Universitat Oberta de Catalunya

uoc.edu

Estudio de viabilidad para el control de existencias mediante reconocimiento visual y redes neuronales convolucionales

Antonio Arencibia Guerra

Máster Universitario en Ciencia de Datos

Minería de datos y machine learning

Jerónimo Hernández González

Jordi Casas Roma

FICHA DEL TRABAJO FINAL

Título del trabajo: Estudio de viabilidad para el control de existencias mediante reconocimiento visual y redes neuronales convolucionales.	*Estudio de viabilidad para el control de existencias mediante reconocimiento visual y redes neuronales convolucionales.*
Nombre del autor:	*Antonio Arencibia Guerra*
Nombre del consultor/a:	Jerónimo Hernández González
Nombre del PRA:	*Jordi Casas Roma*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	Máster Universitario en Ciencia de Datos
Área del Trabajo Final:	Minería de datos y machine learning
Idioma del trabajo:	*Español*
Palabras clave	Detección de objetos, Recuento de objetos, Redes neuronales convolucionales

A Mar,
por siempre estar ahí,
por ser mi fuente de energía e inspiración
y a Diego, nuestro hijo y compañero de madrugadas

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor en este trabajo final de máster, Jerónimo Hernández González, todo el apoyo, ánimo y buenos consejos durante el proceso de realización del mismo. Ha sido un auténtico lujo tenerlo como tutor.

Por supuesto, también a mi familia, que, a pesar de estar pasando momentos complicados, han estado dándome todo el apoyo que necesitaba, sin el cual, no hubiera sido posible finalizar el presente trabajo.

Resumen

En el presente estudio se aborda la viabilidad de un sistema de control de existencias para una variedad limitada de frutas en un entorno tridimensional acotado. El sistema se basa en técnicas de aprendizaje profundo mediante la aplicación de redes neuronales convolucionales para la detección de objetos en imágenes. Para su entrenamiento, se parte de una base de imágenes relevantes que contienen los diferentes tipos de fruta que se quiere identificar.

El reconocimiento de las diferentes clases de fruta en las imágenes se realiza diariamente y su conteo por clases permite obtener instantáneas del estado de las existencias del frutero. El resultado son series temporales con los recuentos diarios de las diferentes clases de fruta. Este histórico alimenta a su vez a un sistema de control de existencias con el que se puede solicitar la necesidad de existencias en base al análisis comparativo de los datos del histórico y pedidos realizados.

Los resultados finales obtenidos han demostrado que la estrategia seguida no es suficiente para conseguir el objetivo de automatización del proceso de control y actualización de existencias. La enorme complejidad en la detección de piezas de frutas apiladas, ocultas o semiocultas, incluso con un número reducido de clases y unidades, en un entorno acotado como es el de un frutero, hacen que se necesite de modelos y técnicas más robustos. No obstante, pese al margen de error, el sistema funciona y puede ser un punto de partida para mejorar su robustez o para servir de base en futuros proyectos.

Abstract

This project investigates the viability of a stock control system for a limited range of fruit in a three-dimensional, controlled environment. The system is based on deep learning techniques with the application of convolutional neural networks being used to detect objects in images. Training begins with relevant images that contain the different types of fruit to be identified.

Recognition of the different types of fruit in the pictures is carried out daily and counting these by type enables a snapshot of the stock status of the fruit bowl to be obtained. The result is a data series of the daily tally of different types of fruit. This data series in turn feeds back into a stock control system which can be used to order the necessary supplies based on a comparative analysis of historical data and orders made.

The final results obtained show that the strategy followed was not sufficient to achieve the objective of the automation of stock control and updating. The huge complexity involved in identifying piled up and hidden or half-hidden fruit, even with a limited types and quantity of fruit, in an enclosed environment like the fruit bowl, mean that more robust techniques and models are needed. However, in spite of its margin of error, the system works and could serve as a starting point to be improved upon or a basis for future projects.

Tabla de contenidos

1. Introducción 1
  1.1. Contexto y justificación del Trabajo 1
  1.2. Objetivos del Trabajo 2
    1.2.1. Objetivo principal 2
    1.2.2. Objetivos secundarios 2
  1.3 Enfoque y método seguido 3
  1.4. Planificación del Trabajo 3
    1.4.1. Recursos 3
    1.4.2. Planificación 4
  1.6. Breve descripción de los capítulos de la memoria 5
2. Estado del arte 6
3. Materiales y métodos 14
  3.1. Anaconda 14
  3.2. Python 14
  3.3. Tensorflow 14
    3.3.1. API para la detección de objetos de Tensorflow 15
    3.3.2. TensorBoard 17
  3.4. Redes neuronales 17
    3.4.1. Funciones de activación 20
    3.4.2. Aprendizaje 21
    3.4.3. Redes neuronales convolucionales 24
  3.5. Detección de objetos en imágenes 28
  3.6. Matriz de confusión para múltiples valores de recuento 31
4. Bases de datos 33
  4.1. Base de imágenes de entrenamiento 33
    4.1.1 Procesamiento de las imágenes 34
    4.1.2. Creación de las anotaciones 35
  4.2. Base de imágenes de producción 37
5. Modelos 38
  5.1. Modelos pre-entrenados ofrecidos por la API 38
  5.2. Modelo pre-entrenado *Faster* R-CNN Inception v2 39
  5.3. Modelo pre-entrenado *Faster* R-CNN ResNet-101 41
  5.4. Sistema de control de existencias 42
6. Ejecución 45
  6.1. Ejecución del entrenamiento de los modelos ajustados 45
  6.2. Ejecución con las imágenes de producción 46
  6.3. Ejecución del sistema de control de existencias 46
7. Resultados 47
  7.1. Resultados con las imágenes de evaluación 47
  7.2. Resultados con las imágenes de producción. 51
  7.3. Resultados del sistema de control de existencias 53
8. Discusión 59
  8.1. Mejoras y modificaciones del proceso. 59
  8.2. Trabajos futuros 60
9. Conclusiones 61
10. Bibliografía 62

1. Introducción

1.1. Contexto y justificación del Trabajo

- Contexto

Tradicionalmente la automatización de sistemas de control ha necesitado de un etiquetado o clasificación previa, ya sea manual o mecánica, del producto que se quería controlar de forma autónoma basándose en dicha etiqueta identificativa. Esta clasificación implica identificar o extraer previamente las características de las clases de los productos que se quieran clasificar, para posteriormente separar, basándonos en dichas características, los productos finales identificados en dichas clases. La propia clasificación se verá limitada, en la mayoría de los casos, por la forma en la que se haya determinado por qué un producto pertenece a una clase y porqué otro no.

Los sistemas de reconocimiento de objetos en imágenes, por medio de técnicas de aprendizaje profundo, permiten extraer las características de las clases más allá de lo obvio o más relevante. Esto puede llevarnos a clasificaciones o detectores de productos, servicios o incluso personas, mucho más eficientes. Esto ya está ahorrando tiempo y dinero, no sólo para las empresas que implanten dichos sistemas inteligentes, sino también a los usuarios que disfruten de esas innovaciones. La necesidad que se cubre es precisamente el aumento de la eficacia y la disminución de errores en el control de existencias en un entorno controlado y acotado.

En un mundo donde la información es principalmente audiovisual, el análisis inteligente de objetos en imágenes y vídeos es enormemente relevante para los estudios de ciencia de datos, siendo, de hecho, una de las líneas de investigación más importantes actualmente. Estamos viviendo una época en donde se está produciendo una verdadera revolución en el modo en que interactuamos con los datos y la información que nos rodea. La unión de las técnicas de inteligencia artificial y el cerebro humano está produciendo formas mucho más eficientes de diagnóstico o control. Muchas de las maneras tradicionales de toma de decisiones, que hoy en día aún son las que predominan, quedaran desfasadas en poco tiempo.

Los sistemas basados en aprendizaje de máquina se basan en proporcionar todos los datos analizados y clasificados a través de un modelo, para que al final un humano los examine y tome la última decisión. En este estudio, el último paso, la toma de decisión es efectuada por el mismo sistema al realizar el pedido o estimando la necesidad de un producto. Se estudiará la forma de que el sistema se pueda retroalimentar de sus propios cambios solicitados para la actualización de existencias. Por otro lado, habrá que tener en cuenta casos inesperados de existencias de una determinada clase que no se hayan solicitado y que, sin embargo, aparezcan en la última imagen de control.

El presente estudio abordará las diferentes vías de conseguir lo que será un sistema de control de existencias de diferentes clases fruta por medio del reconocimiento de objetos en imágenes y la aplicación de técnicas de aprendizaje profundo.

- Justificación.

Debido a la formación y experiencia laboral en departamentos de Informática y Marketing durante los últimos años, mi interés por la extracción de conocimiento para la toma de decisiones, basado en el análisis de los datos, no ha hecho más que crecer.

El presente proyecto proporciona un espacio donde poder sacar partido a gran parte del conocimiento adquirido durante la realización del Máster de Ciencia de Datos, además de permitirme afrontar nuevos desafíos al trabajar con el análisis de imágenes. Es motivante tener la oportunidad de plasmar todo ello en un proyecto innovador.

1.2. Objetivos del Trabajo

1.2.1. Objetivo principal

El objetivo principal es el estudio de la viabilidad de una aplicación real de redes neuronales convoluciones en un problema de detección de clases de fruta en imágenes y su recuento en un determinado momento marcado por una fecha. Esto nos permitirá la posible automatización del proceso de control y actualización de existencias.

El estudio tendrá varios objetivos secundarios que se deberán realizar de forma que se pueda alcanzar el objetivo final deseado.

1.2.2. Objetivos secundarios

- Búsqueda o creación de base de datos de imágenes para entrenar y evaluar el modelo o modelos.
- Exploración de los datos y procesamiento de las imágenes.
- Creación y configuración del modelo para la detección de objetos en imágenes.
- Ejecución del entrenamiento del modelo con las imágenes de la base de imágenes.
- Generación de la base de imágenes de producción.
- Ejecución del modelo entrenado para el reconocimiento de los objetos y sus clases en las imágenes de producción, así como su conteo por clases.
- Desarrollo del sistema de control de existencias que permita realizar actualizaciones basándose en los datos del reconocimiento de las diferentes clases de fruta en las imágenes.
- Evaluación de los resultados obtenidos.

1.3 Enfoque y método seguido

El primer punto para tener en cuenta será la base de datos de imágenes. Se buscará una base de datos con imágenes etiquetadas, con unas dimensiones adecuadas y con los objetos relevantes al estudio. Se crea la base de datos de imágenes de distintos tipos de frutas etiquetadas donde las clases de objetos son los distintos tipos de fruta.

Es de suma importancia este punto dada la naturaleza del estudio, los algoritmos de deep learning hacen uso de esta base de datos para extraer las características de las distintas clases de frutas en las imágenes que se generen para el control de existencias planteado. Con la base de datos adecuada se procede al estudio de las diferentes herramientas y técnicas existentes para la programación de, por un lado, un detector de las distintas clases de frutas en las imágenes de producción y, por otro lado, un contador de clases para obtener un recuento.

Una vez detectadas las clases de fruta y realizado su recuento, se obtiene un listado con las diferentes clases y su recuento. En este punto se compararán los datos con los datos de existencias mínimas. Finalmente se evaluará si existe la necesidad de un pedido o no dependiendo del resultado de dicha comparación. Se deberán valorar también los datos recogidos para formar un histórico de datos y darle el elemento temporal que necesita el sistema. Para ello se añade una fecha a los datos que se recojan durante el estudio. Para todos los cálculos del estudio se usará el punto como separador decimal.

Se plantea utilizar Python como lenguaje de programación y la API para la detección de objetos en imágenes de Tensorflow como base para la creación y entrenamiento del modelo basado en redes neuronales convolucionales. Este modelo debe detectar y contar las clases de frutas en las imágenes de producción.

1.4. Planificación del Trabajo

1.4.1. Recursos

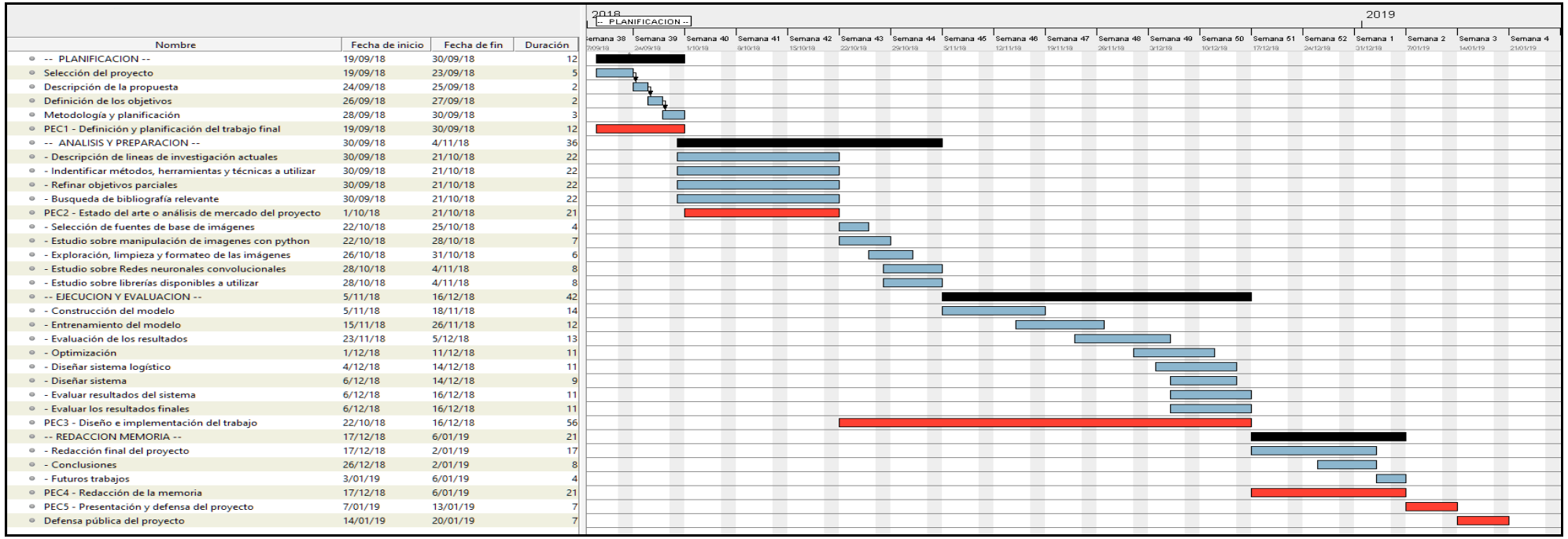
- Ordenador personal de sobremesa con las siguientes características:
    - Procesador: Intel® Core ™i7-6700 CPU @ 4.00 GHZ.
    - RAM instalada: 16 GB.
    - Tarjeta gráfica: NVIDIA GeForce GTX 1060 3GB.
    - Disco Duro SSD Kingston SUV400S37240G.
    - Sistema Operativo Windows 10 Home Edition 64 bits.
- Base de imágenes para el entrenamiento y evaluación del modelo
- Entorno de desarrollo de código abierto Anaconda..
- API para la detección de objetos de Tensorflow.
- Cámara del dispositivo móvil Smartphone Samsung S7 con la que obtendremos las imágenes de producción.

1.4.2. Planificación

En la siguiente tabla se detallan separadamente los trabajos parciales realizados o PECs, que se deben entregar durante el desarrollo del proyecto y sus fechas de entrega.

	Descripción	**Fecha de entrega**
PEC1	Definición y planificación del trabajo final	30/09/18
PEC2	Estado del arte o análisis de mercado	21/10/18
PEC3	Diseño e implementación del trabajo	16/12/18
PEC4	Redacción de la memoria	06/01/19
PEC5	Presentación y defensa del proyecto	13/01/19

La planificación del trabajo y sus diferentes fases están representadas en el siguiente diagrama de Gantt.



Nombre	Fecha de inicio	Fecha de fin	Duración
-- PLANIFICACION --	19/09/18	30/09/18	12
Selección del proyecto	19/09/18	23/09/18	5
Descripción de la propuesta	24/09/18	25/09/18	2
Definición de los objetivos	26/09/18	27/09/18	2
Metodología y planificación	28/09/18	30/09/18	3
PEC1 - Definición y planificación del trabajo final	19/09/18	30/09/18	12
-- ANALISIS Y PREPARACION --	30/09/18	4/11/18	36
- Descripción de lineas de investigación actuales	30/09/18	21/10/18	22
- Indentificar métodos, herramientas y técnicas a utilizar	30/09/18	21/10/18	22
- Refinar objetivos parciales	30/09/18	21/10/18	22
- Busqueda de bibliografía relevante	30/09/18	21/10/18	22
PEC2 - Estado del arte o análisis de mercado del proyecto	1/10/18	21/10/18	21
- Selección de fuentes de base de imágenes	22/10/18	25/10/18	4
- Estudio sobre manipulación de imagenes con python	22/10/18	28/10/18	7
- Exploración, limpieza y formateo de las imágenes	26/10/18	31/10/18	6
- Estudio sobre Redes neuronales convolucionales	28/10/18	4/11/18	8
- Estudio sobre librerías disponibles a utilizar	28/10/18	4/11/18	8
-- EJECUCION Y EVALUACION --	5/11/18	16/12/18	42
- Construcción del modelo	5/11/18	18/11/18	14
- Entrenamiento del modelo	15/11/18	26/11/18	12
- Evaluación de los resultados	23/11/18	5/12/18	13
- Optimización	1/12/18	11/12/18	11
- Diseñar sistema logístico	4/12/18	14/12/18	11
- Diseñar sistema	6/12/18	14/12/18	9
- Evaluar resultados del sistema	6/12/18	16/12/18	11
- Evaluar los resultados finales	6/12/18	16/12/18	11
PEC3 - Diseño e implementación del trabajo	22/10/18	16/12/18	56
-- REDACCION MEMORIA --	17/12/18	6/01/19	21
- Redacción final del proyecto	17/12/18	2/01/19	17
- Conclusiones	26/12/18	2/01/19	8
- Futuros trabajos	3/01/19	6/01/19	4
PEC4 - Redacción de la memoria	17/12/18	6/01/19	21
PEC5 - Presentación y defensa del proyecto	7/01/19	13/01/19	7
Defensa pública del proyecto	14/01/19	20/01/19	7

Para el desarrollo del proyecto se estiman unas 30 horas semanales durante el proceso que va desde la planificación del proyecto hasta la ejecución y evaluación de éste. Repartiéndolas en 5 horas diarias de lunes a sábado. Dependiendo del punto de desarrollo en la fase de ejecución y evaluación se puede ampliar dicho número de hora semanales.

1.6. Breve descripción de los capítulos de la memoria

El presente trabajo comienza con una introducción al contexto donde se enmarcará el mismo, la detección de objetos en imágenes mediante el uso de técnicas de aprendizaje profundo y redes neuronales convolucionales. A continuación, se justifica el motivo o interés por el cual se ha decidido a realizar este tipo de estudio, para continuar con la descripción de los objetivos principales y secundarios propuestos. Seguidamente se le indica el enfoque y el método se seguirá en el estudio, así como la planificación. Dentro de la propia planificación se describen brevemente los recursos y el tiempo de dedicación previsto para cada una de las fases, finalizando así la parte introductoria del trabajo.

El estado del arte es el primer capítulo tras la introducción, donde se describe en pocas páginas como ha estado marcado el estado del arte en el campo de la detección de objetos en imágenes en los últimos años. A continuación, se detallan las herramientas y métodos más importantes que se utilizan en el estudio. En el capítulo dedicado a las fuentes de datos de imágenes, se indican cuáles serán las utilizadas para el entrenamiento y evaluación del modelo, descrito más adelante. De la misma manera se indica cómo se obtendrán las imágenes de producción donde aplicar el modelo. El modelo para la detección de objetos en imágenes se describe seguidamente, dando detalles de este, así como la justificación para su elección. El propio modelo tiene unas modificaciones, las cuales son descritas en el siguiente apartado. El modelo descrito anteriormente, a su vez, alimenta con datos un sistema de control de existencias que se define a continuación. Esto da lugar a un modelo para el control de existencias por medio de visión artificial.

Después de la presentación y preparación previa, se continua en el siguiente capítulo con la ejecución del modelo para el control existencias. Posteriormente a la ejecución, se muestran y evalúan los resultados obtenidos. Con los resultados obtenidos y su interpretación, se sacan finalmente las conclusiones, así como las propuestas de mejora y el posible trabajo futuro a partir del estudio. Se finaliza el documento con las referencias a la bibliografía relevante, así como los anexos añadidos.

2. Estado del arte

El estado del arte en el campo de la visión artificial y más concretamente en la detección de objetos en imágenes, está marcado desde hace años por los diferentes métodos basados en el aprendizaje profundo o *deep learning.* En la actualidad existen numerosas técnicas conviviendo en este campo de investigación, algunas muy recientes y otras con varios años de existencia, pero aún vigentes. De la misma manera, existen otras investigaciones más estrechamente relacionadas con el presente estudio donde, además de abordar la detección de objetos, se afronta el reto adicional de contar los objetos detectados.

Los problemas más comunes a los que se enfrenta el reconocimiento de objetos en la visión artificial son la clasificación y la localización. La clasificación, junto con la localización, puede ser utilizada para categorizar y reconocer los objetos en la imagen de entre muchas categorías posibles.

El primer método con resultados competitivos para la detección de objetos fue un método para la detección de caras (Viola y Jones, 2001). Su motivación inicial fue la detección de caras, pero el método puede ser entrenado para el reconocimiento de otras clases de objetos. Usa clasificadores en cascada basados en funciones de Haar[1] y la selección de las características más relevantes mediante Adaboost (Freund y Schapire, 1999). Adaboost, *adaptative boosting,* es un tipo de algoritmo clasificador supervisado[2] muy popular basado en el entrenamiento de clasificadores débiles[3] de manera iterada, donde cada nuevo clasificador débil se focaliza en los datos mal clasificados por el anterior. Se obtiene así un resultado mejor y un clasificador fuerte, es decir, un clasificador con una precisión muy buena. Otro método similar para la descripción de características en la detección de objetos, en esta ocasión para la detección de personas, es el de Histograma Orientado a Gradientes (HOG, por sus siglas en inglés) (Dalal y Triggs, 2005) junto a Máquinas de vectores de soporte (SVM, por sus siglas en inglés) para la clasificación.

Los métodos anteriores afrontan el problema desde un punto de vista clásico de aprendizaje automático o *machine learning*, donde las características de las clases son recogidas previamente de forma manual para después realizar el aprendizaje de un modelo predictivo. La introducción del aprendizaje profundo marcó un antes y un después en los métodos de clasificación y detección de objetos en imágenes.

La mayor parte de las técnicas de aprendizaje profundo utilizan redes neuronales, las cuales se denominan redes neuronales profundas[4]. Los modelos de aprendizaje

[1] Secuencia de funciones propuesta por Alfred Haar en 1909.

[2] En los clasificadores supervisados las etiquetas de las instancias en el conjunto de entrenamiento son conocidas.

[3] Clasificadores con un nivel de precisión medio o bajo.

[4] Redes neuronales profundas son aquellas redes neuronales con numerosas capas ocultas, pueden ser decenas o cientos.

profundo se entrenan usando grandes conjuntos de datos clasificados previamente y arquitecturas de redes neuronales profundas que aprenden directamente a partir de dichos datos, sin la necesidad de una extracción manual de las características.

Las redes neuronales convolucionales (CNN, por sus siglas en inglés, o ConvNet) son las redes neuronales profundas más conocidas y utilizadas en los modelos de aprendizaje profundo. Una CNN convoluciona[5] las características que aprende con los datos de entrada y usa capas convolucionales en dos dimensiones, lo cual hace que esta arquitectura resulte apropiada para procesar imágenes.

Existe un etiquetado previo de las imágenes, pero no existe una identificación de las características utilizadas para clasificarlas. Las características relevantes no se entrenan previamente, se aprenden mientras la red se entrena con una colección de imágenes.

La utilización de las CNN en diferentes arquitecturas o modelos para la clasificación y localización visual ha definido el estado del arte en este campo desde el 2012, año en el que AlexNet (Krizhevsky, Sutskever y E. Hinton, 2012) ganó por amplia ventaja el *ImageNet*[6] *Large Scale Visual Recognition Challenge* (ILSVRC). Consiguió una tasa de error entre los cinco primeros resultados o *top-5*[7] del 15.3%, quedando el segundo clasificado en un alejado 26.2%. Esta competición es uno de los referentes y escaparate mundial para demostrar las distintas innovaciones realizadas en clasificación y reconocimiento de imágenes. Desde entonces, todos los ganadores de esta competición han basado sus métodos en redes convolucionales. El ILSVRC ha estado impulsando la mejora del estado del arte en este campo de investigación desde su comienzo en 2010. Otros de los puntos de referencia o *benchmark* más importantes para la clasificación y detección de objetos, junto a el ILSVRC, son el Microsoft Common Objects in Context (COCO)[8] y el PASCAL Visual Object Classes (VOC) Challenge[9].

