

- Incrementa el rendimiento de los insumos aplicados.
- Reduce la cantidad de insumos a aplicar, haciendo más controlado su uso.
- Se aumenta la productividad, así como la calidad de los cultivos.

La agricultura de precisión sigue habitualmente un mismo patrón en cuanto a etapas se refiere, un mismo ciclo que se resume en los siguientes 4 pasos (*Ilustración 2*): adquisición de datos, extracción de la información necesario para un correcto análisis, toma de decisiones y aplicación o monitorización del rendimiento. [2]

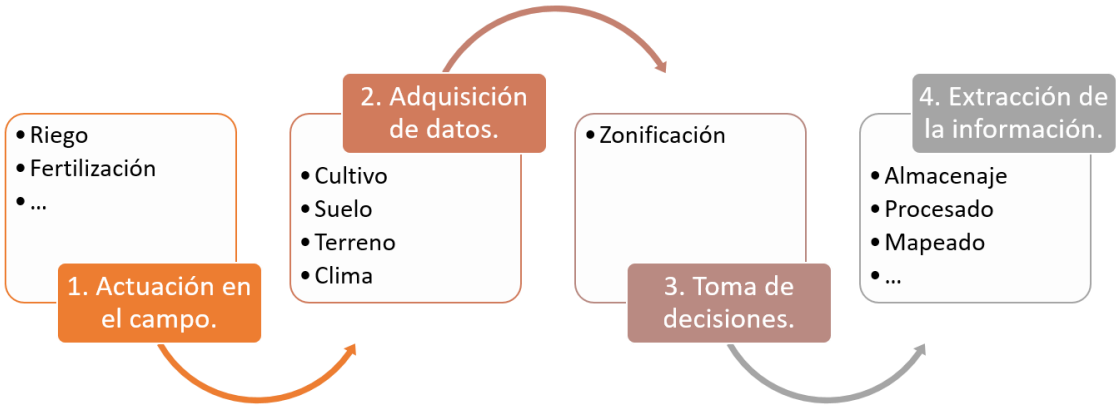


Ilustración 2. Ciclo Agricultura de Precisión

Adquisición de datos.

La adquisición de datos supone la obtención de variables como topografía o propiedades del suelo, detección de plagas, etc. Se hace uso de sistemas GPS y sensores. La cantidad de datos a recoger es tan amplia como nuestra capacidad tecnológica y como la cantidad y variedad de sensores que tengamos disponibles para la recogida de datos. Para ello, se usan sensores, muestreos con georreferencia con el uso de Sistemas Satelitales de Navegación Global (SSNG), etc. [2]

Extracción de la información.

Una vez que hemos obtenido los datos en el paso anterior, el siguiente paso es el procesamiento de éstos, así como su correcta interpretación. De esta forma, se determina si existe una dependencia directa entre las variables para poder establecer modelos o patrones. El objetivo principal consiste en obtener la máxima cantidad de datos de utilidad para el agricultor, para así permitir posteriormente un uso de ellos que permita corregir la eficiencia de su cultivo y mejorarla. Para ello, se usan herramientas estadísticas y matemáticas como es el caso de la clasificación de datos, las gráficas, el mapeado, *AgGis*, geoestadística… [2] [3]

Toma de decisiones, preventivas y de gestión.

Una vez completada la extracción de la información captada, se da paso a la tercera etapa, donde mediante los resultados obtenidos se proporcionan posibles soluciones y mejoras. Es el momento donde se decidirán las operaciones de manejo agronómico que habrá que llevar a cabo. Esto determinará varias cosas, entre ellas, si se continúa con un análisis y manejo uniforme del campo o si éste presenta variabilidad, con lo que se recomienda hacer un manejo diferenciado. Con ello, en esta etapa conseguiremos un ajuste de la dosis de aplicación, control inteligente de malas hierbas, precisión centimétrica en el guiado de la maquinaria agrícola, etc. [2] [3]

Monitorización del rendimiento.

Al finalizar las etapas previamente explicadas, pasamos a la actuación a pie de campo para así poder aplicar los recursos necesarios y realizar las operaciones que se consideren pertinentes. En esta etapa entran en juego las Tecnologías de Actuación Variable (VRT, *Variable Rate Technologies*), las cuales permiten a los equipos autorregularse para modificar así las dosis a aplicar en concordancia con el informe elaborado en la etapa de toma de decisiones. Además, la disponibilidad de almacenar la información entre campañas nos permite conocer nuestra evolución a lo largo de los años.

La agricultura de precisión hace actualmente un uso amplio de tecnologías, como son los Sistemas de Posicionamiento Global, o GPS, sensores, mapeo, etc. [2] [3]

Sistemas de posicionamiento global (GPS y DGPS).

Los Sistemas de Posicionamiento Global *(Ilustración 3)*, (GPS y DGPS), nos permiten la localización de cualquier punto que se quiera analizar, ya sea un punto del terreno, una persona, un objeto… Hoy en día, se dispone de una nueva tecnología de geolocalización, conocida como DGPS, o GPS diferencial. [4]

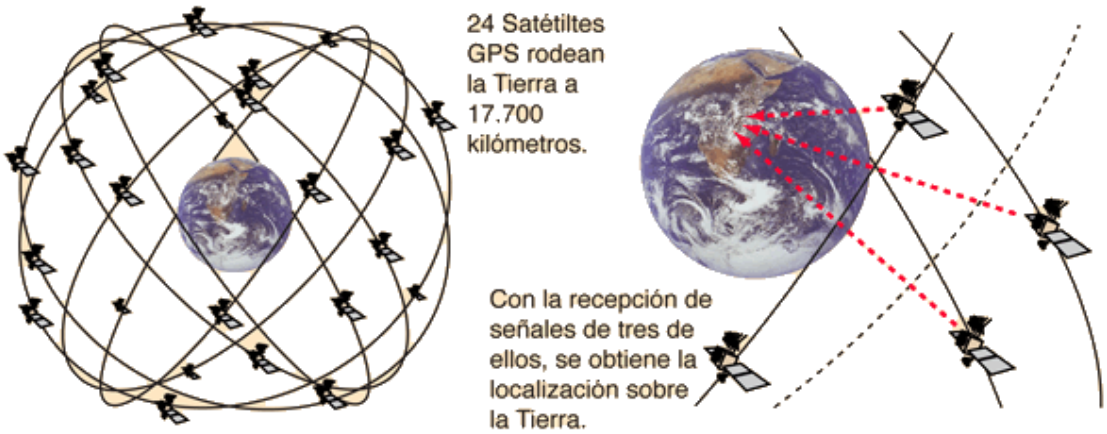


Ilustración 3. Sistemas de Posicionamiento Global

Gracias a esta tecnología, los datos que nosotros recibimos son datos que previamente han sido proporcionados a los receptores de GPS con correcciones de los datos

que reciben por GPS, por lo que la exactitud de medición de la posición calculada será mucho mayor y acertada. Es por ello, que se puede medir fiablemente hasta mediciones de entre 5 y 20 metros en cualquier lugar que se desee.

Dicho de otro modo, el DGPS cancela gran parte de aquellos errores naturales o producidos por el ser humano que interfieren en las mediciones realizadas por GPS. Esta tecnología es principalmente combinada con otras, como son los sensores o las instalaciones de éstos.

Sensores.

Los sensores o las instalaciones de sensores nos permitirán evaluar en cada momento la posición de la máquina, su temperatura, velocidad, etc. A su vez, es esto lo que también nos permitirá determinar posteriormente la cantidad de grano cosechado o la fertilidad del suelo, entre otros.

Mapeo.

Como se comentaba al comienzo, se hace uso también del mapeo. Con el mapeo (*Ilustración 4)* conseguimos una representación de datos registrados durante una cosecha o cultivo. El objetivo principal de realizar un mapeo reside en la facilidad de identificar y entender posteriormente, las necesidades actuales y futuras del cliente. Con ello, conseguiremos ajustarnos a las necesidades posteriores que pueda tener dicho cliente y de este modo, asesorarle en cuanto a la cantidad de recursos requeridos y las actividades a realizar sobre su terreno.

Estos mapas de rendimiento reportan gran cantidad de información acerca de la producción y nos ofrecen parámetros para corregir aquello que produce un bajo o un alto rendimiento y eficiencia dependiendo de la zona.

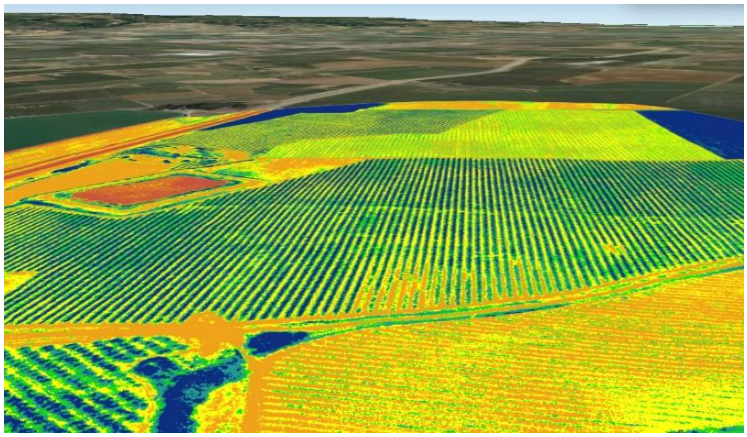


Ilustración 4. Mapeo de terrenos

Por otro lado, están los monitores de rendimiento. Éstos recogen aquella información necesaria procedente de distintos sensores y gracias a un software se calcula el rendimiento de un cultivo en el tiempo y en el espacio, basándose en la información de localización de cada sección de terreno proporcionada por los sistemas GPS o DGPS. El resultado es representado en un mapa gráfico.

Sistema de Información Geográfica (GIS).

A su vez, disponemos también de tecnología GIS (Sistema de Información Geográfica). Con similar funcionamiento al mapeado, aunque diferenciándose de este último en que los datos obtenidos son representados como información y no como mapas convencionales o dibujos. Los datos de GIS contienen la información de un mapa convencional, pero con la diferencia principal de su representación, la cual nos permite en este caso la capacidad de ser más flexibles a la hora de representarlos.

Un GIS es un software cuyo objetivo es almacenar, analizar y representar datos de carácter cartográfico. Nos permite analizar patrones, simulaciones y modelos. El GIS nos proporciona datos e información, al contrario que el mapeado, como se comenta anteriormente. Los datos son almacenados de manera estructurada. Nos ofrece datos por capas: capas para asentamientos humanos, vegetaciones, ríos… (*Ilustración 5*). Gracias a ello, nos ayudará a relacionar una gran cantidad de datos. [4]

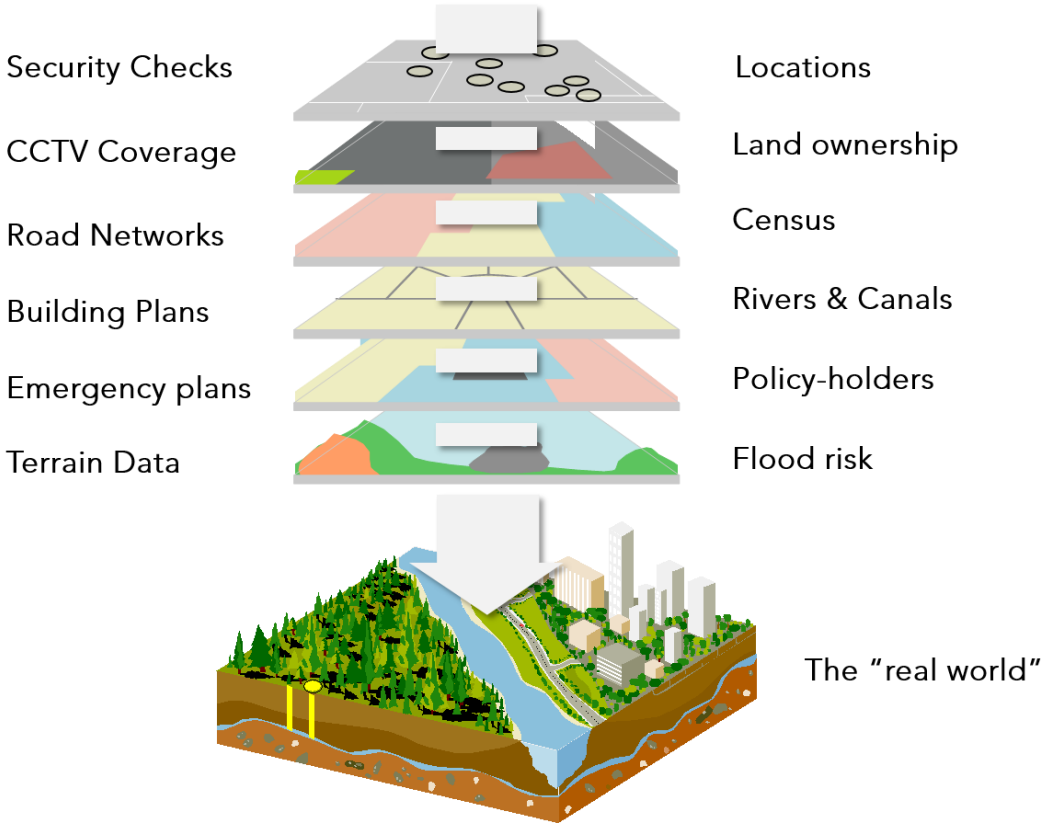


Ilustración 5. Sistemas de Información Geográfica

Con todas estas tecnologías comentadas hasta ahora, y combinándolas entre sí, se obtienen análisis de gran calidad y exactitud acerca de los terrenos analizados. Mediante la información obtenida a través de la primera etapa, con el muestreo de suelos y posteriormente recopilando los datos mediante mapeado, combinando todo con el sistema GIS, que contendría los datos referentes a la explotación y a rendimientos pertenecientes a cosechas o análisis anteriores, generamos un mapa de predicciones con las futuras acciones. [3] [5]

2.2. Deep Learning y su aplicación en el contexto de la Agricultura de Precisión.

Deep Learning.

El *Deep Learning* es una técnica de aprendizaje automático que enseña a los ordenadores a hacer lo que resulta natural para las personas: aprender mediante ejemplos. Con el *Deep Learning*, un modelo informático aprende a realizar tareas de clasificación

directamente a partir de imágenes, texto o sonido. Los modelos de *Deep Learning* pueden obtener una precisión de vanguardia que, en ocasiones supera el rendimiento humano. Los modelos se entrenan mediante un amplio conjunto de datos etiquetados y arquitecturas de redes neuronales que contienen muchas capas. [6]

Antes de profundizar más en el *Deep Learning*, cabe destacar la jerarquía que sigue el aprendizaje profundo. Destacamos así tanto el *Machine Learning* como el *Deep Learning* dentro de este campo. Sin entrar en grandes detalles complejos de la Inteligencia Artificial, he de mencionar dos grupos de gran importancia para saber dónde colocar tanto al *Machine Learning* como al *Deep Learning*. Estos dos grupos son los siguientes:

- **Inteligencia Artificial robusta o *Strong AI*:** es una inteligencia real en el que las máquinas tienen similar capacidad cognitiva que los humanos.
- **Inteligencia Artificial aplicada *Weak AI (Narrow AI o Applied AI)*:** aquí es donde situamos el uso de algoritmos y aprendizajes guiados con *Machine Learning* y *Deep Learning*.

La Inteligencia Artificial, engloba en diferentes campos, como son el *Machine Learning* (aprendizaje automático de máquinas) y el *Deep Learning* (aprendizaje profundo). La función principal y más fácil de comprender, se trata del objetivo de convertir las máquinas en dispositivos inteligentes capaces de interactuar mejor con nosotros, dotándoles de capacidades similares a las de los seres humanos en una medida mucho más precisa y avanzada a lo que nosotros somos capaces de captar.

Para describir brevemente y sin entrar en detalles el *Machine Learning*, o autoaprendizaje, cabe decir que esta tecnología consiste en proporcionar al ordenador una cantidad suficiente de datos como para que consiga un aprendizaje mediante los algoritmos facilitados. Estos algoritmos son la manera de "educar" al ordenador con el fin de proporcionarle autonomía *(Ilustración 6)*.

