Universidad Zaragoza
1542

Trabajo Fin de Grado

Reconocimiento Automático de Patrones en Dispositivos Móviles

Autor

Luis Manuel Muñoz Pérez

Directora

Ana María López Torres

Escuela Universitaria Politécnica de Teruel. Campus de Teruel.

2022

"Reconocimiento Automático de Patrones en Dispositivos Móviles"

Resumen

El presente Trabajo de Fin de Grado consiste en el desarrollo e implementación de una aplicación de identificación de objetos para dispositivos móviles Android que cuenta con la funcionalidad de reconocer automáticamente qué tipo de hortaliza aparece en una imagen.

La aplicación utiliza un modelo previamente entrenado para obtener uno propio mediante la técnica de Transfer-Learning con redes neuronales, y así categorizar qué es lo que aparece en una imagen, de entre aquellos elementos con que ha sido entrenado el modelo final.

Palabras clave: *Aplicación Android, Reconocimiento Automático, Redes Neuronales, Transfer-Learning*.

"Automatic Pattern Recognition on Mobile Devices"

Abstract

The Final Degree Project exposed here consists in the development and implementation of an object identification Android Application able to recognize automatically what kind of vegetable appears in a picture.

The App uses a previously trained model to recognize what shows in the picture from among the eligible elements. This aim is accomplished conducting an adequate Transfer-Learning Neural Network training.

Keywords: *Android Application, Automatic Recognition, Neural Networks, Transfer-Learning*.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. Introducción y Objetivos

El objetivo de este trabajo de fin de grado consiste en la creación de una aplicación Android de reconocimiento de patrones que permita identificar qué tipo de elemento aparece en una imagen de entre un conjunto de clases predefinidas o universo de trabajo.

Si bien se ha enfocado en el reconocimiento de hortalizas, se trata de una de las infinitas posibilidades que nos brinda la implementación de aplicaciones de inteligencia artificial en un dispositivo móvil. Podría adaptarse para realizar una identificación de defectos en un producto, una identificación de futuras avenidas de un río, una identificación de cómo de peligrosa es una especie fotografiada, de la situación climatológica fotografiando las nubes, para conocer el riesgo de mortalidad de una persona en base a una fotografía de su retina o para conocer en qué década se ha realizado una fotografía. No obstante, por mayor simplicidad se ha enfocado en identificar hortalizas, dado que es mucho más sencillo realizar más de 10 000 fotografías de hortalizas que hacerlo de productos manufacturados defectuosos, o de especies peligrosas, o que analizar 10 000 fotografías antiguas o más de 10 000 registros climatológicos.[1][2][3][4]

Con la implementación desarrollada en este trabajo fin de grado será posible identificar entre 8 tipos diferentes de hortalizas: Kiwis, Limones, Manzanas Golden, Naranjas, Peras, Plátanos, Tomates y Zanahorias. Estas han sido seleccionadas en función de su disponibilidad para poder realizar la mayor cantidad de fotografías posible.

Los principales problemas que se abordan en relación con la consecución del presente proyecto son los siguientes:

- Recopilación de una gran cantidad de imágenes que permitan el entrenamiento de un modelo a partir del aprendizaje supervisado.
- Creación de un modelo CNN a partir de un modelo preexistente de reconocimiento de imágenes (Transfer-Learning) que identifique con gran precisión los elementos presentes en las imágenes.
- Implementación del modelo generado con Python (modelo Keras) en Android (modelo TensorFlow Lite).
- Desarrollo de la aplicación Android.

1.1. Contexto

El uso de la Inteligencia Artificial es algo muy extendido ya desde hace décadas. Si bien parece algo novedoso, no lo es en absoluto. No obstante, esta percepción se debe a que en la actualidad se están realizando grandes avances en inteligencia artificial.

Aquellas pruebas con redes neuronales que cuando se plantearon en las décadas de 1980 y 1990 parecían extravagantes y terminaron por dar malos resultados, a día de hoy comienzan a dar

buenos resultados (a veces siendo necesarias millones de imágenes para conseguirlo), y son estos resultados los que promueven el creciente interés en la inteligencia artificial, dado que cada vez más aplicaciones antes imposibles comienzan a hacerse realidad, abriendo un gran abanico de posibilidades para la inteligencia artificial.[5]

No debe confundirse sin embargo lo innovador de las nuevas aplicaciones con la base de conocimientos en que están basadas, ya que la metodología de la mayoría de estas aplicaciones fue definida en las décadas de 1960 y 1970. No terminaron entonces por afianzarse y tuvieron que esperar a tiempos más modernos para que pudiera verse todo su potencial, consolidado ya el método científico en el ámbito la inteligencia artificial y contando con la gran capacidad de procesamiento de los ordenadores actuales.[5]

La construcción de estas aplicaciones viene determinada por las herramientas existentes para su implementación, por esta razón es muy habitual utilizar el lenguaje de programación Python, ya que la mayor parte de las librerías necesarias para Machine Learning han sido creadas para su uso en Python.

Por otro lado, atendiendo a cómo se hace llegar estas implementaciones a los consumidores, hay una innumerable cantidad de posibilidades, si bien hacerlo a través de dispositivos móviles es probablemente la opción más recomendable, pues se consigue llegar a una gran cantidad de usuarios, un rápido acceso a nuestra aplicación, además de poder utilizar las herramientas que proporciona el dispositivo (cámara, grabadora, GPS, sensores de movimiento, acceso a internet).

Android es el sistema operativo de dispositivos móviles por antonomasia, utilizado por más del 88% de los propietarios de smartphones en España, si bien hay que remarcar que en países como Australia o Japón es utilizado por menos del 65% de los usuarios. Es necesario señalar que sea iOS o Android el sistema operativo más utilizado, iOS tiene una comunidad de desarrolladores más reducida, lo que hace de Android la opción preferida entre los desarrolladores.[6]

Con la intención de ayudar al desarrollador para facilitar la publicación de nuevas aplicaciones en su plataforma, desde Google crearon Android Studio en 2014, que viene a sustituir al IDE de Eclipse como entorno de programación habitual de aplicaciones Android.[7]

Es con esta herramienta con la que se desarrolla a día de hoy la gran mayoría de las aplicaciones Android. Estas son cada vez más variadas, y nos permiten contar con funcionalidades muy interesantes en nuestros smartphones, desde leer eBooks, consultar información meteorológica de gran precisión o practicar idiomas fácilmente, hasta realizar observaciones de aves o compartir fotografías de animales con la comunidad científica y descubrir nuevas especies.[8][9][10][11][12]

En el caso concreto de identificación de elementos en una imagen, hay una buena cantidad de aplicaciones móviles cuya funcionalidad es la identificación de animales, monedas, enfermedades de plantas…. Sin embargo, no es mucha la variedad de grupos de elementos que pueden identificarse, por lo que es realmente fácil encontrar un nicho de un grupo de elementos para el que no existe una aplicación que los identifique. Por otro lado, sí que existe alguna aplicación capaz de identificar una elevada cantidad de elementos, englobando buena parte de todos los elementos que conocemos, como Google Lens por ejemplo, que además puede utilizarse para realizar una búsqueda por internet de elementos similares. No obstante, Google Lens no podría ofrecernos información de segundo nivel, como, haciendo referencia a los ejemplos iniciales, como de

peligrosa es una especie, qué tipo de fractura se ha producido en un producto manufacturado o en qué década se hizo una fotografía.[12][13][14][15]

1.2. Preámbulo a las técnicas de aprendizaje automático (Machine Learning)

A grandes rasgos, el aprendizaje automático o Machine Learning consiste en la obtención de un sistema del que se puede extraer una predicción. Esta predicción está basada en el resultado que nos proporciona un modelo a partir de una serie de variables de entrada, que este combina de una manera determinada para ofrecernos un resultado.[16]

En función del tipo de resultado buscado, se puede realizar una clasificación, cuyo objetivo es la asignación de elementos a un conjunto de clases discretas, o una regresión, consistente en la obtención de un determinado valor continuo de salida.[16]

A la hora de construir este modelo a partir de las variables de entrada puede hacerse, en términos generales, bien un aprendizaje supervisado o un aprendizaje no supervisado. En el primer caso, el modelo conoce qué tipo de elementos debe identificar y cuenta con un conjunto de muestras de aprendizaje en las que se conoce el valor a predecir a partir de valores concretos de las variables de entrada. En el segundo, el modelo se centra más bien en técnicas de aprendizaje y debe aprender además qué tipos de elementos existen, ya que no cuenta con un conjunto de muestras de referencia.[5][16]

En lo consiguiente nos referiremos solamente al caso de clasificación con aprendizaje supervisado, que es el caso tratado en el presente trabajo de fin de grado.[16]

El proceso de aprendizaje automático consta de tres fases:

1) Entrenamiento: construcción del modelo en base a un determinado algoritmo de entrenamiento a partir de los datos dispuestos para ello.

2) Validación: comprobación de que el modelo funciona correctamente haciendo uso de un set de datos, de clasificación ya conocida, diferente del utilizado en el entrenamiento.

3) Predicción: se utiliza el modelo generado para realizar la categorización de un set de datos cuya clasificación le es desconocida al sistema a priori.

1.2.1. Entrenamiento y validación del modelo

Para realizar el entrenamiento se debe contar en primer lugar con un conjunto de datos representativo de lo que se quiere modelar que contemple para cada categoría todas aquellas variantes que se desee poder identificar con el modelo. Por esta razón será de gran interés que el conjunto de datos sea lo más grande posible.[16]

Habrá que asegurarse además de que el conjunto de datos esté constituido de manera equitativa y no haya ningún tipo de elemento infrarrepresentado, pues de lo contrario se obtendrían unos malos resultados en la validación, como puede observarse en la Tabla 1, donde se muestra a modo de ejemplo la denominada matriz de confusión y el f1-score que podría resultar de utilizar en el entrenamiento menos muestras para el tipo de elemento "C" que para los tipos de elementos "A" y "B" (los valores mostrados provienen de una clasificación de hortalizas realizada a modo de ejemplo).

Tamaño de la muestra de datos en el entrenamiento: A: 2000 ; B: 2000 ; C: 500		**Tipo de elemento predicho en el proceso de validación**				
		A	B	C		**f1-score**
Tipo de elemento real	A	262	4	0	A	0.92
	B	2	261	1	B	0.76
	C	40	156	68	C	0.41

Tabla 1. Ejemplo de matriz de confusión y f1-score que podrían obtenerse en caso de estar uno de los tipos de elementos infrarrepresentado. Cada fila corresponde a los datos pertenecientes a un tipo de elemento, que podrá ser clasificado en el proceso de validación como el elemento que es (casilla verde) o erróneamente como otro tipo de elemento. Puede observarse que el tipo de elemento "C" está infrarrepresentado en la muestra de datos del entrenamiento, lo que lleva a peores resultados en el f1-score.

Si se corrige la falta de datos del tipo de elemento "C" del ejemplo de la Tabla 1, podrán obtenerse unos resultados mucho mejores para todos los tipos de elementos, como puede observarse en la Tabla 2, en la que se muestra cómo podría resultar la validación en caso de contar en el entrenamiento con 1500 muestras más del tipo de elemento "C".

Tamaño de la muestra de datos en el entrenamiento: A: 2000 ; B: 2000 ; C: 2000		**Tipo de elemento predicho en el proceso de validación**				
		A	B	C		**f1-score**
Tipo de elemento real	A	259	2	5	A	0.98
	B	1	258	5	B	0.97
	C	0	9	255	C	0.96

Tabla 2. Ejemplo de matriz de confusión y f1-score que podrían obtenerse en caso de añadir 1500 muestras del tipo "C" al caso mostrado en la Tabla 1. Cada fila corresponde a los datos pertenecientes a un tipo de elemento, que podrá ser clasificado en el proceso de validación como el elemento que es (casilla verde) o erróneamente como otro tipo de elemento.

La matriz de confusión y el parámetro f1-score, son ejemplos de cómo medir el funcionamiento del modelo tras su entrenamiento. Los valores de f1-score mostrados en las tablas 1 y 2 pueden ser obtenidos según la relación

$$f1\text{-}score_i = \left(\frac{2 \cdot Verdaderos\ Positivos}{2 \cdot Verdaderos\ Positivos + Falsos\ Positivos + Falsos\ Negativos} \right) \quad [1]$$

En esta, los verdaderos positivos son las ocasiones en que la predicción ha resultado exitosa, los falsos negativos son las ocasiones en que se ha predicho que el tipo de elemento objeto de estudio era un tipo de elemento diferente y los falsos positivos son las ocasiones en que otros tipos de elemento han sido predichos como pertenecientes al tipo de elemento objeto de estudio. Se muestra en la Tabla 3 una representación de cómo obtener estos valores de la matriz de confusión.[17]

		Tipo de elemento predicho en el proceso de validación				
		-	-	+	-	-
Tipo de elemento real	-			Falsos Positivos		
	-			Falsos Positivos		
	+	Falsos Negativos	Falsos Negativos	Verdaderos Positivos	Falsos Negativos	Falsos Negativos
	-			Falsos Positivos		
	-			Falsos Positivos		

Tabla 3. Información sobre los valores a utilizar para el cálculo del f1-score a partir de la matriz de confusión. Se muestra el caso correspondiente a la obtención del f1-score del tipo de elemento "**+**" para el caso de un problema de clasificación en 5 clases diferentes.

Antes de comenzar el entrenamiento del modelo, debe realizarse un trabajo de ingeniería de datos para lograr un conjunto de datos adecuado. No solo debe ser un conjunto de datos distribuido equitativamente, como se acaba de explicar, sino que también debe estar igual de diferenciado para cada categoría, dado que una categoría con datos muy similares entre sí también dará como resultado un f1-score menor que el de otras categorías que cuenten con mayor variedad.

Asimismo, el conjunto de datos deberá someterse a un proceso de normalización, que en el caso de tratarse de imágenes supondrá que estas deberán tener las mismas dimensiones dado que se van a realizar diversas operaciones matriciales sobre estas. Lo más adecuado será utilizar imágenes con una relación de aspecto cuadrada (1:1) para facilitar las operaciones matriciales. Si inicialmente las imágenes son alargadas, estas se pueden recortar, deformar o añadir espacio vacío o metamorfoseado con la imagen para conseguir que tengan una relación de aspecto cuadrada.

En caso de deformar las imágenes se necesitará un conjunto de imágenes más extenso para contemplar diferentes deformaciones que puedan producirse al ajustar la imagen. Por otro lado, si se tiene conocimiento sobre lo que aparece en las imágenes o se cuenta con un sistema de detección y extracción de características de la imagen puede optarse por recortar los bordes prescindibles para hacer la imagen cuadrada. Por el contrario, si va a manejarse una extensa cantidad de imágenes, quizá sería más recomendable ampliar la imagen con espacios vacíos para hacerla cuadrada, si bien esto puede afectar a las características de la imagen y provocar efectos negativos en los resultados. Sería más conveniente ampliar la imagen con texturas similares a las que ya se encuentran presentes en esta.

Por supuesto, la mejor opción será que la propia imagen de origen ya tenga una relación de aspecto cuadrada, luego solamente habrá que ampliar o reducir su resolución para ajustarla a las dimensiones objetivo.

Por otro lado deberá realizarse una estructuración de la información para poder acceder a esta de manera sistemática en el proceso de entrenamiento, por lo que se guardará en directorios diferenciados la información perteneciente a cada tipo de elemento.[16]

El proceso de entrenamiento que se realiza a continuación se puede describir esquemáticamente como se muestra en la ilustración 1.

El objetivo del proceso de entrenamiento es obtener un modelo matemático que, a partir de un conjunto de variables de entrada x_i obtenidas de medidas sobre el objeto a identificar produzca una salida o predicción $\tilde{y}_i$ del tipo de objeto. La función encargada de identificar los patrones y transformarlos en una predicción viene definida por un modelo matemático, que depende de un

conjunto de parámetros w_i. En el entrenamiento, se busca los valores w_i que permitan obtener predicciones correctas, los cuales evolucionan según las modificaciones impuestas por el algoritmo de optimización.
Existen diferentes tipos de algoritmos de optimización, que controlan las variaciones en los parámetros w_i con el objetivo de minimizar una función de pérdidas. Esta función, que puede tomar varias formas, está relacionada con el error en la predicción o la diferencia entre la salida real y_i , conocida en las muestras de entrenamiento, y la predicción $\tilde{y}_i$ aportada inicialmente por el modelo.[16]

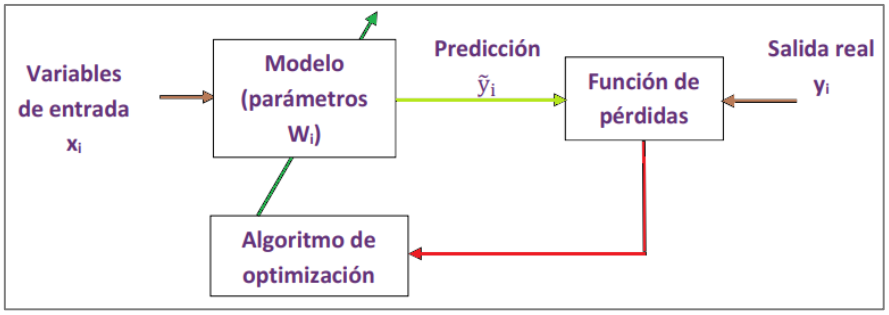


Ilustración 1. Esquema del proceso de entrenamiento.

El objetivo del entrenamiento será modificar los parámetros w_i del modelo hasta el punto de conseguir obtener del modelo la predicción $\tilde{y}_i$ que menor cantidad de error proporcione en la función de pérdidas. En función de una tasa de aprendizaje o Learning Rate, este proceso se realizará con mayor o menor velocidad. Hacerlo muy rápidamente (Learning Rate elevado) tiene como riesgo la posibilidad de no poder alcanzar unos resultados óptimos dado que al aplicar grandes cambios no podrá lograrse un resultado próximo a la capacidad real de predicción. Un valor de Learning Rate de 0.01 o 0.001 suele dar buenos resultados.[16][18]

La función de pérdidas consiste en una medida de optimización, una función de coste. Referido al entrenamiento hablaremos puramente de un proceso de optimización, mientras que referido al proceso de validación podremos hablar ya de aprendizaje automático al no tratarse solamente de cálculo puramente determinista sino también del resultado de la predicción.[19][20][21]

Esta función de pérdidas consiste en el cálculo de la imprecisión de los resultados, que puede realizarse, como se ha mencionado, de diferentes maneras, mediante la función de pérdidas logística, exponencial, tangencial…, o más enfocado en el aprendizaje automático, SVM multiclase, centrado en obtener un f1-score de la categoría correcta mayor a la suma de los demás f1-score, o principalmente cross-entropy, que es la función de pérdidas más comúnmente utilizada, basada en el f1-score multiplicado por 0 o 1 según si es o no correcta la predicción. Hay que notar que la función cross-entropy penaliza gravemente los entrenamientos de gran precisión en términos de optimización cuyo resultado real resulta erróneo, lo que no ocurre con SVM multiclase, dado que en este caso el f1-score puede ser bajo, pero si es muy superior a los demás f1-score, se obtendrá que existe poco error.[19][20][21][22][23]

Por otro lado, para realizar el entrenamiento existen diferentes algoritmos de optimización, como el de descenso de gradiente, basado en la derivada de la función de pérdidas; el de momento, que acelera la convergencia; ADAM, que consiste en momentos de primer y segundo orden adaptativos, etc. De todos ellos, ADAM es considerado el algoritmo de optimización más eficiente.

Dependiendo de cómo se realice la implementación de estos algoritmos, nos encontraremos ante un tipo de modelo determinado. Entre otros, los modelos pueden consistir en árboles de decisión, reglas de asociación para descubrir relaciones interesantes entre variables, algoritmos genéticos que generan mutaciones en búsqueda de mejoras, análisis por agrupamiento que clasifica las observaciones en subgrupos y mide la similitud entre los componentes del subgrupo y los compara con los de otros subgrupos, etc. Si bien nos enfocaremos en el caso de modelos basados en redes neuronales artificiales, las cuales se describen más adelante, dado que este tipo de modelo resultará de mayor interés en el presente trabajo de fin de grado.[23][24]

Una vez realizado el entrenamiento se procede a hacer la validación, que consiste en simular los resultados que se obtendrían de realizar una predicción real con el modelo terminado sobre un conjunto de datos x_i' conocidos pero no utilizado en el entrenamiento. De esta manera se puede conocer el error cometido.
Puede verse un esquema del proceso de validación en la ilustración 2.

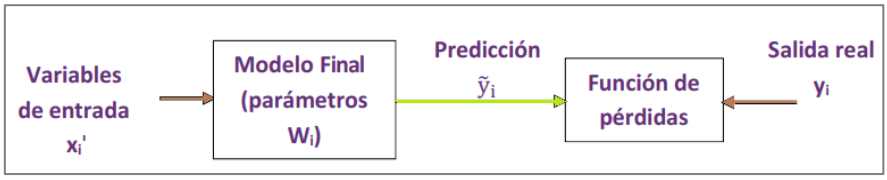


Ilustración 2. Esquema del proceso de validación.

Pueden interpretarse los resultados de la validación bien utilizando las matrices de confusión y los f1-score mencionados anteriormente o bien consultando directamente la precisión media obtenida en la validación según la relación

$$\overline{PR} = \left(\frac{\sum Verdaderos\ Positivos}{\sum Verdaderos\ Positivos + \sum Falsos\ Positivos} \right) \quad [2]$$

En esta, los verdaderos positivos son las ocasiones en que la predicción ha resultado exitosa y los falsos positivos las ocasiones en que ha resultado errónea.[17]

En función del resultado obtenido se decidirá si debe modificarse el conjunto de datos con el que se entrena el modelo o la metodología empleada para realizar el entrenamiento.[22]

No debe cometerse el error de dar por hecho que cuanto más desarrollado o complejo sea el entrenamiento se obtendrá menor error en los resultados de la validación, dado que entrenar el modelo cerca de su máximo nivel de precisión alcanzable puede producir lo que se denomina sobreajuste u Overfitting. Esto ocurre cuando se realiza un entrenamiento con mayor capacidad de la necesaria para realizar la predicción. Cuando esto ocurre, el modelo continúa construyéndose en base a variaciones residuales que no son realmente representativas de lo que se pretende modelar. En consecuencia, se obtienen peores resultados en la validación. Se trataría de modelos muy robustos para el conjunto de datos de entrenamiento, pero con poca capacidad de generalización cuando se enfrenta a muestras desconocidas (ilustración 3).[21][25]

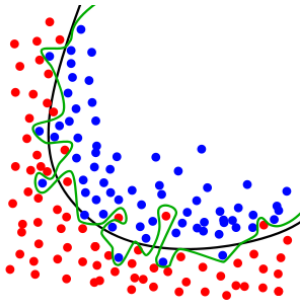


Ilustración 3. La línea verde representa un modelo en el que se produce Overfitting y la negra un modelo regularizado. A pesar de que la línea verde sigue a la perfección los datos de entrenamiento, es muy dependiente de estos, y lo más probable es que se produzca un mayor error que en el modelo regularizado en el caso de que se aplicaran datos nunca antes vistos por el modelo.

1.2.2. Redes Neuronales Artificiales

Una red neuronal artificial es una estructura compuesta por un conjunto de variables de entrada x_i, unidades fundamentales, denominadas neuronas, que una función de activación no lineal φ, transforma en elementos, neuronas, de salida o_i. La combinación lineal de las neuronas de entrada x_i utilizando un conjunto de pesos $\{w_i\}$ es la señal de entrada a la función de activación que produce una salida o_i en función de un umbral de activación θ_i. Puede verse esto más claramente en la ilustración 4.[16][26][27][28]

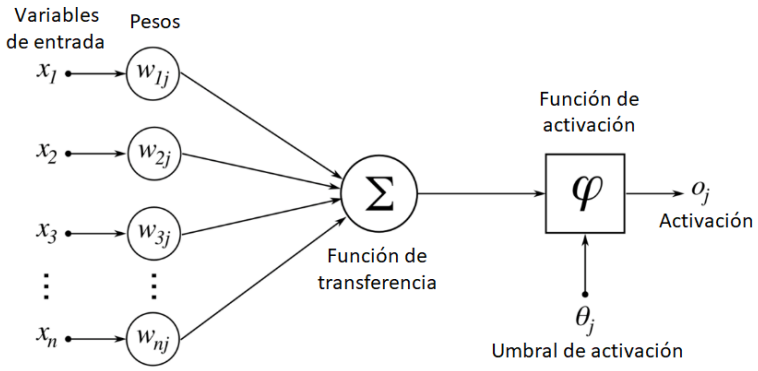


Ilustración 4. Esquema del proceso de activación de las neuronas en una red neuronal.

