Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: 2. Machine Learning

Lista de la compra automática de fruta mediante la detección de objetos en imágenes con *Mask* R-CNN

Autor: Paula León Gil-Gibernau

Tutor: Jerónimo Hernández González

Profesor: Jordi Casas Roma

Barcelona, 17 de mayo de 2020

Créditos/Copyright

Una página con la especificación de créditos/copyright para el proyecto (ya sea aplicación por un lado y documentación por el otro, o unificadamente), así como la del uso de marcas, productos o servicios de terceros (incluidos códigos fuente). Si una persona diferente al autor colaboró en el proyecto, tiene que quedar explicitada su identidad y qué hizo.

A continuación se ejemplifica el caso más habitual, aunque se puede modificar por cualquier otra alternativa:

FICHA DEL TRABAJO FINAL

Título del trabajo:	Lista de la compra automática de fruta mediante la detección de objetos en imágenes con *Mask R-CNN*
Nombre del autor:	Paula León Gil-Gibernau
Nombre del colaborador/a docente:	Jerónimo Hernández González
Nombre del PRA:	Jordi Casas Roma
Fecha de entrega (mm/aaaa):	06/2020
Titulación o programa:	Máster Universitario en Ciencia de Datos (*Data Science*)
Área del Trabajo Final:	M2.879 - TFM - Área 2 aula 1 - *Machine Learning*
Idioma del trabajo:	Español
Palabras clave	Red neuronal convolucional profunda, detección de frutas, aprendizaje automático

Dedicatoria/Cita

Breves palabras de dedicatoria y/o una cita.

Agradecimientos

Si se considera oportuno, mencionar a las personas, empresas o instituciones que hayan contribuido en la realización de este proyecto.

Resumen

El objetivo de este estudio es realizar de forma automática una lista de la compra de fruta. Mediante un histórico de imágenes realizadas diariamente de un cajón de fruta, sin apilar, de una nevera, se aplicará una red neuronal convolucional profunda para detectar seis tipos de frutas en las imágenes. En concreto se usará una *Mask R-CNN*, que permite resolver la segmentación por instancias en imágenes obteniendo así las clases, las máscaras y las cajas delimitadoras de las frutas en las imágenes.

Palabras clave: Red neuronal convolucional profunda, detección de frutas, detección de objetos, máscaras, segmentación de instancias, aprendizaje automático, lista de la compra automática

Abstract

The aim of this project is to generate an automatic fruit shopping list. To do so, we will use a history of images taken daily, of a refrigerator fruit drawer, containing only unstacked fruit. We will apply a deep convolutional neural network to detect six different classed of fruit in the images. In particular, we will use Mask R-CNN, which allows solving the instance segmentation in images, being able to obtain the bounding boxes, the masks and the classes of the fruits which appear in the images.

Keywords: Deep convolutional neural network, Fruit detection, Object detection, Masks, Instance segmentation, Machine learning, Automatic shopping list

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y justificación del Trabajo

Este trabajo pretende realizar de forma automática una lista de la compra de fruta. Para ello se deberán procesar imágenes, reconociendo objetos en ellas, en este caso frutas. Se deberán aplicar algoritmos de aprendizaje profundo para crear modelos que permitan reconocer y contar las frutas en las imágenes.

El reconocimiento de objetos en imágenes está ampliamente usado para solucionar problemas de visión por computación con infinitas finalidades, desde el reconocimiento facial, la detección de peatones hasta para seguir una pelota en los partidos de fútbol.

En la actualidad, la tecnología se aplica cada vez más en el sector de la agricultura. En concreto, los sistemas automáticos inteligentes que se pueden usar para contar la fruta de una cosecha, para la detección de plagas que puedan dañarla o para analizar el rendimiento de dicha cosecha. Un sistema preciso de reconocimiento de fruta en imágenes es el elemento clave para poder realizar cualquiera de estas tareas. Aplicar dichos sistemas les permite a los agricultores, ganar tiempo, precisión y dinero en las tareas citadas anteriormente.

Existen ya muchas empresas que intentan revolucionar la cesta de la compra automática. Desde Amazon con su cesta de la compra por suscripción, que permite recibir con la frecuencia deseada productos seleccionados [9] a la empresa Caper con un carro de la compra que detecta los productos introducidos en él y calcula el importe total [10]. Todo esto ayuda a mejorar la experiencia del cliente al comprar y le ayuda a ahorrar tiempo con la tarea de la compra. Este trabajo es otra idea que busca ayudar al cliente a saber qué, cuánto y cuándo debe comprar fruta, que se podría extender a muchos más productos.

A partir de este trabajo se podrían llegar a aplicaciones de control de existencias de uso doméstico, en una casa, y empresarial, por ejemplo un supermercado.

1.2. Motivación

Estudié Ingeniería Informática en Mención de Computación en la UAB, y lo que más me interesó fue todo lo relacionado con el procesado y reconocimiento de imágenes. He podido seguir profundizando en temas de computación durante la realización de este master en Ciencia de Datos en la UOC, y me parece muy interesante concluir dicho master con un trabajo en el que aplicar todo lo aprendido durante estos años y que está relacionado con un tema que me gusta, la visión por computación.

1.3. Objetivos del Trabajo

1.3.1. Hipótesis

Este proyecto pretende demostrar la siguiente hipótesis:

Automatizar la generación de la lista de la compra de fruta, usando un histórico de imágenes realizadas diariamente del cajón de la fruta, sin apilar, de una nevera.

Se tratarán seis clases de frutas:

- Plátano
- Manzana
- Pera
- Piña
- Limon
- Naranja

1.3.2. Objetivos parciales

Existen dos objetivos parciales que nos ayudarán a demostrar y confirmar la hipótesis anteriormente descrita.

- Detección de existencias. ¿Cuántas piezas de cada fruta hay?. Esto implica la detección de fruta en una imagen. Para ello se propone aplicar un modelo *Mask R-CNN* que usa tanto máscaras como segmentación de instancias.
- Predicción de la compra. ¿Qué he de comprar hoy cuando vaya al supermercado?

1.4. Enfoque y método seguido

Para entender el enfoque y el método seguido para la realización de este proyecto se deben tratar varios puntos:

- La base de datos de imágenes. Este es uno de los puntos más importantes a tratar. Se suele pensar que cuanto mayor es el número de muestras mejor resolverá el modelo. Pero se debe encontrar el punto medio para no caer ni en sobreajuste ni en subajuste. El sobreajuste o sobreentrenamiento puede deberse a un exceso de datos o a un numero de parámetros muy elevado respecto a la cantidad de muestras. Por el contrario el subajuste, en inglés *underfitting*, puede producirse por falta de datos. La falta de datos puede hacer que nuestro modelo sea demasiado general y no tenga buenos resultados, el exceso de datos podría ocasionar que sea demasiado específico a los datos que tenemos y por tanto dará excelentes resultados con esos mismos datos pero no tan buenos para datos distintos a estos usados para entrenar el modelo. Para poder controlar y evitar los dos casos anteriores y para poder validar los resultados del modelo, usaremos un dataset de entrenamiento, otro de test y por último uno para la validación. En relación al último, se generará un dataset para validar el modelo de imágenes de fruta, sin apilar, dentro de un cajón de la nevera. Al no estar apilada, eliminaremos mucho ruido y disminuiremos la probabilidad de error.

- Las clases de frutas. Es importante escoger frutas con características distintas entre sí y otras que se asemejen. Sería interesante ver que el modelo se confunde más entre peras y manzanas que entre plátanos y naranjas. Escoger demasiadas clases de frutas puede aumentar mucho la complejidad del problema, y escoger muy pocas hacerlo demasiado sencillo. Seis clases me parecen razonables y además las frutas escogidas tienen tanto otras que se asemejan como totalmente distintas.

- El modelo para la detección de frutas en la imagen. Existen infinidad de modelos para la detección de objetos en imágenes. El modelo *Mask R-CNN* es el estado del arte en cuanto a la segmentación de instancias, y creo que puede dar resultados excelentes en la detección de frutas en imágenes.

- La detección y predicción de existencias: con ayuda del modelo anterior se calculará, dada una imagen el número de unidades de cada clase de fruta para cada imagen. El dataset de validación mencionado anteriormente será el usado para este punto, siendo este un histórico de imágenes generadas diariamente. Con ello tendremos la cantidad de cada fruta por día durante un periodo concreto de tiempo. A partir de esto se pueden extraer patrones y predecir las compras a realizar.

1.5. Planificación del Trabajo

La planificación de este proyecto se divide principalmente en las entregas parciales que se deben realizar, llamadas PECs. A partir de cada entrega se han desglosado las tareas a realizar, a continuación se puede ver el diagrama de Gantt de las PECs 1 y 2 1.1, el de la PEC 3 1.2 y el de las PECs 4 y 5 1.3:

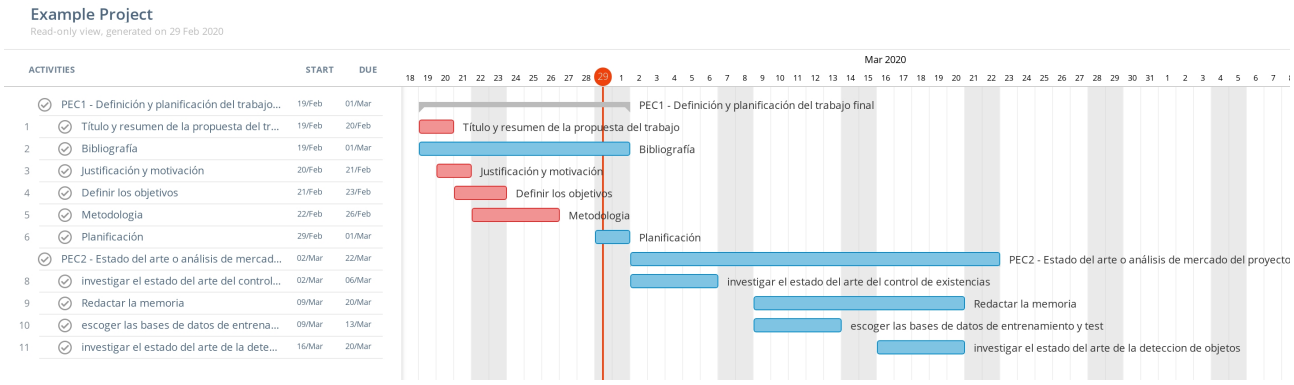

Figura 1.1: Diagrama de Gantt de la planificación del proyecto PECs 1 y 2.

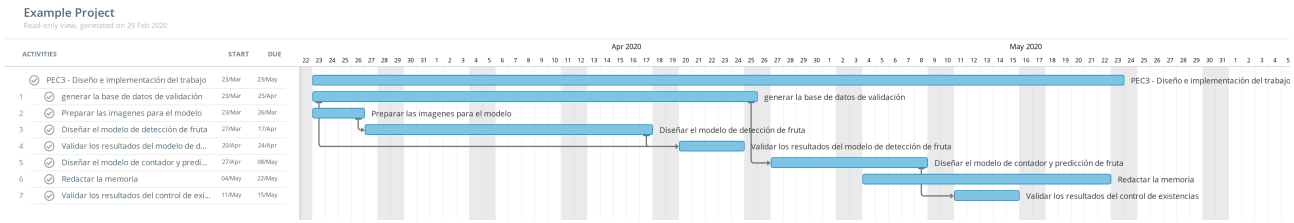

Figura 1.2: Diagrama de Gantt de la planificación del proyecto PEC 3.

