



Escuela
Politécnica
Superior

Aplicaciones para Nevera Inteligente basadas en Deep Learning



Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:

Lamberto Ruiz Martínez

Tutor/es:

Miguel Ángel Cazorla Quevedo

Francisco Gómez Donoso

Julio 2023



Universitat d'Alacant
Universidad de Alicante

Aplicaciones para Nevera Inteligente basadas en Deep Learning

Autor

Lamberto Ruiz Martínez

Tutor/es

Miguel Ángel Cazorla Quevedo

Ciencia de la Computación e Inteligencia Artificial

Francisco Gómez Donoso

Ciencia de la Computación e Inteligencia Artificial



Grado en Ingeniería Informática



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, Julio 2023

Preámbulo

Este proyecto trata de resolver un problema bastante común cuando no tenemos mucha comida en nuestra nevera y tampoco podemos ir al supermercado más cercano a comprar. La aplicación consistirá en realizar una foto de nuestra nevera, reconocer los alimentos que hay en ella y aportarnos una lista de posibles recetas que podemos hacer.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo de mis dos tutores. Sus rápidas respuestas a mis dudas y problemas han sido clave para el correcto desarrollo del proyecto, así como su disposición de aceptar mi propuesta de proyecto de final de grado la cual nos agradó a todos.

Me gustaría también agradecer a todos mis compañeros que me han apoyado a lo largo de todos estos años, siendo posible que finalice el grado con este último trabajo final.

Y por último pero no menos importante, quería agradecer a mi madre por su apoyo incondicional, sobre todo estos últimos meses, gracias al cual he podido estar tranquilo y ser positivo.

A mi padre, que estará viendo desde algún lugar como termino esta etapa.

He fallado una y otra vez en mi vida, por eso he conseguido el éxito

Michael Jordan.

Índice general

1	Introducción	1
2	Objetivos	3
2.1	Objetivo general	3
2.2	Objetivos Específicos	3
3	Estado del Arte	5
4	Marco Teórico	9
4.1	El entrenamiento del modelo	9
4.1.1	Análisis de nuestras neuronas	9
4.1.2	Redes neuronales	12
4.1.3	Entrenamiento de una red neuronal	14
4.1.3.1	Redes supervisadas	15
4.1.3.2	Redes NO supervisadas	15
4.2	Funcionamiento de YOLOv5	17
4.2.1	Versiones de YOLO	17
4.3	Entrenamiento de una red neuronal convolucional (CNN)	18
5	Metodología y Tecnologías	21
5.1	YOLOv5	21
5.2	Datasets de entrenamiento	21
5.3	API de ChatGPT	21
5.4	Flask	22
5.5	Sistema Operativo y otras herramientas	22
6	Desarrollo	23
6.1	YOLOv5	23
6.1.1	Preparación del dataset	23
6.1.2	Pruebas de entrenamiento	26
6.2	API de ChatGPT	28
6.3	Aplicación web Flask	30
7	Resultados	33
7.1	Resultado de entrenamientos YOLOv5	33
7.2	Aplicación Web	41
8	Conclusiones	43
Bibliografía		45

Índice de figuras

3.1	Frigorífico inteligente de Samsung <i>Family Hub</i> RS6HA8880S9	5
3.2	Frigorífico inteligente de Bosch con <i>Home Connect Ready</i>	6
3.3	Frigorífico inteligente LG Smart Instaview	7
4.1	Esquema básico de una neurona del cerebro humano	9
4.2	Possible modelo de caja negra de una neurona humana	10
4.3	Esquema del Perceptrón	11
4.4	Ecuación de salida de un perceptrón	12
4.5	Tipos de redes neuronales	12
4.6	Red neuronal multicapa	13
4.7	Ejemplo práctico de una red neuronal convolucional (CNN)	13
4.8	Ejemplo de entrenamiento y detección de una señal de tráfico	14
4.9	Tipos de aprendizaje de una red neuronal	16
4.10	Benchmark de YOLOv5	17
6.1	Estructura de desarrollo	23
6.2	Ejemplo de una imagen ya procesada por YOLOv5	27
6.3	Esquema básico para un usuario de la aplicación	30
7.1	Ejemplo de los resultados de entrenamiento de YOLOv5	33
7.2	Modelos de YOLOv5	34
7.3	Resultados con modelo YOLOv5s y diferentes épocas	34
7.4	Resultados con modelo YOLOv5m y diferentes épocas	35
7.5	Resultados con modelo YOLOv5l y diferentes épocas	35
7.6	Ejemplo de imagen de nevera	36
7.7	Resultado de la imagen con el modelo "YOLOv5l" con 100 épocas.	37
7.8	Resultado de la imagen con el modelo "YOLOv5lm" con 100 épocas.	37
7.9	Resultado de la imagen con el modelo "YOLOv5s" con 100 épocas.	37
7.10	Resultado de la imagen con el modelo "YOLOv5l" con 150 épocas.	38
7.11	Resultado de la imagen con el modelo "YOLOv5m" con 150 épocas.	38
7.12	Resultado de la imagen con el modelo "YOLOv5s" con 150 épocas.	38
7.13	Resultado de la imagen con el modelo "YOLOv5s" con 500 épocas.	39
7.14	Página de inicio de la aplicación	41
7.15	Página para subir una imagen y procesarla	41
7.16	Página de resultados del reconocimiento de alimentos.	42
7.17	Página de X posibles recetas obtenidas.	42

Índice de cuadros

6.1 Clases del dataset global respecto a cada uno	26
7.1 Observación de resultados de cada modelo sobre una imagen	40

1 Introducción

El reconocimiento de imágenes es una de las áreas más emocionantes y en constante evolución en el campo de la Inteligencia Artificial (IA). Con la creciente cantidad de datos visuales disponibles en la actualidad, la capacidad de las máquinas para procesar y entender imágenes se ha vuelto cada vez más importante en nuestra vida cotidiana. Desde la búsqueda de imágenes en línea hasta la detección de objetos en la seguridad del aeropuerto, el reconocimiento de imágenes tiene una amplia gama de aplicaciones en una variedad de industrias.

En sus inicios, el reconocimiento de imágenes era una tecnología de alta complejidad tanto teórica como técnica, lo que limitaba su acceso a un número reducido de expertos. No obstante, gracias al surgimiento de herramientas como Tensorflow o PyTorch, entre otras, la situación ha cambiado significativamente. Estas herramientas proporcionan interfaces intuitivas y sencillas que permiten a cualquier persona con conocimientos básicos de programación diseñar y entrenar sus propios modelos. En lugar de tener que enfrentarse a complejos algoritmos y ajustes técnicos, los usuarios solo tienen que elegir los parámetros adecuados para entrenar sus modelos con éxito. De esta forma, el reconocimiento de imágenes se ha democratizado, abriendo nuevas posibilidades a investigadores y entusiastas en distintas áreas del conocimiento.

Uno de los múltiples usos podría ser el de las neveras inteligentes. Estas neveras reconocen qué alimentos hay en su interior para poder ayudarte a saber qué hace falta comprar, qué alimentos van a caducar próximamente, qué recetas puedes realizar, entre otras muchas aplicaciones y funciones posibles.

En este proyecto abordaremos el reconocimiento de imágenes de neveras, en las cuales habrá alimentos que la aplicación tendrá que reconocer para poder así saber qué podemos cocinar o darnos algunas ideas con lo que tengamos en nuestra nevera.

En primer lugar, se hará un análisis del estado del arte para así saber qué otras opciones hay disponibles en el mercado y qué otros proyectos son similares a este.

En segundo lugar, se abordará qué herramientas y qué metodologías hemos aplicado para que la aplicación haga su cometido, aportar al usuario una lista de recetas con los alimentos que tengas en la nevera.

En tercer lugar, desarrollaremos en profundidad cada parte de la aplicación, la cual la he dividido en tres grandes bloques: reconocimiento de alimentos, extracción de recetas y el propio sistema de aplicación el cual unifica todo.

En cuarto y último lugar, se realizará un estudio de los resultados obtenidos, así como de las pruebas realizadas, observando cuál ha funcionado mejor y cuál no ha funcionado como se esperaba.

2 Objetivos

En este apartado trataremos de dejar claro cuáles son los objetivos principales y específicos del proyecto.

2.1 Objetivo general

El objetivo del trabajo se centra en el reconocimiento de objetos que hay dentro de una nevera tras hacer una fotografía de la misma. Una vez se reconocen los objetos, la aplicación te indicará qué recetas puedes realizar con los alimentos previamente reconocidos. Todo esto se unificará en una aplicación en la cual el usuario podrá realizar o subir una imagen desde cualquier dispositivo, observar qué alimentos o ingredientes se han detectado, obtener de esa lista el número de recetas que el usuario desee (5, 10 o 20) y, por último, observar la lista de recetas y elegir la que desee cocinar.

2.2 Objetivos Específicos

El Trabajo de Fin de Grado se centrará en estos cuatro puntos básicos:

- Profundizar en el reconocimiento de imágenes mediante algoritmos de detección de objetos de visión artificial basados en redes neuronales convolucionales.
- Aprender a preparar un dataset de imágenes de entrenamiento para que nuestra aplicación aprenda a reconocer objetos nuevos.
- Utilizar modelos de lenguaje de IA como *ChatGPT* (s.f.) para darnos una respuesta sobre las recetas que podemos realizar una vez tengamos la lista de alimentos detectados en la nevera.
- Unificar todas las tecnologías implementadas en una única aplicación la cual sea sencilla, intuitiva, posible de acceder desde varios dispositivos y cómoda para el usuario final.

3 Estado del Arte

En el mercado existen muchas opciones y estilos de neveras inteligentes que se asemejan en parte a este proyecto. No obstante, la finalidad de estas neveras se centra en observar los alimentos que hay dentro de la nevera o intentar automatizar el proceso de la compra en el supermercado. En este apartado trataré de explicar algunas de las alternativas a este proyecto que actualmente puede adquirir cualquier consumidor.

Empecemos con la nevera inteligente *Family Hub* de Samsung (2023), que ofrece una serie de recursos y funcionalidades que mejoran la experiencia en la cocina y promueven la interacción familiar. Estas neveras inteligentes transforman la cocina en un centro de entretenimiento, comunicación y planificación alimentaria, siendo una nevera eficiente energéticamente y con una capacidad interna mayor a lo habitual. Dispone de una pantalla multifunción en la puerta con la que puedes ver la televisión, series, poner notas, apuntar alimentos en la lista de la compra, entre otras muchas funciones.



Figura 3.1: Frigorífico inteligente de Samsung *Family Hub* RS6HA8880S9

Además de las características tecnológicas de Samsung, las neveras *Family Hub* cuentan con un sistema de administración de alimentos y de sugerencias de recetas. Mediante la conectividad con el smartphone, es posible verificar los alimentos dentro de la nevera mediante cámaras y, manualmente, planificar las compras y anticipar las recetas que puedes hacer durante la semana. Esto es algo similar a lo que nuestro proyecto realiza pero la nevera no reconoce de forma autónoma qué alimentos hay, es el usuario el que tiene que poner los alimentos de forma manual y entonces el sistema te aconsejará una serie de recetas que puedes hacer junto con un plan semanal.

Por otro lado, tenemos otra gama de frigoríficos inteligentes de una marca distinta, hablamos de los frigoríficos inteligentes con *Home Connect* de Bosch (2023). Estos frigoríficos son algo más básicos que el que hemos mencionado anteriormente de Samsung pero tienden a ser más espaciosos y con precios más bajos. La tecnología *Home Connect* permite conectar la nevera con tu smartphone y así observar mediante las cámaras del interior qué alimentos hay, controlar la temperatura por zonas y activar o desactivar otras funciones.