Las neuronas se agrupan en lo que se denominan capas. En estas capas cada neurona se encuentra conectada a todas las neuronas de las capas adyacentes. En el proceso de entrenamiento se calculan los diferentes pesos w_i que conectan unas neuronas con otras, así como otros parámetros de la función de activación.

Si se utilizan redes neuronales para la clasificación de imágenes esto supondría una elevada cantidad de parámetros a ajustar dada la gran cantidad de datos de entrada, ya que, en este caso, hay tantas variables de entrada como píxeles tiene la imagen, por lo que se utilizará lo que se denomina una red neuronal convolucional (CNN), compuesta principalmente de capas convolucionales. De esta manera las neuronas solo se conectan a un reducido número de neuronas de las capas adyacentes.[16]

Esto se consigue por medio de pequeños kernels de convolución, máscaras discretas que se aplican a los diferentes píxeles (variables de entrada) de la imagen. La convolución de esta máscara en cada una de las posiciones produce una salida que se corresponde con una neurona de la siguiente

capa. Puede verse una representación de esta transformación en la ilustración 5. Se observa como en este tipo de capas, no todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente. Los elementos de los diferentes kernels de convolución, también llamados atributos de la imagen, son parte de los parámetros w_i del modelo que debe ser entrenado. El resultado de la convolución de la imagen con los diferentes kernels se relaciona con la presencia de diferentes elementos en la imagen, como bordes, esquinas o, en capas más avanzadas, patrones. Puede verse un ejemplo de esto en la ilustración 6.[16][29][30]

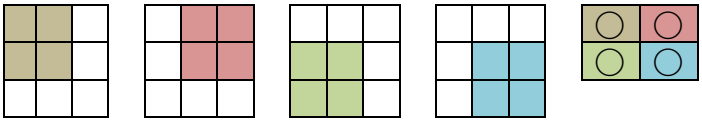


Ilustración 5. Muestra de la aplicación de una máscara de convolución de tamaño 2X2 aplicada a 4 posiciones de una imagen de 9 píxeles. A la derecha se muestran las neuronas resultantes, cada una marcada con el color de la máscara a partir de la cual se han formado.

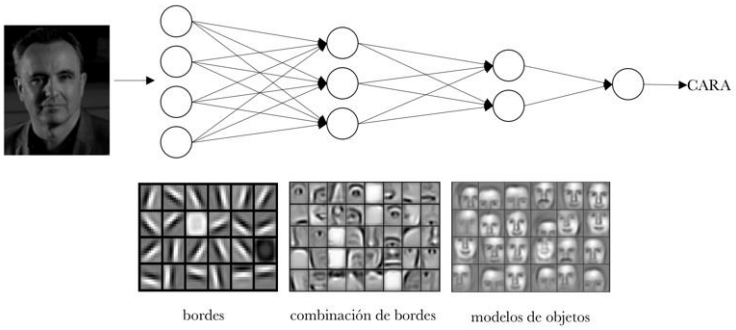


Ilustración 6. Ejemplo de cómo la combinación de bordes obtenidos en los kernels de las capas iniciales puede resultar en patrones identificables en capas posteriores.

Además de capas convolucionales, este tipo de redes neuronales puede incluir en su estructura otro tipo de capas. Entre ellas, cabe destacar las capas totalmente conectadas (dense), que se utilizan en la clasificación final y son alimentadas por características de alto nivel; las capas de rectificación, que se intercalan entre capas de convolución para introducir un grado de no linealidad que mejora el entrenamiento del modelo; las capas de eliminación (dropout), utilizadas cuando se produce Overfitting, que anulan aleatoriamente algunas entradas para que el modelo no se adapte en exceso; o las capas de agrupamiento (pooling), para seleccionar las características más relevantes y frecuentes de cada zona de la imagen según valores mínimos, medios o máximos y que sirven para reducir la dimensionalidad de manera rápida (se muestra en la ilustración 7 un ejemplo).[16][31]

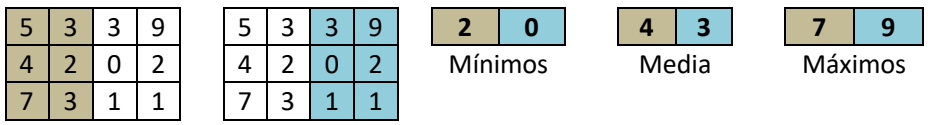
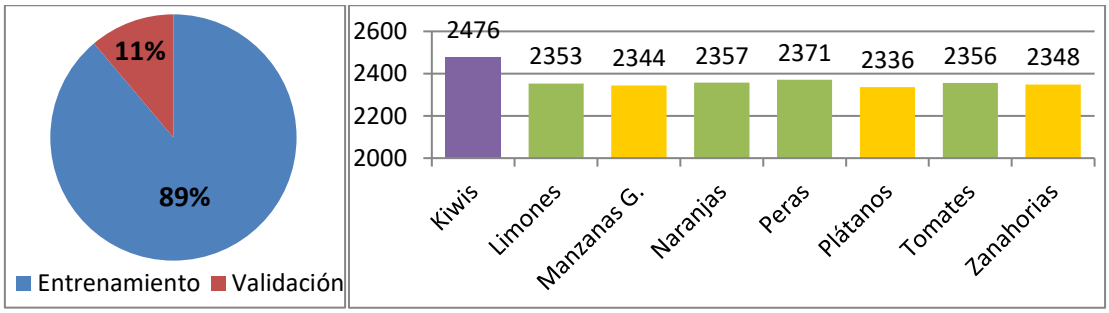


Ilustración 7. Ejemplo de uso de las capas de agrupamiento para el caso de obtención de valores mínimos, medios y máximos.

Para obtener mejores resultados en el entrenamiento y hacer el proceso menos costoso computacionalmente se realiza lo que se denomina Transfer-Learning, que consiste en, tomar un modelo existente de un problema similar y sustituir las últimas capas por nuevas capas finales, las cuales entrenaremos, aprovechando de esta manera las características de bajo nivel ya definidas en el modelo conocido.[16]

2. Trabajo realizado en materia de ingeniería de datos

En este trabajo se ha contado con 18 941 imágenes adecuadas para el entrenamiento y validación del modelo, todas ellas cuadradas, de las cuales 16 826 se utilizarán para el entrenamiento propiamente dicho y 2115 se utilizarán para el proceso de validación (ilustraciones 8 y 9).



Ilustraciones 8 y 9. Izquierda: Porcentaje de imágenes utilizadas para los procesos de Entrenamiento y Validación. Derecha: Número de imágenes utilizadas para cada tipo de hortaliza, con una desviación estándar entre ellas del 1.9%.

A continuación se expone el proceso de recopilación y filtrado de estas imágenes, que se expone de manera ilustrativa en la ilustración 10.

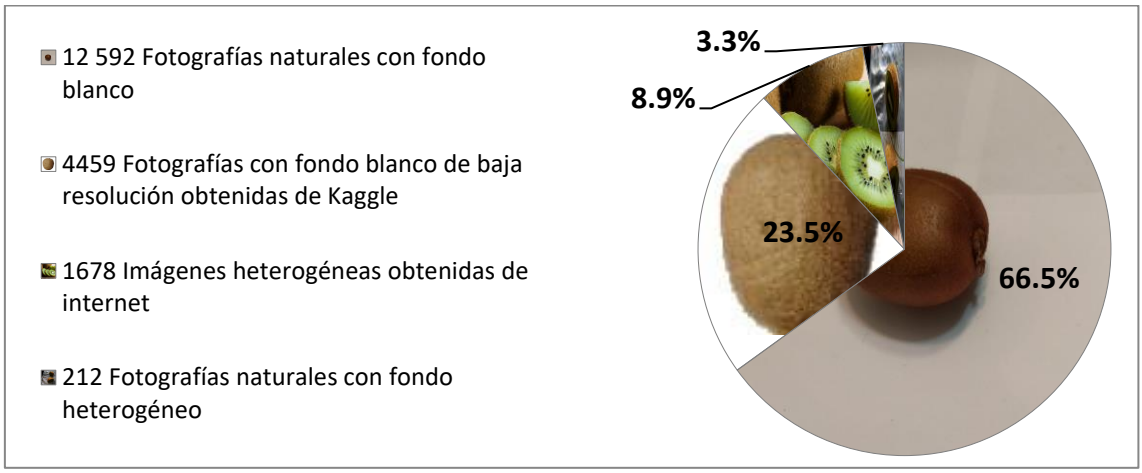


Ilustración 10. Diferentes tipos de imágenes recopiladas y su proporción sobre el total.

2.1. Fotografías naturales con fondo blanco

En primer lugar, se han realizado fotografías de cada tipo de hortaliza con el objetivo de conseguir una rica base de datos. Para cada tipo de hortaliza se han realizado 1494 fotografías (1400 para entrenamiento y 94 para validación) en un set de fotografía.

De las 1494 fotografías de cada tipo de hortaliza, 754 se han hecho fotografiando una única unidad (700 para entrenamiento y 54 para validación) y 740 fotografías se han hecho fotografiando múltiples unidades: 320 con dos unidades (300 para entrenamiento y 20 para validación), 210 con tres unidades (200 para entrenamiento y 10 para validación) y 210 con cuatro unidades (200 para entrenamiento y 10 para validación). Puede verse esto de manera ilustrativa en la ilustración 11.

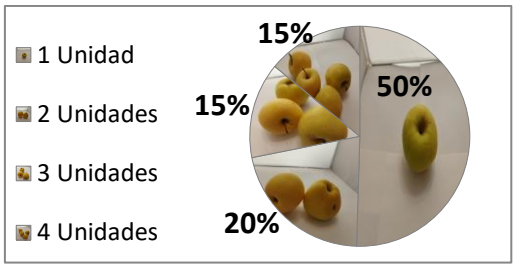


Ilustración 11. Proporción de los diferentes tipos de fotografías naturales con fondo blanco recopiladas.

A estas fotografías habría que añadir 640 fotografías más de Zanahorias realizadas para suplir las imágenes que se deberían haber obtenido de la base de datos de Kaggle, que carece de imágenes de Zanahorias.

En algunos casos, se han realizado fotografías de hortalizas con notables diferencias (Tabla 4):

		Nº Fotografías Individuales	Porcentaje
Limones	Con forma redondeada	280 (+22 de validación)	40%
	Con forma más alargada	280 (+22 de validación)	40%
	Deformes	140 (+10 de validación)	20%
Manzanas Golden	**Más verdes y deformes**	**400** (+27 de validación)	**57%**
	Más amarillas y redondas	300 (+27 de validación)	43%
Plátanos	**Maduros**	**370** (+30 de validación)	**53%**
	Verdes	330 (+24 de validación)	47%
Tomates	**Con sépalos**	**400** (+27 de validación)	**57%**
	Sin sépalos	300 (+27 de validación)	43%
Zanahorias	**De unos 15cm, normales**	**970** (+167 de validación)	**83%**
	De menos de 10cm de largo y delgadas	200 (+57 de validación)	17%

Tabla 4. Variaciones existentes entre un mismo tipo de hortaliza y cuantas fotografías de una única unidad se les ha realizado.

2.2. Fotografías con fondo blanco de baja resolución obtenidas de Kaggle

Se añaden para cada tipo de hortaliza en torno a 640 fotografías (480 para entrenamiento y 160 para validación) de 100X100 píxeles del repositorio “Fruits” de Kaggle para enriquecer el set de imágenes. Kaggle es una plataforma donde se comparten sets de datos, entre ellos imágenes, destinados para ser utilizados en implementaciones de aprendizaje automático.[32]

Como el repositorio de Kaggle no cuenta con imágenes de Zanahorias ya que está más enfocado en las frutas, realizamos, como se ha mencionado anteriormente, más fotografías adicionales de Zanahorias con un fondo y unos ángulos similares a los de Kaggle. Si no se añadieran estas imágenes adicionales se obtendría en el entrenamiento un f1-score mucho más bajo en las Zanahorias, esto es, menos precisión en su predicción con respecto al resto de hortalizas.[17]

2.3. Imágenes heterogéneas obtenidas de internet

Resulta de gran interés añadir a la base de datos imágenes obtenidas de internet. Estas tienen fondos heterogéneos en muchas ocasiones, muestran las hortalizas en diferentes cantidades y

formas y, sobre todo, muestran las hortalizas cortadas o a personas manipulándolas o cortándolas. Se trata de imágenes extravagantes, pero que aportan mucha información de cara al entrenamiento. Si no fuera por ellas, si se intentara identificar por ejemplo un Kiwi cortado, probablemente se obtendría cualquier resultado menos el de un Kiwi.

De esta manera puede considerarse que son mucho más valiosas estas imágenes obtenidas de internet que las fotografías de Kaggle por ejemplo, que son todas ellas muy similares entre sí, no contemplando ángulos diferentes, distintos ejemplares de fruta, diferentes fondos, etc.

Para la búsqueda de imágenes se ha utilizado el buscador avanzado de imágenes de Google. Se ha indicado buscar el nombre en singular y en plural de cada hortaliza en los idiomas español e inglés, se ha seleccionado mostrar solamente las imágenes cuadradas, y que se muestren mayoritariamente imágenes de dominio público (luego se comparan con las que tienen licencia para asegurarse de que realmente son de dominio público). Por último, se han realizado dos búsquedas diferentes, una de imágenes en formato jpg y otra de imágenes en formato png. Esto se ha hecho porque independientemente del formato que se busque, se mostrarán de 550 a 750 imágenes, por lo que de esta manera se aumenta al doble la cantidad de imágenes recopiladas. En el caso de los Tomates se ha obtenido un único set de imágenes, dado que más del 35% de las imágenes resultaban adecuadas y sumaban una cantidad de imágenes superior a las de otras hortalizas que contaban ya con los dos sets.[33]

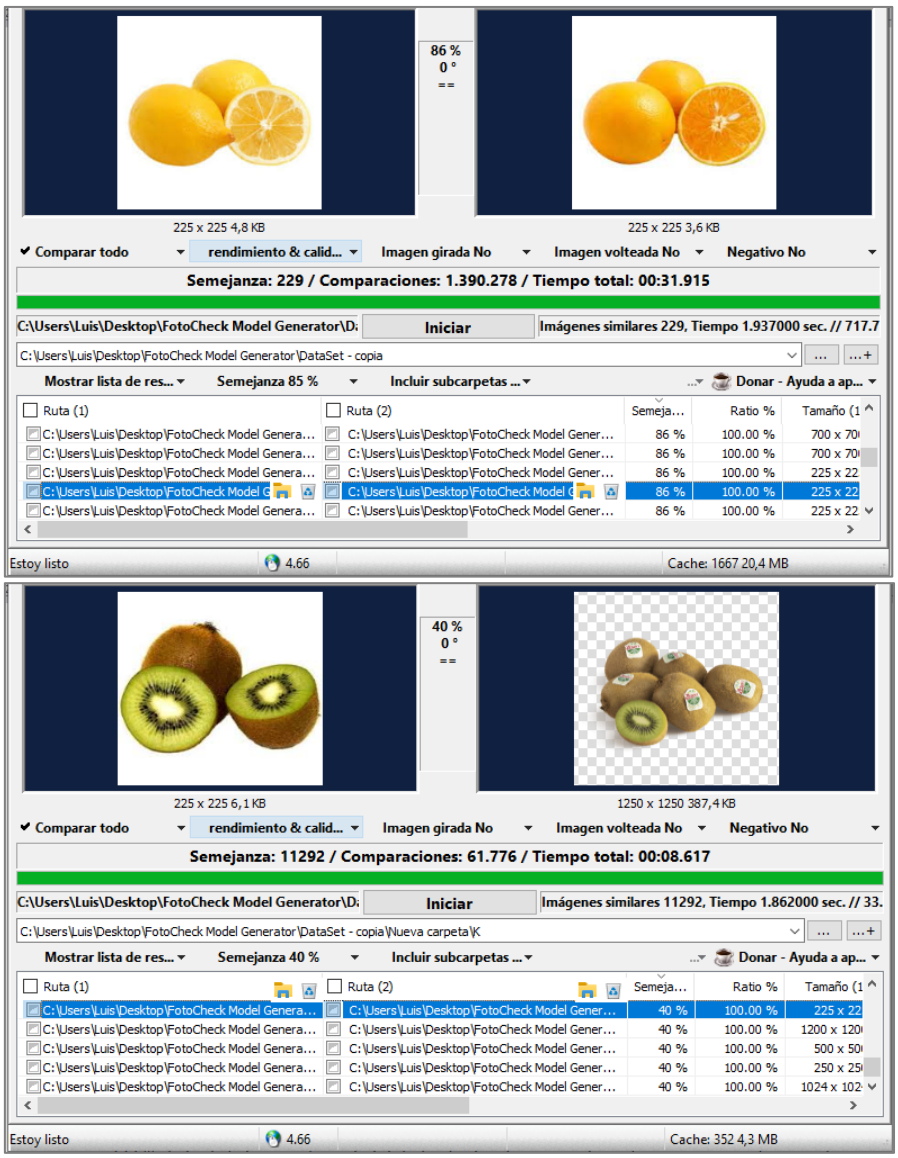
Para realizar el guardado de las imágenes se ha utilizado la extensión de Google Chrome "Download All Images", mediante la cual se descargan las más de 500 imágenes que muestra la página en un archivo Zip.[34]

Posteriormente se comprueban todas las imágenes para eliminar aquellas que no muestran hortalizas o que muestran una variedad diferente de estas, así como las imágenes no aptas para el entrenamiento del modelo, como aquellas con partes transparentes y las que no son cuadradas. Una vez realizado esto, nos queda una media del 21.6% de las imágenes descargadas $\left(\frac{2103}{15\ sets\ de\ imágenes \cdot (550\sim750=650)}\right)$.

Después, con el software "FindSameImages", se borra el mayor número posible de imágenes repetidas (es muy habitual encontrar numerosas imágenes repetidas en este tipo de búsquedas). Aprovechamos la ocasión para borrar también imágenes transparentes que no se hubieran detectado con anterioridad, y nos aseguramos de que no aparezca ningún aviso por consola en el entorno de desarrollo Python de que todavía quedan imágenes transparentes (ilustración 12), de lo contrario se buscarán manualmente para eliminarlas. Puede verse una captura del programa "FindSameImages" durante el proceso de borrado de imágenes en las ilustraciones 13 y 14.[35]

```
Epoch 1/5
 13/358 [>.............................] - ETA: 35:19 - loss:
1.9817 - acc: 0.2733  C:\ProgramData\Anaconda3\lib\site-
packages\PIL\Image.py:951: UserWarning: Palette images with
Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
 21/358 [>.............................] - ETA: 34:23 - loss:
1.7689 - acc: 0.3688
```

Ilustración 12. Aviso que se muestra por consola cuando se detecta una imagen transparente en el set de datos durante el entrenamiento del modelo. Resaltado en amarillo.



Ilustraciones 13 (arriba) y 14 (abajo). Arriba: Proceso de borrado de imágenes repetidas utilizando "FindSameImages", se muestra una comparación entre una imagen de Limones y otra imagen de Naranjas que parecen casi idénticas. Abajo: Detección de imágenes transparentes utilizando dicho programa (en la imagen de la derecha puede verse la textura de cuadrados que indica que la imagen contiene transparencias).

Este borrado de imágenes supone la eliminación de 499 imágenes, lo que significa que, de todas las imágenes descargadas, solamente se mantiene el 16.5% $\left(\frac{1604}{15 \cdot (550 \sim 750 = 650)}\right)$.

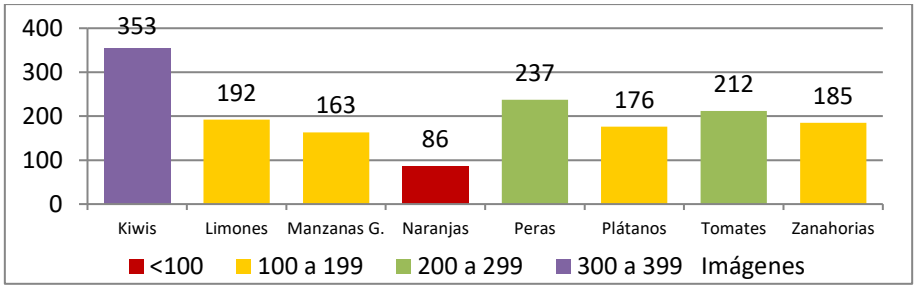
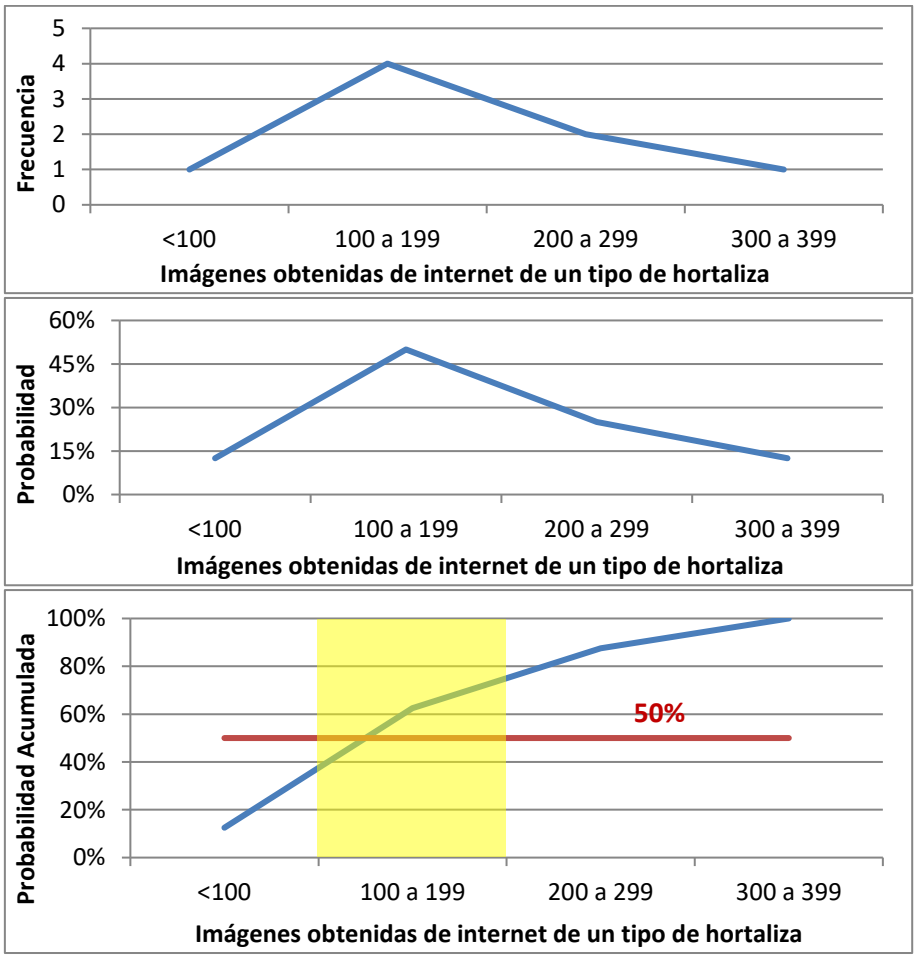


Ilustración 15. Número de imágenes restantes obtenidas de internet por tipo de hortaliza.

