



UNIVERSIDAD DE SEVILLA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO DE TECNOLOGÍAS INFORMÁTICAS

SISTEMA DE AYUDA EN LA TOMA DE DECISIONES PARA PERSONAS INTOLERANTES MEDIANTE TÉCNICAS DE INTELIGENCIA ARTIFICIAL

Realizado por

Juan Miguel Moreno Valero

Dirigido por

Belén Vega Márquez

Departamento

Lenguajes y Sistemas Informáticos

Resumen

Se realiza la entrega de la memoria del trabajo de fin de grado, cuyo objetivo final es un proyecto de investigación para posibilitar un sistema de ayudas a personas con intolerancia alimenticia, mediante dos tipos de clasificaciones: clasificación multiclase, para categorizar los diferentes platos de comida, y clasificación multilabel, para la detección de los ingredientes por plato, en la que los fallos acontecidos han privado de unos resultados más extensos. Además, se busca una comparación entre la capacidad de eficacia que pueda proporcionar una red neuronal convolucional pre entrenada frente a la que puede conseguir una red implementada totalmente por nuestra parte. Sin embargo, como ya se expondrá más adelante, los fallos producidos en la clasificación multilabel han delimitado el contenido del trabajo.

El proyecto nace como ambición personal de llevar a cabo una inmersión en el apabullante mundo del Deep learning. Además, se complementa mi iniciativa en este trabajo debido a mi relación con personas intolerantes a ciertos alimentos que se incluyen en mi círculo cercano, que pueden verse beneficiadas por este proyecto para mejorar su calidad de vida.

Para su implementación y desarrollo, se ha hecho uso de la herramienta de Google Colab ya que, entre sus otras características, facilitaba un entorno de computación óptimo que, junto a las librerías facilitadas por Python, como scikit-learn, matplotlib y keras, entre otras, ayudaría a trabajar de forma más eficiente sobre los datos escogidos y en la implementación de un modelo de clasificación para predecir los ingredientes. Gracias a las técnicas de Deep Learning, basando el modelo en redes neuronales convolucionales (CNN), se consigue desarrollar el proyecto logrando un grado de satisfacción acorde a los recursos se disponen.

Espero que este proyecto sirva, a parte de una riqueza personal en el ámbito de la inteligencia artificial, como un inicio a una futura ampliación del mismo. Posibles proyectos a incluir en un futuro serían la creación de una aplicación que pudiese recoger instantáneas de la comida tomadas por el propio usuario, para poder detectar así los ingredientes que lo componen. Debido a que no existe ninguna aplicación similar en el mercado, donde la competencia serían aplicaciones que escanean los códigos de los productos para identificar los ingredientes, se estaría consiguiendo una doble función: un nicho de mercado poco potenciado y bastante solicitado, ya que el 25% de la población mundial sufre de alguna intolerancia, y la posibilidad de una mayor expansión para servir a más individuos una ayuda para conseguir la identificación de aquellos ingredientes que suponen un riesgo para su salud.

Abstract

The end-of-degree project report is delivered, whose final objective is a research project to enable a system of aid for people with food intolerance, through two types of classifications: multiclass classification, to categorize the different dishes of food, and multilabel classification, for the detection of the ingredients per dish. In addition, a comparison is sought between the efficiency capacity that a pre-trained convolutional neural network can provide versus the one that a network fully implemented by us can achieve. However, as will be explained later, the failures produced in the multilabel classification have delimited the content of the work. The project was born as a personal ambition to carry out an immersion in the overwhelming world of Deep learning. In addition, I complement my initiative in this work due to my relationship with people who are intolerant to certain foods that are included in my close circle, who can benefit from this project to improve their quality of life.

For its implementation and development, we have made use of the Google Colab tool, since, among its other characteristics, it provided us with an optimal computing environment, which, together with the libraries provided by Python, such as scikit-learn, matplotlib and Pandas, among others, it would help us to work more efficiently on the chosen data and in the implementation of a classification model to predict our ingredients. Thanks to Deep Learning techniques, basing our model on convolutional neural networks (CNN), we have managed to develop the project achieving a degree of satisfaction according to the resources available to us.

I hope that this project will serve, apart from personal wealth in the field of artificial intelligence, as a start to a future extension of it. Possible projects to include in the future would be the creation of an application that could collect snapshots of food taken by the user himself, in order to detect the ingredients that make it up. Since there is no similar application on the market, where the competition would be applications that scan the product codes to identify the ingredients, we would be achieving a double function: a little promoted and highly requested market niche, since 25% of the world population suffers from some intolerance, and the possibility of further expansion to serve more individuals helps to identify those ingredients that pose a risk to their health.

Agradecimientos

Principal agradecimiento a mi familia, que son los pilares que siempre sostienen a la persona que soy hoy en día, cuyo esfuerzo continuo me brinda la oportunidad de poder llevar a cabo mis estudios y formarme académica y socialmente.

Referido también a aquellos profesores cuya devoción por la enseñanza hace que el aprendizaje de su asignatura impartida sea mucho más completo, motivándote a seguir en el proceso de formación. Desde secundaria hasta los docentes de la universidad, muy agradecido.

Para todo ese grupo de amistades, que son quienes te acompañan tanto en los buenos como en los adversos momentos de este camino, por haberme hecho disfrutar de mi etapa universitaria y haber sabido facilitarme la compaginación entre estudios y ocio.

Por último, especial mención a mi tutora, Belén Vega Márquez, por su compromiso con este proyecto, así como la disponibilidad, esfuerzo y amabilidad que ha presentado durante el desarrollo del mismo.

Índice general

Resumen	2
Abstract	3
Agradecimientos	4
Índice de tablas	7
Índice de ilustraciones	8
1. Introducción	10
1.1. Motivación	10
1.2. Objetivos del proyecto	10
1.3. Análisis de requisitos	11
2. Planificación	12
2.1 Fases	12
2.2 Planificación temporal	13
2.3 Diagrama de Gantt	16
2.4 Estimación de costes	19
2.4.1 Costes del personal	19
2.4.2 Costes de hardware	19
2.4.3 Costes de software	20
2.4.4 Costes indirectos	20
2.4.5 Coste total	20
2.5 Gestión de riesgos	20
2.5.1 Identificación de los riesgos	21
2.5.2 Plan de contingencia	21
3 Contexto del proyecto	23
3.1 Concienciación del problema	23
3.2 Inteligencia Artificial	27
3.3 Deep Learning	28
3.4 Redes neuronales	29
3.5 Tipos de funciones de activación	31
3.6 Tipos de redes neuronales	33
3.7 Redes neuronales convolucionales	34
4 Metodología	38
4.1 Tecnologías y herramientas empleadas	38
4.1.1 Lenguaje de programación	38
4.1.2 Entorno de desarrollo	38

4.1.3	Librerías utilizadas	40
4.2	Preprocesamiento y adaptación del dataset	50
4.2.1	Análisis del dataset	50
4.2.2	Preprocesamiento	53
4.2.3	Adaptación del dataset	54
4.3	Aspectos generales de los modelos de clasificación	43
4.3.1	Clasificación multiclase y clasificación multilabel	44
4.3.2	Construcción base de los diferentes modelos de clasificación	45
4.3.3	Métricas empleadas	49
5	Resultados	56
5.1	Parámetros prefijados para aceptar el modelo	56
5.2	Desarrollo de los diferentes modelos. Clasificación multiclase	56
5.2.1	Un modelo de clasificación multiclase limitando el número de platos a clasificar	56
5.2.2	Un modelo de clasificación multiclase limitando el número de imágenes por plato a clasificar	66
5.2.3	Un modelo de clasificación multiclase donde la estructura de la red neuronal convolucional es creada por nuestra parte.	73
5.2.4	Conclusión obtenida	82
5.3	Desarrollo de los diferentes modelos. Clasificación multilabel	83
5.3.1	Un modelo de clasificación multilabel de los ingredientes mediante una red neuronal convolucional pre entrenada.	83
6	Problemas encontrados	92
7	Conclusiones	94
8	Proyectos futuros	95
9.	Bibliografía	98

Índice de tablas

Tabla 1. Tareas correspondientes al proceso de iniciación	14
Tabla 2. Tareas correspondientes al proceso de planificación	14
Tabla 3. Tareas correspondientes a la fase de ejecución, seguimiento y control.....	15
Tabla 4. Tareas correspondientes a la fase de cierre.....	15
Tabla 5. Representación de la duración por fase y la duración total del proyecto. Incluye el desfase de la duración real con respecto a la duración estimada.....	16
Tabla 6. Costes de personal.....	19
Tabla 7. Coste total.....	20
Tabla 8. Representación gráfica de las principales funciones de activación.....	33
Tabla 9. Tecnologías empleadas.....	43
Tabla 10. Parámetros para determinar el modelo 1 de clasificación multiclase	57
Tabla 11. Resultados de métricas del modelo 1 de clasificación multiclase	65
Tabla 12. Parámetros para determinar el modelo 2 de clasificación multiclase	67
Tabla 13. Resultados de las métricas del modelo 2 de clasificación multiclase	72
Tabla 14. Elección de los parámetros para seleccionar el modelo 3 de clasificación multiclase	73
Tabla 15. Métricas del 3 modelo de clasificación multiclase	81
Tabla 16. Métricas del 4 modelo de clasificación multiclase	82
Tabla 17. Comparativa final de los resultados en clasificación multiclase	83

Índice de ilustraciones

Figura 1. Diagrama EDT de tareas.	13
Figura 2. Creación de la organización de las tareas.....	17
Figura 3. Fases y marcos mensuales del proyecto.....	17
Figura 4. Diagrama de Gantt. Primera imagen.	18
Figura 5. Diagrama de Gantt. Segunda imagen.	18
Figura 6. Adultos con alergias alimentarias en 2015. (MÉDICA, 2019)	24
Figura 7. Alergias por comunidades autónomas en el año 2015 y 2005. (González, 2015).....	25
Figura 8. Prevalencia a padecer alergia por grupos de edad. Años 2015 y 2005. (González, 2015)	26
Figura 9. Signos y síntomas de sospecha según grupo de edad. (González, 2015).....	26
Figura 10. Inteligencia Artificial. Machine Learning. Deep Learning. (Cloud, s.f.).....	28
Figura 11. Ilustración de una neurona biológica. (IBM, ¿Qué son las redes neuronales?, s.f.)...	29
Figura 12. Representación de una neurona artificial. (IBM, ¿Qué son las redes neuronales?, s.f.)	30
Figura 13. Ilustración de una red neuronal. Perceptrón multicapa. (IBM, ¿Qué son las redes neuronales?, s.f.)	31
Figura 14. Redes recurrentes (RNN). (IBM, ¿Qué son las redes neuronales?, s.f.).....	34
Figura 15. Características extraídas mediante CNN. (IBM, Redes Neuronales Convolucionales, s.f.)	35
Figura 16. MaxPooling. (IBM, Redes Neuronales Convolucionales, s.f.)	35
Figura 17. AveragePooling. (IBM, Redes Neuronales Convolucionales, s.f.)	36
Figura 18. GlobalAveragePooling. (IBM, Redes Neuronales Convolucionales, s.f.).....	36
Figura 19. Red neuronal convolucional (CNN). (IBM, Redes Neuronales Convolucionales, s.f.) .	37
Figura-20. Esquema de CNN de nuestro problema.	37
Figura 21. 101 imágenes que contiene nuestro dataset.	51
Figura 22. Representación de 15 clases, por simplificar.....	52
Figura 23. 5 ingredientes de la lista de ingredientes sin simplificar.	52
Figura 24. Los mismos 5 ingredientes simplificados.	53
Figura 25. Se muestra sólo los primeros 4 elementos para poder apreciar las dimensiones.	54
Figura 26. Método holdout modelo 1 de clasificación multiclase.	59

Figura 27. Estructura de CNN en el modelo 1 de clasificación multiclase.	60
Figura 28. Generadores modelo 1 de clasificación multiclase.	61
Figura 29. Método fit() modelo 1 de clasificación multiclase.	62
Figura 30. Entrenamiento del modelo 1 de clasificación multiclase.	63
Figura 31. Visualización modelo 1 de clasificación multiclase.	64
Figura 32. Predicciones modelo 1 de clasificación multiclase.	65
Figura 33. Métricas del modelo 1 de clasificación multiclase	66
Figura 34. Generadores modelo 2 de clasificación multiclase.	69
Figura 35. Método fit() del modelo 2 de clasificación multiclase.	69
Figura 36. Entrenamiento modelo 2 de clasificación multiclase.	70
Figura 37. Visualización del modelo 2 de clasificación multiclase.	71
Figura 38. Predicciones del modelo 2 de clasificación multiclase.	72
Figura 39. Resultados de las métricas del modelo 2 de clasificación multiclase	73
Figura 40. CNN creada por nuestra parte.	78
Figura 41. Entrenamiento del modelo 3 de clasificación multiclase.	79
Figura 42. Gráfica de resultados del modelo 3 de clasificación multiclase	80
Figura 43. Predicciones del modelo 3 de clasificación multiclase.	80
Figura 44. Estructura de la segunda CNN implementada en clasificación multiclase.	81
Figura 45. Método holdout modelo 1 de clasificación multilabel.	86
Figura 46. Estructura CNN del modelo 1 de clasificación multilabel.	88
Figura 47. Método fit() modelo 1 de clasificación multilabel.	89
Figura 48. Resultados de entrenamiento del modelo de clasificación multilabel.	90
Figura 49. Visualización del modelo de clasificación multilabel.	90

1. Introducción

El presente documento recoge la memoria del trabajo de fin de grado de un proyecto basado en el reconocimiento de ingredientes mediante multilabel learning, aunque por problemas en sus resultados, la base ha sido trasladada a la clasificación multiclase de los diferentes platos de comida. La aplicación del mismo, llevado a cabo mediante técnicas de Deep Learning, será enfocada a posibilitar una ayuda a toda aquella persona que pueda sufrir de un trastorno alimentario.

Actualmente, existen varias aplicaciones que, mediante el escaneo del código del producto a determinar por el usuario, te permite una visualización de los diferentes ingredientes que componen este artículo alimenticio. Sin embargo, nuestra propuesta abarca una idea más ambiciosa e innovadora: la posibilidad de detectar directamente la composición de ingredientes que forman el plato de comida que una persona pretendiese analizar.

Con el desarrollo del proyecto se sientan las bases de esta idea propuesta, cuya única limitación para llevar a cabo la completitud del trabajo sería una mayor capacidad de recursos, consiguiendo así los tiempos de cómputo para poder enfocar los resultados en un espacio de tiempo razonable y ampliar nuestros datos en forma de nuevos platos e ingredientes que clasificar. Debido a que este proyecto se enmarca dentro de los recursos facilitados por la herramienta de Google Colab, dotada de acelerador GPU/TPU de 15 GB de memoria RAM y de un espacio de disco de 166 GB, ha debido adaptarse a los intereses de la viabilidad del entorno de desarrollo.

1.1. Motivación

La motivación inicial de este proyecto fue la búsqueda personal de un acercamiento al mundo de la inteligencia artificial, concretamente al Deep Learning y al uso de las redes neuronales, que tan de moda están hoy en día. Para la puesta en marcha de mi deseo, mi tutora, Belén Vega Márquez, presenta la propuesta que estaría dispuesta a ofrecer y consigue despertar mi atención hacia la idea de la quería hacerme formar parte.

A raíz de esto, nace mi segunda motivación por este trabajo: conseguir brindar una ayuda en la manera de lo posible a aquellos individuos que sufrían de un trastorno alimentario. La relación cercana con personas con este problema, que han sufrido los síntomas de una reacción alérgica tras desconocer los ingredientes de su comida, situación acentuada debido a que nuestra experiencia se estaba desarrollando en un país extranjero, me incita a tratar, dentro de los recursos de los que dispongo, de buscar la mejor solución posible para evitar casos similares que afecten a la salud de personas con el mismo trastorno.

1.2. Objetivos del proyecto

El objetivo principal de este proyecto era conseguir la máxima eficacia y precisión a la hora de clasificar correctamente el mayor número de ingredientes por plato, en el mayor número de platos diferentes posibles, dentro de las capacidades de las que se disponía. Sin embargo, se cambió el rumbo en la hoja de ruta marcada para llevar a cabo la mejor clasificación posible de los diferentes platos de comida. Además, se busca comprobar la diferencia de rendimientos

entre los resultados obtenidos por la implementación de una de las redes pre entrenadas, que a priori debería devolver resultados óptimos, frente a los que posibles obtenidos mediante una construcción autónoma y completa de una red neuronal convolucional por nuestra parte.

Con el fin de asegurar el mayor rendimiento, se lleva a cabo varios modelos, así como diferentes enfoques para cada uno de ellos, para conseguir obtener el mejor resultado posible y, de esta manera, evidenciar que la aproximación al mejor modelo ha sido la más adecuada posible.

A su vez, mencionar de nuevo que el objetivo fundamental del proyecto es conseguir brindar la mejor ayuda que esté a mi alcance para la población que sufre de un trastorno alimentario, o por lo menos conseguir fundar las bases de cara a una ampliación futura que pudiera llegar a un público más extenso.

1.3. Análisis de requisitos

Para poder establecer una línea firme a seguir a lo largo del desarrollo del proyecto, se debe tener claro cuáles son los principales requisitos necesarios para dirigir los objetivos hacia la meta final. Estos requisitos se muestran a continuación:

R01.- Diversidad de los platos e ingredientes por plato. Representación visual de los diferentes platos que se disponen para su análisis, así como los ingredientes que componen cada plato, para ser conscientes en todo momento y no perder de vista el problema que se está relacionando en cada momento.

R02.- Preprocesamiento necesario. Adaptación de las características del dataset a los recursos que se disponen, de forma que se pueda conseguir la intersección más óptima de ambos conjuntos para el mejor resultado posible.

R03.- Entrenamiento de los diferentes modelos. Construcción de los distintos modelos que constituirán el proyecto, así como ser dotados con varias posibilidades de entrenamiento que enriquezcan las alternativas de estos.

R04.- Valoraciones. Representación de los resultados de los modelos obtenidos con las diferentes técnicas que se apliquen a ellos, además de mostrar una representación gráfica de las principales métricas que se han tenido en cuenta para su evaluación, de forma que sea más liviano su entendimiento a partir de una imagen comparativa.

R05.-Comparativa entre redes pre entrenadas y red autónomas creadas. Comparación entre las prestaciones y resultados obtenidos entre las redes entrenadas previamente y una red que sería construida e implementada por nuestra propia cuenta.

2. Planificación

Este apartado será aprovechado para tratar los puntos temporales en los que se dividirá el proyecto, así como una estimación de los diferentes costes que envuelven al trabajo y la realización de una gestión posible de los riesgos que pueden aparecer durante el desarrollo del proyecto.

2.1 Fases

El proyecto se verá dividido en cuatro partes diferenciadas, las cuales se comentan a continuación.

En la primera fase, iniciación, se plantea llevar a cabo una serie de reuniones para determinar cuál será finalmente el proyecto que más puede satisfacer las ambiciones que se están buscando, así como realizar un entendimiento de las diferentes opciones que se manejan, incluyendo la viabilidad y los riesgos que puedan suceder, para finalmente escoger este trabajo.

La siguiente fase, planificación, engloba todo el proceso cronológico y temporal, de modo que ayudará a establecer unos objetivos en un marco de tiempo que facilitará la organización de un proyecto. Además, se incluyen los costes vinculados al desarrollo del proyecto y se propone una gestión de riesgos que permita, una vez identificados estos, poder actuar con rapidez y precisión para solventarlos en la medida de lo posible a la mayor brevedad.

La fase posterior, correspondiente a ejecución, seguimiento y control, será la encargada de organizar el grueso del proyecto. Para ello, se tratarán los temas que involucran directamente el objetivo final, desde un acercamiento y preprocesamiento del dataset, hasta la creación de diferentes modelos con sus respectivas evaluaciones sobre los resultados.

Por último, se entraría en la última fase, la etapa de cierre, en la que se lleva a cabo una documentación ampliamente detallada de todo el proceso en el que ha consistido la elaboración del proyecto, recogida en esta memoria presentada.