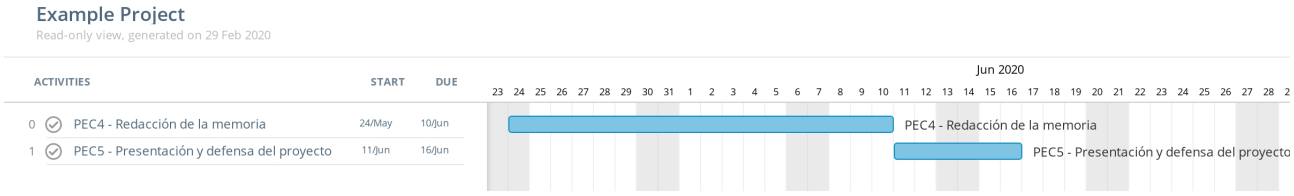

Figura 1.3: Diagrama de Gantt de la planificación del proyecto PECs 4 y 5.

1.6. Breve descripción de los capítulos de la memoria

Este trabajo empieza con una introducción, en la que se explica su contexto y justificación, realizar de forma automática una lista de la compra de fruta. Después se justifica la motivación personal, es decir por qué se quiere hacer este proyecto. A continuación se exponen tanto la hipótesis principal como los objetivos parciales. Por último, en este apartado se explica el enfoque y el método seguido para la realización de este proyecto junto con la planificación y el diagrama de Gantt de las tareas con su duración.

El segundo capítulo de esta memoria es el estado del arte, en el que se relata de forma sintetizada cómo han abordado los investigadores, hasta el momento, el problema que pretende resolver este trabajo. En primer lugar se explican las técnicas del estado del arte de aprendizaje automático para la detección de objetos en imágenes, en segundo lugar las técnicas de aprendizaje profundo y por ultimo como se ha abordado de forma más concreta la detección de fruta en imágenes.

El tercer capítulo de esta memoria es la metodología, donde se explican las herramientas y los materiales usados para la realización del proyecto. En entorno de desarrollo, los lenguajes de programación, las librerías especializadas y los métodos utilizados a lo largo de todo el trabajo.

Capítulo 2

Estado del Arte

La detección de objetos en imágenes es un problema que ha atraído mucha atención durante estos años, pues tiene infinitas aplicaciones. Dicho problema se ha resuelto tanto con algoritmos convencionales de aprendizaje automático como con algoritmos de aprendizaje profundo. Pero la mayor parte de los modelos que son el estado del arte usan aprendizaje profundo.

2.1. Aprendizaje automático

El primer detector de objetos con resultados robustos, fue propuesto en 2001 por Viola y Jones [11]. El objetivo era la detección de caras en tiempo real, pero se ha usado para infinidad de aplicaciones, desde la detección de manos a coches, señales de tráfico y otros objetos. Usaron un algoritmo de aprendizaje basado en *AdaBoost*, que es capaz de obtener clasificadores muy eficientes seleccionando un número pequeño de características visuales relevantes, y además combinaron clasificadores en cascada que permite descartar las regiones de la imagen que no son de interés, para centrarse en las que sí lo son. *AdaBoost*, la abreviatura en inglés de *Adaptive Boosting*, es un algoritmo de aprendizaje automático creado por Yoav Freund y Robert Schapire en 1996, que se usa habitualmente en problemas de clasificación, con el objetivo de convertir un conjunto de clasificadores débiles en fuertes.

El Histograma Orientado a Gradientes, conocido por sus siglas en inglés como *HOG*, es un descriptor de características enfocado a la detección de objetos, que apareció en 1986. La técnica se basa en contar las apariciones de orientación de gradientes en regiones localizadas de una imagen. No fue hasta 2005, que cobro más interés, cuando Dalal y Triggs presentaron su trabajo de detección de peatones usando *HOG* [12], dicho trabajo también fue aplicado a la detección de animales y vehículos en imágenes.

Para reconocer los descriptores *HOG*, se usan las Máquinas de vectores de soporte, por sus siglas en inglés *SVM*, que son algoritmos de aprendizaje supervisado que se usan para

clasificar. Todos estos algoritmos de aprendizaje automático requieren del etiquetado previo de las imágenes al igual que de la selección previa de las características relevantes de la imagen.

2.2. Aprendizaje profundo

El aprendizaje profundo, es denominado así por la profundidad del número de capas por las que los datos se transforman. Consiguen desenredar las abstracciones extraídas en cada capa para elegir las características que mejoran el rendimiento.

Las arquitecturas básicas y más representativas de aprendizaje profundo son las redes neuronales profundas y más concretamente las redes neuronales convolucionales, conocidas por sus siglas en inglés como *CNN*. Las redes neuronales profundas son redes neuronales con varias capas ocultas entre las capas de entrada y salida. Las redes neuronales convolucionales, son un tipo de redes neuronales profundas aplicadas principalmente a problemas de clasificación y segmentación de imágenes. El motivo por el que estas redes se llaman así, es porque se emplea una operación matemática llamada convolución en, como mínimo, una de sus capas.

El objetivo que se busca con estas redes es entrenarlas con imágenes etiquetadas para que el modelo pueda abstraer características que le permitan posteriormente clasificar una nueva imagen sin etiquetar. Por tanto ya no se requiere de la selección previa de dichas características de la imagen, como en los métodos de aprendizaje automático anteriormente explicados.

Las unidades de procesamiento gráfico, conocidas por sus siglas en inglés como *GPU*, permiten ejecutar los métodos de aprendizaje profundo en ella y como son muy paralelizables, permite aumentar altamente la capacidad de cómputo. Además también existen servicios en la nube que ofrecen procesamiento con *GPU*. Por otro lado Google creó una plataforma de aprendizaje automático en el que proporciona modelos ya pre entrenados y la posibilidad de personalizar modelos.

Gracias al desarrollo de las redes neuronales de convolución profunda y a la potencia de cómputo que proporcionan las *GPUs*, las técnicas de detección de objetos en imágenes han teniendo una rápida y gran evolución. En la siguiente figura 2.1, podemos ver los trabajos más importantes relacionados con la detección de objetos que usan *CNNs* desde 2012 hasta la actualidad :

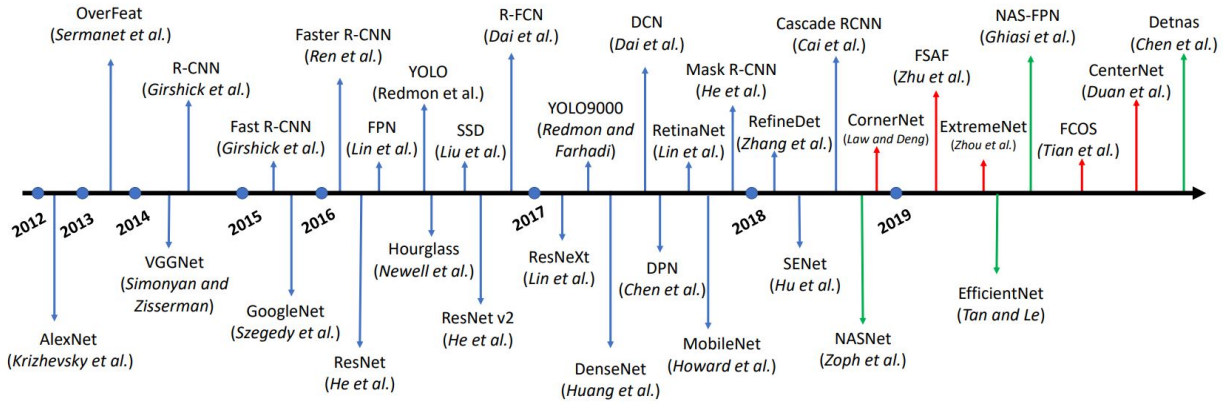

Figura 2.1: Trabajos de detección de objetos basados en *CNNs* desde 2012 hasta 2019.[1]

Las bases de datos más comúnmente usadas para la evaluación de los modelos de detección de objetos son:

- *Pascal VOC2007*: tiene un total de 20 categorías, está compuesto por tres conjuntos, uno de entrenamiento con 2501 imágenes, otro de test con 5011 y uno de validación con 2510 imágenes.
- *Pascal VOC2012*: tiene las mismas 20 categorías que Pascal VOC2007, está compuesto por tres conjuntos, uno de entrenamiento con 5717 imágenes, otro de test con 10991 y uno de validación con 5823 imágenes.
- *MSCOCO*, tiene un total de 80 categorías, está compuesto por tres conjuntos, uno de entrenamiento con 118287 imágenes, otro de test con 40670 y uno de validación con 5000 imágenes.
- *Open Images*: contiene 1.9M de imágenes con 15M de objetos y 600 categorías. La mayoría de estas categorías tiene alrededor de 1000 muestras de entrenamiento.
- *ImageNet*: contiene más de 20.000 categorías y un total de 14 millones de imágenes. Desde 2010 se celebra una competición de los algoritmos de análisis de imágenes sobre esta base de datos llamada *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*.

Los modelos usados para resolver la detección de objetos en imágenes se pueden clasificar en dos tipos, por un lado aquellos tradicionales, que proponen la región del objeto y luego lo clasifican y por otro lado afrontar el problema como un problema de regresión o clasificación,

obteniendo resultados finales directamente, tanto la localización como la categoría a la que pertenece el objeto detectado.

2.2.1. Métodos tradicionales

Los métodos tradicionales más conocidos para la detección de objetos en imágenes son:

- *AlexNet*: un trabajo publicado por Krizhevsky, Sutskever y Hinton, en 2012 [13] fue el primero en usar redes neuronales convolucionales para la resolución del problema de la detección de objetos en imágenes. Aplicado a la base de datos *ImageNet* y posicionándose entre los cinco mejores de la competición *ILSVRC* del año 2012, consiguiendo una tasa de error del 15,3 %. La arquitectura de *AlexNet* tiene ocho capas de las cuales cinco son convolucionales, algunas de estas capas seguidas por capas *max-pooling*, funciones que permiten reducir el tamaño de los datos, y las últimas tres capas completamente conectadas. Usaron además, la función de activación de unidad lineal rectificada, por sus siglas en inglés, *ReLU*, no saturante, que resultó tener mejor rendimiento que las funciones tan y sigmoidea. Consiguieron mejorar el tiempo de entrenamiento gracias al uso de *multi-GPU*, poniendo a entrenar la mitad de las neuronas en una *GPU* y la otra mitad en la otra *GPU*. Además se enfrentaron a problemas de sobre entrenamiento, usando técnicas como el *dropout*, que consiste en desactivar neuronas con una probabilidad determinada y el aumento de datos, generando más imágenes.

- *OverFeat*: arquitectura presentada en 2013 por Sermanet et al. [14], en la que propusieron un algoritmo con una ventana deslizante de escala múltiple usando redes neuronales convolucionales (*CNNs*).