Como puede observarse en la ilustración 15, hay hortalizas que cuentan con un reducido número de imágenes obtenidas de internet en comparación con otras hortalizas. Dada la importancia de utilizar en el entrenamiento sets de datos equilibrados entre las diferentes clases, utilizando el mismo procedimiento expuesto anteriormente (buscando en esta ocasión el nombre de la hortaliza en otros idiomas) se incrementará la cantidad de imágenes de las categorías menos favorecidas, de tal manera que todos los tipos de hortaliza tengan por lo menos una cantidad de imágenes obtenidas de internet correspondiente a la sección que intersecta el 50% de la distribución de probabilidad acumulada, esto es, que las categorías menos favorecidas logren entrar en el rango de imágenes al que pertenece la categoría que representa el 50% de la distribución de probabilidad acumulada (ilustraciones 16, 17 y 18):



Ilustraciones 16, 17 y 18. Superior: Distribución de frecuencia de cada rango de Número de imágenes obtenidas de internet. Central: Distribución de probabilidad de cada rango de Número de imágenes obtenidas de internet. Inferior: Probabilidad acumulada de cada rango de Número de Imágenes obtenidas de internet. En rojo la marca del 50%, que corta la sección de "100 a 199 imágenes".

De esta manera, se debería lograr que el número de imágenes obtenidas de internet de Naranjas se encuentre en el intervalo 100-199 de imágenes (la carencia de imágenes de naranjas se debe probablemente a que el término "Naranja" se utiliza relativamente poco para referirse a la fruta en comparación con otras significaciones, por lo que solo el 14% de las imágenes obtenidas resultan adecuadas).

Una vez recopilados y filtrados 2 nuevos sets de imágenes de Naranjas se contará con 160 imágenes obtenidas de internet de Naranjas (ilustración 19):

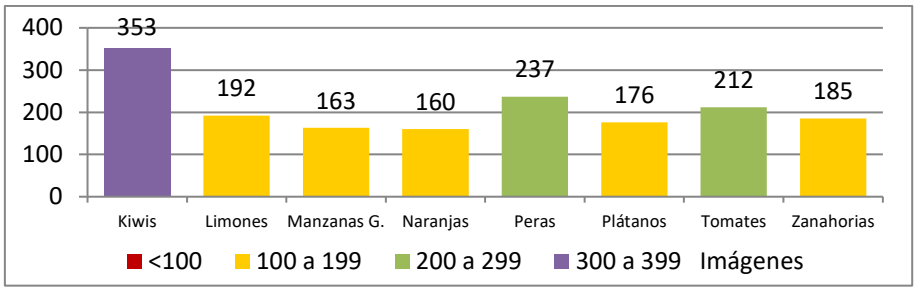


Ilustración 19. Número de imágenes restantes obtenidas de internet por tipo de hortaliza, añadidas las nuevas imágenes de Naranjas.

Finalmente, se destinará el 5% de las imágenes finales de cada tipo de hortaliza para validación (en torno a 83 imágenes en total).

2.4. Fotografías naturales con fondo heterogéneo

Por último, se añaden algunas fotografías adicionales de hortalizas con fondo heterogéneo para que el set de imágenes sea más variado (Tabla 5). Nos centraremos principalmente en los tipos de hortaliza con menor número de imágenes recopiladas de internet, dado que hasta ahora solo las imágenes de internet representan todas las imágenes de fondo heterogéneo. De esta manera tendremos un conjunto de imágenes más variadas y el modelo entrenado ofrecerá mejores resultados.

Kiwis	**Limones**	**Manzanas Golden**	**Naranjas**
9	27	47	64
Peras	**Plátanos**	**Tomates**	**Zanahorias**
N/a	26	10	29

Tabla 5. Fotografías de hortalizas con fondo heterogéneo realizadas para cada tipo de hortaliza. Resaltados en amarillo los tipos de hortaliza con menor número de imágenes recopiladas de internet.

Al igual que con las imágenes obtenidas de internet, se utilizará el 5% de las imágenes para el proceso de validación.

3. Entrenamiento del modelo CNN utilizando la técnica de Transfer-Learning

Para realizar el entrenamiento del modelo CNN se utiliza el lenguaje de programación Python 3.8 y el entorno de desarrollo Spyder (una de las herramientas que ofrece Anaconda).[36]

Hay que destacar que el entrenamiento se realiza solamente entre los 8 tipos diferentes de hortaliza mencionados anteriormente. Esto es algo problemático si se pretende identificar cualquier otra cosa que no sea una de estas hortalizas, y es que las imágenes de cada tipo de hortaliza cuentan con una región de características particular. Estas regiones pueden representarse mediante diagramas de Voronoi en base a los vectores de atributos de la imagen. Si se intenta identificar un tipo de elemento diferente cuyos vectores de atributos se encontrasen justo en mitad de una de las regiones ya definidas, nos encontraríamos con que las imágenes pertenecientes a este nuevo tipo de elemento serían clasificadas como del tipo de elemento definido en dicha región. Puede verse un ejemplo en la ilustración 20. Este se ha creado a modo de ejemplo en base a un diagrama de Voronoi generado con el buscador WolframAlpha (https://www.wolframalpha.com/).[37]

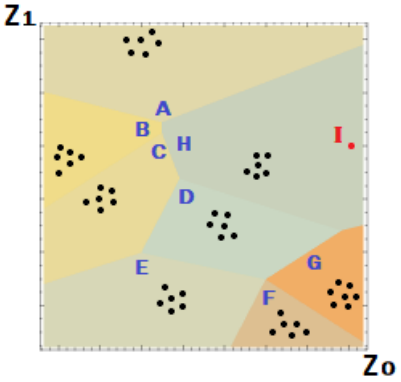


Ilustración 20. Representación de los diferentes vectores de atributos (puntos negros) de cada tipo de elemento en función del valor de los atributos $\{z_0, z_1\}$. Se pueden distinguir las regiones "A", "B", "C", "D", "E", "F", "G" y "H" correspondientes a los diferentes tipos de elementos identificables. Por otro lado, puede verse un vector de atributos (punto rojo) perteneciente a un tipo de elemento no definido en el conjunto (tipo "I"), que será identificado equívocamente como perteneciente al tipo "H".

Se podría pensar que una solución a este problema sería añadir un tipo de elemento "no hortaliza" a modo de cajón de sastre que incluyera espacios vacíos y otros elementos que no vayan a identificarse, pero hacer algo así resultaría perjudicial para el modelo y se necesitaría además una gran cantidad de imágenes. El mayor problema que presenta este planteamiento es que pueda ocurrir justo lo contrario a lo que se pretende, esto es, que imágenes en las que aparece una de las hortalizas objetivo se clasifiquen como "no hortaliza", y es que si se diera el caso, por ejemplo, de añadir a esta categoría adicional una imagen con un fondo similar al fondo de una fotografía en la que sí aparece una de las hortalizas objetivo, la imagen podría ser erróneamente clasificada como "no hortaliza" (ilustraciones 21 y 22).


Ilustraciones 21 y 22. Si la imagen de la izquierda se guardara como tipo de elemento “no hortaliza”, si se intentase luego identificar la imagen de la derecha, podría obtenerse como resultado que es de tipo “no hortaliza” en lugar de Zanahoria.

3.1. Arquitectura del programa Python: entrenamiento y la validación del modelo

Para realizar el entrenamiento del modelo, en primer lugar se cargarán las dependencias de Keras necesarias para trabajar con modelos de redes neuronales, además de otras librerías como Pyplot para mostrar las gráficas con los resultados y Numpy y Sklearn que permiten calcular y mostrar la matriz de confusión y el f1-score (Ilustración 23).

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as pyplot
import sklearn.metrics
from tensorflow.keras import backend
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Ilustración 23. Librerías importadas para posibilitar el entrenamiento del modelo.

Una vez hecho esto se define una fase previa al entrenamiento en la que se pide una confirmación para iniciar la operación (Ilustración 24). Esta confirmación sirve para que quien entrena el modelo sea plenamente consciente de que se está iniciando el proceso, y para no comenzar el entrenamiento por error, lo que podría llevar a la pérdida del modelo generado con anterioridad al sobrescribirse este una vez terminado el entrenamiento.

```
print("INTRODUZCA CERO PARA COMENZAR: ")
op=input()

if op=='0':
    entrenamiento('HORTALIZAS')
```

Ilustración 24. Se pide una confirmación para iniciar el entrenamiento.

En el programa de entrenamiento se define la estructura del modelo que se desea optimizar. Se comenzará por obtener el modelo base (que llamamos “MUB”) a partir del cual se utilizará la técnica de Transfer-Learning (ilustración 25). Se usará el modelo MobileNet (https://arxiv.org/pdf/1704.04861.pdf) porque es el tipo de modelo recomendado para la clasificación de imágenes en aplicaciones portátiles, es eficiente y ligero. El parámetro “weights”, en el que se debe indicar “imagenet”, es simplemente para indicar que va a utilizarse el modelo pre entrenado con la base de datos de imágenes con dicho nombre.
El parámetro “input_shape” es lo que se llama dimensiones objetivo de la imagen. Es un parámetro importarte ya que todas las imágenes de entrada se adaptarán a esas dimensiones. MobileNet nos permitirá trabajar solamente con imágenes de dimensiones (en píxeles) 128X128, 160X160, 192X192 o 224X224. Se utilizará un tamaño de 224X224 ya que es el que permitirá incluir mayor cantidad de detalles.

Por último, indicando False en "include_top", se descarta la capa de los 1000 últimos valores. Estos últimos valores son el número de diferentes opciones de clasificación del modelo. Se sustituirá por nuestra capa de 8 diferentes tipos de hortaliza y se entrenará solo esta capa, junto con una capa de agrupamiento global (las capas anteriores ya se encuentran pre entrenadas).[38][39][40][41]

```
#Modelo Universal de Base (MobileNet)
MUB=tf.keras.applications.MobileNetV2(weights="imagenet",input_shape = (224,224,3)
                                      ,include_top=False)
```

Ilustración 25. Se carga un modelo MobileNet pre entrenado.

A continuación se guarda como "MOut" la salida, o neuronas de salida, del modelo base, para añadirle posteriormente nuestras propias capas finales (ilustración 26).[40]

```
MOut=MUB.output
```

Ilustración 26. Obtención de la salida del modelo base.

Se añade una capa de agrupamiento basada en la media global (se trata de una capa de agrupamiento que no considera los datos por zonas sino en su totalidad, véase a modo de ejemplo la ilustración 27), que bien puede sustituir la capa totalmente conectada (dense) que se suele utilizar previa a la clasificación final. Esta capa de agrupamiento global es comúnmente utilizada con este propósito, dado que de esta manera se realiza el ajuste final de una manera natural al no tener que convertir una capa convolucional en una totalmente conectada utilizando una capa flatten, obteniendo así un mejor rendimiento. Además se reduce el Overfitting al no tener que aprender ningún parámetro en dicha capa de agrupamiento (ilustración 28).[31][40][42]

4	3	1	5
1	3	4	8
4	5	4	3
6	5	9	4

Avg([4, 3, 1, 5] [1, 3, 4, 8] [4, 5, 4, 3] [6, 5, 9, 4]) = 4.3125

Ilustración 27. Ejemplo de la aplicación de una capa de agrupamiento basada en la media global a una capa de tamaño 4X4.

```
MOut=tf.keras.layers.GlobalAveragePooling2D()(MOut)
```

Ilustración 28. Adición de una capa de agrupamiento basada en la media global.

Por último, se crea la capa totalmente conectada de clasificación final, guardada como "UltimProb", utilizando la función de activación Softmax. Una función de activación Softmax sirve explícitamente para obtener la "probabilidad" final (no es realmente probabilidad sino pertenencia) de que la imagen se corresponda con cada una de las clases. Esta capa final obtiene la información de "MOut", esto es, de los resultados obtenidos en la capa de agrupamiento global definida anteriormente. Se indica un tamaño de la capa final de clasificación de 8, que es el número de clases entre las que se realiza la clasificación de hortalizas (ilustración 29).[40][43][44]

```
UltimProb=tf.keras.layers.Dense(8,activation='softmax')(MOut)
```

Ilustración 29. Adición de la capa de clasificación final.

A partir de la estructura definida en este punto, se crea ahora el modelo que va a entrenarse, al que se le indica que los valores de entrada son los del modelo pre entrenado y que los valores de salida son los de la capa totalmente conectada de clasificación final (ilustración 30).[40]

```
#Definimos el modelo indicándole las Entradas y Salidas
model=tf.keras.Model(inputs=MUB.input,outputs=UltimProb)
```

Ilustración 30. Creación del modelo a entrenar.

Debe configurarse además qué capas deben entrenarse y cuáles no (ilustración 31). Para ello se establecen como no entrenables las capas desde el inicio hasta el número de capas del modelo pre entrenado (cuenta con 154 capas), el cual, como se ha mencionado anteriormente, no incluye la capa de clasificación final, y se establecen como entrenables el resto de capas en adelante, las nuevas capas.[40]

```
for layer in MUB.layers[:len(MUB.layers)]: #Indicamos las capas que no entrenameros (todas las del modelo base)
    layer.trainable=False
for layer in MUB.layers[len(MUB.layers):]: #Indicamos que entrenaremos solo las nuevas capas
    layer.trainable=True
```

Ilustración 31. Se configura qué capas se entrenarán y cuáles no durante el entrenamiento.

Se realiza lo configuración del modelo creado (ilustración 32). Se utilizará "categorical_crossentropy" como función de error dado que, como se ha explicado en la introducción, se obtendrán mejores resultados en la validación.
Por otro lado, se utilizará el algoritmo de optimización "ADAM", porque, como recordamos, es el más eficiente de los algoritmos de optimización.[16]

```
model.compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['acc'])
```

Ilustración 32. Se configura el modelo.

Se preparan los sets de datos para el entrenamiento y para la validación.
Para ello, se definen unos generadores de datos (ilustración 33), preparados para convertir la información de las imágenes consistente en valores enteros de 0 a 255, en números decimales con valores de 0 a 1 (valores normalizados).[16][45]

```
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)
```

Ilustración 33. Preparación de los sets de datos para el entrenamiento y la validación.

Antes de entrar más en detalle convendrá hacer un pequeño inciso para explicar tres de los parámetros que controlan el proceso de entrenamiento: época, paso, y el parámetro "batch_size". Una época es cuando un set de datos completo es procesado por la red neuronal una sola vez. Para obtener mejores resultados se ejecutan varias épocas ya que de esta manera se incrementa el número de veces que se adaptan los parámetros del modelo. Los pasos son las partes en las que se divide el entrenamiento (la época), en cada una de las cuales se procesa un conjunto diferente de imágenes del set de datos. Los parámetros del modelo no se actualizan con cada imagen procesada sino en cada uno de estos pasos.
Por otro lado, "batch_size" es el número de imágenes con las que se trabaja en cada paso del entrenamiento.[16][46][47]

Se realiza el traspaso de información de los directorios al set de datos, como puede verse en la ilustración 34. El parámetro "cod", que atendiendo a la ilustración 24, tiene valor "HORTALIZAS" es utilizado para conocer en qué directorio debe buscarse la información.
Por otro lado, se indica un tamaño objetivo de la imagen ("target_size") de 224X224, el mismo indicado en el modelo base.
En cuanto al "batch_size", deberá elegirse un valor intermedio, entre 1 y todas las imágenes del set de datos de entrenamiento. Si se toma un valor de "batch_size" elevado, sería recomendable aumentar asimismo el Learning Rate para mejorar los resultados de validación.[16][47][48][49]

Para tener un "batch_size" del mismo valor en todos los pasos de cada época del entrenamiento, deberá cumplirse la relación

$$batch\_size = \frac{\text{Número total de imágenes del set de datos de entrenamiento}}{\text{Número de pasos de cada época}} \quad [3]$$

Sería de nuestro interés que esta condición se cumpliera, dado que las operaciones matriciales se realizarán con mayor sencillez. Es además una condición imprescindible en el caso de que se utilicen capas dropout, dado que su parámetro "noise_shape" (una máscara del mismo tamaño que "batch_size") debe tener una dimensión constante.
El uso de capas de eliminación (dropout), como ya se ha mencionado, suele ser necesario en casos de sobreajuste, algo que no obstante, no ocurre en nuestro caso, pero de todas maneras cumpliremos esta condición para una mayor naturalidad en las operaciones.[50][51]

Por otro lado, se utilizará el mismo "batch_size" tanto en el entrenamiento como en la validación para que las operaciones se realicen de manera similar, con matrices de la misma cantidad de imágenes. De esta manera se buscará un valor de "batch_size" que sea divisor tanto del Número de Imágenes de Entrenamiento, como del Número de Imágenes de Validación, y que además tenga un valor más o menos elevado para obtener un resultado estable con pocas oscilaciones.[48]

Para ello se ha realizado un pequeño ajuste descartando 2 imágenes del set de datos de validación. De esta manera se cuenta con 16 826 imágenes de entrenamiento y 2115 imágenes de validación, y podrá utilizarse un "batch_size" de valor 47, que consideramos adecuado en base al tamaño de la muestra de datos.

Con "class_mode" simplemente se indica si es una clasificación binaria o múltiple y en "color_mode" se indica que son imágenes a color representadas por matrices 224X224 X 3 canales de color: rojo, verde y azul.

El parámetro "shuffle" sirve para ordenar de manera aleatoria las imágenes de un set de datos. Si se indica "shuffle=False" las imágenes serán ordenadas por orden alfanumérico, lo que supone que se procesarán imágenes de nombre similar en cada paso del entrenamiento.[52][53]

Se indica, por último, "shuffle=False" para no cambiar el orden de las etiquetas y poder construir así la matriz de confusión, si bien solo se hará esto para poder ver la matriz de confusión, dado que los resultados serán mucho mejores con "shuffle=True" porque se mezclarán todas las imágenes. De otra manera, imágenes de nombre similar muy similares entre sí en apariencia (como es el caso de las imágenes de Kaggle, todas ellas aparentemente similares y llamadas "1_100", "2_100", "3_100"…) empeorarían los resultados del entrenamiento.[16][54]

```
train_it = train_datagen.flow_from_directory('DataSet/train'+cod,target_size=(224,224),batch_size=47,
                                             class_mode='categorical',color_mode='rgb',shuffle=True)
test_it = valid_datagen.flow_from_directory('DataSet/test'+cod,target_size=(224,224),batch_size=47,
                                            class_mode='categorical',color_mode='rgb',shuffle=True)
```

Ilustración 34. Se traslada la información de los directorios a los sets de datos a entrenar.

Se utilizará en el entrenamiento un Learning Rate reducido, de 0.001, con la intención de obtener mejores resultados, dado que se ha utilizado un "batch_size" relativamente pequeño, de 47 imágenes, esto es, el 0.28% de todo el set de datos de entrenamiento. Si se utilizara un Learning Rate más elevado podría producirse una mayor oscilación en torno al valor óptimo de precisión en

los resultados de la validación, y si se utilizara un "batch_size" mayor junto con un Learning Rate de valor más elevado podría ser más difícil lograr alcanzar el valor de precisión que se logra conseguir con un "batch_size" reducido (ilustración 35).[48][49]

```
print("Learning rate por defecto:", model.optimizer.learning_rate.numpy()) #Mostramos el Learning Rate
backend.set_value(model.optimizer.learning_rate, 0.001) #Establecemos un valor que nos proporcione buenos resultados
print("Learning rate:", model.optimizer.learning_rate.numpy()) #Mostramos el nuevo valor
```

Ilustración 35. Se establece un Learning Rate de 0.001 para el entrenamiento.

A continuación, ya se puede realizar el entrenamiento del modelo (ilustración 36). Se cargan los sets de datos de entrenamiento y validación y se indican los pasos por época y el número de épocas.
Al finalizar cada una de las épocas se van recalculando además los resultados de la validación, la cual se realiza durante este proceso de entrenamiento con el propósito de ver qué tan bien se está desempeñando el modelo en cada una de las épocas.
Se entrena el modelo durante 5 épocas dado que se obtienen con esta cantidad de épocas unos excelentes resultados en nuestro caso particular y no se produce Overfitting (que se podría producir de entrenar el modelo durante más épocas de las necesarias).[16]

```
history = model.fit(train_it, steps_per_epoch=358,validation_data=test_it
                    ,validation_steps=45, epochs=5)
                    #El producto del Número de Pasos por el 'batch_size' nunca debe ser mayor que el número
                    #total de imágenes porque faltarían datos para realizar el entrenamiento
```

Ilustración 36. Se entrena el modelo.

Terminado el entrenamiento, con el modelo ya construido, se calculan los resultados finales de la validación (la precisión media $\overline{PR}$) y los mostramos por consola (ilustración 37).[16]

```
#Resultados de la validación mostrados al finalizar el entrenamiento
_, acc = model.evaluate(test_it, steps=len(test_it), verbose=0)
print('> %.3f' % (acc * 100.0))
```

Ilustración 37. Mostramos los resultados de la validación (precisión media $\overline{PR}$) al finalizar el proceso de entrenamiento.

Mostramos también dos gráficos con los valores obtenidos durante el proceso de entrenamiento que muestran respectivamente, la precisión y las pérdidas ocurridas tanto en el entrenamiento como en las validaciones realizados en cada época ejecutada (ilustración 38).[16]

```
acc = history.history['acc'] #Precisión (Accuracy) en el entrenamiento
val_acc = history.history['val_acc'] #Precisión (Accuracy) en la validación
loss = history.history['loss'] #Pérdidas en el entrenamiento
val_loss = history.history['val_loss'] #Pérdidas en la validación
epochs = range(1, len(acc) + 1) #Obtención del número de épocas en función del número de valores del vector
#Construcción de los dos gráficos
pyplot.plot(epochs, acc, 'bo', label='Precisión Entrenamiento')
pyplot.plot(epochs, val_acc, 'b', label='Precisión Validación')
pyplot.title('Precisión de Entrenamiento y Validación')
pyplot.legend()
pyplot.figure() #Creación del segundo gráfico
pyplot.plot(epochs, loss, 'bo', label='Pérdidas Entrenamiento')
pyplot.plot(epochs, val_loss, 'b', label='Pérdidas Validation')
pyplot.title('Pérdidas de Entrenamiento y Validación')
pyplot.legend()
pyplot.show() #Se muestran los dos gráficos
```

Ilustración 38. Mostramos los gráficos con los resultados de precisión y pérdidas del entrenamiento y la validación para cada una de las épocas ejecutadas.

Además, será interesante conocer la matriz de confusión y el f1-score de cada tipo de hortaliza. Recordamos que para ello deberá realizarse un entrenamiento independiente indicando "shuffle=False". De otra manera se obtendrán valores incoherentes en la matriz de confusión.

Luego solo tendrán que compararse las precisiones medias de ambos modelos para estimar qué matriz de confusión correspondería al modelo entrenado con “shuffle=True”, que es el que va a utilizarse en la aplicación.

Para calcular la matriz de confusión (“MConfusion”) se obtiene la predicción de las imágenes del set de datos de validación y se comparan con las etiquetas o clases de dicho set para conocer exactamente como qué hortaliza se ha clasificado cada una de las imágenes.
Utilizando la misma predicción a partir de la cual se ha obtenido la matriz de confusión, se calcula ahora el f1-score (“DatosF1Score”) de cada tipo de hortaliza (ilustración 39).[40]

```
#Obtenemos la predicción de los datos de validación para poder ver posteriormente, mediante la matriz de
#confusión, cómo se desvían los resultados de sus clasificaciones reales
Y_pred = model.predict(test_it, test_it.n//test_it.batch_size+1)
y_pred = np.argmax(Y_pred, axis=1)

MConfusion=sklearn.metrics.confusion_matrix(test_it.classes, y_pred) #Obtenemos la matriz de confusión

#Definimos cuales son las etiquetas del modelo
target_names = ['Kiwis', 'Limones', 'Manzanas', 'Naranjas', 'Peras', 'Plátanos', 'Tomates', 'Zanahorias']
#Obtenemos el 'f1-score' (precisión de los resultados de cada categoría)
DatosF1Score=sklearn.metrics.classification_report(test_it.classes, y_pred, target_names=target_names
                                                   ,zero_division=1) #Evitamos divisiones por cero
print(MConfusion)
print(DatosF1Score)
```

Ilustración 39. Cálculo de la matriz de confusión y f1-score.

Finalmente se guarda el modelo Keras (.h5), el cual se puede utilizar en Python (ilustración 40). Si bien no se trata de algo necesario dado que solo se utilizará el modelo en formato TensorFlow Lite (.tflite) para su uso en Android. Se expondrá a continuación cómo se puede convertir el modelo Keras a formato TensorFlow Lite.[16]

```
model.save('model'+cod+'.h5')
```

Ilustración 40. Guardado del modelo Keras (para Python).