Las CNN pueden ser entrenadas desde cero recapitulando un conjunto de imágenes etiquetadas lo más amplio posible. La estructura de la red se diseña para que aprenda las características de los datos etiquetados y el modelo. Este punto de partida suele tener un alto costo computacional y el tiempo de entrenamiento dependerá de la cantidad de datos, de la estructura de la red y los recursos de procesador. Se

[5] Se hace referencia a la convolución bidimensional discreta, en donde el valor del píxel de salida se calcula mediante la suma ponderada de píxeles vecinos. En el procesamiento de imágenes esta convolución se efectúa entre una imagen y una matriz llamada *kernel*.

[6] ImageNet es una gran base de datos de imágenes diseñada para la investigación.

[7] La tasa de error top-5 es el porcentaje de ejemplos de prueba para los cuales la clase correcta no estaba en las 5 clases predichas más importantes.

[8] COCO es un conjunto de datos para la detección, segmentación y subtituladción de objetos a gran escala.

[9] PASCAL VOC Challenge es un punto de referencia en el reconocimiento y la detección de categorías de objetos visuales.

recomienda el uso de GPUs[10] en lugar de CPUs[11] para este tipo de trabajos. Existen arquitecturas de cálculo en paralelo como CUDA[12] que nos permiten trabajar con la GPU de las tarjetas gráficas Nvidia[13] en estos casos.

Otro enfoque ampliamente usado en el aprendizaje profundo con CNN es la transferencia del aprendizaje, proceso que implica el ajuste detallado de un modelo previamente entrenado. De las diferentes arquitecturas que existen actualmente en el estado del arte podemos encontrar algunas previamente entrenadas en Keras[14]. Se comienza con la red previamente entrenada ya existente, como por ejemplo VGG19[15] o InceptionResNetV2[16], y se le proporcionan datos nuevos que contienen clases desconocidas en la red previamente entrenada. Se deben realizar algunos ajustes en la red para que sea posible realizar una tarea nueva, por ejemplo, categorizar solo los elefantes en lugar de 500 objetos distintos. Esto también tiene la ventaja de necesitar muchos menos datos y un menor tiempo de cálculo.

AlexNet abrió una nueva era, introdujo una arquitectura de red profunda con un conjunto de capas convolucionales y de agrupación o *pooling,* más unas capas completamente conectadas. Se tomaron a su vez, conceptos como la unidad lineal rectificada (ReLU, por sus siglas en inglés) para la función de activación o el uso de capas de abandono o *dropout* para reducir la sobreespecialización o *overfitting* del modelo. Desde la demostración de la efectividad del uso de las CNN, se han ido sucediendo los avances e innovaciones en el campo. OverFeat (Sermanet *et al.*, 2013) presenta un entorno integrado para la clasificación, localización y detección usando CNN. La parte de la extracción de características está basada en *AlexNet* y para la localización y la detección introducen un método que se basa en la acumulación y predicción de cuadros delimitadores o *bounding boxes*. La arquitectura *Region-based ConvNet* (R-CNN) (Girshick, Donahue, Darel y Malik, 2014) genera las regiones propuestas o posibles, unas 2000, de una imagen mediante búsqueda selectiva. Estas regiones son recortadas y deformadas en una imagen de 227x227 pixeles. Se extraen entonces 4096 características, mediante el modelo *AlexNet,* de cada imagen y se clasifican mediante un modelo SVM. Por último, se realiza una regresión lineal para restringir el borde del objeto si es que existe. De esta manera, el problema de

---

[10] Unidad de procesamiento gráfico.

[11] Unidad central de procesamiento.

[12] *Compute Unified Architecture* (CUDA) es una plataforma de computación en paralelo creada por nVidia para codificar algoritmos en GPUs de nVidia.

[13] Empresa multinacional especializada en el desarrollo de GPUs.

[14] Librería *open source* escrita en Python y diseñada para la rápida ejecución de redes neuronales profundas.

[15] VGG-19 es una CNN entrenada con más de un millón de imágenes de la base de datos ImageNet.

La red tiene una profundidad de 19 capas y puede clasificar imágenes entre 1000 clases de objetos.

[16] InceptionResNetv2 es una CNN entrenada con más de un millón de imágenes de la base de datos ImageNet. La red tiene una profundidad de 164 capas y puede clasificar imágenes entre 1000 clases de objetos.

localización y detección se resuelve realmente mediante clasificación. Este método proporcionó el camino y el enfoque dominante para la mayoría de las modelos actuales de detección de objetos.

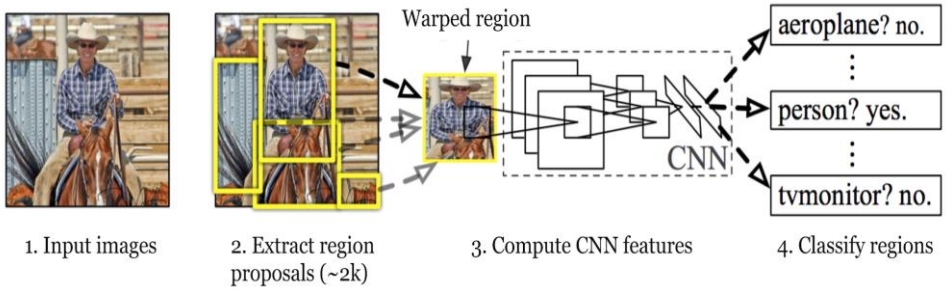


Figura 2.1. La arquitectura R-CNN. (Girshick *et al.,* 2014)

ZFNET (Zeiler, y Fergus, 2014) es el método ganador de ILSVRC 2013, que es básicamente AlexNet con una pequeña modificación: usa un núcleo 7x7 en vez de uno de 11x11 en la primera capa Convolucional para recabar más información. La arquitectura de CNN VGGNet (Simonyan, Zisserman, 2014) sigue siendo una de las más comunes hoy en día por su simplicidad y eficacia. Se trata de una CNN de 19 capas con filtros convolucionales de 3x3 y con un paso o *stride*[17] y un relleno o *padding*[18] de 1 píxel, junto con 2x2 capas de agrupación máxima o *max pooling* con un *stride* de 2 pixeles.

La arquitectura CNN Inception v1 (GoogleNet) (Szegedy *et al.,* 2015) contiene 22 capas y un nuevo concepto, los módulos *inception*. Fue la ganadora del ILSVRC2014 con una tasa de error entre los 5 primeros resultados del 6,7%. Esta no es una arquitectura secuencial como se había visto hasta la fecha, existen partes de la red, los módulos, que trabajan en paralelo. *Fast* R-CNN (Girshick, 2015) es, como su nombre indica, más rápido que R-CNN. Realiza la extracción de las características de la imagen antes de proponer regiones, por lo que la CNN se ejecuta sobre toda la imagen en lugar de sobre 2000 regiones. Además, reemplaza el modelo SVM por una capa softmax[19], con lo que se extiende la red para realizar pronósticos en lugar de la creación de un modelo nuevo. Aun siendo mejor que R-CNN, el lento algoritmo de

[17] Es una medida, en pixeles, para regular el movimiento del filtro o kernel de la capa convolucional a través del espacio de la imagen.

[18] Añade una medida, en pixeles, extra fuera de la imagen. Zero padding quiere decir que todos los valores de los pixeles añadidos son cero.

[19] La función *softmax* es utilizada como capa final de los clasificadores basados en redes neuronales. Asigna probabilidades decimales a cada clase en un caso de clases múltiples. Al ser probabilidades, deben sumar 1.0.

búsqueda selectiva para las regiones propuestas deja un campo de mejora a este método que llegaría con el método *Faster* R-CNN.

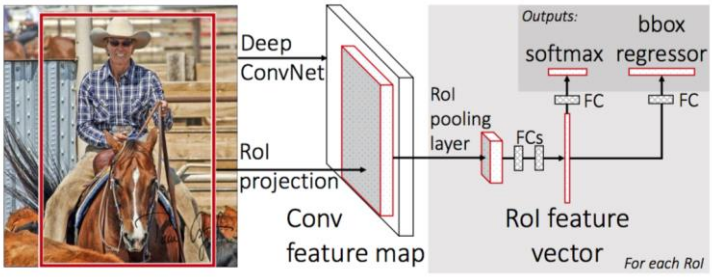


Figura 2.2. La arquitectura *Fast* R-CNN. (Girshick, 2015)

El método *Faster* R-CNN (Ren *et al.*, 2015) introdujo la red de regiones propuestas (RPN, por sus siglas en inglés), inspirado en la propuesta *MultiBox* (Szegedy, Reed, Erhan, Anguelov y Ioffe, 2014), donde una ventana deslizante de 3x3 se mueve a través del mapa de características para asignar un tamaño más pequeño. Para cada una de las posiciones de la ventana deslizante se generan múltiples regiones propuestas basadas en cajas de anclaje o *anchor boxes.* Cada una de esas regiones propuestas constan de un puntaje o *score* para la presencia del objeto en esa región y cuatro coordenadas que delimitan el cuadro de la región. Las regiones propuestas se presentan directamente al modelo *Fast* R-CNN, al que se añade una capa de agrupación o *RoI pooling layer*, unas capas completamente conectadas y una última capa para la clasificación *softmax* y un regresor de cuadros delimitadores. Este método consigue así un gran rendimiento y precisión.

La arquitectura ResNet (He, Zhang, Ren y Sun, 2016) fue introducida por Microsoft y consiguió la primera posición en la tarea de clasificación del ILSVRC2015 con una espectacular tasa de error de 3.6%. Este hecho se considera un hito ya que se había superado la tasa de error de un humano, que se estima que está entre el 5 y el 10% de error. Resnet resuelve, mediante el uso de bloques residuales, los problemas de degradación por saturación o del gradiente de fuga, donde las fluctuaciones se vuelven demasiado pequeñas para ser útiles, de las redes profundas con muchas capas. Estos bloques residuales aprovechan el mapeo residual para preservar las entradas. De esta forma, esta arquitectura introduce el concepto nuevo de *residual learning*.

DenseNet (Huang, Liu, Maaten, Weinberger, 2018) es la versión actual de este tipo de arquitectura donde se consigue alimentar una capa con todas las salidas de las capas anteriores, consiguiendo así una gran propagación de las características por la red.

You Only Look Once (YOLO) (Redmon, Divvala, Girshick, Farhai, 2016) se basa en el desarrollo del concepto *multibox*, donde el problema de la detección del objeto, en

tiempo real, en la imagen o fotograma del video se toma como un problema de regresión. Las características son extraídas de toda la imagen a la vez. Una única red convolucional predice simultáneamente múltiples cuadros delimitadores que enmarcan los objetos en la imagen y predice probabilidades condicionales por cada clase. Es muy rápida, pudiendo llegar a los 45 fotogramas por segundo en ordenadores convencionales.

Single Shot MultiBox Detector (SSD) (Wei *et al.,* 2016) es uno de los mejores métodos basados en regiones, aprovecha el RPN del *Faster* R-CNN, usándolo para clasificar directamente el objeto dentro de cada cuadro anterior en lugar de solamente anotar la confianza del objeto, similar como lo realiza YOLO. La arquitectura está formada principalmente por capas convolucionales y no tiene capas totalmente conectadas.

La arquitectura *Region-based Fully Convolutional Networks* (R-FCN) (Dai *et al.,* 2016) es una variante de *Faster* R-CNN, la cual consigue similares niveles de precisión a mayor velocidad. YOLO9000 (Redmon, Farhadi, 2017) es la mejora de YOLO capaz de detectar más de 9000 clases, de ahí su nombre. Existe otra mejora de YOLO en la actualidad denominada YOLOv3 (Redmon, Farhadi, 2018). *Mask* R-CNN (He *et al.,* 2017) es una de las últimas versiones de las arquitecturas basadas en regiones. Es el resultado de una serie de mejoras sobre *Faster* R-CNN. Esta arquitectura trabaja primero sobre las regiones propuestas donde es posible que esté el objeto. Posteriormente clasifica las regiones propuestas y genera recuadros delimitados y máscaras. Consigue identificar los contornos de los objetos al nivel de un píxel. A este nivel de detalle en la detección se le denomina segmentación de una instancia.

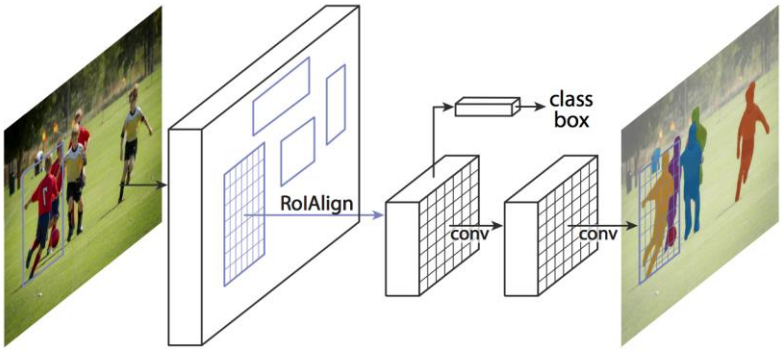


Figura 2.3. *Mask* R-CNN y la segmentación de instancias. (He *et al*., 2017)

La arquitectura *Inception* v3 (Szegedy, Vanhoucke, Ioffe, Shens y Wojna, 2016) reduce el cuello de botella representativo o *bottleneck*[20] y mejora los métodos de factorización en las capas convolucionales. *Inception* v4 (Szegedy, Ioffe, Vanhoucke y Alemi, 2016)

[20] Reducir demasiado las dimensiones puede causar pérdida de información, a este concepto se le conoce como cuello de botella representativo o *bottleneck.*

realiza una versión más uniforme y sencilla de los modulos de la arquitectura *Inception* v3 modificando el tronco o stem de la red. También reduce los bloques de la red.

Inception-RestNet (Szegedy, Ioffe, Vanhoucke y Alemi, 2016) es un hibrido entre las arquitecturas inception y la arquitectura ResNet. Añade conexiones residuales que agregan la salida de la operación de convolución del módulo de inicio, a la entrada.

Xception (Chollet, F., 2017) es una extensión de la arquitectura *inception* donde los módulos *inception* son sustituidos por convoluciones separables en profundidad o *depthwise separable convolutions.*

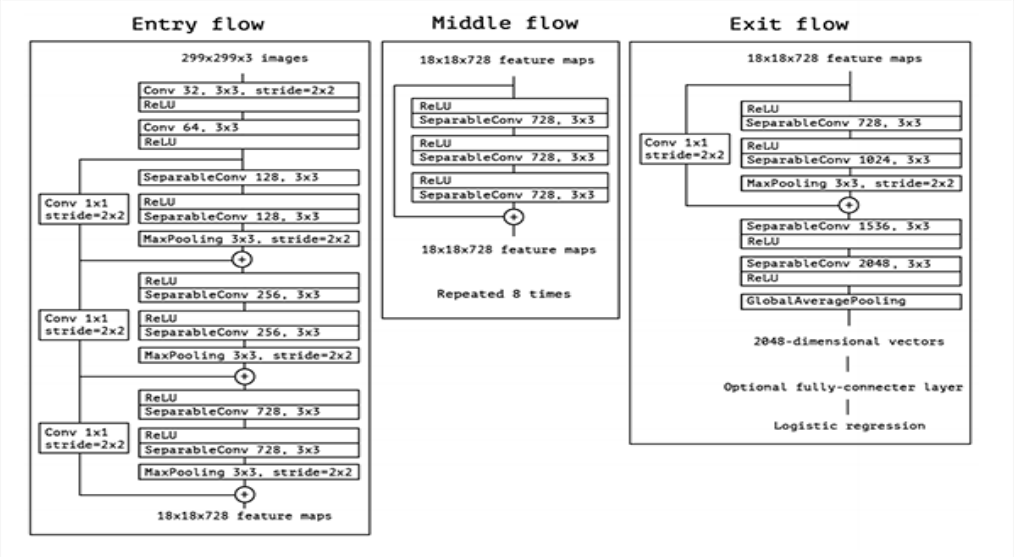


Figura 2.4. Arquitectura Xception (Chollet, 2017)

MobileNets (Howard *et al.,* 2017) propuesta por Google, es un tipo de arquitectura focalizada a su uso por dispositivos móviles o sistemas embebidos con falta de potencia computacional. Usa convoluciones separables en profundidad, como Xception. En realidad, MobileNets es, esencialmente, una versión más simple de la arquitectura Xception y optimizada para el uso de aplicaciones en dispositivos móviles. Estos últimos tipos de arquitecturas CNN, en sus diferentes versiones, son los métodos más rápidos y ligeros en la actualidad en el campo de la clasificación y detección de objetos en imágenes.

La mayoría de los trabajos que han abordado el problema del recuento de objetos en imágenes han realizado previamente la clasificación y localización de dichos objetos, para realizar posteriormente el recuento. Esta aproximación no es la única vía en la que se ha abordado el problema del conteo.

El trabajo de Lempitsky y Zisserman (2010) introdujo el concepto de aprender a contar objetos mediante el mapa de densidad de una imagen, evitando así el tener que detectarlos o localizarlos. En (Zhang, J. *et al.,*2016) se inspiran en el fenómeno de la subitización o *subitizing*, esto es, la capacidad del ser humano para realizar

evaluaciones rápidas de recuento en un entorno con un pequeño número de elementos. Igualmente lo hace (Chattopadhyay *et al.*,2016), para ello realiza una estrategia de divide y vencerás, dividiendo la imagen en regiones, donde la tarea resulta más sencilla, para seguidamente sumar los resultados de todas las regiones. A su vez, investigan cómo puede este conteo mejorar la detección de los propios objetos. Rahnemoonfar y Sheppard, (2017) utilizan una versión modificada de la red entrenada *Inception*-ResNet para la estimación del número de piezas de fruta en la imagen. En diferentes entornos consiguen una precisión del 91% con imágenes reales y un 93% con imágenes no reales.

Más próximos al enfoque que se da en este trabajo es el de Chen *et al.* (2017), donde parten de una R-FCN realizando una transferencia de aprendizaje y un ajuste o *fine-tuning* de la red previamente entrenada VGG-16, para realizar el proceso de recuento. El número de clases en este caso son dos, naranjas y manzanas, en lugar de las 21 clases de la red entrenada. Añaden al trabajo imágenes etiquetadas y tomadas por ellos mismos para el entrenamiento y ajuste de la CNN. Consiguen una buena tasa de error del 13.8 % con las naranjas y un 10.4% con las manzanas. (Sa *et al.,* 2016) afrontan el problema mediante *Faster* R-CNN y transferencia de aprendizaje combinando para el entrenamiento imágenes en color (RGB) e imágenes cercanas al infrarrojo (NIR) de siete clases de fruta. (Bargoti y Underwood, 2016) utilizan también *Faster* R-CNN para su estudio además de *data augmentation*[21] para la reducción del número de imágenes originales de entrenamiento. Consiguen una precisión de más de un 90% en la detección de mangos y manzanas en imágenes de huertos.

La clasificación, localización y detección de objetos en imágenes y videos ha avanzado mucho en pocos años y se han conseguido eficientes clasificadores y detectores, sobre todo en entornos controlados, pero este campo de investigación tiene aún mucho camino por recorrer. Las aplicaciones prácticas, presentes y futuras, de los logros que se consigan en este campo son innumerables.

[21] Aumento de datos de entrada, mediante diferentes técnicas, modificando los datos existentes.

3. Materiales y métodos

3.1. Anaconda

Anaconda es el nombre que recibe el entorno integrado de desarrollo (IDE, por sus siglas en inglés) elegido para el desarrollo del estudio. Anaconda es una distribución gratuita y de código abierto de los lenguajes de programación Python y R para aplicaciones de ciencia de datos y aprendizaje automático, procesamiento de datos a gran escala, análisis predictivo, computación científica, cuyo objetivo es simplificar la administración y el despliegue de paquetes.

3.2. Python

El presente trabajo utiliza el lenguaje de programación Python. Python es un lenguaje de programación interpretado de alto nivel para programación de propósito general. Creado por Guido van Rossum, Python tiene una filosofía de diseño que enfatiza la legibilidad del código, especialmente al usar espacios en blanco significativos. Proporciona construcciones que permiten una programación clara tanto a pequeña como a gran escala.

3.3. Tensorflow

TensorFlow (Abadi et al., 2015) es una libreria *open source* desarrollada por Google que nos permite realizar cálculos numéricos usando diagramas de flujo de datos. A continuación, se describen brevemente los principales conceptos relacionados con Tensorflow. Un grafo de TensorFlow es una descripción de cálculos. Para calcular cualquier cosa dentro de TensorFlow, el grafo debe ser lanzado dentro de una sesión.
La sesión encapsula el ambiente en el que las operaciones, que definimos en el grafo, van a ser ejecutadas y los tensores son evaluados. De la misma forma, la Sesión coloca las operaciones del grafo en los diferentes dispositivos, tales como CPU o GPU, y proporciona métodos para ejecutarlas. Los nodos del grafo, llamados ops por la abreviatura de operaciones, representan operaciones matemáticas, pero también pueden representar los puntos para alimentarse de datos, devolver resultados, o también para leer y escribir variables persistentes. Una op o nodo tiene cero o más tensores, realiza algún cálculo, y produce cero o más tensores. Los arcos o aristas describen las relaciones de entrada y salida entre los diferentes ops o nodos. Estos arcos están representados por los arreglos de datos multidimensionales o tensores. El flujo de los tensores a través del grafo es de donde TensorFlow recibe su nombre. Las variables persistentes mantienen el estado a través de las ejecuciones del grafo. Son *buffers* en memoria que contienen tensores. Se deben inicializar explícitamente y se pueden guardar en el disco para luego restaurar su estado si es necesario. Por otro lado, también existen las llamadas variables simbólicas, contenedores o *placeholders.* Estos nos permiten alimentar las operaciones con los datos durante la ejecución del grafo. Estos contenedores deben ser alimentados con los datos antes de ser evaluados en la sesión, si no, se produce un error.

Por defecto Tensorflow trabaja con la CPU, pero es posible y altamente recomendable trabajar también con la GPU si es posible, incrementando considerablemente la potencia de computo. Según Nvidia de 2 a 10 veces más. Esto se consigue mediante la instalación, si la tarjeta gráfica es compatible, de CUDA y cuDNN.

La arquitectura unificada de dispositivos de cómputo (CUDA, por sus siglas en inglés) es una arquitectura de cálculo paralelo de Nvidia que aprovecha la potencia de la GPU para proporcionar un incremento de la potencia computacional. Esto es posible mediante el coprocesamiento compartido entre la CPU y la GPU. Para la utilización de la GPU en Tensorflow, se comprueba que la tarjeta gráfica es compatible con CUDA en la página de Nvidia[22]. En el caso de que sea compatible se procede con la descarga e instalación local de Cuda *Toolkit* v9.0.[23]. Se ha instalado también cuDNN v7.0. CuDNN es una librería para redes neuronales profundas construidas con CUDA. Proporciona funcionalidad acelerada de GPU para operaciones comunes en redes neuronales profundas. La manera de referenciar la CPU o GPUS en TensorFlow es la siguiente; “/cpu:0” para la CPU del servidor, “/gpu:0” para la GPU del servidor si solo hay una, “/gpu:1” para la segunda GPU del servidor, y así sucesivamente. El equipo con el que se realiza el trabajo posee una tarjeta compatible, la tarjeta gráfica NVIDIA GeForce GTX 1060 3GB, por lo que instalaremos CUDA y cuDNN. De esta manera se tiene la mayor potencia de computo posible.

3.3.1. API para la detección de objetos de Tensorflow

La interfaz de programación de aplicaciones (API, por sus siglas en inglés) de detección de objetos en imágenes de TensorFlow, es un marco de trabajo de código abierto creado sobre TensorFlow que facilita la construcción, entrenamiento y despliegue de modelos para la detección de objetos en imágenes. Proporciona a su vez una colección de modelos previamente entrenados haciendo uso de la base de datos de imágenes *COCO, Kitti y Open images*. A esta colección de modelos previamente entrenados en la API de Tensorflow de le denomina *detection model zoo*.Cada uno de los modelos posee un fichero de configuración donde se pueden cambiar parámetros e hiperparámetros del modelo para el proceso de entrenamiento o evaluación. Existen además ficheros de configuración en la carpeta protos/ de la API, los cuales se pueden modificar para ajustar a unas necesidades específicas del entrenamiento o evaluación del modelo. Por ejemplo, es posible ampliar el número de puntos de control o *checkpoints* durante el proceso de entrenamiento.