Se presenta el diagrama EDT elaborado para el proyecto.

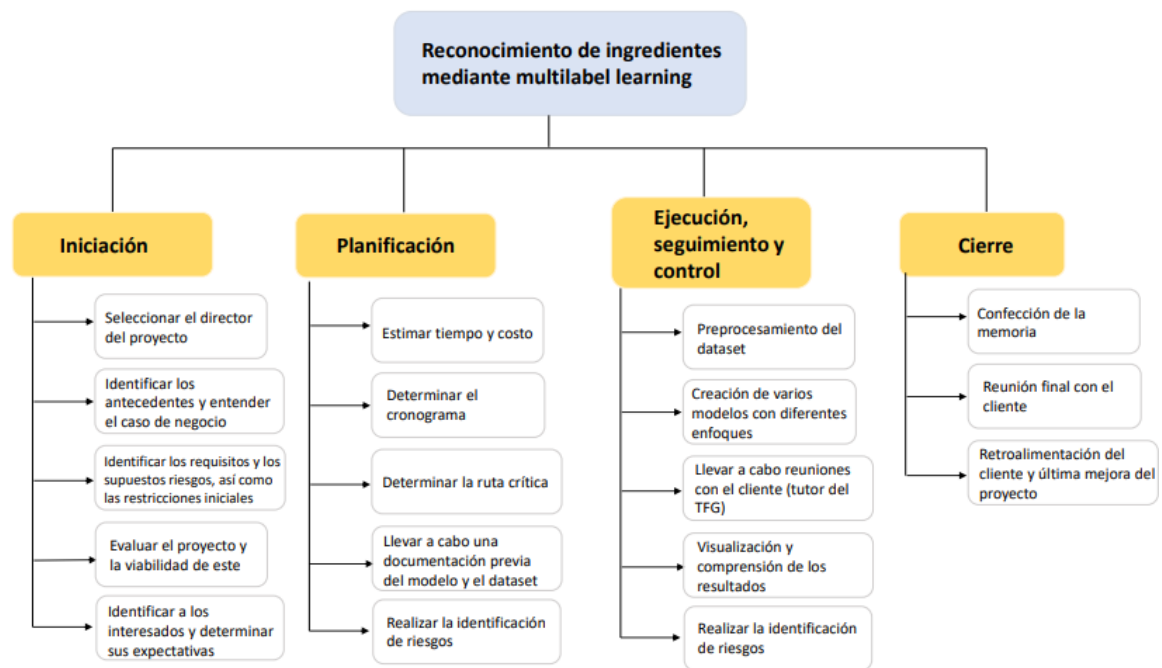


Figura 1. Diagrama EDT de tareas.

2.2 Planificación temporal

El proyecto comienza su hoja de ruta en el mes de septiembre de 2022, donde se inicia una serie de reuniones con diferentes tutores en busca del proyecto que más satisficiera las propias ambiciones personales, concluyendo con la elección del trabajo que se encuentra entre mis manos. Para su planificación temporal, se marcaron las siguientes etapas, que se ilustran en las sucesivas tablas.

Haciendo un puntualización, se aclara que estas tablas están compuestas por el tipo de tareas, en las que se incluyen hitos marcados a lo largo de cada proceso, la duración estimada inicialmente y la que, durante el desarrollo del proyecto, ha sido el tiempo real de cada tarea, y el tipo de rol que debería cumplir cada miembro del equipo de trabajo, en el que se citan hipotéticamente cuales serían, ya que la elaboración del trabajo ha sido llevado a cabo de forma individual, asumiendo la responsabilidad completa de su desarrollo y ejecución.

Fase 1. Iniciación			
Tarea	Duración estimada	Duración real	Rol
Seleccionar el director del proyecto	*	*	*
Identificar los antecedentes y entender el caso de negocio	4 horas	2 horas	Jefe del proyecto
Identificar los requisitos y los supuestos riesgos, así como las restricciones iniciales	10 horas	5 horas	Jefe del proyecto

Evaluar el proyecto y la viabilidad de este	4 horas	3 horas	Jefe del proyecto
Identificar a los interesados y determinar sus expectativas	1 hora	1 hora	Jefe del proyecto
Inicio del proyecto	0 días		
TOTAL	19 horas	11 horas	

Tabla 1. Tareas correspondientes al proceso de iniciación

Fase 2. Planificación			
Tarea	Duración estimada	Duración real	Rol
Estimar tiempo y costo	4 horas	4 horas	“Analista”
Determinar el cronograma	5 horas	4 horas	“Analista”
Determinar la ruta crítica	1 hora	1 hora	“Analista”
Construcción y análisis del cronograma	0 días		
Llevar a cabo una documentación previa del modelo y el dataset	15 horas	9 horas	“Desarrollador junior”
Realizar la identificación de riesgos	3 horas	2 horas	Jefe del proyecto
Realización de toda la documentación necesaria	0 días		
TOTAL	28 horas	20 horas	

Tabla 2. Tareas correspondientes al proceso de planificación

Fase 3. Ejecución, seguimiento y ejecución			
Tarea	Duración estimada	Duración real	Rol
Preprocesamiento del dataset	15 horas	12 horas	“Desarrollador junior”

Creación de varios modelos con diferentes enfoques	120 horas	135 horas	“Desarrollador junior”
Llevar a cabo reuniones con el cliente (tutor del TFG)	10 horas	4 horas	Jefe del proyecto
Resolución de errores y fallos	30 horas	55 horas	“Desarrollador junior”
Visualización y comprensión de los resultados	10 horas	29 horas	“Desarrollador junior”
Mejora continua del código adaptándolo a los requisitos	32 horas	28 horas	“Tester”
Creación de la red neuronal convolucional óptima	0 días		
TOTAL	217 horas	263 horas	

Tabla 3. Tareas correspondientes a la fase de ejecución, seguimiento y control

Fase 4. Cierre			
Tarea	Duración estimada	Duración real	Rol
Confección de la memoria	30 horas	26 horas	Jefe del proyecto
Reunión final con el cliente	2 horas	2 horas	Jefe del proyecto
Retroalimentación del cliente y última mejora del proyecto	4 horas	5 horas	Jefe del proyecto
Obtener la aceptación final del producto (entrega del TFG)	0 días		
TOTAL	36 horas	33 horas	

Tabla 4. Tareas correspondientes a la fase de cierre.

Fase	Duración estimada	Duración real	Desfase (%)
Iniciación	19 horas	11 horas	+0.57%
Planificación	28 horas	20 horas	+0.71%

Ejecución, seguimiento y control	217 horas	263 horas	+1.21%
Cierre	36 horas	33 horas	+0.916%
TOTAL	300 horas	327 horas	+1.09%

Tabla 5. Representación de la duración por fase y la duración total del proyecto. Incluye el desfase de la duración real con respecto a la duración estimada.

La desviación producida en cada una de las fases anteriores será explicada a continuación.

En la primera fase, el tiempo preestablecido para la duración de las distintas fases de iniciación superó la duración real, ya que las tareas han constituido menos tiempo a la hora de ser implementados.

Siguiendo con la fase de planificación, la desviación se produjo por el mismo tema comentado anteriormente, donde se predijo un tiempo mayor que el esperado a la hora de realizar las tareas de planificación.

Sin embargo, en la siguiente fase, ejecución seguimiento y control, ocurre el caso contrario. Las continuas mejoras que se han implementado, así como los problemas que han tenido que superarse para llevar a cabo la completitud del trabajo, siendo finalmente una tarea desplazada en el caso de la clasificación multilabel, han producido un aumento muy considerable en el tiempo establecido.

Por último, la desviación mínima producida en la fase de cierre se traduce en un menor tiempo en la redacción de la memoria, por cuestión de unas horas de diferencia.

2.3 Diagrama de Gantt

A continuación, se pasará a mostrar el diagrama de Gantt producido para el proyecto. El formato de fechas mostrado es yyyy/mm/dd y que la duración en días refleja el período de tiempo en el que dicha tarea se ha estado desarrollando, no la duración de la misma. La indicación del tiempo empleado en cada tarea es el reflejado en las tablas anteriores. Además, se tienen en cuenta los diferentes hitos marcados en el proyecto, visualizados en negrita dentro de las fases y con una duración de tiempo igual a 0.

PLANTILLA DE DIAGRAMA DE GANTT SIMPLE

TÍTULO DEL PROYECTO	Reconocimiento de ingredientes mediante multilabel learning				
GERENTE DE PROYECTO	Juan Miguel Moreno Valero				
NOMBRE DE LA EMPRESA	Universidad de Sevilla				
FECHA	sábado, 24 de septiembre de 2022				

TAREA IDENTIFICACIÓN	TAREA TÍTULO	ROL	EMPEZAR FECHA	PENDIENTE FECHA	DURACIÓN EN DÍAS
1	Iniciación del proyectos				
1,1	Seleccionar el director del proyecto	Jefe del proyecto	2022-09-24	2022-09-25	1
1,2	Identificar los antecedentes y entender el caso de negocio	Jefe del proyecto	2022-09-25	2022-09-26	1
1,3	Identificar los requisitos y los supuestos riesgos, así como las restricciones iniciales	Jefe del proyecto	2022-09-25	2022-09-26	1
1,4	Evaluar el proyecto y la viabilidad de este	Jefe del proyecto	2022-09-26	2022-09-27	1
1,5	Identificar a los interesados y determinar sus expectativas	Jefe del proyecto	2022-09-26	2022-09-27	1
1,6	Inicio del proyecto	Jefe del proyecto			
2	Planificación del proyecto				
2,1	Estimar tiempo y costo	Analista	2023-02-11	2023-02-12	1
2,2	Determinar el cronograma	Analista	2023-02-12	2023-02-13	1
2,3	Determinar la ruta crítica	Analista	2023-02-13	2023-02-14	1
2,4	Construcción y análisis del cronograma				
2,5	Llevar a cabo una documentación previa del modelo y el dataset	Desarrollador	2023-02-24	2023-02-25	1
2,6	Realizar la identificación de riesgos	Jefe del proyecto	2023-02-26	2023-02-27	1
2,7	Realización de toda la documentación necesaria				
3	Ejecución, seguimiento y control				
3,1	Preprocesamiento del dataset	Desarrollador	2023-04-07	2023-04-11	4
3,2	Creación de varios modelos con diferentes enfoques	Desarrollador	2023-04-15	2023-06-12	58
3,3	Llevar a cabo reuniones con el cliente (tutor del TFG)	Jefe del proyecto	2023-04-07	2023-06-30	84
3,4	Visualización y comprensión de los resultados	Desarrollador	2023-04-15	2023-06-12	58
3,5	Mejora continua del código adaptándolo a los requisitos	Desarrollador	2023-05-20	2023-06-12	23
3,6	Creación de la red neuronal convolucional óptima	Tester			
4	Fase de cierre del Proyecto				
4,1	Confección de la memoria	Jefe del proyecto	2023-06-12	2023-06-30	18
4,2	Reunión final con el cliente	Jefe del proyecto	2023-06-30	2023-06-30	
4,3	Retroalimentación del cliente y última mejora del proyecto	Jefe del proyecto	2023-06-30	2023-07-02	2
4,4	Obtener la aceptación final del producto (entrega del TFG)				

Figura 2. Creación de la organización de las tareas.

Se representa el diagrama de Gantt. Debido a haberse realizado sobre una plantilla de Excel que imposibilita su visualización completa, se lleva a cabo su representación a trozos, realizando una explicación para su entendimiento.

La primera imagen constituye con las fases marcadas y detalladas anteriormente, junto con el marco temporal en meses que constituyen. Se muestra la mejor manera posible para su visualización.

UNO. Mes de Septiembre					FASE DOS. Mes de Febrero					TERCERA FASE. Mes de Abril y Junio					FASE CUATRO				
SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 5	SEMANA 6	SEMANA 7	SEMANA 8	SEMANA 9	SEMANA 10	SEMANA 11	SEMANA 12	SEMANA 13	SEMANA 14	SEMANA 15	SEMANA 16	SEMANA 17	SEMANA 18	SEMANA 19	SEMANA 20	SEMANA 21
M	T	W	R	F	M	T	W	R	F	M	T	W	R	F	M	T	W	R	F

Figura 3. Fases y marcos mensuales del proyecto.

En la primera imagen de las dos sucesivas, se muestra el diagrama de Gantt correspondiente a las dos primeras fases, donde se aprecia que se desarrollan entre los meses de septiembre y febrero, respectivamente.



Figura 5. Diagrama de Gantt. Segunda imagen.

18

visualizarlo en su totalidad.

<https://docs.google.com/spreadsheets/d/1cqLbd6D3dSYiMqyHjOf9-nzdxV0aiHI/edit?usp=sharing&oid=106161389273997460743&rtpof=true&sd=true>

2.4 Estimación de costes

En los sucesivos apartados, se lleva a cabo el cálculo de los diferentes tipos de coste que ha producido el proyecto, siendo estos formados por el coste del personal, coste hardware y software, y coste indirecto.

2.4.1 Costes del personal

A continuación, se expone el coste del personal que ha supuesto el desarrollo del proyecto. Para ello, se ha realizado una búsqueda previa para estimar los sueldos medios de cada miembro del equipo, extraídos mediante la web (Indeed, 2023), página que muestra los salarios medios por mensuales y anuales, y se ha calculado el total de horas que han trabajado cada uno de los integrantes. Los resultados obtenidos se muestran a continuación.

Rol	Horas trabajadas	Sueldo por hora	Sueldo medio anual	Salario total obtenido
Jefe del proyecto	50h	17,48€	34.975€	874€
Analista	9h	16,18€	32.447€	145,62€
Desarrollador junior	200h	12,90€	25.860€	2.580€
Tester	28h	11,04€	27.960	309,12€

Tabla 6. Costes de personal.

Haciendo un cálculo de los sueldos de cada miembro teniendo en cuenta las horas trabajadas, se obtiene un total de 3.908,74€ invertidos en este proyecto por parte del personal.

2.4.2 Costes de hardware

Se debe tener en cuenta el coste de los equipos que se necesitan para llevar a cabo el desarrollo del proyecto. A continuación, se detalla el presupuesto según el coste de estos.

En este caso, solo es necesario contar con un ordenador, portátil en nuestro caso, donde se llevará a cabo el desarrollo completo del proyecto. El modelo del que se dispone se trata de un HP Laptop 15-da1xxx, que cuenta con un procesador CPU de 1.99 GHz modelo Intel Core i7-

8565U y una memoria RAM de 8 GB. La adquisición de este dispositivo supuso un desembolso de 799€, constituyendo el coste total de hardware.

2.4.3 Costes de software

El precio del software utilizado, Google Colab, es inicialmente gratuito para todos sus consumidores. Sin embargo, para conseguir algo más de recursos y una mayor prioridad en el acceso a las GPUs y TPUs que facilitan, se consideró favorable realizar una suscripción en la versión Pro por 11,19€ al mes. Debido a que esta decisión se llevó a cabo en el último mes del proyecto para conseguir una disponibilidad de recursos más directa debido al acercamiento de la fecha de finalización del proyecto, la cuantía total de los costes de software es de 11.19€

2.4.4 Costes indirectos

Cabe mencionar una serie de costes que no están directamente relacionados con una actividad específica del proyecto, pero que son necesarios para su realización y correcta gestión. Costes que se pueden incluir en este apartado es el gasto en un suministro eléctrico y contratación de conexión a Internet, ya que el software no se encuentra almacenado en local, sino que se accede a través de la web. Se estima por tanto que dichos costes supondrían una adición de 60€ por mes en la economía del proyecto. Determinando que el proyecto a supuesto un total de 5 meses, aproximadamente, el precio estimado total sería de 300€.

2.4.5 Coste total

Tipo de coste	Precio
Personal	3.908,74€
Hardware	799€
Software	11,19€
Indirecto	300€
TOTAL	5.018,93

Tabla 7. Coste total.

2.5 Gestión de riesgos

En este apartado se describe el plan de gestión de riesgos, que incluye la identificación de estos, así como las soluciones a las adversidades previstas.

2.5.1 Identificación de los riesgos

Se exponen cuáles se han declarado como situaciones de riesgo en nuestro proyecto, que podrían suponer un estancamiento del mismo y provocar un retraso en todas las fases posteriores a la tarea donde se ha producido el inconveniente.

- **Volumen de datos excesivamente amplio** para los recursos que Google Colab ofrece. A la hora de escoger el dataset, uno de los principales puntos a tener en cuenta era que fuese lo suficientemente rico para mostrar un resultado lo más parecido a la realidad posible, que posibilitara varias estrategias para su procesamiento. Sin embargo, un riesgo que suponía esta idea era que el conjunto de datos fuese demasiado grande para poder ser ejecutado en un tiempo de cómputo óptimo.
- **Encontrar los parámetros adecuados a la hora de las diferentes clasificaciones** que se realizan. La posibilidad de que los parámetros fueran erróneamente seleccionados y alejara a los modelos de mejores resultados era uno de los riesgos que más preocupan a la hora de la elección de este proyecto, ya que no existe ninguna forma eficaz para poder establecer qué parámetros funcionarán en el modelo y cuáles no.
- **La creación de la clasificación multilabel.** Debido a no haber trabajado nunca con este tipo de clasificación anteriormente, y ser más compleja que una clasificación multiclase, un riesgo a identificar es que la demora en la implementación de estos modelos excediese de manera notoria los tiempos preestablecidos para su construcción.
- **Dificultad para cumplir los tiempos marcados en cada fase** y no llegar a la fecha de entrega a tiempo. Los retrasos inesperados podrían suponer un agravante en el proyecto hasta el punto de poder imposibilitar la entrega de este.

2.5.2 Plan de contingencia

El plan contingencia está elaborado con las soluciones a la aparición de los riesgos mencionados anteriormente, para poder estar preparados si estos ocurren y actuar de la manera más eficaz posible.

- **Implementación de funciones que pudieran reducir el dataset** en función de los intereses, de modo que se pudiera tratar con un subconjunto de datos del original que se adaptase a las características computacionales.
- **Llevar a cabo una exhausta documentación** de cuáles pueden ser los mejores parámetros para atribuir dependiendo de la situación en la que encuentre el modelo, y realizar múltiples pruebas hasta descubrir la combinación que devuelva los mejores resultados posibles.
- De nuevo, realizar una documentación e **investigar sobre la implementación de ejemplos similares** que poder establecer como modelo a la hora de desarrollar el propio adaptado al problema que se tiene entre manos.

- **Redefinir las etapas y los tiempos, prescindiendo de algunas tareas minoritarias** de ser estrictamente necesario. Otra opción sería rediseñar la idea y objetivo final para adaptarlo a los tiempos disponibles.

3 Contexto del proyecto

En este apartado se pretende aportar un contexto del conocimiento necesario para el entendimiento completo de los conceptos que se desarrollan durante el proyecto. De esta forma, se sigue una estructura donde se plantea la importancia del caso que se trata y un estudio detallado de la Inteligencia Artificial, desembocando en las técnicas de Deep Learning, hasta la explicación de las redes neuronales convolucionales, base fundamental de este trabajo.

3.1 Concienciación del problema

Actualmente, según el estudio publicado por la página (MÉDICA, 2019), un 25% de la población mundial sufre de alguna intolerancia alimenticia, siendo un 3% la parte de la ciudadanía que presenta algún tipo de alergia alimentaria. Para su mayor contextualización y diferenciación, se pasa a apuntar una definición breve de ambos términos.

Intolerancia se corresponde con una patología muy frecuente que ocurre cuando la digestión de un alimento no se realiza de forma adecuada produciendo diversos síntomas como hinchazón, gases, etc. En contraposición, el término alergia alimentaria define el problema que ocurre cuando hay un mecanismo inmunológico que reaccione ante la ingesta de un alimento, provocando síntomas más graves que los ocasionados por una intolerancia. Por tanto, una persona que sufriese de alguna de estas deficiencias estaría, en mayor o menor medida, poniendo en riesgo su salud ante una ingesta de un ingrediente que provocase la reacción de su organismo.