3.2. Arquitectura del programa Python: conversión del modelo Keras a formato TensorFlow Lite para Android

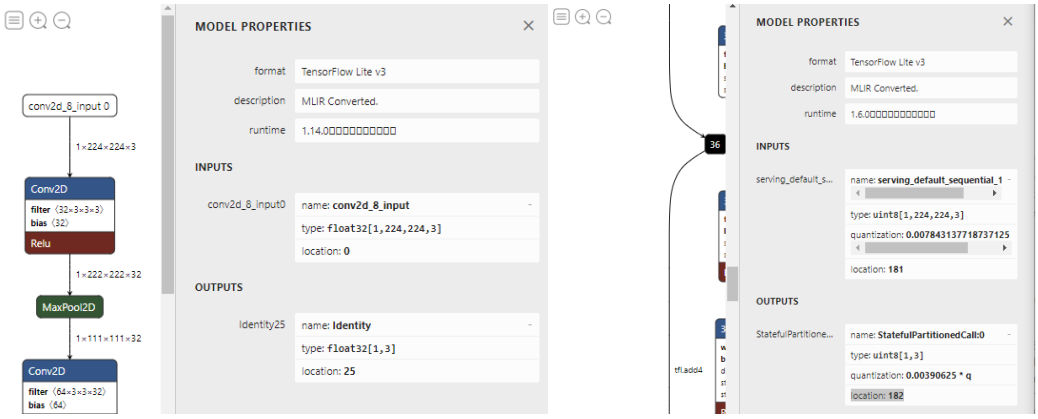
Se deberá guardar el modelo resultante en un formato apto para dispositivos móviles Android. No se podrá utilizar el modelo tal y como se ha generado en Python. Habrá que convertir el modelo original (en formato Keras) a formato TensorFlow Lite, que es un formato más ligero específicamente diseñado para dispositivos portátiles.[55]

Aquí surge una problemática, y es que lo habitual es utilizar modelos TensorFlow Lite en formato UINT8 (valores enteros de 8 bits) ya que las imágenes del dispositivo móvil se obtienen y se opera con ellas en este formato. Por el contrario, el modelo Keras se genera en formato FLOAT32 (valores decimales).

Inicialmente se comenzó haciendo pruebas con modelos TensorFlow Lite en formato UINT8 generados con la herramienta de Google “Teachable Machine”, los cuales funcionaron correctamente en la implementación Android. Si bien se hizo con el único propósito de probar la implementación Android cuando aún no se contaba con el modelo Keras definitivo entrenado con Python. En los modelos generados con “Teachable Machine” no se puede elegir qué algoritmo de entrenamiento utilizar, ni qué función de pérdidas o qué tipo de capas utilizar, ni si quiera se puede

conocer si el modelo que se genera se ha construido utilizando la técnica de Transfer-Learning o si se ha construido de otra manera. Por estas razones no se considera viable la opción de utilizar un modelo generado por "Teachable Machine", ya en formato UINT8, sino que se crea un modelo propio con Python en formato FLOAT32 conociendo todos los detalles sobre cómo se ha construido.[56]

Se intentó de diferentes maneras transformar el modelo Keras generado en Python a UINT8 pero resultaba un proceso complejo, más allá de las indicaciones que proporciona TensorFlow y no terminó por dar buenos resultados.
Se ha utilizado el software "Netron" para la comprobación de las dimensiones, metadatos y formatos (UINT8/FLOAT32) del modelo transformado (ilustraciones 41 y 42):[57][58][59]



Ilustraciones 41 (izquierda) y 42 (derecha). Comprobación de los modelos TensorFlow Lite utilizando "Netron". Izquierda: modelo TensorFlow Lite en formato FLOAT32. Derecha: modelo TensorFlow Lite en formato UINT8 generado con "Teachable Machine", en el que se puede ver en los metadatos cómo tiene unas configuraciones por defecto desconocidas, de tal manera que no se sabe cómo se ha realizado el entrenamiento, y además resultaría extremadamente difícil adaptarlo para obtener diferentes resultados.

Finalmente se optó por adaptar la implementación realizada en Android para acoger al modelo TensorFlow Lite en formato FLOAT32, para lo que debe transformarse en Android la imagen de entrada al modelo a formato FLOAT32, lo cual ha habido que hacerlo una vez convertido el mapa de bits en tensor, y antes de obtener el buffer de este. De esta manera, lo que se hace es convertir dicho tensor de formato UINT8 en un tensor de formato FLOAT32.
Esto se podrá ver más en detalle en el apartado 4.1.3. sobre la arquitectura de la actividad principal de la aplicación Android.

De esta manera, como se puede ver en la ilustración 43, se realiza la conversión de manera sencilla, sin realizar ningún cambio de formato. Se hace además una optimización del modelo para que este ocupe menos tamaño (se realiza una cuantización), y finalmente se guarda como "modelHORTALIZAS.tflite".[60][61][62][63]

```
#Convertimos el modelo keras en un modelo TensorFlow Lite (para Android)
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT] #Cuantizamos el modelo para reducir su tamaño

tfliterecuperado = converter.convert()

open("model"+cod+".tflite", "wb").write(tfliterecuperado) #Guardado del modelo TensorFlow Lite
```

Ilustración 43. Conversión del modelo a formato TensorFlow Lite.

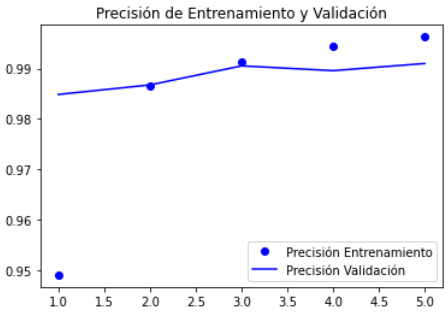
3.3. Resultados obtenidos en los procesos de entrenamiento y validación

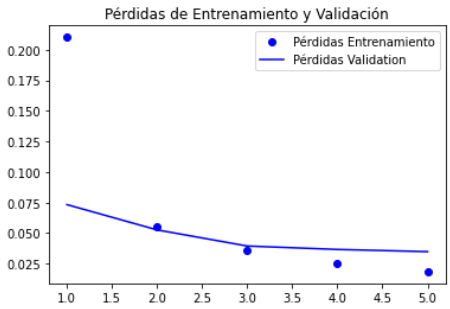
Atendiendo a los resultados que se han obtenido tras la ejecución del programa de entrenamiento, cabe destacar que los resultados en la validación han sido muy buenos, con un 99.1% de precisión media (ilustración 44):

```
Epoch 1/5
358/358 [==============================] - 2339s 7s/step -
loss: 0.4610 - acc: 0.8705 - val_loss: 0.0733 - val_acc:
0.9849
Epoch 2/5
358/358 [==============================] - 2331s 7s/step -
loss: 0.0601 - acc: 0.9861 - val_loss: 0.0527 - val_acc:
0.9868
Epoch 3/5
358/358 [==============================] - 2308s 6s/step -
loss: 0.0379 - acc: 0.9916 - val_loss: 0.0395 - val_acc:
0.9905
Epoch 4/5
358/358 [==============================] - 2112s 6s/step -
loss: 0.0251 - acc: 0.9947 - val_loss: 0.0367 - val_acc:
0.9896
Epoch 5/5
358/358 [==============================] - 2134s 6s/step -
loss: 0.0178 - acc: 0.9968 - val_loss: 0.0349 - val_acc:
0.9910
> 99.102
```

Ilustración 44. Resultados de precisión y pérdidas en el entrenamiento y la validación del modelo principal mostrados por consola relativos a cada una de las 5 épocas ejecutadas. Abajo del todo, resaltado en amarillo, muestra de la precisión media obtenida en el proceso de validación con el modelo completamente entrenado.

Por otro lado, se podrá ver más claramente cómo se ha desempeñado el modelo atendiendo a los gráficos de precisión y pérdidas en el entrenamiento y la validación en función de las épocas ejecutadas (ilustraciones 45 y 46):





Ilustraciones 45 (izquierda) y 46 (derecha). Se muestra en el eje de abscisas el número de épocas ejecutadas. Izquierda: precisión obtenida en los procesos de entrenamiento y en la validación de estos (modelo principal). Derecha: pérdidas obtenidas en los procesos de entrenamiento y en la validación de estos (modelo principal).

Puede verse en el gráfico de la ilustración 45 que el modelo se ajusta bastante bien al valor óptimo de precisión. Se puede ver como la precisión en el entrenamiento se encuentra muy cercana en la época 5 al 100%, por lo que difícilmente se podría continuar el entrenamiento sin sufrir Overfitting. Además, consideramos que un 99.1% de precisión es un resultado bastante bueno, teniendo en cuenta la cantidad de hortalizas pertenecientes a diferentes clases que muestran características similares.

Atendiendo a la matriz de confusión y f1-score obtenidos tras este entrenamiento, podrá verse que tienen valores incoherentes (ilustración 47). Esto se debe a que este entrenamiento se ha realizado con el parámetro “shuffle=True”, que como se ha mencionado con anterioridad, es una manera de mejorar los resultados al ordenar aleatoriamente el set de datos, si bien al ordenarlos aleatoriamente pierden la trazabilidad necesaria para la construcción de la matriz de confusión.

```
[[36 35 35 33 34 24 26 43]
 [33 29 36 26 39 34 39 28]
 [26 37 33 31 28 32 42 35]
 [35 30 36 35 36 25 33 35]
 [23 29 27 41 33 47 27 38]
 [34 36 35 26 35 35 31 31]
 [43 33 36 38 25 32 30 27]
 [36 36 29 39 30 32 34 28]]
              precision    recall  f1-score   support

       Kiwis       0.14      0.14      0.14       266
    Limones        0.11      0.11      0.11       264
   Manzanas        0.12      0.12      0.12       264
   Naranjas        0.13      0.13      0.13       265
      Peras        0.13      0.12      0.13       265
   Plátanos        0.13      0.13      0.13       263
    Tomates        0.11      0.11      0.11       264
 Zanahorias        0.11      0.11      0.11       264
```

Ilustración 47. Matriz de confusión y f1-score obtenidos tras el entrenamiento del modelo que vamos a utilizar en nuestra aplicación. Se puede observar que los valores resultan incoherentes. Esto se debe a que se ha realizado una mezcla de imágenes que no permite conocer los valores reales de la matriz de confusión.

No obstante, se extrapolará una matriz de confusión para el modelo en base a la de otro modelo auxiliar que se entrenará con las mismas características, pero con el parámetro “shuffle=False”.

Se obtiene en el entrenamiento de este modelo auxiliar (creado expresamente para la visualización de la matriz de confusión), una precisión media en la validación de solamente el 97.92%, que, comparada con el 99.10% obtenido en el modelo que vamos a implementar, se trata de un valor un 131% más alejado del cien por ciento de precisión que el modelo principal $\left(\frac{2.08\%-0.90\%}{0.90\%}\right)$.

Si se ve la matriz de confusión y los f1-score obtenidos del entrenamiento de este modelo auxiliar (ilustración 48), se observa que el único error relevante que se aprecia es que algunas peras (el 7.5% de ellas) han sido clasificadas como manzanas.
En vista de que la precisión en la validación del modelo principal se encuentra alejada un 0.90% del cien por ciento de precisión y que la precisión del modelo auxiliar se encuentra alejada un 2.08%, la diferencia bruta de porcentaje de precisión hasta el cien por ciento en el modelo auxiliar es de 2.31 veces la del modelo principal $(2.08\% = 2.31 \cdot 0.9\% = (100\% + 131\%) \cdot 0.9\%)$.
De esta manera consideraremos razonable estimar un error en la precisión de la validación 2.31 veces menor del existente en los resultados del modelo auxiliar para realizar la estimación de una posible matriz de confusión para el modelo principal. Dicha matriz se muestra en la Tabla 6, y los f1-score obtenidos de la matriz de confusión estimada se muestran en la Tabla 7, junto con la precisión media obtenida de la matriz.

Hortaliza predicha

Hortaliza real

```
                Kiwi Limón Manzana Naranja Pera Plátano Tomate Zanahoria
Kiwi      [[260    4      1       0      0    0       1      0]
Limón     [  1  262      0       1      0    0       0      0]
Manzana   [  0    2    262       0      0    0       0      0]
Naranja   [  0    0      0     265      0    0       0      0]
Pera      [  1    2     20       1    240    0       1      0]
Plátano   [  0    2      0       0      1  259       0      1]
Tomate    [  0    0      0       1      0    0     263      0]
Zanahoria [  0    0      0       4      0    0       0    260]]
                          precision    recall  f1-score   support

                   Kiwis       0.99      0.98      0.98       266
                 Limones       0.96      0.99      0.98       264
                Manzanas       0.93      0.99      0.96       264
                Naranjas       0.97      1.00      0.99       265
                   Peras       1.00      0.91      0.95       265
                Plátanos       1.00      0.98      0.99       263
                 Tomates       0.99      1.00      0.99       264
              Zanahorias       1.00      0.98      0.99       264

                accuracy                           0.98      2115
               macro avg       0.98      0.98      0.98      2115
            weighted avg       0.98      0.98      0.98      2115
```

Ilustración 48. Matriz de confusión y f1-score obtenidos tras el entrenamiento del modelo auxiliar.

		Hortaliza predicha							
		Kiwi	Limón	Manzana	Naranja	Pera	Plátano	Tomate	Zanahoria
Hortaliza real	Kiwi	264	2	0	0	0	0	0	0
	Limón	0	264	0	0	0	0	0	0
	Manzana	0	1	263	0	0	0	0	0
	Naranja	0	0	0	265	0	0	0	0
	Pera	0	1	9	0	255	0	0	0
	Plátano	0	1	0	0	0	262	0	0
	Tomate	0	0	0	0	0	0	264	0
	Zanahoria	0	0	0	2	0	0	0	262

Tabla 6. Matriz de confusión estimada para el modelo principal, con 2.31 veces menos error en la precisión que la matriz de confusión del modelo auxiliar.

Precisión media
99.24%

	F1-Score
Kiwis	0.996
Limones	0.993
Manzanas	0.981
Naranjas	0.996
Peras	0.981
Plátanos	0.998
Tomates	1.000
Zanahorias	0.996

Tabla 7. Precisión media $\overline{PR}$ y f1-score obtenidos de la matriz de confusión estimada para el modelo principal.

Pueden verse en las tablas 6 y 7 los resultados de estimar para el modelo principal 2.31 veces menos error que en el modelo auxiliar. La precisión media obtenida en la estimación ha sido de 99.24%, un 0.14% más que la del modelo principal, un valor cercano, dado que así se ha ajustado. Es algo superior debido a los redondeos en la matriz de confusión.
En cuanto al problema de las peras clasificadas como manzanas, el error ahora quedaría reducido del 7.5% al 3.4%, si bien el problema en sí mismo puede haberse debido a utilizar conjuntos de imágenes muy similares entre sí debido a no haber utilizado “shuffle=True”.

No se debe olvidar además que se trata de una estimación. Utilizando la metodología básica e intuitiva que se ha empleado, nunca podrá deducirse la matriz de confusión real del modelo principal en base a la que ha resultado del modelo auxiliar. Esto es debido a la naturaleza caótica de la red neuronal (debida al proceso de aprendizaje automático), que imposibilitará conocer de manera tan sencilla los valores reales que tendría la matriz de confusión del modelo principal.

Como se desconoce de qué manera se realiza el mezclado aleatorio de imágenes cuando se indica “shuffle=True”, y además resulta complicado conocer cómo se realizan las operaciones no lineales internas de la red neuronal a lo largo del entrenamiento, resultará imposible realizar una estimación mucho más realista que la realizada en este apartado.
Lo más adecuado sería encontrar una manera de calcular la matriz de confusión a pesar de que se indique “shuffle=True”, pero se ha considerado que hacerlo mediante una estimación, como se ha hecho, resulta lo suficientemente preciso para el propósito puramente analítico con el que se ha realizado.

4. Aplicación Android

Para la construcción de la aplicación Android se ha utilizado el software "Android Studio", que es el software más utilizado desde hace varios años para la construcción de aplicaciones Android.[7]

Por otro lado, en los scripts principales de la aplicación se ha utilizado el lenguaje de programación Kotlin. Esto se ha hecho de esta manera debido a la simplicidad de su uso, a que es muy comúnmente utilizado en las nuevas aplicaciones y a que se pueden realizar conversiones entre scripts en el lenguaje de programación Java y scripts en Kotlin, por lo que se trata de un lenguaje bastante interoperable.[64]

La aplicación que se ha diseñado, llamada FotoCheck Hortalizas (FotoCheckHor), cuenta principalmente con dos funcionalidades:

- Identificación del tipo de hortaliza presente en una imagen cargada desde Galería o tomada con la cámara del smartphone sin salir de la aplicación.

- Guardado y carga de información relativa a la realización de un control de calidad, concretamente, qué hortalizas se han comprobado y cuál era su grado de calidad.

Se muestra a continuación de manera ilustrativa, en la ilustración 49, un diagrama de flujo del funcionamiento de la aplicación una vez abierta, desde la pantalla principal (se da por hecho que se autoriza el permiso de cámara para simplificar el diagrama):

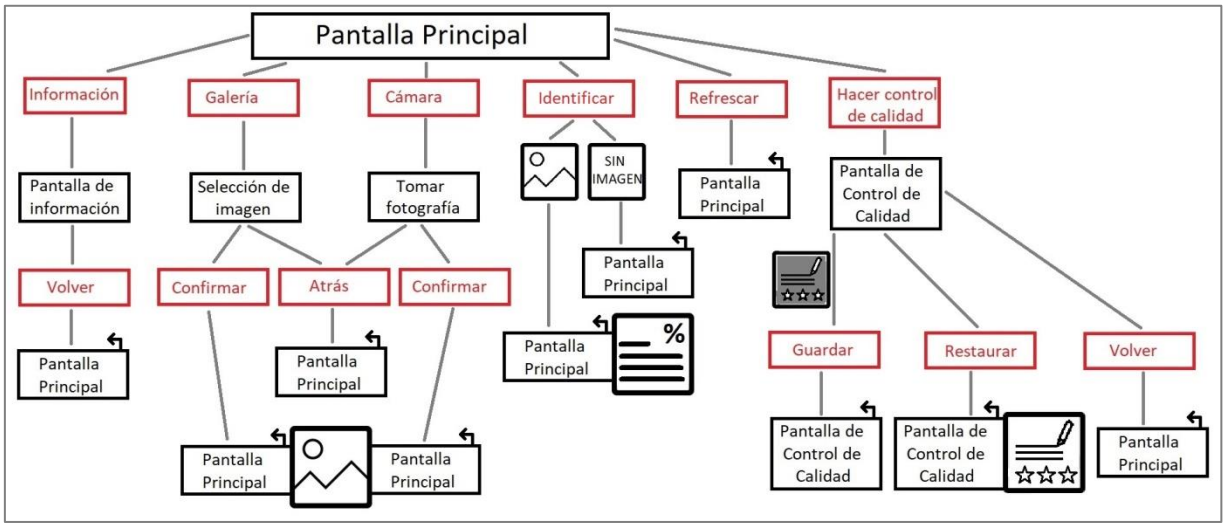


Ilustración 49. Diagrama de flujo descriptivo del funcionamiento de la aplicación.

4.1. Arquitectura de la aplicación Android

Ahora que ya se cuenta con una idea general de cómo funciona la aplicación, entraremos en detalle sobre cómo se ha realizado la implementación en Android Studio.
La aplicación se subdivide en tres clases o programas diferentes (ilustración 50):

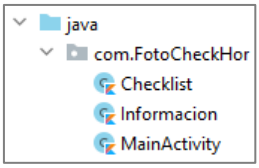


Ilustración 50. Clases de la aplicación.

La actividad principal que desarrolla la aplicación (lo cual se indica en el archivo "AndroidManifest.xml") se encuentra en la clase "MainActivity". La clase "Informacion" sirve para mostrar el tutorial sobre cómo utilizar la aplicación y la clase "Checklist" es la encargada de la funcionalidad de control de calidad.

4.1.1. Configuraciones de la aplicación: Android Manifest

Se muestra en la ilustración 51 dónde puede encontrarse el archivo "AndroidManifest.xml" de la aplicación. En este se definen algunas de las configuraciones más importantes de toda la aplicación.

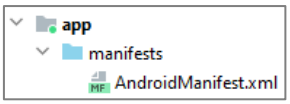


Ilustración 51. Ubicación del archivo "AndroidManifest.xml" dentro del directorio "app". Visto desde Android Studio.

En este archivo de configuraciones deben indicarse los permisos que se van a utilizar: el permiso de cámara para poder realizar las fotografías de las hortalizas, y los permisos de escritura y lectura en memoria para poder guardar y cargar los datos guardados en la funcionalidad de control de calidad (ilustración 52).

```
8     <uses-permission android:name="android.permission.CAMERA"/>
9     <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
10        tools:ignore="ScopedStorage" />
11    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Ilustración 52. Declaración de permisos en "AndroidManifest.xml".

Por otro lado, se establece el nombre de la aplicación en el teléfono, el icono, así como los temas de la aplicación (color primario/secundario). Esto puede verse en la ilustración 53.

```
15    <application
16        android:icon="@mipmap/ic_launcher"
17        android:label="FotoCheck Hortalizas"
18        android:roundIcon="@mipmap/ic_launcher_round"
19        android:supportsRtl="true"
20        android:requestLegacyExternalStorage="true"
21        android:theme="@style/Theme.FotoCheck">
```

Ilustración 53. Definición del estilo de la aplicación (nombre, tema, iconos…) en "AndroidManifest.xml".

Se añade también un "FileProvider" necesario para poder utilizar fotografías hechas con la cámara, dado que en nuevas versiones de Android las fotografías se guardan temporalmente solo como thumbnails (imágenes-icono) de baja calidad, por lo que para mostrar la imagen de lo que se ha fotografiado en la pantalla principal sin que se vea borroso, debe guardarse la imagen en memoria temporalmente para poder recuperarla en su calidad original. Para ello, es necesario configurar un FileProvider (ilustración 54) en el que se indica la dirección de un archivo que debe crearse, el cual contiene la dirección donde se debe guardar la imagen (ilustración 55).[65][66]

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="com.FotoCheckHor.fileprovider"
    android:exported="false"
    android:grantUriPermissions="true">
    <!--Archivo file_paths, en el que indicamos dónde guardar la foto temporal-->
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/file_paths" />
</provider>
```

Ilustración 54. Configuración de un FileProvider en “AndroidManifest.xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<!--Indicamos dónde guardar la foto temporal-->
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="FCheck" path="Android/data/com.FotoCheckHor/files/Pictures" />
</paths>
```

Ilustración 55. Archivo “file_paths.xml” utilizado como recurso del FileProvider de “AndroidManifest.xml”.

Se indica que MainActivity es la clase principal con la que debe iniciarse la aplicación (ilustración 56).

Aunque la acción realizada al pulsar el botón de cámara proporcione la información necesaria a la actividad en segundo plano que se ocupa de hacer la foto, si se rota la pantalla y se reinician los procesos, la actividad en segundo plano se reiniciará, pero no las acciones “onCreate” resultado de haber presionado el botón de cámara, por lo que "ArchivoFoto" (el archivo .jpg temporal) quedaría sin inicializar y no existiría. Por ello es necesario o bien bloquear la orientación, lo cual perjudicaría la experiencia del usuario, o bien evitar el reinicio de los procesos. Esto último ayudaría además a no perder la imagen tomada si se rota la pantalla, algo que ocurre en la mayoría de aplicaciones, pero por una buena razón, ya que si se quiere utilizar una interfaz horizontal/vertical personalizada (la interfaz diseñada específicamente para una orientación concreta) y no el reposicionamiento automático de los elementos, deben reiniciarse los procesos. En nuestro caso, abogando por la utilidad, optaremos por no perder la imagen al rotar la pantalla, mostrándose una interfaz de peor apariencia, pero solo en caso de cambiar la orientación de la pantalla que muestra la imagen (la pantalla principal). Esta configuración se establece mediante el parámetro “android:configChanges”, al que se le da la configuración "orientation|screenSize"'.[67][68]

```
<activity android:name=".MainActivity"
    android:configChanges="orientation|screenSize"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Ilustración 56. Declaración de la clase principal con la que debe iniciarse la aplicación, “AndroidManifest.xml”.