Como ejemplo básico para comprender la funcionalidad del *Machine Learning*, haré referencia al ejemplo común de identificación de un animal, en este caso, de un gato. El ordenador sería programado para así identificar a un animal en concreto, en este caso, un gato. Se escribiría un código el cual indicaría al programar a "elegir" cuando éste identificase a un gato en la imagen proporcionada para su análisis. Esto podría funcionar siempre y

cuando no tuvier que ver una gran cantidad de imágenes donde aparecieran diferentes animales, con una gran cantidad de gatos, y tuviera que determinar cuáles de todas esas imágenes contienen un gato. [7]

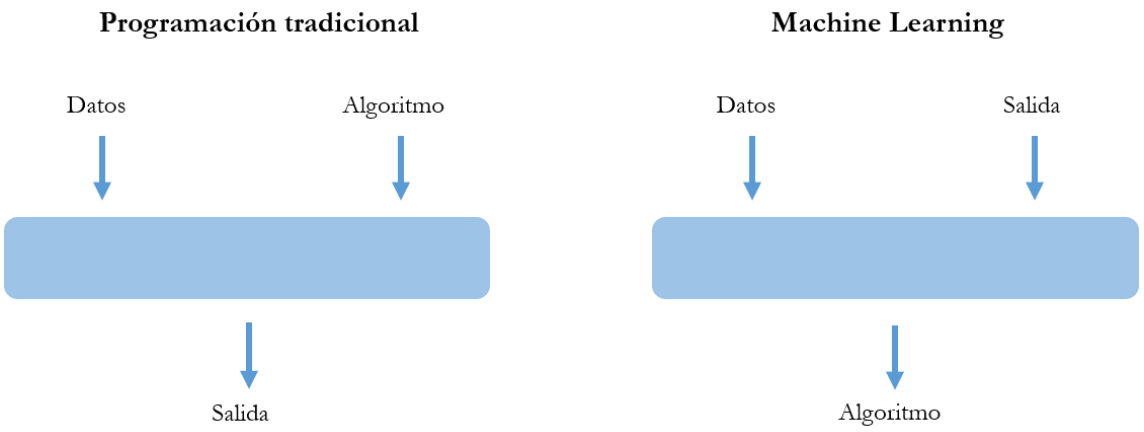


Ilustración 6. Programación tradicional vs Machine Learning

Sin embargo, para corregir el problema anterior, se podría entrenar de diferente manera. Una de esas maneras sería mostrando al programa una gran cantidad de imágenes con diferentes animales, etiquetadas con el nombre correspondiente del animal. Así, el programa aprendería a identificar a los animales que se correspondan con la etiqueta "gato", facilitándole de este modo un reconocimiento variado de diferentes tipos de sujeto.

Por tanto, el programa deberá aprender diferentes combinaciones de características visuales para poder así identificar a los gatos del resto de animales y objetos que aparezcan en la imagen. Por tanto, el programa asociará finalmente esta combinación de características con la palabra "gato" gracias al proceso de aprendizaje o entrenamiento, conocido como "construcción de un modelo de un gato". [7]

Una vez construido el modelo de identificación de "gato", el programa pasaría a prueba intentando identificar los gatos en imágenes que no ha visto previamente. Se mide posteriormente la cantidad de aciertos que ha obtenido y usa esa información para ajustar el modelo nuevamente para mejorar de cara a la próxima prueba.

Por otro lado, y siguiendo la evolución del comentado *Machine Learning*, nos encontramos con una muy reciente y nueva técnica perteneciente al entorno del *Machine Learning*: el *Deep Learning*.

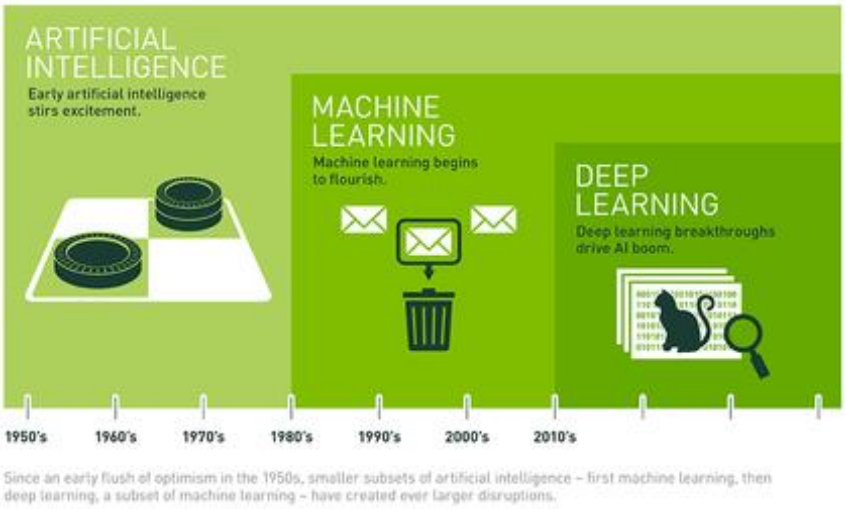


Ilustración 7. Estructura de la Inteligencia Artificial

El *Deep Learning*, es un subconjunto dentro del campo del *Machine Learning* como se puede comprobar en la *Ilustración 7*. En el *Deep Learning*, el principal objetivo reside en proporcionar un modelo (en puesto de dar una gran cantidad de reglas al ordenador), el cual evalúa ejemplos. El *Deep Learning* hace uso de redes neuronales, las cuales se combinan de manera progresiva formando niveles de distinta dificultad para dar lugar a una red neuronal final capaz de realizar una función determinada. El nivel inicial de esta pirámide jerárquica se encargará de aprender algo sencillo, para posteriormente enviarlo al siguiente nivel. En este nuevo nivel, el cual adquiere mayor dificultad que el anterior, se encargará de tomar la información previa del nivel anterior y combinarla para componen una nueva información más compleja. Esta nueva información, acumulada ya de dos niveles, pasa a un tercer nivel, donde otra vez más, se vuelve a combinar con información más compleja para dar paso al siguiente nivel.

Si volvemos a usar el ejemplo anterior del "gato", en esta nueva técnica de *Deep Learning* podríamos, primero, usar las zonas con mayor y menor claridad de la imagen. Posteriormente, tras pasar esta información del primer nivel al segundo, se analizarían los bordes proporcionando formas sencillas. En el tercer nivel, las formas simples se combinarían con objetos más complejos. En el cuarto nivel, la información previa se podría combinar para poder ya identificar la forma de una pata, una oreja, etc. El proceso de identificación continuaría en siguientes niveles hasta alcanzar un último nivel donde el modelo ya estaría entrenado para poder identificar gatos.

Pero, ¿para qué sirve hoy en día el *Deep Learning*? Algunas de las muchas aplicaciones que tiene hoy en día el *Deep Learning* podrían ser las siguientes: identificar marcas o logotipos

de empresas en fotos publicadas en las redes sociales, orientación de anuncios y predicción de las preferencias del usuario, identificación de posibles clientes potenciales, localización de caras o emociones faciales (aplicado mucho a las cámaras de los dispositivos móviles), o, la que se analiza en este proyecto, la agricultura de precisión.

Actualmente, el área de la agricultura está cambiando, más y más rápido, adaptándose así a las nuevas tecnologías como es el *Deep Learning*. El uso de drones y tractores autónomos está cada día más presente entre los agricultores y sus cosechas. A esto podemos llamarle agricultura 4.0. Gracias a este tipo de técnicas podemos obtener, mediante la captación de imágenes satelitales, grandes cantidades de datos muy concretos, a partir de los cuales se elaborarán planes de actividades sobre el terreno.

La agricultura inteligente (o de precisión) compensa con tecnología la drástica reducción de tierra cultivable y el aumento de población, que en 2039 alcanzará los 8500 millones de personas. [8].

Como se comentaba anteriormente, los sensores juegan un gran papel dentro de los elementos de esta revolución tecnológica. A ello, sumamos el uso de drones y vehículos agrónomos autónomos. Según datos del Banco Mundial, el mundo ha perdido casi la mitad de su tierra cultivable por persona. En el caso de España, hemos pasado de tener 0.53 hectáreas arables por persona en 1961 a 0,25 en 2014. Mientras que, por otro lado, la población sigue creciendo. [9].

De esta forma, la agricultura de precisión nos facilita un trato de la tierra que hace uso de menos agua, menos combustible para los vehículos y menos herbicidas, grandes contaminantes en el mundo de la agricultura.

2.3. Deep Learning, Cloud Computing y dispositivos móviles inteligentes. Edge Computing.

Lo primero de todo será detallar el significado de cada término, con el fin de una correcta comprensión de cada concepto a lo largo de este apartado.

Deep Learning

Deep Learning, o *Aprendizaje Profundo*, es un método de aprendizaje dentro del campo de la Inteligencia Artificial, y a su vez, dentro del subcampo del *Machine Learning*. Es considerado como la aproximación a la percepción humana, ya que tras un entrenamiento

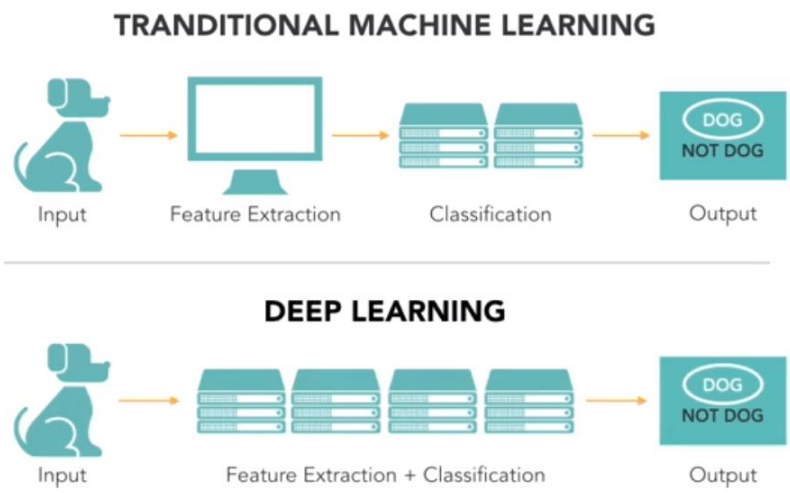


Ilustración 8. Aprendizaje automático tradicional vs Deep Learning

automático obtienen las conclusiones pertinentes acerca de la semántica embebida de los datos. Esta tecnología está en pleno auge gracias al fuerte uso del Big Data y el IoT (*Internet of Things*).

Los algoritmos tradicionales de aprendizaje automático son lineales, mientras que los de aprendizaje profundo forman parte de una estructura jerarquizada que va aumentando su complejidad. Cada algoritmo en la jerarquía estructurada comprende una transformación no lineal en su entrada y usa lo que aprende para crear un modelo estadístico como salida.

En el aprendizaje tradicional (*Ilustración 8*), el proceso se supervisa por un programador. Es por tanto una gran ventaja en el aprendizaje profundo la no dependencia de esta parte del proceso, ya que el programa construye las características por sí mismo sin necesidad de ser supervisado.

Con el objetivo de alcanzar un nivel aceptable de precisión, los programas de aprendizaje profundo requieren acceso a cantidad muy grandes de datos de entrenamiento. Debido a que la programación del aprendizaje profundo es capaz de crear modelos estadísticos complejos directamente a partir de su propia salida iterativa, es capaz de crear modelos predictivos precisos a partir de grandes cantidades de datos no etiquetados y no estructurados. [10]

Esto es importante a medida que el internet de las cosas (*IoT*) continúa haciéndose más penetrante, ya que la mayoría de los datos que los seres humanos y las máquinas crean están desestructurados y no etiquetados.

Cloud Computing.

Para casi todo el mundo es común el uso de *Gmail, Hotmail*, etc. Mediante su uso, también utilizamos de alguna manera el *Cloud Computing*. Pero, ¿en qué consiste? El "cloud" o "la nube" consiste así en el suministro de recursos informático por petición a través de Internet y con un modelo de pago según su uso. El término *Cloud Computing* hace referencia una concepción tecnológica y a un modelo de negocio que unifica ideas tan variadas como el almacenamiento de información, las comunicaciones entre ordenadores, etc. A la hora de definir el concepto de Internet, podríamos destacar que, definida de manera simple, es un conjunto de ordenadores, distribuidos por el mundo y unidos por una tupida malla de comunicaciones, que ofrece espacios de información a todo el que tenga acceso.

Software como servicio (SaaS).

Es el punto más alto de las clasificaciones de componentes de la informática. Aquí se encuentran las aplicaciones finales o los productos terminados. Permite a los usuarios conectarse a aplicaciones basadas en la nube a través de Internet y usarlas. Se puede iniciar sesión y empezar a usar dichas aplicaciones de negocio de manera rápida. Si el sistema falla en algún momento, ningún dato se perdería.

Plataforma como servicio (PaaS).

Este servicio de plataforma pone a disposición del usuario una serie de herramientas, las cuales están disponibles para la realización de desarrollos informáticos. Entorno de desarrollo completo en la nube. Los recursos de compran a un proveedor de servicios en la nube mediante una conexión segura a Internet. Acelera el desarrollo y la comercialización de aplicaciones y reduce la dificulta con *middleware* (software que se sitúa entre un sistema operativo y las aplicaciones que se ejecutan en él) como servicio.

Hay, por tanto, dos ventajas en este servicio: por una parte, el usuario no tiene que adquirir las costosas licencias para desarrollo, y por otra parte, el proveedor se encarga de que estas herramientas estén en la mejor situación de mantenimiento. [11]

Infraestructura como servicio (IaaS).

Es una infraestructura instantánea de computación. Incluye servidores, redes, almacenamiento y espacio en centro de datos. Ofrece así, un espacio de almacenamiento o capacidad de procesamiento en sus servidores.

El usuario tiene acceso, por tanto, a espacio disponible ilimitado. Este servicio se basa en el acceso al uso de hardware radicado en la nube.

Cloud público.

Pertenecen y son administrados por empresas que ofrecen a través de una red pública acceso rápido a recursos informáticos asequibles.

Cloud privado.

Infraestructura que utiliza únicamente una única organización.

Cloud híbrido.

Utiliza una base de *Cloud* privado, combinada con la integración estratégica y el uso de servicios *Cloud* público.

Relacionado también con el *Cloud Computing*, cabe destacar algunos aspectos. Normalmente, los dispositivos conectados se dedicaban al envío de datos a la nube para después ser procesador ahí. Sin embargo, éstos cada vez tienen mayor capacidad de cómputo debido al abaratamiento de la misma y a la mejora de la eficiencia energética que ofrece el hardware más moderno. [12] Surgen así nuevas técnicas y modelos de computación, que sacan mayor rendimiento a los dispositivos extendidos por la red.

Nos centramos en el último en la jerarquía (*Ilustración 9*), el *Edge Layer*, o *Edge Computing*. El *Edge Computing* ocurre en la ubicación física del usuario, de la fuente de datos, o en general, cerca de ellas. Al establecer servicios de computación cerca de esas ubicaciones, los usuarios obtienen servicios más rápidos y confiables. Hoy en día podemos ver el *Edge Computing* en tecnologías como la realidad virtual. Para tener una buena experiencia con estas tecnologías hemos de tener disponibilidad de una velocidad alta y buena potencia informática. Si estos aspectos se ven perjudicados, la experiencia envolvente en estas tecnologías se verá obstaculizada. Por ello, el *Edge Computing* en estas tecnologías permite que se puedan descargar en la nube las partes del proceso que consumen muchos recursos informáticos. Al igual que el resto de dispositivos de *IoT*, se debe procesar una cantidad de datos y una toma de decisiones en tiempo real, o, en su defecto, lo más rápido posible.

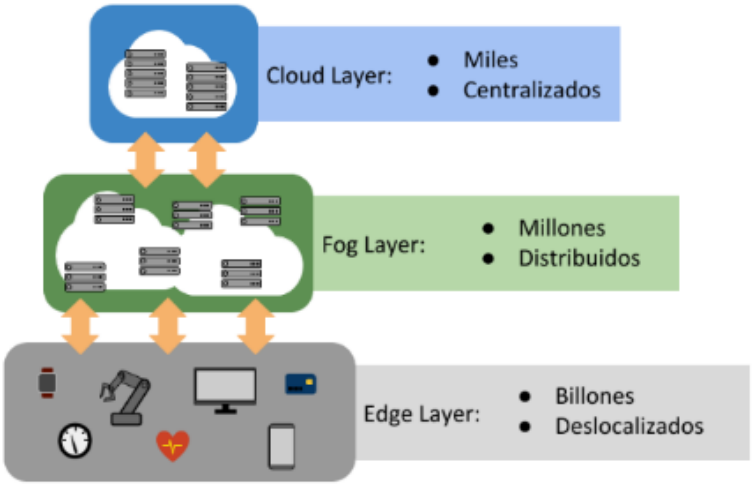


Ilustración 9. Técnicas y modelos de computación.

Otra ventaja notable del *Edge Computing* es la capacidad que ofrece de agregar y analizar datos masivos en el momento, lo cual nos permitirá la toma decisiones casi en tiempo real. Como concepto para resumir estas tecnologías podríamos decir que el *Edge Computing* es la clave para un procesamiento de datos rápido.

2.4. Situación actual de la Agricultura de Precisión.

Uno de los principales problemas antes los que se encuentra actualmente la agricultura de precisión en España es la posible falta de información y formación acerca de estas nuevas tecnologías. Son muchos los agricultores que evitan el contacto con las

tecnologías de la agricultura de precisión ante el gran desconocimiento que poseen acerca de ellas. Sin un adecuado asesoramiento en esta área, el usuario agrícola no conseguirá dar el paso hacia este tipo de agricultura.

Beneficios a destacar de la AP.

Los beneficios más destacables de la agricultura de precisión, que todos los usuarios agrícolas deberían conocer, podrían ser los que se nombran a continuación:

1) Reducción de insumos (pesticidas y fertilizantes).
2) Correcta gestión de las explotaciones agrícolas.
3) Reducción del impacto medioambiental de las plantaciones agrícolas.
4) Obtención de información detallada acerca de la producción.
5) Mejora de la producción final y mayor calidad del producto.

2.4.1. Propósitos y retos para la AP en España.

La agricultura de precisión puede ayudar a los agricultores a conseguir mejores cosechas, con mayor valor del producto obtenido, así como a reducir los costes en insumos, entre otros.