- *R-CNN*: Poco después de la publicación de *OverFeat*, Girshick et al publicaron en 2014 un trabajo en el que presentaron el método llamado *R-CNN*, regiones con características *CNN* [2]. Proponían un modelo de tres pasos. En el primero se extraen posibles objetos con un método de propuesta de regiones. En el segundo, se extraen las características de cada región usando redes neuronales convolucionales (*CNNs*), y por último en el tercer paso, se clasifica cada región usando máquinas de vectores de soporte (*SVMs*). En la siguiente figura 2.2 se muestran estos pasos:

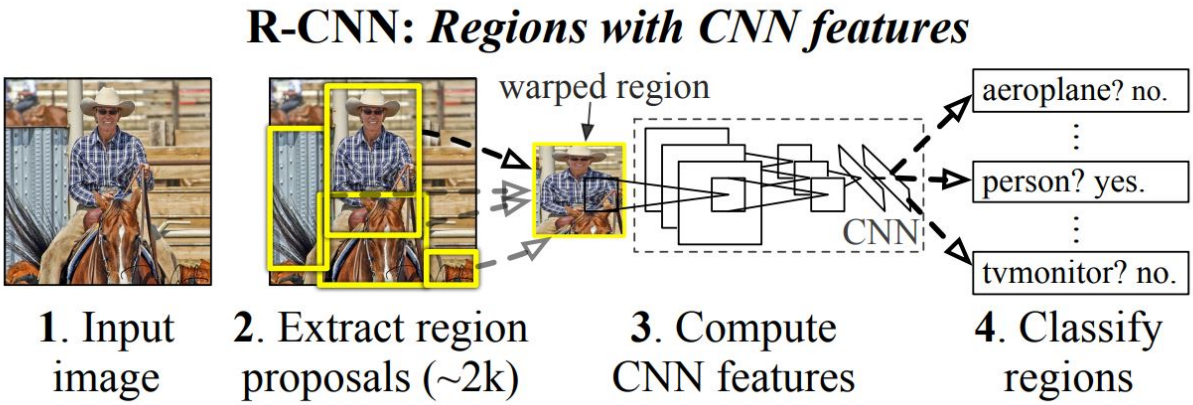

Figura 2.2: Arquitectura *R-CNN*.[2]

- *Fast R-CNN*: Girshick, publicó en 2015 [15], una mejora del anterior modelo *R-CNN*, para hacer el modelo más rápido, aplicando la red neuronal convolucional en toda la imagen de entrada en lugar de sobre cada región.

- *GoogLeNets*: Szegedy et al. publicaron en 2015 [16] una arquitectura también llamada *Inception*, en la que mejoraron el uso de los recursos informáticos dentro de la red, aumentando la profundidad y el ancho de la red.

- *Faster R-CNN*: en 2016, Shaoqing, Kaiming, Girshick y Sun, publicaron otro trabajo [17], la tercera iteración del *R-CNN*, añadiendo el concepto de red de propuesta de regiones, por sus siglas en inglés *RPN*, para hacer que el modelo pudiera entrenarse de principio a fin.

- *ResNets*: en 2016, He, Zhang, Ren y Sun, publicaron un trabajo [18], en el que presentaron la arquitectura *ResNets*, introduciendo el nuevo concepto de funciones residuales de aprendizaje para poder entrenar redes neuronales muy profundas. Ganaron la competición *ILSVRC* de 2015 con el primer puesto en el problema de clasificación.

- *R-FCN*: en 2016, Dai, Li, He y Sun publicaron un trabajo [19] en el que presentaron la arquitectura *R-FCN*, a diferencia de los detectores basados en regiones anteriores, como el *R-CNN*, no aplicaban una red por cada región, sino que compartían los cálculos por toda la imagen.

- *Mask R-CNN*: en 2017 se publicó un trabajo de He, Gkioxari, Dollar y Girshick [3], sobre la arquitectura *Mask R-CNN* para la segmentación de instancias en objetos y la generación

de máscaras de los objetos. Ganaron la competición *COCO* en 2016, en problemas de segmentación de instancias y en encontrar las cajas delimitadoras de objeto. Este es el modelo que se usará en el proyecto para detectar la fruta en las imágenes. La arquitectura permite crear la segmentación a nivel de pixel para cada objeto y además puede separar cada objeto de su fondo. A continuación podemos ver la arquitectura de la segmentación de instancias 2.3.

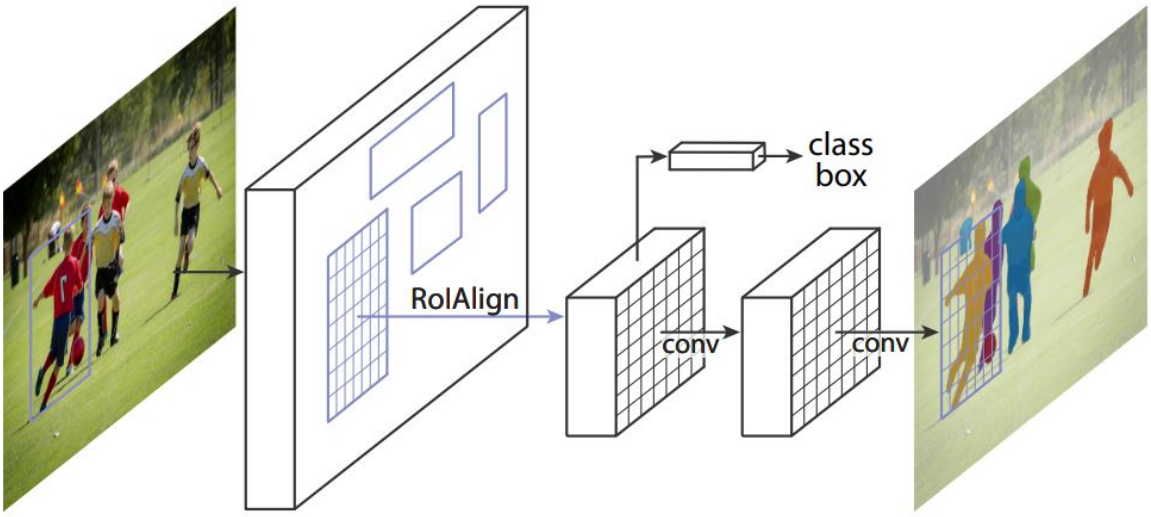

Figura 2.3: Arquitectura de *Mask R-CNN* para la segmentación de instancias.[3]

2.2.2. Métodos de regresión y clasificación

Los métodos de regresión y clasificación más conocidos para la detección de objetos en imágenes son:

- *YOLO (You Only Look Once)*: en 2016, se publicó el trabajo [4] de Redmon, Divvala, Girshick y Farhadi 2016, en el que presentaron una arquitectura basada en una simple red neuronal convolucional, con buenos resultados y rápida de entrenar. Permitiendo por primera vez la detección de objetos en tiempo real. En esta figura 2.4 se puede ver su arquitectura:

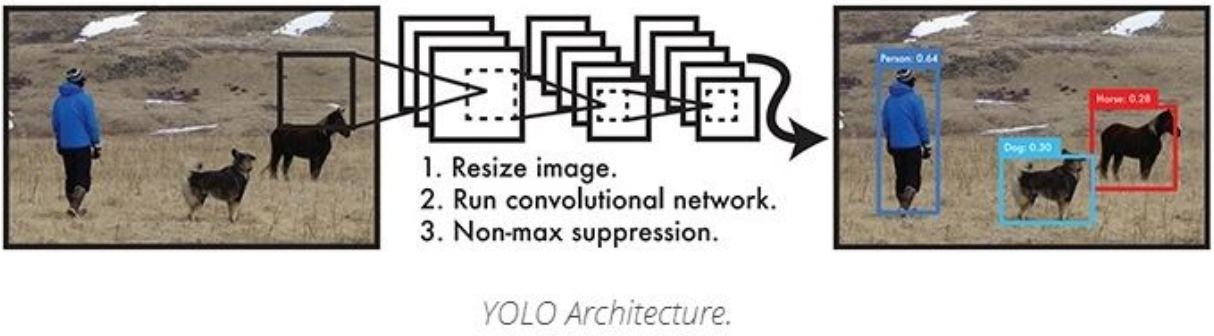

Figura 2.4: Arquitectura *YOLO*.[4]

- *SSD*: en 2016 Liu et al. publicaron un trabajo [20] en el que presentan una arquitectura llamada *SSD (Single Shot MultiBox Detector)*, con una única red neuronal profunda, usando mapas de características convolucionales con múltiples tamaños, haciendo el método más rápido que el *YOLO* y mejorando sus resultados.
- *YOLO900*: En 2017, Redmon y Farhadi publicaron la arquitectura YOLO900 [21], mejorando la velocidad y los resultados de *YOLO*.

De los últimos trabajos presentados entorno a la detección de objetos, en 2018, Zoph, Vasudevan, Shlens y Le, publicaron la arquitectura *NASNet* [22] que propone un nuevo espacio de búsqueda que permite la transferibilidad. Y en 2019, Chen et al. presentaron su trabajo sobre la arquitectura *DetNAS* [23], en el que usan también la búsqueda de arquitectura neural, conocida por sus siglas en inglés como *DetNAS*, para el diseño de mejores conexiones principales para la detección de objetos.

2.3. Detección de fruta

Existen muchos trabajos realizados sobre la detección de fruta en imágenes, en concreto existe uno presentado por Payne, Walsh, Subedi y Jarvis en 2014 [24], en el que se analizaba automáticamente el rendimiento de un cultivo de mango, analizando imágenes nocturnas.

Pero no fue hasta hace pocos años que se aplicaron, de forma más extensiva, métodos de aprendizaje automático y de aprendizaje profundo para la resolución de dicho problema. En 2016, Inkyu et al. [25], publicaron un trabajo en el que se detectaba fruta en imágenes, usando redes neuronales convolucionales, en concreto usando un método basado en *Faster R-CNN*.

En 2017, Rahnemoonfar y Sheppard [26], publicaron un trabajo en el que se usaron también redes neuronales convolucionales pero esta vez con un modelo basado en *Inception-ResNet*, para

contar frutas en imágenes.