Se declara asimismo en “AndroidManifest.xml” el resto de clases de la aplicación, las relativas a la funcionalidad de control de calidad, “Checklist”, y al tutorial de uso de la aplicación, “Informacion” (ilustración 57).

```
<activity android:name=".Checklist" />
<activity android:name=".Informacion" />
```

Ilustración 57. Declaración de las clases Checklist e Informacion en “AndroidManifest.xml”.

4.1.2. Configuraciones de la aplicación: Build Gradle de la aplicación

En el archivo “build.gradle(:app)” se definen las configuraciones relativas a la construcción de la aplicación, a acciones que deben realizarse o dependencias que deben cargarse durante la

construcción de la aplicación. Se muestra en la ilustración 58 dónde se puede encontrar el archivo en la aplicación.

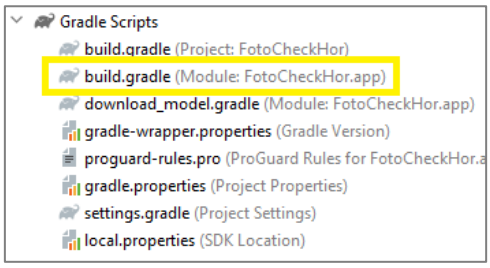


Ilustración 58. Ubicación del archivo "build.gradle(:app)" dentro del apartado "Gradle Scripts" resaltado en amarillo. Visto desde Android Studio.

Se ha diseñado la aplicación con la funcionalidad de obtener de un servicio de almacenamiento en la nube del archivo del modelo TensorFlow Lite que vamos a utilizar. De esta manera, cuando se construye la aplicación se descarga el modelo, por medio de un acortador de enlace, para poder modificar cuando queramos la dirección de destino del archivo a descargar desde el panel de control de "Acortar.link", y se guarda en la carpeta "ml" (ilustración 59).[69]

Se ha intentado guardar el modelo en la nube en diferentes plataformas, como Google Drive, USAupload, FILE.re, etc. Si bien no ha resultado exitoso ya que no se podía obtener un enlace que abriera directamente el archivo, sino que había de por medio una interfaz web que imposibilitaba la descarga automática a la hora de construir el modelo. Solo se encontraron dos servicios de almacenamiento en la nube gratuitos que permitieran acceder a un enlace que abriera directamente el archivo: Filebin y OpenDrive, si bien este último tenía una velocidad de descarga muy lenta y se optó por utilizar Filebin a pesar de que los archivos solamente permanecen guardados 7 días (pero se puede reiniciar la cuenta atrás actualizando los archivos).[70][71][72][73][74]

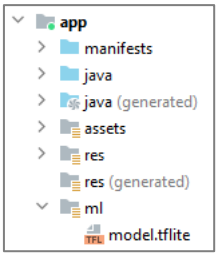


Ilustración 59. Ubicación del modelo en el directorio "app". Visto desde Android Studio.

Así pues, se añade en "build.gradle(:app)" el plugin necesario para realizar esta descarga en la construcción (ilustración 60), y además se indica que se ejecute el script de configuración encargado de hacer la descarga (se muestra el contenido de este archivo en la ilustración 61). Se indica además en "build.gradle(:app)", en características de construcción, que se permita importar modelos (ilustración 62).[75]

```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'de.undercouch.download' //Necesario para realizar la descarga del modelo
}

//Descargamos nuestro modelo de "acortar.link/FotoCheckModel", un acortador de enlace que configuramos para que
//nos redirija a la dirección en donde se encuentra nuestro modelo TensorFlow Lite actualizado.
apply from:'download_model.gradle'
//LA DESCARGA SE REALIZA SOLAMENTE EN LA CONSTRUCCIÓN DE LA APLICACIÓN
```

Ilustración 60. Plugins de "build.gradle(:app)" e indicación de que debe ejecutarse el programa encargado de descargar el modelo.

```
task downloadModelFile(type: Download) {
    //Dirección en donde se encuentra el modelo TensorFlow Lite actualizado
    src 'https://acortar.link/FotoCheckModel'

    //Indicamos la carpeta donde vamos a guardar el modelo y el nombre con el que lo vamos a guardar
    dest 'src/main/ml/model.tflite'

    //Reemplazamos el modelo existente por el actualizado
    overwrite true

    //LA DESCARGA SE REALIZA SOLAMENTE EN LA CONSTRUCCIÓN DE LA APLICACIÓN

}

preBuild.dependsOn downloadModelFile
```

Ilustración 61. Archivo "download_model.gradle".

```
    buildFeatures {
        mlModelBinding true //Permite importar modelos TensorFlow Lite
    }
```

Ilustración 62. Se habilita la importación de modelos en "build.gradle(:app)".

En el script encargado de hacer la descarga, contenido en "Download_model.gradle" (ilustración 61), se indica el enlace de donde debe obtenerse el modelo (https://acortar.link/FotoCheckModel), se indica el lugar de destino donde debe guardarse con el nombre "model.tflite" (ilustración 59), y en la última línea se indica que debe descargarse el modelo para poder realizar la construcción de la aplicación.[75]

Otra de las configuraciones de interés definidas en "build.gradle(:app)" es el apartado de configuración por defecto (ilustración 63), donde se define el ID de la aplicación, que es un identificador único para distinguir, por ejemplo, entre diferentes aplicaciones en Google Play Store. Se definen también la versión de Android mínima en la que se permite instalar la aplicación, la versión objetivo, el código de versión, utilizado por Google Play para diferenciar diferentes versiones de programa y el nombre de dicha versión.

```
android {
    compileSdkVersion 31
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.FotoCheckHor"
        minSdkVersion 24    //Versión mínima de Android, cubre a más del 90% de los usuarios de Android
        targetSdkVersion 31 //Preparado para la versión más nueva de Android (es lo más recomendable)
        versionCode 12      //Código único de la versión de programa
        //V.1 a V.9 Desarrollo de la aplicación
        //V.10 Mejora de la interfaz gráfica en dispositivos con pantalla pequeña
        //V.11 Añadida pestaña de información, botón de refresco de pantalla y otras mejoras visuales
        //V.12 Actualización del nombre de paquete de FotoCheck a FotoCheckHor
        versionName "FCHOR-V12" //Nombre de la versión de programa
    }
```

Ilustración 63. Configuración por defecto de la aplicación definida en "build.gradle(:app)".

La versión mínima de Android en que se permite instalar la aplicación es Android 7.0. Esto se ha decidido así dado que desde Android Studio se recomienda trabajar con las versiones de Android de la 7.0 a la 11.0 (ilustración 64). Además, quedan incluidos de esta manera más del 89% de los dispositivos (ilustración 65). Por otro lado, Android Studio siempre recomienda que la versión Android objetivo sea la última versión de Android, por lo que se ha seleccionado a este efecto Android 12.[76]

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
R	30	x86	Android 11.0 (Google Play)
Q	29	x86	Android 10.0 (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo	27	x86	Android 8.1 (Google Play)
Oreo	26	x86	Android 8.0 (Google Play)
Nougat	25	x86	Android 7.1.1 (Google Play)
Nougat	24	x86	Android 7.0 (Google Play)

Ilustraciones 64 (izquierda) y 65 (derecha). Izquierda: versiones de Android recomendadas en Android Studio. Derecha: cuota de mercado de las diferentes versiones de Android en el 2019, proporcionado por Google.

Deben añadirse además las respectivas dependencias de las diferentes librerías de que se hace uso en las diferentes clases (ilustración 66).[75][77]

```
dependencies {

    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'org.tensorflow:tensorflow-lite-support:0.1.0'
    implementation 'org.tensorflow:tensorflow-lite-metadata:0.1.0'
    implementation 'androidx.media2:media2:1.0.0-alpha04'
    implementation 'androidx.media2:media2-common:1.1.2'
}
```

Ilustración 66. Dependencias definidas en “build.gradle(:app)”.

4.1.3. Actividad principal de la aplicación

Se describen a continuación los diferentes procesos realizados en la actividad principal de la aplicación. Para poder ver más claramente las relaciones que se muestran más adelante, se muestra la interfaz gráfica de la actividad principal en la ilustración 67.

Ilustración 67. Muestra de cómo se ve la actividad principal en un dispositivo de 6.5 pulgadas con orientación Horizontal.

En primer lugar, se definen las variables globales que es necesario utilizar en diferentes funciones de la clase MainActivity o que sirven para vincular elementos de la interfaz gráfica (concretamente botones, imágenes y cuadros de texto) con elementos operables definidos en la clase (ilustración 68).

```
package com.FotoCheckHor

import ...

class MainActivity : AppCompatActivity() {

    //AÑADIR AL VECTOR TANTOS VALORES COMO ETIQUETAS (CATEGORÍAS) TENGA EL MODELO
    val indiceArrayF = arrayOf(0.0F,0.0F,0.0F,0.0F,0.0F,0.0F,0.0F,0.0F)

    //Elementos a vincular con los elementos de la interfaz gráfica
    lateinit var abrirGaleria : Button
    lateinit var camerabtn : Button
    lateinit var hacerPrediccion : Button
    lateinit var img_view : ImageView
    lateinit var text_view : TextView
    lateinit var porcentaje : TextView

    lateinit var ArchivoFoto: File    //Se utiliza para el guardado temporal de la Foto cuando se fotografía
    lateinit var bitmap: Bitmap       //Es el mapa de bits utilizado para el procesado de imagen
    var abletopredict: Boolean=false  //Sirve para no permitir realizar la predicción si no se ha extraido
                                      //todavía el mapa de bits de una fotografía
```

Ilustración 68. Declaración de variables globales en “MainActivity.kt”.

Cabe destacar la definición del vector “indiceArrayF”, que debe tener tantos valores como diferentes tipos de hortalizas puede identificar el modelo. Se ha definido aquí, en un lugar visible, para que, llegado el caso, solo haya que cambiar este vector y el archivo de texto de las etiquetas, “labels.txt”, contenido en el directorio “assets” (ilustración 69), para poder utilizar un modelo con una cantidad diferente de tipos de elementos (esto se deja indicado en el archivo “README.txt” para dar a conocer qué elementos se deben cambiar para utilizar un modelo que identifica una cantidad diferente de elementos; véase la ilustración 70).

Ilustración 69. Archivo “labels.txt”.

```
Autor: Luis Manuel Muñoz Pérez
07/12/2021

La presente aplicación cuenta con una funcionalidad de identificar hortalizas además de con una funcionalidad
de control de calidad que permite guardar y cargar una evaluación de calidad de las hortalizas.

El modelo utilizado para la función de identificación está entrenado para distinguir entre 8 tipos de
Hortalizas: Kiwis, Limones, Manzanas Golden, Naranjas, Peras, Plátanos, Tomates y Zanahorias. Si bien la
aplicación puede adaptarse para ser utilizada para distinguir entre otro tipo diferente de elementos. El
modelo se obtiene de un acortador de enlace en el que podemos indicar la dirección a un modelo cualquiera
que hayamos guardado previamente en un servicio de almacenamiento en la nube que permita obtener el fichero
directamente.
Sí que se deberá realizar un ajuste a nivel de código (no se ha implementado a nivel de usuario, ya que el
usuario tampoco puede cambiar el modelo) la cantidad de elementos a identificar.

Para realizar este ajuste, solamente hay que modificar el fichero "labels.txt" (se encuentra en el directorio
/assets), así como el vector "indiceArrayF" definido en MainActivity.


Utilizando la aplicación, si se realiza el análisis y no hay ninguna de las 8 hortalizas en la imagen, se
producirá una clasificación incorrecta, lo que aparece en la imagen se clasificará en una de las regiones
determinadas y podrá mostrarse que es una u otra fruta con mucha probabilidad cuando en realidad no aparece en
la imagen.
```

Ilustración 70. Archivo “README.txt” de la aplicación. Resaltadas las instrucciones sobre cómo adaptar la aplicación para ser utilizada con un modelo que realiza la identificación de una cantidad diferente de tipos de elementos.

Se puede ver en la ilustración 68 también la declaración de algunas de las variables relativas a la imagen que deben ser usadas en varias funciones, y una variable booleana que afecta a la acción que se produce al presionar “IDENTIFICAR”, la variable “abletopredict”.

Una de las primeras acciones que deberá realizarse es la concerniente a la comprobación de los permisos, que se define a continuación. Si se detecta que no se han autorizado los permisos que solicitamos, se pide al usuario que autorice esos permisos a la aplicación (ilustración 71). Concretamente será necesario conceder el permiso de acceso a cámara para realizar las fotografías. El permiso de almacenamiento no es necesario pedirlo (solo se almacena información en el directorio destinado a archivos de la propia aplicación).[75]

```
fun checkandGetpermissions(){//Se comprueba si tenemos el permiso de cámara y se solicita
                             //en caso de no tenerlo
    if(checkSelfPermission(android.Manifest.permission.CAMERA) == PackageManager.PERMISSION_DENIED){
        //Si el permiso de cámara está denegado solicitamos que se conceda
        requestPermissions(arrayOf(android.Manifest.permission.CAMERA), requestCode: 100)
    }
    else{//Acción realizada cada ejecución en que tenemos garantizado el permiso de cámara
    }
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray
) {//Leemos el resultado de la solicitud del permiso de cámara en caso de que tuviésemos denegado el permiso
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if(requestCode == 100){
        if (grantResults.isNotEmpty()){
        if(grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            Toast.makeText( context: this, text: "Permiso de Cámara concedido", Toast.LENGTH_SHORT).show()
        }
        else{
            Toast.makeText( context: this, text: "Permiso de Cámara denegado", Toast.LENGTH_SHORT).show()
        }} else Toast.makeText( context: this, text: "SIN ACCESO", Toast.LENGTH_SHORT).show()
    }
}
```

Ilustración 71. Comprobación y solicitud de permisos.

A continuación, se creará la función “onCreate” que se ejecuta al inicializar la clase (ilustración 72).

```
@SuppressLint( ...value: "SetTextI18n") //Recomendado para escribir texto concatenado
override fun onCreate(savedInstanceState: Bundle?) {

    //Asociamos el layout (interfaz gráfica) que le corresponde a la actividad principal
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //Asociamos la previsualización de imagen, cuadros de texto y botones con los elementos de la interfaz gráfica
    abrirGaleria = findViewById(R.id.Galeria)
    camerabtn = findViewById(R.id.Camara)
    hacerPrediccion = findViewById(R.id.verResultado)
    img_view = findViewById(R.id.Foto)
    text_view = findViewById(R.id.textView)
    porcentaje = findViewById(R.id.percentage)

    //Comprobamos los permisos y los solicitamos si no están concedidos
    checkandGetpermissions()

    //Guardamos las etiquetas de clasificación del modelo, esto es, cada tipo de elemento soportado
    val labels = application.assets.open( fileName: "labels.txt").bufferedReader().use { it.readText() }.split( ...delimiters: "\n")
```

Ilustración 72. Declaraciones iniciales de la función “onCreate” de la actividad principal.

Como se puede ver en la ilustración 72, en primer lugar, se asocia la actividad principal de la aplicación con su interfaz gráfica “activity_main”, que será la pantalla de inicio que se muestra al abrir la aplicación.

A continuación, se definen los vínculos entre las variables globales mencionadas al principio de este punto y sus respectivos elementos de la interfaz gráfica a los que pertenecen.

Luego se efectúa la comprobación y solicitud de permisos expuesta anteriormente, y después se guardan los valores del archivo de etiquetas “labels.txt” en una variable inmutable.[75]

En la función de inicio “onCreate” también se definen las acciones que se ejecutan cuando interactuamos con los elementos de la interfaz, de ahí que se realice la vinculación, como se aprecia en la ilustración 72, al inicio de la función “onCreate”.

En base a la ilustración 67, al presionar el botón “GALERÍA” (ilustración 73) se limpia la pantalla, se crea un tipo de acción de obtener contenido de tipo imagen (usando el explorador de archivos del dispositivo) y se inicia esta acción en una actividad en segundo plano, que se muestra en la ilustración 74.[65][75]

```
98      //Acción al clicar el botón "Galería"
99      abrirGaleria.setOnClickListener(View.OnClickListener { it: View!
100         limpiarpantalla()
101         val intent : Intent = Intent(Intent.ACTION_GET_CONTENT)
102         intent.type = "image/*"
103         //Iniciamos la actividad en segundo plano de acceder a contenido para obtener una imagen
104         startActivityForResult(intent, requestCode: 250)
105     })
```

Ilustración 73. Acción al presionar el botón de Galería.

```
204 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
205     super.onActivityResult(requestCode, resultCode, data)
206     //Actividad en segundo plano de "Galería"; con RESULT_OK indicamos que hemos obtenido correctamente la imagen
207     if(requestCode == 250 && resultCode== Activity.RESULT_OK){
208         //Recuperamos la dirección de la imagen de la acción que extiende a esta actividad en segundo plano
209         val uri : Uri ?= data?.data
210         //Obtenemos el mapa de bits de la imagen seleccionada en Galería
211         bitmap=MediaStore.Images.Media.getBitmap(contentResolver, uri)
212         //Hacemos la imagen cuadrada (tomamos la parce central y eliminamos los bordes)
213         bitmap = ImagenCuadrada(bitmap)
214         img_view.setImageBitmap(bitmap) //Mostramos la imagen resultante en la aplicación
215         abletopredict=true //Permitimos presionar "Identificar"
216     }
```

Ilustración 74. Actividad en segundo plano desarrollada después de haber presionado “GALERÍA”.

Una vez se elige una imagen (con “resultCode==Activity.RESULT_OK” nos aseguramos de que la imagen se ha obtenido exitosamente), desde la actividad en segundo plano se obtiene su dirección única de memoria “uri”, y a partir de esta se guarda la imagen como mapa de bits en la variable global “bitmap”, se realiza el recorte de la imagen para hacerla cuadrada (para poder ser utilizada posteriormente en el modelo) y se muestra la imagen recortada en la pantalla principal. Finalmente se indica que ya puede ejecutarse el proceso de predicción al presionar el botón “IDENTIFICAR”.[65][75][78]

Al presionar el botón “CÁMARA” (ilustración 75) se limpia la pantalla, se obtiene la dirección de memoria específica donde nuestra aplicación puede guardar sus imágenes, se borran todas las imágenes que pudiera haber guardadas en dicho directorio y se crea un archivo temporal de tipo “.jpg” en este. Si no se borrara la imagen ya presente en el directorio, la nueva imagen temporal, de la manera en que se crea, no sobrescribiría la anterior, sino que se guardaría por separado con su propia dirección única “uri”, por ello es importante borrar el contenido del directorio antes de utilizarlo para el almacenamiento temporal de la imagen de cámara.

Se obtiene la dirección “uri” de la imagen temporal recién creada y se crea un tipo de acción hacer fotografía indicando que la fotografía que se tome deberá guardarse en esa dirección “uri”, sustituyendo la imagen temporal. Luego se inicia la actividad en segundo plano de realizar la fotografía, que se muestra en la ilustración 76.[65][75]

```
107     //Acción al clicar el botón "Cámara"
108     camerabtn.setOnClickListener(View.OnClickListener {  it: View!
109         limpiarpantalla()
110         //Obtenemos la dirección donde podemos guardar las imágenes de nuestra aplicación
111         val DirectorioAlmacenamiento: File? =getExternalFilesDir(Environment.DIRECTORY_PICTURES)
112         //Creamos un archivo de imagen temporal (que luego deberemos borrar; no se reescribe al volver
113         //a crearlo y las imágenes se irían acumulando en caché de no hacerlo)
114         if (DirectorioAlmacenamiento != null) {
115             if (DirectorioAlmacenamiento.isDirectory()) for (child in DirectorioAlmacenamiento.listFiles()){
116                 child.delete() //Borramos todas las imágenes que ha guardado la aplicación
117             }                  //antes de guardar el nuevo archivo temporal
118         }
119         ArchivoFoto=File.createTempFile( prefix: "FotoCheckAUX",  suffix: ".jpg", DirectorioAlmacenamiento)
120         //Obtenemos la dirección del nuevo archivo creado
121         val UriImg:Uri=FileProvider.getUriForFile( context: this,  authority: "com.FotoCheckHor.fileprovider",ArchivoFoto)
122         //Creamos el tipo de acción que queremos efectuar (hacer fotografía)
123         val camera : Intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
124         //Indicamos además que la imagen que se obtenga se guardará en la dirección del nuevo archivo
125         camera.putExtra(MediaStore.EXTRA_OUTPUT,UriImg)
126         startActivityForResult(camera,  requestCode: 200) //Ejecutamos la acción en segundo plano
127     })
```

Ilustración 75. Acción al presionar el botón de Cámara.

```
217     //Actividad en segundo plano de "Cámara"; con RESULT_OK indicamos que hemos obtenido correctamente la imagen
218     else if(requestCode == 200 && resultCode == Activity.RESULT_OK){
219         //Obtenemos el mapa de bits a partir de la imagen temporal que hemos creado
220         bitmap = BitmapFactory.decodeFile(ArchivoFoto.getAbsolutePath()) as Bitmap
221         //Hacemos la imagen cuadrada (tomamos la parce central y eliminamos los bordes)
222         bitmap = ImagenCuadrada(bitmap)
223         img_view.setImageBitmap(bitmap) //Mostramos la imagen resultante en la aplicación
224         ableTopredict = true //Permitimos presionar "Identificar"
225     }
226 }
```

Ilustración 76. Actividad en segundo plano desarrollada después de haber presionado “CÁMARA”.

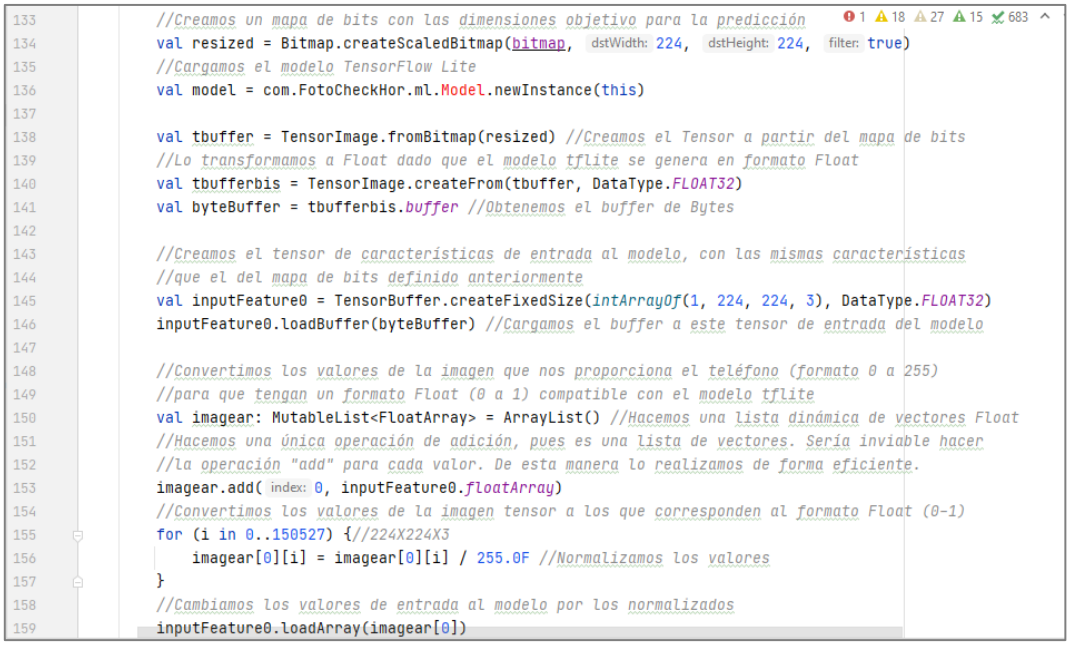
Una vez tomada la fotografía (con “resultCode==Activity.RESULT_OK”, al igual que antes, nos aseguramos de que la fotografía se ha tomado exitosamente) se obtiene su mapa de bits, se guarda en la variable global “bitmap”, y, al igual que en el caso ya mencionado de obtener la imagen de la galería, se realiza el recortado de la imagen y se muestra la imagen recortada por pantalla. Luego se posibilita ejecutar la identificación, el proceso de predicción.[65][75][78]

El proceso de identificación o predicción se realiza al presionar el botón “IDENTIFICAR” que se muestra en la interfaz mostrada en la ilustración 67. Debe haberse obtenido previamente una imagen por medio de la Galería o habiendo tomado una fotografía con la cámara dentro de la aplicación. Si esta condición no se cumple, si no se cuenta con una imagen, simplemente se muestra por pantalla un mensaje, ligeramente diferente al que aparece cuando abrimos la app, que indica que para poder ver los resultados primero se debe cargar una imagen de la Galería o tomar una fotografía (ilustración 77).

```
197     }else{//Si no contamos con una imagen indicamos que debe seleccionarse primero una imagen
198         text_view.setTextColor(Color.BLACK)
199         text_view.setText("Seleccione una imagen pulsando GALERÍA o tome una pulsando CÁMARA para poder ver los resultados")
200     }
201 })
```

Ilustración 77. Acción realizada al presionar “IDENTIFICAR” y no contar con una imagen para hacerlo.