Figura 3.2: Frigorífico inteligente de Bosch con *Home Connect Ready*

Comparando esta nevera con el objetivo de nuestra aplicación, podríamos decir que no tienen demasiado que ver. La nevera de Bosch se centra en controlar funciones elementales de una nevera añadiendo cámaras en su interior para que el usuario pueda observar de forma manual de nuevo qué contiene.

Para terminar de hablar de neveras, vamos a ver la nevera inteligente de LG, *LG Smart Instaview* (ProAndroid (2017)), la cual es interesante ya que puede integrar algunas funciones con Amazon Alexa. Otra característica relevante sería la pantalla, la cual es translúcida y te permite observar los alimentos desde el exterior de la nevera sin abrir la puerta. Esto es una gran ventaja ya que no necesitas abrir las puertas y consumir más energía. No hay demasiadas diferencias entre los frigoríficos de Samsung y LG, ambas tienen una pantalla en la puerta, puedes apuntar cosas, descargarte aplicaciones y saber, por ejemplo, cuándo van a caducar los yogures. Gracias al asistente virtual y otras funcionalidades de la pantalla, LG supera a los frigoríficos Samsung en este aspecto.



Figura 3.3: Frigorífico inteligente LG Smart Instaview

Por comparar esta nevera de LG con nuestra aplicación, ocurre algo similar que con el resto de neveras comentadas, las cuales realizan operaciones de control de los alimentos de su interior pero no de forma autónoma, es decir, el usuario tendrá que poner manualmente los alimentos que hay en su interior o la caducidad de cada uno, por ejemplo. También tienen una opción para indicar qué alimentos tienes y qué posibles recetas puedes hacer pero lo dicho, poniendo manualmente los ingredientes.

La última idea que he encontrado similar a mi proyecto es una aplicación desarrollada por estudiantes de *KeepCoding Tech School (2022)*. Lo que hicieron fue poner cámaras en el interior de una nevera y que cada cierto tiempo estas hicieran una foto de diferentes partes del interior y, mediante técnicas de reconocimiento de imágenes detectar los alimentos y poder realizar de forma prácticamente autónoma un pedido al supermercado que deseas por internet. Esta idea se basa en el mismo concepto de detección de imágenes dentro de una nevera pero no tiene la misma finalidad ni está desarrollada de la misma forma.

Para concluir este apartado, podemos decir que la idea del proyecto y la finalidad es bastante innovadora y original. En el sector de venta de frigoríficos no hay ningún tipo de sistema que reconozca los alimentos del interior de la nevera ni que de forma autonóma te indique qué recetas puedes hacer. Tampoco he logrado encontrar ningún software que realice algo similar ni con el mismo fin, solo el proyecto que he mencionado al final de este apartado.

4 Marco Teórico

En este capítulo trataremos de explicar las bases teóricas de las redes neuronales, así como conceptos elementales del funcionamiento de la IA. Para finalizar explicaremos con qué sistema hemos entrenado nuestra red neuronal.

4.1 El entrenamiento del modelo

Aquí vamos a explicar a grandes rasgos en qué consiste el entrenamiento de una red neuronal, qué tipos hay y cuáles se van a emplear.

4.1.1 Análisis de nuestras neuronas

Para poder comprender cómo funcionan las redes neuronales en la IA, primero hay que entender que están compuestas por neuronas similares a las nuestras. En el proceso creativo del ser humano, el análisis del entorno, la comprensión de la causa y efecto de los fenómenos observados, la categorización, y la posible replicación e implementación de las ideas son inevitables. En la historia del desarrollo tecnológico, este proceso ha sido aplicado una y otra vez, siendo los aeroplanos un claro ejemplo, inspirados en la capacidad de las aves para volar, y el automóvil, inspirado en la locomoción de los caballos que por siglos han acompañado a nuestra especie en sus desplazamientos. Por ello, la base de la IA reside en intentar replicar la estructura y el funcionamiento del cerebro.

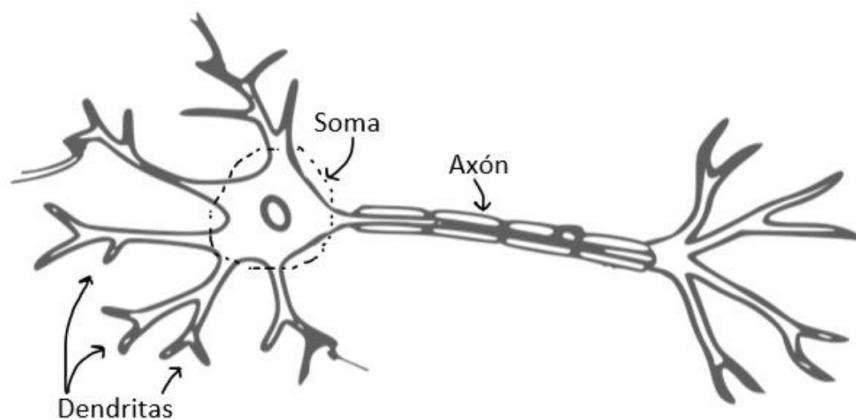


Figura 4.1: Esquema básico de una neurona del cerebro humano

Las neuronas del cerebro humano son células especializadas en la comunicación y procesamiento de información. Cada neurona tiene una estructura básica compuesta por un núcleo o

soma, dendritas que reciben señales de otras neuronas, un axón que transmite señales a otras neuronas, y terminales sinápticos que se comunican con las dendritas de otras neuronas.

El funcionamiento de una neurona humana implica la recepción de señales eléctricas y químicas en las dendritas, que se suman y se procesan en el cuerpo celular de la neurona. Si la señal supera un umbral de excitación, se genera un potencial de acción que viaja por el axón de la neurona hasta los terminales sinápticos, donde se liberan neurotransmisores que se unen a las dendritas de otras neuronas y transmiten la señal.

El proceso de transmisión de señales eléctricas y químicas entre neuronas se conoce como sinapsis y es fundamental para el funcionamiento del cerebro humano, ya que permite la comunicación entre diferentes regiones y áreas del cerebro y el procesamiento de la información. La complejidad del cerebro humano se debe en gran parte a la enorme cantidad de neuronas interconectadas y sinapsis que lo componen, lo que permite la realización de tareas cognitivas sofisticadas y el aprendizaje y la adaptación a diferentes situaciones y entornos.

Por hacer una equivalencia a la informática, el soma o núcleo sería el disco duro de la célula que almacena información que le proporciona otra célula. La dendritas equivaldrían al hardware siendo los periféricos de entrada como sensores, micrófonos, entre otros. Por último estaría el axón, sería la salida del sistema, el cual podría ser una pantalla o una respuesta lumínica, por ejemplo.

Así pues, podríamos crear una especie de modelo de caja negra de una neurona que nos ayudará más adelante para intentar replicarla.

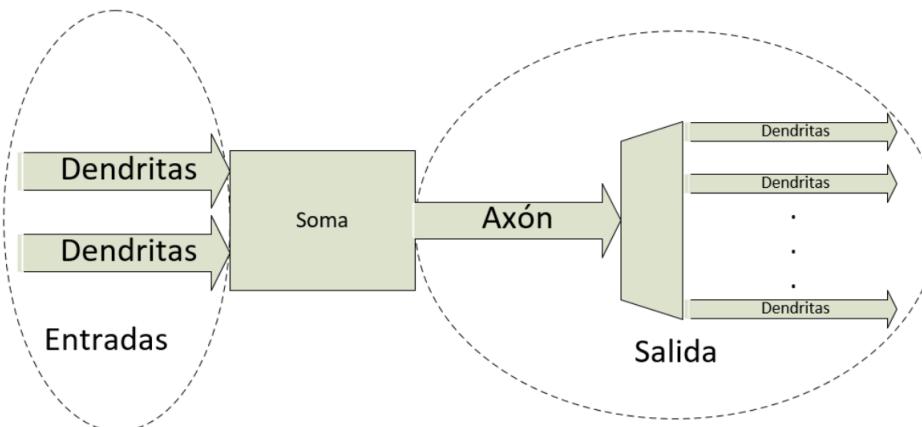
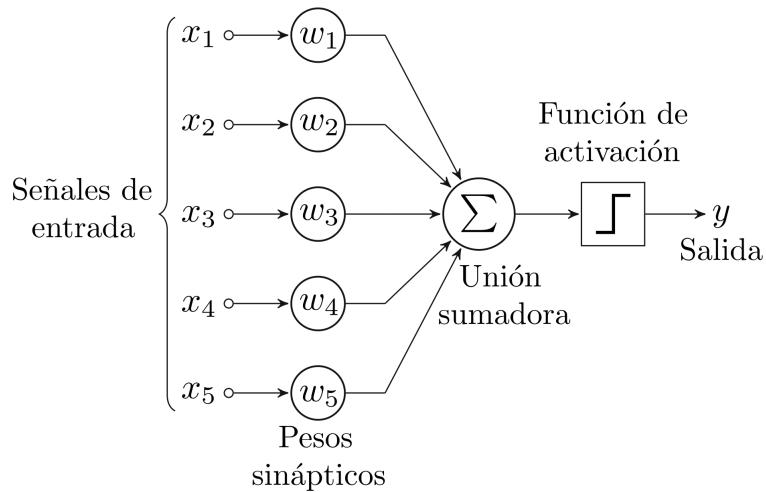


Figura 4.2: Posible modelo de caja negra de una neurona humana

Podemos hacer una nueva analogía con todo lo que hemos visto hasta ahora y definirlo en un contexto relevante. Por lo tanto, en 1960, el psicólogo Frank Rosenblatt (Wikipedia (2022)) introdujo el término "Perceptrón" para describir el modelo conceptual de una neurona artificial.

**Figura 4.3:** Esquema del Perceptrón

En la ilustración anterior, se presentan todos los componentes del perceptrón (Wikipedia (2023)). En primer lugar, se tienen los estímulos externos que conforman las señales de entrada de la neurona artificial. En este caso, se representan hasta cinco estímulos externos como x_i , los cuales forman el vector de entradas. Cada entrada tiene un peso asociado (w_i), que es un parámetro que determina la relevancia lineal de una entrada en particular.

Para comprender esto, podemos tomar un ejemplo simplificado. Imaginemos que tenemos un detector de metales y debemos realizar una búsqueda de limpieza en una playa. El detector emite una señal sonora a medida que nos acercamos a un objeto metálico, aumentando la intensidad del pitido a medida que nos acercamos y disminuyendo a medida que nos alejamos. En este caso, podríamos decir que el peso representa la cercanía al objeto metálico.

En el ámbito de estudio que nos ocupa, el vector de pesos es una herramienta lógico-matemática que utilizamos para modelar la relevancia de una entrada determinada (como una imagen), a fin de determinar de manera unívoca y minimizar la probabilidad de error en la correspondencia entre dicha imagen y los resultados esperados.

Continuando con el análisis de esta neurona artificial, encontramos la unión sumadora (también llamada función de propagación) y la función de activación, que juntas se pueden identificar como una única función de transferencia que genera una salida específica basada en el vector de entradas y su procesamiento.

La función de propagación se encarga de sumar y ponderar todas las entradas con sus respectivos pesos, a lo cual se agrega un sesgo, que es un valor de compensación para ajustar el resultado final de la suma ponderada. Podemos decir que actúa como un factor de corrección.

La función de activación, en resumen, funciona como un interruptor, determinando si el perceptrón está "encendido" o "apagado". En otras palabras, determina si el vector de entradas inicial ha estimulado la neurona, lo que provoca su activación. La salida y se puede expresar matemáticamente como:

$$y = w_0 + \sum_{i=1}^n x_i w_i$$

Figura 4.4: Ecuación de salida de un perceptrón

Para finalizar, podríamos decir que el perceptrón se podría enfocar como un sensor luminoso que capta la luz y la convierte en impulsos eléctricos.