No solo la reducción de los costes es interesante en la agricultura de precisión, sino que también las ventajas medioambientales que esto ofrece son de gran importancia. Actualmente, las dosis desmesuradas e inapropiadas de insumos o fertilizantes en las grandes plantaciones agrícolas, causan cada día una mayor contaminación y son a su vez un mayor riesgo medioambiental. El poder conocer de manera exacta la cantidad correcta que cada sección de la plantación necesita, supone un gran avance en este aspecto medioambiental.

En numerosas ocasiones, los grandes cultivos son tratados en todas sus secciones por igual, lo que ocasiona que unas secciones reciban un correcto tratamiento, pero otras, un tratamiento que no corresponde a las necesidades de ese momento. La aplicación variable de herbicidas, resultaría altamente beneficiosa y ventajosa frente a la dosificación uniforme tradicional (como hablábamos anteriormente). Si podemos obtener una información acerca de las dosis que son necesarias en cada sección, podremos permitirnos reducir las dosis innecesarias en ciertas áreas de la producción agraria.

Principales desafíos.

Algunos de los principales desafíos que nos encontramos en cuanto a la introducción de la agricultura de precisión en España encontramos lo siguiente:

1) Proporcionar la correcta información a los agricultores que aún desconozcan los sistemas de agricultura de precisión, así como motivar al uso de estas tecnologías.
2) Desarrollar sistemas de investigación basados en condiciones locales.
3) Desarrollar habilidades en la recolección e interpretación de datos.
4) Mejorar los sistemas de bajo coste en el entorno de la agricultura de precisión para hacerla accesible a cualquier usuario.

2.4.2. Barreras para la implantación de la AP.

La agricultura de precisión no está disponible a cualquier usuario hoy en día, ya sea por razones de desconocimiento como se comentaba anteriormente, o por el elevado coste que supondría el uso de estas tecnologías para una parcela de tamaño no muy grande.

1) La agricultura de precisión encuentra su principal obstáculo en la accesibilidad que presenta a los usuarios agrícolas debido a su elevado coste en la gran mayoría de ocasiones.
2) Está pensada principalmente para parcelas de gran tamaño y para producciones elevadas.
3) El coste de los equipos se percibe alto por los agricultores sin un previo análisis de las ventajas que estos ofrecen.
4) Se requiere de información acerca de estas tecnologías, así como unas dosis de formación referentes al área informática y de los sistemas *IoT*.
5) A veces se presenta poca cultura referente al área de las nuevas tecnologías, como en este caso, la agricultura de precisión.

III

Capítulo 3

Diseño del sistema.

En este tercer capítulo, se detalla el diseño llevado a cabo para la realización de la aplicación final, destacando mayormente cómo funciona TensorFlow y TensorFlow Lite, este último de gran interés en este proyecto.

Capítulo 3. Diseño del sistema.

Como se comentaba anteriormente, el proyecto se centra en acercar a los usuarios una herramienta sencilla que les permita aplicarla y usarla a pie de campo, así como su uso como complemento a otras tecnologías. Para ello, se desarrolla una aplicación móvil para dispositivos *Android* mediante *Android Studio*. Se hace uso a su vez de *TensorFlow Lite*, para adaptar así el modelo obtenido mediante técnicas de *Deep Learning* y previamente entrenado para su implementación posterior en dispositivos móviles.

Un modelo de *TensorFlow* consiste en una gran estructura de datos que contiene el conocimiento de una red neuronal de "*Machine Learning*". Esta red ha sido previamente entrenada para resolver un problema concreto y así obtener el mencionado conocimiento, en nuestro caso, ha sido entrenada para determinar de manera aproximada la cantidad de producto frutal que posee el elemento analizado a pie de campo. En nuestro caso, el modelo usado está entrenado para la detección de limones en árboles limoneros.

Por otro lado, cabe mencionar que *TensorFlow* Lite es un conjunto de herramientas que se usan para permitir el uso de modelos de *TensorFlow* en móviles, sistemas embebidos o dispositivos *IoT*. Es por ello, que será de gran importancia a lo largo del desarrollo de la aplicación y de la integración del modelo a usar.

Para aplicar este modelo a nuestro proyecto, se hace uso del convertidor de *TensorFlow Lite*, el cual nos permitirá convertir el modelo de *TensorFlow* a *TensorFlow Lite* para ser usado en un dispositivo móvil Android. También se hace uso, dentro de Android Studio, del *interpreter* o intérprete, el cual nos permitirá que el modelo *tflite* pueda ser ejecutado de manera correcta en *Android Studio* por la aplicación desarrollada.

3.1. Introducción.

La AP, como se explicaba en el capítulo 2 de este informe, está cada día más y más presente. Los agricultores buscan nuevas técnicas, más precisas y de uso fácil, para poder sacar el máximo rendimiento a su producción. La mayoría de técnicas existentes de agricultura de precisión requieren de previas inversiones, costosas en su gran mayoría, y no son realmente asequibles para un amplio sector de agricultores.

Su difícil uso y su elevado coste hacen, añadido a su reciente, aunque creciente, entrada en el ámbito de la agricultura, que el agricultor se muestre dudoso ante la posibilidad

de usar este tipo de técnicas. Al ser tecnologías que en su gran mayoría requieren una cantidad de dinero de entrada considerablemente alto, los agricultores cuyas cosechas no son de gran volumen y cuyos ingresos no son realmente elevados, tienden a rechazar estas inversiones al creer que no serán rentables para su producción.

Actualmente, existe un creciente uso de técnicas de visión artificial, las cuales permiten crear nuevas modalidades de agricultura de precisión más asequibles para los agricultores. Estas técnicas ayudan a los agricultores a adquirir datos y conocimientos acerca de sus producciones de manera precisa y mucho más exacta que calculándolo a simple vista. Técnicas como *Deep Learning*, usada en este proyecto, son cada vez más comunes. Con ellas, se intenta acercar una nueva forma de analizar el cultivo al agricultor para su uso a pie de campo de forma sencilla y accesible.

El objetivo de este proyecto se centra en adaptar un modelo de *Deep Learning*, previamente entrenado, a su uso con una aplicación móvil para dispositivos Android. El proyecto se centra en crear una aplicación mediante el uso de Android Studio para su diseño, adaptando este modelo de *Deep Learning* gracias al convertidor de modelos *TensorFlow* a *TensorFlow Lite*. Posteriormente, lo que conseguiremos será una aplicación de sencillo uso que cualquier agricultor podrá tener a su alcance. Tras captar una foto con nuestra aplicación, el modelo muestra la cantidad de producto que existe en el árbol captado, dándonos de forma muy aproximada una estimación de la cantidad frutal que obtendremos de él mediante la coloración de tono rojo de los limones detectados.

De esta forma, para producciones de no muy elevado volumen, el agricultor podrá estimar su producción de forma mucho más precisa y fácil, sin inversiones grandes de dinero y sin técnicas costosas de agricultura de precisión. Por otro lado, esta herramienta será de gran utilidad para aquellos agricultores que ya hagan uso de otras técnicas, siendo esta aplicación un complemento para obtener mejores resultados de cara a la producción final.

A lo largo del capítulo, se hablará del hardware y el software usado. Los dispositivos móviles actualmente pueden llegar a tener grandes diferencias entre ellos. *Android* es un sistema operativo que permite gran flexibilidad a la hora de su personalización. A su vez, ofrece una gran capacidad para ofrecer diferentes experiencias a los usuarios. Sin embargo, no es solo el software lo más importante a comentar. El hardware es otra parte de gran relevancia: el procesador principal, el procesador gráfico, la memoria RAM, almacenamiento interno o la conectividad son algunos de los aspectos que definen el hardware de un

dispositivo móvil, y todo ello influirá notablemente a la hora de ejecutar la aplicación diseñada.

En resumen, cabe destacar los aspectos más importantes que se tratarán a lo largo de este capítulo, de gran importancia para comprender la estructura de este proyecto. Tres de los más importantes son la arquitectura del hardware y software (puntos 3.2 y 3.3), y, por otro lado, la tecnología y métodos usados (3.4 y 3.5). En ellos, se explica detalladamente el uso de las tecnologías usadas para el desarrollo de este proyecto y los métodos implementados. Por último, y muy importante, un esquema del diseño y la estructura de la aplicación general (3.6).

3.2. Arquitectura Hardware.

La aplicación móvil desarrollada consta esencialmente de un modelo previamente entrenado adaptado mediante *TensorFlow Lite* para su uso con un dispositivo móvil *Android*. Hay muchas formas de obtener un modelo de *TensorFlow*. Se pueden usar tanto modelos previamente entrenados como modelos que podríamos entrenar por nosotros mismos. En este proyecto se usa un modelo previamente entrenado y facilitado para adaptarlo a su uso con móviles *Android*.

Este modelo ha sido entrenado para poder identificar de manera automática la cantidad de producto existente en un árbol a pie de campo, con el principal objetivo de facilitar así el uso de esta herramienta para el agricultor. Éste podrá hacer estimaciones aproximadas de su cosecha, siendo esto un gran avance para conocer el estado de su producción.

El objetivo se centra en reconocer la cantidad de fruta existente en el árbol mediante una foto captada por nuestro dispositivo. En esta foto, y aplicando el modelo entrenado de *Deep Learning*, podremos obtener una aproximación de la cantidad de producto en ese árbol.

Haciendo referencia al hardware, para desarrollar una aplicación, para móviles en nuestro caso, contamos con herramientas adecuadas para el proceso de desarrollo. Para ello, en este proyecto se usa un SDK (*Software Development Kit* – Kit de Desarrollo de Software), el cual es un conjunto de herramientas que ayudan a la programación de aplicaciones para un entorno concreto.

Se muestra a continuación un cuadro resumen (*Tabla 1*) de las características del hardware del dispositivo usado para las pruebas y la realización del proyecto

PROCESADOR	
TIPO	HUAWEI Kirin 659
FRECUENCIA	4x2.36GHz (Cortex A53), 4x1.7GHz (Cortex A53)
SISTEMA OPERATIVO	
NOMBRE	Android 8.0, EMUI 8.0
MEMORIA	
INTERNO	4GB RAM, 64GB Flash Tarjeta microSDTM de hasta 256GB
EXTERNO	Debido a las limitaciones en el poder de procesamiento de la CPU y la memoria utilizada por el sistema operativo y las aplicaciones peinstaladas, el espacio disponible para los usuarios será menor que la capacidad de memoria nominal. El espacio de memoria real cambiará junto con las actualizaciones de la aplicación, las operaciones del usuario y otros factores relacionados.
CÁMARA	
TRASERA	16MP f2.2 + 2MP f2.4
FRONTAL	16MP f2.0, FF
BATERÍA	
BATERÍA	3000mAh (típico), 2900mAh (mínimo)
RED	
4G FDD	B1/B3/B7/B8/B20
3G WCDMA	B1/B2/B5/B8
CONECTIVIDAD	
WiFi	802.11 b/g/n/a/ac, 2.4GHz y 5GHz
BLUETOOH	BT 4.2, aptXTM, aptXTM HD
NFC	Sí
USB	USB Type-CTM
TIPO USB	USB 2.0
FUNCIONES	OTG, anclaje a red, MTP, PTP, carga

Tabla 1. Características Hardware usado

3.3. Arquitectura software.

Para la realización de este proyecto, como se ha comentado anteriormente, se ha usado principalmente Android Studio.

Android Studio es el entorno de desarrollo integrado oficial de la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014.

Android Studio está basado en el software *IntelliK IDEA* de *JetBrains* y ha sido publicado de forma gratuita a través de la Licencia Apache 2.0. Está disponible para las plataformas Microsoft Windows, macOS y GNU/Linux. Ha sido diseñado específicamente para el desarrollo de Android.

La última versión estable es la 3.5 y fue lanzada en agosto de 2019. Con esta última versión se trata de mejorar el rendimiento y de pulir las funciones ya existentes. El rendimiento es el gran protagonista de *Android Studio 3.5*. En cuanto al uso de memoria, el propio IDE detecta y recomienda si deberías aumentar la memoria RAM dedicada a la aplicación para mejorar el rendimiento.

Otro detalle que Android Studio comprobará es el antivirus, concretamente, si el análisis en tiempo real de ciertas carpetas usadas en el proyecto puede estar ralentizando el tiempo que se tarda en construir una aplicación. En total, se han corregido 600 errores, 50 *memory leaks*, 20 bloqueos de Android Studio y se ha corregido la velocidad de escritura en XML y *Kotlin*. Se actualiza también a *IntelliJ IDEA* 2019.1 y a *Gradle* 3.5.0. [13]

Como comentábamos anteriormente, *IntelliJ IDEA* es el software en el que está basado Android Studio. *IntelliJ IDEA* es un entorno de desarrollo integrado (IDE) para el desarrollo de programas informáticos. Es desarrollado por *JetBrains* (conocido previamente como *IntelliJ*) y está disponible en dos ediciones: edición para la comunidad y edición comercial.

Las dos ediciones (*Tabla 2*) difieren en su soporte para versión de software y sistemas de control revisiones: [14]

	IntelliJ IDEA Community Edition	IntelliJ IDEA Ultimate Edition
CVS	SÍ	SÍ
Git	SÍ	SÍ
Github	SÍ	SÍ
Mercurial	SÍ	SÍ
Subversion	SÍ	SÍ
Team Foundation Server	NO	SÍ
ClearCase	NO	SÍ
Perforce	NO	SÍ
Visual SourceSafe	NO	SÍ

Tabla 2. Ediciones IntelliJ IDEA

3.4. Tecnología usada.

La API (*Application Programming Interface* – Interfaz de programación de aplicaciones) es aquella que abre la puerta a que dos componentes de software diferentes se puedan comunicar. Así, ambos tendrán acceso a la información el uno del otro. Es por ello, un acceso seguro a desarrolladores para trabajar con programas y servicios externos de manera sencilla.

En *Android* contamos con numerosas APIs, las cuales ofrecen a los desarrolladores la posibilidad de gestionar de diferentes maneras sus dispositivos móviles. La cámara de nuestro *Smartphone* tiene su propia API. En este proyecto se ha hecho uso de *Camera2* API, una interfaz pensada para los desarrolladores. La API *Camera2* comenzó a aplicarse a partir de la versión 5.0 delos móviles *Android*. El objetivo principal que perseguía era facilitar a los desarrolladores los usos más avanzados y precisos de la cámara, como por ejemplo controlar la exposición, el enfoque automático o manual o incluso el tomar fotos en formato RAW.

3.4.1. Android Studio.

A la hora de desarrollar aplicaciones *Android*, usaremos *Android Studio*, el entorno de desarrollo integrado oficial para el desarrollo de apps para Android, basado en *IntelliJ IDEA*. Además de su potente editor de códigos, ofrece más funciones que aumentan la productividad a la hora de desarrollar aplicaciones *Android*. A continuación, se nombran algunas de ellas [15]:

1) Sistema de complicación flexible basado en *Gradle*.
2) Emulador rápido con numerosas funciones.
3) Entorno unificado donde se puede desarrollar para cualquier dispositivo *Android*.
4) Integración con *GitHub*, del cual se hace uso en este proyecto.
5) Compatibilidad con C++ y NDK.
6) Compatibilidad integrada para *Google Cloud Platform*, que facilita la integración con *Google Cloud Messaging* y *App Engine*.

Para comenzar a usar *Android Studio*, necesitaremos descargar el JDK (*Java Development Kit*), así como el paquete de instalación de *Android Studio* junto con el SDK de la plataforma. Una vez descargado *Android Studio* junto con el SDK, pasaremos a configurar el resto de componentes que, como mínimo, se deberán seleccionar para su instalación o actualización:

1) Android SDK Tools.
2) Android SDK Platform-tools.
3) Android SDK Build-tools.
4) Una o más versiones de la plataforma Android (las más recientes).
5) Android Support Repository (extras).
6) Google repository (extras).
7) Google Play services (extras).

El punto 4 es de los más importantes, ya que éste será el que contenga los componentes y librerías necesarias para desarrollar sobre cada una de las versiones concretar de Android. [16]

Para este proyecto se han descargado tanto la API 28 como la API 29, las dos últimas y más recientes para descargar.

A modo de referencia, se adjuntan los componentes descargados para trabajar en Android Studio. Algunos pueden estar previamente ya instalados por defecto:

1. Android SDK Tools - version 26.1.1
2. Android SDK Platform-tools
3. Android SDK Build-tools

4. Android SDK Build-Tools 30-rc2
5. GPU Debugging tools
6. NDK (Side by side)
7. Android Auto API Simulators
8. Android Emulator
9. Documentation for Android SDK
10. Google Play APK Expansion Library
11. Google Play Instant Development SDK
12. Google Play Licensing Library
13. Google Play services
14. Google USB Driver
15. Google Web Driver
16. Intel x86 Emulator Accelerator (HAXM installer)

Contenido del módulo principal.

Carpeta /app/src/main/java

Esta carpeta contiene todo el código de la aplicación desarrollada. Android Studio crea en primer lugar el código básico, *MainActivity*.

Carpeta /app/src/main/res

Contiene los ficheros de todos los recursos necesarios para desarrollar el proyecto:

1) /res/drawable/: imágenes y elementos gráficos usados en la aplicación.
2) /res/mipmap/: iconos de lanzamiento de la aplicación (icono de menú de aplicaciones del dispositivo)
3) /res/layout/: ficheros XML para las diferentes pantallas de la interfaz gráfica de la aplicación.
4) /res/anim/: animaciones usadas.
5) /res/color/: ficheros XML de definición de listas de colores.
6) /res/xml/: ficheros XML de datos.
7) /res/raw/: recursos adicionales.