[22] https://developer.nvidia.com/cuda-gpus

[23] https://developer.nvidia.com/cuda-downloads

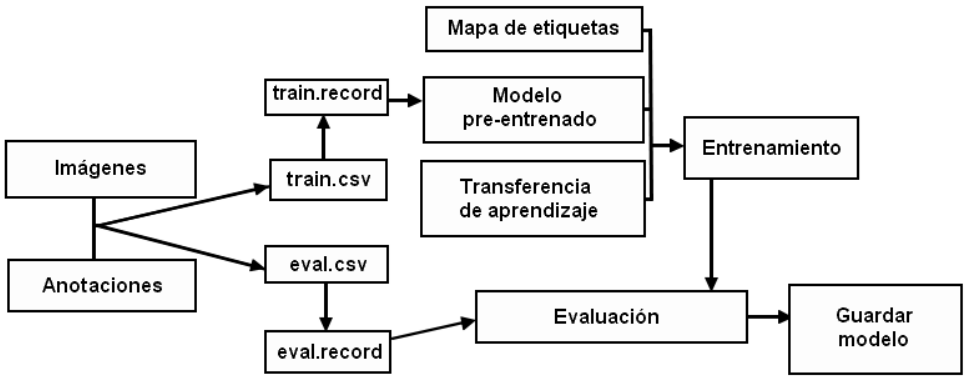


Figura 5. Diagrama de funcionamiento de la API
para la detección de objetos de tensorflow

Una estructura estándar para organizar la información dentro de la API, específicamente en la carpeta *object_detection*/ es la siguiente:

- datos/: Directorio donde se guardan los ficheros csv generados.
- Img/train/: Directorio que contiene las imágenes de entrenamiento.
- Img/eval/: Directorio que contiene las imágenes de evaluación.
- training/: Directorio donde se guardan los puntos *checkpoints* del entrenamiento.
- eval/: Directorio donde se guardan los resultados de las evaluaciones.

Podemos utilizar la API para la detección de objetos de tensorflow directamente. Simplemente se elige un modelo pre-entrenado y se configuran varias líneas de código en uno de sus ejemplos para que apunte al modelo elegido. De esta manera se detectarán los objetos que estén previamente etiquetados en la base de datos con la que haya sido entrenado el modelo. Como se comentó anteriormente, la API de Tensorflow para la detección de objetos ha entrenado sus modelos con las bases de imágenes *COCO, Kitti y Open images*. Las clases de objetos que no se encuentren etiquetadas en dichas bases no existirán para el modelo pre-entrenado y, por lo tanto, no las detectará. Para utilizar la API con nuestras propias imágenes, las cuales pueden contener objetos que el modelo conozca o no, debemos realizar una serie de pasos y cumplir unos requisitos que se explican brevemente a continuación.

Las imágenes deben estar codificadas como jpg o png y además debe haber imágenes para el entrenamiento y para la evaluación del modelo en carpetas separadas. Se debe crear una lista de las anotaciones de los cuadros delimitadores o *bounding boxes*, explicados más adelante. Estos cuadros delimitadores deben tener asociada la clase del objeto al cual enmarca. Se debe generar también un fichero denominado mapa de etiquetas o *labelmap* donde estarán las clases a detectar. A partir de las imágenes y sus anotaciones se crean, primero los ficheros con extensión csv para el conjunto de entrenamiento y evaluación para a continuación generar los ficheros con formato *tfrecord*, los cuales serán las entradas para el nuevo entrenamiento del modelo. El previo entrenamiento del modelo sirve como base para el nuevo entrenamiento con las nuevas imágenes, para lo que se utiliza transferencia

de aprendizaje o *transfer learning*. Se entrena el modelo con los nuevos datos, se evalua y, si es satisfactorio el resultado obtenido, se guarda para inferenciar el nuevo modelo creado sobre nuevos datos.

3.3.2. TensorBoard

TensorBoard es un conjunto de aplicaciones web para inspeccionar, visualizar y comprender las ejecuciones y gráficos de TensorFlow. Puede trazar métricas como la pérdida y la precisión durante una ejecución de entrenamiento, mostrar las visualizaciones del histograma de cómo un tensor está cambiando con el tiempo y mostrar datos adicionales como imágenes. TensorBoard funciona leyendo los archivos de TensorFlow que contienen la información sobre el proceso de entrenamiento como son los *checkpoint* que se generan mientras se entrena el modelo.

Se ejecuta en un navegador y por defecto tendrá el nombre del equipo o *localhost* y número de puerto 6006. En el estudio se utiliza para la evaluación de los modelos de detección de clases de frutas en las imágenes de producción. Se puede usar para comparar ejecuciones de entrenamiento y evaluación, así como recopilar estadísticas de tiempo de ejecución y generar histogramas.

3.4. Redes neuronales

El presente estudio utiliza un modelo para la detección de objetos basado en un tipo de redes neuronales, las redes neuronales convolucionales. Pero antes de hablar de ellas se comenzará repasando brevemente los conceptos más importantes de las redes neuronales en general. El modelo de neurona artificial está basado en el funcionamiento de una neurona bilógica. Las entradas de datos de una neurona artificial pueden ser la salida de otra neurona en la red. Cada una de esas entradas tienen asociado un peso o *weight.* Un elemento importante es el sesgo o *bias*, que también tiene asociado un peso, representa a un valor constante que se le suma a la de la función de activación de la neurona y usualmente tiene el valor 1. Permite cambiar la función de activación a derecha o izquierda, dando flexibilidad aprendizaje a la neurona. En la figura 3.1 se muestra el esquema general de una neurona artificial.

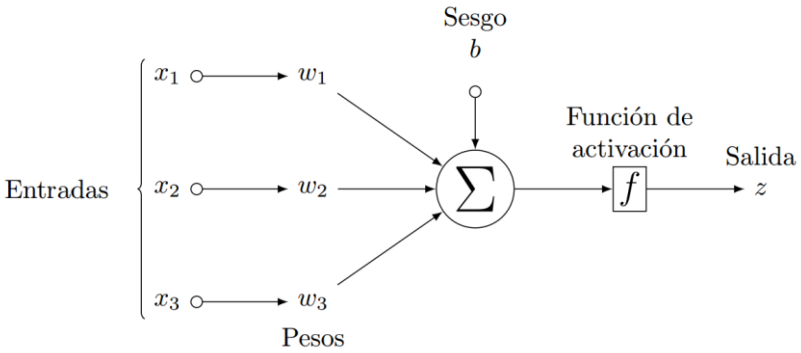


Figura 3.1. Esquema de una neurona artificial

La salida z de una neurona artificial donde sus entradas proceden de un vector x = [X1, … , Xn] se representa en la siguiente ecuación:

$$z = f_{act}\left(\sum_{i=1}^{N} w_i x_i + b\right)$$

Una red neuronal consiste en una serie de capas de neuronas. En una red neuronal totalmente conectada o *dense*, todas las neuronas de una capa se conectan a las neuronas de la siguiente capa, pero existen otros tipos de redes neuronales.

Los cálculos en una red neuronal comienzan en la capa de entrada, esta pasa valores a la capa oculta, se denomina capa oculta al conjunto de capas intermedias en una red neuronal. Si el número de capas intermedias es mayor a uno, a la red se la considera profunda. Desde la capa oculta se transmite la salida a la última capa que contendrá el valor final. Algo importante a tener en cuenta es el hecho de que cada una de las neuronas de entrada no envía múltiples valores de salida a la capa oculta, en realidad solamente hay un valor de salida por neurona. Las neuronas siempre producen un valor, independientemente de cuántas conexiones de salida tengan. En la figura 3.2 se muestra la estructura de una red neuronal con una capa oculta.

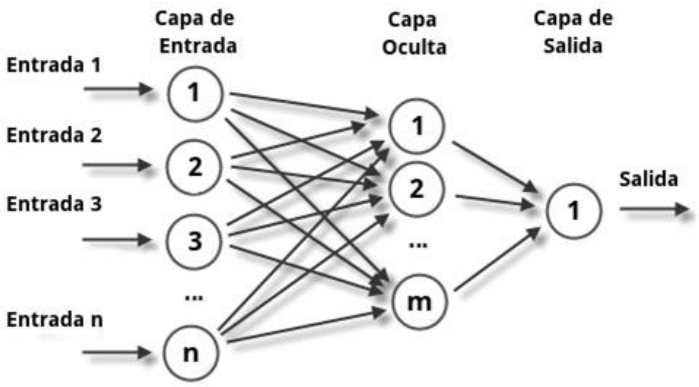


Figura 3.2. Red neuronal con capa oculta [24]

Las redes neuronales realizan un proceso donde se van actualizando los pesos a medida que la red vaya aprendiendo con el proceso de entrenamiento. Este proceso se denomina propagación hacia atrás o *backpropagation*. La propagación hacia atrás es un algoritmo que funciona mediante la determinación de la pérdida o error en la salida y luego propagándolo de nuevo hacia atrás en la red neuronal. De esta forma los pesos se van modificando y actualizando para minimizar el error resultante de cada neurona. Este algoritmo es lo que les permite a las redes neuronales aprender, de

[24] https://es.wikipedia.org/wiki/Deep_Dream#/media/File:RedNeuronalArtificial.png

hecho, el problema de aprendizaje en las redes neuronales se formula en términos de la minimización de la función de error o pérdida asociada. La función de coste o pérdida es uno de los parámetros para cuantificar lo cercano que está una determinada red neuronal de su ideal, el valor mínimo de la curva, mientras está en el proceso de entrenamiento. Generalmente si la función de coste decrece quiere decir que el entrenamiento se aproxima a su valor mínimo objetivo global. Esto no es necesariamente así en algunos casos, por ejemplo, al caer en un mínimo local y quedarse atascado. Otro caso típico es donde la función de coste sigue descendiendo con la muestra de datos de entrenamiento mientras que con la muestra de datos de evaluación no es así, esto indicaría una sobre especialización u *overfitting*. Normalmente las funciones de pérdida o coste se calculan haciendo una media de los
Las pérdidas de todos los ejemplos de entrenamiento tal y como se muestra en la siguiente ecuación:

$$C(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} C_i(\boldsymbol{W}, \boldsymbol{b})$$

Donde W y b hacen referencia a los conjuntos totales de pesos y sesgos de la red respectivamente. Ci representa la función de coste para la instancia i del conjunto de entrenamiento.

En la mayoría de los problemas reales, las variables de entrada tienden a ser altamente interdependientes y afectan a la salida de forma compleja. Las neuronas de las capas ocultas nos permiten capturar interacciones sutiles de nuestras entradas. Las capas ocultas representan atributos de nuestros datos y cada una de las neuronas de la capa oculta sopesa sus entradas de forma diferente, de esta manera aprende características diferentes de los datos. La capa de salida logra capturar estas características intermedias y no solamente las entradas originales. Con más de una capa oculta, la red neuronal pueda aprender sobre varios niveles de abstracción de los datos. Por lo general las primeras capas aprenden características más generales de los datos y las últimas capas se concentran en características más específicas.

Un modelo basado en una red neuronal puede tener parámetros, los cuales son variables de configuración que son internas al propio modelo y cuyo valor puede estimarse a partir de los datos de los que dispongamos. Los hiperparámetros, por el contrario, son variables de configuración que son externas al modelo en sí mismo y cuyo valor en general no se puede estimar a partir de los datos. Los hiperparámetros son definidos por el desarrollador con el objetivo de ajustar los algoritmos de aprendizaje. Existen hiperparámetros tanto a nivel de estructura y topología de la red neuronal (número de capas, número de neuronas, sus funciones de activación, etc) como hiperparámetros a nivel de algoritmo de aprendizaje (*learning rate, momentum, epochs, batch size*, etc).

3.4.1. Funciones de activación

Las funciones de activación en una neurona definen la salida dada una entrada o conjunto de entradas. Cumplen con el objetivo de limitar el rango de salida de la neurona y pueden ser lineales o no lineales.

La función de activación sigmoide tiende a dar valores próximos al 0 o al 1. Siendo la 'z' grande y positiva, la 'y' resulta 1. Siendo la 'z' grande y negativa, la 'y' resulta 0. La razón principal por la que usamos la función sigmoide es porque existe entre 0 y 1. Por lo tanto, se utiliza especialmente para modelos en los que tenemos que predecir la probabilidad como una salida. La función es diferenciable. Eso significa que podemos encontrar la pendiente de la curva sigmoidea en cualquiera de los dos puntos. La función sigmoidea logística puede hacer que una red neuronal se atasque en el momento del entrenamiento.

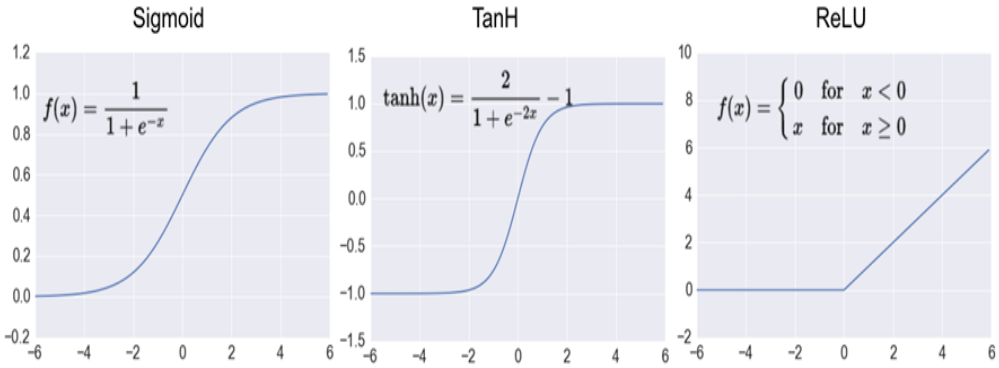


Figura 3.3. Funciones de activación Sigmoide, Tanh y ReLU [25]

La función de activación Tanh es como la función sigmoide, pero con más amplitud. El rango de la función tanh está entre -1 y 1. Tanh también es sigmoidal, es decir, su gráfica tiene forma de s. TanH tiende a converger rápido durante *backpropagation* si los pesos se han inicializado con valores pequeños y homogéneamente distribuidos alrededor de 0.

La mayoría de las redes neuronales actuales usan la función de activación llamada *rectified linear unit* o ReLU. Una de las grandes ventajas de ReLU es que es muy sencilla de calcular, por lo que es muy rápida. En ReLU f (x) es cero cuando x es menor que cero y f (x) es igual a x cuando x es superior o igual a cero. Pero el problema de la función de activación ReLU es que todos los valores negativos se vuelven cero inmediatamente, lo que disminuye la capacidad del modelo para ajustarse o entrenarse con los datos correctamente. Eso significa que cualquier entrada negativa dada a la función de activación ReLU convierte el valor en cero

[25] https://medium.com/@prasadpal107/dictionary-for-cnn-753a1a39db45

inmediatamente en el gráfico, lo que a su vez afecta al gráfico resultante al no asignar los valores negativos de manera adecuada. En la figura 3.4. podemos ver la representación gráfica de las tres funciones de activación mencionadas.

La función de activación logística *Softmax* es la más generalizada y utilizada para clasificación multiclase. *Softmax* nos permite tener una capa de salida con diferentes probabilidades en cada nodo que en total sumen 1, es decir, el 100%. Por ejemplo, en el caso de que se tenga imágenes con una clase de fruta en ellas tendríamos 6 salidas con una probabilidad para cada una, si tenemos una naranja que se parece a un limón podríamos tener probabilidades muy bajas para todas las imágenes que contengan las diferentes clases de frutas excepto para las imágenes que contengan una naranja o un limón, en las cuales deberíamos tener probabilidades superiores al 50%.

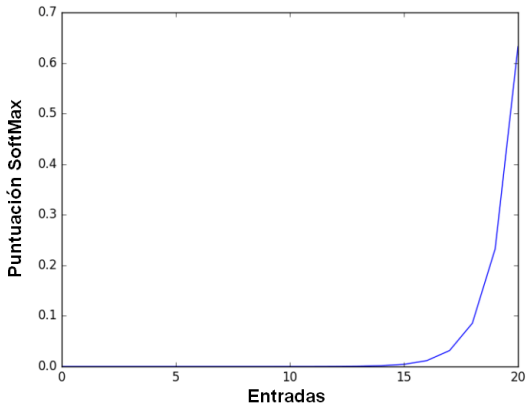


Figura 3.4. Gráfica de la función de activación *Softmax*

La gráfica muestra claramente la propiedad fundamental de la función *softmax*: El mayor valor tendrá la mayor probabilidad. La función *softmax* es generalmente utilizada como capa final de los clasificadores basados en redes neuronales.

3.4.2. Aprendizaje

El procedimiento utilizado para llevar a cabo el proceso de aprendizaje en una red neuronal se denomina entrenamiento. Como se comentaba anteriormente, existen hiperparámetros que controlan el funcionamiento del algoritmo de aprendizaje. Una época o *epoch* hace referencia al conjunto de datos entero que pasa hacia adelante y hacia atrás a través de la red neuronal solamente una vez. Lo que nos indica es el número de veces en las que todos los datos de entrenamiento han pasado por la red neuronal en el proceso de entrenamiento. Muy frecuentemente es inviable pasar todo el conjunto de datos de una vez, una época es demasiado grande para alimentar a la computadora. Por ello dividimos el conjunto de datos en

varios lotes más pequeños o *batches*. Los *batches,* por tanto, son el número de lotes en el que dividimos el conjunto de datos. Por otro lado, *batch size* hace referencia a cuantos datos, por ejemplo, imágenes, pasamos en una época. *Batch size* es número total de ejemplos de entrenamiento presentes en un solo lote.

El descenso de gradiente es un algoritmo muy extendido en aprendizaje de máquina, así como en aprendizaje profundo.

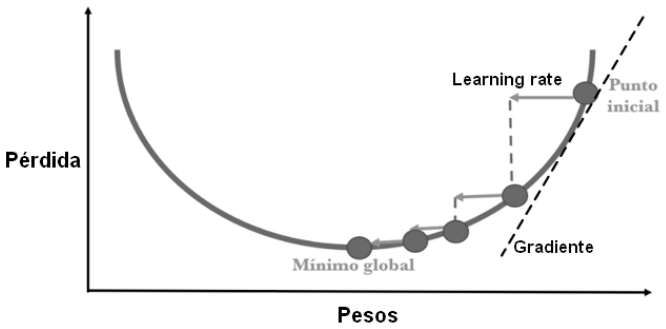


Figura 3.5. Descenso de gradiente

Una variante es el denominado descenso de gradiente estocástico o (SGD, por sus siglas en inglés). En este algoritmo se estima el gradiente y se ajustan los valores de los parámetros a partir de la pérdida observada para cada ejemplo o para cada muestra pequeña de ejemplos también denominada mini lote o *mini batch*. Si los datos están bien distribuidos, un pequeño subconjunto de ellos nos debería dar una idea bastante buena del gradiente y por tanto del próximo paso a dar en el proceso de aprendizaje. Es posible no se obtenga la mejor estimación, pero es más rápido y debido a que es un proceso iterativo, esta aproximación se ajusta a las necesidades.

Un concepto muy importante en el proceso de entrenamiento es la tasa de aprendizaje o *learning rate*. El valor adecuado de este hiperparámetro es muy dependiente del problema en cuestión. Generalmente, si este es demasiado grande, se darán pasos o saltos enormes, de un punto al siguiente, que podrían ser buenos para ir más rápido en el proceso de aprendizaje, pero es posible que se salte el valor mínimo objetivo y, de esta manera, dificultar que el proceso de aprendizaje converja porque al buscar el siguiente punto perpetuamente rebota al azar en el fondo de la curva, es decir, alrededor el valor mínimo sin llegar nunca a alcanzarlo. Si la tasa de aprendizaje es pequeña, se harán avances constantes y pequeños, teniéndose una mejor oportunidad de llegar a un mínimo local, esto puede provocar que el proceso de aprendizaje sea lento.

Una buena práctica es, si nuestro modelo de aprendizaje no funciona todo lo bien que esperamos, disminuir el hiperparámetro tasa de aprendizaje. Normalmente la mejor tasa de aprendizaje es la que disminuye a medida que el modelo se acerca a la solución. Para conseguir esto tenemos otro hiperparámetro llamado *learning rate*

decay. Se utiliza para disminuir la tasa de aprendizaje a medida que van pasando las épocas para permitir que el aprendizaje avance más rápido al principio con tasas de aprendizaje más grandes. A medida que se va avanzando, se van haciendo ajustes en la tasa de aprendizaje para que sea cada vez más pequeña.

Cuando se habla de sobreajuste u *overfitting* se hace referencia al sobre entrenamiento de un modelo sobre un conjunto de datos, especializándose en ellos. En estos casos el modelo es muy bueno solamente para los casos de entrenamiento, pero no lo es para el resto de los casos que se puedan dar, siendo inviable inferir el modelo.

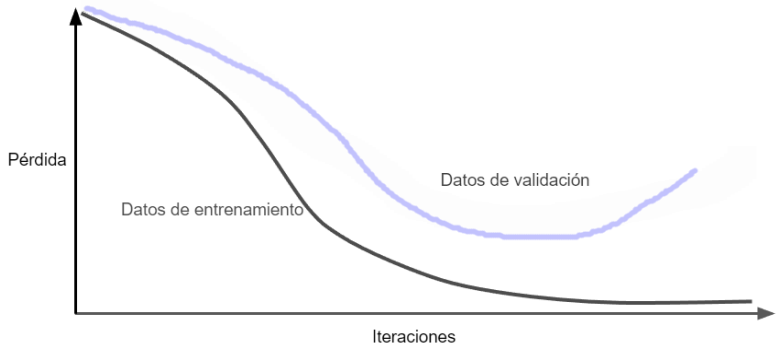


Figura 3.6. Curva de pérdida de entrenamiento y validación

En la práctica se suelen encontrar mínimos locales además de un mínimo objetivo global, que es normalmente el que nos interesa alcanzar. En estos mínimos locales el optimizador puede verse atascado y creerse que ha llegado al mínimo global. Para evitar esta situación, una solución que normalmente se utiliza involucra al hiperparámetro *momentum*, el cual es una constante entre 0 y 1. Se usa para ponderar los gradientes anteriores con el fin de salir del mínimo local.

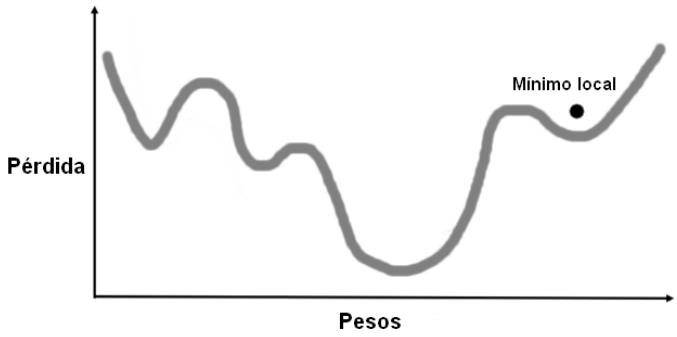


Figura 3.7. Ejemplo de curva con mínimos locales

Otro aspecto importante a tener en cuenta, a la hora de entrenar un modelo basado en redes neuronales, es el inicializar los pesos con unos valores aleatorios y pequeños para romper la simetría entre diferentes neuronas, si dos neuronas tienen los mismos pesos tendrán el mismo gradiente, eso supone que ambas tengan los mismos valores en las siguientes iteraciones y no serán capaces de aprender características diferentes. Continuando con los conceptos en el entrenamiento de redes neuronales, por iteración o *iteration* se entiende el número de lotes o *batches* necesarios para completar una época, por lo que el número de *batches* es igual al número de *iterations* para una época.

Por ejemplo, tenemos 160 imágenes de entrenamiento que vamos a utilizar, podemos dividir el conjunto de datos de 160 ejemplos en mini-lotes o *mini-batches* de 1 batch size cada uno, es decir, de 1 ejemplo cada uno. Se obtienen de esta manera 160 mini-lotes. Por lo que se necesitarán 160 iteraciones para completar 1 época. Donde el *batch size* es 1 y las iteraciones son 160, para 1 época completa.

La regularización de la red es otro de los conceptos importantes en el proceso de entrenamiento de la red. La regularización *dropout* es un método para la prevención del sobreajuste en redes profundas que consiste en suprimir aleatoriamente neuronas y sus conexiones de la red neuronal durante el entrenamiento. *Dropout* perturba la red en cada pasada de entrenamiento, eliminando al azar algunas de las unidades de cada capa.

El objetivo de esto es que, al introducir ruido en el proceso de entrenamiento, evitamos el sobreajuste, dado que en cada paso de la iteración estamos limitando el número de unidades que la red puede usar para ajustar las respuestas. La regularización *dropout* entonces busca una reducción en el sobreajuste que sea más provechosa que el consecuente aumento en el sesgo. Podemos hacer *dropout* de la capa de entrada, generalmente se utiliza en estos casos valores pequeños en torno a 0.2. En este caso, estamos evitando que el modelo dependa fuertemente de variables individuales. Por ejemplo, en procesamiento de imágenes, no queremos que por sobreajuste algunas predicciones estén ligadas fuertemente a un solo pixel, aun cuando en entrenamiento puede ser que un pixel separe bien los casos que nos interesa clasificar. En capas intermedias se usan generalmente valores más grandes alrededor de 0.5.

3.4.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés, o ConvNet) son un tipo de red neuronal profunda. Este tipo de redes tiene neuronas con parámetros en forma de pesos o *weights* y sesgos o *bias* que pueden ser aprendidos, tal y como ocurre en las redes neuronales clásicas como el perceptrón multicapa. En estas redes se presupone que las entradas son imágenes, esto permite que se puedan reducir los parámetros en la red y de esta manera solucionar el problema de escalabilidad de las redes neuronales profundas clásicas con una estructura totalmente conectada. Por ejemplo, una imagen mediana de 250 x 250 pixeles con colores RGB, da lugar a

neuronas con 187500 pesos, el resultado de la multiplicación 250x250x3. Esto supone un desperdicio de parámetros y recursos de computo enorme.