4.1.2 Redes neuronales

Después de examinar la unidad más básica, podemos abordar el conjunto a nivel operativo. El término "red neuronal" se refiere a un conjunto de neuronas artificiales que trabajan juntas hacia un objetivo común, comunicándose entre sí mediante señales. Existen diferentes categorías según la topología de la red y el método de aprendizaje. Para empezar, podemos observar en el siguiente esquema los tipos de redes neuronales según su estructura de red (Calvo (2017)).

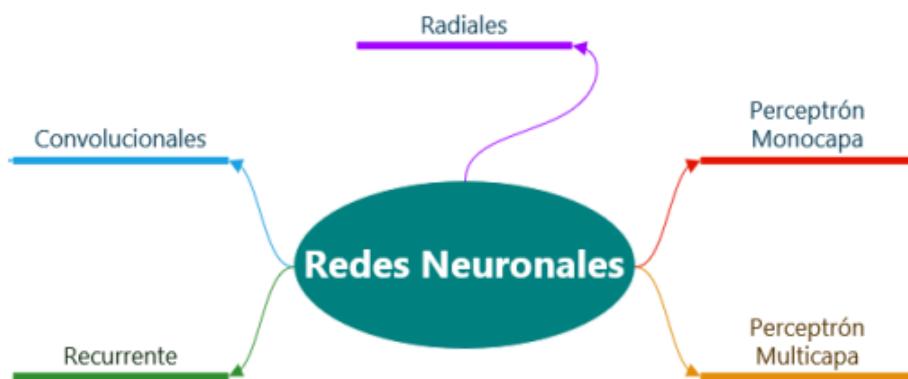


Figura 4.5: Tipos de redes neuronales

Hasta ahora, hemos examinado la unidad básica neuronal conocida como perceptrón. Es importante destacar que un único perceptrón puede funcionar como una red neuronal en sí misma. Su principal aplicación se encuentra en la regeneración de vectores de entrada en redes neuronales donde la información puede estar distorsionada o incompleta. Por ejemplo, puede actuar como un regenerador de señal en enlaces de comunicación por fibra óptica, ayudando a mitigar las distorsiones causadas por la atenuación y el ruido presentes en el canal.

A continuación, podemos combinar varios perceptrones para formar una red multicapa. En esta configuración, los distintos bloques con funciones de agregación y activación se agrupan en capas ocultas. Dependiendo de las conexiones entre ellos, podemos encontrar una estructura de malla parcial o total.

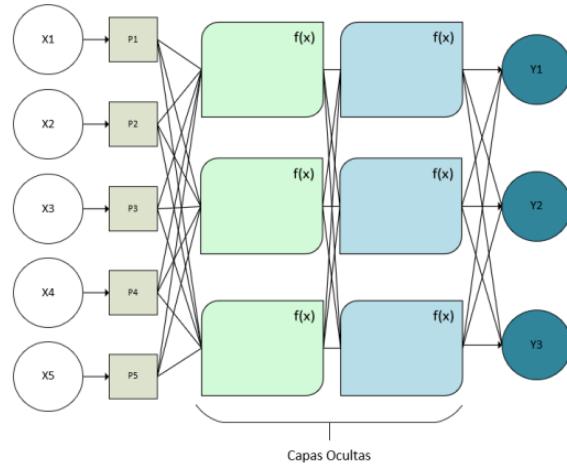


Figura 4.6: Red neuronal multicapa

Al agrupar los perceptrones de esta manera, logramos aumentar la escalabilidad de la red, al mismo tiempo que brindamos la capacidad de especialización a las neuronas artificiales.

En nuestro caso, vamos a explicar las redes neuronales convolucionales, las cuales son las más utilizadas para la identificación de imágenes. A diferencia de las redes multicapa de perceptrones, las redes neuronales convolucionales no presentan una interconexión total entre todas sus capas, sino que se establecen conexiones específicas a ciertas capas. Esto reduce el uso de las neuronas artificiales involucradas en la red y también disminuye el costo computacional. Esta arquitectura se basa en un enfoque geométrico de una matriz bidimensional sobre la cual se realizan operaciones. Este enfoque permite llevar a cabo tareas de segmentación y clasificación, que son fundamentales para la visión artificial.

Estas redes reciben su nombre debido a la operación de convolución que se realiza con las entradas y cada conjunto de neuronas artificiales. En este contexto, la convolución se refiere al producto escalar matricial. A continuación se muestra una ilustración de una arquitectura de una Red Neuronal Convolucional (CNN) (Swapna (2020)).

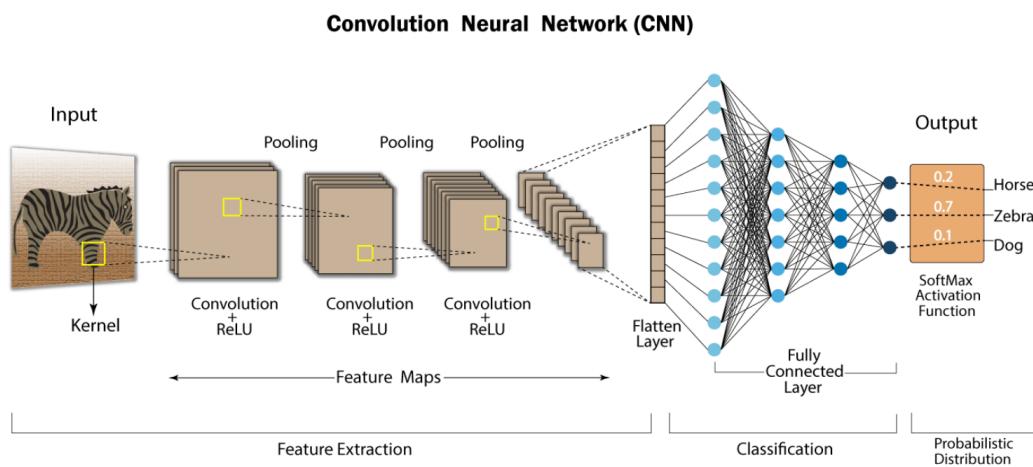


Figura 4.7: Ejemplo práctico de una red neuronal convolucional (CNN)

Podemos observar que la entrada de la red es una imagen, de la cual extraemos una región inicial que llamaremos kernel. A continuación, iniciamos un proceso de análisis a nivel elemental de la región seleccionada, evaluando, por ejemplo, las intensidades de color dentro de la matriz RGB. Luego, realizamos un muestreo mediante un fragmento de la matriz en la que estamos trabajando, aplicando el producto escalar con sus respectivos pesos. Esto da como resultado una matriz de dimensiones más pequeñas, que luego comparamos con la función de activación, en este caso, la ReLU. A partir de ahora, llamaremos a este proceso "sondeo", que no es más que la aplicación sistemática de la convolución junto con la función de activación.

Desde una perspectiva geométrica, podríamos decir que estamos proyectando la matriz con la que trabajamos, reduciendo su dimensión al mínimo. Una vez finalizada la etapa de mapeo, que implica sucesivos sondeos, habríamos completado la extracción de características y estaríamos listos para pasar a la etapa de clasificación. En esta etapa, utilizaríamos una red neuronal de perceptrones multicapa con conexiones completas, que finalizaría en una distribución probabilística que representa todos los posibles resultados esperados.

4.1.3 Entrenamiento de una red neuronal

Durante el proceso de entrenamiento, el objetivo es ajustar los diferentes pesos de la red neuronal de modo que las capas de salida se alineen con los datos proporcionados por los modelos. Mediante el aprendizaje, buscamos capacitar a la red neuronal para que pueda discernir si los resultados de sus operaciones son correctos o no. Para comprender mejor lo que buscamos, consideremos un ejemplo aplicado a la visión artificial. Todos los conceptos que se proporcionan a continuación pueden verse en "Calvo (2017)".

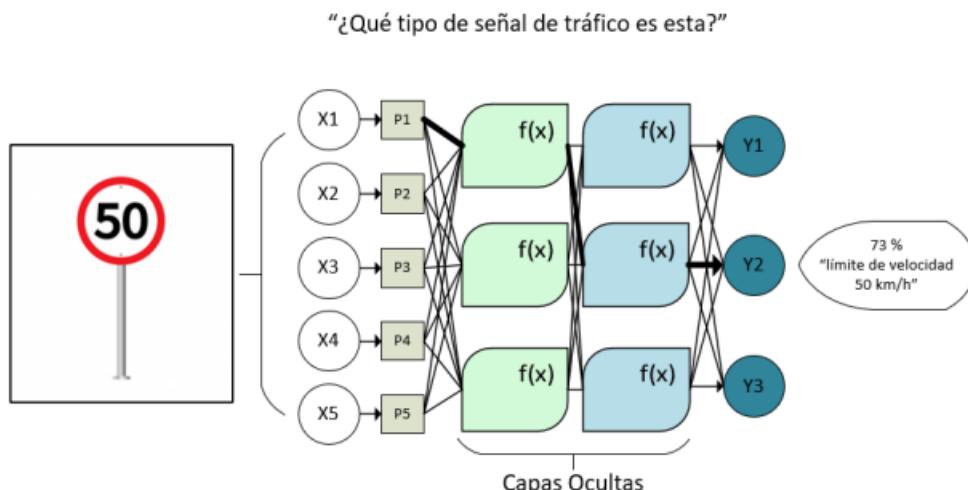


Figura 4.8: Ejemplo de entrenamiento y detección de una señal de tráfico

Al ajustar los valores de los pesos, la salida se vuelve más definida (se ha representado con un trazo más grueso para ilustrar este aumento en el peso 1). Cada posible resultado durante la fase de entrenamiento lleva asociada una probabilidad que indica la certeza del resultado. En este caso, nuestra red neuronal tiene un 73% de certeza de que corresponde a un límite de velocidad de 50 km/h, con un 27% de probabilidad de haber cometido un error.

al identificarlo como cualquier otro elemento reconocible en su conjunto de clases.

Este ejemplo es solo una de las muchas iteraciones necesarias para que una red neuronal pueda identificar con precisión la señal de tráfico mostrada. Cuanto mayor sea el conjunto de imágenes utilizado en la fase de entrenamiento (es decir, cuanto más amplio sea el conjunto de datos), menor será la probabilidad de cometer errores en el reconocimiento.

El entrenamiento puede ser categorizado según la supervisión de la red misma o según su dinámica. En términos de dinámica, existen varias formas de aplicar el proceso de aprendizaje en función de los pesos. Nuestra red puede aprender a partir de un entrenamiento previo antes de su implementación, utilizando un conjunto de datos de entrenamiento y un conjunto de datos de prueba. A esto se le llama entrenamiento OFF-LINE.

Por otro lado, nuestra red también puede aprender durante su funcionamiento, a medida que se le presenta nueva información al sistema. A esto se le conoce como entrenamiento ON-LINE. Para una mejor visualización y brevemente, se propone la siguiente enumeración con los tipos de entrenamiento disponibles:

4.1.3.1 Redes supervisadas

Son aquellas en las que se supervisa el aprendizaje a través de una entidad externa que determina la salida basándose en una entrada específica.

Dentro de este tipo de aprendizaje, podemos identificar los siguientes tipos:

- **Supervisado por corrección de error:** Se trata de ajustar los pesos en las conexiones de la red para adaptarlos a la salida deseada. Esta dinámica se encuentra presente en varios campos de la ingeniería, especialmente en el ámbito del control automático.
- **Supervisado por refuerzo:** En este tipo de aprendizaje supervisado, en lugar de contar con un ejemplo específico de la salida deseada como en el caso anterior, la supervisión se realiza mediante una señal de refuerzo que asigna un valor de +1 si la salida se ajusta al resultado deseado, o un valor de -1 en caso contrario. Debido a esto, el proceso es más lento.
- **Supervisado Estocástico:** En este enfoque, invertimos el proceso. En lugar de ajustar los pesos basados en una entrada determinada, lo que se hace es cambiar aleatoriamente los valores de los pesos y se observa si se obtiene la salida deseada. Un ejemplo de este método de aprendizaje es una red basada en la máquina de Boltzmann (Eswitha_reddy (2020)).