8) /res/values/: recursos XML de la aplicación, como strings.xml, styles.xml, colors.xml…

Fichero /app/src/main/AndroidManifest.xml

Es un archivo de configuración de la aplicación. En él podemos configurar la app de manera básica, permitiendo en él los permisos necesarios para el uso de los diferentes recursos en la app (cámara, Internet, acceso a galería, etc) Este archivo describirá la información esencial de la app. Deberá contener el nombre del paquete de la aplicación, los componentes de la aplicación (actividades, servicios, etc), los permisos necesarios para poder acceder a las partes protegidas del sistema, y las funciones de hardware y software que requiere la aplicación.

Fichero /app/build.gradle

Gradle es un sistema de compilación que unifica las mejores prestaciones de otros sistemas de compilación. Está basado en JVM (*Java Virtual Machine*). Esto quiere decir, que, al escribir nuestro propio script de java, Android Studio lo entenderá y usará. Contiene la información referida a la compilación de la aplicación, como la versión SDK de Android que se usa y la mínima versión de Android que soportará la app, las librerías externas usadas, etc. En nuestro caso, la mínima versión SDK ha sido la 21. En este fichero también será donde declaremos las "dependencies" de la app. Una de ellas, y muy importante, será la de TensorFlow Lite:

```
implementation 'org.tensorflow:tensorflow-lite:1.12.0'
```

Componentes de una app Android.

Activity

Como bien indica su nombre (actividad), es cada una de las actividades, o pantallas, que forman una aplicación. Cada vez que queramos crear una nueva pantalla o vista para nuestra aplicación, crearemos un nuevo "*activity*". Dicho "*activity*" tendrá un archivo *.java* y un archivo *.xml*. El archivo *.xml* será aquel en el que trabajemos su diseño gráfico, mientras que el archivo .java será el archivo desde donde manejaremos el nuevo "*activity*".

Services

Los servicios son las tareas que se realizan en segundo plano. No contienen ninguna interfaz o vista por pantalla para el usuario.

Content Provider

Content Provider o "proveedor de contenido", el cual proporciona los datos a la app. Con ello podemos compartir datos sin preocuparnos por el almacenamiento. Con él se comparten datos entre aplicaciones. Se pueden compartir determinados datos de nuestra aplicación, todo ello sin mostrar detalles sobre su almacenamiento interno o su estructura.

Intent

Los "*intents"* son objetos usados para mandar mensajes y se considera el elemento básico de comunicación entre los distintos componentes Android descritos anteriormente. Gracias a un "*intent"*, podremos mostrar y lanzar las actividades.

Widget

Los "*widgets*" forman una parte esencial de la personalización de las pantallas visuales de la aplicación a desarrollar.

3.4.2. TensorFlow y TensorFlow Lite.

Actualmente, la Inteligencia Artifical (IA) está cambiando constantemente nuestro día a día. Tanto las técnicas de *Machine Learning* como de *Deep Learning* nos demuestran nuevos horizontes en el desarrollo de aplicaciones basadas en IA. *TensorFlow* es una biblioteca de *Machine Learning* y redes neuronales liberada por Google, pionero en IA, y es así una de las principales herramientas para el uso de IA.

Gracias a herramientas y tecnologías tan innovadoras como *TensorFlow*, la Inteligencia Artificial adaptada al día a día ya es posible. Se estima que en 2021 el 70% de las empresas emplearán tecnologías de IA. Las aplicaciones son extraordinarias, desde el reconocimiento de imágenes y voz, hasta el desarrollo de *bots* conversacionales. Podemos reconocer patrones, clasificar, recomendar o predecir en base a los datos. [17]

TensorFlow es la plataforma de Aprendizaje Profundo más importante del mundo. Este desarrollo *open-source* de Google va más allá de la Inteligencia Artificial, pero su flexibilidad y gran comunidad de desarrolladores lo ha posicionado como la herramienta líder en el sector del *Deep Learning*.

Más concretamente *TensorFlow* es una biblioteca de software de código abierto para computación numérica, que usa gráficos de flujo de datos. Los nodos en las gráficas representan operaciones matemáticas, mientras que los bordes de las gráficas representan las matrices de datos multidimensionales (tensores) comunicadas entre ellos. Es una gran plataforma para construir y entrenar redes neuronales, que permiten detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos. [18]

TensorFlow se construyó pensando en el código abierto y en la facilidad de ejecución y escalabilidad. Permite ser ejecutado en la nube, pero también en local. La idea es que cualquier tenga acceso y pueda ejecutarlo. Se puede ver a personas que lo ejecutan en una sola máquina, un único dispositivo, o en grandes clústeres. Si realmente se desea escalar, la nube es un gran lugar para hacerlo. Se puede obtener mucha automatización. [18]

En el mundo del IoT (*Internet of Things* – Internet de las cosas) es interesante el uso de técnicas de *Deep Learning*. Cada vez poseemos más y más datos, los cuales son altamente amplios, heterogéneos e indefinidos. Las técnicas de *Deep Learnig* y *Machine Learning* nos permiten crear con estos datos modelos de predicción, entrenados previamente para ese objetivo. Gracias a *TensorFlow*, podemos permitirnos la ejecución e implementación en distintos dispositivos o arquitecturas. *TensorFlow* es una plataforma disponible tanto para expertos como para aprendices, que hace fácil construir y entrenar modelos de *Machine Learning*.

Por otra parte, además de *TensorFlow* nos encontramos con *TensorFlow Lite*, la versión ligera de *TensorFlow* para dispositivos móviles o sistemas embebidos.

En este proyecto se usa como fuente principal *TensorFlow Lite*, que al igual que *TensorFlow*, es una plataforma de código abierto. La mayor diferencia entre ambos, es que *TensorFlow Lite* es usada para dispositivos móviles, por lo que será primordial en este proyecto.

Gracias a *TensorFlow Lite* podemos ejecutar modelos de *TensorFlow* en una gran cantidad de dispositivos. Los modelos de *TensorFlow* pueden ser modelos previamente entrenados o entrenados por nosotros mismos. Para poder usar un modelo *TensorFlow* en *TensorFlow Lite*, se deberá convertir antes para su posterior uso mediante el *converter*.

En nuestro caso, usaremos un modelo previamente entrenado. Nuestro modelo consiste principalmente en la identificación de la cantidad de limones existentes en un árbol examinado a pie de campo mediante una fotografía. Gracias a la imagen captada con nuestro dispositivo Android, se implementa en ella el modelo adaptado a *TensorFlow Lite* y añadido a la aplicación con *Android Studio*, para así obtener como resultado una imagen final con los limones hallados pintados en color rojo.

¿Cómo se obtiene un fichero en formato *TensorFlow Lite*?

Partiendo de nuestro modelo de *TensorFlow* previamente entrenado, lo que haremos será hacer uso de *TensorFlow Lite converter.* Con ello obtendremos el archivo *.tflite*, necesario a la hora de usar nuestro modelo *TensorFlow* en una aplicación Android. En nuestro caso, como se menciona, la aplicación será para dispositivos Android y se desarrollará mediante *Android Studio*. Una vez tengamos nuestro archivo de *TensorFlow Lite* para poder ser usado en Android, tendremos que hacer uso del *interpreter,* así como de la API Java, con la que trabajaremos a lo largo del proyecto para el desarrollo de la aplicación. El proceso a seguir se muestra en la siguiente imagen (*Ilustración 10*):

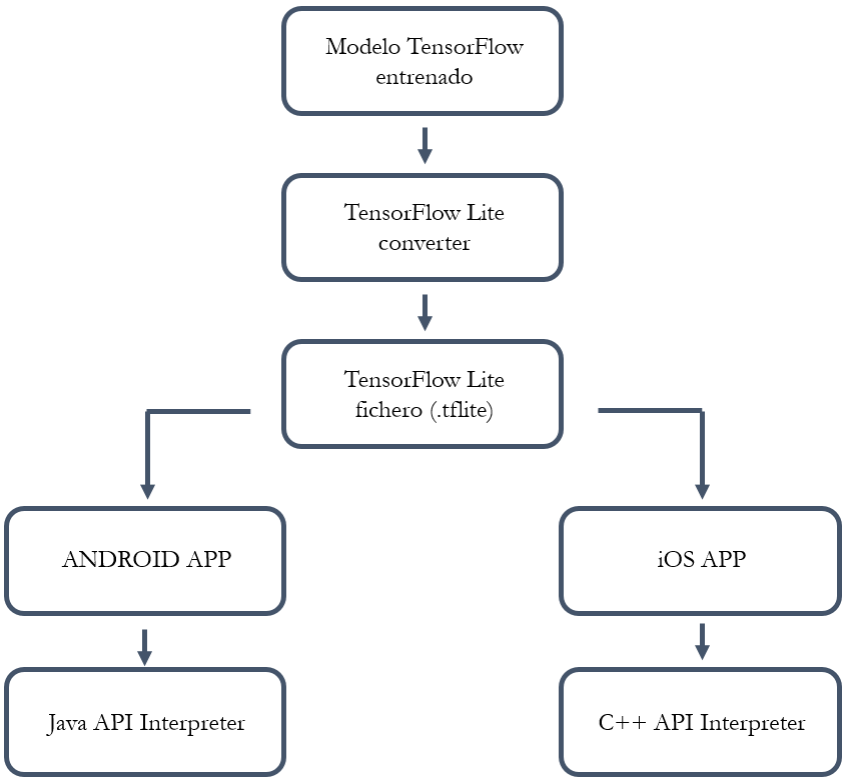


Ilustración 10. Estructura TensorFlow

Siguiendo la línea de la configuración de *TensorFlow Lite* para Android, será necesario añadir las dependencias correspondientes, como se muestra a continuación, dentro del *build.gradle*:

```
dependencies {
    implementation 'com.google.android.material:material:1.0.0'
    implementation 'androidx.annotation:annotation:1.0.2'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.0.0'
    annotationProcessor 'com.jakewharton:butterknife-compiler:10.0.0'
    implementation 'androidx.appcompat:appcompat:1.0.0-beta01'

    implementation 'com.camerakit:camerakit:1.0.0-beta3.11'
    implementation 'com.camerakit:jpegkit:0.1.0'
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk7:1.3.11'
    implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.0.0'
    implementation 'org.tensorflow:tensorflow-lite:1.12.0'
    implementation 'com.jakewharton:butterknife:10.0.0'

    implementation 'androidx.constraintlayout:constraintlayout:1.1.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test:runner:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'
```

Fragmento de código 1. Dependencies TensorFlow Lite

A la hora de importar el *interpreter*, será necesario, primeramente, importar la librería correspondiente. Esto se realizará en el *Classifier.java*, que será donde se haga uso de dicho *interpreter* para la evaluación de las sub-imágenes.

```
package com.irs.lemonapp;

import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.media.Image;
import android.widget.ImageView;

import org.tensorflow.lite.Interpreter;
```

Fragmento de código 2. Librería "interpreter"

Dentro del *Classifier.java*, será donde declararemos el *interpreter*. El código que se muestra a continuación se ha obtenido se la documentación facilitada por la página de TensorFlowLite.

```
public class Classifier {

    private static Interpreter interpreter;

    private Classifier(Interpreter interpreter) {
        this.interpreter = interpreter;
    }

    public static Classifier classifier(AssetManager assetManager, String modelPath) throws IOException {
        ByteBuffer byteBuffer = loadModelFile(assetManager, modelPath);
        Interpreter interpreter = new Interpreter(byteBuffer);
        return new Classifier(interpreter);
    }

    private static ByteBuffer loadModelFile(AssetManager assetManager, String modelPath) throws IOException {
        AssetFileDescriptor fileDescriptor = assetManager.openFd(modelPath);
        FileInputStream inputStream = new FileInputStream(fileDescriptor.getFileDescriptor());
        FileChannel fileChannel = inputStream.getChannel();
        long startOffset = fileDescriptor.getStartOffset();
        long declaredLength = fileDescriptor.getDeclaredLength();
        return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);
    }
```

Fragmento de código 3. "Interpreter" TF Lite

Por último, crearemos una nueva carpeta llamada "*assets*" en donde situaremos nuestro fichero *.tflite* con el modelo en la versión *TensorFlow Lite* para poder ser usada mediante Android.

3.5. Métodos usados.

3.5.1. Descripción del modelo de TensorFlow usado.

El modelo *TensorFlow* usado para esta aplicación móvil corresponde a un modelo de *Deep Learning* enfocado a la agricultura de precisión. Éste consiste en un sistema de predicción inteligente entrenado para estimar la cantidad de fruta (en nuestro caso, limones) que hay en un árbol a partir de una foto tomada con un dispositivo móvil Android.

Con este modelo basado en las técnicas de *Deep Learning*, y a través de una foto captada mediante una cámara *RGB*, se obtiene como respuesta de salida una identificación en dicha imagen de la fruta identificada en el árbol analizado.

El objetivo principal del entrenamiento del modelo es obtener un sistema capaz de identificar los limones que aparecen en imágenes de árboles limoneros usando técnicas de clasificación de imágenes.

Se obtienen, a partir de once imágenes reales clasificadas manualmente de 4608x2592 píxeles y, después de los procedimientos de pre-procesamiento y aumento, 641520 imágenes de 32x32. [19]

El proceso de aprendizaje está supervisado, ya que es necesario etiquetar las imágenes que serán usadas como ejemplo de forma manual. Mediante un *script Python* para esta tarea, se cargan así las imágenes de 32x32 para mostrar la imagen global. Posteriormente, una persona selecciona qué partes contienen elementos frutales. Una vez que la selección manual es realizada, cada una de las imágenes de 32x32 es clasificada como 1 o 0, siendo 1 cuando hay una fruta, y 0 cuando no la hay. Con la información sobre estas imágenes de 32x32, sus etiquetas de clasificación son correctamente guardadas en un archivo que será usado durante el proceso de aprendizaje y validación.

Los modelos de *Deep Learning* funcionan mejor tras un entrenamiento largo. Por ello, y para aumentar el número de muestras disponibles, evitando también la tediosa tarea de etiquetar manualmente una gran cantidad de nuevas imágenes, se usa una técnica de aumento de datos. Específicamente, se incluyen nuevas muestras sintéticas en el conjunto de datos aplicando diferentes tipos de transformaciones a muestras reales que ya estaban previamente etiquetadas. Estas transformaciones son rotaciones y "*mirroring*" (operaciones de copia de datos en tiempo real).

En cuanto a la arquitectura de la red neuronal, cabe destacar que está diseñada y basada en una clásica Red Neuronal Convolucional (*Convolutional Neural Network* – CNN). Ésta es usada para reconocer parones en imágenes genéricas. Es una técnica de *Deep Learning* profunda y ha sido seleccionada ya que trabajar con una CNN requiere un menor esfuerzo durante la etapa de pre-procesamiento, puesto que no requiere de la extracción de características explícitas.

De manera general, una CNN consiste en un conjunto secuencial de capas que transforma la entrada que se le proporciona en una salida determinada. Para este modelo, se hace uso de diferentes capas, las cuales se describen a continuación:

- La capa de entrada recibe imágenes *RGB* de tamaño 32x32, con 3 canales (espacio de color rojo, verde y azul). Cada imagen de entrada es correctamente convolucionada usando la función de *TensorFlow* "*tf.nn.conv2d*". Por tanto, por cada imagen de 32x32x3 de entrada, se obtiene una salida de 32x32x10. [20]

- Las redes convolucionales se van añadiendo. De esta forma, la salida de la primera capa convolucional se usa como entrada de la siguiente etapa definida como un paso suplementario en la convolución. Se añade una ReLU (*Rectified Linear Unit* – Unidad

Linear Rectificadora) para reducir la linearidad que puede ocasionar una convolución. Por tanto, se usa para potenciar la no-linearidad. [20]

- Se aplica entonces una segunda capa convolucional, definida como un conjunto de 15 filtros de 4x4. Obtenemos así, a partir de una entrada de 32x32x10, una salida de 16x16x15 a la que volveremos a aplicar una ReLU. [20]

3.5.2. Conversión del modelo TensorFlow a TensorFlow Lite.

A la hora de introducir nuestro modelo *TensorFlow* en la aplicación Android, necesitaremos previamente convertirlo a formato *TensorFlow Lite* (*.tflite*), para lo cual, haremos uso del *TensorFlow Lite converter*.