Las CNN eliminan la necesidad de una extracción de características manual, por lo que no es necesario identificar las características utilizadas para clasificar las imágenes. Las CNN funcionan mediante la extracción de características directamente de las imágenes. Las características relevantes no se entrenan previamente, se aprenden mientras la red se entrena con una colección de imágenes. Esta extracción de características automatizada hace que los modelos de *Deep Learning* sean muy precisos para tareas de visión artificial, tales como la clasificación de imágenes.

Las CNN toman de forma consecutiva pequeñas piezas de información de la imagen para posteriormente combinar toda esa información en las capas más profundas. A media que se avanza por las capas de una CNN se va yendo de unas primeras capas donde se extraen las características más generalistas de las imágenes a unas capas finales donde se extraen las características más precisas. Una primera capa convolucional puede aprender elementos básicos como bordes y una segunda capa convolucional puede aprender patrones compuestos de elementos básicos aprendidos en la capa anterior. Y así sucesivamente hasta ir aprendiendo patrones muy complejos.

Esto permite que las redes neuronales convolucionales aprendan eficientemente conceptos visuales cada vez más complejos y abstractos. Una conocida CNN es la que se muestra en la figura 3.8, la denominada VGG16. Esta CNN ha sido usada como extractora de características en las primeras versiones de *Faster* R-CNN.

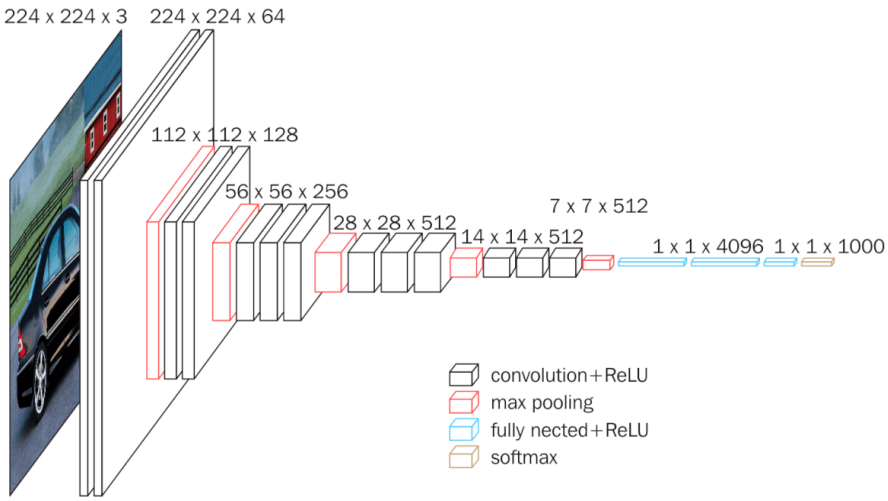


Figura 3.8. CNN VGG16. 32. Simonyan, K. y Zisserman, A. (2014.)

Las CNN basadas en regiones o R-CNN son normalmente usadas para la detección de objetos en imágenes. En R-CNN, la CNN se ve obligada a concentrarse en una sola región a la vez porque de esa manera se minimiza la interferencia ya que se espera que solo un objeto de interés domine en una región determinada. Las regiones en la R-CNN se detectan mediante un algoritmo de búsqueda selectiva seguido de cambio de tamaño, normalmente reducción de tamaño, para que las regiones tengan el mismo tamaño antes de que se envíen a una CNN que efectúe la clasificación y la regresión del cuadro delimitador o *bounding box*.

De forma general las CNN van a estar constituidas por una estructura que contiene 3 tipos capas, de convolución, de reducción o *pooling* y clasificadora totalmente conectada o *dense*. Las capas que definen realmente a las CNN se pueden definir como un grupo de neuronas especializadas en dos operaciones, la operación de convolución y la operación de reducción o *pooling.*

Las capas convoluciones operan sobre tensores en tres dimensiones, llamados mapas de características o *feature maps*, con dos ejes espaciales de altura y anchura, además de un eje de canal también llamado profundidad o *depth*. Para una imagen de color RGB, la dimensión del eje *depth* es 3, pues la imagen tiene tres canales: rojo, verde y azul (*red, green, blue*). La diferencia fundamental entre una capa completamente conectada o dense y una capa convolucional es que la capa completamente conectada aprende patrones globales en su espacio global de entrada, la imagen completa, mientras que las capas convolucionales aprenden patrones locales en pequeñas ventanas de dos dimensiones llamadas *kernel*.

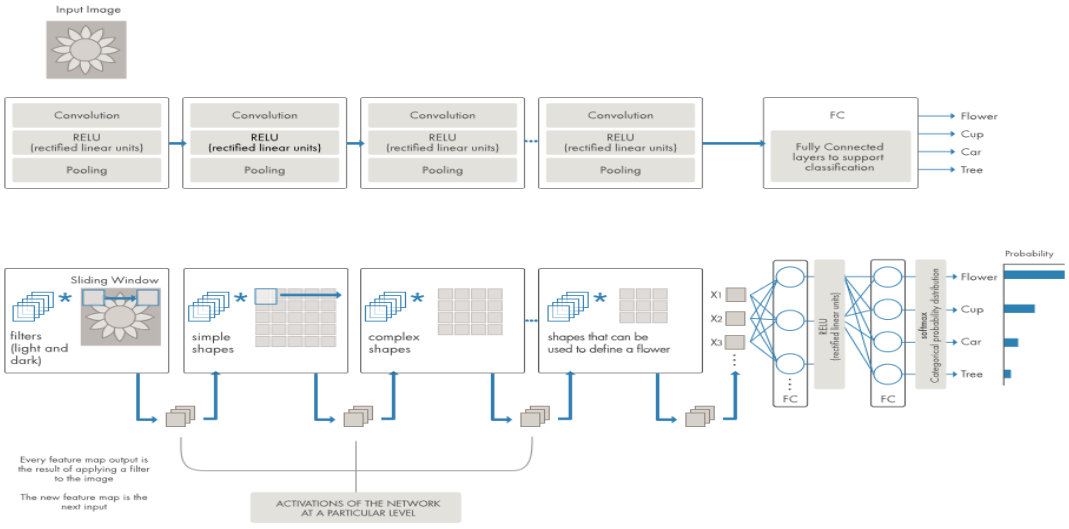


Figura 3.9. Ejemplo de CNN. Se aplican filtros a cada imagen de entrenamiento con distintas resoluciones, y la salida de cada imagen convolucionada se emplea como entrada para la siguiente capa.[26]

[26] https://www.mathworks.com/discovery/deep-learning.html

Las capas de reducción o de *pooling* suelen ser aplicadas inmediatamente después de las capas convolucionales como se observa en el ejemplo de la figura 3.9. Las capas de *pooling* hacen una simplificación de la información recogida por la capa convolucional y crean una versión condensada de la información contenida en estas. Existen diferentes formas de condensar esta información, de entre ellas la más usada es la conocida como *max-pooling*. Donde el valor que se conserva es el valor máximo de los que había en la ventana de entrada observada. En la figura 3.10 se muestra un ejemplo de la operación que realiza *max-pooling*. En general el *max-pooling* tiende a funcionar mejor que las soluciones alternativas como es el caso de *average-pooling*, donde cada grupo de puntos de entrada se transforma en el valor promedio del grupo de puntos.

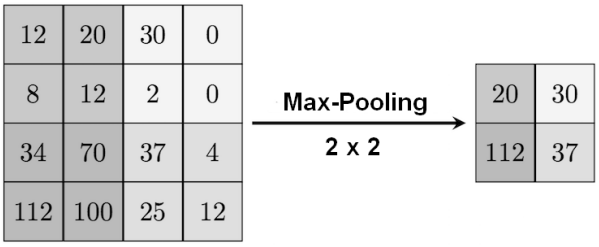


Figura 3.10. Ejemplo de operación de *Max-pooling*

Su utilidad principal radica en la reducción de las dimensiones espaciales, el ancho y el alto, del volumen de entrada para la siguiente capa convolucional. No afecta a la dimensión de profundidad. La operación realizada por esta capa también se llama reducción de muestreo, ya que la reducción de tamaño conduce también a la pérdida de información.

Reducir demasiado las dimensiones puede causar pérdida de información. A este concepto se le conoce como cuello de botella representativo o *bottleneck*. Sin embargo, una pérdida de este tipo puede ser beneficioso para la red por dos razones. La disminución en el tamaño conduce a una menor sobrecarga de cálculo para las próximas capas de la red y también reduce el sobreajuste.

Al final de las capas convolucional y de reducción se suele utilizar capas completamente conectadas o *dense*, en las que cada píxel se considera como una neurona separada al igual que en un perceptrón multicapa. La última capa de esta red es una capa clasificadora que tendrá tantas neuronas como el número de clases a predecir.

En las redes neuronales convolucionales tenemos varios hiperparámetros asociados a las propias operaciones de convolución. El tamaño de la ventana del filtro de convolución es uno de ellos y es el que mantiene la información de píxeles cercanos espacialmente. Tiene un mismo alto y ancho, sus dimensiones suelen estar entre 2x2 y 16x16. Un paso o *stride* es otro de estos hiperparámetros, el cual nos indica el

número de pasos en que se mueve la ventada de los filtros. A pesar de que valores de paso o *stride* grandes harían decrecer el tamaño de la información que se pasara la siguiente capa, los pasos en las capas convolucionales para reducir el tamaño son raramente utilizados, para ello se usan usualmente las operaciones de reducción o *pooling*. El número de filtros nos indica el número de características que queremos manejar, la cual será la profundidad de salida o *output_depth*. Acostumbran a ser de 32 o 64 el número de filtros a aplicar en las capas de convolución. De esta manera el resultado son 32 o 64 niveles de profundidad los que se obtienen.

En el caso de que se quiera obtener una imagen de salida de las mismas dimensiones de la entrada, se puede usar para ello el hiperparámetro *padding* en las capas convolucionales. Con *padding* podemos agregar ceros alrededor de las imágenes de entrada antes de hacer deslizar la ventana por ella. Para que el filtro de salida tenga el mismo tamaño que la imagen de entrada, se puede añadir a la imagen de entrada una columna a la derecha, una columna a la izquierda, una fila arriba y una fila debajo de ceros.

3.5. Detección de objetos en imágenes

Uno de los problemas más conocido en visión artificial consiste en clasificar una imagen en una de muchas categorías diferentes. Similar a la clasificación, la localización encuentra la ubicación de un solo objeto dentro de la imagen. Se puede combinar con la clasificación no solo para ubicar el objeto sino también para clasificarlo en una de las muchas categorías posibles. La detección de objetos es el problema de encontrar y clasificar un número, que puede ser variable, de objetos en una imagen. En contraste con problemas como la clasificación, la salida de la detección de objetos es variable dado que la cantidad de objetos detectados puede cambiar de una imagen a otra.

A la hora de evaluar un modelo para la detección de objetos en imágenes es necesario medir la calidad de sus predicciones, para ello se utilizan diferentes métricas. La métrica *accuracy* mide la precisión de tus predicciones, es decir, el porcentaje de sus predicciones positivas que son correctas. Como exhaustividad, sensibilidad o *recall* se entiende la proporción de casos positivos encontrados. Se volverá con estos dos conceptos en el apartado de las matrices de confusión.

La intersección sobre la unión (IoU, por sus siglas en inglés) es una métrica de evaluación de la superposición entre dos regiones. Mide qué tan buena es nuestra predicción en el detector de objetos con la verdad básica, el límite del recuadro que contiene al objeto real. A menudo se usa esta métrica de evaluación utilizada en los desafíos de detección de objetos, como el popular *PASCAL VOC*.

Para aplicar la métrica de evaluación IoU necesitamos los cuadros delimitadores de la verdad fundamental, es decir, los cuadros delimitadores o *bounding boxes*, definidos en las imágenes de entrenamiento, que especifican en qué parte de la imagen se encuentra nuestro objeto. Además de los cuadros delimitadores necesitaremos también las cajas delimitadoras previstas de nuestro modelo. Mientras tengamos estos

dos conjuntos de cuadros delimitadores, podremos aplicar IoU como una división entre el numerador, el cual calculamos como el área de superposición entre el cuadro delimitador predicho y el cuadro delimitador de la verdad y el denominador que es el área de unión, o más simplemente, el área que abarca tanto el cuadro delimitador predicho como el cuadro delimitador de la verdad. Dividir el área de superposición por el área de unión produce nuestra puntuación IoU final. Generalmente una IoU > 0.5 se considera una predicción correcta.

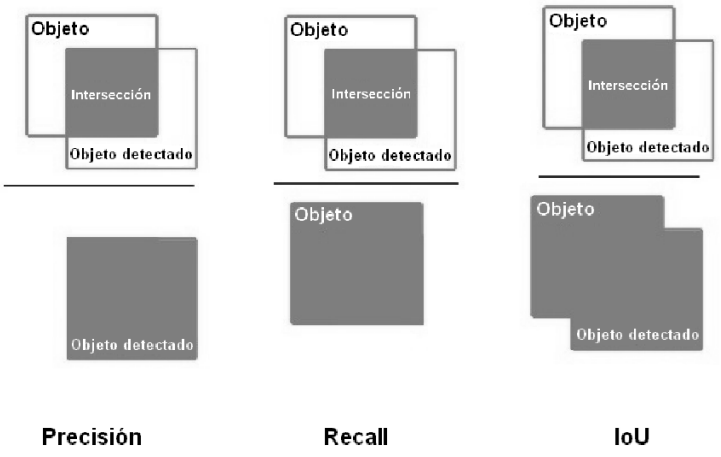


Figura 3.11. Visualización de las métricas de precisión, *recall* y IoU.

La ecuación para el cálculo de la métrica IoU donde el recuadro que enmarca al objeto a detectar es A y el recuadro que enmarca el objeto detectado es B sería el siguiente:

$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Una métrica ampliamente utilizada para calificar la calidad de un modelo es la exactitud media o *average precision* (AP, por sus siglas en inglés). La precisión promedio es un valor único el cual se usa para resumir la curva que forma la relación entre las métricas de *precision* y *recall.*

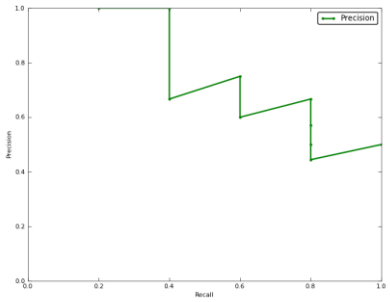


Figura 3.12 Visualización de la curva
que relaciona las métricas de *precision* y *recall*. [27]

De forma estricta la precisión promedio es la precisión media en todos los valores de *recall* entre 0 y 1, esto es lo mismo que el área bajo la curva *precision-recall*. Existe otra aproximación denominada exactitud media interpolada o *interpolated average precision,* pero también llamada comúnmente como AP. Este es el caso de *the PASCAL Visual Objects Challenge,* la cual ha usado ésta como su métrica de evaluación de modelos y la denomina PASCAL VOC AP@0.5IOU*.*

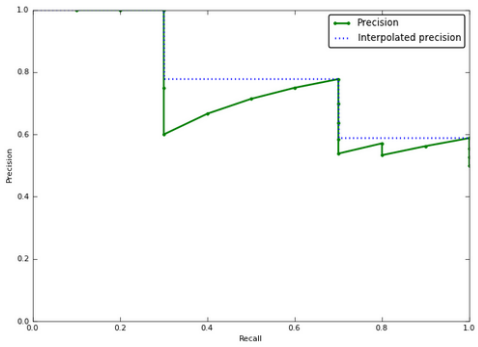


Figura 3.13. Aproximación de la
precisión media interpolada a la curva. [28]

Precisión media interpolada, en lugar de usar la precisión que se observó en una recuperación de n imágenes, usa la precisión máxima observada en todos los cortes con una *recall* mayor. Usualmente se toman muestras en 11 puntos fijos de 0 a 1, por ejemplo {0, 0.1, 0.2, …, 0.9, 1.0} para su cálculo.

La métrica utilizada comúnmente para medir en su conjunto los desafíos de detección de objetos se denomina exactitud media promedio (mAP, por sus siglas en inglés). Es

[27] https://sanchom.wordpress.com/tag/average-precision/

[28] https://sanchom.wordpress.com/tag/average-precision/

simplemente la media de las exactitudes promedio calculadas sobre todas las clases detectadas. Se calcula mediante la media de los AP para cada clase, dándonos un mAP para cada clase individual. Para muchos conjuntos de datos es interesante examinar el mAP por clase para poder detectar si el detector de objetos tiene dificultades con una clase en particular. Obteniendo la media de todos los mAP de las clases se obtiene el mAP final para el conjunto de datos de N clases. La siguiente ecuación expresa este cálculo:

$$mAP = \frac{1}{N} \sum_{class=1}^{N} AP_{class}$$

3.6. Matriz de confusión para múltiples valores de recuento

Con el objetivo de evaluar los resultados obtenidos usaremos una matriz de confusión para cada una de las clases. Una matriz de confusión nos dará una mejor idea de cómo está detectando y realizando el recuento nuestro modelo, dándonos un conteo de los aciertos y errores de cada uno de los recuentos y detecciones de las clases que se estén detectando. De esta manera, se puede comprobar si el modelo está confundiéndose entre clases de frutas o no está detectando correctamente y en qué medida.

Cada columna de la matriz representará el número de detecciones para cada clase de fruta realizada por el modelo y cada fila los valores reales de existencias por cada clase de fruta. Los recuentos para cada una de las clases quedan agrupados en la matriz en:

- Verdaderos positivos (TP, por sus siglas en inglés), son la cantidad de verdaderos positivos. Detecciones y recuentos correctos con respecto a los datos reales para la clase.
- Falsos negativos (FN, por sus siglas en inglés) son el número de falsos negativos. A estos casos también se les denomina errores de tipo 2. Los falsos negativos quieren decir que no se ha detectado esa clase de fruta o que se ha detectado, pero erróneamente como otra clase de fruta, cuando esa clase de fruta sí que existe y debería haber sido detectada y contabilizada.
- Falsos positivos (FP, por sus siglas en inglés) FP son el número de falsos positivos. A estos casos también se les denomina errores de tipo 1. Los falsos positivos resultarán de la detección y recuento de una clase de fruta cuando realmente no se encuentra esa detección y recuento en los datos reales.
- Verdaderos negativos (TN, por sus siglas en inglés) son el número de verdaderos negativos. Los valores de detección y recuento de la clase no son detectados y en los valores reales registrados confirman esta información.

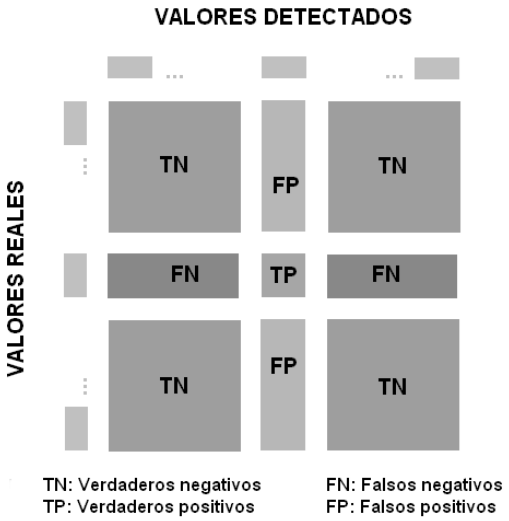


Figura 3.14. Matriz de confusión para múltiples valores de recuento en una clase

En la figura 3.14. se muestra como quedan distribuidos los valores TP, TN, FP y FN en una matriz de confusión para múltiples valores de recuento en una clase de fruta determinada durante el proceso de estudio. Con estos los valores se pueden calcular las siguientes métricas para evaluar el sistema propuesto.

- Exhaustividad o *recall*: también se la llama sensibilidad o *tasa de verdaderos positivos*. Nos da la probabilidad de que, dada una observación realmente positiva, el modelo acierte.

$$Sensibilidad = TP / (TP+FN)$$

- Especificidad o *specificity*: también llamado *ratio de verdaderos negativos*. Nos da la probabilidad de que, dada una observación realmente negativa, el modelo la detecte como tal.

$$Especificidad = TN / (TN+FP)$$

- Exactitud o *precision*: también llamado *valor de predicción positiva*. Nos da la probabilidad de que, dada una detección positiva, la realidad sea positiva también.

$$Precision= TP / (TP+FP)$$

- Precisión *o accuracy*: Porcentaje total de los aciertos de nuestro modelo.

$$Accuracy = (TP+TN) / (TP+TN+FP+FN)$$

- Prevalencia o *prevalence*: La probabilidad de un positivo en el total de la muestra.

$$Prevalencia =(TP+FN) / (TP+TN+FP+FN)$$

- *F1-score* es la media de *precision* que tiene una evaluación. Se emplea para determinar un valor único ponderado de las métricas *precision* y *recall.*

$$F1\text{-}score = 2*((precision * recall) / (precision + recall))$$

4. Bases de datos

4.1. Base de imágenes de entrenamiento

Las fuentes de imágenes utilizadas han sido la base de imágenes denominada COCO y conjunto de imágenes *ImageNet*. Las imágenes de ambas son accesibles de forma online.

COCO (*Common Objects in Context*)[29] es un gran conjunto de imágenes para la detección, segmentación y el subtitulado de objetos. Contiene más de 300000 imágenes de las cuales más de 200000 están etiquetadas. En dichas imágenes tienen identificados más de 1500000 de objetos de 80 categorías diferentes. El conjunto de imágenes COCO es el utilizado por los modelos utilizados en el presente estudio para su previo entrenamiento. Forman parte de un conjunto de modelos previamente entrenados, denominado *detection model zoo*, proporcionado por la API para la detección de objetos de *tensorflow*.

ImageNet[30] es un conjunto de datos de imágenes organizado de acuerdo con la jerarquía de *WordNet. WordNet* es una base de datos léxica del idioma inglés, donde cada concepto significativo está descrito por varias palabras o frases de palabras denominados "conjunto de sinónimos" o "*synset*". Hay más de 100000 *synsets* en *WordNet. ImageNet* tiene actualmente más de 14000000 imágenes y unos 22000 *synsets*, con un promedio de 1000 imágenes para cada *synset*. El número de imágenes con anotaciones para la delimitación de recuadros o *bounding boxes* es más de 1000000.

ImageNet proporciona la fuente perfecta para completar, con imágenes específicas de las clases que nos interesan en este estudio, el modelo previamente entrenado con el conjunto de imágenes COCO. Las diferentes clases de objetos a detectar en el estudio son 6 tipos de frutas, esto es; manzanas, peras, naranjas, plátanos, piñas y limones. En ImageNet existen los seis tipos de frutas, pero no es así en el conjunto de imágenes COCO, donde no se encuentran imágenes etiquetadas con peras, piñas o limones. Debido a esto, el modelo previamente entrenado con el conjunto de imágenes COCO no detectará estos tipos de frutas cuando se encuentren en la imagen.

Un ejemplo de esta limitación se muestra en la figura 4.1 para el modelo pre-entrenado "*Faster*_rcnn_inception_v2_coco". Donde, en la identificación de las clases de frutas en las imágenes, se observa que los limones o las peras no se identifican.

[29] http://cocodataset.org/

[30] http://www.image-net.org/

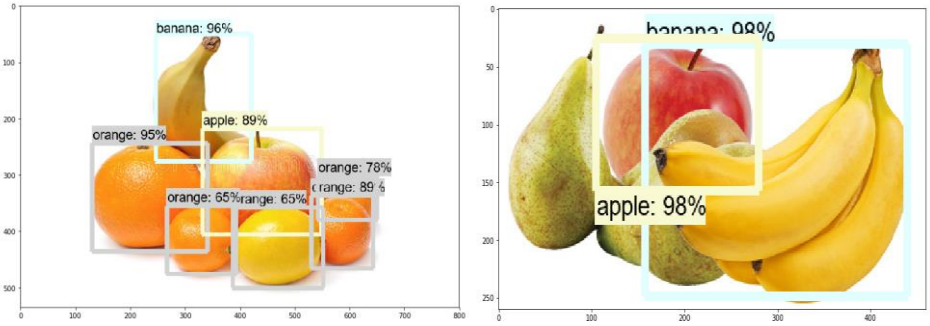


Figura 4.1. Limitación de clases de frutas con el modelo pre-entrenado "*Faster*_rcnn_inception_v2_coco"

4.1.1 Procesamiento de las imágenes

Se han seleccionado 200 imágenes para cada una de las clases descargadas de ImageNet. Esta selección ha buscado la heterogeneidad de las imágenes que compondrán el conjunto de entrenamiento y testeo. Siendo todas las imágenes totalmente diferentes entre sí, sin ningún patrón destacable entre ellas más allá de la misma clase de fruta que pueda aparecer en ellas. La selección se dividirá en un 80%, 160 imágenes por clase, para el conjunto de imágenes de entrenamiento y en un 20%, 40 imágenes por clase, para el conjunto de imágenes de evaluación. Las muestras de entrenamiento y evaluación se cogerán de forma aleatoria del conjunto de las 200 imágenes por clase.