4.1.3.2 Redes NO supervisadas

Estas redes neuronales no requieren de un agente externo para modificar los pesos de las conexiones entre sus neuronas. Su funcionamiento se basa en la búsqueda de características, regularidades, correlaciones y categorías en los datos de entrada. Las posibles interpretaciones de las salidas de estas redes son las siguientes:

- El grado de similitud entre la entrada actual y las entradas previas.
 - La categoría a la que pertenece la entrada recibida (clusterización).
-

- El mapeo de características presentes en la entrada.
- La codificación de la entrada recibida, generando una versión codificada con menor cantidad de bits sin perder información crítica.

La última interpretación es la clave en la que se basan las operaciones de convolución dentro de las redes neuronales convolucionales.

En este tipo de aprendizaje podemos listar entre:

- **Aprendizaje Hebbiano:** el objetivo de estas redes neuronales es extrapolar características y establecer relaciones de similitud a partir de los datos de entrada. Para lograrlo, se ajustan los pesos en las conexiones en función de la correlación entre las salidas de las neuronas conectadas.
- **Aprendizaje competitivo y colaborativo:** en este tipo de aprendizaje, se busca la proximidad a un patrón de entrada, promoviendo la competitividad entre las neuronas. La neurona que se acerque más al patrón ajustará su peso y será la única en activarse.

Para concluir este apartado, en el siguiente esquema podemos observar las diferentes formas de aprendizaje anteriormente descritas:

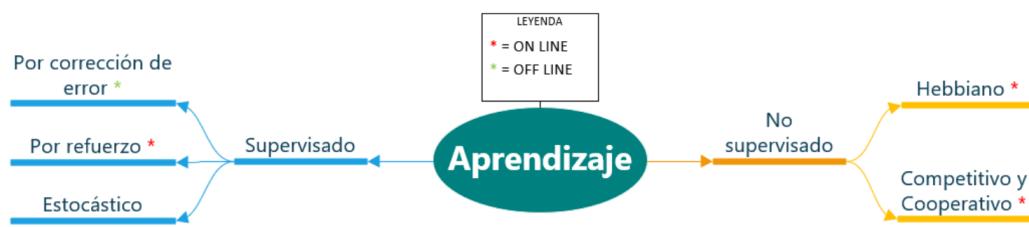


Figura 4.9: Tipos de aprendizaje de una red neuronal

Ya habríamos visto toda la teoría básica necesaria para entender cómo funcionan las redes neuronales. Ahora vamos a proceder a explicar de forma breve cómo funciona YOLOv5.

4.2 Funcionamiento de YOLOv5

La arquitectura YOLO se presentó por primera vez en 2015 en el artículo "You Only Look Once: Unified, Real-Time Object Detection" escrito por Joseph Redmon. Su uso se popularizó rápidamente debido a su alta precisión y su capacidad para funcionar en tiempo real.

YOLO divide la imagen en una cuadrícula, donde cada celda se encarga de verificar la presencia de un objeto de interés en su interior. Para lograr esto, cada celda genera múltiples bounding boxes¹ de diferentes tamaños y establece niveles de confianza para predecir la existencia de un objeto. Desde la publicación del artículo, se han desarrollado varias versiones de la red YOLO, llegando a ocho versiones hasta la fecha.

4.2.1 Versiones de YOLO

Cada iteración de la arquitectura YOLO introduce mejoras significativas respecto a sus versiones anteriores. Sin embargo, YOLOv3 se destaca por su impacto significativo en el campo. Al reemplazar la función *softmax* por clasificadores logísticos independientes y agregar extracción de características, YOLOv3 se convirtió en una innovación disruptiva al ofrecer un rendimiento similar a otras redes de detección en una sola etapa (*one-stage*) y una velocidad notablemente superior.

Después de YOLOv3, se introdujeron YOLOv4 y YOLOv5, que presentaron cambios significativos que mejoraron los *benchmarks* de detección de objetos. Entre estos cambios se incluye el uso de *Mosaic Data Augmentation* durante el entrenamiento y la implementación de *Cross Stage Partial Networks* como columna vertebral de las arquitecturas. Estas modificaciones resultaron en mejoras notables en la precisión de los resultados y en la velocidad de entrenamiento. Es importante destacar que YOLOv5 ofrece resultados similares a su versión anterior, pero con un tiempo de entrenamiento aún más reducido. Además, esta arquitectura se implementa nativamente en *PyTorch*, a diferencia de las versiones anteriores que estaban basadas en *Densenet*.

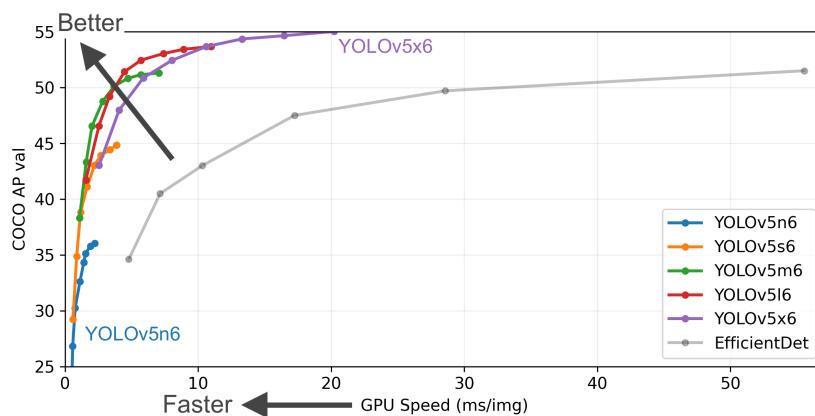


Figura 4.10: Benckmark de YOLOv5

¹Cuadro delimitador que encierra una región específica de una imagen. Está compuesto por cuatro números que definen las coordenadas de los límites de la región en la imagen

Al igual que YOLO tiene diferentes versiones, dentro de cada versión existen varias arquitecturas. Específicamente para YOLOv5 (Jocher (2023)), se cuentan con cinco modelos denominados P5: YOLOv5n (nano), YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large) y YOLOv5x (extra large). Además, hay modelos P6, siendo la única diferencia que los modelos anteriores están diseñados para ser entrenados con imágenes de resolución 640×640 , mientras que P5 se utiliza para una resolución de 1280×1280 . En general, el aumento en el tamaño de los modelos resulta en una mayor precisión, pero también en una disminución en la velocidad de inferencia, por lo que la elección de la arquitectura depende de la importancia de la rapidez en cada caso.

En conclusión, he utilizado YOLOv5 por su rapidez y eficacia a la hora de detectar objetos y su gran abanico de datasets disponibles en la red. En el siguiente capítulo explicaré qué pruebas he realizado y cómo he utilizado esta tecnología para el proyecto.

4.3 Entrenamiento de una red neuronal convolucional (CNN)

El entrenamiento de una CNN se realiza en varias etapas llamadas épocas. Una época representa una pasada completa por todo el conjunto de datos de entrenamiento. Durante cada época, la red neuronal ajusta los pesos de sus conexiones internas con el objetivo de mejorar su rendimiento en la tarea específica para la que está siendo entrenada.

El proceso de entrenamiento se realiza de la siguiente manera:

1. Se selecciona un lote (batch) de datos de entrenamiento. Un lote consiste en un subconjunto de ejemplos del conjunto de datos total. Utilizar lotes en lugar de todo el conjunto de datos permite procesarlos de forma más eficiente y aprovechar las capacidades de cómputo paralelo de las GPUs.
2. Se envían los datos del lote a la red neuronal, que realiza una propagación hacia adelante (forward propagation) para calcular las salidas de la red.
3. Se compara la salida de la red con las etiquetas o valores deseados, y se calcula una medida de error, generalmente utilizando una función de pérdida (loss function).
4. A continuación, se realiza una propagación hacia atrás (backpropagation) para calcular los gradientes de error con respecto a los pesos de la red. Estos gradientes indican la dirección y magnitud en la que los pesos deben ajustarse para reducir el error.
5. Se actualizan los pesos de la red utilizando un algoritmo de optimización, como el descenso de gradiente estocástico (SGD) u otros métodos más avanzados como Adam o RMSprop. Esta actualización de pesos se realiza en base a los gradientes calculados durante el paso anterior.
6. Repitiendo los pasos anteriores para todos los lotes del conjunto de datos de entrenamiento, se completa una época. Es decir, se ha realizado una pasada completa por todo el conjunto de datos de entrenamiento y se han actualizado los pesos de la red en base a los errores cometidos.
7. Se repiten las épocas hasta alcanzar un número predefinido de épocas o hasta que se observe una mejora en el rendimiento de la red en un conjunto de datos de validación.

El número de épocas es un hiperparámetro que debe ajustarse durante el entrenamiento. Un número insuficiente de épocas puede resultar en una red que no ha tenido suficiente tiempo para aprender patrones en los datos, mientras que un exceso de épocas puede conducir a un sobreajuste, donde la red memoriza los datos de entrenamiento pero no generaliza bien a nuevos ejemplos.

En resumen, el entrenamiento de una CNN se lleva a cabo mediante la repetición de pasos de propagación hacia adelante, cálculo del error, propagación hacia atrás y actualización de pesos durante varias épocas, con el objetivo de mejorar gradualmente el rendimiento de la red en la tarea para la que está siendo entrenada.

5 Metodología y Tecnologías

En esta sección explicaré detalladamente qué tecnologías he empleado y cómo las he utilizado para poder desarrollar este proyecto.

5.1 YOLOv5

Como se ha explicado anteriormente, YOLOv5 (Jocher (2023)) es un modelo de detección de objetos de última generación que utiliza una arquitectura de red neuronal convolucional profunda para identificar y localizar objetos en imágenes y videos en tiempo real.

En este proyecto se empleado por su rapidez y precisión para detectar los alimentos o ingredientes de una imagen de una nevera abierta. Se han realizado algunas pruebas con diferentes datasets, épocas y arquitecturas de la red hasta que se ha obtenido una red bastante bien entrenada pero con algunas discrepancias (como veremos más a fondo en siguientes apartados).

5.2 Datasets de entrenamiento

Un dataset es una colección o conjunto de información que se emplea para entrenar modelos de aprendizaje automático. En nuestro caso, es una colección de imágenes donde cada imagen tiene asociado un fichero de texto con las clases o ingredientes que hay en esa imagen. Todos los datasets empleados para el entrenamiento, validación y test se han obtenido mediante Roboflow (2023). Tras una búsqueda exhaustiva y realizar algunas pruebas, se ha optado por juntar cuatro datasets bastante similares mediante un pequeño programa escrito en Java el cual unificaba las clases, las imágenes y las etiquetas de cada imagen. Esta unión ha provocado que la parte de la aplicación de reconocimiento de imágenes tenga 4079 imágenes de entrenamiento, 797 para validación y 423 para test. Posteriormente observaremos las diferentes pruebas y resultados con estos datasets.

5.3 API de ChatGPT

ChatGPT (s.f.) es un modelo de lenguaje desarrollado por OpenAI basado en la arquitectura GPT-3.5. Es una potente herramienta de procesamiento de lenguaje natural que permite interactuar con los usuarios en forma de conversación.