Para comprender mejor en qué consiste esta conversión, se muestra a en la *Ilustración 16* un esquema a modo de resumen:

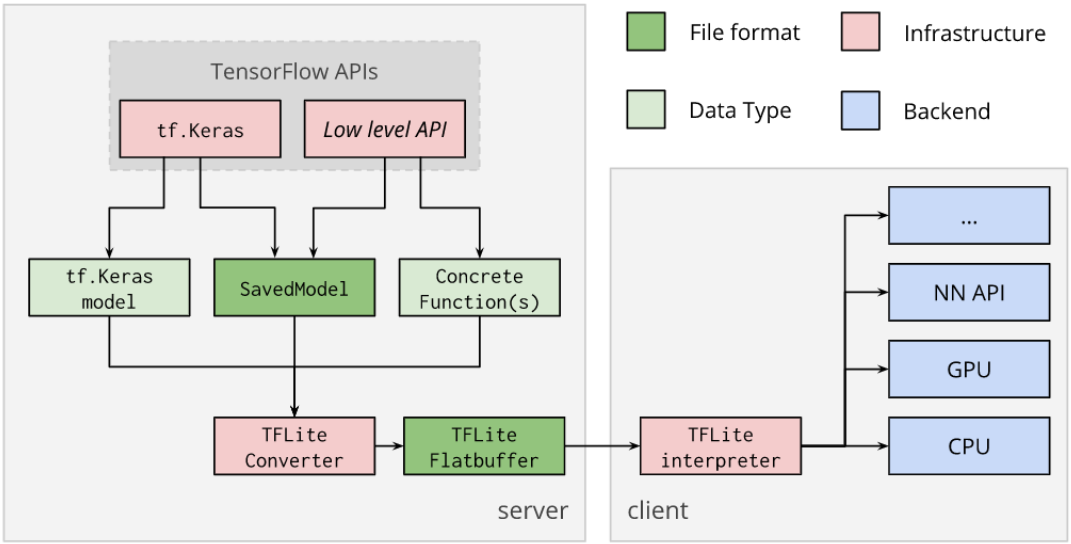


Ilustración 11. Device Deployment Tensor Flow

El *TensorFlow Lite converter*, consiste principalmente en la conversión de un modelo *TensorFlow* a *TensorFlow Lite*. Genera de este modo, un archivo *FlatBuffer* (*.tflite*) a partir del modelo *TensorFlow* proporcionado.

TensorFlow Lite converter soporta diferentes tipos de formatos de entrada como se aprecia en la figura. En nuestro caso, se usó el tipo *SavedModel*. Dicho tipo, contiene un programa completo de *TensorFlow* el cual no requiere la estructura original del código del modelo para ejecutarse, lo que lo hace mucho más versátil a la hora de compartirlo o transformarlo a formatos como *TensorFlow Lite*.

Una vez tengamos nuestro *SavedModel*, pasaremos a hacer uso del *TensorFlow Lite converter*, generando así el archivo *TensorFlow Lite Flatbuffer* para posteriormente, poder crear nuestro archivo *.tflite* con el que trabajaremos en Android Studio y que nos permitirá que nuestra aplicación móvil pueda leer dicho modelo. Por último, haremos uso del *TensorFlow Lite interpreter* para poder ejecutarlo en la aplicación móvil Android. De esto, hablaremos más adelante.

Para convertir un modelo en formato *SavedModel*, será necesario hacer uso de los siguientes comandos en *Python*:

```
#Importar tensorflow
import tensorflow as tf
#Convertir el modelo
converter =  tf.lite.TFLiteConverter.from_saved_model(export_dir)
tflite_model = converter.convert()
```

Fragmento de código 4. Conversión modelo TF a TF Lite

3.6. Diseño y estructura de la aplicación móvil.

La aplicación cuenta con un menú principal ubicado en el *MainActivity.java*. A partir de éste, se podrá acceder a las diferentes opciones que ofrece la aplicación: información de uso, galería o la opción de salir de la aplicación.

Mediante la opción de información, podremos encontrar una breve explicación acerca de cómo usar la app. Con el botón de galería, accederemos a un segundo activity (*Inicio.java*) donde podremos acceder a la galería del teléfono y seleccionar allí la imagen que queramos analizar. Una vez haya sido seleccionada dicha imagen, automáticamente comenzará el análisis. Al terminar éste, se muestra por pantalla una aproximación tanto de la

cantidad de limones como de su peso total de la producción del árbol estudiado. Por último, encontramos en el menú principal el botón salir, para, como bien dice, cerrar la aplicación y salir de ella.

A partir del esquema que se facilita a continuación (*Ilustración 12*), se muestra la estructura que sigue la aplicación diseñada.

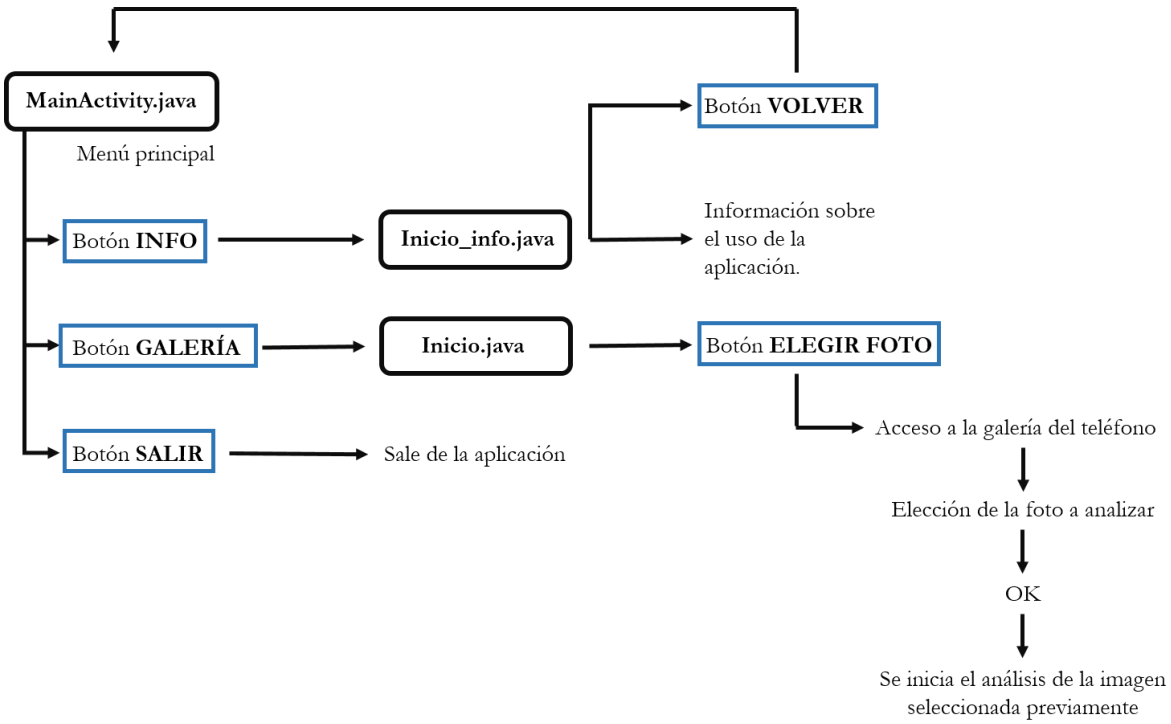


Ilustración 122. Esquema de funcionamiento de la app

El siguiente esquema ofrece una visión de los *layouts* correspondientes a cada actividad:

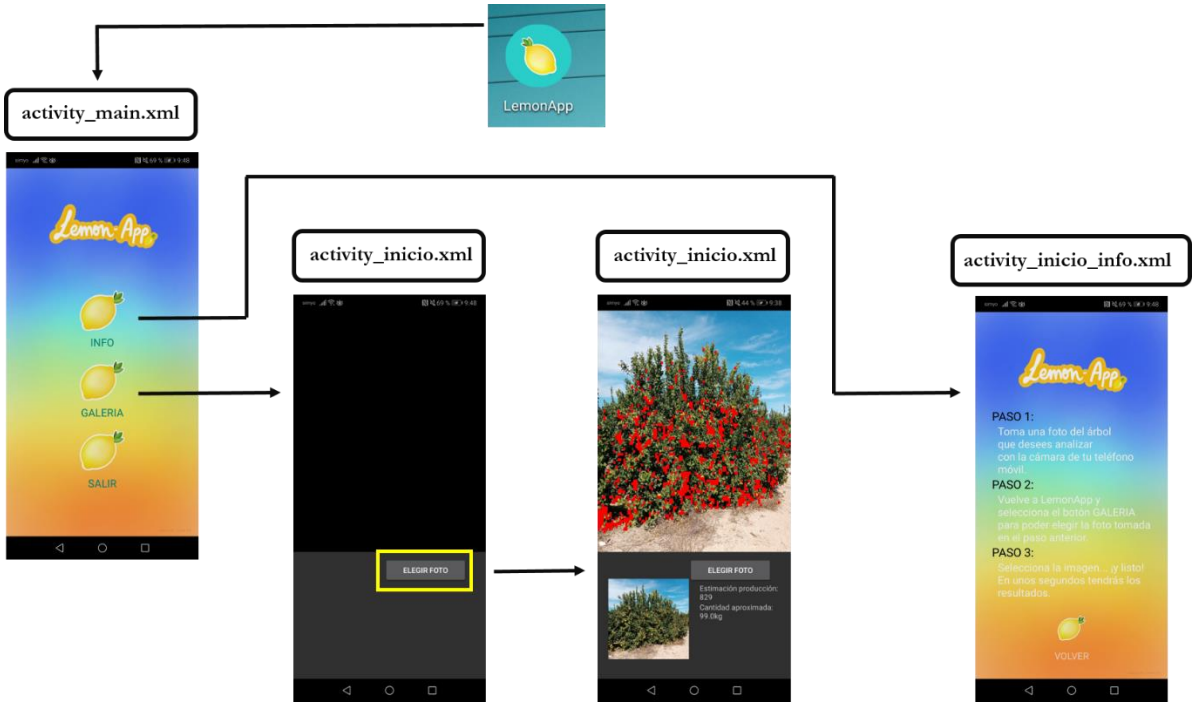


Ilustración 133. Esquema de las interfaces de usuario de la app

Capítulo 3.

IV

Capítulo 4

Procesamiento de la imagen

En este cuarto capítulo, se detalla el proceso de toma de imágenes, el procesado, pre-procesado y reconstrucción final de la imagen que se muestra por pantalla como resultado.

Capítulo 4. Procesamiento de la imagen.

4.1. Introducción.

La imagen será tomada con la aplicación por defecto de cámara de nuestro móvil, para posteriormente acceder a la galería desde la aplicación *LemonApp*. A partir de ahí, elegimos la imagen que deseamos analizar para obtener la información correspondiente.

Para este análisis, no solo necesitaremos el modelo de *TensorFlow* Lite, sino que será necesario una correcta lectura de dicha imagen. El modelo usado trabaja de una forma concreta, como se explicaba en el capítulo 3 de este informe: recibe imágenes de un tamaño específico (32x32), las cuales, antes de ser enviadas, serán estudiadas una a una para verificar si a dicha sección de la imagen original le corresponde un 1 (limón) o un 0 (no limón). A continuación, se describe detalladamente el proceso seguido para un correcto análisis de la imagen.

4.2. Selección de imagen.

El procedimiento de análisis de imágenes será a través de la galería del teléfono. Tomaremos las fotos deseadas con la aplicación de la cámara de nuestro teléfono móvil a pie de campo, y, una vez hechas, accederemos mediante *LemonApp* a la galería, donde seleccionaremos una a una las fotos que queramos analizar posteriormente.

Como se detallaba al comienzo, será necesario permitir el acceso y el uso a la galería del teléfono, para lo que necesitaremos habilitar los permisos correspondientes en el Android *Manifest*.

```
<uses-feature android:name="android.hardware.camera2.full" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Fragmento de código 5. Permisos acceso a galería

El procedimiento seguido para la adquisición de imágenes desde la galería se muestra a continuación.

En primer lugar, y dentro del *MainActivity.java*, declararemos el evento *onClick* sobre el botón de la galería. Con ello, accederemos a la siguiente actividad (*Inicio.java*). Dentro de ésta, podremos acceder, mediante un segundo botón de selección de imagen, a la galería del teléfono.

```
//BOTON GALERIA
ImageButton galeria = (ImageButton) findViewById(R.id.galeria);
galeria.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent1 = new Intent( packageContext: MainActivity.this, Inicio.class);
        startActivity(intent1);
    }
});
```

Fragmento de código 6. Evento onClick galería

Una vez hayamos accedido a la pantalla de elección de imagen de la galería, comenzará, en primer lugar, el pre-procesado de dicha imagen. Todo este proceso se lleva a cabo en "*Inicio.java*" y su correspondiente *layout* será "*activity_inicio.xml*".

Centrándonos principalmente en "*Inicio.java*", comenzaremos con la selección de la imagen de nuestra galería, tal y como se muestra a continuación.

```
    btnTakePhoto.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
            photoPickerIntent.setType("image/*");
            startActivityForResult(photoPickerIntent, PICK_IMAGE_REQUEST);
        }
    });

}
```

Fragmento de código 7. Acceso a galería para selección de imagen

4.3. Pre-procesado.

Una vez hayamos seleccionado la imagen, será necesario transformarla a formato *Bitmap*. Para ello, se hace uso de funciones como *BitmapFactory* (como se muestra a continuación). Además, será necesario que nuestra imagen original, una vez transformada a

formato *Bitmap*, sea también configurada en formato *ARGB_8888* ya que el modelo usado ha sido entrenado para reconocer imágenes en este formato concreto.

```
final int REQUEST_IMAGE_CAPTURE = 1;

@Override
protected void onActivityResult(int reqCode, int resultCode, Intent data) {
    super.onActivityResult(reqCode, resultCode, data);
    if (resultCode == RESULT_OK || reqCode == REQUEST_IMAGE_CAPTURE) {
        try {
            final Uri imageUri = data.getData();
            final InputStream imageStream = getContentResolver().openInputStream(imageUri);
            BitmapFactory.Options bitmapLoadingOptions = new BitmapFactory.Options();
            bitmapLoadingOptions.inPreferredConfig = Bitmap.Config.ARGB_8888;

            Bitmap origin_Image = BitmapFactory.decodeStream(imageStream,  outPadding: null, bitmapLoadingOptions);
            setPicture_origin(origin_Image);
```

Fragmento de código 8. Configuración Bitmap en formato ARGB_8888

Una vez hecho esto, el siguiente paso a realizar será la división de la imagen original en sub-imágenes de 32x32x3 que corresponden al tamaño de *input* del modelo de clasificación *TFLite*. Estas serán analizadas una a una para determinar si en ellas hay un limón o no.

Tomaremos el ancho y el alto de la imagen para así, posteriormente, definir cuántas sub-imágenes tendremos por fila y columna. Cada sub-imagen está definida por un índice de fila y columna, que indican su posición dentro de la imagen y que será necesario para la reconstrucción final.

Partiendo de la base de que la imagen no tiene por qué tener un división entera y exacta de píxeles en grupos de 32x32, deberemos contar con la posibilidad de que, en la gran mayoría de los casos, siempre habrá una pequeña sección que quedará fuera de los grupos de 32x32. Por ello, una vez hecha la división, tanto del ancho como del alto de la imagen, entre 32, deberemos sumar 1 al número de columnas y filas obtenidas.

```
int num_columns = (int)(original_image_width / 32) + 1;
int num_lines = (int)(original_image_height / 32) + 1;
```

Fragmento de código 9. Declaración de variables para columnas y filas

Una vez hemos recorrido la primera fila y sus respectivas columnas, tal y como se muestra en la imagen, se pasa a analizar la última sección que forma parte de la columna "falsa".

Para visualizar mejor este procedimiento, tomando como ejemplo la división no entera de columnas, se facilita el siguiente esquema:

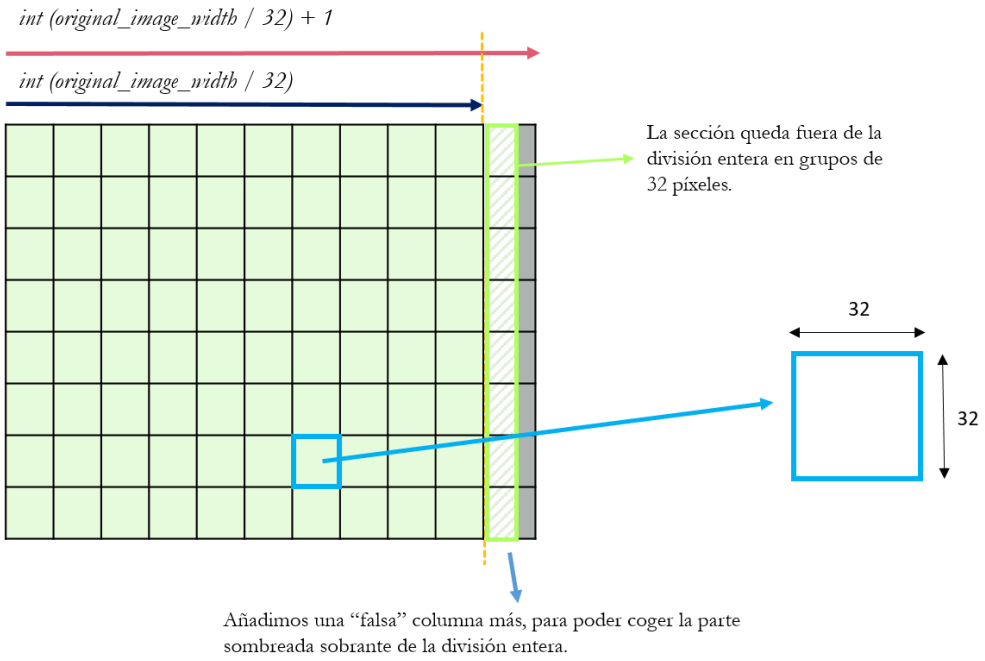


Ilustración 14. División no entera de columnas/filas de la imagen

De esta forma, si el ancho fuera, por ejemplo, de 560 píxeles, al hacer la división entera por 32 el resultado sería de 17,5 trozos de tamaño 32 píxeles. Sin embargo, la variable tomaría el valor 17 como entero y estaríamos obviando el resto. Incrementando en 1 este valor, pasaríamos a tener 18 columnas, pudiendo tener en cuenta el último índice.

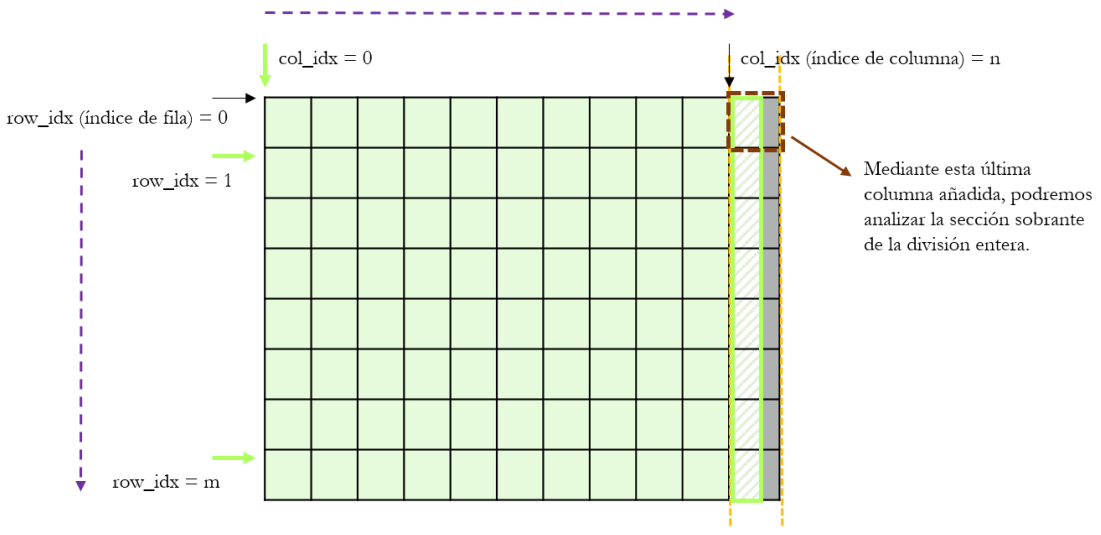


Ilustración 15. Lectura en sub-imágenes de la imagen original

El recorrido de la imagen se realizará mediante un doble *bucle for* que dividirá a la imagen original en cada una de las sub-imágenes obtenidas.