Una misma imagen no podrá estar en el conjunto de entrenamiento y de evaluación de su misma clase. Las imágenes de evaluación, de esta manera, estarán ocultas al proceso de entrenamiento. Con la intención de preparar las imágenes para sus anotaciones y configuración en la API para la detección de objetos de tensorflow, las 160 imágenes por clase del conjunto de entrenamiento se colocarán en un directorio llamado train/. De la misma manera las 40 imágenes por clase del conjunto de testeo se colocan en un directorio llamado eval/.

Con respecto a la reducción de imágenes, en el fichero de configuración del modelo en la API para la detección de objetos de tensorflow, es posible ejecutar un preprocesamiento y realizar modificaciones en las imágenes para el entrenamiento. Entre estas modificaciones previas al entrenamiento está la de reducir la dimensión espacial de las imágenes. Es posible también añadir la posibilidad de usar el denominado aumento de los datos o *data augmentation* en las imágenes en el mismo fichero de configuración del modelo de la API. Entre otras opciones tenemos el volteo horizontal o *horizontal flipping* y el volteo vertical o *vertical flipping* de las imágenes de entrenamiento. En las figuras 4.2 y 4.3 se muestran ejemplos de estas dos opciones.

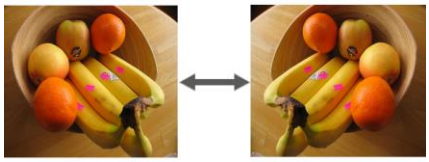


Figura 4.2. *Horizontal flipping* para *data augmentacion*.

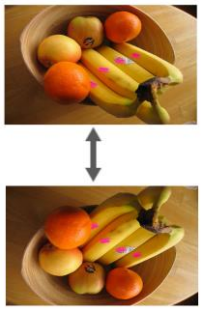


Figura 4.3. *Vertical flipping* para *data augmentacion*.

4.1.2. Creación de las anotaciones

La creación de los cuadros delimitadores de los objetos en las imágenes o *bounding boxes* es un punto fundamental en la preparación de las imágenes de entrenamiento para la detección de objetos. En el presente trabajo, cada cuadro delimitador dentro de una imagen delimita una de las seis clases de frutas que hemos definido. Estos cuadros se definen mediante las llamadas anotaciones, las cuales son las cuatro coordenadas en pixeles que describen las cuatro esquinas del cuadro que rodea al objeto dentro de la imagen.

Mediante la herramienta LabelImg se procede manualmente a realizar las delimitaciones de los recuadros de los objetos en las imágenes de entrenamiento, la cual genera las anotaciones de cada una de las *bounding boxes*. La herramienta está escrita en Python y usa Qt para su interfaz gráfica, Qt es un marco de trabajo multiplataforma orientado a objetos usado para desarrollar programas con interfaz gráfica de usuario. En la figura 4.4 se muestra el momento en el que se crea un cuadro delimitador para contener a una pera en LabelImg.

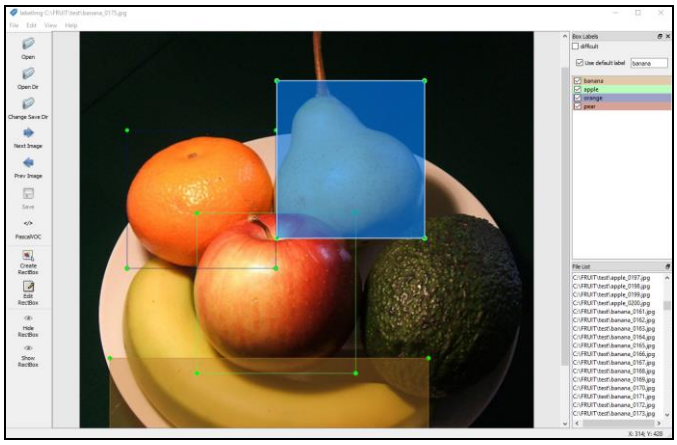


Figura 4.4. Creando *bounding boxes* en LabelImg

Las anotaciones realizadas por LabelImg se guardan como archivos con extensión xml, en formato VOC PASCAL. Este es el formato utilizado por ImageNet. El fichero con extensión xml resultante tendrá el mismo nombre de la imagen a la que se le estén realizando la anotación o anotaciones. Una vez que haya etiquetado y guardado cada imagen, habrá un archivo .xml para cada imagen en los directorios eval/ y train/ de la API. En el anexo 1 se muestra un ejemplo de fichero xml en formato VOC PASCAL con una anotación registrada para la clase "*apple*".

Se ha descargado 200 imágenes desde ImageNet para cada uno de los *synsets apple, pear, orange, banana, pineapple* y *lemon*. No se han descargado las anotaciones de las imágenes dado que este punto se realiza manualmente en las imágenes descargadas, tal y como se describe anteriormente. Se decide conservar los nombres en inglés de las distintas clases de frutas para la identificación de anotaciones y etiquetas.

Con las imágenes etiquetadas y con sus respectivas anotaciones en los ficheros xml, generamos un fichero csv que contenga todas las anotaciones de todas las imágenes de entrenamiento y otro que contenga todas las anotaciones de todas las imágenes de entrenamiento. Cada línea en cada uno de los dos ficheros contendrá el nombre de la imagen, el ancho y alto de la imagen, el nombre de la clase y las cuatro coordenadas del cuadro delimitador xmin, ymin, xmax y ymax. Estos dos ficheros csv son necesarios para generar con ellos los ficheros tfrecords. Los ficheros tfrecords son los ficheros que proporcionan los valores de entrada para el entrenamiento del modelo. Los identificadores de las clases en estos tfrecords deben ser exactamente iguales que los identificadores de las clases indicados en otro fichero importante en la preparación para el nuevo entrenamiento del modelo con nuestras 6 clases de frutas. Este fichero es el denominado mapa de etiquetas o *labelmap*, y debe tener la extensión pbtxt. En el Anexo 2 se muestra el fichero labelmap.pbtxt donde se encuentran las seis clases de frutas y su correspondientes identificadores.

4.2. Base de imágenes de producción

Las imágenes de producción son las imágenes que se captan del frutero. El frutero contendrá las distintas clases de frutas que hemos definido en las imágenes de entrenamiento del modelo. Las imágenes están tomadas directamente, por medio de un dispositivo móvil Samsung S7, para seguidamente realizar una detección de las diferentes clases predefinidas de fruta en ellas. La resolución de las imágenes de producción será de 540 x 540 pixeles. Durante el periodo de estudio de 30 días se sacan 5 fotos diarias al frutero donde se coloca la fruta diariamente, desde diferentes perspectivas, delante, detrás, a la derecha, a la izquierda y una última desde arriba. El total de imágenes de producción previsto para el estudio es de 150.

Figura 4.5. Las 5 imágenes correspondientes al día 6 de estudio.

Las imágenes de producción se toman en un entorno controlado, donde las distintas clases de frutas estarán siempre colocadas dentro de un frutero circular de cristal transparente. Este frutero tiene unas dimensiones de 30 cm de diámetro por 15 cm de alto.

Aun siendo un entorno controlado, la disposición de las piezas de fruta en su interior es variable y no predecible. Las piezas de fruta se van colocando a medida que llegan después de recibir un pedido. Dicho pedido diario puede ser de una o varias clases de frutas. Las diferentes clases de frutas pueden quedar apiladas y distribuidas en el frutero de una infinidad de maneras. En algunos casos las frutas serán completamente visibles y se identificarán fácilmente y en otros casos será un reto tanto su detección como su recuento debido, entre otras cosas, a que pueden quedar ocultas en el frutero por otras frutas. Las imágenes se han capturado siempre desde la misma distancia y con un mismo tipo de luz blanca para la no distorsión de los colores.

5. Modelos

5.1. Modelos pre-entrenados ofrecidos por la API

Tensorflow detection model zoo es un conjunto de modelos pre-entrenados para la detección de objetos con Tensorflow. Para el entrenamiento estos modelos han utilizado conocidos conjuntos de imágenes como *COCO dataset, Kitti dataset, Open Images dataset*, AVAv2.1 dataset y *iNaturalist Species Detection Dataset*. Se han entrenado estos modelos utilizando una tarjeta Nvidia Geforce GTX TITAN X.

En este punto se introduce el concepto de transferencia de aprendizaje o *transfer learning*. Entrenar desde cero una red neuronal convolucional completa es algo muy costoso, en términos de dinero y tiempo. Por estas razones, lo que normalmente se hace en investigaciones sin grandes recursos computacionales es lo que se denomina transferencia de aprendizaje. Para ello, se usa una red previamente entrenada como el punto de inicialización para su propio modelo, es decir, se usan los pesos de la CNN previamente entrenada como los pesos iniciales de su propio modelo. En el caso que queramos realizar predicciones sobre unas clases nuevas que no estaban categorizadas en el modelo pre-entrenado podemos hacer lo que se denomina afinar el modelo o *fine-tuning*.

La tarea de afinar una red es modificar los parámetros de una red ya entrenada para que se adapte a la nueva tarea en cuestión, en nuestro caso detectar una serie especifica de clases de fruta. Como se explicaba anteriormente, las capas iniciales aprenden características muy generales y a medida que avanzamos en la red, las capas tienden a aprender patrones más específicos para la tarea en la que se está capacitando. Por lo tanto, para el ajuste, se quiere mantener las capas iniciales intactas, lo que se denomina congelar dichas capas, para volver a entrenar las capas posteriores en la especifica tarea que se desee.

La transferencia de aprendizaje y *fine-tuning* evita dos grandes limitaciones. La cantidad de datos requeridos no es necesario que sea muy grande porque no estamos entrenando a toda la red y la parte que está siendo entrenada no está entrenándose desde cero. De la misma forma como los parámetros que deben actualizarse son menores, la cantidad de tiempo necesario también será menor.

Se ha añadido una función para los modelos que se usan en el estudio para, de una manera simple, realizar el recuento de las distintas clases de frutas en cada captura. La función contadora “contador” se ha añadido al script “visualization_utils.py” para el recuento de las cajas delimitadores detectadas para cada clase de fruta en cada imagen. En el Anexo 3 se muestran las líneas de código añadidos al script.

La elección de los modelos se ha decidido en base a los limitados recursos de computo. Se han elegido dos de los mejores modelos a los que poder ajustar y entrenar en el equipo usado.

5.2. Modelo pre-entrenado *Faster* R-CNN *Inception* v2

Para la detección de las clases de frutas en las imágenes de producción entrenaremos dos modelos de los ofrecidos por la API de Tensorflow. El primero de ellos es el modelo pre-entrenado "*Faster*_rcnn_inception_v2_coco". Un modelo que consiste en la unión de la arquitectura de *Faster R-CNN* una de las arquitecturas de redes neuronales convolucionales para la detección de objetos más conocidas, con el extractor de características Inception v2.

La arquitectura de *Faster* R-CNN se compone de 3 partes:

- Una red convolucional profunda realizando la función de extractor de características. Esta es la parte a la que sustituye Inception v2 en el modelo pre-entrenado.
- Una red convolucional profunda encargada de realizar propuestas de regiones de interés (RPN, por sus siglas en inglés) de recoger la salida del extractor de características y devolver regiones que contienen propuestas para las clases de objetos que se quiere detectar.
- El clasificador *Fast* R-CNN, que utiliza las regiones propuestas para asignar las clases que correspondan a cada objeto.

La diferencia principal con su antecesor, la arquitectura *Fast* RCNN, es que ésta utiliza la búsqueda selectiva para generar las propuestas de regiones. El costo de tiempo de generar propuestas de regiones es mucho menor en RPN que en la búsqueda selectiva, ya que RPN comparte la mayoría de los cálculos con la red de detección de objetos. Rápidamente, RPN clasifica los cuadros de región, llamados anclajes o *anchors* y propone los que probablemente contienen los objetos.

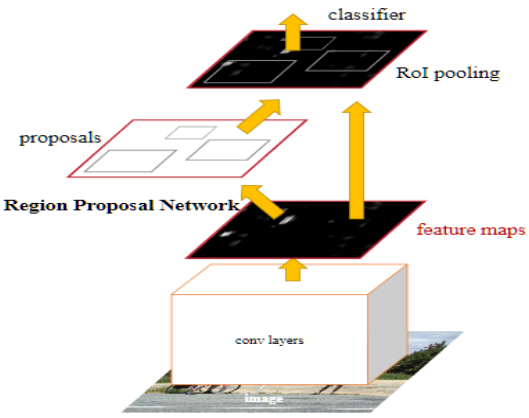


Figura 5.1. Arquitectura de *Faster* R-CNN. (Ren *et al*., 2016)

El concepto de *anchor* juega un papel importante en *Faster* R-CNN. Un *anchor center* es un punto de referencia y su función es la de servir de base para generar propuestas de cuadros delimitadores también llamados anclajes o *anchors*. Los tipos de *anchors* se deciden por dos factores, las escalas y las relaciones de aspecto. La configuración de *Faster* R-CNN que se usa para ambos modelos en el presente estudio considera 4

escalas y 3 relaciones de aspecto por lo que hay en total 4x3, es decir, 12 *anchors* por cada punto de anclaje. La idea es colocar sobre la imagen simétricamente distribuidos los puntos de anclaje o *anchor centers*. Sobre cada punto de anclaje se colocan diferentes recuadros delimitadores o *anchors*, y cada uno de estos puede ser candidato de contener un objeto de interés. Sobre cada punto de anclaje se coloca un número k=12 *anchors*, 4 escalas y 3 relaciones de aspecto.

El resultado de una red de propuestas de regiones (RPN) es un conjunto de propuestas de estos recuadros o *anchors* que serán examinados por un clasificador y un regresor para verificar finalmente la ocurrencia de objetos. Para ser más precisos, RPN predice la posibilidad de que un *anchor* sea de fondo o de primer plano y refina el anclaje.

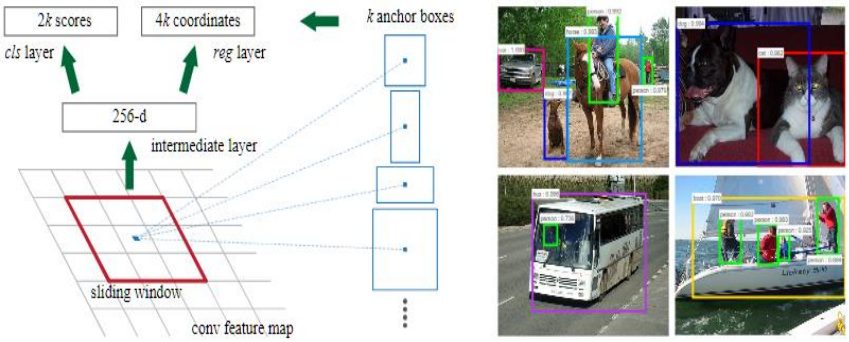


Figura 5.2. Generación de propuestas sobre
la imagen de RPN (Ren et al., 2016)

Como se muestra en la figura 5.2.1, con RPN obtenemos regiones propuestas con diferentes tamaños. Regiones de diferentes tamaños significa mapas de características o *feature maps* de diferente tamaño. No es fácil hacer una estructura eficiente para trabajar con características con diferentes tamaños. La agrupación de regiones de interés ROI simplifica este problema al reducir los mapas de características al mismo tamaño. A diferencia de *Max-Pooling*, que tiene un tamaño fijo, ROI *Pooling* divide el mapa de características de entrada en un número fijo k de regiones aproximadamente iguales, y luego aplica *Max-Pooling* en cada región. Por lo tanto, la salida de ROI *Pooling* es siempre k independientemente del tamaño de la entrada.

Inception v2 es una CNN que realiza convoluciones con filtros de diferentes tamaños os. Esto se hace debido a que el tamaño del objeto que se desea clasificar es muy variable con respecto a la imagen. Inception v2 comparte la premisa de su predecesor, Inception v1, pero consigue un rendimiento mucho mejor. El modelo pre-entrenado utiliza el extractor de características Inception v2. Inception v2 persigue reducir el cuello de botella representacional o *bottleneck*. Reducir demasiado las dimensiones

puede causar la pérdida de información, lo que se conoce como un cuello de botella representativo. Usando métodos inteligentes de factorización, las convoluciones pueden hacerse más eficientes en términos de complejidad computacional. La solución planteada fue factorizar la convolución 5x5 a dos operaciones de convolución 3x3 para mejorar la velocidad de cálculo. Sin embargo, una convolución 3x3 es equivalente a realizar primero una convolución 1x3 y luego realizar una convolución 3x1 en su salida. Descubrieron que este método es un 33% más eficiente que la única convolución de 3x3. De esta forma los bancos de filtros en el módulo se expandieron y se ampliaron en lugar de aumentar su profundidad.

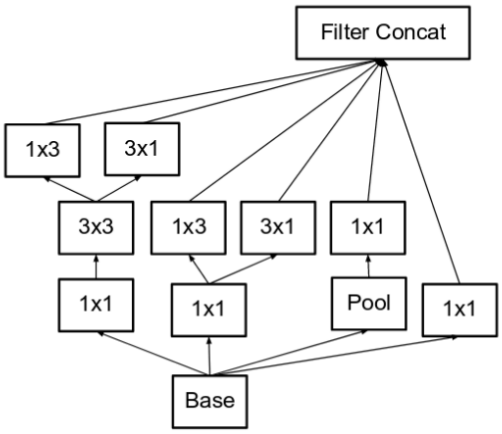


Figura 5.3. El módulo Inception ampliado en
Inception v2 (Szegedy et al., 2015).

Por último, la tarea del clasificador es la de un clasificador de imágenes clásico, recibe el mapa de características de las regiones propuestas como entrada y les asigna una clase. Se añade una clase extra, la cual hace referencia al fondo de la imagen, ésta se utiliza para descartar regiones mal propuestas. Además, en esta última fase del detector se modifica el cuadro delimitador que contendrá el objeto para ajustarlo más a los límites del objeto.

En el anexo 4 se muestra la configuración realizada del modelo en el fichero de configuración "*Faster*_rcnn_inception_v2_coco.config" para el proceso de entrenamiento y evaluación.

5.3. Modelo pre-entrenado *Faster* R-CNN ResNet-101

El segundo modelo que se entrena con las clases e imágenes del presente estudio es el modelo pre-entrenado "*Faster* R-CNN ResNet-101". En este caso se usa también la arquitectura de *Faster* R-CNN y el extractor de características es sustituido en esta ocasión por la red ResNet-101.

Ya se hablado en el anterior módulo de la arquitectura *Faster* R-CNN por lo que se pasa a describir el extractor de características ResNet-101. Se trata de una CNN con 101 capas convolucionales que utiliza conexiones residuales para hacer frente a problemas de degradación por saturación o del gradiente de fuga, donde las fluctuaciones se vuelven demasiado pequeñas para ser útiles, de las redes neuronales profundas con muchas capas.

Estos bloques residuales aprovechan el mapeo residual para preservar las entradas. En esta CNN las conexiones residuales permiten conexiones directas entre capas no consecutivas, es decir, una conexión directa entre capas que se salta las capas intermedias entre ellas. Mediante este método, las características se pueden propagar eficientemente por la red, siendo posible de esta manera entrenar CNN con más de 100 capas o incluso más de 1000. El bloque residual para ResNet-101 es de 3 capas de profundidad.

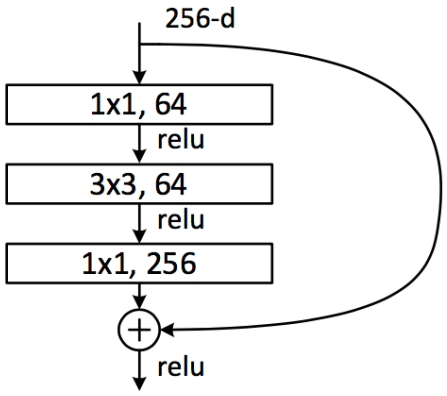


Figura 5.4. Bloque residual para
ResNet-101 (He, Zhang, Ren y Sun, 2016)

En el anexo 5 se muestra la configuración realizada del modelo en el fichero de configuración "*Faster*_rcnn_ResNet101_coco.config" para el proceso de entrenamiento y evaluación.

5.4. Sistema de control de existencias

El sistema de control de existencias se basa en la información detectada por el modelo con mejores resultados en la fase de evaluación. El estudio consta de 30 días, donde el recuento y el pedido se realizan diariamente a primera hora de la mañana. La fruta, del pedido realizado, se compra y se recibe en el plazo de una hora y las distintas clases de fruta se pueden consumir a cualquier hora del día. La rutina de ejecución diaria del sistema se describe a continuación.

- Recuento de piezas de fruta en el frutero.

Con el objetivo de reducir el error al contar las distintas clases de frutas, se diseña una rutina diaria para la toma de múltiples imágenes al frutero. La idea del recuento en base a múltiples imágenes se basa en la lógica de que, cuantas más visualizaciones de un grupo de objetos se tenga y, sobre todo en un contexto en 3 dimensiones, más fiable será el recuento que se obtenga.

Para la implementación de la anterior idea planteada, se toman 5 fotos diarias al frutero donde se coloca la fruta diariamente, desde diferentes perspectivas, delante, detrás, a la derecha, a la izquierda y una última desde arriba. Se realiza la detección y el recuento de las clases de fruta en cada una de las imágenes y guardaremos esa información temporalmente en un fichero.

A continuación, se guarda el máximo valor obtenido por clase de entre las 5 listas de detecciones y recuentos de fruta obtenidas en las 5 imágenes. Esto da como resultado un recuento más fiable que el que se pudiera obtener con sólo una imagen. A ese recuento final diario se le añade la fecha de las capturas y se guarda como una nueva línea en un fichero histórico de capturas al que llamaremos “historico_detectado.csv”.

Ejemplo de una de las líneas del fichero “historico_detectado.csv”:

{'apple':5, 'pear':4, 'orange':6, 'lemon':4, 'banana':2, 'pineapple':0, 'Fecha':'2018-10-09 '}

- Comprobaciones previas.

Antes de guardar la nueva línea de recuento en el fichero “historico_detectado.csv”, se realizan unas comprobaciones previas. El sistema debe comprobar los valores mínimos y máximos de existencias y corregir los valores de recuento en base a ellos. Cada una de las clases de frutas tiene un valor mínimo de 1 unidad y un máximo de 3, salvo en el caso de la piña en donde su valor mínimo y máximo es 1 unidad.

En base a los limites mínimos, si la cantidad de una clase es 0 se genera un nuevo pedido para esa clase de fruta. Por defecto se realiza un pedido 3 unidades de fruta cada vez que no haya unidades de esa clase, salvo en el caso de la piña que sólo se realiza un pedido de una unidad cuando no haya piña en el frutero. En base a los límites máximos, si en el recuento final del día de estudio se han obtenido más unidades que las que indica el máximo para la clase, debido a un error en el recuento, el sistema entenderá que el valor obtenido es el máximo para cada clase. En el caso de la piña 1 unidad y en el resto de las clases de frutas 3 unidades.

Con todos los pedidos de las clases se genera una nueva línea de pedido diaria en un fichero al que llamaremos “pedidos.csv”, a la que se le añade la fecha del pedido. En las líneas de pedido las clases con valor 0 quieren decir que no hay que comprar fruta de esa clase ese día.

Ejemplo:

{'apple':0, 'pear':3,'lemon':3, 'orange':0,'banana':0,'pineapple':3,'Fecha':'2018-10-15'}

- Cálculo de consumo diario.

Por otro lado, se calcula el consumo diario de cada clase de fruta basándonos en las líneas de recuento del fichero “historico_detectado.csv”. Se ha calculado viendo las diferencias con respecto a los recuentos del día anterior del mismo fichero y tomando sólo los valores negativos. Los valores negativos, en esa diferencia, indican consumo en unidades de las clases de fruta y los positivos, ingresos de unidades de fruta en el frutero debido a un pedido realizado.

Para comprobar la veracidad del recuento se comprueba si el consumo medio de cada clase de fruta es mayor al valor de la diferencia del recuento del día con respecto al día anterior y sumando el posible pedido realizado el día anterior después del recuento. Si el consumo medio es mayor a la diferencia más el pedido realizado entre ayer y hoy, esto quiere decir que existe la posibilidad de que haya fruta de esa clase en el frutero no detectada. En cambio, si el consumo medio es igual o menor a la diferencia más el pedido realizado entre ayer y hoy, esto querrá decir que posiblemente el recuento de esa clase sea correcto.

Ejemplo: El consumo medio de manzanas hasta hoy es de 1 manzanas. Ayer no había manzanas al no detectarse ninguna en el recuento final realizado ayer, por lo que se realizó un pedido de 3 manzanas. Hoy solo se detecta 1 manzana en el frutero en el recuento final realizado. En este caso existe la posibilidad de que haya 1 manzana en el frutero que no haya sido detectada.

La implementación del código del sistema de control de existencias se puede ver detallado en el Anexo 6.