En este proyecto concretamente y para obtener las recetas de los ingredientes proporcionados, se ha empleado la API de ChatGPT *gpt-3.5-turbo* (*API de ChatGPT* (s.f.)) que, aunque sea un poco lenta, funciona a la perfección por sus respuestas más simples. En la sección "Desarrollo" veremos más en profundidad qué preguntas le formulamos y cómo nos responde.

5.4 Flask

Flask es un framework web ligero y flexible para construir aplicaciones web en Python. Se basa en el concepto de "microframework", lo que significa que proporciona solo lo esencial para crear aplicaciones web sin imponer una estructura rígida. Flask facilita la creación de rutas URL, la gestión de solicitudes y respuestas HTTP, y la renderización de plantillas HTML.

Nosotros hemos empleado Flask de forma bastante simple y sin grandes complicaciones. En los inicios del proyecto iba a ser una aplicación móvil desarrollada con Ionic y React pero por diversos problemas de compatibilidad y de dispositivos descarté esa opción. Por ello, ahora tenemos una aplicación multidispositivo y responsive, la cual podemos utilizar donde y cuando queramos.

5.5 Sistema Operativo y otras herramientas

Para terminar este apartado, me ha parecido interesante comentar algunas de las herramientas que he empleado para realizar las diferentes partes del proyecto.

Por un lado tendríamos la parte del entrenamiento de la red neuronal convolucional mediante YOLOv5, el cual ha sido posible gracias a un servidor remoto que dispone el departamento, el cual dispone de cuatro GPUs para este objetivo. Desde un ordenador dentro del sistema operativo Ubuntu y conectándose al servidor mediante *SSH*, puedes enviar y recibir información y dejar entrenando la red durante largos períodos de tiempo.

Por otro lado tendríamos el desarrollo de la aplicación web, la cual se ha realizado en Windows porque durante el desarrollo del proyecto Ubuntu se corrompió y hubo una pérdida grande del proyecto. Dado que la aplicación (Flask), YOLOv5 y la API de ChatGPT están desarrolladas en Python y que en el sistema operativo Windows es bastante sencillo instalar Python, de forma muy sencilla se puede implementar la aplicación y sus diferentes partes.

Por último, me gustaría destacar que la aplicación ha sido desplegada en local, quiere decir que un ordenador hace de servidor dentro de una red local y que el resto de dispositivos conectados a esa red pueden acceder a la aplicación web. No obstante, el proyecto está diseñado para poder desplegarlo en internet y que la aplicación web se pueda utilizar desde cualquier lugar y dispositivo.

6 Desarrollo

El desarrollo de aplicación consta de tres grandes partes: preparación del dataset y entrenamiento de la red neuronal de YOLOv5, el empleo de la API de ChatGPT y la propia implementación de la aplicación web (siendo esta la parte visible del proyecto). Se podría seguir este esquema para entender mejor las fases y desarrollo del proyecto:

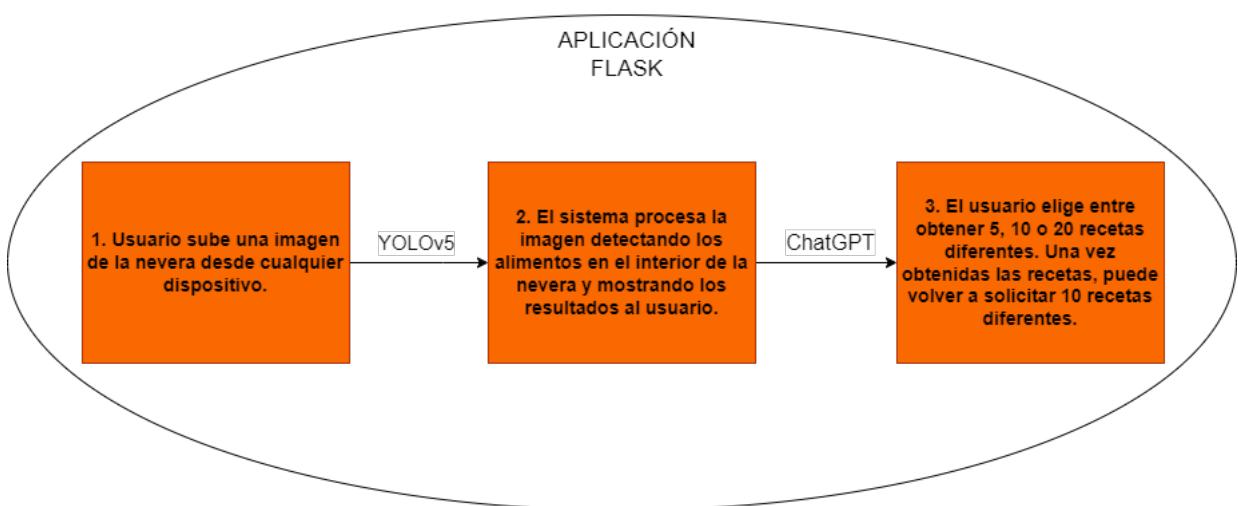


Figura 6.1: Estructura de desarrollo

6.1 YOLOv5

El reconocimiento de los alimentos en el interior de la nevera se ha realizado mediante YOLOv5, por ello, se va a proceder a explicar que problemas han existido empleando esta tecnología y cómo se han solucionado. Además, se han realizado numerosas pruebas de este modelo con diferentes épocas y configuraciones que se podrán observar en el apartado de "Resultados".

6.1.1 Preparación del dataset

En un primer momento y como se ha mencionado con anterioridad, se hicieron numerosas pruebas con diferentes datasets y épocas para observar cómo se comportaba el modelo con diferentes datasets. Se eligieron los siguientes datasets para hacer pruebas y, posteriormente, juntarlos:

1. Dataset 1 creado por aicook (2022).

2. Dataset 2 creado por SmarterChef (2023).
3. Dataset 3 creado por Matima (2022).
4. Dataset 4 creado por Aguinaco (2023).

Tras hacer esto, se contemplaban dos grandes problemas: el dataset era muy pobre respecto a las clases que tenía (tenía solamente 20 clases o menos) o el número de clases era muy elevado y algunas incluso se repetían.

Haciendo incapié en el último problema, algunos de los datasets que existen en Roboflow tienen la particularidad de tener clases repetidas. Concretamente se daba el caso de que existían varias instancias del mismo alimento pero con diferente clase. Un ejemplo claro serían las clases "Lemon", "lemon" y "lemons". Esto ocasionaba que algunas instancias de estas clases apareciesen más que otras y la muestra se descompensase cuando realmente es el mismo alimento.

Para poder solucionar ambos problemas, se juntaron varios datasets con diferentes clases. Esto suponía otro problema, el de unificar esas clases en un único dataset para que a la hora de entrenar las clases fueran las mismas. Por ello, la solución fue implementar un pequeño programa en Java el cual cambiase de todos los ficheros de texto asociados a cada imagen la clase a una global asignada por nosotros. La siguiente tabla muestra el nombre de la clase (alimento o ingrediente), el número global y cada número que representa esa clase respecto al número de la clase global. Por ejemplo, en el dataset 2 la clase "Apple" tiene el número 2 pero en el dataset global, tendrá el número 0, por ello hay que hacerlo para cada clase y dataset:

ID	NOMBRE	DATASET			
		1	2	3	4
0	Apple		7	13 / 14	0
1	Asparagus			15	
2	Aubergine		8	16	15
3	Bacon			17	
4	Banana		9	0 / 18 / 19	1
5	Bazlama			20	
6	Beef	0		21	2
7	Beetroot			1	
8	Blueberries		10	22	3
9	Bread	1	11	23	
10	Broccoli		12	2 / 24	4
11	Butter	2	13	25	5
12	Carrot		14	26 / 27	6
13	Cauliflower			3	7
14	Cheese	3	15	28	8
15	Chicken	4	16	29	9
16	Chicken breast			30	
17	Chocolate			32	10
18	Chocolate chips			31	

ID	NOMBRE	DATASET			
		1	2	3	4
19	Corn			33	11
20	Courgettes		17	34	
21	Cream	5		36	
22	Cream cheese			35	12
23	Cucumber				13
24	Dates			37	14
25	Eggs	6	18	38	16
26	Flour			39	
27	Ginger		19	40	17
28	Goat cheese			41	
29	Green beans		20	42 / 45	19
30	Green bell pepper			43	20
31	Green chilies		21	44	21
32	Ground beef			46	22
33	Ham		22	47	
34	Heavy cream			48	23
35	Juice			49	
36	Kiwi				24
37	Lemon		0 / 23	4 / 50 / 51	25
38	Lettuce		24	52	26
39	Lime		25	53	27
40	Mango			54	
41	Meat			55	
42	Milk	7	26	56	28
43	Mint				30
44	Mushroom		27	58 / 59	31
45	Olive			60 / 61	32
46	Onion		1 / 28	5 / 62	33
47	Orange		2 / 29	6 / 63	34 / 52
48	Parsley			64	35
49	Peach		30	65	36
50	Peas		3	7	37
51	Pickles				38
52	Potato		4 / 32	8 / 67 / 68	39
53	Radish				40
54	Red bell pepper			66 / 69	41
55	Red cabbage				42
56	Red grapes			70	18 / 43
57	Red onion		33	71	44
58	Salami	8		72	45
59	Sauce			73	
60	Sausage			74	46
61	Shallot			9	

ID	NOMBRE	DATASET			
		1	2	3	4
62	Shrimp			75	47
63	Spinach		34	76	48
64	Spring onion		35	77	49
65	Strawberry	5 / 36		10 / 78 / 79	50
66	Sugar			80	
67	Sweet potato		37	81	51
68	Sweetcorn			11	
69	Tomato	6 / 38		12 / 83 / 84	53
70	Tomato paste	9		82	54
71	Water			57	29
72	Yellow bell pepper			85	55
73	Yoghurt	10	39	86	56
74	Zucchini				57

Cuadro 6.1: Clases del dataset global respecto a cada uno

6.1.2 Pruebas de entrenamiento

Una vez está solucionado el problema con los datasets, ahora se va a utilizar el dataset grande compuesto por cuatro datasets para entrenar nuestra red neuronal. Tras hacer varias pruebas con diferentes modelos de YOLOv5 y no obtener buenos resultados (sobreentrenamiento de la red sobre todo), se optó por emplear el modelo "YOLOv5s", el cual según indica el creador de YOLOv5, sería el segundo modelo más lento en procesar pero que para este proyecto es el que mejor ha funcionado. El resto de modelos se ciñen demasiado al dataset de entrenamiento (o incluso tienen sobreentrenamiento con las mismas épocas) y, cuando se añade una foto nueva, no reconoce prácticamente nada y lo poco que reconoce lo reconoce de forma errónea. Por tanto, el modelo "YOLOv5m" era el que hacía esto en menor medida y por ello ha sido el elegido.

En el apartado de "Resultados" se expondrán las diferentes pruebas realizadas con los modelos de YOLOv5 y con diferentes épocas. No obstante, para escoger el modelo "YOLOv5m" entrenado con 150 épocas se ha tomado en consideración la gráfica "val/obj_loss" de los resultados obtenidos y dos pruebas realizadas directamente en una nevera real para ver cuál interpreta mejor lo que hay en la nevera. Este sería un ejemplo del modelo "YOLOv5m" en una de las imágenes mencionadas:



Figura 6.2: Ejemplo de una imagen ya procesada por YOLOv5

Como podremos observar más adelante, el usuario tendrá su imagen ya procesada y el recuento de alimentos detectados en la imagen. Esto se ha podido hacer gracias a las herramientas que proporciona YOLOv5 para obtener los resultados tanto en una imagen como en un archivo TXT del cual puedo extraer lo que se ha detectado.

A partir de aquí, el usuario podrá elegir entre obtener 5, 10 y 20 recetas, lo cual nos lleva al siguiente subapartado.