```
for (int row_idx = 0; row_idx < num_lines; ++row_idx) {
    for (int col_idx = 0; col_idx < num_columns; ++col_idx) {
        if (col_idx < num_columns - 1) {
            col_coord = 32 * col_idx;
        }
        else{
            col_coord = original_image_width - 33;
        }
        if (row_idx < num_lines - 1) {
            row_coord = 32 * row_idx;
        }
        else{
            row_coord = original_image_height - 33;
        }
        Bitmap crop_Image = Bitmap.createBitmap(origin_Image, col_coord, row_coord,
                                                crop_image_width, crop_image_height);
        long[][] crop_result = Classifier.recognizeImage(crop_Image);

        if (crop_result[0][0] == 1) {
            ++limones;
        }

        results_list[col_idx + row_idx*num_columns] = crop_result[0][0];
    }
}
```

Cada sub-imagen se envía al "*Classifier*" para su análisis con el modelo TFLite. .

Fragmento de código 110. Bucle for para recorrer la imagen por filas y columnas

En primer lugar, posicionamos ambos índices al comienzo de la imagen, ambos a 0. Una vez analizadas cada una de las columnas de la fila en la que estemos situados, se pasa a la siguiente y se vuelven a analizar todas sus columnas. De esta manera, se recorre la imagen de izquierda a derecha pasando por cada fila.

Mientras que la columna sea menor que el número de columnas determinado menos 1 (ya que habíamos incrementado en 1 ambas variables, para el número de columnas y para el número de filas), seguiremos avanzando hasta llegar a la última columna, o último índice. En este caso, la coordenada o índice deberá ser reubicado tomando el valor del tamaño original del ancho de la imagen y restándole 33.

```
for (int row_idx = 0; row_idx < num_lines; ++row_idx) {
    for (int col_idx = 0; col_idx < num_columns; ++col_idx) {
        if (col_idx < num_columns - 1) {
            col_coord = 32 * col_idx;
        }
        else{
            col_coord = original_image_width - 33;
        }
```

Comprobamos que el índice no sea mayor del último índice (el cual contiene la última sección que queda fuera de la división entera).

En el caso de ser superior, sería un indicador de que hemos llegado a la sección final. Por tanto, se realiza una reubicación del índice para analizar así la sección que falta.

Fragmento de código 11. Lectura de píxeles fuera de la división entera por 32

Conseguiremos así tomar la última sección que quedaba dentro de la "falsa" columna. (Se sigue el mismo proceso para las filas).

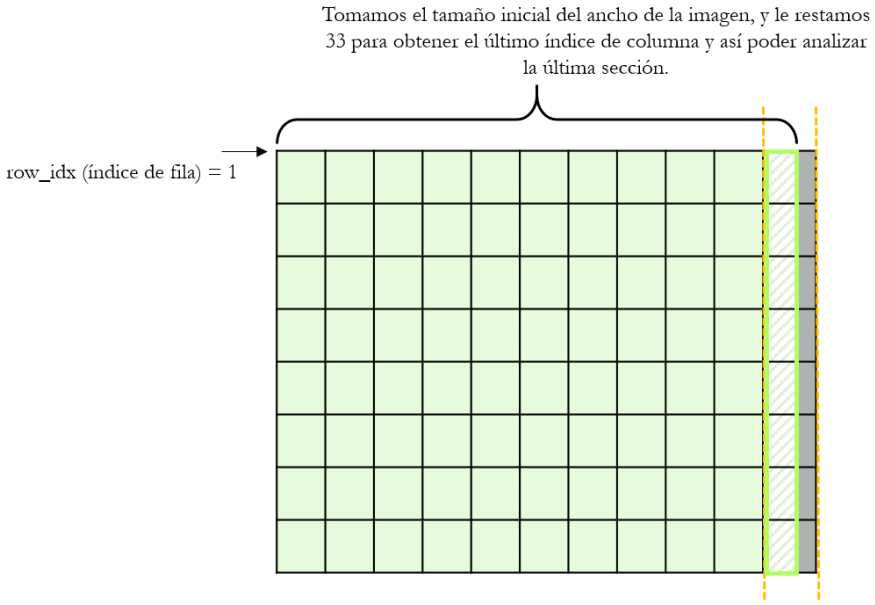


Ilustración 16. Columna falsa para el análisis de los píxeles sobrantes de la división entera por 32

En el caso de haber llegado a la última columna, pasaremos a realizar la reubicación del índice. Tomamos el tamaño inicial del ancho (o del largo, dependiendo si reubicamos índices de columnas o filas), y le restamos 33 para así poder contar con los últimos píxeles de la imagen pertenecientes a la columna "falsa" añadida anteriormente. En el siguiente esquema se muestra gráficamente en qué consiste esto.

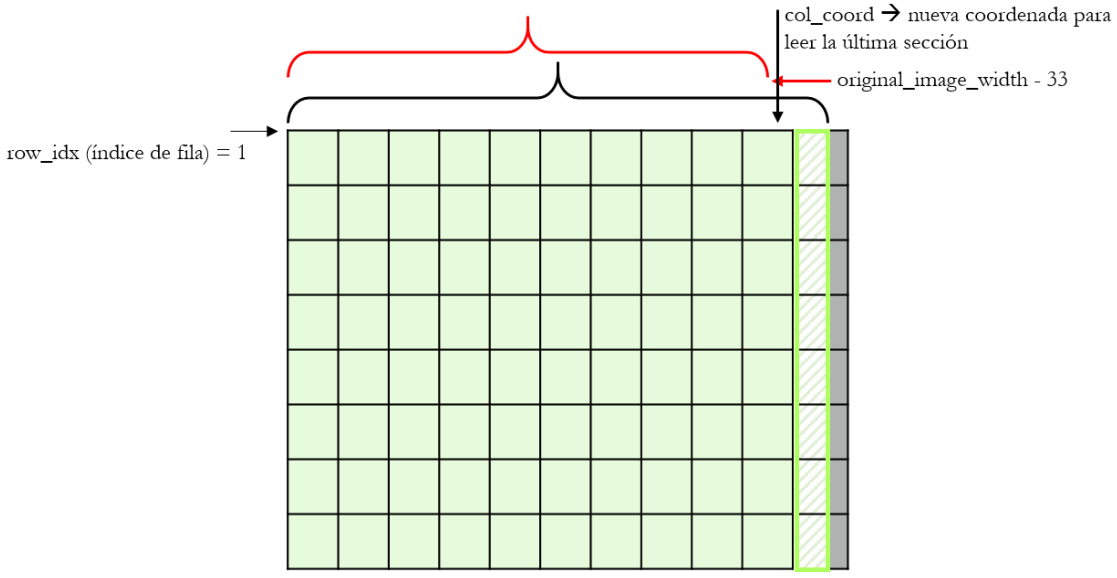


Ilustración 17. Reubicación del último índice de columna

Tras reubicar el índice en su correcta posición, obtendremos de nuevo una sub-imagen de 32x32. Tras ello, avanzaremos a la siguiente fila y comenzaremos una vez más con el análisis de las columnas, siguiendo el mismo proceso descrito anteriormente.

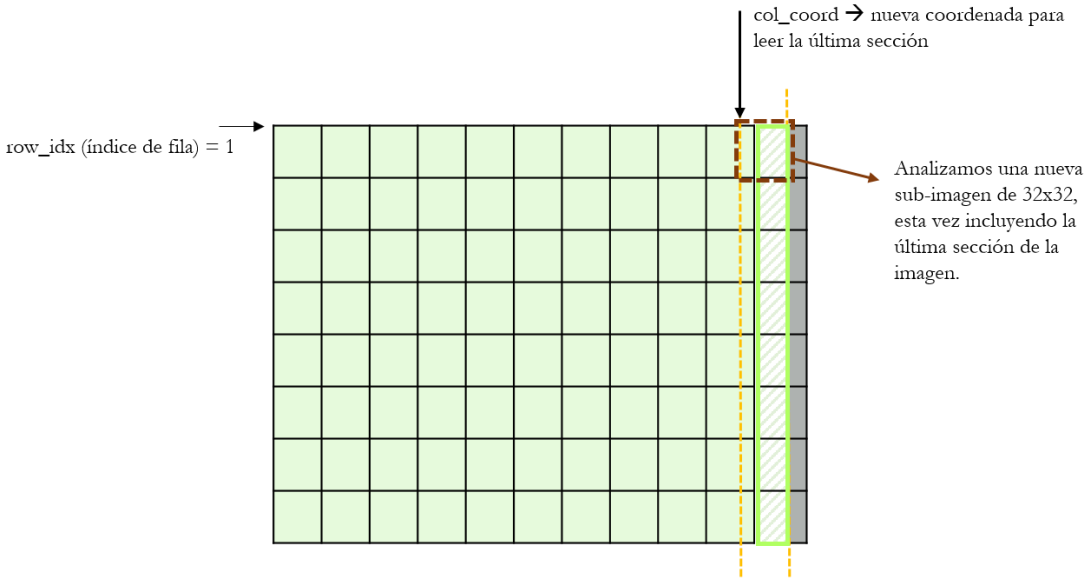


Ilustración 18. Lectura de la nueva columna a partir de la reubicación del índice de columna

Con el índice de la columna reubicado, analizamos una nueva sub-imagen de 32x32, incluyendo la sección que quedaba fuera de la división entera por 32. Una pequeña sección volvería a ser analizada ya que la nueva sub-imagen toma la parte anterior que había sido previamente analizada. Sin embargo, a efectos finales de imagen reconstruida no afectará, ya que será prácticamente inapreciable, y, sin embargo, conseguiremos incluir la última sección de píxeles que, de otra manera, quedarían sin ser analizados.

A lo largo del recorrido de la imagen, iremos guardando cada sub-imagen obtenida en un *array* de resultados, el cual utilizaremos más tarde para la reconstrucción final de la imagen resultado.

```
Bitmap crop_Image = Bitmap.createBitmap(origin_Image, col_coord, row_coord,
                                        crop_image_width, crop_image_height);
long[][] crop_result = Classifier.recognizeImage(crop_Image);

if (crop_result[0][0] == 1) {
    ++limones;
}

results_list[col_idx + row_idx*num_columns] = crop_result[0][0];
```

Índice de columna actual — Índice de fila actual — Número de columnas: *(ancho_imagen/32) + 1*

Fragmento de código 12. Array de resultados

4.4. Procesado y clasificación de sub-imágenes.

Partiendo del bucle *for* que comentábamos en el punto anterior, cabe destacar que cada sub-imagen de 32x32 obtenida, es enviada al *Classifier* para su análisis junto con el modelo *TFLite*. Estas sub-imágenes han de ser convertidas a formato *Bitmap* para un correcto procesado. Por tanto, conforme recorremos la fila pasando por cada columna y tomando las correspondientes sub-imágenes de 32x32, las iremos enviando en formato *Bitmap* al *Classifier*, concretamente, a la función "*recognizeImage*".

```
Bitmap crop_Image = Bitmap.createBitmap(origin_Image, col_coord, row_coord,
                                        crop_image_width, crop_image_height);
long[][] crop_result = Classifier.recognizeImage(crop_Image);
```

Fragmento de código 13. RecognizeImage

Una vez que esta sub-imagen se envía al *Classifier*, nos encontramos con el siguiente procedimiento: la función *recognizeImage* recibe un *Bitmap*, el cual deberá, a su vez, convertir a formato *ByteBuffer* para que el modelo pueda evaluarlo. Una vez entramos en la función *recognizeImage*, pasamos inmediatamente a la siguiente función: *convertBitmapToByteBuffer*. Aquí, como indica su nombre, convertiremos el *Bitmap* recibido en la función anterior (*recognizeImage*) a *ByteBuffer*, y obtendremos sus canales RGB para que más tarde el *interpreter* evalúe si en el *Bitmap* recibido hay o no un limón.

```
public static long[][] recognizeImage(Bitmap bitmap) {
    ByteBuffer byteBuffer = convertBitmapToByteBuffer(bitmap);
    long[][] result = new long[1][1];
    interpreter.run(byteBuffer, result);
    return result;
}

private static ByteBuffer convertBitmapToByteBuffer(Bitmap bitmap) {
    ByteBuffer byteBuffer = ByteBuffer.allocateDirect(ModelConfig.MODEL_INPUT_SIZE);
    byteBuffer.order(ByteOrder.nativeOrder());
    int[] pixels = new int[ModelConfig.INPUT_IMG_SIZE_WIDTH * ModelConfig.INPUT_IMG_SIZE_HEIGHT];
    bitmap.getPixels(pixels, offset: 0, bitmap.getWidth(), x: 0, y: 0, bitmap.getWidth(), bitmap.getHeight());
    for (int pixel : pixels) {
        float rChannel = (pixel >> 16) & 0xFF;
        float gChannel = (pixel >> 8) & 0xFF;
        float bChannel = (pixel) & 0xFF;
        byteBuffer.putFloat(rChannel);
        byteBuffer.putFloat(gChannel);
        byteBuffer.putFloat(bChannel);
    }
    return byteBuffer;
}
```

Una vez que el bitmap ha sido convertido a ByteBuffer, podemos enviarlo al "interpreter" para evaluar si en él hay un limón o no.

Obtenemos los valores RGB correspondientes a ese Bitmap

Fragmento de código 14 RecognizeImage, Classifier.java

Una vez la función *convertBitmapToByteBuffer* ha terminado, nos devolverá nuestro *byteBuffer*, el cual, recibirá el *interpreter* para ser analizado y evaluado.

Para poder cargar el *interpreter* y el modelo, se necesitará el siguiente código:

```
private static Interpreter interpreter;

private Classifier(Interpreter interpreter) {
    this.interpreter = interpreter;
}

public static Classifier classifier(AssetManager assetManager, String modelPath) throws IOException {
    ByteBuffer byteBuffer = loadModelFile(assetManager, modelPath);
    Interpreter interpreter = new Interpreter(byteBuffer);
    return new Classifier(interpreter);
}

private static ByteBuffer loadModelFile(AssetManager assetManager, String modelPath) throws IOException {
    AssetFileDescriptor fileDescriptor = assetManager.openFd(modelPath);
    FileInputStream inputStream = new FileInputStream(fileDescriptor.getFileDescriptor());
    FileChannel fileChannel = inputStream.getChannel();
    long startOffset = fileDescriptor.getStartOffset();
    long declaredLength = fileDescriptor.getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, declaredLength);
}
```

Fragmento de código 15. Interpreter, Classifier.java

Una vez se haya recorrido toda la imagen como se explicaba en el punto anterior, ya tendremos, a partir de la imagen original, las correspondientes sub-imágenes de 32x32x3, las cuales habrán pasado una a una por el *interpreter* para determinar si en ellas hay un limón o no (lo cual se determina a partir de los canales RGB existentes en dicha sub-imagen).

A partir de este momento en el que todas las sub-imágenes ya están analizadas, se pasa a la reconstrucción de la imagen final, con las sub-imágenes que contengan un limón pintadas de color rojo. Esto nos dará una imagen final con los limones encontrados pintados de dicho color rojo, lo cual nos permitirá realizar una estimación tanto de la producción como del peso total que ese árbol podría tener.

4.5. Reconstrucción de la imagen final.

Para partir de una idea clara y general, diremos que, en primer lugar, recorremos la imagen original tomando pequeñas sub-imágenes a partir de ésta del tamaño de 32x32x3. Cada una de esas sub-imágenes obtenidas es analizada por el *interpreter* y el modelo *TFLite*, gracias a los cuales se determinará si en dicha sub-imagen hay un limón o no, devolviendo como valor indicador de que existe un limón un 1, y en caso contrario, un 0.

Una vez se haya recorrido toda la imagen original, con los respectivos análisis de cada sub-imagen, ya tendríamos el *array* de resultados completo con los valores finales.

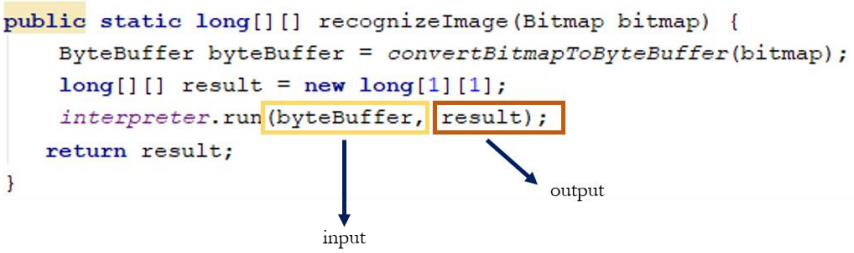
A su vez, se mostrarán por pantalla tanto la imagen original como la imagen resultado, con los limones detectados pintados en color rojo.

La reconstrucción parte de los resultados que se obtienen del análisis de sub-imágenes en el *Classifier*. Una vez se ha obtenido dicha sub-imagen, ésta pasaba a ser procesada en la función "*recognizeImage*".