6. Ejecución

6.1. Ejecución del entrenamiento de los modelos ajustados

Los modelos pre-entrenados para la detección de objetos en imágenes que se entrenan y evalúan son "*Faster R-CNN Inception v2*" y "*Faster R-CNN ResNet-101".* En el entrenamiento de ambos modelos se han añadido las opciones de *horizontal flip*, *vertical flip* y *rotation90* como opciones de *data augmentation*. Para el entrenamiento del modelo *Faster* R-CNN Inception v2 se ha usado un *learning rate* de 0.0002 para 200000 pasos. Los *mini-batches* en cada paso tienen un *batch-size* de 1, es decir, de 1 imagen. En el anexo 4 se pueden ver los detalles de configuración del modelo.

Para el entrenamiento del modelo *Faster R-CNN ResNet-101* se ha usado un learning rate de 0.0003 para 200000 pasos con un *batch-size* de 1 imagen. En el anexo 5 se pueden ver los detalles de configuración del modelo. En ambos casos se ha optado por usar una tasa de aprendizaje manual por paso y un valor para el hiperparámetro optimizador *momentum* de 0.9. La relación de aspecto de los anclajes ha sido igual en ambos [0.5, 1.0, 2.0] así como sus escalas iniciales de los anclajes [0.25, 0,5, 1.0, 2.0]. El número máximo de regiones propuestas es de 300 para ambos, con un umbral IoU de 0.7.

En la figura 6.1 observamos el proceso de entrenamiento de ambos modelos en las gráficas de tensorboard "*Losses/Loss/BoxClassifierLoss/classification_loss*" y "*Losses/Loss/BoxClassifierLoss/localization_loss*". En dicha figura se muestra la pérdida para la clasificación de los objetos detectados a lo largo del periodo de entrenamiento, así como para su localización. En ambos casos se indica que tan bien los modelos están actualizando sus pesos de acuerdo con sus datos en el proceso en clasificación y localización respectivamente. La línea naranja muestra los valores aplicados a los ejemplos de entrenamiento del modelo ajustado *Faster* R-CNN ResNet-101, y la línea roja muestra los del modelo ajustado *Faster* R-CNN Inception v2.

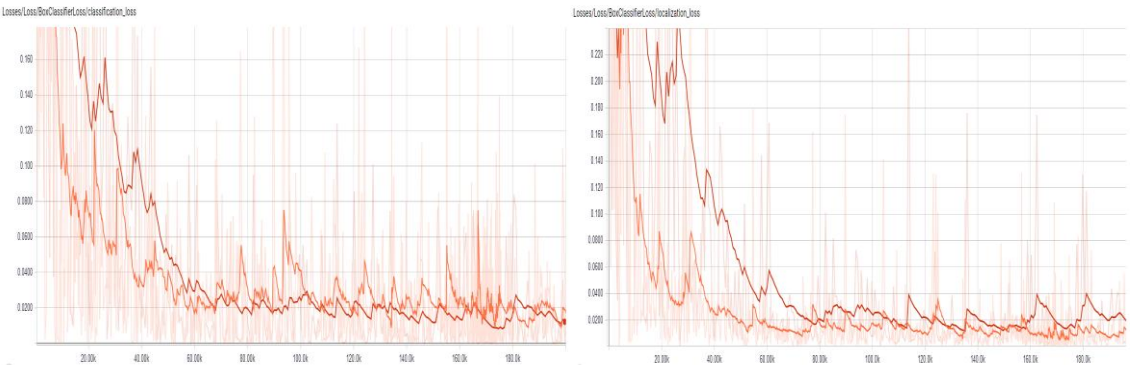


Figura 6.1. A la izquierda se muestran los valores de la función de pérdida en clasificación, y a la derecha los de localización para cada paso del entrenamiento.

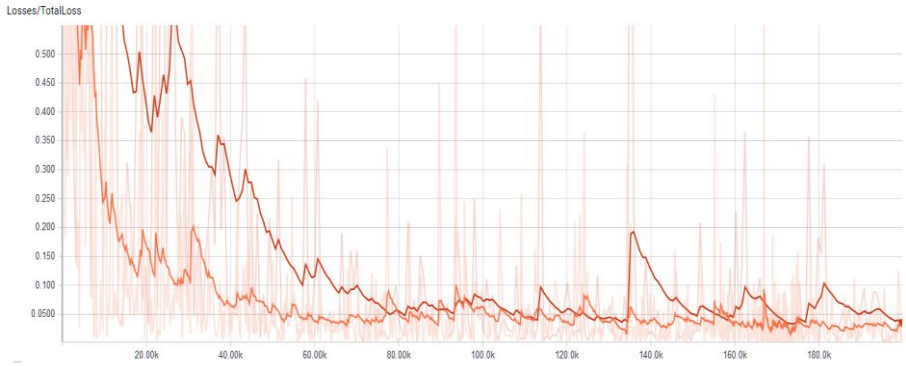


Figura 6.2. Valores globales de la función de pérdida para cada paso del entrenamiento.

Se observa un comportamiento similar a lo largo del entrenamiento en las funciones de pérdida en clasificación y localización en ambos modelos. Más pronunciado al comienzo del entrenamiento en el caso del modelo *Faster R-CNN ResNet-101* pero convergiendo entre ellos en numerosas ocasiones aproximadamente tras el primer tercio de entrenamiento. Los picos observados pueden deberse a estancamientos temporales durante el entrenamiento debido a mínimos locales. El valor global de la función de pérdida obtenido por el modelo *Faster R-CNN ResNet-101* al finalizar el entrenamiento de 200000 pasos ha sido de 0.00195, por 0.002505 del modelo *Faster* R-CNN Inception v2.

6.2. Ejecución con las imágenes de producción

Para la ejecución del modelo con las imágenes en producción se usará básicamente el código que proporciona la API de Tensorflow apuntando a nuestro modelo ajustado y teniendo en cuenta la carpeta donde se ubican las imágenes del día de estudio. Además, se vaciará el fichero temporal que usamos diariamente para el recuento basado en múltiples imágenes.

6.3. Ejecución del sistema de control de existencias

La ejecución del código para el control de existencias se realiza diariamente justo después de la ejecución del modelo para la detección y recuento de las clases de frutas en las 5 imágenes tomadas ese día. En el anexo 6 se muestra la salida por pantalla paso a paso de la ejecución del código para el control de existencias del día 30 de estudio.

7. Resultados

7.1. Resultados con las imágenes de evaluación

Durante el proceso de entrenamiento del modelo *Faster* R-CNN Inception v2 se han obtenido valores de detección de entre 0.16 y 0.18 segundos por paso, lo que equivale procesar por segundo unos 6 pasos aproximadamente. Esto ha dado para 200000 pasos unas 9 horas entrenamiento. Para el modelo *Faster R-CNN ResNet-101* se han obtenido valores de detección de entre 0.44 y 0.5 segundos por paso, lo que equivale procesar por segundo unos 2 pasos aproximadamente. Esto ha dado para 200000 pasos unas 27 horas entrenamiento. Para evaluar los resultados obtenidos en ambos modelos usaremos la métrica PASCAL VOC AP@0.5IOU.

En la figura 7.1 se observa el proceso de evaluación de ambos modelos en las gráficas de tensorboard "*Losses/Loss/BoxClassifierLoss/classification_loss*" y "*Losses/Loss/BoxClassifierLoss/localization_loss*". En la figura se muestra la pérdida para la clasificación y localización de los objetos detectados a lo largo del periodo de evaluación.

La línea azul oscura muestra los valores obtenidos a los ejemplos de entrenamiento del modelo ajustado *Faster* R-CNN ResNet-101, y la línea celeste muestra los del modelo ajustado *Faster* R-CNN Inception v2. Como en las figuras anteriores 6.1. y 6.2., La línea naranja muestra los valores aplicados a los ejemplos de entrenamiento del modelo ajustado *Faster* R-CNN ResNet-101, y la línea roja muestra los del modelo ajustado *Faster* R-CNN Inception v2.

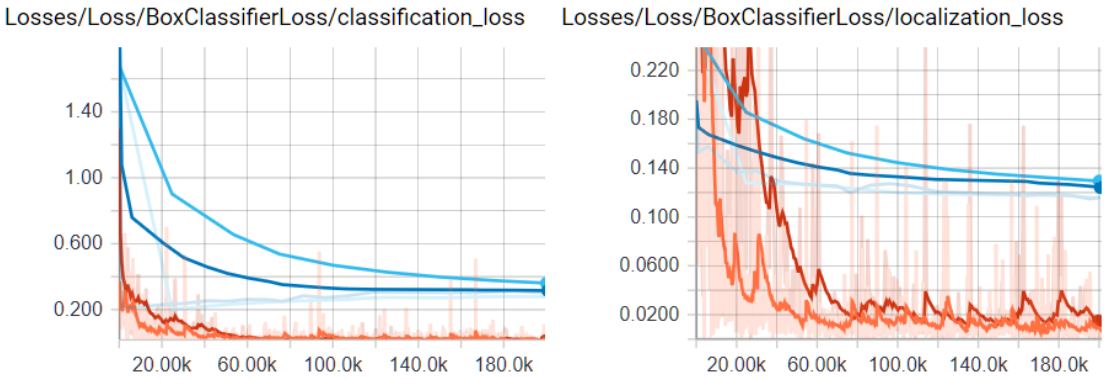


Figura 7.1. Valores de la función de pérdida del proceso de evaluación sobre los valores obtenidos durante el entrenamiento de los dos modelos, a la izquierda los valores obtenidos en clasificación y a la derecha los obtenidos en localización.

En ninguno de los dos modelos se observa sobreajuste durante el proceso de entrenamiento y evaluación, pero sí un estancamiento en el aprendizaje a partir de los 120000 pasos aproximadamente. Esto indica que los pesos no se están actualizando y por lo tanto el modelo no continúa aprendiendo como en pasos anteriores. Esto si puede indicar un subajuste causado por una falta de fuentes de imágenes de entrenamiento, con lo que el modelo no generalizará lo suficientemente bien el conocimiento adquirido. Entre las mejores planteadas del sistema se tiene en cuenta este hecho para enfatizar en el aumento de imágenes en un nuevo proceso de entrenamiento y evaluación con los modelos propuestos.

A continuación, se muestra una comparación de los AP@0.5IOU PASCAL VOC para cada una de las clases y el mAP@0.5IOU para la evaluación global de la detección de los modelos. Al igual que anteriormente la línea azul oscura muestra los valores obtenidos a los ejemplos de entrenamiento del modelo ajustado *Faster* R-CNN ResNet-101, y la línea celeste muestra los del modelo ajustado *Faster* R-CNN Inception v2.

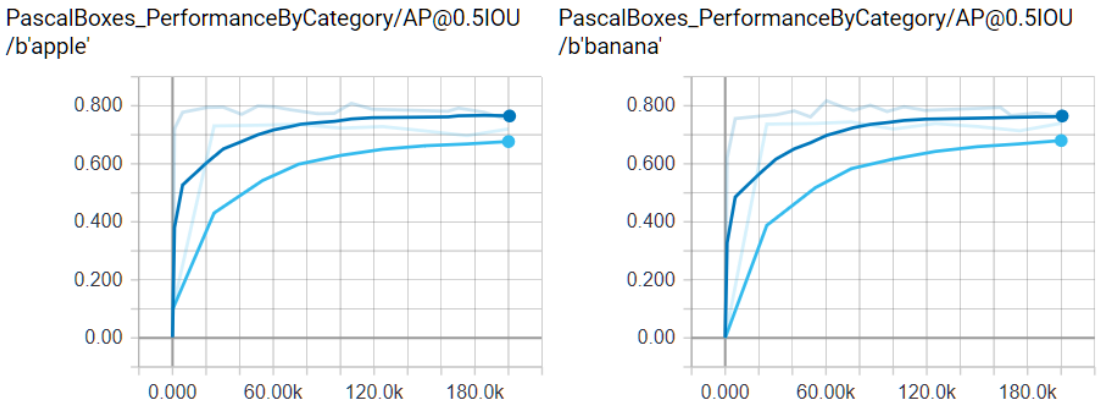


Figura 7.2. Comparación entre los valores PASCAL VOC AP@0.5 obtenidos para la detección de las manzanas, a la izquierda y para los plátanos, a la derecha.

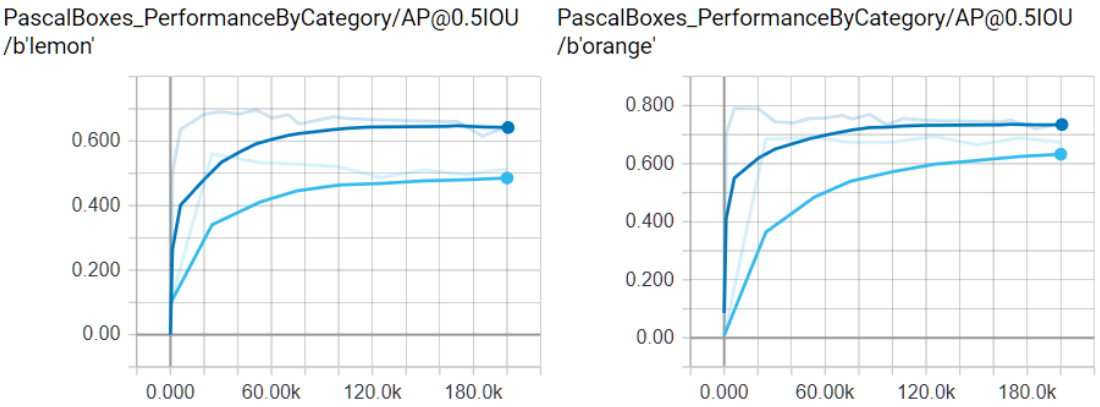


Figura 7.3. Comparación entre los valores PASCAL VOC AP@0.5 obtenidos para la detección de los limones, a la izquierda y para las naranjas, a la derecha.

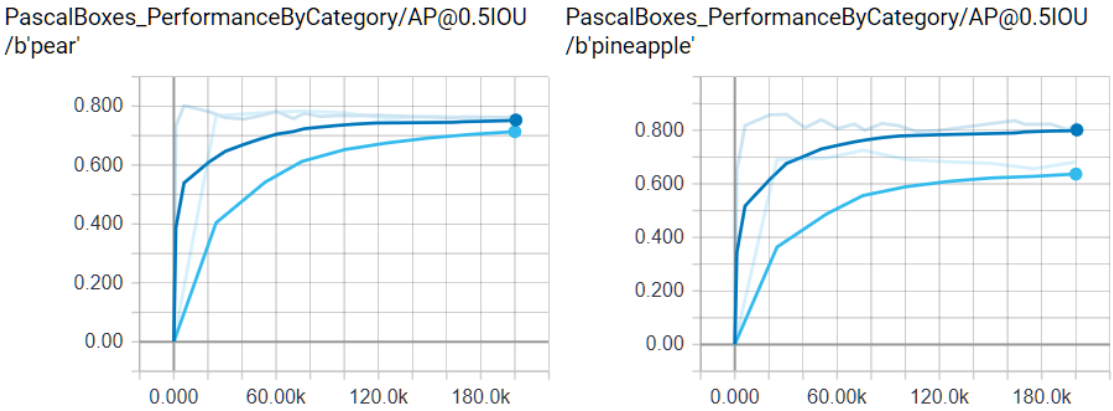


Figura 7.4. Comparación entre los valores PASCAL VOC AP@0.5 obtenidos para la detección de las peras, a la izquierda y para las piñas, a la derecha.

La evolución de la AP, en cada una de las clases de frutas, tiene un comportamiento similar al comparar los datos de evaluación de ambos modelos, tal y como se observa en las figuras 7.2, 7.3 y 7.4. Donde se ve claramente una superior AP del modelo *Faster* R-CNN ResNet-101 desde el comienzo del entrenamiento en todas las clases del estudio.

En la tabla 7.1. se pueden ver los resultados obtenidos, en porcentajes, en las detecciones positivas del modelo *Faster* R-CNN ResNet-101 durante el proceso de evaluación. Se observa una pobre capacidad para detectar limones, un 63%, y una buena detección de las piñas. En cuanto a la detección de plátanos y peras, tienen ambos una detección aceptable de un 78% y 77% respectivamente. Un poco peor se comporta el modelo al detectar naranjas con un 74% y manzanas, con un 75%

Manzanas	Plátanos	Limones	Naranjas	Peras	Piñas
75%	78%	63%	74%	77%	82%

Tabla 7.1 Porcentajes de detecciones positivas en el conjunto de imágenes de evaluación con el modelo *Faster* R-CNN ResNet-101

En la tabla 7.2 se ven los resultados obtenidos, en porcentajes, en las detecciones positivas del modelo *Faster* R-CNN Inception v2 durante el proceso de evaluación. Se observa un sorprendente bajo porcentaje de detección en el caso de las piñas, sólo un 68% frente al 82% que presentaba el modelo *Faster* R-CNN ResNet-101 para la misma clase. La clase con peor porcentaje de detecciones sigue siendo los limones, pero en este caso con un peor porcentaje de un 51%. En este caso, casi la mitad de las veces el detector no detecta la presencia de la clase en las imágenes. El modelo

tiene un porcentaje bajo también en la detección de las naranjas, con un 67%. Con respecto a las peras, plátanos y manzanas el porcentaje de detección del modelo ha sido similar con un 75%, 73% y 72% respectivamente.

Manzanas	Plátanos	Limones	Naranjas	Peras	Piñas
72%	73%	51%	67%	75%	68%

Tabla 7.2 Porcentajes de detecciones positivas en el conjunto de imágenes de evaluación con el modelo *Faster* R-CNN Inception v2

Se observa que el modelo *Faster* R-CNN ResNet-101 ha demostrado una mayor eficacia en la detección durante el proceso de evaluación en cada una de las clases de fruta y lógicamente una mejor mAP global. En base a estos resultados, éste es el modelo que se usará para la detección de las clases de frutas en las imágenes de producción y también para alimentar con los datos obtenidos en las detecciones, las clases detectadas y su recuento para cada día de estudio, al sistema de control de existencias planteado para las unidades de fruta en el frutero. En la figura 7.5. está representado la evolución de la métrica PASCAL VOC mAP@0.5IOU de ambos modelos durante el proceso de evaluación.

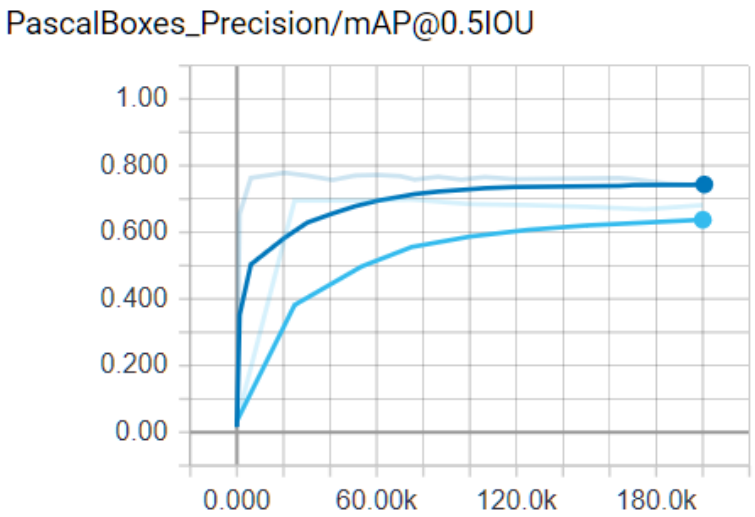


Figura 7.5. Comparación de la métrica PASCAL VOC mAP@0.5 para ambos modelos.

En ambos modelos la máxima eficacia en la detección se estabiliza sobre los 120000 pasos de entrenamiento, esto coincide con el estancamiento observado en la función de pérdida durante el proceso de entrenamiento de los modelos a partir de aproximadamente del mismo número de pasos de entrenamiento.

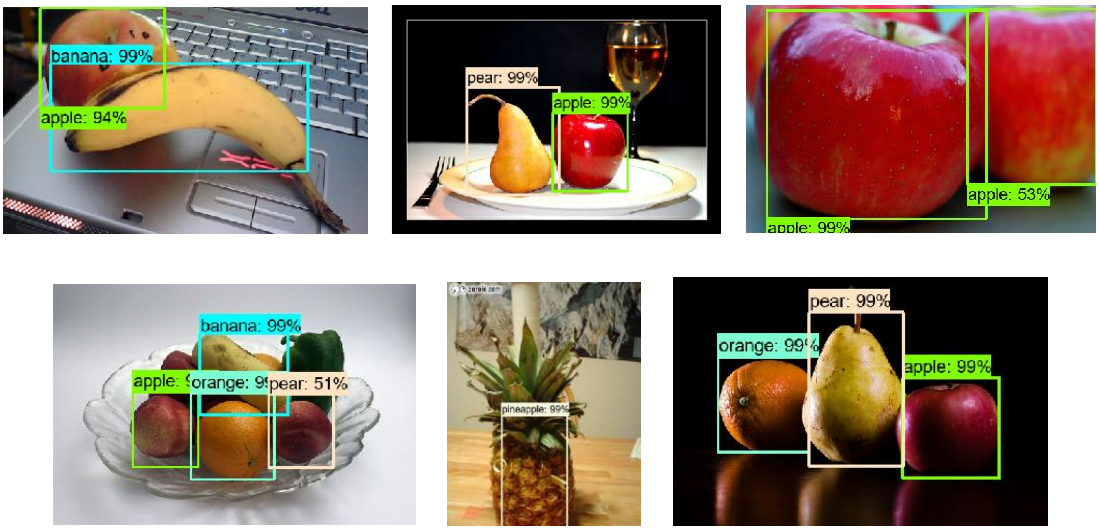


Figura 7.6. Selección de resultados del modelo
Faster R-CNN ResNet-101 con las imágenes de evaluación

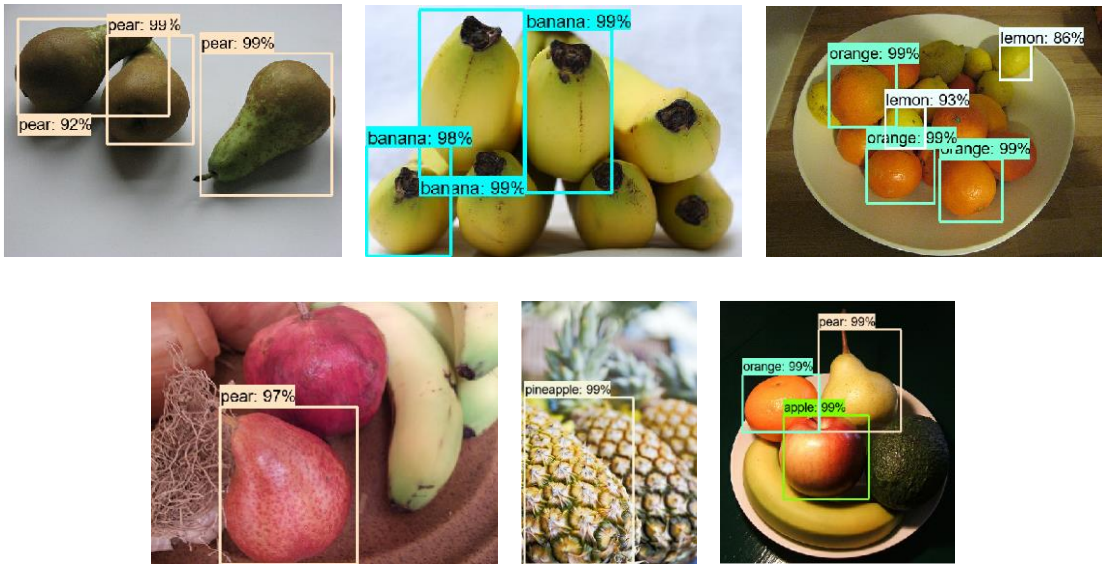


Figura 7.7. Selección de resultados con las imágenes
de evaluación del modelo *Faster* R-CNN Inception v2

7.2. Resultados con las imágenes de producción.

En la figura 7.8. se observa la detección de las 6 clases de fruta, en las 5 imágenes tomadas, del frutero el día 1 de estudio. Los resultados se han obtenido de la ejecución del modelo seleccionado *Faster* R-CNN ResNet-101.

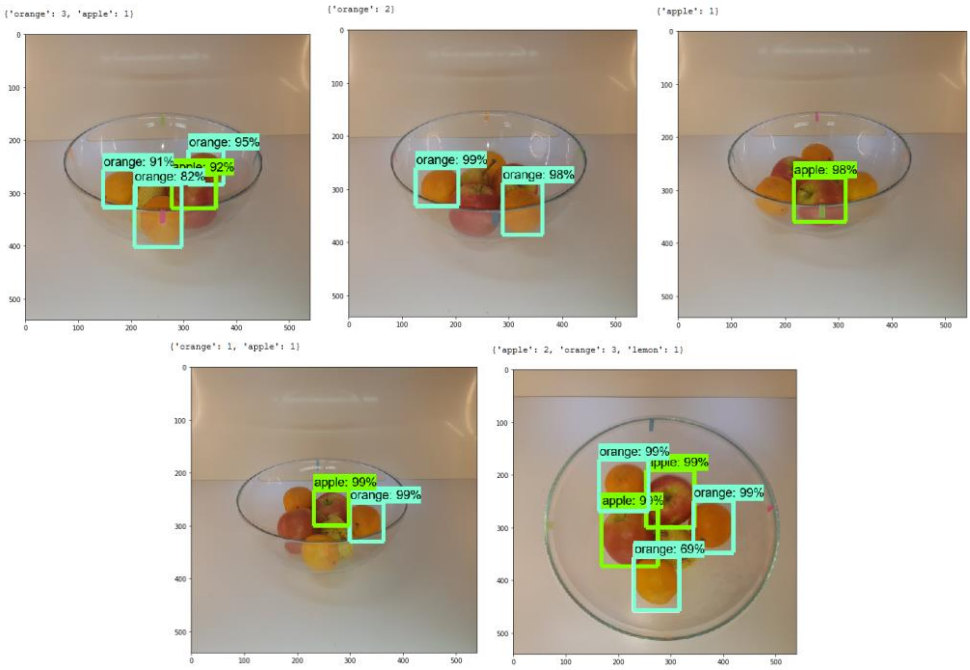


Figura 7.8. Fruta detectada el día 1 de estudio.