Además, estos estudios apuntan que cada vez es más frecuente el número de casos de personas con algún tipo de estas anomalías, provocado posiblemente, entre otras hipótesis, por la fuerte protección que existe hacia el sistema inmunológico en los países industrializados, donde más se reflejan estos casos. La prevención del organismo frente a infecciones con innumerables vacunas y un cuidado excesivo de la higiene causa que el individuo no se exponga tanto a los gérmenes, afectando al sistema inmunitario y derivando en un incremento de reacciones alérgicas.

Si se traslada esta información a datos puramente empíricos, se estaría hablando de un total de 140 millones de personas de la población mundial que sufriría alguna de estas patologías, siendo la cifra de 17 millones en Europa, y en torno a los 2 millones en España. Números que revelan la importancia de una especial atención a este problema de salud.

A modo visual, se presenta un gráfico con las alergias alimentarias más extendidas en España en el año 2015:

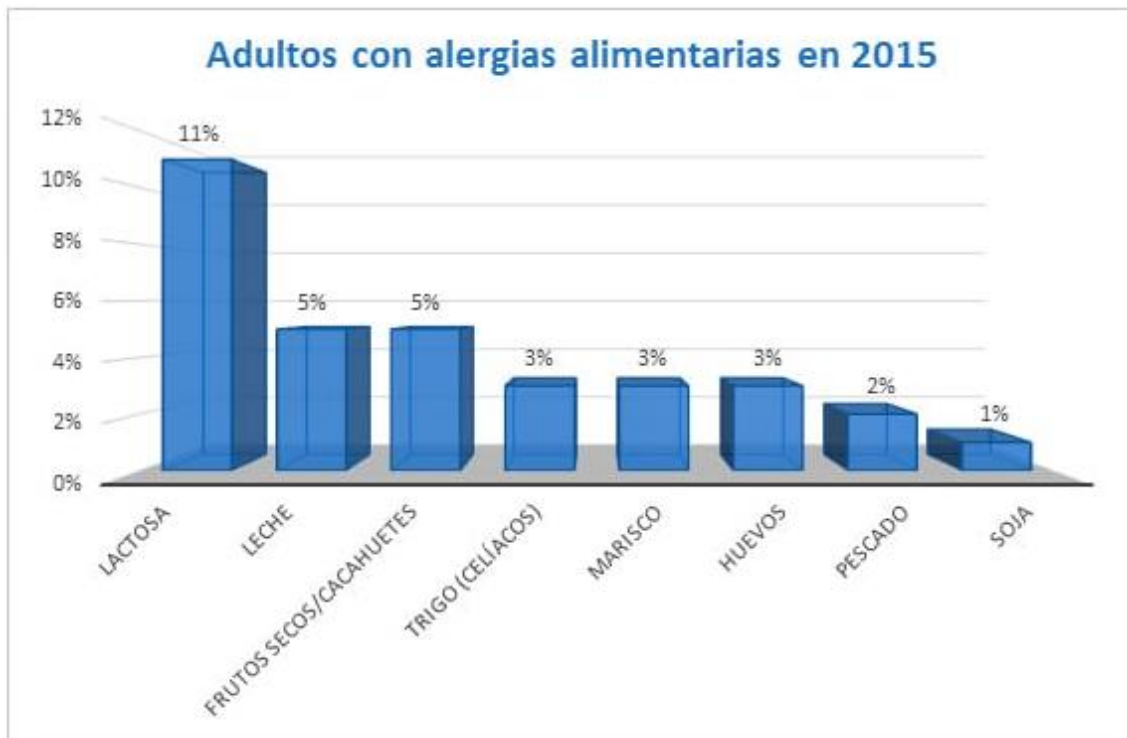


Figura 6. Adultos con alergias alimentarias en 2015. (MÉDICA, 2019)

A su vez, un gráfico de barras circular sobre la prevalencia de alergias a los alimentos por comunidades autónomas en 2015 y 2005.

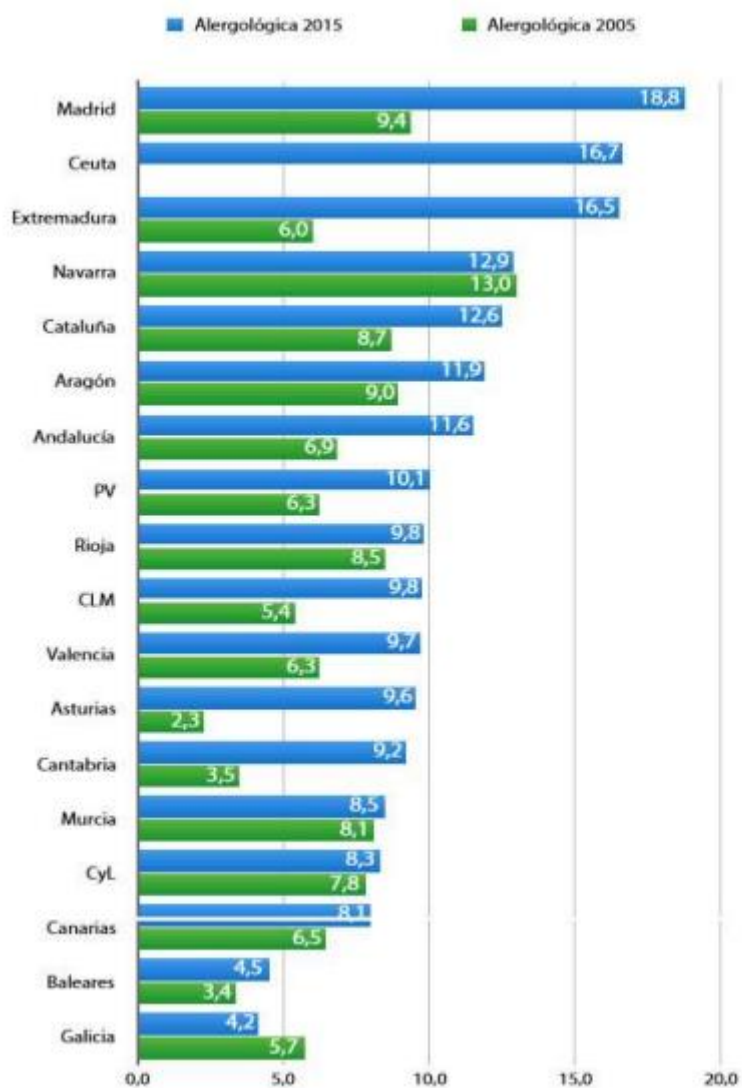


Figura 7. Alergias por comunidades autónomas en el año 2015 y 2005. (González, 2015)

Se muestra a continuación la prevalencia a padecer una alergia según el grupo de edad.

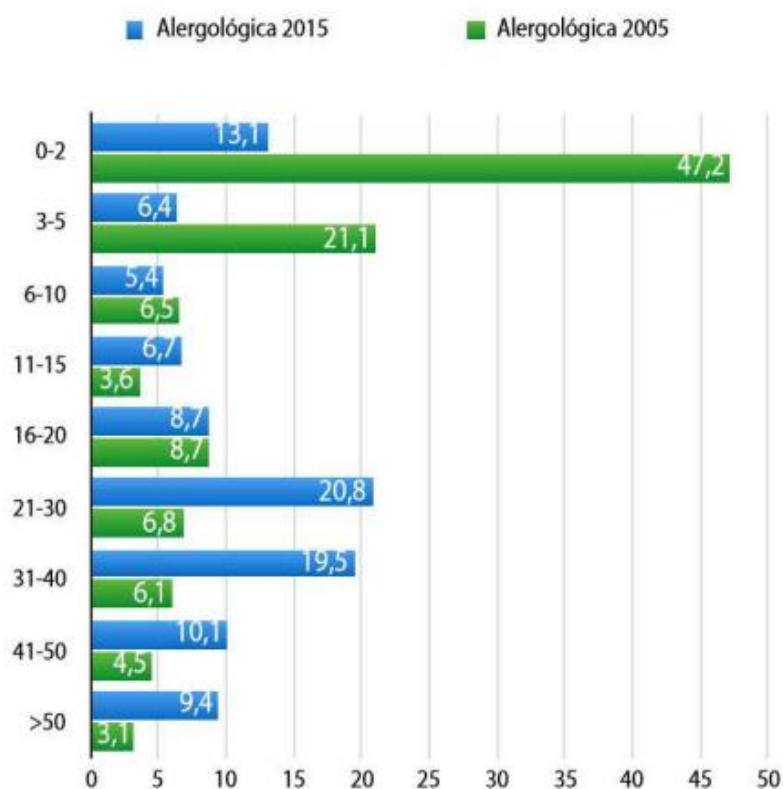


Figura 8. Prevalencia a padecer alergia por grupos de edad. Años 2015 y 2005. (González, 2015)

Se adjunta una imagen sobre los signos y síntomas de sospecha según el grupo de edad.

NIÑOS	ADOLESCENTES	ADULTOS
<ul style="list-style-type: none"> Diarrea Anorexia Vómitos Dolor abdominal Irritabilidad Apatía Introversión Tristeza 	<ul style="list-style-type: none"> Frecuentemente asintomáticos Dolor abdominal Cefalea Artralgias Menarquia retrasada Irregularidades menstruales Estreñimiento Hábito intestinal irregular 	<ul style="list-style-type: none"> Dispepsia Diarrea crónica Dolor abdominal Síndrome de intestino irritable Dolores óseos y articulares Infertilidad, abortos recurrentes Parestesias, tetania Ansiedad, depresión, epilepsia, ataxia

Figura 9. Signos y síntomas de sospecha según grupo de edad. (González, 2015)

Debido a todo lo mencionado anteriormente, cada vez es más la conciencia que se tiene de este riesgo y se buscan más medidas para poder ofrecer soluciones a aquellas personas que padecen de estas intolerancias. Métodos que se observan con cada vez más frecuencia son las aplicaciones para detectar los ingredientes que tiene un producto seleccionado en un supermercado, mediante su código nutricional, así como la inclusión de advertencias de los principales alérgenos más comunes en las cartas de los restaurantes para favorecer la identificación de estos por parte del cliente.

3.2 Inteligencia Artificial

La Inteligencia Artificial (IA) se refiere al campo de estudio y desarrollo de sistemas y programas informáticos que pueden realizar tareas que normalmente requerirían de la inteligencia humana. La IA busca crear programas y algoritmos capaces de simular procesos cognitivos como el razonamiento, el aprendizaje, la percepción, la comprensión del lenguaje natural y la toma de decisiones. La Inteligencia Artificial comprende las siguientes ramas:

- **Aprendizaje Automático (Machine Learning).** Es una rama de la IA que se centra en desarrollar algoritmos y modelos que pueden aprender de los datos y mejorar su rendimiento a medida que se les proporciona más información. Incluye técnicas como el aprendizaje supervisado, no supervisado y por refuerzo.
- **Redes neuronales artificiales.** Son modelos inspirados en el funcionamiento del cerebro humano. Estas redes están compuestas por unidades llamadas neuronas interconectadas y se utilizan para realizar tareas de aprendizaje automático, reconocimiento de patrones, procesamiento de imágenes y procesamiento de lenguaje natural, entre otros.
- **Procesamiento del lenguaje natural.** Se enfoca en desarrollar algoritmos y técnicas para comprender, interpretar y generar lenguaje humano de manera natural. Esto implica tareas como el reconocimiento de voz, la traducción automática, la generación de texto y el análisis de sentimientos.
- **Visión por computadora.** Se ocupa del análisis y procesamiento de imágenes y vídeos para permitir a las máquinas 'ver' y comprender su entorno visual. Incluye tareas como la detección de objetos, el reconocimiento facial, la segmentación de imágenes y la reconstrucción en 3D.
- **Robótica.** Se enfoca en la creación de robots y sistemas autónomos capaces de interactuar con su entorno y realizar tareas específicas. La IA juega un papel crucial en la planificación de movimientos, la percepción del entorno y la toma de decisiones en tiempo real.
- **Sistemas expertos.** Son programas de software que emulan la experiencia y el conocimiento de un experto humano en un dominio específico. Utilizan reglas lógicas y heurísticas para razonar y tomar decisiones.

- **IA basada en reglas.** Se basa en la construcción de sistemas que siguen un conjunto de reglas y condiciones predefinidas para tomar decisiones y realizar tareas específicas.

Una vez definidas sus principales ramas, se realiza una inmersión dentro del machine learning, hasta concluir en Deep Learning, donde se realiza una explicación más detallada a continuación.

3.3 Deep Learning

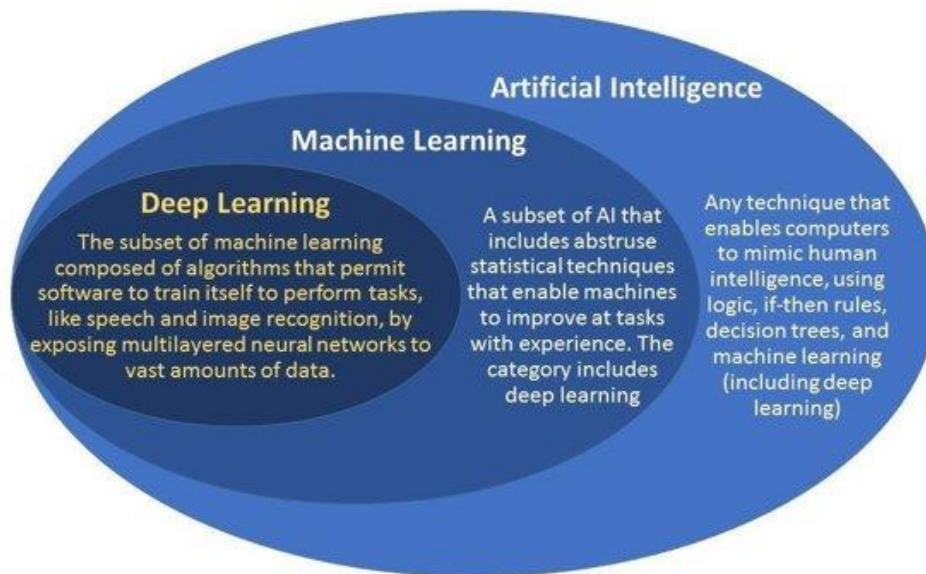


Figura 10. Inteligencia Artificial. Machine Learning. Deep Learning. (Cloud, s.f.)

Como se puede observar en la fotografía anteriormente mostrada, el Deep Learning, también conocido como aprendizaje profundo, es una rama del machine learning (o aprendizaje automático) que se basa en redes neuronales artificiales profundas. Se centra en el desarrollo de modelos y algoritmos capaces de aprender y realizar tareas de alto nivel a partir de grandes conjuntos de datos.

A diferencia del aprendizaje automático tradicional, que puede requerir una extracción manual de características relevantes de los datos, el Deep Learning busca aprender automáticamente características y representaciones de alto nivel directamente de los datos sin una intervención humana significativa. Esto se logra a través de las famosas redes neuronales artificiales profundas, los cuales son modelos que contienen múltiples capas de unidades de procesamiento interconectadas llamadas neuronas.

El Deep Learning ha experimentado un gran avance de los últimos años debido a múltiples factores, incluida la disponibilidad de grandes conjuntos de datos, avances en hardware de computación como (GPUs y TPUs), especializados en operaciones matriciales y paralelismo masivo, algoritmos de optimización eficientes y avances en arquitecturas de redes neuronales, como las redes neuronales convolucionales (CNN) y las redes neuronales recurrentes (RNN).

Para el problema que se tiene entre manos, el uso del Deep Learning es una herramienta fundamental, ya que proporciona la capacidad de analizar grandes volúmenes de imágenes y

detectar entre ellas patrones generales que desembocarán en las características que permiten clasificar e identificar los diferentes elementos que se encuentran en ellas.

3.4 Redes neuronales

Debido a que la técnica fundamental que se ha empleado en este proyecto han sido las redes neuronales, se dedica una sección para profundizar en su comprensión y composición, así como de una explicación de los diferentes tipos de redes neuronales artificiales que existen.

En primer lugar, la inspiración biológica de la que se nutren estas técnicas son las neuronas reales que componen el sistema nervioso del ser humano. Una neurona es una célula fundamental en el procesamiento y transmisión de señales eléctricas y químicas en el cerebro y sistema nervioso, siendo las células responsables de la comunicación entre las diferentes partes del cuerpo y el control de las funciones corporales.

La estructura de una neurona biológica consta de tres partes principales:

- **Cuerpo celular (soma).** Es la parte principal de la neurona que contiene el núcleo y otras estructuras celulares importantes. El soma coordina las funciones básicas de la célula y realiza procesos metabólicos.
- **Dendritas.** Son extensiones ramificadas y corta que se extienden desde el cuerpo celular. Las dendritas reciben señales y entradas de otras neuronas o células sensoriales y las transmiten hacia el cuerpo celular.
- **Axón.** Es la prolongación larga y delgada que se origina en el cuerpo celular y puede ramificarse hacia el final. El axón lleva la señal eléctrica generada por la neurona y la transmite hacia otras neuronas o hacia células efectoras, como las células musculares o glandulares.

Una representación de una neurona biológica sería la siguiente.

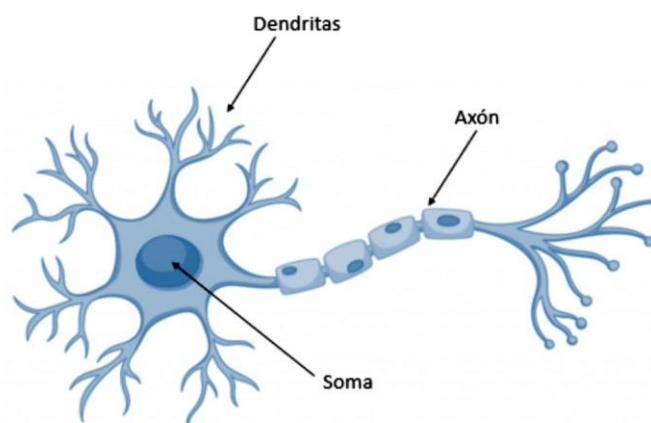


Figura 11. Ilustración de una neurona biológica. (IBM, ¿Qué son las redes neuronales?, s.f.)

Una vez entendida la base biológica que proporciona el fundamento de las redes neuronales, se procede a atacar la técnica en cuestión: ¿qué son las redes neuronales artificiales?

En el cerebro humano, miles de millones de neuronas trabajan en paralelo para procesar información y realizar tareas cognitivas y sensoriales. Cada neurona puede recibir múltiples señales de entrada y generar una salida que se propaga a través de sus axones hacia otras neuronas. Esto permite un procesamiento masivamente paralelo y distribuido en el cerebro, lo que contribuye a su capacidad de realizar tareas complejas y rápidas.

Las redes neuronales artificiales aprovechan esta característica para lograr un paralelismo masivo, pero a través de la arquitectura de hardware utilizada para su implementación. Una red neuronal artificial puede consistir en capas de nodos (denominadas neuronas) interconectados, donde cada nodo realiza operaciones matemáticas en paralelo con otros nodos, aplicando una función de activación a la salida para evitar la linealidad en los resultados.

Por tanto, una única neurona (artificial) tendría el siguiente aspecto.

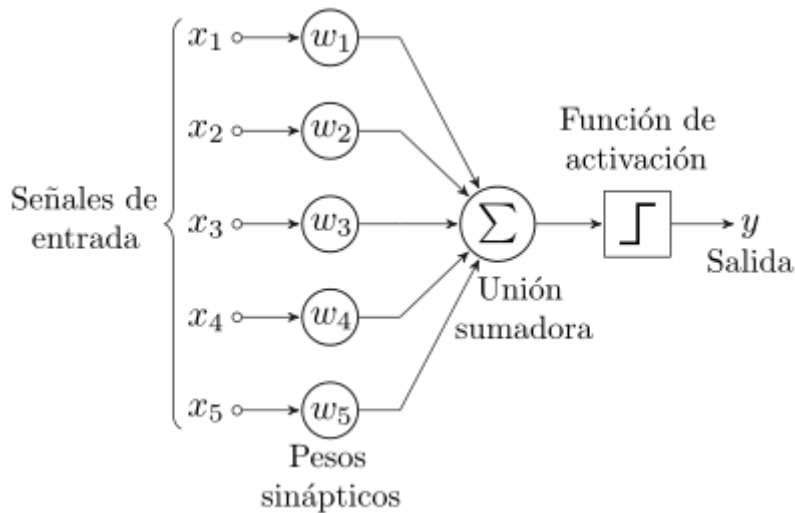


Figura 12. Representación de una neurona artificial. (IBM, ¿Qué son las redes neuronales?, s.f.)