```
Bitmap crop_Image = Bitmap.createBitmap(origin_Image, col_coord, row_coord,
                                        crop_image_width, crop_image_height);
long[][] crop_result = Classifier.recognizeImage(crop_Image);
```

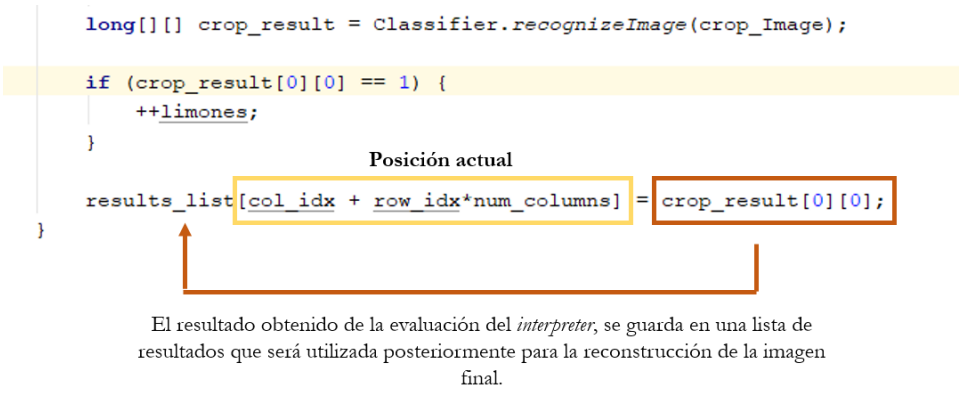
Fragmento de código 16. Resultado interpreter, "crop_result"

Una vez enviada la imagen en formato *bitmap* de 32x32, se convertía a formato *ByteBuffer* para posteriormente, pasarla al *interpreter*. Declaramos también la variable *result*, que será de tipo *long[ ] [ ]*. El *interpreter* recibe como *input* la imagen en formato *ByteBuffer*, y como output proporcionará un valor a la variable *result*.



Fragmento de código 17. Interpreter.run

El resultado que nos da el *interpreter* se guarda en la variable *long* "*crop_result*". A su vez, crearemos una lista de resultados, donde indicaremos qué valor ha evaluado el *interpreter* en cada posición que estemos analizando. Por ello, se indicará mediante el índice de columna actual y el índice de fila multiplicado por el número de columnas totales que hay en la imagen.



Fragmento de código 18. Lista de resultados "result_list"

El resultado será valor 1 o 0, lo cual nos permitirá más tarde modificar el color del *bitmap* donde se haya obtenido como valor un 1 (limón). El *bitmap* final se reconstruye a través de la función *reconstruct_image*.

```
Bitmap result_Image = reconstruct_image(origin_Image, results_list);
```

Fragmento de código 19. Reconstrucción del bitmap final a partir de "reconsruct_image"

Para comenzar con la reconstrucción de la imagen resultado, en primer lugar, crearemos una copia de la imagen original (*bpo*) en un nuevo *bitmap (bp)*, que será aquel en el cual modificaremos las sub-imágenes que contengan un limón. La modificación consistirá en analizar la lista de resultados según la posición en la que estemos en ese momento, y en el caso de tener como valor un 1, pondremos esa sub-imagen en color rojo.

De esta forma, seguiremos teniendo la imagen original seleccionada para poder mostrarla por pantalla, así como la imagen modificada y reconstruida que también se visualizará junto con la original.

Nos centramos ahora en la función "*reconstruct_image*". En ella será donde al comienzo, crearemos la nueva imagen *bitmap* (*bp*) a partir de la original (*bpo*). Igualmente, tendremos los mismos parámetros que para el bucle de lectura de la imagen original, donde se declaraban los valores de ancho y alto que tomaban las sub-imágenes.

Creamos el nuevo bitmap a partir del bitmap original.
Aquí será donde hagamos las correspondientes modificaciones de la imagen final.

```
private static Bitmap reconstruct_image(Bitmap bpo, long[] classification_list)
{
    Bitmap bp = bpo.copy(bpo.getConfig(), isMutable: true);
    int prediction_image_width = bp.getWidth();
    int prediction_image_height = bp.getHeight();

    int crop_image_width = 32;
    int crop_image_height = 32;
    int col_coord;
    int row_coord;
```

Copiamos del bitmap original su configuración en el nuevo bitmap bp.

Fragmento de código 20. Creación de nuevo bitmap para imagen final a partir del bitmap original

De nuevo, declaramos el número de columnas y filas, sumando 1 a cada variable para seguir el mismo proceso que en el pre-procesado.

```
int num_columns = (int)(prediction_image_width / 32) + 1;
int num_lines = (int)(prediction_image_height / 32) + 1;
```

Fragmento de código 21. Variables para el número de columnas y filas

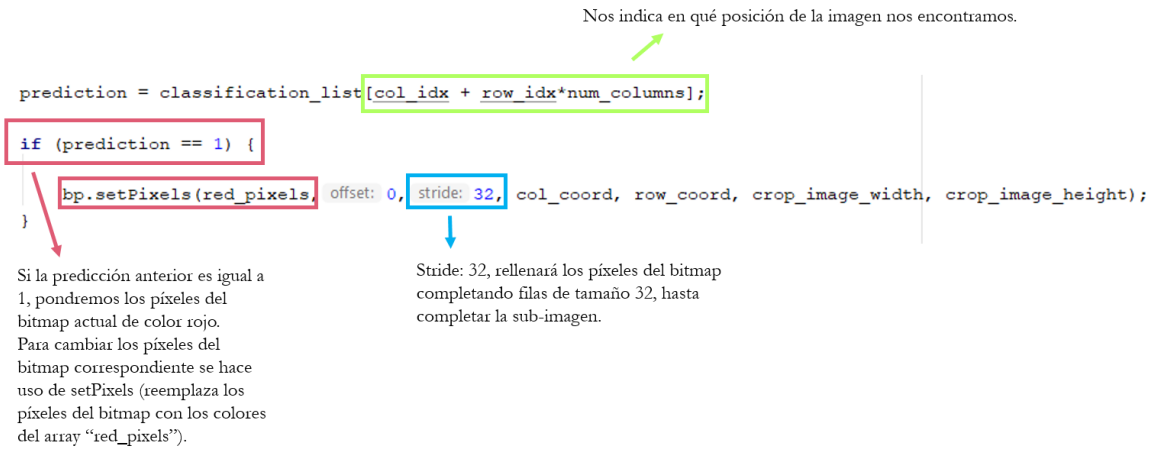
El bucle para analizar la imagen será exactamente igual al del pre-procesado. Al comienzo declaramos una variable de tipo *long* que se llamará "*prediction*".

```
long prediction;

for (int row_idx = 0; row_idx < num_lines; ++row_idx) {
    for (int col_idx = 0; col_idx < num_columns; ++col_idx) {

        if (col_idx < num_columns - 1) {
            col_coord = 32 * col_idx;
        }
        else{
            col_coord = prediction_image_width - 33;
        }

        if (row_idx < num_lines - 1) {
            row_coord = 32 * row_idx;
        }
        else{
            row_coord = prediction_image_height - 33;
        }
```

El proceso de análisis de la imagen será el mismo que el seguido en el pre-procesado, explicado detalladamente anteriormente en el apartado 4.3.

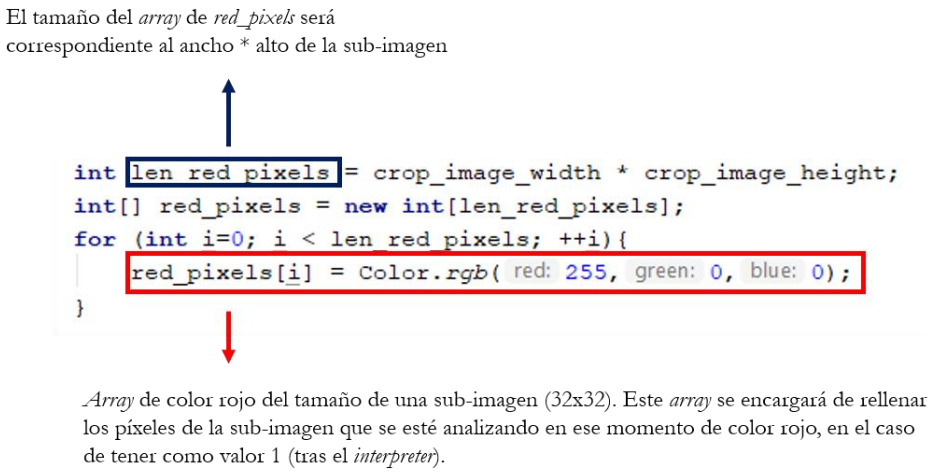
Fragmento de código 22. Variable long "prediction" para reconstrucción de imgen final

La variable "*prediction*" recibe el valor de la posición correspondiente en ese momento, que previamente ya se ha analizado en el *interpreter*, para determinar si es un 1 o un 0. Si la predicción anterior es 1, pintaremos de color rojo los píxeles de la sub-imagen correspondiente de 32x32.



Fragmento de código 23. Funcionamiento de la variable "prediction" y función "setPixels"

Usamos la función "*setPixels*" en el *bitmap bp* para cambiar los píxeles de la sub-imagen correspondiente. Los píxeles se cambiarán a color rojo ("*red_pixels*").



Fragmento de código 24. Array de color rojo para rellenar los píxeles de la sub-imagen final

De esta forma, seguiremos teniendo la imagen original sin modificar, así como la imagen final con la reconstrucción de sub-imágenes en color rojo. El efecto final será una imagen resultado con los limones que hay en ella pintados de rojo.

La propuesta que se hace en este proyecto es una estimación de la producción. Se toma cada sub-imagen de color rojo como si ésta fuera un limón. Al hacer el recuento total de sub-imágenes detectadas con valor 1, se tomará ese valor como la cantidad total de limones que hay en el árbol. Como la imagen se toma desde una sola perspectiva, solo se analiza el árbol por una zona. Esto equivaldría a tomar varias fotos del árbol teniendo en cuenta para la estimación sus diferentes perspectivas.

Para determinar el peso estimado de la producción del árbol, se toma el valor total de limones detectados y se multiplica por el peso medio de un limón (120gr). En el capítulo 6, apartado 6.2 (trabajo futuro), se proponen algunas mejoras para esta parte del proyecto.

V

Capítulo 5

Resultados finales obtenidos

En este quinto capítulo, se muestran los resultados finales obtenidos tras la realización de varias pruebas a pie de campo.

Capítulo 5. Resultados obtenidos.

5.1. Introducción.

Para la prueba de la aplicación desarrollada se tomaron diferentes imágenes a pie de campo en una finca de limoneros. Se realizaron varias fotos a pie de campo para su posterior procesamiento. Las fotos fueron tomadas con la aplicación de cámara del móvil y posteriormente, seleccionadas a través de la aplicación *LemonApp* mediante la opción de "galería".

5.2. Resultados obtenidos.

Las fotos corresponden con diferentes árboles y tamaños, para así poder comprobar el correcto procesamiento sea cual sea el tamaño de la imagen. Para una mejor visualización, se recomienda que la fotografía tomada sea vertical. Los resultados aproximados tanto de cantidad como de peso de la producción se aproximan notablemente a la realidad.

Los resultados obtenidos tras el análisis de algunas de las fotos tomadas a pie de campo fueron los siguientes:

Ilustración 19. Resultado 1

Ilustración 20. Resultado 2

Ilustración 21. Resultado 3

Ilustración 22. Resultado 4

Ilustración 23. Resultado 5

VI

Capítulo 6

Conclusiones y trabajo futuro

En este sexto capítulo, se exponen las conclusiones obtenidas tras el desarrollo de este Trabajo Fin de Grado. Se proponen también sugerencias de trabajo futuro para la mejora e implementación de la app desarrollada.

Capítulo 6. Conclusiones y trabajo futuro.

6.1. Conclusiones.

A lo largo del desarrollo de esta aplicación, y tras haber investigado en profundidad acerca de las diferentes técnicas nombradas en los anteriores capítulos, he obtenido varias conclusiones.

En primer lugar, cabe destacar que el modelo usado funciona de una manera muy concreta, lo cual determina la forma en la que las imágenes deberán ser pasadas como *input* a éste. Al inicio del proyecto se investigó acerca de un procesamiento en tiempo real, lo cual, fue descartado debido a la incompatibilidad del modelo con este sistema de análisis. El modelo tarda varios segundos e incluso minutos en procesar con el teléfono móvil la imagen cargada, por lo que un procesamiento de imágenes en tiempo real es simplemente incompatible.

El modelo trabaja recorriendo la imagen por filas y columnas, recibiendo como *input* una sub-imagen de 32x32x3 y analizando esta, para finalmente decidir si la sub-imagen tiene limón o no. Si el procesamiento se hiciera en tiempo real, y contando con que debe recorrer la imagen por filas y columnas, debemos tener en cuenta que la imagen en tiempo real nunca será fija, por lo que sería imposible procesar en sub-imágenes las tomas en tiempo real. Por todo ello, pasé a las siguientes opciones: toma de imágenes individuales y acceso a galería.

Para la siguiente opción (la toma de imágenes individuales a través de la aplicación), se intentó el uso de la API *Camera2* para la toma de imágenes desde la aplicación desarrollada. Esta API ofrecía numerosos inconvenientes a lo largo de su desarrollo. Dichos inconvenientes estaban ligados, en su mayoría, al dispositivo móvil que se usaba ya que requiere de diferentes funciones que deberá proporcionar el teléfono móvil. Uno de esos problemas era el *focus*. Al intentar capturar la imagen, ésta se tomaba antes de ser enfocada y, por tanto, no se podía procesar. Por otro lado, había problemas derivados, como se explicaba, al teléfono móvil usado.

Por último, la opción escogida fue la toma de imágenes fuera de la aplicación desarrollada, mediante el uso de la aplicación de cámara por defecto del dispositivo móvil.

De esta forma, la imagen sería fija, se podrían tomar diferentes fotografías a la vez y una vez hecho esto, podrían ser seleccionadas una a una para su procesamiento individual. Además, permite así una variada toma de imágenes, las cuales podremos analizar

tranquilamente y seleccionando aquellas que más nos interesen. A su vez, la estimación de la producción resultó ser muy próxima a la realidad. Esta estimación se añadió con el fin de proporcionar un campo de información adicional al análisis del árbol. Actualmente, el cálculo es meramente estimativo. Esta parte se propone como trabajo futuro para su mejora. La cantidad de sub-imágenes detectadas como "limón" será definida como la cantidad de limones del árbol analizado, multiplicando cada una de estas sub-imágenes por el peso medio de un limón (se marcó un peso de 120gr).

6.2. Trabajo futuro.

En cuanto al trabajo futuro, se plantea primeramente la posibilidad de integrar la API *Camera2* a partir del código ya proporcionado dentro de la aplicación en Android Studio, o simplemente, la toma de imágenes a través de la propia aplicación.

La toma de imágenes dentro de la aplicación se podría hacer usando tanto la API Camera2, como delegando el trabajo a la cámara del teléfono móvil a partir de la app desarrollada. Para el uso de la API *Camera2*, se proporciona el código necesario para su implementación. Será también necesario asegurarse de que el teléfono móvil que se vaya a usar para el desarrollo sea compatible con dicha API (esto podrá determinarse como se comentaba anteriormente, mediante una aplicación llamada *Camera2 Probe*). Otra interesante implementación será la captura de la imagen desde la propia aplicación, facilitando ambas opciones: el acceso a galería para seleccionar desde ella las imágenes que se deseen procesar, y la captura directa de fotografía desde la aplicación móvil, habilitando, por tanto, la delegación de esta actividad desde la app a la cámara del teléfono usado.

Otros aspectos y detalles a mejorar serán la selección del peso medio del limón en los árboles que se analicen o una sección de la aplicación que guarde los resultados obtenidos en análisis anteriores.

Referente al peso de los limones, cabe destacar que, dependiendo del tamaño del árbol y los años que éste tenga, el peso del limón podrá variar entre los 100 - 200 gramos aproximadamente. El usuario podría seleccionar el peso medio de los limones que los árboles de su producción proporcionan, por lo que finalmente la estimación sería más aproximada.

En cuanto a la segunda mejora, sería interesante la creación de un nuevo *activity* donde se guarden aquellos resultados que se deseen, creando una lista de estimaciones en el tiempo y pudiendo así comprobar cómo evoluciona la producción a lo largo de los meses.

Anexos

Anexos

Como trabajo futuro se proponen varias opciones. Principalmente, se basan en la adaptación de la aplicación a la toma de fotografías desde ella, o delegando el trabajo a la aplicación de cámara predeterminada. Se hace una breve introducción a la toma de imágenes mediante Android Studio. Para habilitar la cámara y captura de imágenes en Android Studio para nuestra aplicación móvil, se requieren de diversos permisos en el fichero *Android Manifest*. En él, especificaremos todos aquellos permisos referentes al uso de la cámara del dispositivo móvil, la captura de imágenes, la lectura de la galería, o los permisos referidos a las distintas *APIs*. [21] [22]

En primer lugar, habilitaremos los permisos de la cámara de nuestro móvil, así como el permiso al acceso de cámara que se pedirá la primera vez que se ejecute la aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.frogermcs.mnist">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature android:name="android.hardware.camera2.full" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
```