Resultados obtenidos el día 1 de estudio:

- Historico_detectado.csv, línea día 1:

"{'apple':2,'pear':0,'lemon':1,'orange':3,'banana':0,'pineapple':0,'Fecha': '2018-10-01'}"

- Historico_real.csv, línea día 1:

{'apple': 2,'pear': 1,'orange': 3,'lemon': 0,'banana': 0,'pineapple': 0,'Fecha': '2018-10-01}

Al comparar los resultados detectados contra los valores reales se observa que el modelo no ha detectado correctamente todas las clases. El sistema, gracias a la toma de múltiples imágenes para el recuento, ha minimizado el error y sólo ha cometido una equivocación con respecto a los valores reales.

No obstante, se observa que el detector ha cometido numerosos errores en cada una de las detecciones de frutas en el frutero desde diferentes ángulos. Este patrón se observa repetidamente durante los días de estudio, numerosos errores en la detección de cada una de las imágenes, pero el error se reduce por la toma de múltiples imágenes desde diferentes ángulos para el recuento final del día de estudio.

En la figura 7.9 se puede ver una selección de ejemplos en la detección de las seis clases de frutas estudiadas con las imágenes de producción. Las imágenes han sido seleccionadas de forma aleatoria con respecto al día de estudio y el ángulo de la imagen tomado.

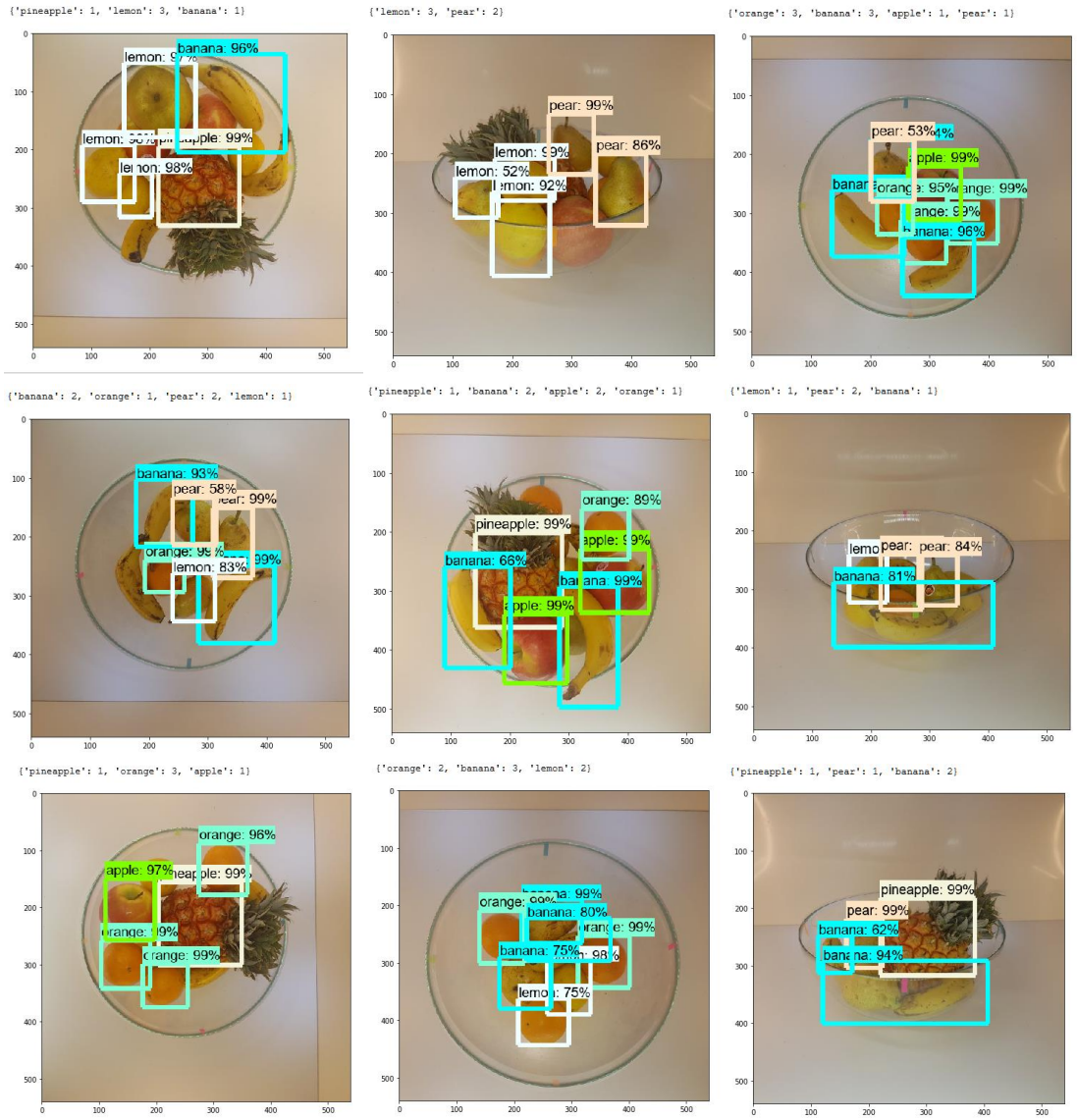


Figura 7.9. Selección de resultados del modelo
Faster R-CNN ResNet-101 con las imágenes de producción

7.3. Resultados del sistema de control de existencias

Los resultados obtenidos se analizarán por cada clase, dado que no sólo se va a evaluar al sistema de la identificación de cada clase sino también de la exactitud de su recuento. Para evaluar el sistema debemos saber también los datos reales diarios de cada una de las clases de fruta en el frutero y confrontaros con los detectados por el modelo. Para ello, se apunta diariamente el recuento real de las clases de frutas que se encuentran en el frutero en el fichero “historico_real.csv”, de la misma forma que se

genera el histórico del recuento de las clases de frutas detectadas y se guarda en el fichero “historico_detectado.csv”.

De esta manera se puede realizar una matriz de confusión para cada una de las clases de frutas y analizar cómo se ha comportado el modelo en cada una ellas. Con la matriz de confusión confrontando los valores de ambas series temporales de datos obtendremos los valores con los que podremos calcular las métricas de evaluación del sistema de control de existencias basado en la detección de las clases de frutas en el frutero. Los resultados obtenidos por el sistema de control de existencias con cada una de las clases tras 30 días de estudio se muestran en las siguientes tablas.

La tabla 7.3 muestra los valores de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos obtenidos de la tabla de confusión correspondiente a los recuentos posibles en cada día de estudio para la clase *apple*, donde se han confrontado los datos obtenidos por el sistema de control de existencias en el fichero ‘historico_detectado.csv’ y los datos de recuento reales del fichero ‘historico_real.csv’ para las detecciones y recuentos de las manzanas.

Se muestran también las métricas de evaluación de exhaustividad o *recall,* especificidad o *specificity,* exactitud o *precision,* precisión o *accuracy y* la medida *f1-score* totales para la clase, así como para cada uno de sus posibles recuentos en cada día de estudio. En el caso de las manzanas los recuentos en cada día de estudio pueden ser entre 0 y 3 manzanas detectadas en el frutero.

Número de unidades detectadas					
	0	**1**	**2**	**3**	**Total**
Valores de la matriz de confusión					
TP	6	6	3	2	17
TN	20	19	19	19	77
FP	2	0	2	9	13
FN	2	5	6	0	13
Métricas de evaluación					
recall	75 %	55 %	33 %	100 %	66 %
specificity	91 %	100 %	90 %	68 %	87 %
precision	75 %	100 %	60 %	18 %	63 %
accuracy	87 %	83 %	73 %	70 %	78 %
f1-score	75 %	71 %	43 %	31 %	55 %

Tabla 7.3 Resultados obtenidos de la tabla de confusión y métricas de evaluación del modelo para la clase *apple*

La columna con el fondo en gris de la tabla 7.3 muestra los valores para un posible recuento final de 1 manzana en un día de estudio, para un período de 30 días. Donde se observa que los verdaderos positivos son 6, es decir, las detecciones y recuentos de 1 manzana que coinciden con datos reales registrados paralelamente son 6. Se

observa también que los verdaderos negativos son 19, es decir, las detecciones y recuentos que no son de 1 manzana y esa información coincide con los datos reales registrados. Con respecto a los falsos positivos no obtenemos ninguno, es decir, que ninguno de los recuentos de 1 manzana en los valores reales se ha detectado como erróneo. Como falsos negativos tenemos un valor de 5, es decir, detecciones y recuentos de 1 manzana que no coinciden con los valores reales registrados. De la misma forma a continuación, en esta columna se muestran las métricas de evaluación calculadas para un posible recuento final de 1 manzana. Se observa una tasa de verdaderos positivos o *recall* de un 55%. Es decir, la probabilidad de que, dado un registro real de recuento de 1 unidad para las manzanas, el modelo lo acierte.

Como la tasa de verdaderos negativos o *specificity* obtenemos un 100% de probabilidad. Esto es la probabilidad de que, dado un registro real de recuento diferente a 1 unidad para las manzanas, el modelo lo detecte también como erróneo. En cuanto a la tasa de predicciones positivas o *precision*, se obtiene también un 100% de probabilidades, es decir, la probabilidad de que una detección y recuento positiva de 1 manzana coincida con esa información en los datos reales. Como porcentaje total de aciertos o *accuracy* en las detecciones y recuentos para un valor de 1 manzana, obtenemos un 83% de aciertos.

La última métrica mostrada es la de *f1-score* o media ponderada de las métricas *precision* y *recall*, en donde se obtiene un valor de un 71% de media en la precisión que tiene la evaluación del modelo. La tabla muestra también una columna “Total” donde están representados los sumatorios de TP, TN, FP y FN para todos los valores posibles de recuento de manzanas. En esta misma columna total se muestran las medias de las métricas de evaluación.

La descripción de los valores y métricas anteriores se pueden extrapolar al resto de columnas de la tabla 7.3 donde se representan el resto de los valores posibles para el recuento de manzanas, así como para las tablas 7.4, 7.5, 7.6, 7.7 y 7.8.

En la tabla 7.3 se observa que generalmente estas métricas de evaluación disminuyen a medida que aumenta el número de piezas de fruta a detectar a partir de una unidad. Lo que indica una pérdida de eficacia general del sistema a medida que el problema se complica. Un valor que puede representar correctamente la media de precisión general del sistema para la detección y recuento de manzanas es el valor total para la métrica *f1-score*, en donde se obtiene un 55%.

La información presentada en la tabla 7.4 es la misma que la presentada en la tabla 7.3 pero en esta ocasión en referencia a las detecciones y recuentos de las peras.

Número de unidades detectadas					
	0	**1**	**2**	**3**	**Total**
Valores de la matriz de confusión					
TP	7	4	1	0	12
TN	13	14	22	23	72
FP	4	1	6	7	18
FN	6	11	1	0	18
Métricas de evaluación					
recall	54 %	27 %	50 %	0 %	33 %
specificity	76 %	93 %	79 %	77 %	81 %
precision	64 %	80 %	14 %	0 %	39 %
accuracy	67 %	60 %	77 %	77 %	70 %
f1-score	58 %	40 %	22 %	0 %	30 %

Tabla 7.4 Resultados obtenidos de la tabla de confusión y métricas de evaluación del modelo para la clase *pear*

En la tabla 7.4 se observa un patrón similar, donde generalmente las métricas disminuyen a medida que aumenta el número de piezas de fruta a detectar a partir de una unidad. Como f1-score total para la detección de las peras obtenemos un 30%. Muy inferior al obtenido para la detección de las manzanas.

La información presentada en la tabla 7.5 es la misma que la presentada en la tabla 7.3 pero en esta ocasión en referencia a las detecciones y recuentos de los plátanos.

Número de unidades detectadas					
	0	**1**	**2**	**3**	**Total**
Valores de la matriz de confusión					
TP	3	4	2	1	10
TN	19	14	18	19	70
FP	4	1	7	8	20
FN	4	11	3	2	20
Métricas de evaluación					
recall	43 %	27 %	40 %	33 %	36 %
specificity	83 %	93 %	72 %	70 %	80 %
precision	43 %	80 %	22 %	11 %	39 %
accuracy	73 %	60 %	67 %	67 %	67 %
f1-score	43 %	40 %	29 %	17 %	32 %

Tabla 7.5 Resultados obtenidos de la tabla de confusión y métricas de evaluación del modelo para la clase *banana*

En este caso se obtiene como f1-score total de un 32%, muy similar al porcentaje general obtenido en la detección de peras y muy inferior al obtenido en la detección de manzanas. La diminución de las métricas es observada también en la tabla. La información presentada en la tabla 7.6 es la misma que la presentada en la tabla 7.3 pero en esta ocasión en referencia a las detecciones y recuentos de las naranjas.

Número de unidades detectadas					
	0	**1**	**2**	**3**	**Total**
Valores de la matriz de confusión					
TP	3	4	3	2	12
TN	20	17	16	19	72
FP	1	4	5	8	18
FN	6	5	6	1	18
Métricas de evaluación					
recall	33 %	44 %	33 %	67 %	44 %
specificity	95 %	81 %	76 %	70 %	81 %
precision	75 %	50 %	38 %	20 %	46 %
accuracy	77 %	70 %	63 %	70 %	70 %
f1-score	46 %	47 %	35 %	31 %	40 %

Tabla 7.6 Resultados obtenidos de la tabla de confusión
y métricas de evaluación del modelo para la clase *orange*

La disminución de las métricas es observada también en la tabla 7.6, donde se observa un *f1-score* total de un 40%, superior al registrado en la detección de peras y plátanos, pero inferior al registrado en la detección de las manzanas. La tabla 7.3 hace referencia a las detecciones y recuentos de los limones.

Número de unidades detectadas					
	0	**1**	**2**	**3**	**Total**
Valores de la matriz de confusión					
TP	5	2	1	6	14
TN	16	18	19	21	74
FP	9	3	2	2	16
FN	0	7	8	1	16
Métricas de evaluación					
recall	100 %	22 %	11 %	86 %	55 %
specificity	64 %	86 %	90 %	91 %	83 %
precision	36 %	40 %	33 %	75 %	46 %
accuracy	70 %	67 %	67 %	90 %	73 %
F1-score	53 %	29 %	17 %	80 %	44 %

Tabla 7.7 Resultados obtenidos de la tabla de confusión
y métricas de evaluación del modelo para la clase *lemon*

Nuevamente se vuelve a observar el patrón general en la disminución general de las métricas a medida que aumenta el número de unidades a detectar. En el caso de la detección de los limones la métrica de f1-score total es de un 44%.

La información presentada en la tabla 7.8 es la misma que la presentada en la tabla 7.3 pero en esta ocasión en referencia a las detecciones y recuentos de las piñas. En el caso de las piñas los recuentos en cada día de estudio pueden ser 1 piña detectada en el frutero o ninguna.

Número de unidades detectadas			
	0	**1**	**Total**
Valores de la matriz de confusión			
TP	15	12	27
TN	12	15	27
FP	1	2	3
FN	2	1	3
Métricas de evaluación			
recall	88 %	92 %	90 %
specificity	92 %	88 %	90 %
precision	94 %	86 %	90 %
accuracy	90 %	90 %	90 %
F1-score	91 %	89 %	90 %

Tabla 7.8 Resultados obtenidos de la tabla de confusión
y métricas de evaluación del modelo para la clase *pineapple*

La detección de las piñas es la única clase de las planteadas que obtiene buenos resultados con un alto porcentaje en las detecciones acertadas, donde se obtiene f1-score total de un 90%. Esto es debido, sin duda, a su fácil detección dado su tamaño y aspecto en relación con el resto de las clases de fruta y al trabajar sólo con una unidad de esta clase.

8. Discusión

8.1. Mejoras y modificaciones del proceso.

Un sistema de control de existencias de fruta, basado en la detección de objetos en imágenes por medio de CNN, como el que se ha planteado en el presente estudio necesita de unos modelos de detección robustos y una cantidad suficiente de imágenes que alimenten el aprendizaje de estos modelos. El presente estudio ha sido una aproximación al complejo problema planteado que, de forma modesta, con poco poder computacional y con pocas imágenes, ha conseguido resultados. Los resultados, eso sí, tienen un margen de error y de mejora considerable. A continuación, se plantean una serie de mejoras.

- El problema de un posible subajuste apreciado en los resultados del estudio, figura 7.1, hace que una mejora del estudio conlleve un aumento de la cantidad de imágenes relevantes por clase. El aumento de imágenes debe mantener el balanceo de clases y evitar las largas series de imágenes con características idénticas y solo representativas parcialmente de una clase. No obstante, la calidad de las imágenes para el entrenamiento es fundamental, si la cantidad de imágenes es muy grande pero su calidad general es mala, el modelo resultante será, con bastante seguridad, de baja calidad. Es recomendable realizar una selección y procesamiento previo exhaustivo de las imágenes de entrenamiento y evaluación. Es recomendable también mantener los datos de evaluación ocultos al entrenamiento, tal y como se ha hecho en el presente estudio.

- La transferencia de aprendizaje, explicada en el capítulo 5, a partir de un modelo más robusto. Esto conlleva una mayor capacidad de computo en la mayoría de los casos. El problema planteado de detección no persigue la rapidez en la detección sino su precisión, por lo que habrá que probar diferentes modelos dependiendo de la capacidad de computo con la que se pueda trabajar.

- Un factor adicional que potenciar es la idea planteada en el presente estudio de la captación de múltiples imágenes desde distintos ángulos, figura 7.8 y 7.9, para afinar al máximo el recuento de un conjunto de diferentes clases de fruta apiladas, parcialmente ocultas y colocadas en un recipiente. El concepto más trabajado y con algunos ajustes, podría dar resultados muy positivos. Por ejemplo, encontrar el número medio de imágenes a tomar que disminuya el error hasta un mínimo razonable o la búsqueda de la mejor elección de los ángulos en los que tomar dichas imágenes.

- La búsqueda de un modelo más robusto para la detección de objetos conlleva a su vez una búsqueda de sus hiperparámetros y parámetros que mejor resultados nos puedan ofrecer. Esta búsqueda es a menudo una investigación basada en la experiencia del investigador y de un proceso de prueba y error. Algunos de los hiperparámetros que puede ser interesante modificar, en búsqueda de un mejor rendimiento, son la generación de los anchors, la tasa de aprendizaje o el umbral IoU en la primera etapa del modelo. Con la modificación de las escalas y relaciones de

aspecto en la generación de los *anchors* se pueden buscar nuevos ajustes que sean más eficaces para conseguir el objetivo del estudio. Por ejemplo, si las fotos que tomamos de las frutas son muy cercanas quizás no nos interese una generación de *anchors* muy pequeños en relación con la imagen. Interesante puede resultar también realizar reducciones en la tasa de aprendizaje, en momentos predefinidos, durante el proceso de entrenamiento de manera manual. Por ejemplo, a los 180000 pasos de entrenamiento reducir la tasa de aprendizaje de 0.0002 a 0.00002. También se puede probar otro hiperparámetro, *learning rate decay*, el cual va reduciendo la tasa de aprendizaje a medida que el modelo se acerca a la solución. Se debe tener cuidado con el exceso de ajuste de estos valores ya que puede hacer que el proceso de entrenamiento se ralentice y aumente mucho su duración. El umbral IoU se podría reducir de 0.7 en la primera etapa de extracción de características con el fin de no ser tan estricto con los cuadros delimitadores considerados válidos en esta primera etapa.

- Con mayor capacidad de computo y el modelo adecuado, por ejemplo, Mask R-CNN, además de una base de imágenes previamente procesadas correctamente, sería interesante investigar en este problema la detección de fruta mediante máscaras y las técnicas de segmentación de instancias en lugar de mediante cuadros delimitadores. La segmentación de instancias consigue identificar los contornos de los objetos al nivel de un píxel, por lo que es muy exacta. Un factor a tener en cuenta este sentido es la similitud entre las formas de muchas frutas, sobre todo en algunas perspectivas, lo que puede dificultar en gran medida el obtener buenos resultados entre estas frutas con este tipo de técnicas.

8.2. Trabajos futuros

Los trabajos futuros de los que el presente estudio puede servir de base son muchos, el problema del recuento de objetos en imágenes o videos es un campo de investigación de vanguardia y complejo de resolver con éxito en muchos casos. El presente estudio, de la misma forma, puede servir como punto de partida para continuar con el estudio aquí planteado, llevándolo más allá. Para ello se podrían incorporar las mejoras planteadas en el anterior apartado, así como nuevas propuestas o puntos de vista. Se pueden plantear objetivos mucho más ambiciosos como puede ser el ampliar el número de clases de frutas a estudiar, ampliar el entorno hacia un entorno no controlado y donde las variables a tener en cuenta aumenten. Se puede extrapolar lo aprendido en el estudio para ejecutarlo con tipo de objetos y objetivos específicos, como puede ser la detección y recuento de piezas mediante reconocimiento de objetos en una cadena de montaje.

Otro de estos futuros proyectos podría ser el del recuento de fruta en árboles frutales de un capo de cultivo. La cámara con la que se captarían las múltiples imágenes sobre un árbol, para realizar el recuento de la fruta que contiene, estaría montada en un dron multiplicando así las posibilidades y velocidad del recuento. Con cada árbol el dron captaría un número imágenes a determinar. Con ellas se podría afinar un recuento de los valores obtenidos en las imágenes de forma similar a como se ha realizado en el presente estudio.

9. Conclusiones

El estudio realizado ha demostrado la ejecución de un sistema de control de existencias basado en la información que suministraba un modelo basado en redes neuronales convolucionales para la detección de objetos en imágenes. Los resultados obtenidos no han sido destacables, sin embargo, se ha podido comprobar el potencial de la transferencia de aprendizaje para ajustar un modelo con pocas imágenes y clases para una tarea en particular, como es la detección y recuento de distintas clases de fruta colocadas en un frutero.

El problema de detectar diferentes objetos apilados y ocultos o semiocultos en un entorno en tres dimensiones, contarlos y que los datos obtenidos no tengan un margen de error considerable es enorme. Un concepto que puede ayudar es el presentado en el estudio, donde se concluye que la estrategia de obtener múltiples imágenes desde diferentes ángulos para afinar un recuento de las diferentes clases de frutas colocadas en un frutero ayuda a reducir este margen de error en el recuento.

10. Bibliografía

1. Bargoti, S. y Underwood, J. (2016). *Deep fruit detection in orchards*. Recuperado de arXiv *preprint* arXiv:1610.03677v2. arxiv.org
2. Buduma, N. (2016). *Fundamentals of Deep Learning: Designing Next-Generation Artificial Intelligence Algorithms. O'Reilly Media Inc*.
3. Chattopadhyay, P., Vedantam, R., Selvaraju, R., Batra, D. y Parikh, D. (2016). *Counting everyday objects in everyday scenes. Georgia institute of technology*. En: CVPR 2016. arXiv *preprint* arXiv:1604.03505v3. arxiv.org
4. Chollet, F. (2017). *Deep learning with python. Manning Publications co*.
5. Chollet, F. (2017). Xception: *Deep Learning with Depthwise Separable Convolutions.* En: Honolulu, HI, USA. IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR), 1800-1807. arXiv *preprint* arXiv:1610.02357v3. arxiv.org
6. Chen, S.W., Shivakumar, S., Dcunha, S, Das, J., Okon,E., Qu,C., Taylor,C. y Kuma V. (2017). *Counting apples and oranges with deep learning: a data-driven approach.* En: IEEE *Robotics and automation Letters. (Volume: 2, Issue: 2, April 2017).* doi: 10.1109/LRA.2017.2651944
7. Dai, J., YI, I., He, K. y Jian, S. (2016). R-FCN: *Region-based fully convolutional networks. arXiv preprint* arXiv:1605.06409v2. arxiv.org
8. Dalal, N. y Triggs, B. (2005). *Histograms of oriented gradients for human detection. En: Proceedings of Conference on Computer Vision and Pattern Recognition.* Recuperado de https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
9. Freaud, Y. y Schapire, R.E. (1999). *A short introduction to boosting. en: journal of japanese society for artifical intelligence.* Recuperado de https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf
10. Girshick, R. (2015). *Fast* R-CNN. En: 2015 IEEE *International Conference on Computer Vision* (ICCV) 1440-1448. Santiago, Chile. Recuperado de https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf
11. Girshick, R., Donahue, J., Darrell, T. y Malik, J. (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation.* En: 2014 IEEE *Conference on Computer Vision and Pattern Recognition*. 580-587. arXiv *preprint* arXiv:1311.2524v5. arxiv.org
12. Goodfellow, I.; Bengio, Y. y Courville, A. (2016). *Deep learning book. MIT Press Book.* Recuperado de http://www.deeplearningbook.org
13. Gulli, A. y Pal, S. (2017). *Deep learning with keras. Birmingham, Packt publishing ltd.*

14. Keras *convolutional layers* (2018). [en línea] [fecha de consulta: 10 de octubre de 2018]. Recuperado de https://keras.io/layers/convolutional/

15. Keras *models* (2018). [en línea] [fecha de consulta: 16 de octubre de 2018] Recuperado de https://keras.io/applications/

16. Krizhevsky, A.; Sutskever, I. y Hinton, G.E. (2012). *Imagenet classification with deep convolutional neural networks.* En: *Neural Information Processing Systems.* 25. 10.1145/3065386.