La estructura de una red neuronal estaría formada por:

- **Una capa de entrada, o *input layer*.** Esta capa recibe las características o datos de entrada para el modelo. Cada neurona en esta capa representa una dimensión o característica de los datos de entrada.
- **Capas ocultas, o *hidden layers*.** Estas capas se encuentran entre la capa de entrada y la capa de salida. Son responsables de extraer y aprender características y patrones complejos a partir de los datos de entrada. Cada neurona en una capa oculta recibe conexiones de las neuronas de la capa anterior y produce una salida que se transmite a las neuronas de la capa siguiente.
- **Capa de salida, u *output layer*.** Esta capa produce el resultado final o la predicción de la red neuronal. La cantidad de neuronas en esta capa depende del tipo de problema que se esté abordando. Por ejemplo, en un problema de clasificación binaria, puede haber

una neurona de salida que representa la probabilidad de pertenecer a una clase. En un problema de clasificación multiclase, puede haber varias neuronas de salida, cada una representando la probabilidad de pertenecer a una clase específica. En el caso de clasificación multilabel, habrá tantas neuronas como etiquetas haya, que representaran la probabilidad binaria de pertenecer a dicha clase o no. Estas dos últimas clasificaciones son las que se utilizan en el trabajo.

La siguiente figura representa la estructura de la red, correspondiendo al tipo de red neuronal artificial más simple, el perceptrón multicapa.

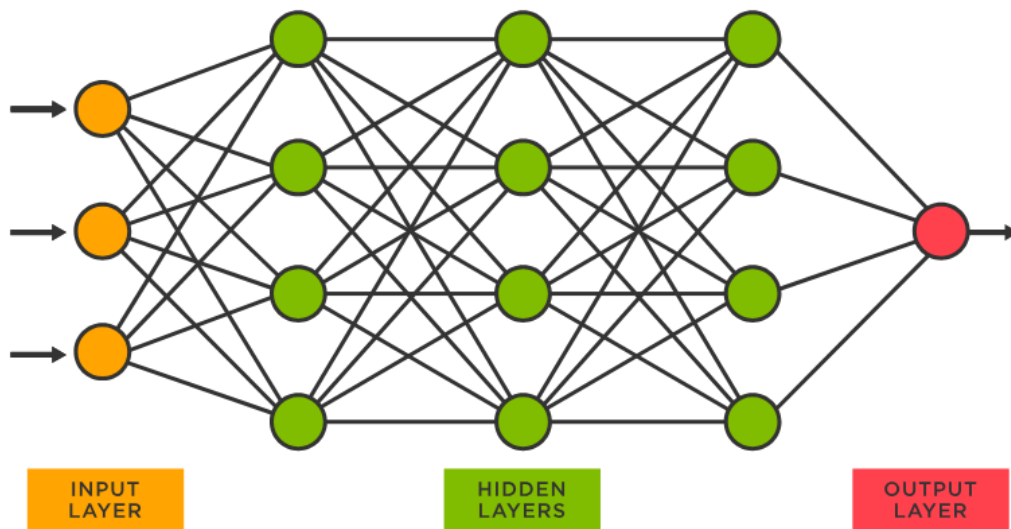


Figura 13. Ilustración de una red neuronal. Perceptrón multicapa. (IBM, ¿Qué son las redes neuronales?, s.f.)

3.5 Tipos de funciones de activación

La función de activación en las redes neuronales es una función matemática aplicada a la salida de una neurona o capa para introducir no linealidad en la red, permitiendo que las redes neuronales aprendan y modelen relaciones y patrones complejos en los datos. Sin la función de activación, las redes neuronales serían meramente combinaciones lineales de las entradas, y no podrían capturar relaciones no lineales entre características.

A continuación, se pasa a citar y describir las principales funciones de activación que existen.

- **Función de activación Sigmoide (Sigmoid):** La función sigmoid transforma los valores de entrada en un rango entre 0 y 1. Tiene forma de curva sigmoide y es adecuada para problemas de clasificación binaria donde se desea asignar una probabilidad a cada clase. Sin embargo, la función sigmoid puede sufrir del problema de "desvanecimiento del gradiente" en redes neuronales profundas.
- **Función de activación Tanh:** La función tangente hiperbólica (tanh) transforma los valores de entrada en un rango entre -1 y 1. Es simétrica alrededor del origen y

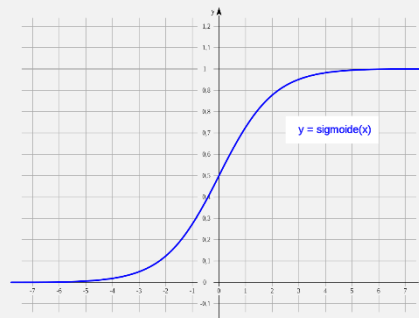
proporciona una salida negativa para valores negativos de entrada. La función tanh se utiliza a veces en capas ocultas de redes neuronales.

- **Función de activación ReLU (Rectified Linear Unit):** La función ReLU es la función de activación más común en redes neuronales. Se define como $f(x) = \max(0, x)$, es decir, toma el valor de entrada si es positivo y cero en caso contrario. La función ReLU es fácil de calcular y acelera el proceso de entrenamiento, además de ayudar a mitigar el problema del desvanecimiento del gradiente. Es el tipo más comúnmente usado.
- **Función de activación Leaky ReLU:** La función Leaky ReLU es una variante de la función ReLU que soluciona el problema de la "neurona muerta" que puede ocurrir en la función ReLU cuando los valores de entrada son negativos. La función Leaky ReLU tiene una pendiente pequeña para los valores negativos, lo que permite que el gradiente fluya incluso para valores negativos.

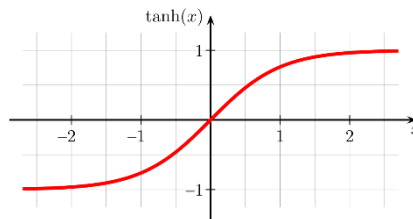
Función de activación

Gráfica

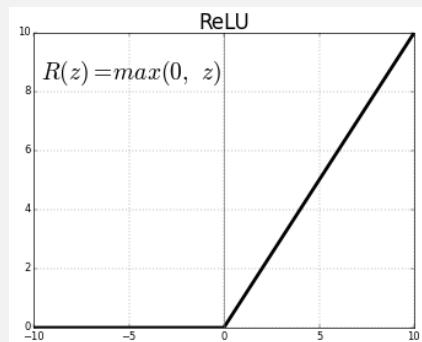
Sigmoide(x)



Tanh(x)



Relu



Leaky Relu

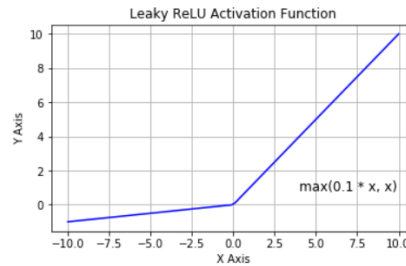


Tabla 8. Representación gráfica de las principales funciones de activación.

3.6 Tipos de redes neuronales

Existe una amplia variedad de redes neuronales, dependiendo de la función específica que quiera cubrir cada una de ellas. A continuación, se exponen las más conocidas:

- **Perceptrón multicapa.** El perceptrón multicapa es una red neuronal artificial con capas de neuronas interconectadas entre sí. Incluye una capa oculta entre la capa de entrada y la capa de salida y se entrena ajustando los pesos mediante algoritmos de aprendizaje supervisado. Puede modelar funciones y relaciones complejas, siendo efectivo en tareas de reconocimiento de patrones. Es ampliamente utilizado en problemas de clasificación, regresión y procesamiento de datos de alta dimensionalidad.
- **Redes neuronales convolucionales (CNN).** La CNN es una red neuronal especializada en procesar datos con estructura de cuadrícula, como imágenes. Utiliza capas de convolución, pooling y activaciones no lineales para detectar características locales, realizando el entrenamiento por medio del ajuste de los pesos mediante algoritmos de optimización. Es altamente eficiente en tareas de visión por computadora y reconocimiento de objetos. Su arquitectura se basa en el concepto de compartición de pesos y el reconocimiento de patrones locales. Son las redes en las que se basa el proyecto.
- **Redes neuronales recurrentes (RNN).** Las redes neuronales recurrentes (RNN) son un tipo de red neuronal que se utiliza para procesar datos secuenciales, como texto o series de tiempo. Tienen conexiones retroalimentadas que les permiten tener memoria y capturar dependencias a largo plazo en las secuencias. Las RNN utilizan unidades de memoria llamadas células, como las LSTM o GRU, para modelar la secuencia, entrenándose mediante algoritmos de optimización para ajustar los pesos. Son efectivas en tareas de procesamiento de lenguaje natural, traducción automática y generación de texto.

Se presenta un ejemplo de redes recurrentes, tanto en su versión *fold* como en su versión *unfold*. El perceptrón multicapa fue mostrado anteriormente y en el siguiente apartado se realiza una visualización de una red neuronal convolucional (CNN).

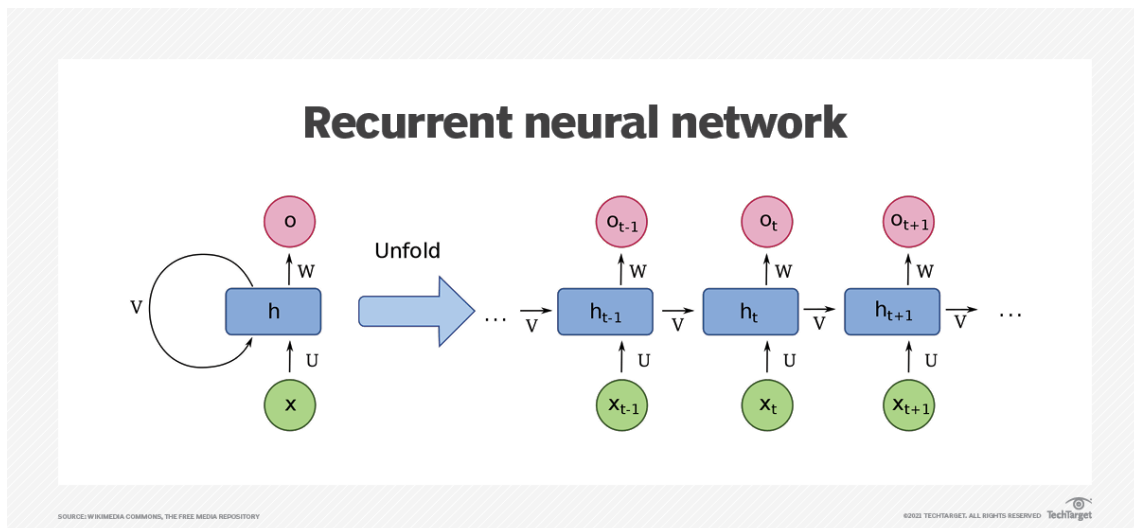


Figura 14. Redes recurrentes (RNN). (IBM, ¿Qué son las redes neuronales?, s.f.)

3.7 Redes neuronales convolucionales

Debido a ser el tipo de red neuronal utilizada, se pasa a llevar a cabo una explicación más detallada de esta.

Las redes neuronales convolucionales (CNN) fueron desarrolladas originalmente en la década de 1980 por Yann LeCun, junto con otros investigadores, mientras trabajaban en el Laboratorio de Inteligencia Artificial y Ciencias de la Computación de Bell Labs. LeCun es considerado uno de los pioneros en el campo de las CNN y ha realizado contribuciones significativas en su desarrollo. En particular, introdujo una arquitectura de red conocida como LeNet-5, que fue una de las primeras aplicaciones exitosas de las CNN. Esta red se utilizó para clasificar dígitos escritos a mano en cheques bancarios y jugó un papel importante en el avance del reconocimiento óptico de caracteres (OCR).

Desde entonces, las CNN han evolucionado y se han vuelto cada vez más sofisticadas, permitiendo mejoras significativas en áreas como la visión por computadora, el procesamiento de imágenes y el aprendizaje profundo. Las contribuciones de LeCun y su equipo en la década de 1980 sentaron las bases para el desarrollo posterior de las CNN y su aplicación en una amplia gama de campos.

La idea principal detrás de las CNN es la convolución, una operación matemática que implica aplicar un filtro o *kernel* a una región local de los datos de entrada y generar una representación de esa región. Esta operación se repite en todas las ubicaciones posibles de la entrada, lo que permite capturar características locales en diferentes partes de la imagen.

La arquitectura básica de una CNN consiste en varias capas, incluyendo capas de convolución, capas de pooling y capas completamente conectadas, que pasará a definir a continuación.

- **Las capas de convolución** son responsables de extraer características relevantes de las imágenes mediante la aplicación de filtros. Estos filtros se ajustan durante el entrenamiento para capturar características específicas, como bordes, texturas o patrones visuales. Estas características se verían de la siguiente forma, aplicado a un dataset de perros y gatos, clasificando una imagen de un canino.

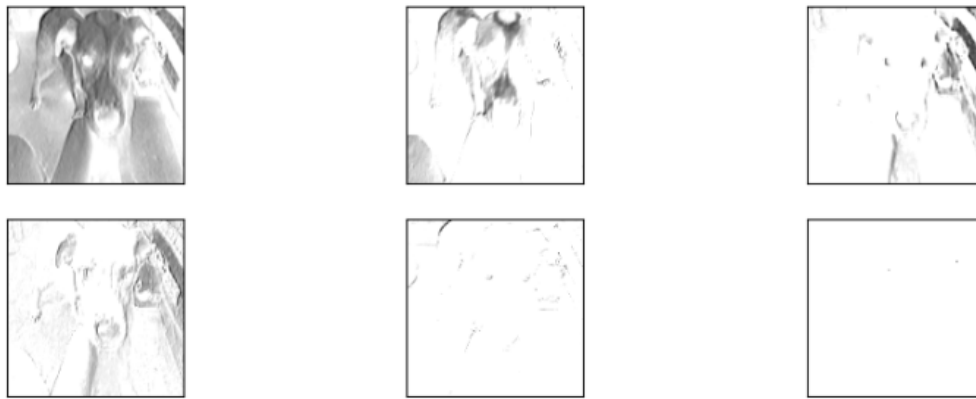


Figura 15. Características extraídas mediante CNN. (IBM, Redes Neuronales Convolucionales, s.f.)

- **Las capas de pooling** reducen la dimensionalidad espacial de las características extraídas, preservando las características más importantes y descartando la información redundante. Esto permite reducir la cantidad de parámetros y computaciones necesarias en la red. Existen tres tipos de capas de pooling utilizadas:
 - **Capa de Pooling Máximo (MaxPooling):** Esta es la capa de pooling más común. En esta capa, se divide la entrada en regiones y se toma el valor máximo de cada región como salida. El pooling máximo ayuda a preservar las características más prominentes y dominantes en la imagen.

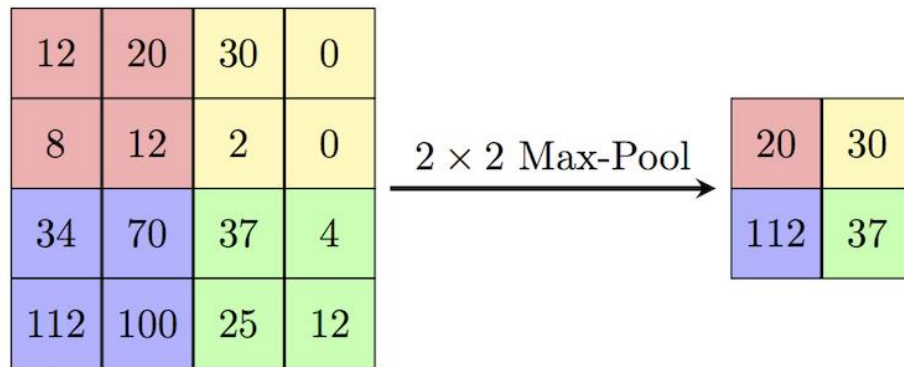


Figura 16. MaxPooling. (IBM, Redes Neuronales Convolucionales, s.f.)

- **Capa de Pooling Media (AveragePooling):** En esta capa de pooling, se calcula el valor promedio de cada región de la entrada y se utiliza como salida. A diferencia del pooling máximo, esta capa tiende a suavizar las características y reducir el nivel de detalle.

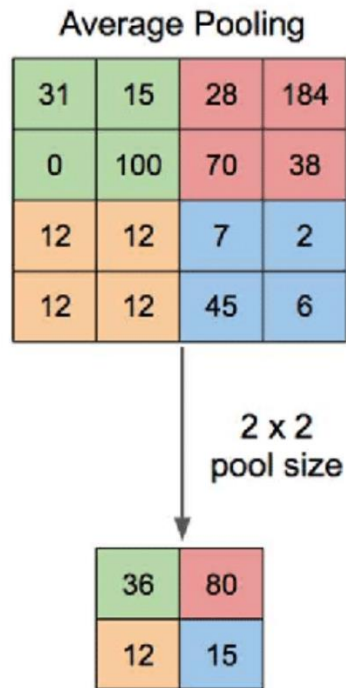


Figura 17. AveragePooling. (IBM, Redes Neuronales Convolucionales, s.f.)

- **Capa de Pooling Global (GlobalAveragePooling):** Su función es la de reducir la dimensionalidad espacial de un tensor tridimensional que representa una imagen. En lugar de mantener las dimensiones espaciales (ancho y alto), esta capa calcula el promedio de cada canal del tensor a lo largo de toda la imagen, generando un único valor para cada canal. Esto significa que la capa "colapsa" las dimensiones espaciales en una sola dimensión, manteniendo la información de cada canal por separado.

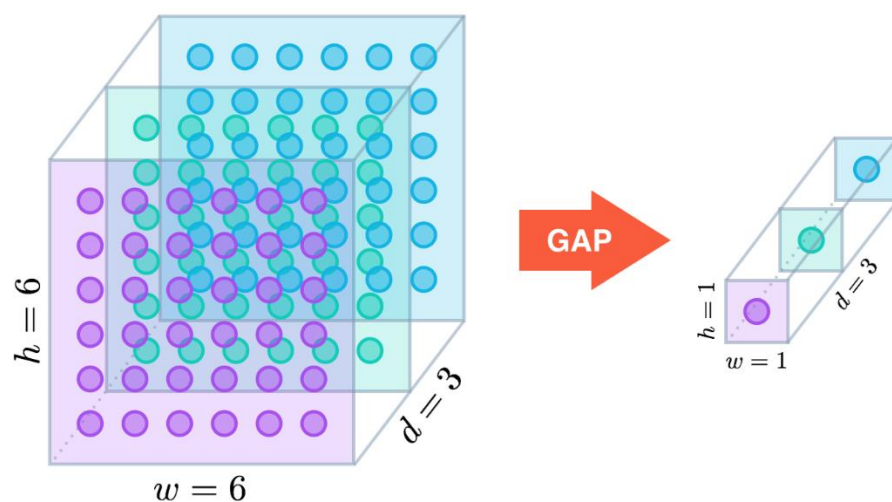


Figura 18. GlobalAveragePooling. (IBM, Redes Neuronales Convolucionales, s.f.)

- **Las capas completamente conectadas, o capas densas,** al final de la CNN se utilizan para realizar la clasificación o predicción final. Estas capas toman las características extraídas por las capas anteriores y las procesan para generar las salidas deseadas, como las probabilidades de pertenecer a diferentes clases en un problema de clasificación.

Durante el entrenamiento de una CNN, se ajustan los pesos de las capas de convolución y las capas completamente conectadas utilizando algoritmos de optimización, como el descenso del gradiente. El objetivo es minimizar la diferencia entre las salidas predichas y las salidas reales. Una de las características clave de las CNN es el concepto de compartición de pesos, que significa que los mismos filtros se aplican a diferentes partes de la imagen, lo que permite la detección de características invariantes a la ubicación. Esta compartición de pesos reduce la cantidad de parámetros entrenables y ayuda a la generalización de la red.