Si cuando es presionado “IDENTIFICAR” se cuenta ya con una imagen, comienza todo el proceso de conversión del mapa de bits (la variable global “bitmap” construida a partir de la imagen) para hacerlo apto como entrada al modelo (habrá que convertir además los datos a formato FLOAT32, como se ha mencionado en el apartado 3.2. sobre la conversión del modelo Keras). Se puede ver este proceso en la ilustración 78.



```
133     //Creamos un mapa de bits con las dimensiones objetivo para la predicción
134     val resized = Bitmap.createScaledBitmap(bitmap, dstWidth: 224, dstHeight: 224, filter: true)
135     //Cargamos el modelo TensorFlow Lite
136     val model = com.FotoCheckHor.ml.Model.newInstance(this)
137
138     val tbuffer = TensorImage.fromBitmap(resized) //Creamos el Tensor a partir del mapa de bits
139     //Lo transformamos a Float dado que el modelo tflite se genera en formato Float
140     val tbufferbis = TensorImage.createFrom(tbuffer, DataType.FLOAT32)
141     val byteBuffer = tbufferbis.buffer //Obtenemos el buffer de Bytes
142
143     //Creamos el tensor de características de entrada al modelo, con las mismas características
144     //que el del mapa de bits definido anteriormente
145     val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1, 224, 224, 3), DataType.FLOAT32)
146     inputFeature0.loadBuffer(byteBuffer) //Cargamos el buffer a este tensor de entrada del modelo
147
148     //Convertimos los valores de la imagen que nos proporciona el teléfono (formato 0 a 255)
149     //para que tengan un formato Float (0 a 1) compatible con el modelo tflite
150     val imagear: MutableList<FloatArray> = ArrayList() //Hacemos una lista dinámica de vectores Float
151     //Hacemos una única operación de adición, pues es una lista de vectores. Sería inviable hacer
152     //la operación "add" para cada valor. De esta manera lo realizamos de forma eficiente.
153     imagear.add( index: 0, inputFeature0.floatArray)
154     //Convertimos los valores de la imagen tensor a los que corresponden al formato Float (0-1)
155     for (i in 0..150527) {//224X224X3
156         imagear[0][i] = imagear[0][i] / 255.0F //Normalizamos los valores
157     }
158     //Cambiamos los valores de entrada al modelo por los normalizados
159     inputFeature0.loadArray(imagear[0])
```

Ilustración 78. Proceso de conversión del mapa de bits a un tipo de información de entrada válida para el modelo.

Como se ve en la ilustración 78, se crea en primer lugar a partir de la información almacenada en la variable global “bitmap” un nuevo mapa de bits con las dimensiones objetivo del modelo, al que llamamos “resized”; este contendrá la información relativa a la imagen, con una resolución ajustada a las dimensiones objetivo 224X224X3.[75][79]
Por otro lado se carga el modelo del directorio “ml”. Aparece que existe un error, pero esto se debe a que si se importa el modelo manualmente al directorio “ml”, Android Studio identifica el modelo y permite referenciarlo como “Model”, mientras que si se descarga el modelo, a priori no es identificado por Android Studio antes de la construcción, pero una vez construida la aplicación no existe ningún problema (ilustraciones 79 y 80) .[75][80]

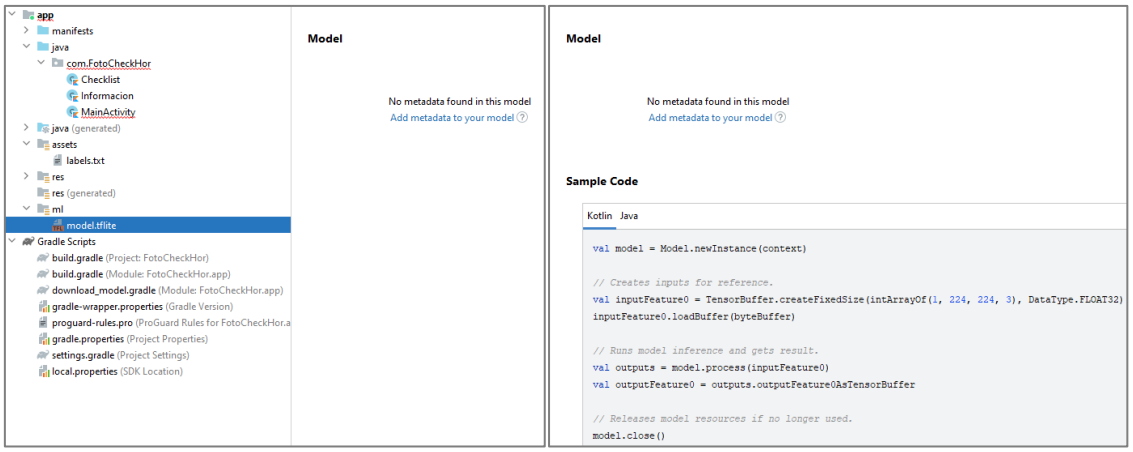
Después de cargar el modelo, se crea un tensor a partir del mapa de bits “resized”. Este, por defecto, es un tensor de datos en formato UINT8 ya que la imagen a partir de la cual se ha generado está conformada de valores UINT8. Siendo así, se crea un nuevo tensor “tbufferbis” a partir del anterior, para el que se indica en su configuración que alberga datos en formato FLOAT32.
Una vez hecho esto ya se puede obtener el buffer de bytes “byteBuffer” del tensor “tbufferbis”, buffer necesario para que el modelo adquiera la información de la imagen:[75][79][80][81]

Se crea a continuación un tensor de características de entrada al modelo “inputFeature0”, que será el tensor que finalmente se proporciona al modelo a modo de entrada. Este tensor se define con las dimensiones objetivo del modelo 224X224X3, al igual que el mapa de bits “resized” al que se ha adaptado la imagen, y se indica que los datos se encontrarán en formato FLOAT32 (recordamos que el modelo se encuentra en formato FLOAT32).
Hecho esto, se carga el buffer generado a partir de la imagen, “byteBuffer”, a este tensor de entrada al modelo “inputFeature0”.[75][80]

Una vez se tiene el tensor de entrada al modelo con la información de la imagen, es necesario no obstante normalizar los valores de esta información, ya que, aunque en formato FLOAT32, siguen

teniendo valores en el rango 0-255, que deben ser normalizados (convertidos al rango 0-1) para que se pueda obtener una predicción satisfactoria del modelo.

Para ello, atendiendo a la ilustración 78, se crea una lista dinámica de vectores "imagear", a la que se añade un vector con los valores del tensor "inputFeature0" ($224 \cdot 224 \cdot 3 = 150\,528\ valores$). Esto se ha hecho de esta manera dado que surgían muchos problemas al intentar almacenar directamente los valores del tensor en un vector. Guardando la información del tensor en un vector parte de una lista de vectores no se han producido problemas.
Finalmente se normalizan los valores del vector dividiéndolos entre 255 y se sustituyen los valores del tensor de entrada al modelo "inputFeature0" por los valores normalizados.[82][83]



Ilustraciones 79 y 80. Izquierda: se muestra cómo se visualiza el modelo en Android Studio cuando se descarga el modelo. Derecha: se muestra cómo se visualiza el modelo en Android Studio cuando se carga manualmente el modelo.

Se realiza a continuación la predicción en base al tensor de entrada "inputFeature0" y se extrae el tensor de salidas "outputs" (ilustración 81).
De este tensor se obtienen los resultados de la predicción a través de un buffer y se guardan en "outputFeature0".
Luego se hace una búsqueda de qué etiqueta (tipo de hortaliza) tiene el valor más elevado en los resultados de la predicción (esto es, con qué tipo de hortaliza queda más vinculada la imagen). Se muestra la función utilizada con este propósito en la ilustración 82, la cual está preparada para poder modificarse con objeto de obtener las tres etiquetas con mejor resultado en lugar de solamente la primera.[75][80]

Conociendo ya qué tipo de hortaliza debe indicarse como resultado al usuario, se muestra en el cuadro de texto principal la etiqueta correspondiente al tipo de hortaliza que ha obtenido el mejor resultado. Si el valor que se ha obtenido como resultado es menor que 88% se muestra el texto en rojo. Este es un valor que se ha estimado adecuado para delimitar entre buenos y malos resultados, no tiene una base teórica, pero ha resultado correcto en las pruebas experimentales que se han realizado.
Por otro lado, se muestra también por pantalla el resultado al usuario en forma de porcentaje. El porcentaje se verá rojo hasta el 88%, negro hasta el 94% y verde por encima del 94%.

```
164         val outputs = model.process(inputFeature0)
165         //Extraemos los valores de salida del modelo, los resultados de la predicción
166         val outputFeature0 = outputs.outputFeature0AsTensorBuffer
167
168         //Obtenemos a qué etiquetas corresponden los tres valores mayores
169         val max = obtenerMax(outputFeature0.floatArray)
170
171         //Mostramos en el cuadro de texto el resultado
172         text_view.setText(">> "+labels[max[0]] //Mostramos el nombre de la etiqueta con mayor puntuación
173                 + if (outputFeature0.floatArray[max[0]] < 0.88) {
174                 text_view.setTextColor(Color.RED) //Mostramos un mensaje en rojo si hay poca precisión
175                 "\n\nPoca precisión en los resultados. Inténtelo con otra fotografía."
176             }else{ //Si el resultado no tiene poca precisión se muestra el texto en negro
177             text_view.setTextColor(Color.BLACK)
178             ""
179             }
180         )
181         //Mostramos el porcentaje de precisión (accuracy) de la categoría más probable
182         porcentaje.setText((outputFeature0.floatArray[max[0]]*100).toInt().toString()
183                 +"."+((outputFeature0.floatArray[max[0]]*1000).toInt()
184                 -((outputFeature0.floatArray[max[0]] * 100).toInt() * 10)).toString()
185                 + "% ")//Si el porcentaje es superior al 94% el texto será verde
186         porcentaje.setTextColor(Color.GREEN)
187         //Si el porcentaje está entre 88 y 94% el texto será negro
188         if ((outputFeature0.floatArray[max[0]]*100).toInt()<94){
189             porcentaje.setTextColor(Color.BLACK)
190         }//Si el porcentaje es menor que 88% el texto será rojo
191         if ((outputFeature0.floatArray[max[0]]*100).toInt()<88){
192             porcentaje.setTextColor(Color.RED)
193         }
```

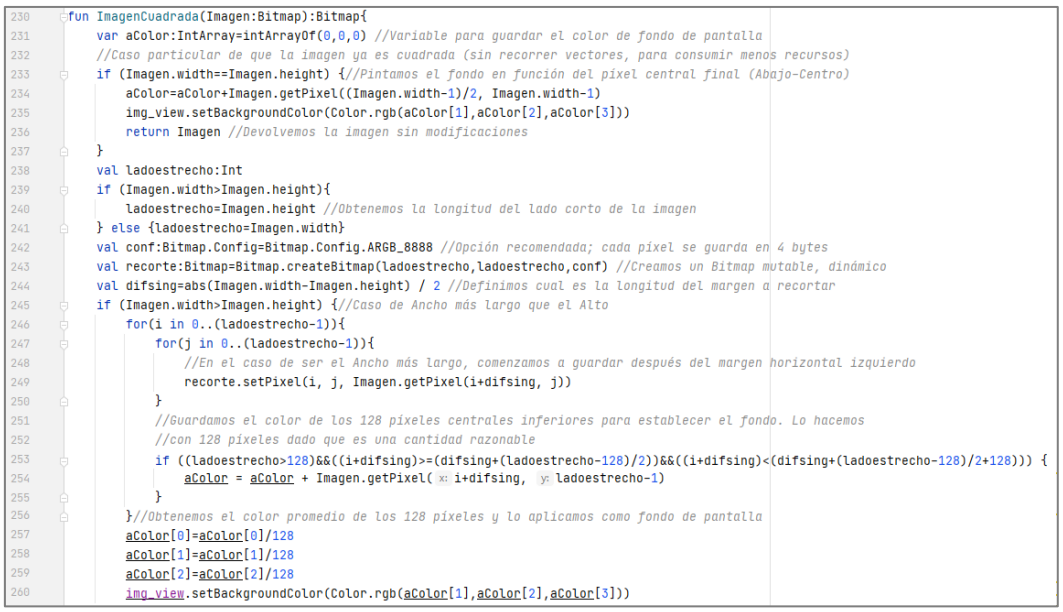
Ilustración 81. Realización de la predicción y muestra de resultados.

```
284     fun obtenerMax(arr:FloatArray) : Array<Int>{
285         //Vector que contendrá los N° de Etiqueta ordenados de mejor a peor resultado
286         val indiceArray= arrayOf(0/*,0,0*/)
287         for(i in 0..(indiceArrayF.size-1)){//Copiamos los resultados de la predicción en un vector auxiliar
288             indiceArrayF[i]=arr[i]
289         }
290         arr.sortDescending() //Ordenamos los resultados de la predicción de mejor a peor resultado
291         for(i in 0..(indiceArrayF.size-1)){
292             if(indiceArrayF[i]==arr[0]){ //Comparamos los valores de cada etiqueta con el mejor resultado
293                 indiceArray[0]=i         //y obtenemos el N° de Etiqueta del mejor resultado
294             }
295             /*if(indiceArrayF[i]==arr[1]){ //Comparación con el segundo mejor resultado
296                 indiceArray[1]=i
297             }
298             if(indiceArrayF[i]==arr[2]){ //Comparación con el tercer mejor resultado
299                 indiceArray[2]=i
300             }*/
301         }
302         return indiceArray
303     }
```

Ilustración 82. Función para conocer qué etiqueta (tipo de hortaliza) presenta mejores resultados en la predicción.

Por otro lado, la función encargada de recortar la imagen a predecir, no solamente se encarga de recortar la imagen, sino que también obtiene de esta el color de su borde inferior con el propósito de utilizarlo como fondo de pantalla (ilustración 83). Al mostrar un fondo de pantalla metamorfoseado con la imagen se hace más agradable la experiencia del usuario.

En el caso de que la imagen de entrada ya sea cuadrada solamente se toma el color del pixel central del borde inferior de la imagen (ya que no hay que recorrer ningún vector no ralentizamos el proceso más de lo necesario recorriendo el borde para obtener el color).
En caso de tener que recortar la imagen, se crea un nuevo mapa de bits con lados de igual longitud al lado estrecho de la imagen original y se copian los píxeles de la zona central de la imagen original en la nueva imagen recortada. Cuando se hace dicha operación en el borde inferior de la imagen, si esta tiene una longitud superior a 128 píxeles, se guarda el valor de los colores para obtener la media en un vector de enteros durante los 128 píxeles centrales. Se hace solamente durante 128 píxeles, dado que se trata de una cuestión estética y parece resultar suficiente la media de 128 píxeles para la obtención del color de fondo.[84][85]

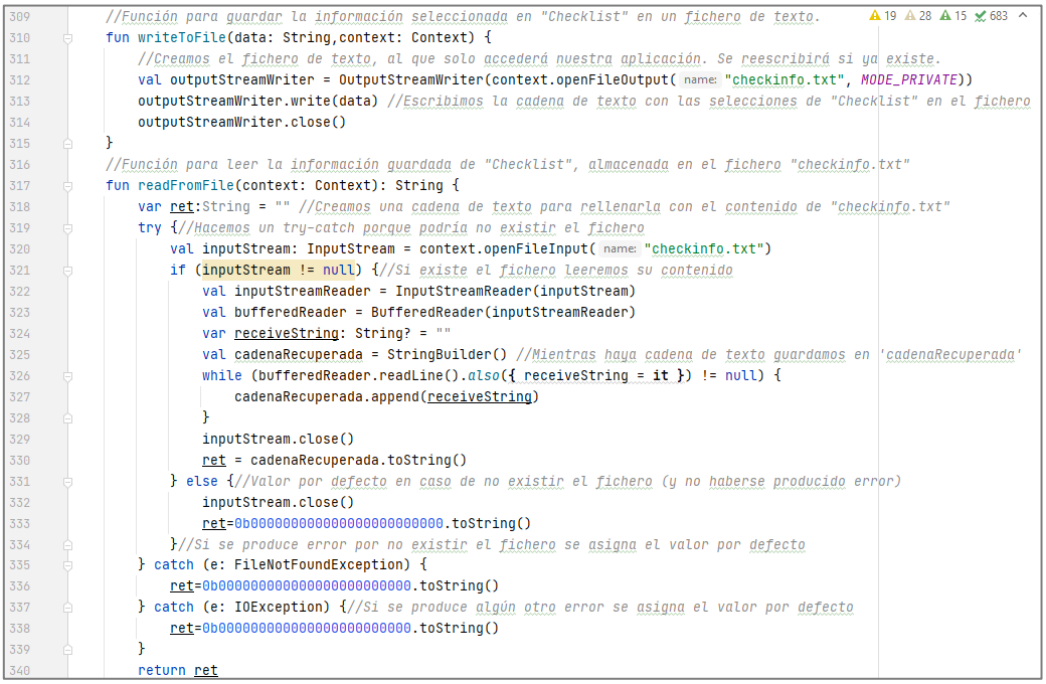


```
fun ImagenCuadrada(Imagen:Bitmap):Bitmap{
    var aColor:IntArray=intArrayOf(0,0,0) //Variable para guardar el color de fondo de pantalla
    //Caso particular de que la imagen ya es cuadrada (sin recorrer vectores, para consumir menos recursos)
    if (Imagen.width==Imagen.height) {//Pintamos el fondo en función del pixel central final (Abajo-Centro)
        aColor=aColor+Imagen.getPixel((Imagen.width-1)/2, Imagen.width-1)
        img_view.setBackgroundColor(Color.rgb(aColor[1],aColor[2],aColor[3]))
        return Imagen //Devolvemos la imagen sin modificaciones
    }
    val ladoestrecho:Int
    if (Imagen.width>Imagen.height){
        ladoestrecho=Imagen.height //Obtenemos la longitud del lado corto de la imagen
    } else {ladoestrecho=Imagen.width}
    val conf:Bitmap.Config=Bitmap.Config.ARGB_8888 //Opción recomendada; cada pixel se guarda en 4 bytes
    val recorte:Bitmap=Bitmap.createBitmap(ladoestrecho,ladoestrecho,conf) //Creamos un Bitmap mutable, dinámico
    val difsing=abs(Imagen.width-Imagen.height) / 2 //Definimos cual es la longitud del margen a recortar
    if (Imagen.width>Imagen.height) {//Caso de Ancho más largo que el Alto
        for(i in 0..(ladoestrecho-1)){
            for(j in 0..(ladoestrecho-1)){
                //En el caso de ser el Ancho más largo, comenzamos a guardar después del margen horizontal izquierdo
                recorte.setPixel(i, j, Imagen.getPixel(i+difsing, j))
            }
            //Guardamos el color de los 128 pixeles centrales inferiores para establecer el fondo. Lo hacemos
            //con 128 pixeles dado que es una cantidad razonable
            if ((ladoestrecho>128)&&((i+difsing)>=(difsing+(ladoestrecho-128)/2))&&((i+difsing)<(difsing+(ladoestrecho-128)/2+128))) {
                aColor = aColor + Imagen.getPixel( x: i+difsing,  y: ladoestrecho-1)
            }
        }//Obtenemos el color promedio de los 128 pixeles y lo aplicamos como fondo de pantalla
        aColor[0]=aColor[0]/128
        aColor[1]=aColor[1]/128
        aColor[2]=aColor[2]/128
        img_view.setBackgroundColor(Color.rgb(aColor[1],aColor[2],aColor[3]))
```

Ilustración 83. Parte de la función encargada de recortar la imagen. Además establece el color de fondo en función del borde inferior de la imagen.

También se encuentran definidas en la actividad principal las funciones de escribir y leer ficheros en memoria (la zona de memoria específica para nuestra aplicación), las cuales se utilizan desde la clase “Checklist” (pueden verse estas funciones en la ilustración 84). Se guarda y se lee siempre un fichero de texto que llamamos “checkinfo.txt”, el cual contiene toda la información relativa a la funcionalidad de control de calidad.

En el caso del guardado, simplemente se crea dicho fichero y se guarda ahí la información, mientras que para la lectura se lee la información por medio de un buffer que la concatena y devuelve esta información concatenada como salida. Se establece que el valor que se debe devolver en caso de no existir todavía el fichero sea el valor por defecto, con todas las casillas de verificación y evaluaciones del control de calidad desmarcadas.[86]



```
    //Función para guardar la información seleccionada en "Checklist" en un fichero de texto.
    fun writeToFile(data: String,context: Context) {
        //Creamos el fichero de texto, al que solo accederá nuestra aplicación. Se reescribirá si ya existe.
        val outputStreamWriter = OutputStreamWriter(context.openFileOutput( name: "checkinfo.txt", MODE_PRIVATE))
        outputStreamWriter.write(data) //Escribimos la cadena de texto con las selecciones de "Checklist" en el fichero
        outputStreamWriter.close()
    }
    //Función para leer la información guardada de "Checklist", almacenada en el fichero "checkinfo.txt"
    fun readFromFile(context: Context): String {
        var ret:String = "" //Creamos una cadena de texto para rellenarla con el contenido de "checkinfo.txt"
        try {//Hacemos un try-catch porque podría no existir el fichero
            val inputStream: InputStream = context.openFileInput( name: "checkinfo.txt")
            if (inputStream != null) {//Si existe el fichero leeremos su contenido
                val inputStreamReader = InputStreamReader(inputStream)
                val bufferedReader = BufferedReader(inputStreamReader)
                var receiveString: String? = ""
                val cadenaRecuperada = StringBuilder() //Mientras haya cadena de texto guardamos en 'cadenaRecuperada'
                while (bufferedReader.readLine().also({ receiveString = it }) != null) {
                    cadenaRecuperada.append(receiveString)
                }
                inputStream.close()
                ret = cadenaRecuperada.toString()
            } else {//Valor por defecto en caso de no existir el fichero (y no haberse producido error)
                inputStream.close()
                ret=0b00000000000000000000000000000000.toString()
            }//Si se produce error por no existir el fichero se asigna el valor por defecto
        } catch (e: FileNotFoundException) {
            ret=0b00000000000000000000000000000000.toString()
        } catch (e: IOException) {//Si se produce algún otro error se asigna el valor por defecto
            ret=0b00000000000000000000000000000000.toString()
        }
        return ret
```

Ilustración 84. Funciones encargadas de la escritura y lectura de ficheros de texto, concretamente “checkinfo.txt”.