Capítulo 3

Metodología

3.1. Detección de objetos

El objetivo de la detección de un objeto en imágenes implica por un lado clasificar y etiquetar el objeto y por otra localizarlo en la imagen, obteniendo la caja delimitadora de dicho objeto. La detección de más de un objeto en una imagen requiere por un lado de la detección del objeto, pero además de la segmentación de instancias, que es la detección de los objetos en las imágenes a nivel de pixel. En la siguiente imagen 3.1, se muestra la diferencia entre el problema de la detección de un solo objeto o varios objetos en una imagen:

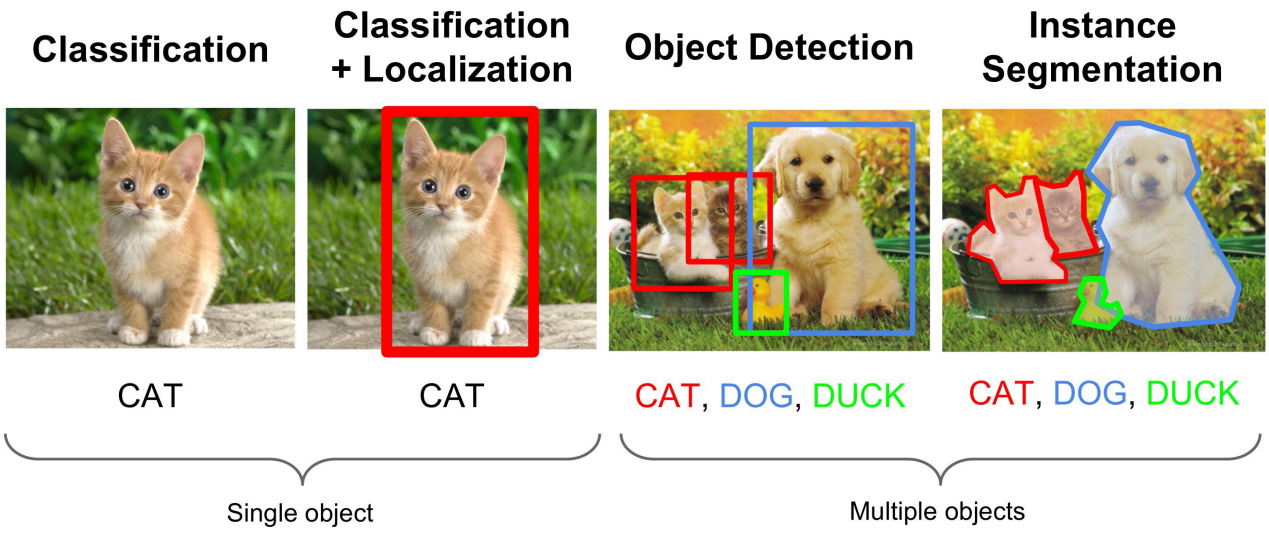

Figura 3.1: Diferencia entre la detección de un solo objeto o varios en imágenes.[5]

3.2. Redes neuronales artificiales

En este trabajo se han utilizado un tipo de red neuronal convolucional, la *Mask RCNN*. Pero antes de explicar este tipo de red, empezaremos por explicar conceptos generales de las redes neuronales y en concreto de las redes neuronales convolucionales.

Las redes neuronales artificiales imitan el funcionamiento de las redes neuronales de los organismos vivos. Están formadas por un conjunto de unidades llamadas neuronas artificiales conectadas entre sí para transmitir señales.

Cada neurona aplica una función de entrada determinada a los datos de entrada que puede haber recibido de la interconexión con otra neurona y convierte este valor en el de salida aplicando una función de activación. Este valor será la entrada de otra neurona o la salida de la red. En la siguiente imagen 3.2 podemos ver el esquema de una neurona artificial.

Figura 3.2: Esquema de una neurona artificial.

Cada conexión de entrada en una neurona tiene un peso, la función de entrada combina las diferentes entradas con su peso, y agregar los valores obtenidos por las conexiones de entrada y obtiene un único valor. La función de activación se aplica sobre el valor obtenido por la función de entrada anteriormente explicado. Esta modifica el valor obtenido de forma limitadora antes de propagarse a otra neurona.

La arquitectura de una red neuronal artificial es la organización de las neuronas en distintas capas, y los parámetros de configuración de las neuronas, como por ejemplo las funciones de entrada o activación. En el caso de que todas las neuronas de una capa estén conectadas a las neuronas de la siguiente capa, la red neuronal artificial es denominada totalmente conectada o densa. La más simple está formada por una sola neurona conectada a todas las entradas y con una salida. Otra arquitectura muy simple seria la red monocapa, que consiste en un capa de

entrada, otra oculta con un conjunto determinado de neuronas y una capa de salida con una o varias neuronas. Las capas ocultas son las capas intermedias de una red neuronal, entre la capa de entrada y la capa de salida, son las que contiene las unidades no observables. Se denomina capa oculta a Un número demasiado elevado de neuronas en la capa oculta puede causar sobre entrenamiento. Se pueden añadir un número infinito de capas, lo que se conoce como redes multicapa. En la siguiente imagen 3.3 podemos ver la arquitectura de una red neuronal con capas ocultas.

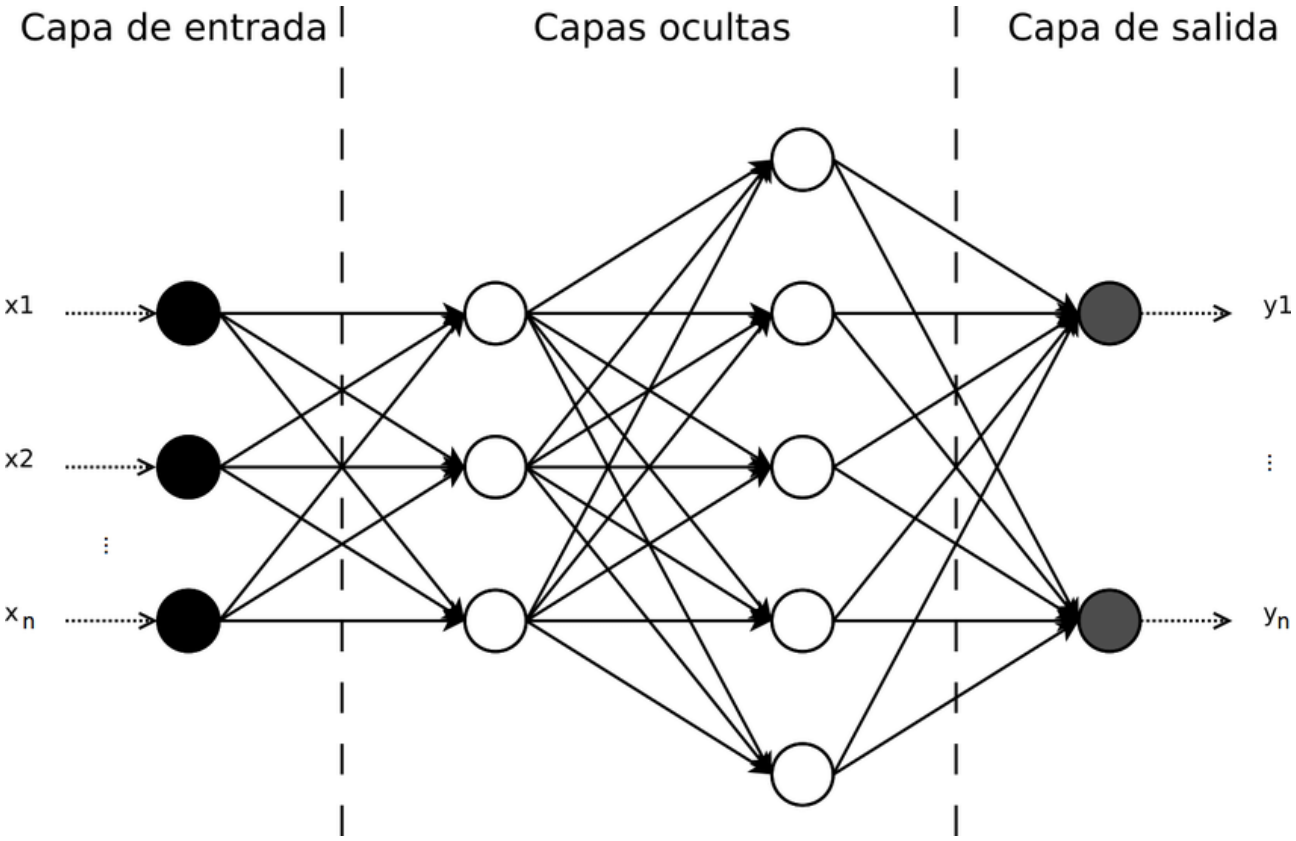

Figura 3.3: Arquitectura de una red neuronal artificial con capas ocultas.[6]

3.2.1. Entrenamiento de una red neuronal artificial

El entrenamiento, es el procedimiento para llevar a cabo el proceso de aprendizaje en una red neuronal. El proceso de aprendizaje de una neurona se basa en el conjunto de pesos de la entrada de la neurona y en el valor umbral de la función de activación, también denominado sesgo. El objetivo es minimizar la función de error o pérdida.

El error que comete una red neuronal es la diferencia entre el resultado esperado y el obtenido. Cada vez que aplicamos una instancia de entrenamiento, en inglés denominada *epoch*,

comparamos la salida con el resultado esperado y si este difiere se modifican los pesos y el sesgo de las neuronas para que la red aprenda la salida deseada. El objetivo es llevar la red a unos valores que nos proporcionen el mínimo error. La función de perdida está compuesta por el error, y por el término de regularización, que se utiliza para evitar el sobre aprendizaje.

En el caso de las redes multicapa no podemos saber a priori cuales son los valores de salida correctos, por tanto debemos definir como modificar los pesos y tasas de aprendizaje de las neuronas de las capas ocultas a partir del error obtenido de la capa de salida. El método de retro propagación es el que realiza esto. Se basa en dos pasos:

- Propagación hacia delante, que consiste en aplicar una instancia de entrenamiento y obtener la salida.
- Propagación hacia atrás, que calcula el error de la capa de salida y lo propaga hacia atrás para calcular los valores de error de las neuronas de las capas ocultas.

Cada una de estas instancias de entrenamiento empieza por calcular los valores de salida desde la capa de entrada, pasando por las ocultas, hasta la de salida y luego propaga hacia atrás el error obtenido en la capa de salida, pasando por las capas ocultas nuevamente hasta la capa de entrada. La idea es propagar el error de forma proporcional a la influencia o peso que ha tenido cada neurona de las capas ocultas en relación al error final producido por las neuronas de la capa de salida.

Para no pasar todo el conjunto de datos a la vez en una instancia de entrenamiento, se realiza el aprendizaje por lotes, en inglés *batches*.

El método de descenso de gradiente es un algoritmo de entrenamiento muy extendido que permite controlar la tasa o velocidad de entrenamiento. La tasa de entrenamiento es un parámetro que determina el tamaño del paso de cada iteración mientras vamos hacia el mínimo de la función de perdida.

3.2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales, conocidas por sus siglas en inglés como *CNN*, son un tipo de redes neuronales profundas, es decir que tienen varias capas ocultas entre las capas de entrada y salida. Son aplicadas principalmente a problemas de clasificación en imágenes.

Las redes neuronales convolucionales proponen una arquitectura que se aprovecha de las características particulares de las imágenes. En la que la posición de una característica en la imagen es relevante. Cada uno de los pixeles de la imagen se asocia a una entrada en la capa de entrada. Estas redes no sestan totalmente conectadas es decir que las neuronas de una capa se conectan solo a un conjunto de las neuronas de la siguiente capa. Además todas

las neuronas ocultas de una capa comparten los mismos pesos y sesgos. Por tanto todas las neuronas de una capa oculta detectan la misma característica, pero en distintos puntos de la imagen. Es decir se aplica el mismo detector de características por toda la imagen. Los pesos y sesgos compartidos definen un *kernel* o filtro. El hecho de compartir pesos y sesgos en las neuronas de una capa oculta se traduce directamente en un entrenamiento más rápido. Permiten construir redes neuronales profundas con capas convolucionales. A menudo, después de las capas convolucionales se suelen usar capa de agrupación, que simplifican la salida de la capa de convolución. Una de las funciones de agrupación más conocida es la de *max-pooling*, que selecciona el valor máximo de un conjunto de entradas.