Esta imagen ilustra lo que supondría la estructura de una red neuronal convolucional (CNN), donde se observa una composición de dos capas de convolución, dos de pooling y tres capas densas finales, siendo dos de ellas intermedias y una última la capa de salida.

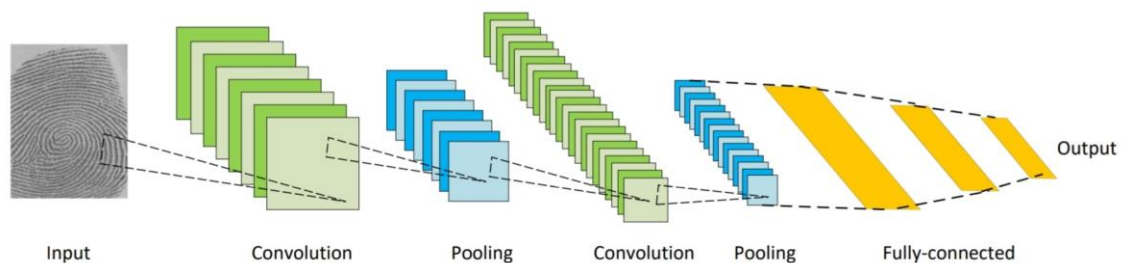


Figura 19. Red neuronal convolucional (CNN). (IBM, Redes Neuronales Convolucionales, s.f.)

Trasladando esta estructura al problema, se construye un esquema simplificado como el siguiente, donde se utilizan las CNN para llevar a cabo una clasificación inicialmente multiclase, clasificando los platos de comida o *dishes*, y posteriormente llevar a cabo una clasificación multilabel, o multietiqueta, que se encargaría de clasificar los diferentes ingredientes de cada tipo de comida.

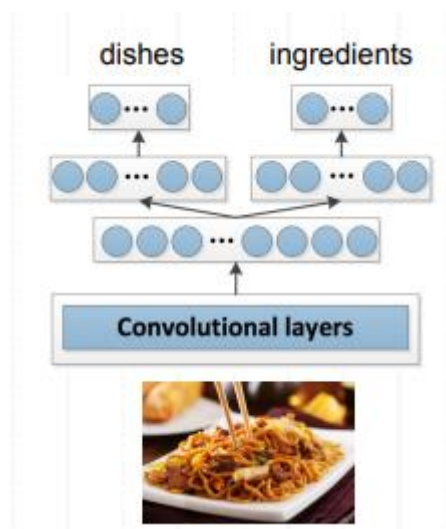


Figura-20. Esquema de CNN de nuestro problema.

4 Metodología

En esta sección se lleva a cabo una descripción de la metodología empleada en el proyecto, que sigue la siguiente estructura.

Se empieza explicando las diferentes tecnologías y herramientas empleadas que se han utilizado para facilitar la creación del proyecto, englobando el lenguaje de programación utilizado y el entorno en el que se ha desenvuelto el proyecto, así como las librerías que han sido utilizadas.

Seguidamente, se sigue una explicación detallada de los aspectos generales de los modelos de clasificación, es decir, el esquema base que se utilizará para construir los modelos tanto para la clasificación multiclase como para la clasificación multilabel. Además, se hará una explicación de los parámetros prefijados para aceptar el modelo, donde se establecen unos valores a modo de umbral que los resultados de nuestro modelo tendrían que superar para considerarse óptimos. Por último, se describirán las métricas que han sido utilizadas para comprobar la efectividad de los modelos implementados.

El último punto que se tratará será una explicación sobre el preprocesamiento y adaptación que ha necesitado el dataset para acomodar el volumen de datos del que se disponía a los recursos que proporcionaba el entorno de desarrollo de Google Colab. Mediante un análisis del dataset para hacer un acercamiento a este, se proponen una serie de estrategias que se adoptan para llevar a cabo este fin.

4.1 Tecnologías y herramientas empleadas

Se pasa a exponer las principales tecnologías y herramientas empleadas a lo largo del proyecto, donde se incluye como apartados el lenguaje de programación, entorno de desarrollo y las diferentes librerías empleadas.

4.1.1 Lenguaje de programación

El proyecto ha sido desarrollado en su completitud en el lenguaje de programación Python. La elección de este lenguaje en concreto reside en las múltiples ventajas que ofrece, contando con un número extenso de librerías diseñadas para realizar labores de machine learning y Deep Learning de forma más automática y sencilla, haciendo de este lenguaje el ecosistema idóneo para un proyecto orientado hacia labores de Inteligencia Artificial, como se enmarca el nuestro.

4.1.2 Entorno de desarrollo

El entorno de desarrollo seleccionado ha sido Google Colab. Google Colab es un entorno de desarrollo en la nube que proporciona una plataforma gratuita para ejecutar código en lenguaje Python y R.

Colab se basa en Jupyter Notebook, que es una aplicación web de código abierto que permite crear y compartir documentos interactivos que contienen código en diferentes lenguajes de programación. La ventaja de Colab es que se ejecuta en la nube de Google, lo que significa que no es necesario instalar nada en tu computadora local y puedes acceder a tus proyectos y archivos desde cualquier dispositivo con acceso a Internet.

Las principales características que aporta son:

- **Ejecución gratuita en la nube.** Colab proporciona acceso gratuito a máquinas virtuales con recursos computacionales, como CPU, GPU y RAM, lo que permite ejecutar código sin la necesidad de poseer hardware potente.
- **Acceso a bibliotecas y herramientas.** Google Colab viene preinstalado con muchas bibliotecas y herramientas populares de Python, como TensorFlow, Keras y NumPy, lo que facilita el desarrollo de proyectos de aprendizaje automático y análisis de datos.
- **Integración con otros servicios de Google.** Se integra con otros servicios de Google, como Google Drive, lo que te permite importar y exportar fácilmente archivos desde y hacia tu unidad de Google.

Todas estas ventajas han sido incluidas en el proyecto y han facilitado el camino hacia el éxito de este.

Concretando en las características específicas de los recursos ofrecidos por Google Colab, simplificados en memoria RAM y GPUs, se distinguen dos conjuntos. Se puntualiza que, aunque el uso de TPU es a priori mucho más rápido y eficiente que el de GPU, en Google Colab se ha comprobado que la TPU ofrecida proponía 6s/step de media en cada época, siendo esta reducida a 2s/step utilizando la GPU que facilitaban. Este hecho es el que ha producido que se escogiera la tecnología de Unidad de Procesamiento Gráfico (GPU) en vez de La Unidad de Procesamiento Tensorial (TPU).

- **Versión gratuita.**
 - **RAM.** La cantidad de RAM (memoria de acceso aleatorio) disponible en Colab puede variar, pero suele ser alrededor de 12 GB a 25 GB. En nuestro caso se ha tenido una memoria RAM prefijada en 15 GB normalmente en todos los usos que se utilizado.
 - **GPU.** La versión gratuita de Colab proporciona acceso a una GPU en la mayoría de los casos, aunque en el último mes vimos enormemente reducido nuestro acceso a esta tecnología. La GPU disponible en la versión gratuita es la T4, que proporciona una RAM de GPU de 15 GB.
- **Versión Pro.**
 - **RAM.** Colab Pro proporciona acceso a una mayor cantidad de RAM en comparación con la versión gratuita, siendo esta aumentada hasta el valor de 25 GB de capacidad de memoria RAM

- **GPU.** Colab Pro ofrece acceso a GPUs más potentes y de mayor rendimiento en comparación con la versión gratuita. Esto puede incluir GPU NVIDIA Tesla A100 o V100, que son más rápidas y adecuadas para proyectos de aprendizaje automático y otros cálculos intensivos. Sin embargo, se ha comprobado que estas GPUS, catalogadas como Premium, agotan rápidamente nuestras unidades de computación, por lo que su uso prácticamente ha sido nulo, ya que no se ha podido acceder a ellas en la mayoría de ocasiones. Sin embargo, el acceso a la GPU T4 ha sido siempre inmediato, por lo que con esta versión se aseguraba acceso a una GPU siempre que se necesitase.
- Además, cabe mencionar una gran diferencia respecto a la versión gratuita. Los **tiempos de ejecución** en este modo de pago son 24 horas, es decir, la ejecución de la celda en cuestión sólo será interrumpida si se excediese durante su interacción los límites de memoria RAM disponibles. Sin embargo, en la versión gratuita, el límite de tiempo por ejecución era de 39 minutos, por lo que dificultaba un número de épocas amplio.

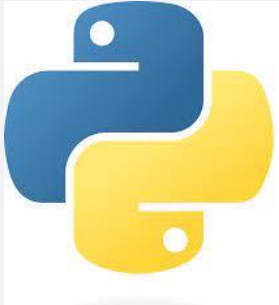



4.1.3 Librerías utilizadas

Han sido varias las librerías utilizadas que han facilitado la labor del proyecto, haciendo más liviano el esfuerzo empleado y siendo más efectivo el desarrollo producido. A continuación, se detallan las librerías en cuestión.

- **Scikit-learn.** Constituye una biblioteca de aprendizaje automático en Python que ofrece una amplia gama de algoritmos y herramientas para tareas de minería de datos y análisis predictivo. Permite la construcción y entrenamiento de modelos de aprendizaje automático para clasificación, regresión, clusterin, etc. Incluye funciones para el preprocesamiento de datos, evaluación de modelos y selección de características. Es ampliamente utilizada y valorada por su facilidad de uso, documentación detallada y comunidad activa. Ha sido utilizada para llevar a cabo el preprocesado de las etiquetas en la clasificación multiclase.
- **TensorFlow.** Es una librería de código abierto para el aprendizaje automático y la inteligencia artificial. Proporciona un entorno flexible y eficiente para la construcción y entrenamiento de redes neuronales y modelos de aprendizaje profundo. TensorFlow permite el procesamiento de datos en forma de tensores, lo que facilita el trabajo con estructuras multidimensionales y el despliegue en una amplia gama de dispositivos y plataformas. Posibilita el soporte para la siguiente librería.
- **Keras.** Librería de aprendizaje profundo de código abierto que se ejecuta sobre TensorFlow. Proporciona una interfaz de alto nivel y fácil de usar para construir y entrenar redes neuronales. Keras es conocido por su simplicidad y legibilidad, permitiendo a los desarrolladores crear modelos de aprendizaje profundo de manera rápida y eficiente, siendo compatible con una amplia gama de arquitecturas de redes neuronales y utilizado ampliamente en aplicaciones de visión por computadora, procesamiento del lenguaje natural, etc. Es la librería encargada de llevar el peso de toda la construcción de los modelos mediante redes neuronales convolucionales, desde la

estructura de estas hasta los optimizadores y diferentes parámetros utilizados para la implementación total del modelo.

- **NumPy.** NumPy es una biblioteca fundamental en Python para el procesamiento numérico y científico. Proporciona un objeto de matriz multidimensional eficiente, llamado "ndarray", que permite realizar operaciones matemáticas y manipulaciones de datos de manera rápida y eficiente. NumPy ofrece funciones para realizar operaciones algebraicas, estadísticas, transformadas, entre otras. Además, se integra adecuadamente con otras bibliotecas de análisis de datos y aprendizaje automático, como pandas y scikit-learn, lo que lo convierte en una herramienta esencial en el ecosistema de Python para la computación numérica. Esencial para el tratamiento numérico y operaciones matriciales.
- **Pandas.** Librería de código abierto en Python para la manipulación y análisis de datos. Proporciona estructuras de datos flexibles y eficientes, como los DataFrames, que permiten almacenar y manipular datos tabulares de manera intuitiva. Pandas ofrece una amplia gama de funcionalidades para filtrar, ordenar, transformar y combinar datos, así como para realizar cálculos estadísticos y manipulaciones complejas. Es ampliamente utilizado en tareas de preparación y limpieza de datos, exploración y visualización de datos, y análisis de datos en general. En nuestro caso, no ha sido utilizado para una limpieza de datos, sino para posibilitar su adaptación para una posterior representación.
- **Matplotlib.** Es una biblioteca de visualización de datos en Python que permite crear gráficos de alta calidad de manera flexible y personalizable. Proporciona una amplia variedad de gráficos, como gráficos de líneas, barras, dispersión, histogramas, entre otros. Matplotlib desarrolla una buena integración con otras bibliotecas de análisis de datos en Python, como NumPy y Pandas, lo que facilita la representación visual de datos. Además, ofrece opciones para personalizar detalles como colores, etiquetas, títulos y leyendas. Es una herramienta clave para explorar y comunicar información visualmente a partir de datos. Su uso en nuestro proyecto se ha extendido a todas las representaciones que se han realizado, mostrando su eficiencia y adaptación a las necesidades que se requería.
- Otras librerías utilizadas han sido OpenCV, para el procesamiento de las imágenes, la biblioteca Glob, Os y Shutil, herramientas muy útiles para la búsqueda y eliminación de archivos y directorios con un patrón específico, esencial en la creación de subdirectorios para organizar nuestras imágenes; así como la librería Zipfile, determinante para extraer el dataset comprimido al sincronizar con Google Drive.

Librería	Icono	Descripción
Python		Lenguaje de programación de alto nivel, versátil y fácil de aprender, conocido por
Google Colab		Entorno de programación en la nube que permite ejecutar y colaborar en notebooks de Jupyter de forma gratuita.
Scikit-learn		Biblioteca de aprendizaje automático de código abierto para Python que proporciona una amplia gama de algoritmos y herramientas para la minería y el análisis de datos.
TensorFlow		Biblioteca de aprendizaje automático de código abierto desarrollada por Google que proporciona un marco flexible para la construcción y entrenamiento de modelos de inteligencia artificial, especialmente redes neuronales.
Keras		Biblioteca de aprendizaje profundo de código abierto que se ejecuta sobre TensorFlow, que simplifica y acelera el desarrollo de modelos de aprendizaje automático al ofrecer una interfaz de alto nivel y modularidad.

Numpy		Biblioteca de Python que proporciona soporte para trabajar con arreglos multidimensionales y funciones matemáticas de alto rendimiento. Es ampliamente utilizado en el ámbito científico y de la computación numérica debido a su eficiencia y facilidad de uso.
Pandas		Biblioteca de Python que ofrece estructuras de datos y herramientas para el análisis y manipulación de datos, facilitando tareas como la limpieza, transformación y exploración de conjuntos de datos tabulares.
Matplotlib		Biblioteca de visualización de datos en Python, ideal para crear gráficos y representar información de manera clara y efectiva.
OpenCV		Biblioteca de visión por computadora en código abierto, utilizada para el procesamiento y análisis de imágenes y videos.

Tabla 9. Tecnologías empleadas.

4.2 Aspectos generales de los modelos de clasificación

En esta sección, se pasa a identificar las diferencias entre los diferentes tipos de clasificación y a desarrollar el esquema básico que hay que seguir a la hora de construir un modelo de clasificación mediante redes neuronales convolucionales, así como los parámetros establecidos para garantizar un grado de satisfacción óptima del modelo, acorde al volumen de datos y los recursos de los que se disponen.

4.2.1 Clasificación multiclase y clasificación multilabel

Para el completo entendimiento de los modelos posteriores, es de vital importancia tener claro cuáles son los tipos de clasificación que se está dispuestos a llevar a cabo y cuáles son las principales diferencias que proponen uno frente al otro.

Se presentan los tipos de clasificación diferentes que existen, siendo utilizados para este proyecto la clasificación multiclase y la clasificación multilabel.

- **Clasificación binaria.** La clasificación binaria es un tipo de problema de clasificación en el que se buscan asignar instancias de datos a una de dos clases posibles. La última capa del modelo generalmente consiste en una sola neurona con una función de activación sigmoide, que produce una salida en el rango de 0 a 1, indicando la probabilidad de pertenecer a una clase determinada.
- **Clasificación multiclase.** La clasificación multiclase es un problema en el que se buscan asignar instancias de datos a una de varias clases posibles, donde el número de clases es mayor a dos. En este caso, la última capa del modelo generalmente consiste en un número de neuronas igual al número de clases, y se utiliza una función de activación de *softmax*, que produce una distribución de probabilidad para todas las clases, asegurando que la suma de las probabilidades sea igual a 1.
- **Clasificación multilabel.** Consiste en un tipo de problema en el que se busca asignar múltiples etiquetas a una instancia de datos, en lugar de una única clase. En este caso, la última capa del modelo también consiste en un número de neuronas igual al número de etiquetas, pero se utiliza una función de activación sigmoide en cada neurona, permitiendo que cada etiqueta tenga una probabilidad independiente de estar presente o no.

A continuación, se pasa a exponer las diferencias a la hora de los diferentes usos que tienen entre la clasificación multiclase y multilabel.

- **Clasificación multiclase.** Este enfoque se utiliza cuando se busca asignar una sola clase a cada instancia de datos. Se aplica en casos como la clasificación de imágenes en categorías exclusivas, reconocimiento de dígitos escritos a mano, clasificación de documentos en diferentes temas, entre otros. En nuestro caso, para clasificar las imágenes en el tipo de comida que representan.
- **Clasificación multilabel.** En contraposición, la clasificación multilabel se utiliza cuando se busca asignar múltiples etiquetas a cada instancia de datos. Se aplica en casos en los que una instancia puede tener diferentes características o atributos simultáneamente. Algunos ejemplos de uso son el etiquetado de documentos con múltiples temas, clasificación de imágenes con múltiples etiquetas de contenido, recomendación de productos con múltiples categorías relevantes, entre otros. En el caso que nos concierne, se utiliza para la detección de múltiples ingredientes en un mismo plato de comida

4.2.2 Construcción base de los diferentes modelos de clasificación

Antes de la explicación de los diferentes modelos que se han implementado a lo largo del desarrollo del proyecto, se pasa a comentar el esquema que siguen los modelos de clasificación multiclase y multilabel en redes neuronales convolucionales. Para la implementación de estos modelos de clasificación, se debe seguir los siguientes procedimientos.

- **Separación del conjunto de datos en entrenamiento y prueba.** La separación del conjunto de datos en entrenamiento y prueba es necesaria para evaluar el rendimiento y generalización de un modelo de aprendizaje automático. El conjunto de entrenamiento se utiliza para ajustar los parámetros del modelo y aprender patrones en los datos, mientras que el conjunto de prueba, que contiene datos no vistos durante el entrenamiento, se utiliza para evaluar el rendimiento objetivo del modelo en situaciones reales y para detectar problemas de sobreajuste. Es esencial para una evaluación objetiva y precisa del modelo antes de aplicarlo a datos no vistos previamente.

La separación de los datos puede incluir una división adicional de validación, que ayuda a medir la capacidad de generalización del modelo y el ajuste de hiperparámetros. Esta nueva división se lleva a cabo para la implementación de los modelos por parte del usuario. En el caso que nos concierne, se lleva a cabo la clasificación multiclase mediante redes pre entrenadas, no es necesario ese ajuste de hiperparámetros, ya que estos han sido previamente testados mediante múltiples pruebas.

- **Elección de la red neuronal convolucional.** En esta fase, se lleva a cabo la implementación de la CNN que sustentará las bases de la clasificación. Para ello, existen dos alternativas.
La primera consiste en que el usuario establezca su propia estructura de la CNN. En este caso, el individuo es el máximo responsable de la composición de la red y tiene la máxima libertad posible para implementar una red neuronal acorde a sus intereses y sus necesidades. Como se ha comentado anteriormente, con esta estrategia el desarrollador se ve envuelto en la problemática de realizar un ajuste hiperparámetros por medio de pruebas hasta garantizar que el resultado es el más óptimo posible.
La otra alternativa consiste en seleccionar una CNN pre entrenada de las ya existentes, que a priori garantizarán un gran rendimiento debido a su eficiencia demostrada en aplicaciones con gran multitud de datos. Existen varios modelos de redes pre entrenadas, donde se pasa a describir algunas de las más comúnmente utilizadas. (Lerch, 2018)

- **VGG16.** Es un modelo de red neuronal convolucional ampliamente utilizado en el campo del aprendizaje automático y la visión por computadora. Fue desarrollado por el equipo de investigadores de Visual Geometry Group (VGG) en la Universidad de Oxford. La arquitectura VGG16 consta de 16 capas, incluyendo capas convolucionales y capas completamente conectadas. Esta red se ha entrenado previamente en conjuntos de datos masivos, como ImageNet, que contiene millones de imágenes en diferentes categorías. Debido a su entrenamiento exhaustivo, la VGG16 puede extraer características de imágenes de manera eficiente y se utiliza como base para la transferencia de

aprendizaje en tareas de clasificación de imágenes y reconocimiento visual. Constituye la red que se utiliza en los modelos.