Fragmento de código 25. Permisos de cámara

Además, habilitaremos los permisos de autofocus para así facilitar la tarea de enfocado de la imagen, ya que sin un correcto *focus*, la imagen no será bien procesada.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.frogermcs.mnist">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature android:name="android.hardware.camera2.full" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
```

Fragmento de código 26. Hardware.camera.autofocus

Para hacer uso de la Camera 2 API de Android, se proporcionará un permiso específico para dicha API.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.frogermcs.mnist">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature android:name="android.hardware.camera2.full" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
```

Fragmento de código 27. Hardware.camera2

Finalmente, tanto para guardar imágenes en la galería como para acceder a ellas posteriormente, se necesitarán dos tipos de permisos, "*write external storage*" y "*read external storage*" respectivamente. También se podrá habilitar el uso de Internet en nuestra aplicación.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.frogermcs.mnist">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-feature android:name="android.hardware.camera2.full" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.INTERNET"/>
```

Fragmento de código 28. Permisos galería e Internet

Por otra parte, si lo que preferimos es desarrollar nuestra propia herramienta de captura de fotos con especificaciones más concretas a la hora de la captura, deberemos hacer uso de alguna de las API de cámara que ofrece Android. Conseguiremos así el objetivo de

controlar la cámara, ya sean los parámetros, el formato en el que se toma la imagen, etc. No todos los móviles son aptos a todas las API de cámara por lo que a la hora de hacer uso de cierta API habrá que comprobar que nuestro dispositivo móvil esté adaptado a su uso.

A lo largo de este proyecto, se investigó acerca del uso de diferentes técnicas para conseguir capturar imágenes en el formato requerido y aceptado por el modelo usado a través de la propia aplicación desarrollada. Por ello, comenzaré haciendo una breve descripción acerca de la toma de imágenes y de los posibles métodos existentes.

El más sencillo de explicar sería la captura de fotos delegando el trabajo a la aplicación predeterminada del móvil que estemos usando. Sin embargo, este método permite tomar fotos directamente con la cámara del dispositivo móvil, lo cual, a la hora de modificar o elegir los diferentes formatos de la imagen que se toma, sería algo más tedioso.

Por otra parte, se podrá usar una API como la que en este caso se propone, la *Camera2* API de Android. Con ella, podremos modificar y desarrollar nuestra propia funcionalidad de cámara. Para controlar la cámara como hemos dicho, se requerirá mucho más código que para delegar el trabajo a una aplicación de cámara existente en el dispositivo. A su vez, el objetivo de la mejora será desarrollar una aplicación con cámara y especializada en algo concreto, en nuestro caso, la imagen tomada deberá ser capturada en el formato que el modelo pueda procesar. El modelo usado necesita un *input* de imagen en formato *YUV_420_888* así como sus canales RGB. Posteriormente, esta imagen será procesada para obtener de ella la información correcta que el modelo necesitará como *input* para determinar la salida final, y, por tanto, la imagen resultado.

Una API nos sirve para mejorar la comunicación entre dos softwares, es decir, lo que la API hará será comunicar entre sí dos componentes de software. Entre ambos, podrán intercambiar información. En nuestro caso, y con el uso de una API relacionada con la cámara, podríamos conseguir, por ejemplo, acceder con nuestra app a la captura en sí, así como modificar el formato de la toma de imagen de la cámara que nuestra app utilizará.

Camera 2 API es una interfaz pensada principalmente para desarrolladores. Comenzó a introducirse en los sistemas Android a partir de la versión 5.0. Con ella se pretende facilitar al desarrollador un uso más concreto sobre la cámara y la captura de imágenes. Con esta API se podrá manejar el control de exposición, el enfoque manual, el formato de la imagen, etc.

Es por ello, que esta API de cámara nos ofrece una gran versatilidad a la hora de la toma de fotos. Por otra parte, esta API no tiene por qué estar disponible en todos aquellos

dispositivos con *Android Lollipop* o superior, ya que también dependerá del fabricante del dispositivo móvil, que puede o no incluir el uso de esta API en cuatro niveles distintos.

Estos niveles son *Legacy, Limited, Full y Level 3*. La versión *Legacy* solo admite las funciones de la antigua *Camera1*, la versión *Limited* trae algunas funciones simples; la versión *Full* admite todas las funcionalidades estándar de la *API* y, por último, la versión *Level 3*, que añade funciones extra, como la captura en formato *RAW*. [23]

¿Cómo saber qué nivel de Camera2 API tiene mi teléfono?

Descargando la aplicación *Camera2 Probe* podremos comprobar las especificaciones técnicas del sistema referentes a la cámara que nuestro dispositivo móvil nos ofrece. Con ello, podremos comprobar si podremos hacer uso de esta API y a qué nivel. [24]

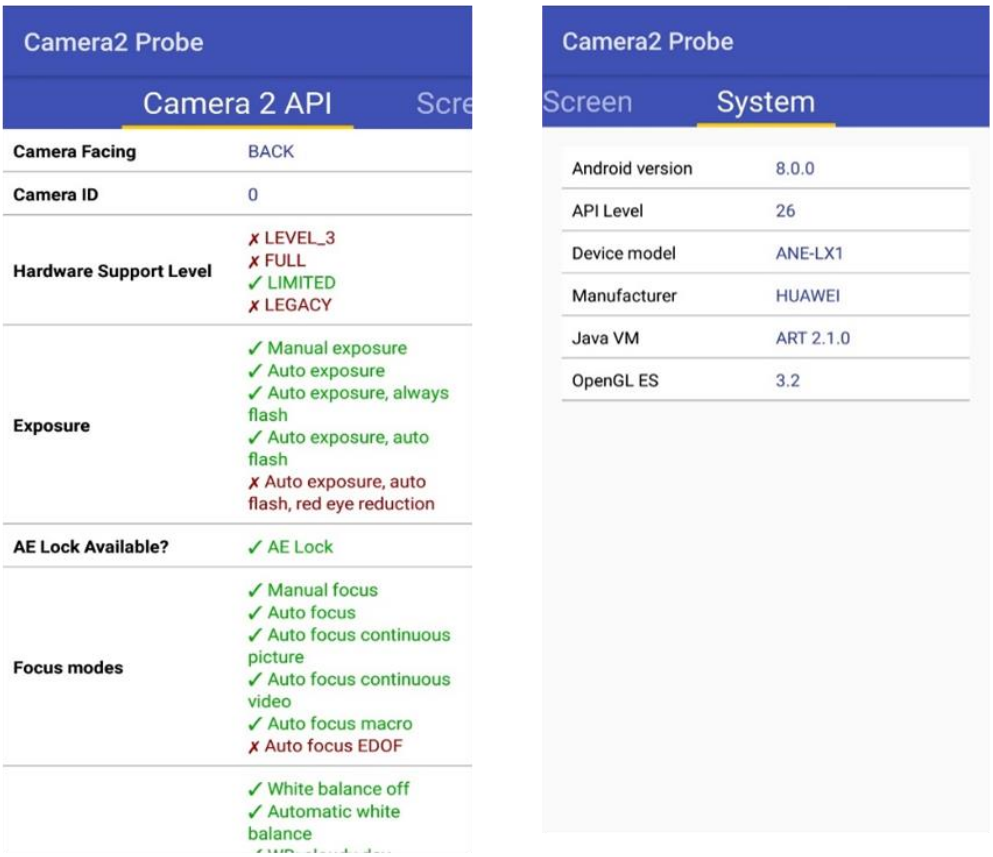


Ilustración 24. Camera2 Probe

Como se explicaba anteriormente, los niveles de hardware que estén soportados por nuestro sistema, harán que las funciones respecto a la API Camera2 varíen notablemente. Tras hacer un análisis de las características del teléfono usado para este proyecto, se puede comprobar en las imágenes cómo solo está disponible la versión Limited. Con ella solo tendremos disponibles algunas funciones simples, por lo que, como bien dice la propia palabra, nos limitará en el uso de esta API. Por otra parte, también podremos obtener información referente al sistema Android que estemos usando actualmente en nuestro dispositivo móvil y la API que tengamos activa en él.

A la hora de integrar la API Camera 2, crearemos 3 nuevos ficheros en Android Studio: *Camera2BasicFragment.java, CameraActivity.java y AutoFitTextureView.java*.

Principalmente, nos centraremos en el archivo *Camera2BasicFragment.java*, en el cual residen todos aquellos conceptos importantes acerca de la toma de imágenes.

Procedimiento de toma de imágenes con Camera2 API.

Partiendo de la interfaz creada, tendremos un botón que nos redirigirá directamente a la parte de la toma de imágenes con *Camera2*. En esta sección, habrá dos botones encargados de dos tareas diferentes.

En primer lugar, la estructura que se propone es la siguiente: tenemos el botón "*picture*", que nos llevará a tomar la imagen, y, por otro lado, el botón "*OK*", que se encargaría del procesado de la imagen, así como de sacar la imagen final reconstruida.

Basándonos en el código del Camera2BasicFragment, encontramos el evento *onClick* sobre ambos botones:

```
@Override
public void onClick(View view) {
    switch (view.getId()) {
        case R.id.picture: {
            takePicture();
         //   onImageAvailable();
            break;
        }
```

Fragmento de código 29. TakePicture( ) - camera2 API

Al pulsar el botón de toma de imagen, éste nos redirigirá a "*takepicture( )*". Con esta función se nos permitirá realizar la imagen deseada. Dentro de la función "*takepicture( )*", encontramos la siguiente función: "*lockFocus( )*"

```
private void takePicture() {
    Log.d("Camera2BasicFragment", "Taking picture");
    lockFocus();
}

/**
 * Lock the focus as the first step for a still image capture.
 */
private void lockFocus() {
    try {
        mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE, CameraMetadata.CONTROL_AF_MODE_AUTO);
        // This is how to tell the camera to lock focus.
        mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_TRIGGER,
                CameraMetadata.CONTROL_AF_TRIGGER_START);
        // Tell #mCaptureCallback to wait for the lock.
        mState = STATE_WAITING_LOCK;
        mCaptureSession.capture(mPreviewRequestBuilder.build(), mCaptureCallback,
                mBackgroundHandler);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }
}
```

Fragmento de código 30. TakePicture ( ), lockFocus ( )

Más adelante comentaré unos de los problemas encontrados con esta función. Nos centramos ahora en la parte de la lectura de la imagen tomada, donde una vez leída y tomada en el formato *YUV_420_888*, la enviaremos a "*ImageUtils.java*" para su procesado.

```
mImageReader = ImageReader.newInstance(largest.getWidth(), largest.getHeight(),
        ImageFormat.YUV_420_888, /*maxImages*/2);
mImageReader.setOnImageAvailableListener(
        mOnImageAvailableListener, mBackgroundHandler);
```

Fragmento de código 31. ImageFormat YUV_420_888

Una vez leída la última imagen mediante la función que aparece anteriormente, pasamos a enviar esta imagen a "*ImageUtils.java*".

```
@Override
public void onImageAvailable(ImageReader reader) {
    mImageReader = ImageReader.newInstance(mImageBigFuckinSize.getWidth(), mImageBigFuckinSize.getHeight(),
            ImageFormat.YUV_420_888, 10);
    mImageReader.setOnImageAvailableListener(this, null);
    Image image = reader.acquireLatestImage();

    if (image != null) {
        //byte[][] ImageData = ImageUtils.prepareImagesForPrediction(image);
        byte[] ImageData1 = MnistClassifier.prepareData(image);
        //  image.close();
    }

    image.close();
}
```

Fragmento de código 32. onImageAvailable

Dentro de "*ImageUtils.java*" será donde tendría lugar el procesamiento de la imagen.

Referencias

Referencias

[1] J. Pastor, «Xataka,» Xataka, 23 Enero 2018. [En línea]. Available: https://www.xataka.com/internet-of-things/edge-computing-que-es-y-por-que-hay-gente-que-piensa-que-es-el-futuro. [Último acceso: 09 10 2019].

[2] U. d. Lleida, «Universidad de Lleida,» 07 04 2020. [En línea]. Available: http://www.grap.udl.cat/es/presentacion/ap.html.

[3] Agtech, «Redagrícola,» Agosto 2019. [En línea]. Available: https://www.redagricola.com/cl/gestiones-terrenos-especificos-utilizando-tecnologias-velocidad-variable/.

[4] «GPSags,» [En línea]. Available: http://www.gpsags.com/resources/gps-faqs.

[5] J. M. Fernández, «Grupo Fertiberia,» Diciembre 2017. [En línea]. Available: https://www.grupofertiberia.com/es/blog/2017/diciembre/agricultura-inteligente-2-agricultura-de-precision/.

[6] «MathWorks,» [En línea]. Available: https://es.mathworks.com/discovery/deep-learning.html.

[7] C. Kozyrkov, «Medium,» 8 Julio 2018. [En línea]. Available: https://medium.com/datos-y-ciencia/la-explicaci%C3%B3n-m%C3%A1s-simple-de-machine-learning-que-jam%C3%A1s-habr%C3%A1s-leido-4fa7ac6243ba.

[8] J. Pedraza, «El Pais,» 21 Abril 2017. [En línea]. Available: https://elpais.com/elpais/2017/04/20/talento_digital/1492704966_190402.html.

[9] J. Pedraza, «El País,» 21 Abril 2017. [En línea]. Available: https://elpais.com/elpais/2017/04/20/talento_digital/1492704966_190402.html.

[10] M. Rouse, «Search Data Center,» Abril 2017. [En línea]. Available: https://searchdatacenter.techtarget.com/es/definicion/Aprendizaje-profundo-deep-learning.

[11] «TicBeat,» 2 12 2014. [En línea]. Available: https://www.ticbeat.com/cloud/que-es-cloud-computing-definicion-concepto-para-neofitos/.

[12] Gradiant, «Gradiant,» 12 04 2018. [En línea]. Available: https://www.gradiant.org/blog/edge-fog-computing-cloud/.

[13] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Android_Studio.

[14] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/IntelliJ_IDEA#Tecnolog%C3%ADas_y_frameworks.

[15] A. Developers, «Android Developers,» [En línea]. Available: https://developer.android.com/studio/intro?hl=es-419.

[16] SGOliver, 20 12 2014. [En línea]. Available: http://www.sgoliver.net/blog/entorno-de-desarrollo-android-android-studio/.

[17] ParadigmaDigital, «Paradigma Digital,» [En línea]. Available: https://www.paradigmadigital.com/lineas-servicio/tensorflow/.

[18] P. Digitales, «Puentes digitales,» [En línea]. Available: https://puentesdigitales.com/2018/02/14/todo-lo-que-necesitas-saber-sobre-tensorflow-la-plataforma-para-inteligencia-artificial-de-google/.

[19] M. Forcén, *Deep Learning in Precision Agriculture: An intelligent system for estimating fruit in lemon trees.*.

[20] M. Forcén, *Deep Learning in Precision Agriculture: An intelligent system for estimating fruit in lemon trees.,* 2019.

[21] «Android Developers,» [En línea]. Available: https://developer.android.com/reference/android/hardware/camera2/package-summary.

[22] «Developers,» [En línea]. Available: https://developer.android.com/training/camera/photobasics?hl=es-419.

[23] M. Hurtado, 21 Noviembre 2017. [En línea]. Available: https://www.tuexpertomovil.com/2017/11/21/que-es-camera2-y-como-saber-si-tu-movil-lo-tiene/.

[24] M. Hurtado, «Tu Experto Móvil,» 21 Noviembre 2017. [En línea]. Available: https://www.tuexpertomovil.com/2017/11/21/que-es-camera2-y-como-saber-si-tu-movil-lo-tiene/.

[25] Na8, «Aprender Machine Learning,» 29 Noviembre 2019. [En línea]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

Figuras

Figuras

Fragmentos de código

Fragmentos de código