3.2.2.1. Redes neuronales *Mask R-CNN*

Las redes neuronales convolucionales *Mask R-CNN*, son un modelo que permite resolver la segmentación de instancias en imágenes. Existen dos pasos, en primer lugar genera propuestas de regiones donde puede haber un objeto y en segundo lugar predice la clase del objeto, su caja delimitadora y por último genera la máscara a nivel de píxel de dicho objeto dentro de la región propuesta en el primer paso.

La parte de la red responsable de la detección de las cajas delimitadoras deriva del modelo *Faster R-CNN*, que tiene una red de propuesta de regiones, conocida por sus siglas en inglés como *RPN* que propone regiones donde podría haber objetos. Escanea el mapa de características, pero para poder vincular una característica a la ubicación de la imagen sin procesar existen los anclajes. Los anclajes son un conjunto de cajas con unas ubicaciones predefinidas en la imagen. Las clases de los objetos y sus cajas delimitadoras se asignan a anclajes individuales. Dado que los anclajes a distintas escalas se unen en diferentes niveles del mapa de características, la red de propuesta de regiones usa estos anclajes para determinar que parte del mapa de características debería obtener un objeto y de qué tamaño es su caja delimitadora.

En segundo lugar, se aplica un proceso similar a la red de propuesta de regiones, pero sin anclajes, se usa *ROIAlign*, un método que ayuda a ubicar las áreas más relevantes del mapa de características, y hay una rama que genera mascaras para casa objeto a nivel de pixel. *ROI*, corresponde por sus siglas en inglés a *Region Of Interest*, es decir la región de interés. La región de interés es una parte de la imagen que se quiere filtrar o con la que se quiere operar. En este caso el método *ROIAlign* divide estas regiones de interés en 9 cajas del mismo tamaño y luego aplica interpolación bilineal dentro de cada una de estas, este método no pierde información como lo hace el método usado por *Faster R-CNN* llamado *ROIPool*.

En la siguiente figura 3.4 podemos ver todos los elementos explicados que intervienen en el modelo *Mask R-CNN*.

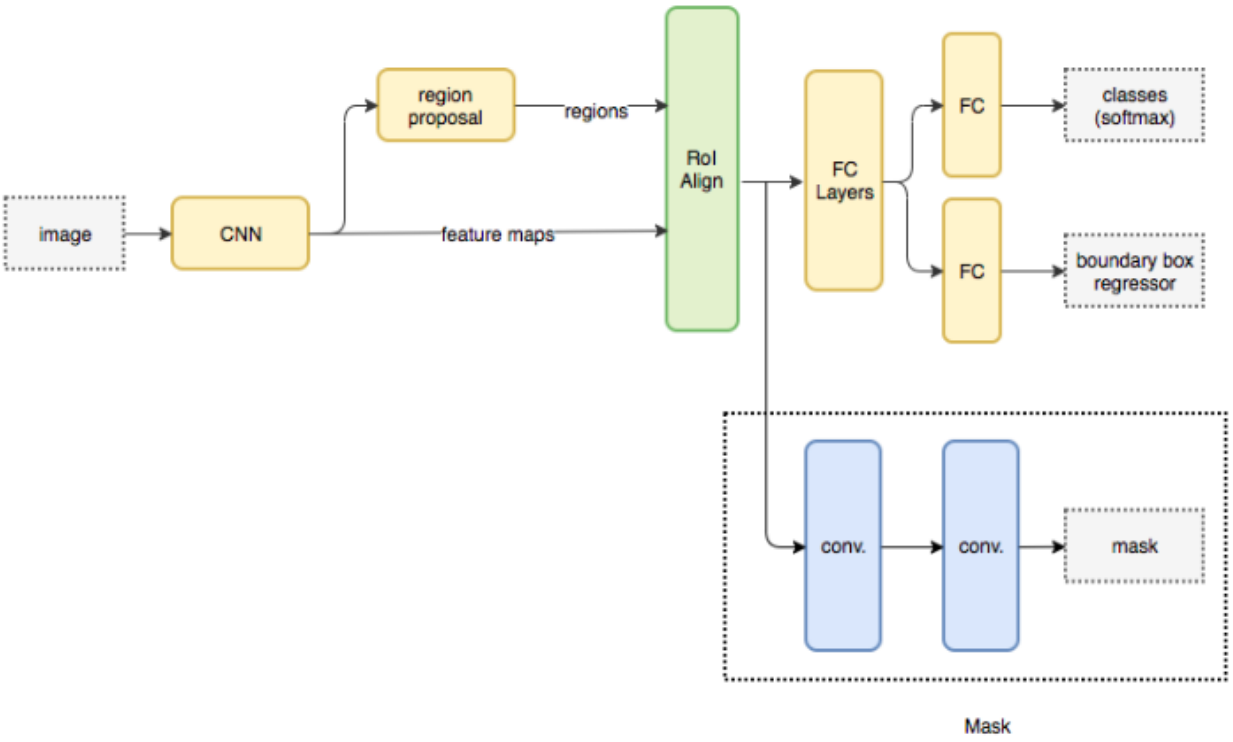

Figura 3.4: Esquema de una red neuronal *Mask R-CNN*.[7]

3.3. TensorFlow

TensorFlow es una librería especializada de código abierto y gratuita, desarrollada por Google, para el aprendizaje automático. El nombre de TensorFlow deriva de las operaciones que realizan las redes neuronales sobre los tensores. Los tensores son los arrays multidimensionales. TensorFlow permite crear grafos. Un grafo de TensorFlow es un grafo computacional que es una red de nodos, en el que cada nodo realiza una operación. Cada arista del grafo, que une un nodo con otro, representa un tensor. Tiene una arquitectura flexible y permite desarrollar en plataformas CPUs, GPUs o TPUs.

CPU, por sus siglas en inglés, la unidad central de procesamiento, es el hardware que interpreta las instrucciones de un programa realizando operaciones básicas aritméticas, lógicas y de entrada y salida del sistema. Se conoce como multiprocesamiento, cuando un ordenador tiene más de una CPU.

Trabajar con GPUs es muy recomendable, según la página oficial de Nvidia [27], con el uso de sus últimas GPUs Pascal, TensorFlow se ejecuta un 50 % más rápido. Ejecutándolo en un sistema Linux e instalando CUDA y cuDNN. La arquitectura unificada de dispositivos

de cómputo, por sus siglas en inglés, CUDA, es una plataforma de computación en paralelo desarrollada por Nvidia. CUDA utiliza el paralelismo de sus múltiples núcleos, permitiendo lanzar un elevado número de hilos a la vez. Es por eso que si la aplicación a ejecutar, está diseñada para poder lanzar tareas independientes a la vez esta plataforma ofrece un gran rendimiento. La librería de Nvidia de aprendizaje profundo para CUDA, por sus siglas en inglés, cuDNN, permite la aceleración por GPUs de las primitivas de las redes neuronales profundas.

TPU son las siglas en ingles de unidad de procesamiento del tensor, es un circuito integrado de aplicación específica, por sus siglas en inglés ASIC, desarrollado por Google específicamente para el aprendizaje automático y adaptado a TensorFlow. A diferencia de las GPUs, está diseñado para la ejecución del modelo y no para el entrenamiento en sí, es decir permiten un mayor volumen de cálculo pero con una precisión reducida.

Para la realización de este trabajo se ha usado en una plataforma CPUs y la versión 1.12 de Tensorflow.

3.3.1. API de TensorFlow para la detección de objetos

TensorFlow desarrolló una interfaz de programación de aplicaciones, por sus siglas en inglés API, para la detección de objetos, que es de código abierto y permite usar modelos de detección de objetos previamente entrenados o crear y entrenar modelos haciendo uso del aprendizaje por transferencia. Esto es muy útil puesto que implementar un modelo desde cero puede ser difícil y requiere mucha capacidad computacional. Ofrecen una colección de modelos pre entrenados, llamados modelos de detección *zoo*, entrenados sobre los conjuntos de imágenes de COCO [28], Kitti [29], Open Images [30], AVA v2.1 [31] y el conjunto de imágenes de detección de especies de iNaturalist [32]. Cada uno de estos modelos contiene un fichero de configuración que fue el utilizado para el entrenamiento de dicho modelo, los grafos obtenidos y los puntos de control, en inglés *checkpoints*.

3.3.2. TensorBoard

TensorBoard es la interfaz de visualización que permite inspeccionar, comprender y analizar las ejecuciones y los grafos de TensorFlow mediante un conjunto de aplicaciones web.

Dependiendo de la información que se ha agregado a la ejecución del modelo, TensorBoard puede mostrar distintos campos, como por ejemplo:

- Escalares: Muestra diferentes métricas como por ejemplo la función de pérdida o aprendizaje durante el entrenamiento del modelo

- Gráficos: muestra el modelo, es decir todo el grafo de tareas. Desde el entrenamiento en sí, la alimentación de los tensores hasta cuando se guarda el modelo.
- Histograma: muestra la distribución de los tensores durante el entrenamiento del modelo
- Distribución: otra forma de visualizar los histogramas
- Imágenes: Permite ver las imágenes de evaluación con las cajas delimitadoras y las máscaras encontradas por el modelo.

3.4. Aprendizaje por transferencia

Un modelo previamente entrenado es una red que se entrenó previamente con un gran conjunto de datos, con el objetivo, típicamente, de realizar una clasificación de imágenes a gran escala. Se puede usar el modelo pre entrenado directamente o realizar el aprendizaje por transferencia personalizando el modelo para una determinada tarea.

El aprendizaje por transferencia, se base en que si un modelo se entrena con un conjunto de datos lo bastante grande y general, este servirá de forma eficiente como un modelo genérico. Este modelo podrá usarse como base, sin tener que empezar desde cero, con todo lo que ya ha aprendido. Existen dos formas de personalizar un modelo pre entrenado, por un laso con extracción de características y por otro aplicando un proceso de ajuste fino, que se conoce en inglés como *fine-tuning*.

La extracción de características se basa en la idea de utilizar las representaciones aprendida por el modelo base para extraer características significativas de nuevas muestras. Se agrega un nuevo clasificador, que entrenara desde cero, encima de este modelo previamente entrenado, pero reutilizara los mapas de características aprendidos previamente. No requiere entrena de nuevo todo el modelo. La red convolucional base ya dispone de características genéricas para clasificar imágenes, pero la parte final de clasificación es específica para el nueva tarea. El proceso de ajuste fino implica ajustar las representaciones de características más relevantes del modelo base para hacerlas más relevantes para la nueva tarea específica. Esto se consigue agregando nuevas capa de clasificador y utilizando las últimas capas del modelo base.

En este proyecto se usará el aprendizaje por transferencia, se partirá de un modelo previamente entrenado con un conjunto grande de imágenes *COCO* con un número elevado de clases, y se personalizara para detectar seis clases de frutas, entrenado nuevamente el modelo con un conjunto mucho más pequeño de imágenes, de las seis clases de fruta, extraídas del conjunto de imágenes *ImageNet*. Esto por tanto permite trabajar con un conjunto de datos reducidos, puesto que no partimos de cero.