17. He, K., Xiangyu Z., Shaoqing R. y Jian, S. (2016). *Deep residual learning for image recognition.* En: 2016 IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR) 770–778. Las Vegas, Estados Unidos. doi: 10.1109/CVPR.2016.90

18. He, K. Georgia G., Piotr D. y Ross G. (2017). *Mask* R-CNN. En: 2017 IEEE *International Conference on Computer Vision* (ICCV) 2980-2988. Venecia, Italia. doi: 10.1109/ICCV.2017.322

19. Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. y Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.* arXiv *preprint* arXiv:1704.04861v1. arxiv.org

20. Imagenet *project* (2018) [en línea] [fecha de consulta: 10 de octubre de 2018] Recuperado de http://image-net.org

21. Lempitsky, V. y Zisserman, A. (2010). *Learning to count objects in images*. En: *Annual Conference on Neural Information Processing Systems* (NIPS), 1324-332. Vancouver, British Columbia, Canada. Recuperado de http://papers.nips.cc/paper/4043-learning-to-count-objects-in-images.pdf

22. Opencv (2018). [en línea] [fecha de consulta: 15 de octubre de 2018] Recuperado de https://opencv.org/

23. Rahnemoonfar, M. y Sheppard, C. (2017). *Deep count: fruit counting based on deep simulated learning.* En: *Sensors* 17, 905. Recuperado de https://www.researchgate.net/publication/316945117_Deep_Count_Fruit_Counting_Based_on_Deep_Simulated_Learning

24. Redmon, J., Santosh D., Ross G. y Ali F. (2016). *You only look once: unified, real-time object detection*. En: 2016 IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR), 779-788. doi: 10.1109/CVPR.2016.91

25. Redmon, J. y Farhadi, A. (2017). YOLO9000: Better, *Faster*, Stronger. En: 2017 IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR) 6517-6525. Honolulu, Estados Unidos. doi: 10.1109/CVPR.2017.690

26. Redmon, J. y Farhadi, A. (2018). YOLOv3: *An incremental improvement.* arXiv preprint arXiv:1804.02767v1. arxiv.org

27. Ren, S, Kaiming, H., Ross G. y Jian, S. (2015). *Faster* R-CNN: *Towards real-time object detection with region proposal networks*. arXiv:1506.01497v3. arxiv.org

28. Reyes, A.K., Caicedo, J.C y Camargo, J.E. (2015). *Fine-tuning Deep Convolutional Networks for Plant Recognition*. doi: 10.1007/978-3-319-52277-7_57

29. Sa, I., Ge, Z., Dayoub, F., Upcroft, B., Perez, T. y Mccool, C. (2016). *Deepfruits: A fruit detection system using deep neural networks.* En: *Sensors.* 16. 1222. doi: 10.3390/s16081222

30. Sethy, P.K., Panda, S. y Bhoi, N. (2016). *On tree detection and counting of mature and immature fruit of carica papaya using image processing technique*. En: *International Journal of Computer Applications* (0975 –8887) Volume 156– No.8. Recuperado de https://pdfs.semanticscholar.org/5abc/814e97c98e409259a0fa3ec09829f66baa70.pdf

31. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. y Lecun, Y. (2013). *Overfeat: Integrated recognition, localization and detection using convolutional networks.* En: ILSVRC2013. arXiv *preprint* arXiv:1312.6229v4. arxiv.org

32. Simonyan, K. y zisserman, a. (2014.) *Very deep convolutional networks for large-scale image recognition.* En: ICLR2015. arXiv *preprint* arXiv:1409.1556v6. arxiv.org

33. Szegedy, C.,Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. y Rabinovich, *A. Going deeper with convolutions*. En: Proceedings of the IEEE conference on computer vision and pattern recognition, páginas 1-9, 2015.

34. Szegedy, C., Reed, S., Erhan, D. y Anguelov, D. (2014). *Scalable, High-Quality Object Detection.* En: ILSVRC 2014. arXiv *preprint* arXiv:1412.1441v3. arxiv.org

35. Szegedy, C., Vanhoucke, V., Ioffe, S., Shens, J. y Wojna, Z. (2016). *Rethinking the Inception Architecture for Computer Vision.* En: 2016 IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR), 2818-2826. Las Vegas, Estados Unidos. doi: 10.1109/CVPR.2016.308

36. TensorFlow (2018). [en línea] [Fecha de Consulta: 17 de octubre de 2018] Recuperado de https://www.tensorflow.org/

37. Tran, Dat (2017). *Raccoon Detector Dataset*. [en línea] [Fecha de Consulta: 20 de noviembre de 2018] Recuperado de https://github.com/datitran/raccoon_dataset

38. Viola, P. y Jones, M.J. (2001). *Robust Real-time object Detection. En: second international workshop on statistical and computational theories of vision–modeling, learning, computing, and sampling.* Vancouver, Canadá. Recuperado de http://www.hpl.hp.com/techreports/Compaq-DEC/CRL-2001-1.pdf

39. Liu, W., Anguelos, D., Erhan, D., Reed, S., FU, C-Y. y Berg, A.C. (2016). SSD: *Single Shot MultiBox Detector*. En: *Proceedings of the European Conference on Computer Vision* (ECCV). arXiv *preprint* arXiv:1512.02325v5. arxiv.org

40. Zeiler, M.D. y Fergus, R. (2014). *Visualizing and Understanding convolutional networks.* En: *European Conference on Computer Vision* (ECCV). 8689. 818-833. arXiv *preprint* arXiv:1311.2901v3. arxiv.org

41. Zhang, J., Ma, S., Sameki, M. Sclaroff, S., Betke, M.,Lin, Z.,Shen, Z., Price, B. y Mech, R. (2016). *Salient object subitizing*. En: IEEE *Conference on Computer Vision and Pattern Recognition* (CVPR), 4045-4054. Boston, Estados Unidos. doi: 10.1109/CVPR.2015.7299031

Anexo 1

Fichero apple_001.xml, en formato VOC Pascal, y anotación generada por LabelImg para la imagen apple_0001.jpg con. En el fichero se ha definido un *bounding box* en la imagen que indica que contiene una manzana o un objeto de la clase *apple*.

```
<annotation>
	<folder>train</folder>
	<filename>apple_0001.jpg</filename>
	<path>C:\FRUIT\train\apple_0001.jpg</path>
	<source>
		<database>Unknown</database>
	</source>
	<size>
		<width>500</width>
		<height>375</height>
		<depth>3</depth>

	</size>
	<segmented>0</segmented>
	<object>
		<name>apple</name>
		<pose>Unspecified</pose>
		<truncated>0</truncated>
		<difficult>0</difficult>
		<bndbox>
			<xmin>79</xmin>
			<ymin>25</ymin>
			<xmax>408</xmax>
			<ymax>364</ymax>
		</bndbox>
	</object>
</annotation>
```

Anexo 2

Fichero labelmap.pbtxt conteniendo las seis clases de fruta y sus identificadores.

```
item {
  id: 1
  name: 'apple'
}
item {
  id: 2
  name: 'banana'
}
item {
  id: 3
  name: 'orange'
}
item {
  id: 4
  name: 'lemon'
}
item {
  id: 5
  name: 'pineapple'
}
item {
  id: 6
  name: 'pear'
}
```

Anexo 3

Líneas de código añadido a “visualization_utils.py”

```
544 def contador(className,dict):
545     if str(className) in dict:
546         dict[str(className)] = dict[str(className)] +1
547     else:
548         dict[str(className)] = 1
```

…

Dentro de la función “visualize_boxes_and_labels_on_image_array”

…

```
643         contador(class_name,dict)
```

…

```
685 print(dict)
686 with open("../temporal.csv", "a") as temporalcsv:
687   temporal_diario = csv.writer(temporalcsv, delimiter=",", lineterminator="\n")
688   temporal_diario.writerow([dict])
```

Anexo 4

Fichero de configuración "*Faster*_rcnn_inception_v2_coco.config"

```
model {
  Faster_rcnn {
    num_classes: 6
    image_resizer {
keep_aspect_ratio_resizer {
        min_dimension: 200
        max_dimension: 650
      }
    }
    feature_extractor {
      type: 'Faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    initial_crop_size: 14
    maxpool_kernel_size: 2
```

```
    maxpool_stride: 2
    second_stage_box_predictor {
      mask_rcnn_box_predictor {
        use_dropout: false
        dropout_keep_probability: 1.0
        fc_hyperparams {
          op: FC
          regularizer {
            l2_regularizer {
              weight: 0.0
            }
          }
          initializer {
            variance_scaling_initializer {
              factor: 1.0
              uniform: true
              mode: FAN_AVG
            }
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}
train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 250000
            learning_rate: .00002
          }
          schedule {
            step: 500000
```

```
          learning_rate: .000002
        }
       }
     }
     momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint:
"object_detection/Faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  num_steps: 200000
  data_augmentation_options {
    random_horizontal_flip {
    }
    random_vertical_flip {
    }
    random_rotation90 {
    }
  }
}
train_input_reader: {
  tf_record_input_reader {
    input_path: "object_detection/train.record"
  }
  label_map_path: "object_detection/training/labelmap.pbtxt"
eval_config: {
  # Numero de ejemplos que queremos que se evalúen.
    num_examples: 226
  #Número de imágenes a mostrar en tensorboard
    num_visualizations: 10
  # Numero de evaluaciones a ejecutar.
     max_evals: 10
  # Conjunto de métricas de detección Pascal VOC
  metrics_set: "pascal_voc_detection_metrics”
}
eval_input_reader: {
  # Ruta donde se encuentra el fichero eval.record
  tf_record_input_reader{input_path: " object_detection/eval.record"}
# Ruta donde se encuentra el fichero labelmap.pbtxt
  label_map_path: " object_detection/training/labelmap.pbtxt"
  # Muestra aleatoria entre las imágenes de evaluación
  shuffle: false
    }
```

Anexo 5

Fichero de configuración "*Faster*_rcnn_resnet_101_coco.config"

```
model {
  Faster_rcnn {
    num_classes: 6
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 200
        max_dimension: 650
      }
    }
    feature_extractor {
      type: 'Faster_rcnn_ResNet-101'
      first_stage_features_stride: 16
    }
    first_stage_anchor_generator {
      grid_anchor_generator {
        scales: [0.25, 0.5, 1.0, 2.0]
        aspect_ratios: [0.5, 1.0, 2.0]
        height_stride: 16
        width_stride: 16
      }
    }
    first_stage_box_predictor_conv_hyperparams {
      op: CONV
      regularizer {
        l2_regularizer {
          weight: 0.0
        }
      }
      initializer {
        truncated_normal_initializer {
          stddev: 0.01
        }
      }
    }
    first_stage_nms_score_threshold: 0.0
    first_stage_nms_iou_threshold: 0.7
    first_stage_max_proposals: 300
    first_stage_localization_loss_weight: 2.0
    first_stage_objectness_loss_weight: 1.0
    initial_crop_size: 14
    maxpool_kernel_size: 2
    maxpool_stride: 2
```

```
    second_stage_box_predictor {
      mask_rcnn_box_predictor {
        use_dropout: false
        dropout_keep_probability: 1.0
        fc_hyperparams {
          op: FC
          regularizer {
            l2_regularizer {
              weight: 0.0
            }
          }
          initializer {
            variance_scaling_initializer {
              factor: 1.0
              uniform: true
              mode: FAN_AVG
            }
          }
        }
      }
    }
    second_stage_post_processing {
      batch_non_max_suppression {
        score_threshold: 0.0
        iou_threshold: 0.6
        max_detections_per_class: 100
        max_total_detections: 300
      }
      score_converter: SOFTMAX
    }
    second_stage_localization_loss_weight: 2.0
    second_stage_classification_loss_weight: 1.0
  }
}
train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0003
          schedule {
            step: 250000
            learning_rate: .00003
          }
          schedule {
            step: 500000
            learning_rate: .000003
```

```
        }
       }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  gradient_clipping_by_norm: 10.0
  fine_tune_checkpoint:" object_detection/Faster_rcnn_ResNet-
101_coco_2018_01_28/model.ckpt"
  from_detection_checkpoint: true
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
}
train_input_reader: {
  tf_record_input_reader {
    input_path: " object_detection/train.record"
  }
  label_map_path: " object_detection/training_101/labelmap.pbtxt"
}
eval_config: {
  num_examples: 226
  max_evals: 10
  metrics_set: "pascal_voc_detection_metrics"
}
eval_input_reader: {
  tf_record_input_reader {
    input_path: " object_detection/eval.record"
  }
  label_map_path: " object_detection/training_101/labelmap.pbtxt"
  shuffle: false
  num_readers: 10
}
```

Anexo 6

Código para la ejecución de la rutina diaria del sistema de control de existencias y Ejemplo de salida para el día 30 de estudio.

```
import ast
import pandas as pd
import numpy as np
import datetime as dt
pd.set_option('precision', 0)
#Leemos el fichero temporal.csv con la información de las detecciones del dia
df = pd.read_csv('../temporal.csv')
tmp_df = pd.DataFrame([ast.literal_eval(i) for i in df.data.values])
tmp_df= tmp_df.fillna(0)
# Mostramos el recuento de las diferentes clases en las diferentes capturas realizadas sobre el frutero
tmp_df
```

	apple	orange	pineapple
0	0	1	1
1	0	0	0
2	1	1	1
3	0	1	1
4	1	3	1

```
# Obtenemos el valor maximo obtenido para cada una de las clases de frutas
tmp_df = tmp_df.max(axis=0)
```

```
#Los recuentos maximos para cada clase de fruta son:
tmp_df
```

```
apple        1
orange       3
pineapple    1
dtype: float64
```

```
#Inicializamos con valores a cero un nuevo diccionario para una nueva linea de recuento
recuento ={'apple':0,'pear':0,'lemon':0,'orange':0,'banana':0,'pineapple':0}
```

```
#Comprobamos primero que existe la clase de fruta en el recuento.
#Una vez comprobado, se añade al recuento final diario para guardarlo en el histórico.
    if 'apple' in tmp_df:
         recuento['apple']+=tmp_df.apple
    if 'pear' in tmp_df:
        recuento['pear']+=tmp_df.pear
    if 'lemon' in tmp_df:
        recuento['lemon']+=tmp_df.lemon
    if 'orange' in tmp_df:
        recuento['orange']+=tmp_df.orange
    if 'banana' in tmp_df:
        recuento['banana']+=tmp_df.banana
    if 'pineapple' in tmp_df:
        recuento['pineapple']+=tmp_df.pineapple
```

```
#Del ultimo recuento, los maximos valores por cada clase:
print(recuento)
```

```
{'apple': 1.0, 'pear': 0, 'lemon': 0, 'orange': 3.0, 'banana': 0, 'pineapple': 1.0}
```

```
#Si debido a un error se han detectado un número mayor al número de recuento máximo por clase.
#El sistema supone que, en ese caso, existe en el frutero el recuento máximo para esa clase.
    if recuento['apple']>3:
         recuento['apple']=3
    if recuento['pear']>3:
        recuento['pear']
    if recuento['lemon']>3:
        recuento['lemon']=3
    if recuento['orange']>3:
        recuento['orange']=3
    if recuento['banana']>3:
        recuento['banana']=3
    if recuento['pineapple']>1:
        recuento['pineapple']=1
```

```
#Se añade la fecha actual al recuento final y guardamos los datos en una nueva linea del fichero histórico.
recuento['Fecha'] = datetime.now().strftime("%Y-%m-%d")
print(recuento)
with open("../historico_detectado.csv", "a") as pedidocsv:
    nueva_captura = csv.writer(pedidocsv, delimiter=",", lineterminator="\n")
    nueva_captura.writerow([recuento])
```

```
{'apple': 1.0, 'pear': 0, 'lemon': 0, 'orange': 3.0, 'banana': 0, 'pineapple': 1.0, 'Fecha': '2018-10-30'}
```

```
pd.set_option('precision', 0)
#Leemos el fichero historico.csv
df = pd.read_csv('../historico_detectado.csv')
hist_df = pd.DataFrame([ast.literal_eval(i) for i in df.data.values])
hist_df= hist_df.fillna(0)
hist_df.head()
```

	Fecha	apple	banana	lemon	orange	pear	pineapple
0	2018-10-01	2	0	2	3	0	0
1	2018-10-02	0	1	2	2	1	0
2	2018-10-03	2	2	2	2	0	0
3	2018-10-04	2	1	3	1	0	0
4	2018-10-05	3	1	2	1	1	0

```
# Convertimos la fecha para que haga de indice
hist_df.Fecha = pd.to_datetime(hist_df.Fecha)
hist_df.set_index('Fecha', inplace=True)
```

```
#Diferencias entre los valores de recuento de un dia y el anterior por clases. Ultimos valores obtenidos.
hist_df.diff().tail(3)
```

Fecha	apple	banana	lemon	orange	pear	pineapple
2018-10-28	-2	-1	1	-1	2	-1
2018-10-29	0	0	0	-1	-2	0
2018-10-30	1	-1	-1	3	0	1

```
# Media de los valores negativos entre las diferencias de un dia y el anterior.
# Lo que significa consumo medio diario de una clase de fruta. Se cambian los valores a positivos.
dif_df =hist_df.diff()
consumo_medio= dif_df[dif_df < 0].mean()*-1
consumo_medio
```

```
apple        2
banana       1
lemon        1
orange       1
pear         1
pineapple    1
dtype: float64
```

```
#Las capturas y recuentos de frutas de hoy y de ayer
ayer_hoy =hist_df.tail(2)
print(ayer_hoy)
```

```
            apple  banana  lemon  orange  pear  pineapple
Fecha
2018-10-29      0       1      1       0     0          0
2018-10-30      1       0      0       3     0          1
```

```
# Diferencia entre la captura y recuento de frutas de hoy y el de ayer
dif_ayer_hoy = ayer_hoy.diff().tail(1).mean()
#De ayer a hoy se han consumido (valores negativos)
#De ayer a hoy se han añadido (valores positivos)
dif_ayer_hoy
```

```
apple        1
banana      -1
lemon       -1
orange       3
pear         0
pineapple    1
dtype: float64
```

```
# Se lee el fichero de pedidos
pf= pd.read_csv('../pedidos.csv')
ped_df = pd.DataFrame([ast.literal_eval(i) for i in pf.data.values])
ped_df= ped_df.fillna(0)
# Ultima linea de pedido del fichero
ult_linea_pedido_ayer= ped_df.tail(1)
print('El pedido de ayer fue...')
ult_linea_pedido_ayer
```

```
El pedido de ayer fue...
```

	Fecha	apple	banana	lemon	orange	pear	pineapple
28	2018-10-29	3	0	0	0	0	1

```
dif_mas_ult_pedido = dif_ayer_hoy + ult_linea_pedido_ayer
dif_mas_ult_pedido
```

	Fecha	apple	banana	lemon	orange	pear	pineapple
28	NaN	4	-1	-1	3		2

```
#Se compara la diferencia entre el consumo medio y la diferenecia de valores entre hoy y ayer + pedido de ayer
i=0
for i in range(0,6):
    if consumo_medio[i] > dif_mas_ult_pedido.values[0,i+1]:
        print("Clase de Fruta: " + dif_mas_ult_pedido.columns.values[i+1])
        print("Media de consumo: "+ str(round(consumo_medio[i],2)))
        print("El ultimo pedido contabilizado: "+ str(ult_linea_pedido_ayer.values[0,i+1]))
        print("Diferencia entre el recuento de ayer y hoy más el pedido: "+ str(dif_mas_ult_pedido.values[0,i+1]))
        print("La media de consumo es mayor a la diferencia entre el recuento de ayer y hoy más el pedido")
        print("Probabilidad de que existan "+dif_mas_ult_pedido.columns.values[i+1]+" no detectadas en el frutero"
        print("----------------------")
        i=i+1
    else:
        print("Clase de Fruta: " + dif_mas_ult_pedido.columns.values[i+1])
        print("Media de consumo: "+ str(round(consumo_medio[i],2)))
        print("El ultimo pedido contabilizado: "+ str(ult_linea_pedido_ayer.values[0,i+1]))
        print("Diferencia entre el recuento de ayer y hoy más el pedido: "+ str(dif_mas_ult_pedido.values[0,i+1]))
        print("La media de consumo es menor a la diferencia entre el recuento de ayer y hoy más el pedido")
        print("Los datos parecen estar bien")
        print("----------------------")
        i=i+1
```

```
Clase de Fruta: apple
Media de consumo: 1.55
El ultimo pedido contabilizado: 3
Diferencia entre el recuento de ayer y hoy más el pedido: 4.0
La media de consumo es menor a la diferencia entre el recuento de ayer y hoy más el pedido
Los datos parecen estar bien
----------------------
Clase de Fruta: banana
Media de consumo: 1.44
El ultimo pedido contabilizado: 0
Diferencia entre el recuento de ayer y hoy más el pedido: -1.0
La media de consumo es mayor a la diferencia entre el recuento de ayer y hoy más el pedido
Probabilidad de que existan banana no detectadas en el frutero
----------------------
Clase de Fruta: lemon
Media de consumo: 1.42
El ultimo pedido contabilizado: 0
Diferencia entre el recuento de ayer y hoy más el pedido: -1.0
La media de consumo es mayor a la diferencia entre el recuento de ayer y hoy más el pedido
Probabilidad de que existan lemon no detectadas en el frutero
----------------------
Clase de Fruta: orange
Media de consumo: 1.2
El ultimo pedido contabilizado: 0
Diferencia entre el recuento de ayer y hoy más el pedido: 3.0
La media de consumo es menor a la diferencia entre el recuento de ayer y hoy más el pedido
Los datos parecen estar bien
----------------------
Clase de Fruta: pear
Media de consumo: 1.25
El ultimo pedido contabilizado: 0
Diferencia entre el recuento de ayer y hoy más el pedido: 0.0
La media de consumo es mayor a la diferencia entre el recuento de ayer y hoy más el pedido
Probabilidad de que existan pear no detectadas en el frutero
----------------------
Clase de Fruta: pineapple
Media de consumo: 1.0
El ultimo pedido contabilizado: 1
Diferencia entre el recuento de ayer y hoy más el pedido: 2.0
La media de consumo es menor a la diferencia entre el recuento de ayer y hoy más el pedido
Los datos parecen estar bien
----------------------
```

```
#Inicializamos con valores a cero un nuevo diccionario para una nueva linea de pedido si hiciera falta.
pedidos ={'apple':0,'pear':0,'lemon':0,'orange':0,'banana':0,'pineapple':0}
```

```
last=hist_df.tail(1)
print(last)
```

```
            apple  banana  lemon  orange  pear  pineapple
Fecha
2018-10-30      1       0      0       3     0          1
```

```
i=0
for i in range(0,6):
    if last.values[0,i]==0 and last.columns.values[i] == 'pineapple':
        print("No queda: "+last.columns.values[i]+". Añadir a linea de pedido 1 unidad:")
        pedidos[last.columns.values[i]]+=1
    elif last.values[0,i]==0 and last.columns.values[i] != 'pineapple':
        print("No queda: "+last.columns.values[i]+". Añadir a linea de pedido 3 unidades:")
        pedidos[last.columns.values[i]]+=3
```

```
No queda: banana. Añadir a linea de pedido 3 unidades:
No queda: lemon. Añadir a linea de pedido 3 unidades:
No queda: pear. Añadir a linea de pedido 3 unidades:
```

```
print(pedidos)
```

```
{'apple': 0, 'pear': 3, 'lemon': 3, 'orange': 0, 'banana': 3, 'pineapple': 0}
```

```
#Guardamos la nueva linea de pedido en el fichero "pedidos.csv"
pedidos['Fecha'] = datetime.now().strftime("%Y-%m-%d")
print(pedidos)
with open("../pedidos.csv", "a") as pedidocsv:
    linea_pedido = csv.writer(pedidocsv, delimiter=",", lineterminator="\n")
    linea_pedido.writerow([pedidos])
```

```
{'apple': 0, 'pear': 3, 'lemon': 3, 'orange': 0, 'banana': 3, 'pineapple': 0, 'Fecha': '2018-10-30'}
```