Por último, se encuentran definidas en la actividad principal las funciones utilizadas para cambiar de pantalla vinculadas a sus respectivos botones y la función que resetea los elementos visuales “limpiarpantalla” (ilustración 85). En cuanto a las funciones de cambio de pantalla, no se ve la vinculación en la definición de las funciones porque la activación de estas está definida en la interfaz gráfica; es una alternativa al método expuesto hasta ahora en el que se definía una función “setOnClickListener”, véase la ilustración 86) Para hacer un cambio de pantalla se define y ejecuta un tipo de acción de cambio de clase en el que se indica qué clase se desea ejecutar.

```
343     //Función para ir a la página de Checklist al clicar el botón "Checklist"
344     fun to_checklist(view: View?) {
345         val intent = Intent( packageContext: this, Checklist::class.java)
346         startActivity(intent)
347     }
348
349     //Función para abrir la página de información al clicar el icono de la EUPT con la "i"
350     fun to_info(view: View?) {
351         val intent = Intent( packageContext: this, Informacion::class.java)
352         startActivity(intent)
353     }
354
355     //Función para refrescar la pantalla principal
356     @SuppressLint( ...value: "SetTextI18n")
357     fun refrescar(view: View?) {
358         limpiarpantalla()
359         val intent = Intent( packageContext: this, MainActivity::class.java)
360         startActivity(intent)
361     }
362
363     fun limpiarpantalla(){
364         img_view.setImageResource(0)  //Reseteamos la imagen mostrada
365         porcentaje.setTextColor(Color.BLACK)
366         text_view.setTextColor(Color.BLACK)
367         porcentaje.setText("00% ")    //Reseteamos el porcentaje mostrado
368         text_view.setText("Selecciona una imagen pulsando GALERÍA o bien haz una fotografía pulsando CÁMARA, luego pulsa
369     }
370 }
```

Ilustración 85. Funciones encargadas de cambiar de pantalla y de resetear los valores de la interfaz visual.

```
184     <!--Al pulsar el botón se ejecuta la función "to_checklist" de cambio de pantalla-->
185     <Button
186         android:id="@+id/irChecklist"
187         android:layout_width="wrap_content"
188         android:layout_height="48dp"
189         android:onClick="to_checklist"
190         android:text="Hacer Control de Calidad"
191         app:layout_constraintBottom_toBottomOf="@+id/Foto"
192         app:layout_constraintEnd_toEndOf="@+id/toolbarInferior"
193         app:layout_constraintStart_toStartOf="@+id/toolbarInferior"
194         app:layout_constraintTop_toTopOf="@+id/toolbarInferior"
195         app:layout_constraintVertical_bias="0.975" />
```

Ilustración 86. Botón “HACER CONTROL DE CALIDAD” definido en la interfaz gráfica (layout) de la actividad principal. Resaltado en amarillo la vinculación con la función “to_checklist”, que ejecuta la clase “Checklist” cuando se presiona el botón.

4.1.4. Funcionalidad de control de calidad

Se definen para la clase “Checklist” de control de calidad, las variables globales relativas a los elementos de la interfaz gráfica (layout) así como sus vinculaciones en la función “onCreate”, al igual que se hizo en la actividad principal (ilustraciones 87 y 88). Gran parte de estos elementos serán casillas de verificación y valoraciones por estrellas (véase la ilustración 89), cuyo valor podrá ser modificado por el usuario desde la interfaz gráfica y podrá ser guardado o restaurado. Se trata de una función cuya utilidad consiste en que el usuario pueda llevar una cuenta de las hortalizas que ya ha fotografiado, así como guardar cuál era su grado de calidad en base a su criterio personal, indicando de 0 a 3 estrellas.

```
//Elementos a vincular con los elementos de la interfaz gráfica
lateinit var guardarCheckA : CheckBox
lateinit var guardarCheckB : CheckBox
lateinit var guardarCheckC : CheckBox
lateinit var guardarCheckD : CheckBox
lateinit var guardarCheckE : CheckBox
lateinit var guardarCheckF : CheckBox
lateinit var guardarCheckG : CheckBox
lateinit var guardarCheckH : CheckBox
lateinit var ratingA : RatingBar
lateinit var ratingB : RatingBar
lateinit var ratingC : RatingBar
lateinit var ratingD : RatingBar
lateinit var ratingE : RatingBar
lateinit var ratingF : RatingBar
lateinit var ratingG : RatingBar
lateinit var ratingH : RatingBar
```

```
//Asociamos el layout (interfaz gráfica) que le corresponde a Checklist
super.onCreate(savedInstanceState)
setContentView(R.layout.checklist)

//Asociamos los botones, Checkboxes y Valoraciones (Ratings) con los elementos de la interfaz gráfica
guardarCheckA = findViewById(R.id.checkBoxA)
ratingA = findViewById(R.id.ratingBarA)
guardarCheckB = findViewById(R.id.checkBoxB)
ratingB = findViewById(R.id.ratingBarB)
guardarCheckC = findViewById(R.id.checkBoxC)
ratingC = findViewById(R.id.ratingBarC)
guardarCheckD = findViewById(R.id.checkBoxD)
ratingD = findViewById(R.id.ratingBarD)
guardarCheckE = findViewById(R.id.checkBoxE)
ratingE = findViewById(R.id.ratingBarE)
guardarCheckF = findViewById(R.id.checkBoxF)
ratingF = findViewById(R.id.ratingBarF)
guardarCheckG = findViewById(R.id.checkBoxG)
ratingG = findViewById(R.id.ratingBarG)
guardarCheckH = findViewById(R.id.checkBoxH)
```

Ilustraciones 87 y 88. Izquierda: declaración de variables globales en “Checklist.kt”. Derecha: asociación de la clase “Checklist” con su layout y de las variables globales con sus respectivos elementos de la interfaz gráfica.

Ilustración 89. Muestra de cómo se ve la actividad de la funcionalidad de control de calidad (“Checklist”) en un dispositivo de 6.5 pulgadas con orientación Horizontal.

El modo en que se manipula la información ha sido diseñado de tal manera que se guardan valores verdaderos o falsos y valores de 0 a 3 (00, 01, 10, 11) en posiciones determinadas de una cadena de texto en formato binario.

De esta manera, cada evaluación de 0 a 3 estrellas ocupa 2 bits en la cadena y cada casilla de verificación, que puede tener valor verdadero o falso 1 bit. Se define, pues, una cadena de texto binaria 0b000000000000000000000000 llamada “CodigoChecklist”, de 24 valores para los 8 tipos de hortalizas para los que se puede guardar esta información.

Para guardar en “CodigoChecklist” el valor de una casilla de verificación cuando esta es marcada/desmarcada, se realiza una operación XOR sobre “CodigoChecklist” con el valor 1 activado en el lugar correspondiente a la casilla de verificación. De esta manera, el valor correspondiente a dicha casilla en “CodigoChecklist” alternará entre 1 y 0 dependiendo de si se marca o se desmarca la casilla (ilustración 90).

```
guardarCheckA.setOnClickListener(View.OnClickListener { it: View!
    CodigoAuxiliar = 0b000000000000000000000100
    CodigoChecklist = CodigoAuxiliar.xor(CodigoChecklist)
})
```

Ilustración 90. Marcado/desmarcado mediante una operación XOR de la casilla de verificación “A” en “CodigoChecklist” al clicar sobre la casilla.

Cuando se presiona el botón “GUARDAR” lo que se hace es guardar esta cadena “CodigoChecklist” en memoria, en el fichero “checkinfo.txt”, por medio de la función definida con anterioridad en la actividad principal. Si bien antes de guardar el contenido de esta cadena, deben ser añadidos a esta los valores de la valoración por estrellas de cada hortaliza (ilustración 91).
Por último, una vez guardada la información, se resetean los valores de los diferentes elementos, tanto de las variables como los marcados que se muestran por interfaz gráfica, para facilitar al usuario realizar una nueva evaluación desde cero.

```
Guardar.setOnClickListener(View.OnClickListener {//Acción al clicar el botón "Guardar"
    val cA: Int = parseInt(toBinaryString(ratingA.getRating().roundToInt()).plus( other: ""), radix: 2)
    val cB: Int = parseInt(toBinaryString(ratingB.getRating().roundToInt()).plus( other: "00"), radix: 2)
    val cC: Int = parseInt(toBinaryString(ratingC.getRating().roundToInt()).plus( other: "000000"), radix: 2)
    val cD: Int = parseInt(toBinaryString(ratingD.getRating().roundToInt()).plus( other: "00000000"), radix: 2)
    val cE: Int = parseInt(toBinaryString(ratingE.getRating().roundToInt()).plus( other: "000000000000"), radix: 2)
    val cF: Int = parseInt(toBinaryString(ratingF.getRating().roundToInt()).plus( other: "0000000000000000"), radix: 2)
    val cG: Int = parseInt(toBinaryString(ratingG.getRating().roundToInt()).plus( other: "00000000000000000000"), radix: 2)
    val cH: Int = parseInt(toBinaryString(ratingH.getRating().roundToInt()).plus( other: "000000000000000000000000"), radix: 2)
    //Hacemos una suma, si hicieramos un XOR se cambiarian todos los valores cada vez
    CodigoValoraciones = cA + cB + cC + cD + cE + cF + cG + cH
    CodigoChecklist = CodigoValoraciones + CodigoChecklist
    //Escribimos la Cadena en un fichero (la función se encuentra en el MainActivity), escribe en checkinfo.txt
    MainActivity().writeToFile(toBinaryString(CodigoChecklist), getApplicationContext())
```

Ilustración 91. Al presionar "GUARDAR", obtención de los valores de la valoración por estrellas y guardado de toda la información (incluyendo la relativa a las casillas de verificación) en el fichero "checkinfo.txt".

Si se presiona el botón "RESTAURAR" se carga la cadena de texto guardada por medio de la función que lee el fichero "checkinfo.txt" definida en la actividad principal. Luego se realiza una adaptación, dado que la información recuperada no tiene por qué ocupar los 24 valores de la cadena original de información, puede haberse reducido automáticamente su tamaño si contenía muchos ceros a la izquierda o bien puede haberse guardado con más ceros.

De esta manera, se eliminan o se añaden ceros a la izquierda de la cadena recuperada para que esta tenga un tamaño de 24 bits (ilustración 92).
Partiendo de la cadena de texto corregida se extraen los valores de los diferentes elementos y se aplican a las variables si corresponde así como a la visualización de los elementos en la interfaz gráfica.

```
    Restaurar.setOnClickListener(View.OnClickListener { it: View!
        //Posición+(Longitud Cadena Guardada-24)=Posición'
        //Leemos el fichero (la función está contenida en MainActivity), abre el txt con la Cadena de configuraciones
        var Cadenatxt:String= MainActivity().readFromFile(getApplicationContext())
        //Eliminamos los elementos adicionales, ya que por defecto contine espacios
        Cadenatxt=Cadenatxt.replace( oldValue: " ", newValue: "")
        var AuxS:String="" //Definimos una variable para recuperar correctamente la información de la Cadena
        val Tam:Int=Cadenatxt.length //Obtenemos la longitud de la Cadena guardada
        //Si es más corta añadimos ceros hasta los 24 valores y así poder obtener los valores por índice
        if ((24-Tam)>0){
            for(i in 1..(24-Tam)){
                AuxS=AuxS.plus( other: "0")
            }//Añadimos los ceros a la izquierda para completar los 24 valores de información
            AuxS=AuxS.plus(Cadenatxt)
        } else{//Si la Cadena es más grande ignoramos los primeros valores
            AuxS=Cadenatxt[Tam-24].toString()
            for(j in 1..23) {
                AuxS=AuxS.plus(Cadenatxt[Tam-24+j].toString())
            }
        }//Leemos los Ratings de la Cadena corregida
        //No podemos transformar directamente Char a Número, debemos convertirlo primero a String
        //00=0(0*2+0); 01=1(0*2+1); 10=2(1*2+0); 11=3(1*2+1)
        ratingA.setRating(AuxS[22].toString().toFloat()*2+AuxS[23].toString().toFloat())
                                        ● ● ●
        //Si no lo reseteáramos, en lugar de cambiar la información, invertiríamos la
        //información de los Checklists borrando la marca cuando correspondiese indicarla
        CodigoChecklist=0b00000000000000000000000
        CodigoAuxiliar =0b00000000000000000000000

        if (AuxS[21]=='1'){ //Comprobamos uno por uno si los Checklists en la Cadena están marcados
            guardarCheckA.isChecked=true //Marcamos la casilla en la interfaz gráfica
            CodigoAuxiliar = CodigoAuxiliar+0b00000000000000000000100
        }else{guardarCheckA.isChecked=false}
                                        ● ● ●
        //Hacemos la operación XOR con la Cadena que ya contine los Ratings
        CodigoChecklist = CodigoAuxiliar.xor(CodigoChecklist)
        //Debug.- Para ver qué Cadena de texto resulta
        //textView.text = (AuxS.toString())
    })
}
```

Ilustración 92. Recuperación y adaptación de la cadena de texto con la información del control de calidad cuando se presiona "RESTAURAR".

Por último, se tiene la función de cambio de página (ilustración 93), cuyo efecto es volver a la pantalla principal (se ejecuta cuando se presiona el icono con la flecha hacia atrás), funciona de la misma manera que las funciones de cambio de página expuestas en la actividad principal.

```
219     //Función para volver a la página principal al clicar el icono de la EUPT con una flecha hacia atrás
220     fun to_mainpage(view: View?) {
221         val intent = Intent( packageContext: this, MainActivity::class.java)
222         startActivity(intent)
223     }
224 }
```

Ilustración 93. Función de cambio de página para volver a la pantalla principal.

4.1.5. Tutorial de la aplicación

En la clase encargada de mostrar el tutorial sobre cómo usar la aplicación (la clase “Informacion”) solamente será necesario indicar la interfaz gráfica con que debe ser vinculada y la función de cambio de página para el botón de volver a la pantalla principal. No es necesario definir nada más porque las imágenes que muestran el tutorial (se han utilizado varias imágenes porque no era posible utilizar una sola imagen por la limitación existente en el tamaño máximo con que se puede mostrar una imagen) se encuentran definidas en el layout (interfaz gráfica) de la clase (ilustración 94).[87]

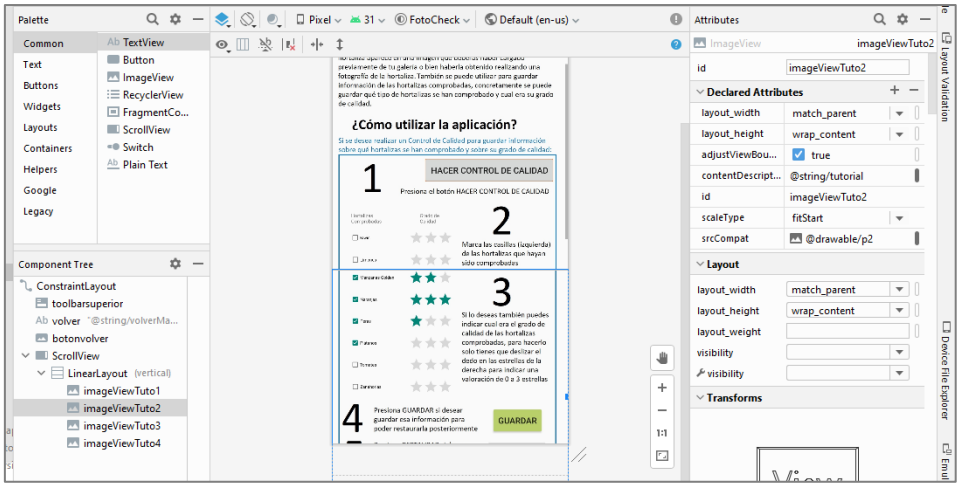


Ilustración 94. Muestra de cómo se han definido cuatro imágenes diferentes para mostrar el tutorial en la interfaz gráfica destinada a tal efecto.

4.1.6. Interfaz gráfica

En el directorio de layout se definen las diferentes interfaces gráficas (ilustración 95). Para las interfaces gráficas de la actividad principal (“activity_main”) y del control de calidad (“checklist”), se han definido interfaces personalizadas para el caso de orientación vertical, orientación horizontal y para el caso de tablets.

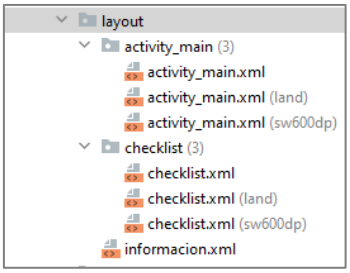


Ilustración 95. Directorio layout, que contiene la definición de las interfaces gráficas. Visto desde Android Studio.

Se han realizado los diseños de tal manera que el posicionamiento sea siempre relativo a otros elementos y ajustando el tamaño de los elementos a su contenido cuando era necesario. También se han realizado pruebas con diferentes tamaños de pantalla y diferentes orientaciones para corregir cualquier problema con el posicionamiento de los elementos que pudiera empeorar la experiencia del usuario.

Se muestra a continuación la previsualización de las interfaces vertical y horizontal de la actividad principal (pantalla principal) y de la funcionalidad de control de calidad para una pantalla de 5 pulgadas (ilustración 96):

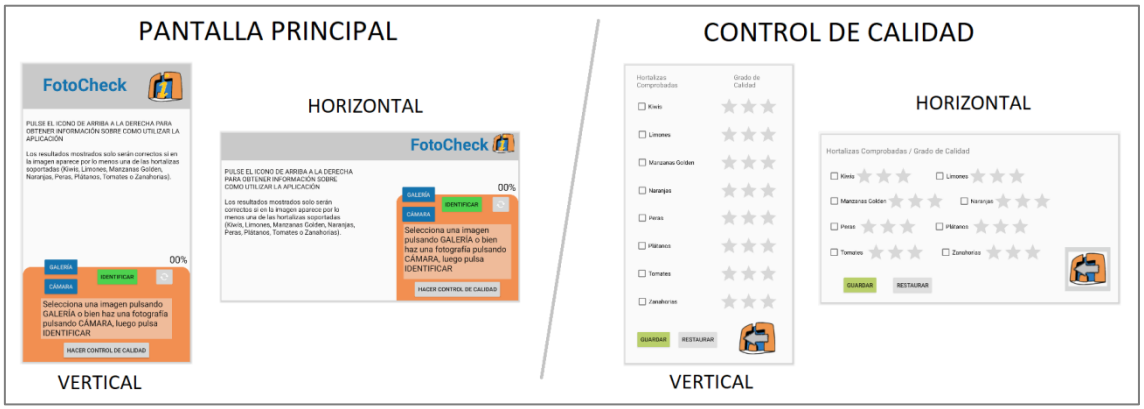


Ilustración 96. Previsualización de las interfaces vertical y horizontal de la actividad principal y de la funcionalidad de control de calidad para una pantalla de 5 pulgadas.

4.2. Publicación de la aplicación de acceso anticipado en Google Play Store e impresión 3D de una moneda con código QR para acceder a la página de la tienda

Un paso importante una vez se ha creado la aplicación es hacerla llegar a los usuarios, por lo que resulta de gran interés hacer que la aplicación esté disponible en la tienda de aplicaciones Android, Google Play Store.

4.2.1. Publicación de la aplicación de acceso anticipado disponible para testers

Para publicar la aplicación se ha utilizado Google Play Console. Deberá hacerse una configuración inicial de la aplicación que va a publicarse indicando el nombre (FotoCheck Hortalizas), la categoría a la que pertenece (la categoría de herramientas), así como respondiendo una serie de preguntas para obtener una clasificación por edades, etc.[88]

Por otro lado, se configura la ficha de la página de nuestra aplicación en la tienda, añadiendo una descripción de la aplicación y aportando capturas de pantalla de la aplicación para mostrarlas en la página (ilustración 97).

La aplicación puede publicarse en la tienda de diferentes maneras. Pueden realizarse pruebas internas, cerradas o abiertas. Finalmente se puede optar por "enviar la aplicación a producción", que es la publicación oficial de la aplicación.

Las pruebas internas se hacen con intención de probar la aplicación entre miembros del equipo de diseño de la aplicación, solo puede ser distribuida a 100 testers.
Las pruebas cerradas son similares a las anteriores, se puede distribuir la aplicación a una mayor cantidad de testers, cuyo correo electrónico debe figurar en el listado de testers de la aplicación.
Por último están las pruebas abiertas, que consiste en que cualquiera puede unirse a las pruebas por medio de Google Play Store. La aplicación entonces se encuentra en estado de acceso anticipado. Puede fijarse un límite de testers de la aplicación.[89]

De esta manera, decidimos publicar la aplicación en pruebas abiertas (también se pueden hacer diferentes tipos de pruebas simultáneamente). Para cargar la aplicación a Google Play Console debe crearse desde Android Studio lo que se llama un App Bundle firmado. Luego se carga, se indica qué novedades trae esta versión y se ordena publicar.

Quedará sujeto a revisión durante varios días y una vez aprobado podrán acceder todos los usuarios de Google Play Store de los países habilitados (se ha habilitado la aplicación para todos los países hispanohablantes) que deseen probar la aplicación (ilustración 98).

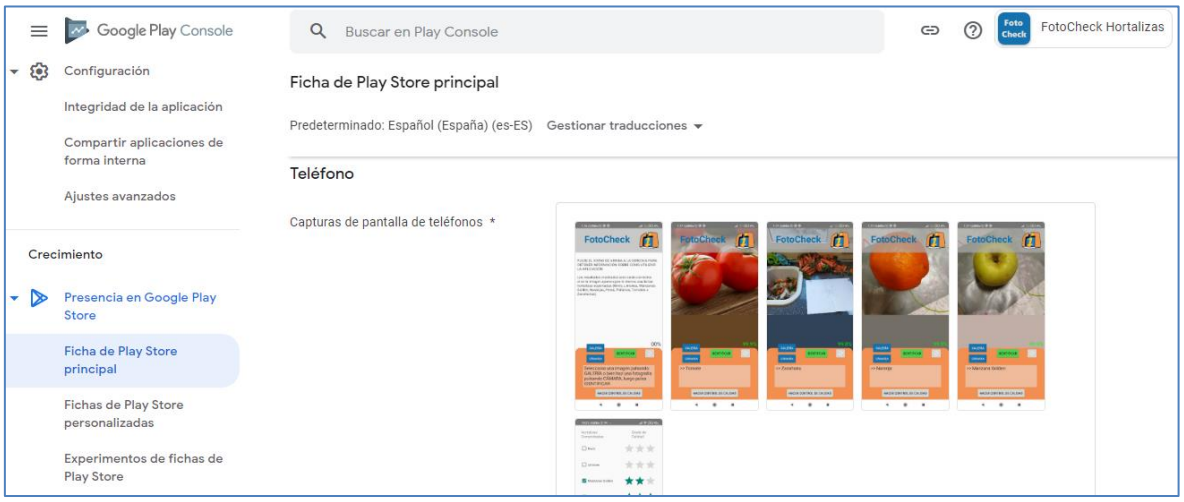


Ilustración 97. Ficha de la aplicación vista desde Google Play Console.

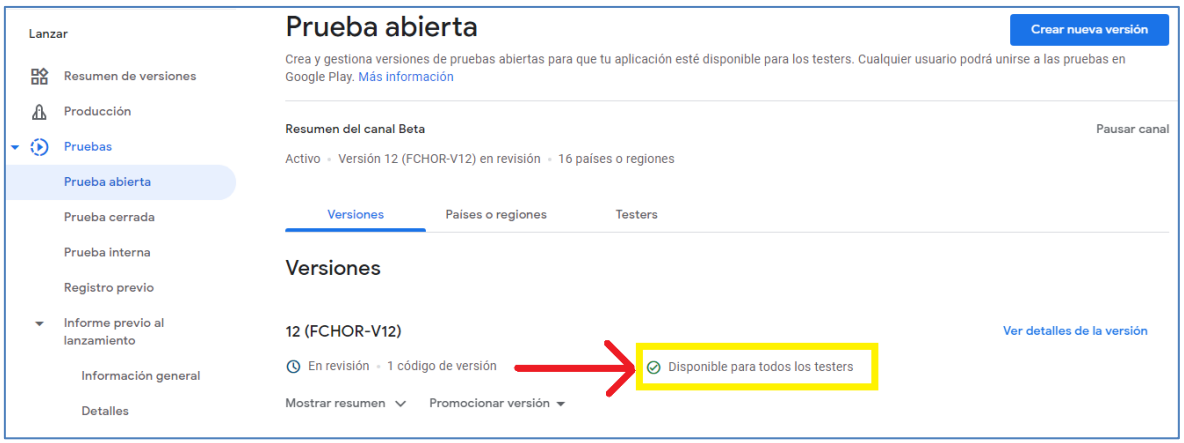
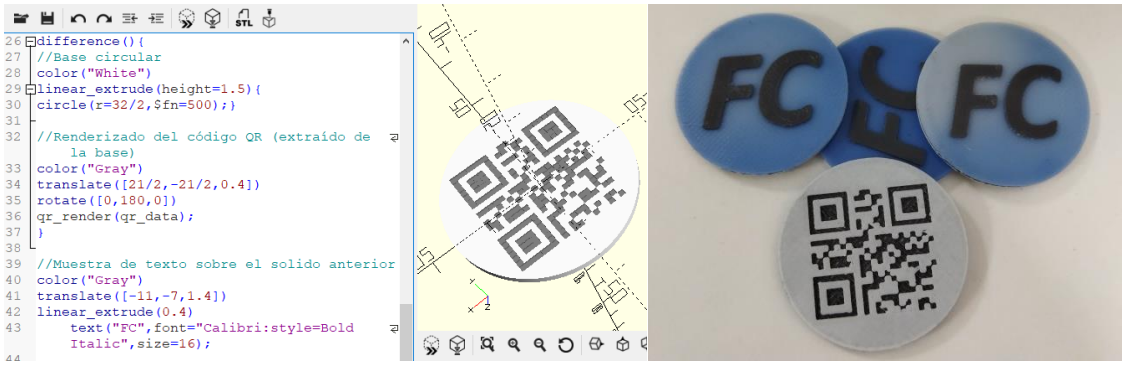


Ilustración 98. Panel de control de pruebas abiertas. Se ve cómo recién cargado el App Bundle aparecerá que se encuentra en revisión. Se muestra en un recuadro amarillo cómo aparecerá en el panel de control cuando se haya revisado y aprobado y esté disponible para todos los usuarios que quieran probar la aplicación.