6.2 API de ChatGPT

Como se acaba de mencionar, el usuario podrá elegir entre 5, 10 o 20 recetas en función de sus necesidades. Esto quiere decir que se enviará una petición similar a esta a la API de ChatGPT:

Buenas, quiero que me digas 10 posibles recetas con estos alimentos: Counter('Fresa/s': 13, 'Limón/es': 8, 'Patata/s': 7, 'Huevos': 6, 'Pimiento/s verde': 5, 'Tomate/s': 5, 'Leche/s': 4, 'Manzana/s': 3, 'Naranja/s': 3, 'Uva/s': 2, 'Berenjena/s': 2, 'Pimiento/s rojo': 2, 'Perejil': 1, 'Broccoli/s': 1, 'Lechuga/s': 1)

Básicamente lo que se hace es formular la petición con el número de recetas seleccionado y la lista proporcionada por el reconocimiento de los alimentos previos.

De esta forma y separando en párrafos para mostrar la información de la mejor forma posible, nos devuelve una lista con las 10 recetas. Este sería (continuando con el ejemplo anterior) una posible respuesta:

1. Ensalada de patatas con huevo y pimiento verde
2. Tortilla de patatas con pimiento rojo y verde
3. Tarta de manzana con crema de limón
4. Ensalada de tomate y fresas con vinagreta de limón
5. Risotto de limón con parmesano y perejil
6. Batido de leche con fresas y naranjas
7. Ensalada de brócoli con fresas y uvas
8. Berenjenas rellenas de tomate y queso gratinado
9. Patatas asadas con salsa de yogur al limón y hierbas frescas
10. Pollo a la plancha con salsa de limón y ensalada de lechuga y tomate.

Las recetas pueden ser un tanto difíciles de realizar o también se emplean alimentos que no han sido detectados, lo cual puede suponer un problema para el usuario ya que puede no disponer del ingrediente. No obstante, también podemos decir que es una ventaja ya que enriquece las recetas proporcionadas y puede dar ideas extras al usuario.

Por otra parte, el cliente puede obtener otras 10 recetas nuevas si lo desea pulsando el botón "Quiero 10 recetas diferentes", lo cual envía una petición nueva similar a esta:

*Buenas, quiero que me digas 10 posibles recetas muy elaboradas con estos alimentos:
Counter('Fresa/s': 13, 'Limón/es': 8, 'Patata/s': 7, 'Huevos': 6, 'Pimiento/s verde': 5,
'Tomate/s': 5, 'Leche/s': 4, 'Manzana/s': 3, 'Naranja/s': 3, 'Uva/s': 2, 'Berenjena/s': 2,
'Pimiento/s rojo': 2, 'Perejil': 1, 'Broccoli/s': 1, 'Lechuga/s': 1)*

La petición es muy similar a la inicial pero con la principal diferencia que ahora es más precisa y se le pide que sean "10 posibles recetas muy elaboradas". Esto provoca que las recetas sean algo más complicadas de realizar pero más realistas y con posibilidad de estar más ricas para el cliente final. Otro punto a favor de realizar esto es que las recetas nunca van a ser las mismas exactamente incluso otras nuevas ya que, al pedirle que sean más elaboradas, podrá añadir o cambiar algunos ingredientes a las recetas anteriormente proporcionadas. Además, como estamos utilizando el mismo hilo de peticiones y respuestas es difícil que se repitan las recetas. Esta sería la respuesta del ejemplo anterior:

1. Ensalada de patata, huevo cocido, pimiento verde y rojo, tomate y lechuga con una vinagreta de limón y perejil.
2. Crema de patata y pimiento rojo con trocitos de fresa y croutons de pan.
3. Huevos en cazuela con patatas, tomates, pimientos verdes y rojos, y un toque de limón.
4. Tarta de limón y fresas con base de galleta y decorada con rodajas de limón y fresas.
5. Ensalada de queso fresco, manzana, uvas, fresas y nueces, aderezada con limón y aceite de oliva.
6. Soufflé de patata y queso con trocitos de pimiento verde y servido con salsa de tomate casera.
7. Pollo al limón con patatas y acompañado por una ensalada de brocoli, tomate y limón.
8. Berenjenas rellenas de carne picada, tomate y pimiento verde, gratinadas con queso y gratinadas en el horno.
9. Granizado de naranja y limón con trocitos de fresa y rodajas de limón para decorar.
10. Merluza al horno con patatas, tomate, pimiento verde, limón y ajo, servida con una salsa de leche y perejil picado.

6.3 Aplicación web Flask

Para poder juntar la CNN ya entrenada con el módulo de la API de OpenAI, he empleado el framework Flask. Este software me ha permitido no cambiar de lenguaje de programación además de otorgar una gran flexibilidad, escalabilidad al proyecto y la posibilidad de hacer la aplicación para cualquier dispositivo (responsive). El esquema de la aplicación de forma muy general sería el siguiente:

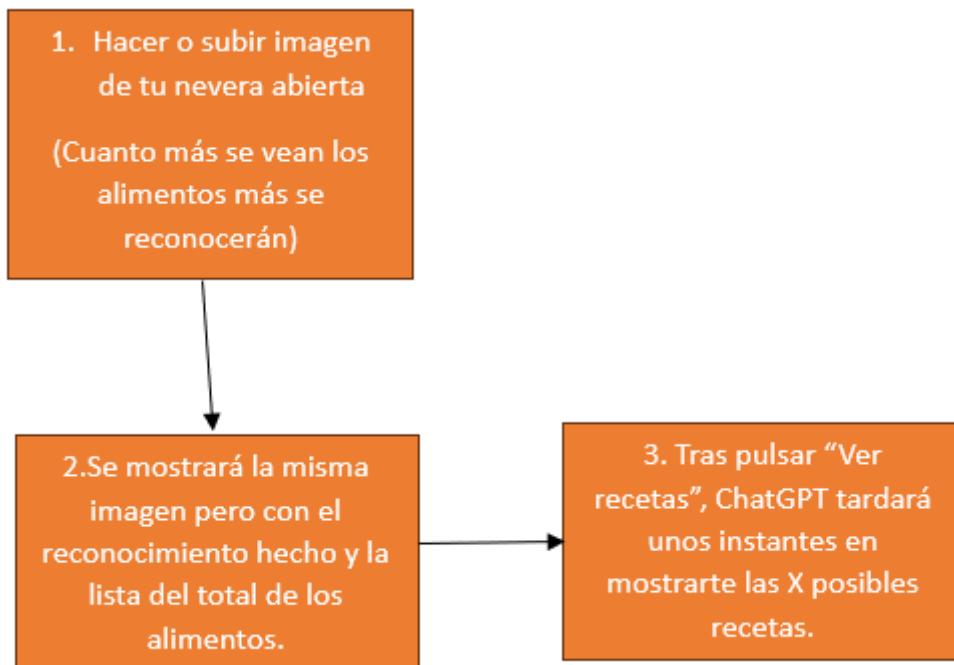


Figura 6.3: Esquema básico para un usuario de la aplicación

Explicaré mucho más en detalle cada apartado o vista de la aplicación en el apartado de "Resultados". La aplicación consta de dos ficheros y otros dos directorios muy importantes:

1. El fichero "app.py" donde se ubica toda la estructura de enlaces, funciones y métodos de la aplicación web. Sería el motor de la aplicación y la que prácticamente al desplegarse hace que todo funcione correctamente. Desde llamar a la clase de ChatGPT para utilizar la API hasta mostrar las vistas HTML. También es el encargado de manejar las excepciones o los errores de todos los usuarios y controlar el flujo de datos con YOLOv5 y ChatGPT sin cometer errores.
2. El fichero "recetas_chatgpt.py", el cual contiene la clase "chatGPT", encargada de enviar y extraer datos directamente de la API.
3. El directorio "templates" donde se alojan todas las vistas HTML de la aplicación. Es una de las partes más importantes ya que es lo que el usuario puede ver.

4. Y por último la carpeta "static" en la cual se ubican todos los elementos visuales (ya sean temporales o fijos) que se emplean en las vistas. Aquí están los ficheros CSS y JavaScript que permiten una mejor visualización de las vistas, las imágenes empleadas en las vistas también, los pesos para emplear la función detect de YOLOv5 para detectar los alimentos de la imagen subida o dichas imágenes guardadas temporalmente para poder procesarlas.

7 Resultados

En este apartado trataremos de exponer los resultados obtenidos de dos partes importantes de la aplicación. Primero se analizarán los resultados obtenidos del entrenamiento de varios modelos de YOLOv5 con diferentes épocas y luego se expondrá el resultado final de la aplicación con diferentes capturas de pantalla y en qué consiste cada fase.

7.1 Resultado de entrenamientos YOLOv5

A continuación, se realizará un estudio de todas las pruebas realizadas con los modelos de YOLOv5 *S*, *M* y *L* y el motivo de por qué se fueron reduciendo las épocas. De todas las gráficas y resultados de entrenamiento y validación obtenidos, nos centraremos en observar la gráfica de validación "val/obj_loss" ya que es la que indica si existe sobreentrenamiento. Para detectar esto se puede observar como a partir de X época, el error comienza a aumentar. Este sería un ejemplo de las gráficas obtenidas de un entrenamiento de 500 épocas de YOLOv5:

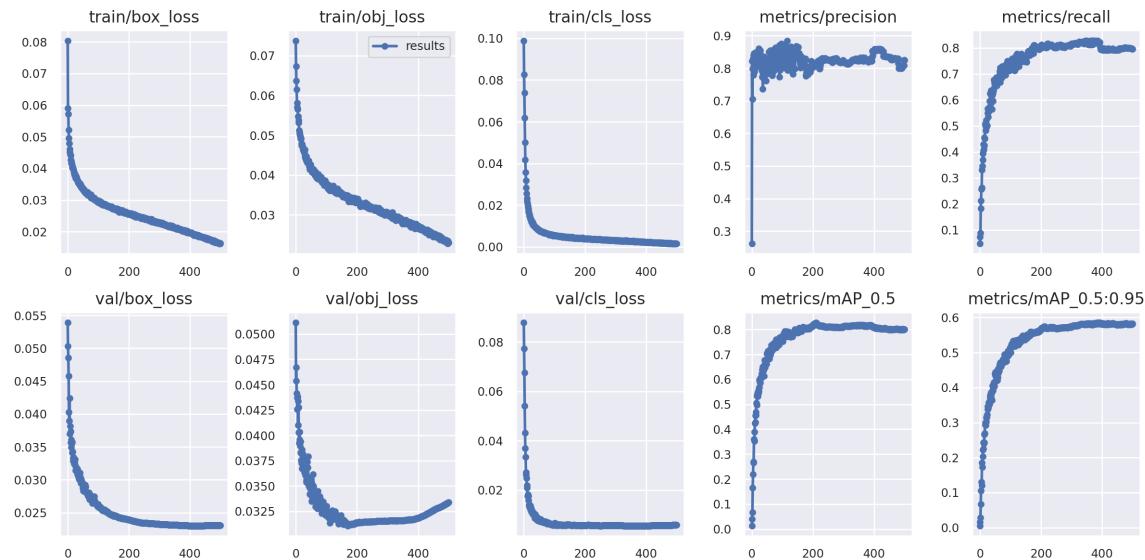


Figura 7.1: Ejemplo de los resultados de entrenamiento de YOLOv5

A partir de aquí, he agrupado las gráficas "val/obj_loss" de los resultados en los diferentes modelos empleados con distintas épocas. Se han hecho pruebas con los modelos "YOLOv5s6", "YOLOv5m6" y "YOLOv5l6". Estos serían todos los modelos junto con su eficiencia:

Se ha descartado el modelo "YOLOv5n6" ya que su velocidad es muy lenta y no se obtienen resultados relevantes para nuestro proyecto. Por otra parte, el modelo "YOLOv5x6" no se