- **ResNet50.** ResNet50 es una red neuronal convolucional profunda que consta de 50 capas. Se basa en la arquitectura de las redes residuales, que permiten entrenar redes más profundas con un rendimiento mejorado. Es ampliamente utilizada en tareas de visión por computadora, como la clasificación de imágenes y la detección de objetos. Gracias a su arquitectura profunda, ResNet50 es capaz de extraer características muy precisas de las imágenes, lo que le permite lograr un alto rendimiento en una variedad de tareas.
- **Inceptionv3.** Inceptionv3 es otra red neuronal convolucional profunda diseñada para la clasificación de imágenes y la detección de objetos. Fue desarrollada por Google como parte de la serie Inception de arquitecturas de redes, destacando por su capacidad para capturar características en diferentes escalas utilizando módulos de convolución llamados "Inception modules". Estos módulos combinan múltiples operaciones de convolución con diferentes tamaños de filtro para capturar características a diferentes niveles de detalle. Inceptionv3 es conocida por su rendimiento sobresaliente y ha sido utilizada en muchas competiciones de reconocimiento de imágenes.
- **Xception.** Consiste en una red neuronal convolucional profunda que se inspira en la arquitectura Inception. Su nombre es una combinación de "eXtended PerceptioN" y "Exceptional Accuracy". Xception se caracteriza por su enfoque extremadamente profundo y su capacidad para capturar relaciones de características a diferentes escalas. La arquitectura Xception utiliza operaciones de convolución separables en lugar de convoluciones tradicionales, lo que reduce significativamente la cantidad de cálculos y parámetros requeridos. Esto hace que Xception sea más eficiente en términos computacionales y permita un entrenamiento más rápido.
- **Aplicación de la técnica de Fine-tuning para redes pre entrenadas.** Esta técnica representa un enfoque que consiste en tomar una red neuronal pre entrenada en un conjunto de datos grande y luego ajustarla a un conjunto de datos más específico. Se procede a congelar todas las capas junto con sus pesos y se descongelan y se modifican los últimos niveles de la red, seleccionables por el desarrollador, para adaptarla a la nueva tarea. Al utilizar el conocimiento previo de la red pre entrenada, se acelera el proceso de entrenamiento y se logra un mejor rendimiento en la nueva tarea con menos datos. Es una estrategia eficaz para transferir el aprendizaje de una tarea relacionada a otra y se utiliza ampliamente en tareas de visión por computadora como la clasificación de imágenes y la detección de objetos.
- **Uso de las técnicas para combatir el sobreajuste.** En el caso propuesto, dos serán las técnicas que se implementan para advertir de la aparición del denominado *overfitting*.
 - **Drop Out.** Consiste en una técnica de 'apagado de neuronas' de la capa anterior. Esto significa que, durante el entrenamiento, el tanto por ciento

establecido en la capa de Drop out de las activaciones generadas en la capa antecedente, serán descartas de forma aleatoria. La raíz de este método es provocar un mayor esfuerzo de entrenamiento por parte de la red neuronal, favoreciendo el aprendizaje de patrones generales, para evitar que memorice características específicas de los datos de entrenamiento. Se establece el número aplicado al porcentaje de apagado igual al 20%, debido a los buenos resultados que esta cifra aporta a los modelos convolucionales.

- **Data Augmentation.** Es la segunda técnica aplicada para combatir el sobreajuste. Consiste en una serie de modificaciones aleatorias a los datos de entrenamiento, generando nuevas muestras similares a los datos originales, pero con pequeñas alteraciones. De esta forma, conseguimos aumentar el conjunto de entrenamiento, propiciando el aprendizaje de nuevos patrones que identifiquen características nuevas para la detección, en el caso del proyecto, de los diferentes platos de comida o ingredientes.
- **Elección del optimizador.** Un optimizador en el contexto de clasificación mediante redes neuronales convolucionales (CNN) es un algoritmo utilizado para ajustar los pesos y los sesgos de la red durante el entrenamiento con el objetivo de minimizar la función de pérdida y mejorar el rendimiento de la clasificación. Definimos a continuación los optimizadores más comúnmente utilizados.
 - **Descenso del gradiente estocástico (SGD).** Es un algoritmo básico de optimización que actualiza los pesos en la dirección opuesta al gradiente de la función de pérdida. Puede requerir una cuidadosa selección de la tasa de aprendizaje para lograr una convergencia eficiente.
 - **RMSprop.** Es un optimizador que utiliza una tasa de aprendizaje adaptativa por parámetro. Ajusta la tasa de aprendizaje en función del promedio móvil de los gradientes cuadráticos anteriores.
 - **Adagrad.** Es un optimizador que adapta la tasa de aprendizaje de cada parámetro en función del historial acumulado de gradientes. Tiene en cuenta el comportamiento de cada parámetro en el entrenamiento anterior.
 - **Adam (Adaptive Moment Estimation).** Es un algoritmo de optimización que combina el descenso de gradiente estocástico (SGD) con adaptación de tasa de aprendizaje individual por cada parámetro y ajuste de momento, logrando con esto un ajuste dinámico y preciso. Utiliza momentos de primer y segundo orden para adaptar la tasa de aprendizaje a la geometría local del espacio de parámetros. Además, normaliza los momentos acumulados para evitar sesgos hacia cero en las primeras etapas del entrenamiento. Ha sido el optimizador utilizado.
- **Creación de callbacks.** La técnica de callback es una herramienta utilizada durante el entrenamiento de redes neuronales convolucionales (CNN) para realizar acciones específicas en momentos determinados. Un callback puede ser utilizado para detener el entrenamiento si se alcanza cierto criterio de convergencia o para guardar los pesos

del modelo en determinados puntos. Esta cuenta con varios parámetros, de los que se describen los dos más usados e implementados en nuestros modelos.

- **Parámetro de paciencia.** El uso de la paciencia implica establecer un umbral de mejora mínimo y detener el entrenamiento si no se cumple después de un número determinado de épocas.
- **Restore_best_weights.** Permite restaurar los pesos del modelo a la mejor versión guardada durante el entrenamiento, basándose en una métrica de evaluación específica, como la precisión o la pérdida. Esto garantiza que el modelo utilice los pesos óptimos alcanzados durante el entrenamiento, en lugar de los últimos pesos, lo cual puede ser útil para evitar el sobreajuste y obtener el mejor rendimiento en la tarea deseada.
- **Entrenamiento del modelo creado.** Para el entrenamiento se recurre al método *modelo.fit()*, el cual realiza iteraciones llamadas épocas, donde cada época procesa una vez todos los datos de entrenamiento. Durante cada época, los datos se dividen en lotes más pequeños, y la CNN realiza la propagación hacia adelante (forward propagation) para calcular las predicciones y luego realiza la propagación hacia atrás (backpropagation) para ajustar los pesos y sesgos mediante el algoritmo de optimización elegido, como el descenso del gradiente estocástico (SGD). Los parámetros que se le ofrecerán a este método son los siguientes:
 - **X:** Datos de entrada para el entrenamiento.
 - **y:** Etiquetas correspondientes a los datos de entrada. Dependiendo del tipo de clasificación, estas etiquetas serán independientes entre sí (clasificación multiclase) o relacionadas entre sí (clasificación multilabel).
 - **batch_size:** Tamaño del lote de datos utilizado en cada iteración de entrenamiento.
 - **epochs:** Número de épocas, es decir, el número de veces que se iterará sobre los datos de entrenamiento.
 - **validation_data:** Datos de validación utilizados para evaluar el rendimiento del modelo en cada época.
 - **callbacks:** Lista de callbacks utilizados para personalizar el comportamiento durante el entrenamiento, como la detención anticipada o el guardado de pesos.
 - **steps_per_epochs:** indica el número de pasos o iteraciones que se realizarán en cada época durante el entrenamiento de una red neuronal.
 - **validation_steps:** aplica la misma definición que el parámetro anterior pero aplicado al conjunto utilizado como validación.
 - Otros parámetros que contempla este método pero que no han sido utilizados son *verbose*, *shuffle* e *initial_epoch*.

4.2.3 Métricas empleadas

A continuación, se pasa a definir la función de pérdida utilizada en los diferentes modelos implementados, así como las distintas métricas utilizadas para el cálculo del rendimiento y efectividad real de estos.

- **Función de pérdida o *loss*.** También conocida como función objetivo o función de error, es una medida que evalúa qué tan bien o mal está funcionando un modelo de aprendizaje automático durante el entrenamiento. Su objetivo principal es cuantificar la discrepancia entre las salidas predichas por el modelo y los valores reales de los datos de entrenamiento.
La elección de la función de pérdida depende del tipo de problema que se esté abordando, como clasificación, regresión o tareas específicas. A continuación, se describen algunas de las funciones de pérdida más comunes.
 - **Entropía cruzada binaria (Binary Cross-Entropy).** Es utilizada en problemas de clasificación binaria, donde se busca predecir una única clase de dos posibles. Esta función mide la discrepancia entre las probabilidades predichas y las etiquetas verdaderas, y busca minimizar la pérdida. Será la función establecida para el cálculo de los diferentes ingredientes, ya que estos se entenderán como la posibilidad de ser seleccionados o no, para permitir la selección de más de un tipo.
 - **Entropía cruzada categórica (Categorical Cross-Entropy).** Se utiliza en problemas de clasificación multiclase, donde se tienen más de dos clases mutuamente excluyentes. La función de pérdida compara las distribuciones de probabilidad predichas y las verdaderas para calcular la discrepancia. Es la empleada en la clasificación multiclase de los diferentes platos de comida.
 - **Pérdida cuadrática (Mean Squared Error - MSE).** Es comúnmente utilizada en problemas de regresión, donde se busca predecir un valor numérico continuo. La pérdida se calcula como la diferencia al cuadrado entre las predicciones y los valores reales.
- **Métrica *accuracy* (precisión).** Mide la proporción de muestras clasificadas correctamente sobre el total de muestras. Es una medida común para evaluar el rendimiento de un modelo de clasificación en el que, cuanto más alto sea el valor de *accuracy*, mejor es la capacidad del modelo para realizar predicciones correctas. Sin embargo, la precisión puede verse afectada por el desbalance de clases en los datos, por lo que es importante considerar otras métricas en situaciones de desbalance.
- **Métrica *recall* (recuperación).** La métrica *recall* mide la proporción de muestras positivas correctamente identificadas sobre el total de muestras positivas. Es especialmente útil en problemas de clasificación donde el objetivo es identificar correctamente todos los casos positivos. Un valor alto de *recall* indica que el modelo puede capturar la mayoría de las muestras positivas, pero esto no priva de la posible existencia que puede haber de un mayor número de falsos positivos.

- **F1-score.** La métrica F1-score combina las métricas de precisión y recall en una sola medida para evaluar el rendimiento de un modelo de clasificación, calculando la media armónica entre precisión y recall, lo que proporciona una medida equilibrada de la precisión y exhaustividad del modelo. Un valor alto de F1-score indica un buen equilibrio entre la precisión y el recall, lo que significa que el modelo tiene un buen rendimiento tanto en la identificación de muestras positivas como en la exclusión de muestras negativas. Es especialmente útil cuando hay un desequilibrio entre las clases.

4.3 Preprocesamiento y adaptación del dataset

En este apartado se lleva a cabo una descripción del conjunto de datos que se tienen, explicando su estructura y proponiendo un dataset adicional para un proyecto más ambicioso futuro. A su vez, se indica si ha sido necesario algún tipo de preprocesamiento y cómo ha sido este, y se cita a la vez que se describe las diferentes estrategias que se siguen para adaptar el conjunto de datos a las necesidades.

4.3.1 Análisis del dataset

La composición del dataset que se ha utilizado, llamado 'Food101', obtenido de la página empresarial 'Kaggle', consta de 101 platos de comida diferentes que contienen 1.000 imágenes, por lo que proporciona un volumen de 101.000 imágenes totales. En cuanto al conjunto de ingredientes, proporciona 446 ingredientes variados. Teniendo en cuenta que estos son cogidos de las recetas de cada uno de los platos correspondientes, se realiza un preprocesado y limpieza de marcas industriales y directrices que se encontraban en ellos, así como de algunos de los ingredientes menos comunes, obteniendo un listado final de 227 ingredientes, donde cada plato cuenta con una composición de entre 5 y 10 ingredientes, siendo la media en torno a 7.

Para ilustrar el dataset Food101, se pasa a exponer una serie de imágenes que facilitaran su comprensión.

- La primera imagen muestra un listado de todos los platos de comida disponibles en el conjunto de datos, ordenados alfabéticamente. Se comprueba que su totalidad son 101 tipos diferentes de comida que además ofrecen gran diversidad, ya que se puede encontrar varias categorías en las que se podrían relacionar las diferentes comidas, como carnes, hortalizas, postres y platos combinados, entre otras.

```
!ls /content/Food101/food-101/images
```

apple_pie	eggs_benedict	onion_rings
baby_back_ribs	escargots	oysters
baklava	falafel	pad_thai
beef_carpaccio	filet_mignon	paella
beef_tartare	fish_and_chips	pancakes
beet_salad	foie_gras	panna_cotta
beignets	french_fries	peking_duck
bibimbap	french_onion_soup	pho
bread_pudding	french_toast	pizza
breakfast_burrito	fried_calamari	pork_chop
bruschetta	fried_rice	poutine
caesar_salad	frozen_yogurt	prime_rib
cannoli	garlic_bread	pulled_pork_sandwich
caprese_salad	gnocchi	ramen
carrot_cake	greek_salad	ravioli
ceviche	grilled_cheese_sandwich	red_velvet_cake
cheesecake	grilled_salmon	risotto
cheese_plate	guacamole	samosa
chicken_curry	gyoza	sashimi
chicken_quesadilla	hamburger	scallops
chicken_wings	hot_and_sour_soup	seaweed_salad
chocolate_cake	hot_dog	shrimp_and_grits
chocolate_mousse	huevos_rancheros	spaghetti_bolognese
churros	hummus	spaghetti_carbonara
clam_chowder	ice_cream	spring_rolls
club_sandwich	lasagna	steak
crab_cakes	lobster_bisque	strawberry_shortcake
creme_brulee	lobster_roll_sandwich	sushi
croque_madame	macaroni_and_cheese	tacos
cup_cakes	macarons	takoyaki
deviled_eggs	miso_soup	tiramisu
donuts	mussels	tuna_tartare
dumplings	nachos	waffles
edamame	omelette	

Figura 21. 101 imágenes que contiene nuestro dataset.

- Se procede a visualizar y comprobar el conjunto de imágenes que se disponen, pasando a la representación de una imagen aleatoria de cada una de las diferentes clases y se muestra, agregándole como título a cada *subplot* el nombre del plato, y como 'pie de imagen' las dimensiones de esta. Este es el resultado obtenido:



Figura 22. Representación de 15 clases, por simplificar.

- Seguidamente, se realiza una comparación entre el archivo 'ingredients.txt' y el archivo 'ingredients_simplified.txt'. Como se puede apreciar, la lista de ingredientes simplificados es mucho más reducida y obvia términos como "Potatoes O'Brien", entendiéndolo como una marca, así como 'freshly ground' referido a 'black pepper', para eliminar todo aquello que no sea la esencia del ingrediente en sí.

- cubed bread,milk,white sugar,butter,salt,eggs,vanilla extract,heavy cream
- rolls,bacon,eggs,Potatoes O'Brien,green onions,shredded cheddar cheese,flour tortillas,salsa
- plum tomatoes,garlic,extra-virgin olive oil,balsamic vinegar,fresh basil leaves,salt,freshly ground black pepper,baguette
- garlic,plain greek yogurt,parmesan cheese,wochestershire sauce,dijon mustard,lemon juice,anchovy paste,salt,pepper,romaine lettuce,cROUTONS
- confectioners sugar,part-skim ricotta cheese,slivered almonds,semi-sweet chocolate morsels,amaretto liqueur,cannoli shells,unsweetened cocoa powder,cocktail cherries

Figura 23. 5 ingredientes de la lista de ingredientes sin simplificar.

- bread,milk,sugar,butter,salt,egg,vanilla
- rolls,bacon,egg,brie,onion,cheddar,flour,salsa
- plum,garlic,gin,balsamic vinegar,basil,salt,black pepper,baguette
- garlic,plain greek yogurt,cheese, Worcestershire,dijon mustard,lemon, anchovy,salt,pepper,lettuce,cROUTONS
- sugar,cheese,almond,chocolate,liqueur,cannoli shells,cocoa,cocktail

Figura 24. Los mismos 5 ingredientes simplificados.

En lo referente al segundo dataset, se analizó que era sumamente excesivo, ya que con el conjunto de datos 'Food101' existían suficientes combinaciones posibles para ejecutar, ya que era imposible hacer una clasificación de todo el volumen de datos con los recursos que se disponían. Por tanto, 'Recipes5k', que constituye realmente una ampliación de 'Food101', contaría con 4.826 recetas, por las 101 del anterior dataset, con un total de 3.213 ingredientes, con una media más de 10 ingredientes por plato. Como ya se ha comentado, esta elección de conjunto de datos estaría enfocada a unos planes futuros de expandir el proyecto mediante mayor posibilidad de recursos y así lograr el objetivo final de este proyecto: alcanzar el mayor número de ingredientes identificados en el mayor número de platos posibles para facilitar una ayuda a las personas con intolerancias alimentarias.

Por último, como breve puntuación, se aclara que el dataset originalmente contiene unos ingredientes por plato genéricos, sin hacer ningún énfasis en aquellos que podrían suponer un riesgo para la salud, por lo que la idea sería, una vez conseguido un buen rendimiento y precisión de nuestro modelo, extrapolar el proyecto a unos datos que contuvieran los ingredientes más propensos estadísticamente a producir algún tipo de intolerancia para poder ser identificados por cada plato. Además, se podrían crear varios datasets y modelos distintos en función de las comidas más típicas de cada país, facilitando una internacionalización de nuestra idea y posibilitando al usuario establecer su región para testear su comida dentro del modelo y datos entrenados para dicho país.

4.3.2 Preprocesamiento

Como preprocesamiento del dataset se puede citar que realmente no se ha alterado a priori ninguno de los datos de nuestro conjunto de datos, ya que está compuesto por imágenes que no han necesitado ningún tipo de procesamiento antes de realizar la creación de los diferentes modelos.

Sin embargo, aunque se lleve a cabo de una forma diferente a las transformaciones que se realizarían en un conjunto de datos numéricos o categóricos, sí que se aplica un preprocesamiento antes de llevar a cabo el entrenamiento del modelo, que se pasa a comentarlas a continuación.

Como en cualquier red neuronal convolucional, se necesita que todas nuestras imágenes tengan el mismo tamaño, ya que el modelo no puede entrenar imágenes con dimensiones diferentes. Para determinar los *shapes* de nuestros datos, se muestran las dimensiones en la parte inferior de cada imagen:



Figura 25. Se muestra sólo los primeros 4 elementos para poder apreciar las dimensiones.

Como se puede apreciar, todas las imágenes son a color, por lo que tienen 3 canales (RGB), y difieren entre ellas en cuanto al alto y largo de las fotografías. Debido a esta circunstancia, se debe prefijar un tamaño fijo que permita redimensionar todas las imágenes bajo esa escala, sin que esta sea relativamente pequeña para no perder excesiva información. Por tanto, se adopta las dimensiones de 240x240 por 3 canales, medidas que deben ser suficientemente grandes para que se pueda identificar la completitud de los ingredientes en cada plato, evitando dejar fuera del procesamiento una cantidad elevada de estos.