4.2.2. Impresión 3D de una moneda para acceder a la página de la aplicación

Adicionalmente se ha diseñado con OpenSCAD una moneda que contiene un código QR que nos llevará a través de un acortador de enlace a la página de FotoCheck Hortalizas en Google Play Store. Se utiliza un acortador de enlace porque así se puede hacer el código QR de menor tamaño, y por lo tanto puede imprimirse mejor; además, de esta manera se puede cambiar la página de destino en cualquier momento, algo muy útil una vez ya se encuentra impresa la moneda.[90]

A la hora de imprimir la moneda en 3D resulta imprescindible que la parte exterior al código QR sea clara y la parte interior oscura, por lo que se ha decidido imprimirlo en blanco y gris respectivamente. Por otro lado, en la otra cara de la moneda imprimiremos las iniciales de la aplicación, “FC”, en gris, con un fondo azul, como en el icono de la aplicación. Si se escanea el código del diseño o de la fotografía de la moneda (ilustraciones 99 y 100) puede observarse como en efecto nos redirige a la página de FotoCheck Hortalizas a través del acortador de enlace https://acortar.link/FChe.



Ilustraciones 99 y 100. Izquierda: Diseño de la moneda en OpenSCAD. Derecha: Monedas impresas.

Las monedas se han impreso con PLA 850 orgánico biodegradable (el PLA 850 es un tipo de PLA mejorado, con mayor resistencia térmica y mecánica que el PLA normal).[91]
Se han impreso en 5 capas diferentes de PLA utilizando filamento de color blanco, de color gris y filamento que cambia de color de azul a blanco a los 30ºC.

5. Conclusiones

Se ha diseñado una aplicación que ya se encuentra disponible en fase de acceso anticipado para todos los usuarios Android de todos los países hispanohablantes del mundo, que simplemente con buscar en Google Play Store "Hortalizas" o "FotoCheck", pueden encontrar la aplicación y descargarla, para pasar a formar parte de las pruebas internas de la aplicación.
Esta aplicación es capaz de identificar exitosamente entre 8 tipos diferentes de hortaliza en base a un modelo basado en redes neuronales que se ha creado a partir de otro modelo ya entrenado utilizando la técnica conocida como Transfer-Learning. La implementación que se ha diseñado puede adaptarse con facilidad para su uso con un número diferente de elementos, que no tienen por qué ser hortalizas, lo que ofrece una ilimitada cantidad de posibilidades.
Para reunir los datos necesarios para la construcción de un modelo de identificación de hortalizas robusto para la aplicación de identificación de hortalizas que se ha desarrollado, se han realizado más de diez mil fotografías de hortalizas, que han permitido crear un modelo con una precisión en el proceso de clasificación de hortalizas superior al 99%.

La realización de este trabajo de fin de grado ha revestido variadas dificultades, si bien muchas de ellas se han podido solucionar más rápidamente que otras gracias a toda la información existente en diferentes y variadas fuentes de información.

La realización de fotografías conllevó un trabajo de varios días, si bien no fue algo especialmente difícil. Sí que revistió más complicación la recopilación de imágenes de internet, dado que hubo que hacer muchos filtrados para eliminar imágenes con otro contenido, transparentes, repetidas, alargadas….
Por otro lado, lo que supuso la mayor inversión de tiempo en la realización del trabajo fue la búsqueda de la mejor manera de realizar el entrenamiento del modelo, y hacer una conversión de este al formato apto para Android que funcionara correctamente desde la aplicación.
La aplicación Android por su parte, ha resultado ser la parte en la que mayor cantidad de problemas han surgido, si bien ha resultado ser la parte menos problemática dado que resultaba agradable trabajar con Android Studio y además, a base de trabajar sobre ello podían solucionarse con rapidez todos los problemas que iban surgiendo gracias a toda la información disponible.

Como posibles mejoras que se podrían hacer en la aplicación cabe destacar la posibilidad de realizar una clasificación en tiempo real, sin necesidad de realizar una fotografía. Esto podría llegar a consumir muchos recursos, pero sería viable realizando una clasificación cada 5 segundos, por ejemplo.

Otra posible mejora sería hacer una adaptación para poder identificar otros tipos de elementos, alguno de los expuestos en la introducción, o, por poner otro ejemplo, se podrían recopilar decenas de miles de fotos de diferentes paisajes (árboles, arbustos, campos de cultivo…) donde podrá apreciarse en qué estación nos encontramos, otoño, verano, invierno…, y no solo podría ser posible identificar la estación en que nos encontramos, sino que se podría intentar incluso, que a partir de una foto de un rosal silvestre realizada en primavera, se pudiera conocer que las aves más probables de encontrar ahí pueden ser mirlos, ruiseñores y mosquiteros, por ejemplo, o que, de una foto de una laguna, sea probable encontrar avefrías, grullas, garzas, aguiluchos pálidos.... De esta manera se podría llegar a conocer qué fauna puede verse afectada si se deseca una laguna, en

qué época del año puede hacerse esto sin perjudicar a la fauna, qué viñas pueden verse más afectadas por cuervos o dónde es más conveniente colocar molinos de viento, por ejemplo.

Se puede acceder a los archivos del trabajo de fin de grado a través del siguiente enlace:
https://drive.google.com/drive/folders/19nsob2mBFj5sESvhUnUpZPBisTZoVRvc?usp=sharing

Referencias bibliográficas

[1] Zhu, Z., Shi, D., Guankai, P. et al. «Retinal age gap as a predictive biomarker for mortality risk», *British Journal of Ophthalmology*. <https://bjo.bmj.com/content/early/2021/11/17/bjophthalmol-2021-319807> [Consulta: 28/01/2022].

[2] Saari, C. «iNaturalist Computer Vision Explorations», *iNaturalist*. <https://www.inaturalist.org/pages/computer_vision_demo> [Consulta: 28/01/2022].

[3] «GLOBE Clouds and Matched Satellite Data», *GLOBE Observer*. <https://observer.globe.gov/get-data/clouds-data> [Consulta: 29/01/2022].

[4] Muñoz Pérez, L. M. «Creación de un Predictor Climático basado en Deep Learning», *Noticiario Climático*. <https://noticiarioclimatico.blogspot.com/2021/12/creacion-de-un-predictor-climatico.html> [Consulta: 28/01/2022].

[5] Russell, S., Norvig, P. *Artificial Intelligence: A Modern Approach (3rd edition)*. Malaysia, Pearson, 2016.

[6] «Cuota de mercado de smartphones», *Kantar*. <https://www.kantar.com/es/campaigns/cuota-de-mercado-de-smartphones> [Consulta: 27/01/2022].

[7] «Android Studio», *Wikipedia*. <https://en.wikipedia.org/wiki/Android_Studio> [Consulta: 27/01/2022].

[8] READERA LLC «ReadEra - lector de libros pdf, epub, word», *Google Play*. <https://play.google.com/store/apps/details?id=org.readera> [Consulta: 01/02/2022].

[9] Malcolm, D. «We show weather simulation data!», *FLOWX*. <https://www.flowx.io/> [Consulta: 01/02/2022].

[10] Elevate Labs «Stay sharp, build confidence, and boost productivity.», *ELEVATE*. <https://elevateapp.com/> [Consulta: 01/02/2022].

[11] The Cornell Lab of Ornithology «Discover a new world of birding...», *eBird*. <https://ebird.org/home> [Consulta: 01/02/2022].

[12] Loarie, S., Ueda, K., Leary, P. et al. «iNaturalist», *iNaturalist*. <https://www.inaturalist.org> [Consulta: 01/02/2022].

[13] Miccron «CoinDetect: Reconocer monedas de euro», *Play Store*. <https://play.google.com/store/apps/details?id=com.miccron.coindetect> [Consulta: 01/02/2022].

[14] Plantix «Plantix - doctor de cultivos», *Play Store*. <https://play.google.com/store/apps/details?id=com.peat.GartenBank> [Consulta: 01/02/2022].

[15] Google «Busca lo que ves», *Google Lens*. <https://lens.google/> [Consulta: 01/02/2022].

[16] López Torres, A. M. *Apuntes de la Asignatura Visión por Computador (2020-2021)*.

[17] «F1Score», *TensorFlow*. <https://www.tensorflow.org/addons/api_docs/python/tfa/metrics/F1Score> [Consulta: 28/01/2022].

[18] Sumernok, P. «Estimating an Optimal Learning Rate for a Deep Neural Network», *Towards data science*. <https://towardsdatascience.com/estimating-optimal-learning-rate-for-a-deep-neural-network-ce32f2556ce0> [Consulta: 05/02/2022].

[19] «Loss functions for classification», *Wikipedia*. <https://en.wikipedia.org/wiki/Loss_functions_for_classification> [Consulta: 03/02/2022].

[20] «Loss function», *Wikipedia*. <https://en.wikipedia.org/wiki/Loss_function> [Consulta: 03/02/2022].

[21] Goodfellow, I. et al. *Deep Learning*. MIT Press, 2016.

[22] Parmar, R. «Common Loss functions in machine learning», *Towards data science*. <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23> [Consulta: 03/02/2022].

[23] «Aprendizaje automático», *Wikipedia*. <https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico> [Consulta: 03/02/2022].

[24] Doshi, S. «Various Optimization Algorithms For Training Neural Network», *Towards data science*. <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6> [Consulta: 05/02/2022].

[25] «Overfitting», *Wikipedia*. <https://en.wikipedia.org/wiki/Overfitting> [Consulta: 03/02/2022].

[26] «Red neuronal artificial», *Wikipedia*. <https://es.wikipedia.org/wiki/Red_neuronal_artificial> [Consulta: 03/02/2022].

[27] «Explain Activation Function in Neural Network and its types», *i2tutorials*. <https://www.i2tutorials.com/explain-activation-function-in-neural-network-and-its-types/> [Consulta: 04/02/2022].

[28] Freire, E., Silva, S. «Redes Neuronales», *Bootcamp AI*. <https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb> [Consulta: 05/02/2022].

[29] Torres, J. «Deep Learning», *Jordi TORRES AI*. <https://torres.ai/deep-learning-inteligencia-artificial-keras/> [Consulta: 04/02/2022].

[30] «Reconocimiento de imágenes: Ingenio humano tras la Inteligencia Artificial», *INTELIGENCIA FUTURA*. <https://inteligenciafutura.mx/blog/reconocimiento-de-imagenes-ingenio-humano-tras-la-inteligencia-artificial> [Consulta: 04/02/2022].

[31] Versloot, C. «What are Max Pooling, Average Pooling, Global Max Pooling and Global Average Pooling?», *MachineCurve*. <https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/#pooling-layers-in-the-keras-api> [Consulta: 05/02/2022].

[32] Oltean, M. «Fruits 360: A dataset with 90380 images of 131 fruits and vegetables», *Kaggle*. <https://www.kaggle.com/moltean/fruits/version/22> [Consulta: 10/01/2022].

[33] «Búsqueda avanzada de imágenes», *Google*. <https://www.google.es/advanced_image_search> [Consulta: 29/01/2022].

[34] Mobile First LLC «Download All Images», *Chrome web store*. <https://chrome.google.com/webstore/detail/download-all-images/ifipmflagepipjokmbdecpmjbibjnakm> [Consulta: 29/01/2022].

[35] «Download Find.Same.Images.OK 4.77», *SoftwareOK*. <https://www.softwareok.com/?Download=Find.Same.Images.OK> [Consulta: 29/01/2022].

[36] «The Scientific Python Development Environment», *Anaconda*. <https://anaconda.org/anaconda/spyder> [Consulta: 30/01/2022].

[37] Pajares Martinsanz, G., de la Cruz García, J. M. *Visión por Computador: Imágenes digitales y aplicaciones (2º edición)*. Madrid, RA-MA, 2007.

[38] Culfat, F. «Transfer Learning using Mobilenet and Keras», *Towards data science*. <https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299> [Consulta: 30/01/2022].

[39] «MobileNet and MobileNetV2», *Keras*. <https://keras.io/api/applications/mobilenet/> [Consulta: 05/02/2022].

[40] «Keras with MobilenetV2 for Deep Learning», *RidgeRun*. <https://developer.ridgerun.com/wiki/index.php?title=Keras_with_MobilenetV2_for_Deep_Learning> [Consulta: 05/02/2022].

[41] «MobileNet initialization fails for small input_shape», *GitHub*. <https://github.com/keras-team/keras/issues/8090> [Consulta: 05/02/2022].

[42] «Python keras.applications.MobileNet() Examples», *Program Creek*. <https://www.programcreek.com/python/example/112461/keras.applications.MobileNet> [Consulta: 05/02/2022].

[43] Ponce Cruz, P. *Inteligencia Artificial con Aplicaciones a la Ingeniería*. Barcelona, Alfaomega, 2011.

[44] Wood, T. «What is the Softmax Function?», *DeepAI*. <https://deepai.org/machine-learning-glossary-and-terms/softmax-layer> [Consulta: 05/02/2022].

[45] Jarvis, D. «Image Interactor, understanding keras ImageDataGenerator», *Medium*. <https://medium.com/@jarvissan21/image-interactor-understanding-keras-imagedatagenerator-3afd09a4a41d> [Consulta: 05/02/2022].

[46] Sharma, S. «Epoch vs Batch Size vs Iterations», *Towards data science*. <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9> [Consulta: 08/02/2022].

[47] Brownlee, J. «Difference between a Batch and an Epoch in a Neural Network», *Machine Learning Mastery*. <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/> [Consulta: 06/02/2022].

[48] Brownlee, J. «How to Control the Stability of Training Neural Networks With the Batch Size», *Machine Learning Mastery*. <https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/> [Consulta: 06/02/2022].

[49] Shen, K. «Effect of batch size on training dynamics», *Medium*. <https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e> [Consulta: 06/02/2022].

[50] «Dropout layer», *Keras*. <https://keras.io/api/layers/regularization_layers/dropout/> [Consulta: 06/02/2022].

[51] «Keras Dropout with noise_shape», *Stack Overflow*. <https://stackoverflow.com/questions/46585069/keras-dropout-with-noise-shape> [Consulta: 06/02/2022].

[52] «Image data preprocessing», *Keras*. <https://keras.io/api/preprocessing/image/> [Consulta: 08/02/2022].

[53] «shuffle», *Kite*. <https://www.kite.com/python/docs/keras.backend.moving_averages.distribution_strategy_context.distribute_lib.dataset_ops.BatchDataset.shuffle> [Consulta: 08/02/2022].

[54] «Very good validation accuracy but bad predictions», *Stack Overflow*. <https://stackoverflow.com/questions/56815476/very-good-validation-accuracy-but-bad-predictions> [Consulta: 06/02/2022].

[55] Khandelwal, R. «A Basic Introduction to TensorFlow Lite», *Towards data science*. <https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292> [Consulta: 06/02/2022].

[56] Google «Teachable Machine», *Teachable Machine*. <https://teachablemachine.withgoogle.com/train/image> [Consulta: 09/11/2021].

[57] «Cuantización de enteros posterior al entrenamiento», *TensorFlow*. <https://www.tensorflow.org/lite/performance/post_training_integer_quant> [Consulta: 06/02/2022].

[58] The TensorFlow Authors «Post-training integer quantization», *Google Colab*. <https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/lite/g3doc/performance/post_training_integer_quant.ipynb> [Consulta: 06/02/2022].

[59] Roeder, L. «Netron», *Netron*. <https://netron.app/> [Consulta: 29/11/2021].

[60] «Conversor de TensorFlow Lite», *TensorFlow*. <https://www.tensorflow.org/lite/convert> [Consulta: 06/02/2022].

[61] «Cuantización posterior al entrenamiento», *TensorFlow*. <https://www.tensorflow.org/lite/performance/post_training_quantization> [Consulta: 06/02/2022].

[62] «tf.lite.Optimize», *TensorFlow*. <https://www.tensorflow.org/api_docs/python/tf/lite/Optimize> [Consulta: 08/02/2022].

[63] «TensorFlow model optimization», *TensorFlow*. <https://www.tensorflow.org/model_optimization/guide> [Consulta: 08/02/2022].

[64] «Desarrolla apps para Android con Kotlin», *Android Developers*. <https://developer.android.com/kotlin> [Consulta: 07/02/2022].

[65] «Cómo tomar fotos», *Android Developers*. <https://developer.android.com/training/camera/photobasics> [Consulta: 07/02/2022].

[66] «FileProvider», *Android Developers*. <https://developer.android.com/reference/androidx/core/content/FileProvider> [Consulta: 07/02/2022].

[67] «App crashes after taking photo with Android camera app in different orientation than it was started», *Stack Overflow*. <https://stackoverflow.com/questions/50468129/app-crashes-after-taking-photo-with-android-camera-app-in-different-orientation> [Consulta: 07/02/2022].

[68] «Android: Camera Crash after taking landscape picture», *Stack Overflow*. <https://stackoverflow.com/questions/43045024/android-camera-crash-after-taking-landscape-picture> [Consulta: 07/02/2022].

[69] «Links», *Acortar.link*. <https://acortar.link/admin#links> [Consulta: 08/02/2022].

[70] «Acceso sencillo y seguro a todo tu contenido», *Google Drive*. <https://www.google.com/intl/es_es/drive/> [Consulta: 08/02/2022].

[71] «Upload, share and manage your files for free.», *USAupload*. <https://usaupload.com/> [Consulta: 08/02/2022].

[72] «FILE.re - Temporary file hosting», *FILE.re*. <https://file.re/> [Consulta: 08/02/2022].

[73] «Filebin», *Filebin*. <https://filebin.net/> [Consulta: 08/02/2022].

[74] «OpenDrive», *OpenDrive*. <https://www.opendrive.com/files/> [Consulta: 08/02/2022].

[75] TensorFlow «TensorFlow Examples», *GitHub*. <https://github.com/tensorflow/examples> [Consulta: 08/02/2022].

[76] del Vayo, A. «Google vuelve a publicar la cuota de mercado de cada versión de Android», *El androide libre*. <https://www.elespanol.com/elandroidelibre/noticias-y-novedades/20211118/google-vuelve-publicar-cuota-mercado-version-android/628187872_0.html> [Consulta: 08/02/2022].

[77] «Procesar datos de entrada y salida con la biblioteca de compatibilidad de TensorFlow Lite», *TensorFlow*. <https://www.tensorflow.org/lite/inference_with_metadata/lite_support> [Consulta: 09/02/2022].

[78] «Back Button on Camera Crashes», *Stack Overflow*. <https://stackoverflow.com/questions/58425194/back-button-on-camera-crashes> [Consulta: 08/02/2022].

[79] «Bitmap», *Android Developers*. <https://developer.android.com/reference/android/graphics/Bitmap> [Consulta: 09/02/2022].

[80] «How I can convert float array to Bitmap or Uri to set it into image», *Stack Overflow*. <https://stackoverflow.com/questions/67364102/how-i-can-convert-float-array-to-bitmap-or-uri-to-set-it-into-image> [Consulta: 09/02/2022].

[81] «float array to byte array for displaying bitmap on canvas in android», *Stack Overflow*. <https://stackoverflow.com/questions/5961669/float-array-to-byte-array-for-displaying-bitmap-on-canvas-in-android> [Consulta: 09/02/2022].

[82] «Better way to create FloatArray in Kotlin», *Stack Overflow*. <https://stackoverflow.com/questions/52724882/better-way-to-create-floatarray-in-kotlin> [Consulta: 09/02/2022].

[83] «How to add an item to an ArrayList in Kotlin?», *Stack Overflow*. <https://stackoverflow.com/questions/46235579/how-to-add-an-item-to-an-arraylist-in-kotlin> [Consulta: 09/02/2022].

[84] «Creating an empty bitmap and drawing though canvas in Android», *Stack Overflow*. <https://stackoverflow.com/questions/5663671/creating-an-empty-bitmap-and-drawing-though-canvas-in-android> [Consulta: 09/02/2022].

[85] «Basic types», *Kotlin*. <https://kotlinlang.org/docs/basic-types.html#integer-types> [Consulta: 10/02/2022].

[86] «Read/Write String from/to a File in Android», *Stack Overflow*. <https://stackoverflow.com/questions/14376807/read-write-string-from-to-a-file-in-android> [Consulta: 09/02/2022].

[87] «"Bitmap too large to be uploaded into a texture"», *Stack Overflow*. <https://stackoverflow.com/questions/10271020/bitmap-too-large-to-be-uploaded-into-a-texture/30288557> [Consulta: 09/02/2022].

[88] Google «Google Play Console», *Google Play Console*. <https://play.google.com/console/u/0/developers> [Consulta: 09/02/2022].

[89] Google «Configurar una prueba abierta, cerrada o interna», *Ayuda de Play Console*. <https://support.google.com/googleplay/android-developer/answer/9845334?hl=es> [Consulta: 09/02/2022].

[90] «OpenSCAD: The Programmers Solid 3D CAD Modeller», *OpenSCAD*. <https://openscad.org/> [Consulta: 10/02/2022].

[91] «PLA 850 i3D Tested Gris 1,75 mm», *Impresoras3D.com*. <https://www.impresoras3d.com/producto/pla-850-i3d-tested-gris-175-mm/> [Consulta: 10/02/2022].

Otra bibliografía consultada

Blankenship, T. et al. *Android Apprentice (2nd edition): Beginning Android Development with Kotlin*. Razeware, 2019.

Cheng, Y., Olivares Dominguez, A. *Advanced Android App Architecture (1st edition): Real-world app architecture in Kotlin 1.3*. Razeware, 2019.

Smyth, N. *Android Studio 4.2 Development Essentials – Java Edition*. Payload Media, 2021.

Smyth, N. *Android Studio 4.2 Development Essentials – Kotlin Edition*. Payload Media, 2021.

Verma, Y. «A Complete Understanding of Dense Layers in Neural Networks», *Analytics India Magazine*. <https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/> [Consulta: 21/12/2021].

Créditos de Imagen

IMG-NAR-01, IMG-PLA-01, IMG-TOM-01. Fotografías de Dolores Muñoz Valero.

IMG-NAR-02, IMG-NAR-03. Fotografías de Daniel Rubio Ramírez.

IMG-LIM-01, IMG-LIM-02, IMG-LIM-03, IMG-LIM-04, IMG-NAR-04, IMG-NAR-05, IMG-NAR-06, IMG-NAR-07, IMG-PLA-02, IMG-PLA-03, IMG-PLA-04, IMG-PLA-05, IMG-ZAN-01, IMG-ZAN-02, IMG-ZAN-03, IMG-ZAN-04. Fotografías de José Ignacio Arnaudas Pontaque.

220px-Peras_en_C3A1rbol. Mariana Fuentes Linares CC BY-SA 4.0. Detalles de la licencia.

Imágenes con nombre de archivo terminado en "_100". Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Technical Report, Babes-Bolyai University, 2017