3.5. Modelos

En este proyecto se han escogido dos modelos del conjunto de modelos de detección *zoo* de la API de detección de objetos de TensorFlow. Proporcionan un total de cuatro modelos que usan *Mask R-CNN* y que por tanto tienen como salida mascaras además de cajas delimitadoras. Se han escogido los modelos *Mask R-CNN Inception ResNet v2* y *Mask R-CNN ResNet 101*, puesto que son los que mejores resultados de media de precisión, por sus siglas en inglés AP (*Average Precision*), han obtenido sobre el conjunto de imágenes *COCO*, a pesar de que son los que más despacio procesan las imágenes. En la siguiente tabla 3.5 podemos ver cada modelo pre entrenado con la velocidad y la media de precisión obtenida con el conjunto de imágenes *COCO*. Se entiende por velocidad lo que se ha tardado en procesar una imagen de tamaño 600x600 y usando una tarjeta Nvidia GeForce GTX TITAN X, que es la que han usado para entrenar todos los modelos de detección *zoo*.

Modelo	Velocidad (ms)	COCO mAP
mask_rcnn_inception_resnet_v2_atrous_coco	771	36
mask_rcnn_resnet101_atrous_coco	470	33
mask_rcnn_resnet50_atrous_coco	343	29
mask_rcnn_inception_v2_coco	79	25

Cuadro 3.1: Tabla de los modelos, con su velocidad y media de precisión en *COCO*.

3.5.1. Modelo pre entrenado *Mask R-CNN Inception ResNet v2*

Uno de los modelos pre entrenados que se usarán para entrenar en este proyecto es *Mask R-CNN Inception ResNet v2*, a continuación se describe el modelo. Dicho modelo usará una red neuronal *Mask R-CNN*, explicada anteriormente, y con *Inception-Resnet-v2* como el extractor de características.

Inception-Resnet-v2 es una red neuronal convolucional que consta de un total de 27 capas, dicha estructura de capas se puede apreciar en la siguiente figura 3.5. Está basada en la combinación de la estructura de la red neuronal *Inception* y la conexión de la red neuronal Residual, conocida como *ResNet*. Los múltiples filtros convolucionales son combinados con conexiones residuales. El uso de conexiones residuales evita el problema de degradación causado por estructuras muy profundas, y también reduce el tiempo de entrenamiento. En la siguiente figura

se puede apreciar la estructura de capas de una capa Inception.

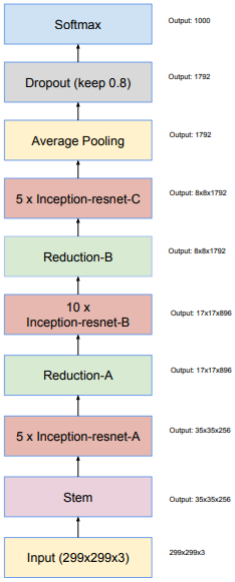

Figura 3.5: Esquema de una red neuronal *Inception-Resnet-v2*.[8]

Inception es una red neuronal convolucional que consta también de un total de 27 capas. Entre estas capas las capas *Inception* son el concepto más importante de la arquitectura. Las capas *Inception* son en realidad la combinación de otras. El hecho de poner una capa convolucional 1x1 al principio se usa para reducir la dimensionalidad. Las capas internas permiten escoger el tamaño del filtro que es relevante para aprender la información necesaria. Eso significa que si el tamaño del objeto es distinto en dos imágenes, la capa trabaja acorde a él y es capaz de reconocerlo.

ResNet es una red neuronal que introduce el concepto de unidad de aprendizaje residual, que pretende aliviar la degradación en las redes neuronales profundas. Esto produce una mejor precisión en la clasificación pero sin aumentar la complejidad del modelo.

3.5.2. Modelo pre entrenado *Mask R-CNN ResNet 101*

El segundo modelo pre entrenado que se usará para entrenar en el presente estudio es *Mask R-CNN ResNet 101*, a continuación se describe el modelo. Dicho modelo usará una red neuronal *Mask R-CNN*, explicada anteriormente, y con *Resnet-101* como el extractor de características.

Resnet-101 es una red neuronal convolucional que consta de 101 capas. Utiliza varias características de las múltiples capas residuales para capturar los cambios del tamaño del objeto, que puede interpretarse como conjuntos de abundantes redes menos profundas. Las entradas de una capa residual inferior se pasan a los nodos de la capa residual superior para poder obtener

simultáneamente características en las salidas de las capas de convolución superiores, inferiores y medias.

3.6. Git

Se ha optado por utilizar un sistema de control de versiones git sobre GitHub. Git es gratuito y de código abierto. Su uso, ayuda no solo a tener el control de las versiones, sino a controlar el avance del proyecto y hacerlo público. Además GitHub permite organizar y documentar el trabajo de forma muy eficiente.

Este proyecto se puede encontrar en el siguiente enlace: https://github.com/pleongi/AutomaticFruitShoppingList.

3.7. Docker

Para garantizar que el proyecto es fácilmente reproducible y portable, se ha utilizado Docker. Docker es un proyecto de código abierto y gratuito que permite ejecutar aplicaciones dentro de contenedores de software, proporcionando aislamiento del sistema. Esto permite ejecutar el código del proyecto en sistemas operativos Linux y Windows, por tanto proporciona también compatibilidad. Un Dockerfile es un documento que contiene todos los comandos necesarios para generar una imagen Docker. El Dockerfile que contiene todo el software y las instalaciones necesarias para ejecutar todo el código de esta práctica se puede encontrar en el Anexo A.

3.8. Jupyter notebook

Para la realización de este trabajo se ha utilizado Jupyter Notebook como entorno de desarrollo. Jupyter Notebook es una aplicación de código abierto y gratuita creada por la organización sin ánimo de lucro Proyecto Jupyter. Es un entorno informático interactivo basado en la web que permite crear y compartir documentos JSON de Jupyter Notebook. Los documentos creados en Jupyter pueden exportarse, entre otros formatos, a HTML, PDF, Markdown o Python . Además, permiten incluir en ellos desde código junto con su salida de ejecución hasta gráficos y texto (usando Markdown).

3.9. Python

Se ha empleado el lenguaje de programación Python para desarrollar en código en este proyecto. Se ha escogido este lenguaje puesto que es el que recomienda Tensorflow para la

implementación de redes neuronales, además cuenta con una amplia comunidad que proporciona soporte continuo a cualquier problema. Se trata de un lenguaje interpretado, es decir no necesita ser previamente compilado, y se detectan los errores en tiempo de ejecución. Es un lenguaje de código abierto y gratuito. Se ha utilizado la versión Python 3.6.

Capítulo 4

Desarrollo

4.1. Preparación de los datos

Los conjuntos de imágenes que se han usado en este proyecto son los de *ImageNet* y los de COCO. COCO[28], por sus siglas en inglés *Common Objects in Context*, es un conjunto de imágenes para la detección de objetos y la segmentación. Tiene un total de 330.000 imágenes de las que más de 200.00 están etiquetadas con 80 categorias de objetos distinto y un total de 1,5 millones de objetos detectados en dichas imagenes. COCO es el conjunto de datos utilizado para pre entrenar los modelos de detección *zoo*, proporcionados por la API de detección de objetos de TensorFlow, que se van a usar en este proyecto.

ImageNet[33] es un conjunto de datos organizado según la jerarquía de sustantivos extraída de *WordNet*[34], una base de datos léxica de la lengua inglesa, en el que cada sustantivo se agrupa en conjuntos de sinónimos cognitivos denominados en inglés *synsets*. Existen más de 100.000 conjunto de sinónimos en *WordNet*. *ImageNet* tiene una media de 1000 imágenes por cada conjunto de sinónimos. En total dispone de 14.197.122 imágenes, de 1.034.908 anotaciones de cajas delimitadoras de imágenes y un total de 21.841 *synsets*. Este conjunto de datos se usará en este proyecto para entrenar el modelo previamente entrenado con las imágenes de COCO.

En nuestro caso queremos detectar seis clases de frutas, plátanos, manzanas, peras, piñas, limones y naranjas, y *ImageNet* dispone de imágenes de todas estas clases de frutas, a diferencia de COCO, que solo dispone de imágenes de tres de las seis clases de frutas que queremos reconocer. COCO no dispone de imágenes de peras, piñas ni de limones. Es por eso que el modelo pre entrenado no reconocerá estos tres tipos de fruta sin ser entrenado nuevamente con *ImageNet*.

En la siguiente figura mostramos un ejemplo de la detección de fruta con el modelo pre entrenado de detección *zoo* proporcionado por la API de detección de objetos de TensorFlow. Podemos observar...

4.1.1. Adquisición y partición de las imágenes

Para adquirir los datos se ha desarrollado el código que se puede encontrar en el Anexo B. Este código adquiere un total de 250 imagenes para cada clase de fruta, un total de 1500 imágenes de la página oficial de *ImageNet*. De todas las imágenes se ha decidido partirlas en un conjunto de entrenamiento y un conjunto de evaluación.

Para el conjunto de entrenamiento se han seleccionado un 80 % de las imágenes de cada clase de fruta, por tanto para entrenar el modelo se usaran 200 imágenes de cada clase de fruta, un total de 3000 imágenes de entrenamiento.

Para el conjunto de evaluación, se usara el 20 % restante, por tanto se seleccionaran 50 imágenes de cada clase de fruta, un total de 300 imágenes para evaluar los modelos.

4.1.2. Generar las anotaciones de las imágenes

Los modelos de detección de objetos requieren que los objetos de las imágenes estén anotados. En nuestro caso el modelo necesita por un lado las cajas delimitadoras de los objetos en las imágenes y por otro lado las máscaras de los objetos en las imágenes. De esta forma el modelo puedo extraer por un lado las coordenadas de la caja delimitadora de un objeto en la imagen y por otro los pixel para la localización del objeto dentro de la caja delimitadora.

4.1.2.1. Generar las cajas delimitadoras

En el procesamiento de las imágenes, una caja delimitadora corresponde a las coordenadas del borde rectangular que encierra completamente un objeto en una imagen, en nuestro caso una pieza de fruta en las imágenes. *ImageNet* ya tiene anotadas muchas de estas cajas delimitadoras en las imágenes, de hecho de las seis clases de frutas, cinco tenían todas las anotaciones de las cajas delimitadoras disponibles para descargar, pero en el caso de las peras faltaban muchas anotaciones por realizar.

Para la generación de las cajas delimitadores de las peras en las imágenes que faltaban por anotar, se ha usado una herramienta gráfica llamada *LabelImg* [35], que además está recomendada por TensorFlow. Esta herramienta está escrita en Python y se ha usa Qt para su interfaz gráfica, que es un software de código abierto usado por multitud de programas para generar su interfaz gráfica de usuario.

A continuación en la siguiente figura 4.1 podemos ver un ejemplo en el que creamos una caja delimitadora de una pero en una de las imágenes.