4.3.3 Adaptación del dataset

Por otro lado, debido al enorme conjunto de datos que suministraba 'Food101' en contraposición con los limitados recursos que la herramienta Pro de Google Colab proporcionaba, se ha tenido que abrir diferentes alternativas para encontrar la estabilidad entre volumen de imágenes y recursos para conseguir el modelo más eficiente posible. Seguidamente se presentan los enfoques que se han dado como solución a esta incógnita que se manejan.

- **Un modelo de clasificación multiclase limitando el número de platos a clasificar.** Inicialmente, y a método de llevar a cabo un primer acercamiento de entrenamiento del dataset, se lleva a cabo una clasificación multiclase, es decir, una clasificación de los platos de comida (no de los ingredientes) de las imágenes que se disponían, de modo que se llevaría a cabo distintas pruebas para conseguir clasificar el mayor número de platos posibles en un tiempo de cómputo razonable. La creación de un modelo de clasificación sólo multiclase se debe a que su clasificación es menos costosa computacionalmente por lo que inicialmente proporcionará mayor capacidad de jugabilidad con el número de clases a establecer, ya que cada clase (plato de comida) es independiente entre sí, en contraposición con las etiquetas (correspondiente a los ingredientes), que pueden ser compartidos entre varias clases.
- **Un modelo de clasificación multiclase limitando el número de imágenes por plato.** La segunda alternativa que se propone es estudiar el mismo caso que el anterior, una clasificación multiclase de platos de comida, pero en el que en este caso se limita el número de imágenes por cada plato, siendo inicialmente 1.000. Con esto, se busca clasificar el mayor número de platos posibles con el menor número de imágenes posibles, dentro de un espacio de tiempo razonable para la aplicación que se estaba adoptando. Este modelo, junto con el anterior, darán las claves de qué método es más efectivo para llevar a cabo la clasificación: si prescindir de platos de comida

directamente, o si evaluar el sacrificio de un porcentaje de imágenes por plato a cambio de obtener mayor número de platos clasificables dentro de los parámetros temporales establecidos.

Una vez analizados la eficiencia de ambos modelos, se extrapolará al objetivo final con la mejor estrategia estudiada.

- **Un modelo de clasificación multiclase donde la estructura de la red neuronal convolucional es creada por nuestra parte.** Para descifrar si los resultados que propondría una red neuronal convolucional CNN implementada de forma autónoma se conseguirían igualar o superar los resultados producido por una CNN previamente entrenada, se dedica este apartado, ya que, al trabajar con una clasificación más leve a nivel de cómputo, se observaría su rendimiento antes de someterla a mayor necesidad de recursos.
- **Un modelo de clasificación multilabel de los ingredientes mediante una red neuronal convolucional pre entrenada.** Una vez escogido el criterio a seguir, se pasa a atacar directamente el problema: la clasificación de los ingredientes o etiquetas, que constituyen los *labels*. Para ello, se escogen una de las redes entrenada previamente que existen, para conseguir con ello un porcentaje de acierto en nuestras clasificaciones lo más alto posible y detectar los diferentes ingredientes con la mayor precisión alcanzable.
- **Un modelo de clasificación multilabel de los ingredientes mediante una red neuronal convolucional creada completamente por nosotros.** Con el fin de evidenciar si se es capaz de crear un modelo pudiera superar los valores de las métricas que ofrece el caso anterior, se llevará a cabo una implementación de un modelo propio que intente superar los resultados con mayores rendimientos. Además, de no cumplir nuestro objetivo, esto servirá para evidenciar la diferencia que existe entre una red pre entrenada, donde se comprobó la efectividad de su composición mediante la clasificación de un número muy alto de imágenes, y una red que desde nuestro propio conocimiento se considera que podría producir unos resultados suficientemente óptimos.

5 Resultados

En esta parte del documento, se realiza la parte correspondiente a los resultados obtenidos de los diferentes modelos en las diferentes clasificaciones. Para ello, se determina un apartado para la explicación de los parámetros que actuarán a modo de umbral para delimitar la efectividad de los modelos, donde serán considerados como óptimos aquellos que modelos que presenten resultados por encima de estas cifras.

Seguidamente, se lleva a cabo el desarrollo de los diferentes modelos para cada tipo de clasificación. Inicialmente, se presenta la clasificación multiclase, correspondiente a los diferentes platos de comida, donde el principal objetivo será establecer cómo se comportan los modelos frente al volumen de datos que se tienen, y para determinar qué parámetros son los que mejores resultados proporcionan, para poder extrapolarlos a la clasificación multilabel. Posteriormente, se describen y se analizan los resultados producidos por el segundo tipo de clasificación, para la detección de los ingredientes en los platos de comida.

5.1 Parámetros prefijados para aceptar el modelo

Debido a que se necesita una adaptación al volumen de datos que se tiene y a los recursos que Google Colab proporciona, se debe prefijar unos parámetros temporales y métricos que actúen como umbral para aceptar la efectividad del modelo con respecto a la situación a la que se debe adaptar. Los términos que se busca para los modelos son:

- **Tiempo de cómputo** en torno a los 45 minutos, para no provocar una demora excesivamente larga, pero que permita recorrer una serie de épocas suficientes para poder dotar al modelo de suficiente entrenamiento.
- Un **valor de precisión** en el conjunto de prueba superior al 75%, ya que se necesita una alta precisión que ayude a detectar, posteriormente, el mayor número de platos o ingredientes posibles.

5.2 Desarrollo de los diferentes modelos. Clasificación multiclase

Este apartado se dedica para la realización de los diferentes modelos que se implementarán para la clasificación multiclase, clasificación dedicada para la catalogación de los diferentes platos de comida.

5.2.1 Un modelo de clasificación multiclase limitando el número de platos a clasificar

Primer modelo de clasificación del tipo multiclase, en el que la estrategia es limitar el número de platos. Su descripción se dividirá en una primera parte de implementación, donde se desarrollará la parte correspondiente a la creación del modelo; y una segunda parte de

test y predicciones, donde se evaluará el rendimiento del modelo mediante las métricas establecidas.

5.2.1.1 Implementación

En este primer apartado, se cita todo lo relativo al correcto funcionamiento del entrenamiento de nuestro modelo. Se tratan aspectos como el número de clases seleccionadas, una descripción del método holdout, la implementación de la estructura del modelo y las técnicas para combatir el *overfitting*, así como el uso de un generador y la selección del clasificador. Por último, se expondrá el entrenamiento realizado y las visualizaciones de los resultados.

5.2.1.1.1 Número de clases seleccionadas

Para la limitación de las clases diferentes del modelo, se crea una función donde el parámetro será el número de clases que se quiere clasificar, seleccionadas de forma aleatoria, y el resultado serán los diferentes platos de comida elegidos. De la lista de comidas inicial se elimina el archivo 'DS_Store', debido a no contener imágenes de una única categoría.

Para encontrar un modelo acorde a los parámetros temporales y métricos que se establece como óptimos, se lleva a cabo una serie de entrenamientos variando el parámetro diferencial en esta estrategia utilizada, el número de clases a clasificar. En la siguiente tabla se muestran cuáles han sido los rendimientos obtenidos tras llevar a cabo varios entrenamientos del modelo. Cabe mencionar que estos resultados dependen de los datos entrenados en ese momento, por lo que no proponen una ciencia cierta invariable. Se lleva a cabo hasta 5 pruebas diferentes para encontrar el mayor número de clases posibles a clasificar.

Prueba nº	Número de clases N	Accuracy	Tiempo de duración
1	15	0.8213	40 min
2	25	0.7574	1h 03 min
3	20	0.8388	44 min
4	22	0.7980	1h 04 min
5	21	0.8257	59 min

Número de clases seleccionadas: 20

Tabla 10. Parámetros para determinar el modelo 1 de clasificación multiclase

- En la prueba número 1, estableciendo el número de clases en 15, se puede comprobar cómo tanto el accuracy como el tiempo de duración eran valores óptimos para nuestro modelo, según se había establecido. Por tanto, se consideró como ejemplo óptimo, pero se buscó subir las clases para acercarse al máximo número de clases clasificables.
- Siguiendo con esta idea, se realiza la prueba número 2, en la que se aumentó el número de clases hasta el valor de 25, el accuracy ofrecido superaba el umbral establecido, de 75%, pero el tiempo de ejecución por encima del marcado suponía que este modelo tuviera que ser desechado.
- Seguidamente, se implementó un modelo con 20 clases, que ofreció unas prestaciones de accuracy bastante altas y el tiempo se enmarcaba dentro del óptimo. Por tanto, viendo el buen rendimiento, se concluyó que este era el mayor número de clases clasificables que el modelo podía soportar dentro de las circunstancias establecidas.
- La cuarta prueba se realiza con un total de 22 clases y, aunque el accuracy superó el umbral marcado, el tiempo de ejecución se disparó hasta los 57 minutos, por lo que se tuvo que descartar este modelo.
- Por último, en la última de las pruebas, se intentó aumentar en una clase respecto al mejor ejemplo al respecto. Sin embargo, aunque el accuracy presentaba resultados similares a los valores de este ejemplo, el tiempo de duración se elevó hasta 59 minutos, por lo que se determinó como tiempo excesivo

Por tanto, se determina que el número máximo de clases que nuestro modelo puede clasificar bajo las circunstancias preestablecidas es 20, ofreciendo un accuracy del 0.8388, donde una parada por sobreajuste y una gran precisión con respecto a modelo con número similar de clases hace que sea el número de clases elegidas.

Una vez seleccionadas las clases que formarán parte del modelo, se lleva a cabo una división del conjunto de datos en entrenamiento y prueba, denominado método *holdout*. En este caso, se establece una división del 80% para train y 20% para test. Se obviará el conjunto de validación ya que, como ya se comentó anteriormente, al implementar una red neuronal pre entrenada, una división de validación carece de sentido debido a que sus hiperparámetros ya están establecidos.

5.2.1.1.2 Método holdout

Una vez se tiene marcada la división, se crea un subdirectorio *multiclass_1* donde, a su vez, se añaden tres carpetas nuevas, *train* y *test*, donde se copian en estos directorios las imágenes correspondientes a cada división de los platos de comida seleccionados como clases anteriormente.

Para conseguir que las divisiones estén balanceadas, punto clave para el buen rendimiento del modelo, se opta por recorrer todas las clases disponible y por cada clase recorrida, llevar a cabo la división *holdout* de sus imágenes. Al copiar las imágenes por cada plato de comida en los dos tipos de subdirectorios, *train* y *test*, se garantiza una proporción correcta que se traducirá en el mismo número de imágenes en ambos conjuntos por cada clase. Para un mejor entendimiento de todo lo explicado anteriormente, se adjunta el código implementado para esta tarea.

```
# TRAIN/VAL/TEST: 80/20%
SPLIT_RATIO_TEST=0.80

for cl in CLASSES_mc1:
    # path to the images of class "cl"
    img_path = os.path.join(base_dir, cl)

    # we obtain the list of all the images
    images = glob.glob(img_path + '/*.jpg')

    # TRAIN & TEST (80%-10%)
    # determine how many images constitute 80%.
    num_train = int(round(len(images)*SPLIT_RATIO_TEST))

    # split the images into two sets (train, test)
    train, test = images[:num_train], images[num_train:]

    # we create the directory train/clase and test/clase
    if not os.path.exists(os.path.join(multiclass1_dir, 'train', cl)):
        os.makedirs(os.path.join(multiclass1_dir, 'train', cl))
    if not os.path.exists(os.path.join(multiclass1_dir, 'test', cl)):
        os.makedirs(os.path.join(multiclass1_dir, 'test', cl))

    # we move the pictures to their corresponding paths
    for t in train:
        shutil.copy(t, os.path.join(multiclass1_dir, 'train', cl))
    for tst in test:
        shutil.copy(tst, os.path.join(multiclass1_dir, 'test', cl))
```

Figura 26. Método *holdout* modelo 1 de clasificación multiclase.

5.2.1.1.3 Implementación del modelo. Estructura y reducción de overfitting

A continuación, tras asegurar que el método *holdout* está bien implementado, se procede a la construcción del modelo. Se ha decidido utilizar una red pre entrenada que a priori devolverá mejores resultados que un modelo que se crea de manera autónoma. Para ello, se ha seleccionado VGG-16, que será también la red utilizada para los sucesivos modelos, para imposibilitar que los resultados se vean alterados por una composición diferente en los modelos.

Para redimensionar las fotografías para que tengan una dimensión común, se establece como tamaño de las imágenes las dimensiones de 240x240 por 3 canales, por ser a color, y se realiza la técnica de Fine-tuning, con lo que se consigue evitar un aprendizaje innecesario de los pesos en las primeras capas, donde la red previamente entrenada habrá encontrado patrones muy generales que no aportarán información relevante, y entrenar las últimas capas del modelo, donde se detectan las características más notorias que permiten la distinción para la clasificación de unas imágenes con respecto a otras. El criterio adoptado ha sido acceder a entrenar las dos últimas capas del modelo.

Como ya se describió anteriormente, una red neuronal convolucional está compuesta por dos partes diferenciadas, siendo la primera la propiamente dicha como convolucional, compuesta por capas de convolución y pooling, que es aportada por la red pre entrenada VGG16, y una parte de capas interconectadas que formarían la transición entre la parte convolucional y la capa de salida. Por tanto, una vez implementado la primera parte de la red neuronal convolucional, es necesario llevar a cabo la segunda parte de la red.

Para ello, se necesita realizar una transformación unidimensional de los resultados matriciales que se han obtenido, dado a que las imágenes se representan en dos dimensiones. Con motivo de esto, es necesario añadir la capa de pooling *GlobalAverage2D()*, la cual solventa la problemática anteriormente comentada. A su vez, para ir disminuyendo en el número de neuronas hasta llegar a las necesarias para su clasificación, se añade una capa densa que desarrollará esta función, siendo de 256 neuronas, cuya cifra ha sido testeada y comprobada con otros ejemplos sobre un conjunto de datos similar. La función de activación será *relu*, debido a la buena eficacia estudiada que presenta. Además de estas, añado una capa de Drop out para combatir el sobreajuste, capa que ya se explicó anteriormente.

Por último, queda establecer la capa de salida. Debido a que se está en un problema de clasificación multiclase, la función de activación implementada será la de *softmax*, contando con un total de n neuronas de salida, siendo n el número de clases necesarias para su clasificación. De esta forma, se obtiene salidas que equivalen a probabilidades enmarcadas entre los valores 0 y 1, donde un valor cercano a uno querrá evidenciar que la probabilidad de que la imagen de input, en este caso plato de comida, pertenezca a dicha clase sea muy alta, siendo valores cercanos a 0 el caso contrario.

Por tanto, la estructura completa del modelo quedaría de la siguiente forma.

```
vgg_multiclass_1 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# We freeze the layers of the pretrained model and apply Fine-tuning.
for layer in vgg_multiclass_1.layers[:-2]:
    layer.trainable = False

# We add to the model a GlobalAveragePooling2D layer and two Denses(128,len(CLASSES))
x = GlobalAveragePooling2D()(vgg_multiclass_1.output)
x = Dense(256, activation='relu')(x)
# DROP OUT
x = Dropout(0.2)(x)
output = Dense(len(CLASSES_mc1), activation='softmax')(x)

vgg_model_multiclass_1 = Model(inputs=vgg_multiclass_1.input, outputs=output)
```

Figura 27. Estructura de CNN en el modelo 1 de clasificación multiclase.

Se incluye la otra técnica para combatir el sobreajuste, denominada Data Augmentation, estableciendo en las fotografías las siguientes modificaciones:

- Alteración en el grado de rotación
- Cambio en el ancho de la imagen
- Cambio en la altura de la imagen
- Aplicación de zoom
- Volteo horizontal

5.2.1.1.4 Generador y optimizador

Sucesivamente, se crea un generador para las imágenes de los diferentes conjuntos, donde se incluyen estas modificaciones y el preprocesamiento base aplicado por la red VGG16, como la normalización de los píxeles y la reordenación de los canales, tanto en el conjunto de train como en el de prueba. Para contextualizar en qué consiste un generador, se apunta una definición breve de este.

Un generador es una función que produce una secuencia de valores en lugar de almacenarlos en la memoria, generando los valores sobre la marcha cuando se solicitan. Los generadores ayudan a ahorrar memoria RAM ya que no necesitan almacenar toda la secuencia de valores en la memoria al mismo tiempo ya que, en lugar de eso, generan los valores bajo demanda, uno a la vez, lo que reduce la carga de memoria.

A continuación, se implementa el optimizador Adam, el cual obtiene eficientes resultados, con la función de pérdida *sparse_categorical_crossentropy*, función diseñada para el entrenamiento de modelos de clasificación multiclase donde el valor de las clases lo dan los subdirectorios en los que se encuentran las respectivas imágenes. Se centra la atención en la medida *accuracy*, o precisión, para determinar la efectividad del modelo. Tras varias pruebas para establecer el *learning rate*, elegimos el valor de 0,0001 como el valor que mejores prestaciones presenta.

Se construye los generadores de train y validación mediante los directorios donde se ubican las imágenes correspondientes a cada partición, se establece un tamaño de batch de 100 imágenes y se adopta el modo de clase *sparse*, acorde a la clasificación multiclase que se desea realizar. Se visualiza el total de fotografías pertenecientes a cada división, así como el número de clases totales que se está clasificando.

```
BS = 100
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(240,240),
                                                    batch_size=BS,
                                                    class_mode='sparse')
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(240,240),
                                                  batch_size=BS,
                                                  class_mode='sparse')

Found 16000 images belonging to 20 classes.
Found 4000 images belonging to 20 classes.
```

Figura 28. Generadores modelo 1 de clasificación multiclase.

5.2.1.1.5 Entrenamiento del modelo

Finalmente, se procede a entrenar nuestro modelo, estableciendo como parámetros:

- El generador de train implementado anteriormente.
- El generador de test.
- Un número de épocas que, tras varias comprobaciones, se establece a 15.
- Los valores de *steps_per_epochs* y *validation_steps*.
- Se prefija un 'callback' con paciencia igual a 2 y con capacidad de restaurar los mejores pesos tras agotarse la paciencia marcada.

Quedando así la implementación del modelo:

```
history_multiclass_1 = vgg_model_multiclass_1.fit(
    train_generator,
    steps_per_epoch=train_generator.n // BS,
    epochs=15,
    validation_data=test_generator,
    callbacks=[es_callback],
    validation_steps=test_generator.n // BS
)
```

Figura 29. Método *fit()* modelo 1 de clasificación multiclase.

La salida que ofrece el método *fit* del modelo constituye una representación época tras época de los diferentes resultados que va produciendo. Entre ellos, se procede a analizar las siguientes características.

- **Función de pérdida de entrenamiento, o *loss*.** Valor que indica la diferencia entre las predicciones que ofrece nuestro modelo al clasificar las diferentes clases, y los valores reales de estas. Este valor irá disminuyendo paulatinamente en el conjunto de entrenamiento, marcado por cada vez una mejor clasificación y un acercamiento más preciso de los datos.
- **Función de pérdida en el conjunto de test, o *val_loss*.** Presenta las mismas características que el apartado anterior, pero tiene un factor diferencial frente a la pérdida en entrenamiento: determina si se ha producido sobreajuste. Los valores de *val_loss* también se verán reducidos con el paso de las épocas, pero a diferencia de la función *loss*, este irá sobrepasando una serie de mínimos hasta alcanzar el mínimo absoluto, a partir del cual sus valores empezarán a aumentar lentamente. Esto marcará el principio del sobreajuste y se decidirá parar el modelo tras ver que en tantas épocas sucesivas como marcadas en el parámetro *paciencia*, su valor no ha conseguido descender por debajo del mínimo absoluto. El modelo recuperará los pesos de la época en la que la función *val_loss* obtuvo los resultados más bajos.