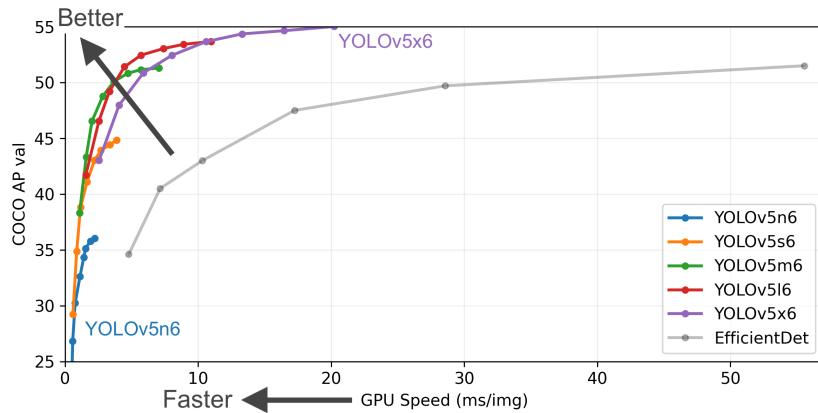


Figura 7.2: Modelos de YOLOv5

podía ejecutar en el servidor anteriormente mencionado aportado por el departamento por un tema de infraestructura de las GPUs. Por ello, se han realizado las siguientes pruebas en los tres modelos restantes. Comenzaremos analizando los resultados del modelo "YOLOv5s":

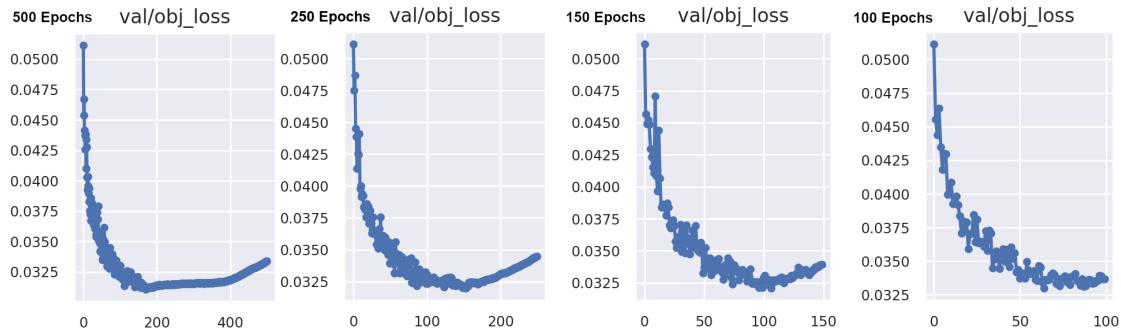


Figura 7.3: Resultados con modelo YOLOv5s y diferentes épocas

En un primer lugar y dado que disponemos de bastante capacidad de procesamiento para poder entrenar los modelos, se obtó por entrenar con 500 épocas. Esto fue una mala idea ya que, como se puede observar, a partir de la época 200 aproximadamente aumenta de forma notoria el error, lo que provoca el sobreentrenamiento de la red.

Más tarde, se hizo otro entrenamiento con 250 épocas, sacando en claro que el punto en el que la red comenzaba a sobreentrenar era la época 150.

Por último, se optó por realizar otro entrenamiento con 150 épocas y, otra vez más, existía sobreentrenamiento a partir de la época 100 aproximadamente. Esto causó que se hiciera otra prueba de entrenamiento con 100 épocas, en la cual ya no existe el mismo problema. A partir de aquí, se tomó la decisión de hacer dos entrenamientos en los siguientes modelos con solamente 100 y 150 épocas. Los siguientes entrenamientos a analizar son los obtenidos del modelo "YOLOv5m":

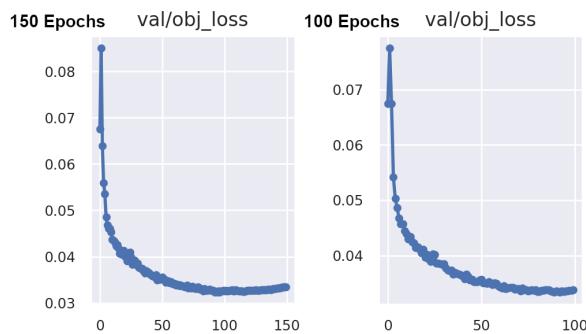


Figura 7.4: Resultados con modelo YOLOv5m y diferentes épocas

Analizando los resultados del modelo "YOLOv5m" podemos observar como ambas son muy similares aunque tienen sutiles diferencias. Primero se aprecia un ligero sobre entrenamiento en los resultados de 150 épocas pero también una ligera reducción del error obtenido. Respecto a la gráfica de las 100 épocas, no tiene sobre entrenamiento pero el error aumenta ligeramente.

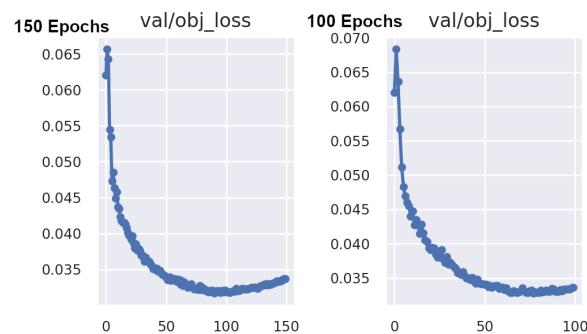


Figura 7.5: Resultados con modelo YOLOv5l y diferentes épocas

Por último, se hicieron ambas pruebas con el modelo "YOLOv5l" con las mismas épocas que con el modelo anterior. Los resultados sob prácticamente los mismos que con el modelo "YOLOv5m" pero hay un ligero sobre entrenamiento en ambas gráficas. Ahora la de 100 épocas aumenta su error ligeramente entorno a la época 70/80.

Una vez se han analizado las diferentes pruebas obtenidas del entrenamiento de distintos modelos con números de épocas dispares, se realizaron pruebas visuales para observar cuál de ellos funciona mejor con una imagen completamente nueva. A groso modo, se introdujeron 5 imágenes nuevas, indicando visualmente cuántos alimentos hay en su interior. Luego se ejecutaron uno por uno los modelos detectando los alimentos que hay en el interior y, se realizó una comparativa de todos. Una vez analizadas las observaciones, se eligió el modelo que mejor funcionaba en todas las imágenes, el cual era el modelo "YOLOv5m" entrenado con 150 épocas. El resto de modelos entrenados tenían diferentes problemas, como que detectaban alimentos que no había en la nevera o no podían reconocer algunos que eran bastante obvios. Por ejemplo, se analizaba la siguiente imagen y se mostraban en una tabla qué alimentos hay junto con su cantidad:



Figura 7.6: Ejemplo de imagen de nevera

Estos serían los resultados de todos los modelos de la imagen anterior. Como se puede observar, existen ligeras diferencias entre los resultados de un modelo y otro:



Figura 7.7: Resultado de la imagen con el modelo "YOLOv5l" con 100 épocas.



Figura 7.8: Resultado de la imagen con el modelo "YOLOv5lm" con 100 épocas.



Figura 7.9: Resultado de la imagen con el modelo "YOLOv5s" con 100 épocas.



Figura 7.10: Resultado de la imagen con el modelo "YOLOv5l" con 150 épocas.



Figura 7.11: Resultado de la imagen con el modelo "YOLOv5m" con 150 épocas.



Figura 7.12: Resultado de la imagen con el modelo "YOLOv5s" con 150 épocas.



Figura 7.13: Resultado de la imagen con el modelo "YOLOv5s" con 500 épocas.

El siguiente paso es contar los alimentos que hay en el interior y realizar la detección de los alimentos con cada uno de los modelos entrenados los cuales se corresponden con las imágenes anteriores, obteniendo la siguiente tabla, donde la columna "REAL" representa los alimentos que hay realmente en la foto, siendo las siguientes columnas lo que cada modelo detecta:

IMAGEN PRUEBA	Modelos							
	REAL	100s	100m	100l	150s	150m	150l	500s
Patatas	7	9	3	4	9	7	8	9
Tomates	10	11	8	5	8	5	5	9
Guisantes	0	1	1	2	0	0	0	0
Naranjas	4	4	5	7	5	3	6	3
Limones	6	14	8	10	13	8	10	10
Fresas	8	7	7	9	10	13	9	10
Berenjenas	2	1	2	1	1	2	3	2
Perejil	2	2	1	1	2	1	1	2
Pimiento verde	2	1	2	3	3	5	5	2
Huevos	7	4	5	4	4	6	5	5
Manzanas	4	2	2	1	1	3	1	1
Leche	6	4	3	6	4	4	3	3
Pimiento rojo	2	1	2	4	2	2	2	2
Lechuga	2	1	1	1	1	1	1	1
Brocoli	1	1	2	1	1	1	1	1
Calabacín	0	0	0	0	1	0	1	0
Zanahoria	9	0	0	0	1	0	0	0
Judias verdes	0	0	0	0	1	0	0	0
Cebolletas	0	0	0	0	0	0	0	1
Platanos	0	0	1	0	0	0	0	0
Cebolla roja	0	0	2	0	0	0	1	0
Embutido	0	0	0	2	0	0	0	0
Uvas	0	0	0	0	0	2	0	0
Colifor	1	0	0	0	0	0	0	0
Aqua	9	0	0	0	0	0	0	0

IMAGEN PRUEBA	Modelos							
	REAL	100s	100m	100l	150s	150m	150l	500s
Yogurth	2	0	0	0	0	0	0	0
Carne	2	0	0	0	0	0	0	0
COINCIDENCIAS	-	17	15	14	15	19	14	17

Cuadro 7.1: Observación de resultados de cada modelo sobre una imagen

Tomando este ejemplo, nos fijamos en el número de patatas que hay, el modelo que obtiene el número que debería existir es el "YOLOv150m". Para comprender mejor esto, ahora elegimos el número de huevos detectados. El número real es 7 pero ningún modelo ha detectado tantos, el que más se acerca en este caso vuelve a ser el mismo modelo. Se realiza esta acción con cada alimento y finalmente se contabilizan el número de coincidencias entre la observación real y cada modelo. En este caso el modelo que más se ajusta a la realidad sería el modelo "YOLOv150m" el cual tiene 19 coincidencias respecto a los alimentos reales y detectados(o es el que más se acerca en según que alimentos). El resto de modelos tienen un número menor de coincidencias, por tanto el elegido en esta imagen es el modelo "YOLOv150m".

Esta misma prueba se ha realizado con 5 imágenes de prueba diferentes, en las cuales el modelo elegido en 3 de ellas es el anteriormente mencionado, quedando segundo en las otras dos imágenes restantes. Por ello, podemos decir que el modelo que más se ajusta es el "YOLOv150m".

7.2 Aplicación Web

Aquí vamos a abordar el resultado final de la aplicación en general, viendo y contrastando con diferentes imágenes de nevera cómo funciona la aplicación y sus diferentes fases.