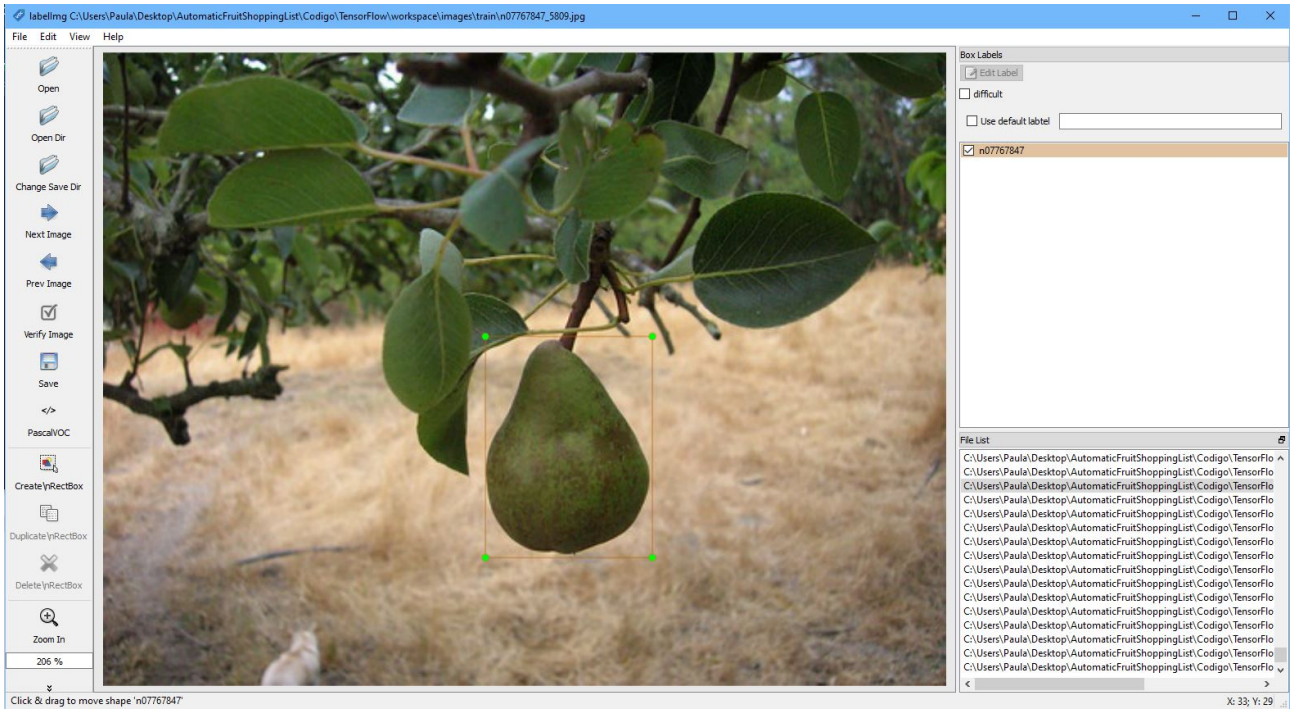

Figura 4.1: Generación de una caja delimitadora de una pera de una imagen del conjunto de entrenamiento.

Las anotaciones de las cajas delimitadores generadas con la herramienta se guardan en formato XML si se escoge el formato *PASCAL VOC*, pero también soporta formato *YOLO* y entonces genera las anotaciones e las cajas delimitadores en ficheros JSON. En nuestro caso usaremos *PASCAL VOC* puesto que es el formato que usa *ImageNet* para generar sus anotaciones de cajas delimitadoras.

Podemos ver el fichero XML resultante de la anotación creada para esta imagen 4.1 en el Anexo C.

En este fichero XML podemos observar los siguientes campos:

- Carpeta: en el que se indica el directorio donde se encuentran la imagen.
- Nombre del fichero de la imagen.
- Tamaño: en el que se indica la anchura y altura y la profundidad de la imagen. En el caso de una imagen a color la profundidad es 3.
- Objeto: se indica el nombre o etiqueta del objeto que se identifica, en nuestro caso el *sysnet* de la fruta. Se indica si el objeto es la totalidad de la imagen con un 1 al campo *truncated* o si por el contrario no ocupa toda la imagen se indica con un 0. Existe un

campo de dificultad, para expresar si es difícil de reconocer el objeto en la imagen con un 1 o si es fácil con un 0. Y por último se especifican las posiciones de los pixeles de la caja delimitadora como *xmin*, *ymin*, *xmax* y *ymax*.

En función de la cantidad de objetos reconocidos en una imagen habrán uno o más campo de objetos en el fichero XML.

4.1.2.2. Generar las máscaras

Para la generación de máscaras de las frutas en las imágenes, se ha utilizado una herramienta llamada *Pixel Annotation Tool* [36]. La herramienta está desarrollada utilizando Qt para su interfaz gráfica y permite generar la máscara manualmente de forma rápida. Utiliza el algoritmo de transformación divisoria [37], en inglés *watershed* de OpenCV. OpenCV es una librería de código abierto desarrollada por Intel. Es una de las librerías más populares para visión artificial, de hecho sus siglas corresponden a Visión Artificial Abierta, en inglés *Open Computer Vision*. Se colorean los objetos deseados de un color y el resto de otro color y el algoritmo se ejecuta para que acabe de detectar todos los objetos sin necesidad de pintar toda la imagen, solo con pintar ciertos puntos de la imagen es suficiente. En la siguiente figura 4.2 podemos ver los pasos seguidos para la generación de una máscara de un limón de una de las imágenes del conjunto de entrenamiento.

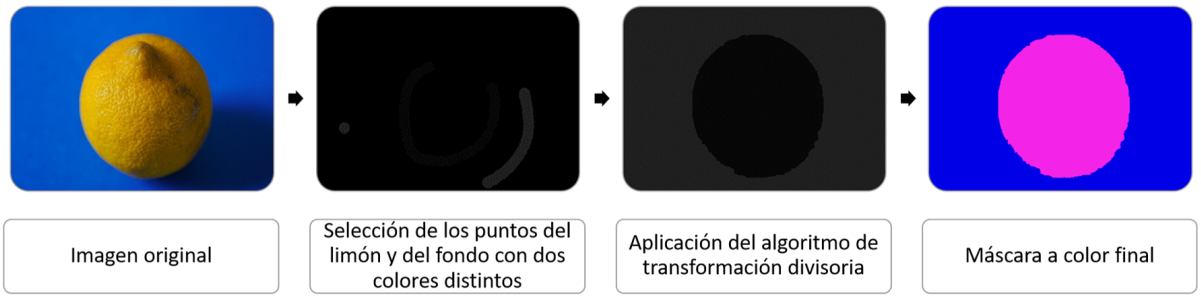

Figura 4.2: Generación de una máscara de un limón de una imagen del conjunto de entrenamiento.

Las máscaras a color obtenidas con esta herramienta se guardan en formato PNG, y se han ubicado todas en una carpeta llamada *masks*.

4.1.3. Crear el mapa de etiquetas

La API de detección de objetos de TensorFlow requiere de un mapa de etiquetas con extensión .pbtxt, es decir necesita un mapa de los nombres de las clases de los datos con un valor numérico. En nuestro caso las clases o etiquetas son los valores de los *synsets* proporcionados por *ImageNet*, que corresponden a las seis clases de fruta. En la siguiente tabla 4.1.3 se muestra la correspondencia de estos *synsets* con las clases de frutas y con su identificador.

Fruta	*Synset*	Identificador numérico
Manzana	n07739125	1
Plátano	n07753592	2
Naranja	n07747607	3
Limón	n07749582	4
Piña	n07753275	5
Pera	n07767847	6

Cuadro 4.1: Tabla de correspondencia, entre las clases de frutas, su *synset* y su identificador numérico.

Para el caso de las manzanas el mapa de etiquetas generado para TensorFlow sería:

```
item {
    id: 1
    name: 'n07739125'
}
```

El mapa de etiquetas completo generado para el proyecto se encuentra en el Anexo D. En el repositorio del proyecto el mapa de etiquetas se encuentra en el siguiente directorio.

4.1.4. Generar los TensorFlow Records

La API de detección de objetos de TensorFlow lee los datos usando un formato de fichero llamado TFRecord. Este fichero es una representación comprimida de la imagen, sus cajas delimitadoras y la máscara. En el Anexo E podemos ver el código que se ha usado para convertir las anotaciones de las cajas delimitadoras de las imágenes en formato XML y *PASCAL VOC*,

junto con las máscaras generadas en formato PNG y las imágenes en un fichero TFRecord. Se debe proporcionar también el fichero con el mapa de etiquetas para etiquetar las frutas con un identificador numérico. Se han generado dos fichero TFRecord, uno para el conjunto de entrenamiento y otro para el conjunto de evaluación. Estos dos ficheros TFRecord se han ubicado en la carpeta.

4.2. Configurar los modelos

https://github.com/tensorflow/models/blob/master/research/object$_detection/g3doc/configuring_j$

4.3. Entrenar los modelos

4.4. Evaluar los modelos

Bibliografía

[1] D. S. y. S. C. H. Xiongwei Wua, "Recent advances in deep learning for object detection.," URL https://arxiv.org/pdf/1908.03673.pdf.

[2] T. D. y. J. M. Ross Girshick, Jeff Donahue, "Rich feature hierarchies for accurate object detection and semantic segmentation.," 2014. URL https://arxiv.org/abs/1311.2524.

[3] P. D. y. R. B. G. K. He, G. Gkioxari, "Mask r-cnn.," 2017. URL https://arxiv.org/pdf/1703.06870.pdf.

[4] R. G. y. A. F. Joseph Redmon, Santosh Divvala, "You only look once: Unified, real-time object detection.," 2016. URL https://arxiv.org/abs/1506.02640.

[5] A. Ouaknine, "Review of deep learning algorithms for object detection.," 2018. URL https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852.

[6] M. Alvarado, "Pronóstico del tipo de cambio usd/mxn con redes neuronales de retropropagación," 2017. URL https://www.researchgate.net/figure/Red-neuronal-artificial-de-cuatro-capas_fig1_323985249.

[7] T. Razmi, "Mask r-cnn," 2019. URL https://medium.com/@tibastar/mask-r-cnn-d69aa596761f.

[8] V. V. y. A. A. Christian Szegedy, Sergey Ioffe, "Inception-v4, inception-resnet and the impact of residual connections on learning," 2016. URL https://arxiv.org/pdf/1602.07261.pdf.

[9] V. M. OSORIO, "Amazon lanza la cesta de la compra por suscripción.," URL https://www.expansion.com/economia-digital/companias/2017/02/01/589187f0e5fdeabb0d8b45f1.html.

[10] F. RETAIL, "¿un carro de la compra inteligente? caper lo consigue.," URL https://www.foodretail.es/industria-auxiliar/carro-inteligente-compra-caper_2_1290790909.html.

[11] P. V. y Michael J. Jones, "Robust real-time object detection," 2001. URL https://www.researchgate.net/publication/215721846_Robust_Real-Time_Object_Detection.

[12] N. D. y Bill Triggs, "Histograms of oriented gradients for human detection," 2005. URL https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

[13] I. S. y. G. E. H. A. Krizhevsky, "Imagenet classification with deep convolutional neural networks.," 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[14] X. Z. M. M. R. F. y. Y. L. Pierre Sermanet, David Eigen, "Overfeat: Integrated recognition, localization and detection using convolutional networks.," 2014. URL https://arxiv.org/abs/1312.6229.

[15] R. Girshick, "Fast r-cnn.," 2015. URL https://arxiv.org/abs/1504.08083.

[16] Y. J. P. S. S. R. D. A. D. E. V. V. y. A. R. Christian Szegedy, Wei Liu, "Going deeper with convolutions.," 2015. URL https://arxiv.org/abs/1409.4842.

[17] R. G. y. J. S. Shaoqing Ren, Kaiming He, "Faster r-cnn: Towards real-time object detection with region proposal networks.," 2016. URL https://arxiv.org/abs/1506.01497.

[18] S. R. y. J. S. Kaiming He, Xiangyu Zhang, "Deep residual learning for image recognition.," 2016. URL https://arxiv.org/abs/1512.03385.

[19] K. H. y. J. S. Jifeng Dai, Yi Li, "R-fcn: Object detection via region-based fully convolutional networks.," 2016. URL https://arxiv.org/abs/1605.06409.

[20] D. E. C. S. S. R. C.-Y. F. y. A. C. B. Wei Liu, Dragomir Anguelov, "Ssd: Single shot multibox detector.," 2016. URL https://arxiv.org/abs/1512.02325.

[21] J. R. y A. Farhadi, "Yolo9000: Better, faster, stronger.," 2017. URL https://arxiv.org/pdf/1612.08242.pdf.

[22] J. S. y. Q. V. L. Barret Zoph, Vijay Vasudevan, "Learning transferable architectures for scalable image recognition," 2018. URL https://arxiv.org/abs/1707.07012.

[23] X. Z. G. M. X. X. y. J. S. Yukang Chen, Tong Yang, "Detnas: Backbone search for object detection.," 2019. URL https://arxiv.org/abs/1903.10979.

[24] W. K. S. P. y. J. D. Payne, A., "Estimating mango crop yield using image analysis using fruit at 'stone hardening' stage and night time imaging," 2014. URL https://www.researchgate.net/publication/259511023_Estimating_mango_crop_yield_using_image_analysis_using_fruit_at_'stone_hardening'_stage_and_night_time_imaging.

[25] F. D. B. U. T. P. y. C. M. Inkyu Sa, Zongyuan Ge, "Deep-fruits: a fruit detection system using deep neural networks.," 2016. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5017387/.

[26] M. R. y Clay Sheppard, "Deep count: Fruit counting based on deep simulated learning.," 2017. URL https://www.mdpi.com/1424-8220/17/4/905.

[27] "Nvidia," URL https://developer.nvidia.com/.

[28] "Common objects in context," URL http://cocodataset.org/#home.

[29] "The kitti vision benchmark suite," URL http://www.cvlibs.net/datasets/kitti/.

[30] "Open images dataset," URL http://www.cvlibs.net/datasets/kitti/.

[31] "Ava," URL https://research.google.com/ava/.

[32] "inaturalist competition," URL https://github.com/visipedia/inat_comp/blob/master/2017/README.md#bounding-boxes.

[33] "Imagenet," URL http://www.image-net.org/.

[34] "Wordnet. a lexical database for english," URL https://wordnet.princeton.edu/.

[35] "Labelimg," URL https://github.com/tzutalin/labelImg.

[36] A. Bréhéret, "Pixelannotationtool," 2017. URL https://github.com/abreheret/PixelAnnotationTool.

[37] "Transformación divisoria," 2019. URL https://es.wikipedia.org/wiki/Transformaci%C3%B3n_divisoria.

Anexos

Apéndice A

Anexo I: Dockerfile del proyecto.

```
# Copyright 2018 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied
    .
# See the License for the specific language governing permissions and
# limitations under the License.
#
    #==========================================================

FROM tensorflow/tensorflow:nightly-devel-py3

# Get the tensorflow models research directory, and move it into tensorflow
# source folder to match recommendation of installation
RUN git clone --branch r1.13.0 --depth 1 https://github.com/tensorflow/models.git && \
    mv models /tensorflow/models
```

```
ARG DEBIAN_FRONTEND=noninteractive

RUN python3 -m pip --no-cache-dir install --upgrade \
    pip \
    setuptools
# Install object detection api dependencies
RUN pip install Cython && \
    pip install contextlib2 && \
    pip install beautifulsoup4 && \
    pip install opencv-python && \
    pip install lxml && \
    pip install matplotlib

# Install the Tensorflow Object Detection API from here
#RUN pip install -U --pre tensorflow=="2.*" until 1.9 works and it is already there

# Get protoc 3.0.0, rather than the old version already in the container
RUN curl -OL "https://github.com/google/protobuf/releases/download/v3.0.0/protoc
    -3.0.0-linux-x86_64.zip" && \
    unzip protoc-3.0.0-linux-x86_64.zip -d proto3 && \
    mv proto3/bin/* /usr/local/bin && \
    mv proto3/include/* /usr/local/include && \
    rm -rf proto3 protoc-3.0.0-linux-x86_64.zip

# Run protoc on the object detection repo
RUN cd /tensorflow/models/research && \
    protoc object_detection/protos/*.proto --python_out=.

# Set the PYTHONPATH to finish installing the API
ENV PYTHONPATH $PYTHONPATH:/tensorflow/models/research:/tensorflow/models/
    research/slim



RUN pip install /tensorflow/models/research/

#ENV cd /tensorflow/models/research && python setup.py build
```

```
#ENV cd /tensorflow/models/research && python setup.py install


# Install pycocoapi
RUN git clone --depth 1 https://github.com/cocodataset/cocoapi.git && \
    cd cocoapi/PythonAPI && \
    make -j8 && \
    cp -r pycocotools /tensorflow/models/research && \
    cd ../../ && \
    rm -rf cocoapi

WORKDIR /tensorflow


EXPOSE 8888
CMD ["jupyter", "notebook", "--allow-root", "--notebook-dir=/", "--ip=0.0.0.0",
    "--port=8888", "--no-browser"]
```

Apéndice B

Anexo II: Código usado para la adquisición de las imágenes.

```
from bs4 import BeautifulSoup
import numpy as np
import requests
import cv2
import urllib
import os
import shutil
img_rows, img_cols = 32, 32 #number of rows and columns to convert the images to
input_shape = (img_rows, img_cols, 3)#format to store the images (rows, columns,channels)
    called channels last
def url_to_image(url):
    # download the image, convert it to a NumPy array, and then read it into OpenCV
        format
    resp = urllib.request.urlopen(url)
    image = np.asarray(bytearray(resp.read()), dtype="uint8")
    image = cv2.imdecode(image, cv2.IMREAD_COLOR)
    return image
n_of_training_images=200#the number of training images to use
n_of_validating_images=50#the number of validating images to use
synsetsIds=['n07739125','n07767847','n07753592','n07749582','n07747607','n07753275']
synsetsNames=['n07739125','n07767847','n07753592','n07749582','n07747607','n07753275']
#synsetsNames=['apple','pear','banana','lemon','orange', 'pineapple']
```

```
for i in range(0,len(synsetsIds)):
    url="http://www.image-net.org/api/text/imagenet.synset.geturls.getmapping?wnid="
        +synsetsIds[i]
    page = requests.get(url)
    train_path="TensorFlow/workspace/images/train/"+synsetsNames[i]+"/"
    validation_path="TensorFlow/workspace/images/test/"+synsetsNames[i]+"/"

    shutil.rmtree(train_path, ignore_errors=True)
    os.makedirs(train_path)

    shutil.rmtree(validation_path, ignore_errors=True)
    os.makedirs(validation_path)

    #puts the content of the website into the soup variable, each url on a different line
    soup = BeautifulSoup(page.content, 'html.parser')

    str_soup=str(soup)#convert soup to string so it can be split
    type(str_soup)
    split_urls =str_soup.split('\r\n')#split so each url is a different possition on a list
    #Train data:
    print("Downloading training data: "+synsetsNames[i])
    progress=0
    progress_tmp=0
    while True:
    #for progress in range(n_of_training_images):#store all the images on a directory
        if split_urls[progress_tmp] != None:
            try:
                I = url_to_image(split_urls[progress_tmp].split(' ', 1)[1])
                if (len(I.shape))==3: #check if the image has width, length and channels
                    save_path = train_path+str(split_urls[progress_tmp].split(' ', 1)[0])+'
                        .jpg'#create a name of each image
                    cv2.imwrite(save_path,I)
                    progress+=1
                    #print ("Progress:",(progress/n_of_training_images)*100)
            except:
                None
```

```
            progress_tmp+=1
            if progress == n_of_training_images:
                break

    #Testing data:
    print("Downloading testing data: "+synsetsNames[i])
    progress2=0
    progress_tmp2=0
    while True:
        #for progress in range(n_of_validating_images):#store all the images on a directory
        if not split_urls[progress+progress2] == None:
            try:
                #get images that are different from the ones used for training
                I = url_to_image(split_urls[progress_tmp+progress_tmp2].split(' ', 1)[1])
                if (len(I.shape))==3: #check if the image has width, length and channels
                    save_path = validation_path+str(split_urls[progress_tmp+progress_tmp2
                        ].split(' ', 1)[0])+'.jpg'#create a name of each image
                    cv2.imwrite(save_path,I)
                    progress2+=1
                    #print ("Progress:",(progress2/n_of_validating_images)*100)
            except:
                None
            progress_tmp2+=1
            if progress2 == n_of_validating_images:
                break
```

Apéndice C

Anexo III: Ejemplo de una anotación de una caja delimitadora.

A continuación se muestra la anotación de una caja delimitadora generada con la herramienta *LabelImg* en formato *PASCAL VOC* de la imagen n07767847_5209.jpg, en la que aparece una pera. Por eso podemos ver un solo objeto anotado con el nombre del *sysnset* que corresponde a la pera, n07767847.

```
<annotation>
    <folder>n07767847</folder>
    <filename>n07767847_5809.jpg</filename>
    <path>C:\Users\Paula\Desktop\AutomaticFruitShoppingList\Codigo\TensorFlow\
        workspace\images\train\n07767847\n07767847_5809.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>500</width>
        <height>375</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>n07767847</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        < difficult >0</difficult>
```

```
        <bndbox>
            <xmin>227</xmin>
            <ymin>168</ymin>
            <xmax>324</xmax>
            <ymax>298</ymax>
        </bndbox>
    </object>
</annotation>
```

Apéndice D

Anexo IV: El mapa de etiquetas del proyecto.

A continuación se muestra el fichero label_map.pbtxt, que contiene el mapa de etiquetas del proyecto. Con los *sysnsets* de las seis clases de frutas y sus correspondientes identificadores numéricos.

```
item {
  id: 1
  name: 'n07739125'
}

item {
  id: 2
  name: 'n07753592'
}

item {
  id: 3
  name: 'n07747607'
}

item {
  id: 4
  name: 'n07749582'
}
```

```
item {
  id: 5
  name: 'n07753275'
}

item {
  id: 6
  name: 'n07767847'
}
```

Apéndice E

Anexo V: Código usado para generar los ficheros de entrenamiento y evaluación en formato TFRecord.