En primer lugar tendríamos el Inicio de la aplicación, el cual contiene una imagen de bienvenida, con una pequeña descripción del proyecto y un esquema muy resumido del funcionamiento del mismo. Desde aquí, podemos pasar a la primera fase que sería la de subir una imagen desde el menú superior o desde el botón del final:

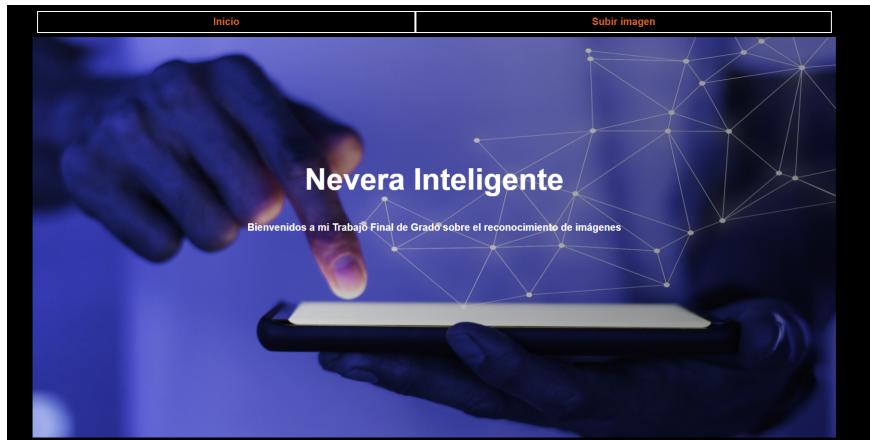


Figura 7.14: Página de inicio de la aplicación

Tras esto, ahora nos aparece una página en la que podremos subir una imagen. Si estamos en un dispositivo móvil, nos dará la opción de hacer una fotografía con la cámara del móvil y poder así subirla. Cuando la imagen se suba (solo puedes subir en formato .png, .jpg y .jpeg), aparecerá el nombre del archivo que se ha subido y ya se puede dar a "Enviar imagen". Tras esto, la imagen se procesará con el modelo ya entrenado (YOLOv5) y tras unos segundos pasará al apartado de resultados. Esta sería la página mientras la imagen se está procesando:



Figura 7.15: Página para subir una imagen y procesarla

En la página de resultado se pueden observar varias cosas, en el lado izquierdo se pueden observar un resumen con todos los alimentos que se han detectado. En el lado derecho está la imagen ya procesada por YOLOv5 en la cual aparecen marcados con un recuadro de un

color (*Bounding Boxes*) lo que se detecta en concreto y un número del 0 al 1 el cual indica la seguridad de que el objeto detectado sea correcto. Esto quiere decir que cuanto más cerca al 1, más seguro está el algoritmo de haber detectado bien el alimento. Por otra parte, aparecen 3 botones con los que podemos elegir 5, 10 y 20 recetas para enviar a ChatGPT. Tras pulsar uno de los tres botones, se enviará un mensaje a la API de ChatGPT para que procese los alimentos y una vez responda con las respuestas se pasará a la siguiente página. A continuación, se puede ver una captura de pantalla de un ejemplo de la página de resultados:



Figura 7.16: Página de resultados del reconocimiento de alimentos.

Ahora el usuario tendrá que elegir si quiere 5, 10 o 20 recetas pulsando cualquiera de los 3 botones. Como se ha explicado anteriormente, se enviará una petición mediante la API de ChatGPT para que nos indique las X posibles recetas que podemos realizar. Otra funcionalidad extra que tiene esta última página es la de poder solicitar otras 10 recetas un poco más elaboradas pulsando un botón. Esta sería la página final donde encontraríamos a la izquierda los alimentos y a la derecha la respuesta obtenida:

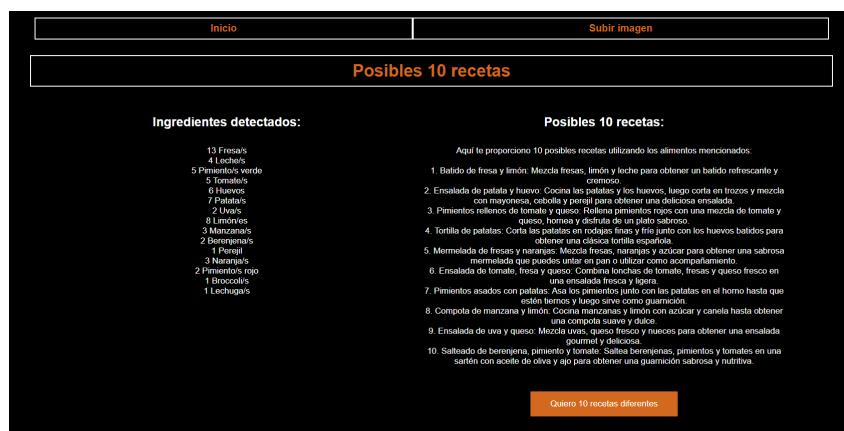


Figura 7.17: Página de X posibles recetas obtenidas.

8 Conclusiones

Para concluir este proyecto, podemos volver a ver qué objetivos se han cumplido en su totalidad y de la forma deseada y cuales no del todo o han existido problemas.

Para empezar, el empleo de YOLOv5 nos ha permitido profundizar en el reconocimiento de imágenes. En este caso han sido alimentos dentro de una nevera pero se puede reconocer prácticamente cualquier objeto con los datasets y pruebas adecuados. Después, se ha empleado el modelo de lenguaje de IA ChatGPT para poder obtener las recetas que deseasemos. Estos modelos de lenguaje son muy potentes y tienen prácticamente infinitas posibilidades. Por último y para poder unificar ambas partes, se ha empleado Flask para realizar una aplicación web multidispositivo y responsive para cualquier usuario.

Con todo esto, el resultado final ha sido prácticamente el deseado excepto por dos importantes inconvenientes. El primero sería el propio reconocimiento de los alimentos, los cuales tienen que estar a la vista para que el modelo pueda reconocerlos de forma fácil y, aún así, no los termina de reconocer bien del todo (en el vídeo de demostración ocurre esto). El segundo es la amplia gama de recetas que el modelo de lenguaje nos proporciona, algunas pueden no resultar agradables para el usuario. Esto se ha intentado solucionar poniendo el botón del final el cual permite conocer otras 10 recetas extras.

La idea inicial era original e innovadora y el resultado ha sido el esperado. No obstante, si se desea seguir desarrollando la idea de manera más profesional, se tendrían que pulir ciertos aspectos de la aplicación. Se pueden hacer un sin fin de mejoras, las cuales he querido agrupar en 3 grandes bloques:

1. Mejorar la propia aplicación web de forma visual y añadiendo más funcionalidades y contenido, como que el usuario pueda modificar manualmente los alimentos eliminando, cambiando o añadiendo.
2. Mejorar el reconocimiento de imágenes probando otros modelos y comparándolos con los resultados obtenidos.
3. Cambiando la forma de extraer las recetas utilizando por ejemplo un dataset de recetas u otra tecnología.

Por último, aquí estaría el vídeo del funcionamiento de la aplicación web donde se puede observar su correcto funcionamiento tanto en un ordenador como en un caso real con un dispositivo móvil. Se ha empleado Clideo (2023) para juntar ambos vídeos: <https://youtu.be/STU6D11AzGs>.

Bibliografía

- Aguinaco, U. (2023). *"Dataset 4 empleado para el dataset final"*. Descargado de <https://universe.roboflow.com/unai-aguinaco/gh-xmjfz/dataset/1>
- aicook. (2022). *"Dataset 1 empleado para el dataset final"*. Descargado de <https://universe.roboflow.com/aicook/aicook-self-annotated/dataset/4>
- Api de chatgpt.* (s.f.). <https://openai.com/>. (Consultado el 27 de junio de 2023)
- Bosch. (2023). *"Frigorífico inteligente de Bosch"*. Descargado de <https://www.bosch-home.es/especiales/electrodomesticos-inteligentes-home-connect/frigorificos?contentType=products>
- Calvo, D. (2017). *"Clasificación de las redes neuronales artificiales"*. Descargado de <https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/>
- Chatgpt.* (s.f.). <https://openai.com/chatgpt>. (Consultado el 27 de junio de 2023)
- Clideo. (2023). *"Software para juntar vídeos"*. Descargado de <https://clideo.com/es>
- Eswitha_reddy. (2020). *"Types of Boltzmann Machines"*. Descargado de <https://www.geeksforgeeks.org/types-of-boltzmann-machines/>
- García, L. M. (2021). *"Viabilidad y rendimiento de YOLOv5 en Raspberry Pi 4 modelo B"*. Descargado de <https://idus.us.es/bitstream/handle/11441/126961/TFG-3731-MUÑIZGARCIA.pdf>
- JIMÉNEZ, T. M. V. (2023). *"EVALUACIÓN DE RENDIMIENTO DE YOLOV5 Y ALGORITMOS DE SEGUIMIENTO EN UNA JETSON NANO 2GB"*. Descargado de <https://repositorio.uchile.cl/bitstream/handle/2250/191847/Evaluacion-de-rendimiento-de-YOL0v5-y-algoritmos-de-seguimiento-en-una-Jetson-Nano-2GB.pdf?sequence=1>
- Jocher, G. (2023). *"Código base de YOLOv5 en Github"*. Descargado de <https://github.com/ultralytics/yolov5>
- KeepCoding Tech School. (2022). *"Aplicación de detección de los alimentos en tu nevera"*. Descargado de <https://youtu.be/qqiIf-NtZXE>
- LG presenta un nuevo frigorífico inteligente con Alexa.* (s.f.). <https://www.proandroid.com/lg-nuevo-frigorifico-inteligente-alexa/>. (Consultado el 20 de junio de 2023)
- Martínez, L. R. (2023). *"Vídeo del funcionamiento de la aplicación web"*. Descargado de <https://youtu.be/STU6D11AzGs>

- Matima, T. (2022). *"Dataset 3 empleado para el dataset final"*. Descargado de <https://universe.roboflow.com/tk-matima-unqyz/food-in-fridge-2slx4/dataset/1>
- ProAndroid. (2017). *"Frigorífico inteligente de LG"*. Descargado de <https://www.proandroid.com/lg-nuevo-frigorifico-inteligente-alexa/>
- Roboflow. (2023). *"Lugar de obtención de los datasets"*. Descargado de <https://roboflow.com/>
- Ronacher, A. (2023). *"Documentación de Flask"*. Descargado de <https://flask.palletsprojects.com/en/2.3.x/>
- Samsung. (2023). *"Frigorífico Family Hub de Samsung"*. Descargado de <https://www.samsung.com/es/refrigerators/side-by-side/rs8000nc-side-by-side-refrigerator-with-family-hub-rs8000nc-side-by-side-refrigerator-with-family-hub-614l-silver-rs6ha8880s9-ef/>
- SmarterChef. (2023). *"Dataset 2 empleado para el dataset final"*. Descargado de <https://universe.roboflow.com/masterchef/smarterchef/dataset/5>
- Swapna. (2020). *Convolutional Neural Network (CNN)*. Descargado de <https://developersbreach.com/convolution-neural-network-deep-learning/>
- Wikipedia. (2022). *Frank Rosenblatt*. Descargado de https://es.wikipedia.org/wiki/Frank_Rosenblatt
- Wikipedia. (2023). *Perceptrón*. Descargado de <https://es.wikipedia.org/wiki/Perceptrón>

Lista de Acrónimos y Abreviaturas

AAS	Australian Acoustical Society.
ADAA	Asociación de Acústicos Argentinos.
AES	Audio Engineering Society.
APA	American Psychological Association.
ASA	Acoustical Society of America.
CNN	Red Neuronal Convolucional.
CNNs	Redes Neuronales Convolucionales.
CSIC	Consejo Superior de Investigaciones Científicas.
EAA	European Acoustics Association.
I-INCE	International Institute of Noise Control Engineering.
IA	Inteligencia Artificial.
ICA	International Congress on Acoustics.
IEEE	Institute of Electrical and Electronics Engineers.
IIAV	International Institute of Acoustics and Vibration.
IOA	Institute Of Acoustics.
ISRA	International Symposium on Room Acoustics.
ISVA	International Seminar on Virtual Acoustics.
SEA	Sociedad Española de Acústica.
TFG	Trabajo Final de Grado.
TFM	Trabajo Final de Máster.