- **Accuracy.** Como ya se explicó anteriormente, calcula la proporción de muestras clasificadas correctamente sobre el total de muestras. Se representará su valor tanto para el conjunto de entrenamiento como para el conjunto de test o prueba.
- **Número de época.** El entrenamiento se realiza época por época, analizando y entrenando los datos que la componen y mostrando la información obtenida para cada una de ellas. La época ejecutada está marcada a la izquierda de cada fila de representación, evidenciando el número total de época establecido para nuestro modelo.
- **Tiempo total de ejecución por época.** Consiste en el tiempo empleada para la ejecución de cada una de las respectivas épocas. En este caso, este valor suele estar en torno a los 270 segundos, es decir, unos 4 minutos 30 segundos de media.
- **Steps_per_epochs,** o steps por época. Marca cuantas steps serán ejecutadas en cada época, y su valor dependerá de la cantidad de datos del conjunto de train y del tamaño de batch elegido. En este caso, su cifra es de 160.
- **El tiempo ejecutado por step.** Debido al uso de la GPU ofrecida por Google Colab, conseguimos un tiempo por step de 2 segundos aproximadamente que, si se multiplica por el número de steps por época que se tiene, da un tiempo total de ejecución de 320 segundos de media. Como se puede comprobar, su tiempo realmente es algo menor porque ofrece tiempos de ejecución menores al de la media.

A continuación, se detalla los resultados obtenidos en el entrenamiento. Como se puede apreciar, ambas curvas de las funciones de pérdida, tanto del conjunto de train como la del conjunto de test, tienen una orientación a reducir sus resultados. Sin embargo, a diferencia de la función de *loss*, el valor de *val_loss* llegará a un punto donde alcanza el mínimo absoluto y a partir de ahí empezará a crecer. En nuestro caso, este mínimo hace su aparición en la época número 8, indicando que, a partir de esta cifra, sus valores aumentarán y será el indicativo de que se está produciendo el denominado sobreajuste. Por tanto, se detiene el modelo tras dos épocas de no mejoría.

```
Epoch 1/15
160/160 [=====] - 271s 2s/step - loss: 2.3284 - accuracy: 0.4436 - val_loss: 1.0874 - val_accuracy: 0.6842
Epoch 2/15
160/160 [=====] - 266s 2s/step - loss: 1.1302 - accuracy: 0.6684 - val_loss: 0.8591 - val_accuracy: 0.7565
Epoch 3/15
160/160 [=====] - 271s 2s/step - loss: 0.8681 - accuracy: 0.7458 - val_loss: 0.7664 - val_accuracy: 0.7770
Epoch 4/15
160/160 [=====] - 268s 2s/step - loss: 0.7477 - accuracy: 0.7756 - val_loss: 0.6848 - val_accuracy: 0.8077
Epoch 5/15
160/160 [=====] - 271s 2s/step - loss: 0.6404 - accuracy: 0.8101 - val_loss: 0.6644 - val_accuracy: 0.8188
Epoch 6/15
160/160 [=====] - 267s 2s/step - loss: 0.5708 - accuracy: 0.8281 - val_loss: 0.6448 - val_accuracy: 0.8210
Epoch 7/15
160/160 [=====] - 268s 2s/step - loss: 0.5092 - accuracy: 0.8432 - val_loss: 0.6391 - val_accuracy: 0.8253
Epoch 8/15
160/160 [=====] - 269s 2s/step - loss: 0.4513 - accuracy: 0.8587 - val_loss: 0.5973 - val_accuracy: 0.8388
Epoch 9/15
160/160 [=====] - 267s 2s/step - loss: 0.4149 - accuracy: 0.8702 - val_loss: 0.5980 - val_accuracy: 0.8378
Epoch 10/15
160/160 [=====] - 267s 2s/step - loss: 0.3709 - accuracy: 0.8839 - val_loss: 0.6232 - val_accuracy: 0.8342
```

Figura 30. Entrenamiento del modelo 1 de clasificación multiclase.

5.2.1.1.6 Visualizaciones

En el apartado de visualizaciones, se lleva a cabo una representación de los resultados obtenidos, que se mostrarán a partir de gráfica donde se visualizará las funciones de pérdida y las métricas analizadas.

Comenzando por las funciones de pérdida, se observa cómo la gráfica de *loss* va disminuyendo su valor hasta ir tomando cada vez valores más cercanos al 0, mientras que la función de *val_loss* replicará a su compañera de entrenamiento hasta alcanzar un mínimo absoluto, producida en la época 8, valor 7 de la x. Los dos sucesivos valores mostrados corresponderán a los resultados ofrecidos por las diferentes gráficas hasta que la paciencia establecida se agotase.

Presentando seguidamente la descripción sobre las gráficas de *accuracy*, se puede presenciar que ambas toman valores muy parecidos durante todo el recorrido posible marcado por las épocas, simplemente diferenciándose al principio, donde para épocas primerizas el *val_accuracy* muestra resultados mejores que su métrica respectiva de entrenamiento, pero que en las últimas épocas consigue imponer resultados ligeramente más óptimos.

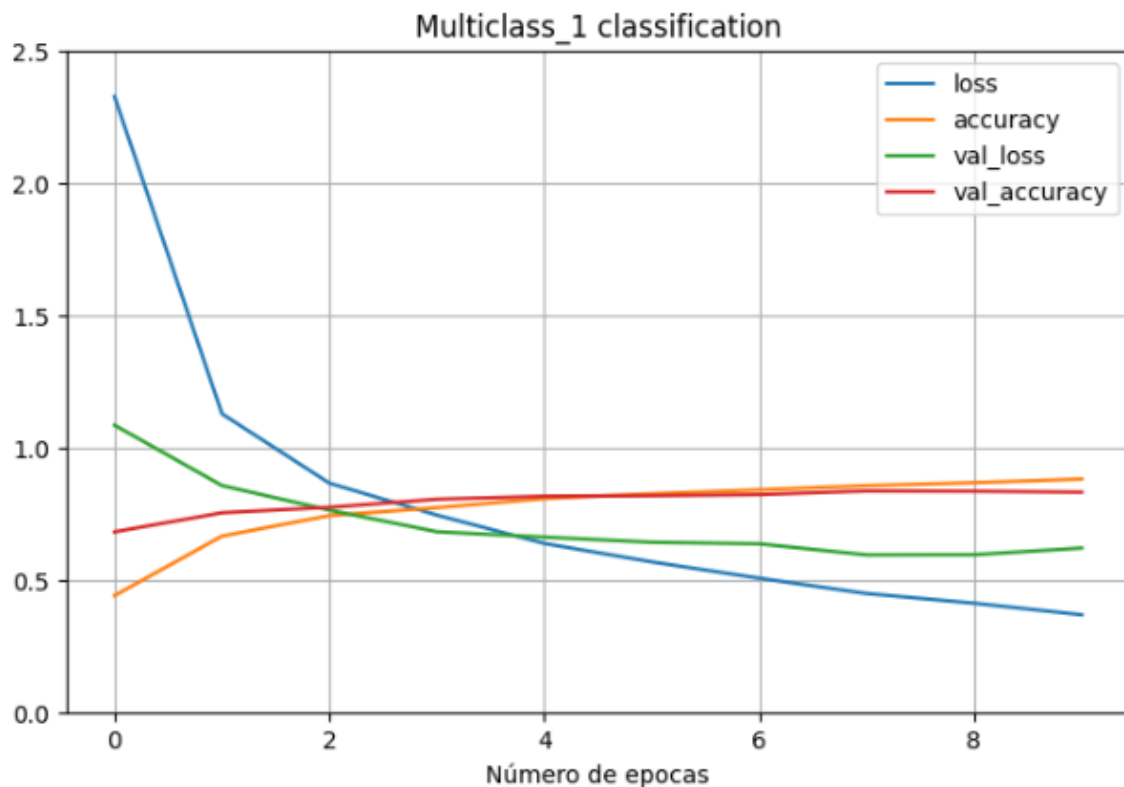


Figura 31. Visualización modelo 1 de clasificación multiclase.

5.2.1.2 Test y predicciones

Para comprobar la efectividad de clasificación de nuestro modelo, se debe cuantificar su capacidad de predicción, mediante el cálculo de las métricas establecidas, que resulta imprescindible para determinar el grado de rendimiento del modelo. Para ello, se utiliza el conjunto de test.

Como siempre, se busca ofrecer un resultado más visual que ayude al completo entendimiento, realizando una implementación de varias funciones que permiten seleccionar una imagen aleatoria de cada una de las clases, o platos de comida, de las que se disponen, y aplicar su clasificación, mostrando por pantalla la imagen seleccionada, atribuyendo un título que cumplirá con las siguientes características: si la imagen ha sido clasificada correctamente, se mostrará el plato de comida con la información de ser correcta en color verde. En contraposición, si la predicción aplicada a la imagen difiere de su verdadera clase, se pasa a mostrar el plato de comida con la información de ser incorrecta en rojo, agregando además el tipo de plato de comida en el que se tendría que haber clasificado.

Un ejemplo de resultado obtenido, mostrando solo las primeras 12 clases del total para simplificar, sería el siguiente:



Figura 32. Predicciones modelo 1 de clasificación multiclase.

Aparte de realizar la clasificación de una imagen aleatoria por cada una de las clases, se debe determinar las métricas para todo el conjunto de prueba, ya que los resultados anteriores pueden depender del azar y no atribuir una realidad empírica a nuestro modelo de clasificación.

Por tanto, se realiza el estudio de una serie de métricas que proporcionan unos valores sobre todo el conjunto de test. Las métricas en las que se realiza un estudio son las siguientes.

Métrica	Valor
Accuracy	0.8388
Recall	0.7895
F-score	0.7872

Tabla 11. Resultados de métricas del modelo 1 de clasificación multiclase

- **Accuracy.** Medida mostrada directamente por el modelo por decisión nuestra, produciendo un valor de 0,8388. Supera con creces el valor de 75% de precisión establecido inicialmente para catalogar el modelo como óptimo para nuestras condiciones.
- **Recall.** Métrica que ha sido calculada mediante la función de *metrics* que, tras recibir un modelo, en este caso el de clasificación multiclase con limitación de clases, y un generador de test, el determinado durante el desarrollo del modelo, permite calcular esta medida, produciendo un valor de 0,7895.
- Por último, se hace un análisis de la métrica **F1-Score**, calculada a su vez mediante la función *metrics*, y devolviendo un valor de 0,7872.

Los resultados se muestran a continuación.

```
Recall: 0.7895
F1-score: 0.7871900343779057
```

Figura 33. Métricas del modelo 1 de clasificación multiclase

El análisis de estas métricas afirma que, aunque el valor de *accuracy* sea ligeramente superior al de las otras dos métricas, *recall* y *F1-score*, todas presentan valores muy similares, por lo que este equilibrio entre estas métricas sugiere que el modelo está realizando predicciones precisas y exhaustivas en general. Es un indicativo de que el modelo está logrando un buen rendimiento en la clasificación de las clases y que no se está enfocando excesivamente en una métrica específica a expensas de las demás.

5.2.2 Un modelo de clasificación multiclase limitando el número de imágenes por plato a clasificar

En el segundo modelo se tomará el enfoque de llevar a cabo una clasificación también multiclase, prediciendo los diferentes platos de comida seleccionados, pero estableciendo una reducción en el número de imágenes que componen cada plato. Con esto, se consigue una nueva alternativa y, por tanto, una nueva forma de evaluar cómo se comportaría más eficaz el modelo, si con una reducción de platos, o con una reducción de imágenes por plato, entendiendo como eficiencia la mayor precisión posible en la clasificación del mayor número de comidas diferentes. Tras deducir esta incógnita, se podrá extrapolar el método más efectivo al modelo de clasificación multilabel de ingredientes.

5.2.2.1 Implementación

Tras varias pruebas para conseguir el mejor resultado de métricas posible dentro del marco temporal óptimo que se había marcado como efectivo y superando la precisión acordada, se ha determinado los siguientes parámetros.

Prueba nº	Número de clases N	Número de imágenes	Accuracy	Tiempo de duración
1	20	700	0.80	33 min
2	30	700	0.7514	52 min
3	32	600	0.7682	57 min

Número de clases seleccionadas: 30

Número de imágenes por clase elegidas: 700

Tabla 12. Parámetros para determinar el modelo 2 de clasificación multiclase

- En la prueba número 1, con un número de clases igual a 20 y un número de imágenes por plato de 700, se observa como el accuracy y el tiempo de duración de la ejecución entera del entrenamiento son buenos, por lo que es un modelo que se puede considerar óptimo.
- En la siguiente prueba, se probó a subir el número de clases a 30, manteniendo el número de imágenes, y se alcanzó un accuracy prácticamente igual al acordado, y un tiempo de duración que se excedía en 7 minutos de lo establecido, pero que se decidió contar como aprobable debido a no ser una diferencia sumamente excesiva para descartarlo, ya que estos minutos de exceso corresponden a la duración de una sola época más, aproximadamente.
- Encontrado el número máximo de clases para el valor de 700 imágenes por plato, se optó subir en clases mientras se bajaba el número de fotografías por comida para comprobar si el modelo sufría una gran degradación o no. Se observa como el accuracy consiguió aumentar en su valor, pero que el tiempo de ejecución se elevó por encima del establecido.

Por tanto, pese a que el tiempo de duración está al límite, se establece como máximo número de clases la cifra de 30 platos diferentes de comida, siendo el número máximo de imágenes por clase de 700 ya que, tanto la subida en las clases seleccionada como una disminución del total de imágenes por plato, provoca un tiempo de ejecución mayor al acordado.

A raíz de esta selección, se pasa a seguir los mismos pasos que en el ejemplo anterior para la implementación de nuestro nuevo modelo, desarrollados a continuación.

- **Se crea un nuevo directorio: *multiclass_2*.** Con la creación de un nuevo directorio aparte del ya existente, lo que se busca es una independencia total del conjunto de datos de ambos modelos, aislándolos mediante la creación de carpetas que permitan una mejor organización de los datos.
- **Se divide el conjunto de datos en train y test.** Al igual que en el método anterior, es necesario esta división para poder comprobar cuál es la verdadera efectividad de las métricas sobre un conjunto de prueba independiente del conjunto de datos de entrenamiento. Además, la misión de detectar el sobreajuste para combatirlo y no presentar un modelo inútil ante datos desconocidos obliga a llevar a cabo una partición total en el conjunto de datos.
- **Se copia cada una de las imágenes** correspondientes a cada uno de estos subdirectorios, teniendo en cuenta que ahora hay un total de *IMAGES_NUMBER* de imágenes por cada plato. Se establece un límite máximo en la variable mencionada anteriormente, que determinará el número total de fotografías que se analizarán por cada tipo diferente de plato. Con esto se busca comprobar si se es capaz de, mediante un menor número de imágenes por clase, conseguir analizar un número mayor de platos de comida sin que el rendimiento se vea muy mermado.
- **Se inicializa la red pre entrenada VGG16.** Como ya se comentó, se vuelve a hacer uso de la red pre entrenada implementada por el grupo de investigadores Visual Geometry Group, debido a su gran eficiencia en el caso anterior, así como para no producir una alteración de los resultados debido al uso de redes completamente distintas que pudieran afectar antes que los parámetros que se ha marcado como factor diferencial para ambos modelos.
- **Se aplica la congelación de capas y el método Fine-tuning.** Las últimas dos capas serán las únicas de la red en la que se permite un entrenamiento, debido a que presentarán las características más concretas que aportarán la información relevante para detectar los diferentes patrones en nuestras imágenes.
- **Se añaden las capas restantes** para que nuestro modelo tenga la misma composición que el método anterior. Se sigue la misma estructura, añadiendo una capa de pooling de *GlobalAverage2D()*, la cual reducirá la dimensionalidad de la imagen para poder aplicar posteriormente capas interconectadas, o densas, hasta llegar a la capa de salida. Estas capas interconectadas contarán con sólo una unidad, de 256 neuronas, debido a su buen resultado obtenido en ejemplos similares, así como su efectividad mostrada en el modelo anterior. Por último, la capa de salida mostrará tantas neuronas como clases seleccionadas se hayan establecido, siendo la función de activación *softmax*, debido a que se busca una clasificación multiclase.
- **Se aplican las técnicas para combatir el sobreajuste.** Se introduce una capa de Drop Out antes de la capa de salida, con un valor de 0.2 recomendable para el uso de CNN, y se implementa una serie de modificaciones aleatorias en nuestras imágenes, técnica denominada como Data Augmentation, que producirá un

conjunto de datos de entrenamiento mayor que favorecerá el aprendizaje de patrones generales frente a la memorización de estos.

- **Se crean los diferentes generadores de imágenes** para los conjuntos de train y validación. El *target_size* escogido será de 240x240, al igual que se hizo en el método anterior, y se establecerá un tamaño de batch de 100. Además, se utiliza el método *sparse* en el parámetro *class_mode*, que significa que las etiquetas de clase se generarán como números enteros en lugar de usar la codificación one-hot. Se muestra a continuación.

```
BS=100
train_generator_mc2 = train_datagen_mc2.flow_from_directory(train_dir,
                                                            target_size=(240,240),
                                                            batch_size=BS,
                                                            class_mode='sparse')
test_generator_mc2 = test_datagen_mc2.flow_from_directory(test_dir,
                                                          target_size=(240,240),
                                                          batch_size=BS,
                                                          class_mode='sparse')

Found 16800 images belonging to 30 classes.
Found 4200 images belonging to 30 classes.
```

Figura 34. Generadores modelo 2 de clasificación multiclase.

- **Se implementa el optimizador Adam**, con un *learning rate* testado de 0,0001. Su gran rendimiento en el anterior modelo hace se decante por este valor de nuevo, y probar si su efectividad se sigue manteniendo en el nuevo modelo propuesto.

Una vez recorridos toda esta serie de pasos, se lleva a cabo el entrenamiento del modelo, cuyo método *fit* quedaría de la siguiente manera.

```
history_multiclass_2 = vgg_model_multiclass_2.fit(
    train_generator_mc2,
    steps_per_epoch=train_generator_mc2.n // BS,
    epochs=15,
    validation_data=test_generator_mc2,
    callbacks=[es_callback],
    validation_steps=test_generator_mc2.n // BS
)
```

Figura 35. Método *fit()* del modelo 2 de clasificación multiclase.

El resultado que produce el entrenamiento mediante la función *fit()* se adjunta a continuación. Al igual que en el anterior método, la composición del resultado es exactamente la misma, variando ligeramente el número total de *step_per_epochs*, debido a que, aunque el número de clases se ha incrementado, el número de fotografías por clase se ha visto reducido, produciendo un volumen total de datos similar al que se

analizó en el modelo anterior. Sin embargo, ahora se determinará el impacto que tiene esa reducción de imágenes frente a mayor número de platos de comida.

```
Epoch 1/15
168/168 [=====] - 286s 2s/step - loss: 3.3321 - accuracy: 0.2452 - val_loss: 1.8681 - val_accuracy: 0.4938
Epoch 2/15
168/168 [=====] - 286s 2s/step - loss: 1.8297 - accuracy: 0.4966 - val_loss: 1.4227 - val_accuracy: 0.6136
Epoch 3/15
168/168 [=====] - 286s 2s/step - loss: 1.4473 - accuracy: 0.5973 - val_loss: 1.2345 - val_accuracy: 0.6676
Epoch 4/15
168/168 [=====] - 282s 2s/step - loss: 1.2213 - accuracy: 0.6562 - val_loss: 1.1283 - val_accuracy: 0.6990
Epoch 5/15
168/168 [=====] - 284s 2s/step - loss: 1.0865 - accuracy: 0.6896 - val_loss: 1.0420 - val_accuracy: 0.7186
Epoch 6/15
168/168 [=====] - 283s 2s/step - loss: 0.9618 - accuracy: 0.7239 - val_loss: 1.0001 - val_accuracy: 0.7307
Epoch 7/15
168/168 [=====] - 286s 2s/step - loss: 0.8854 - accuracy: 0.7461 - val_loss: 0.9866 - val_accuracy: 0.7350
Epoch 8/15
168/168 [=====] - 285s 2s/step - loss: 0.8043 - accuracy: 0.7658 - val_loss: 0.9649 - val_accuracy: 0.7464
Epoch 9/15
168/168 [=====] - 286s 2s/step - loss: 0.7316 - accuracy: 0.7856 - val_loss: 0.9362 - val_accuracy: 0.7514
Epoch 10/15
168/168 [=====] - 288s 2s/step - loss: 0.6588 - accuracy: 0.8005 - val_loss: 0.9611 - val_accuracy: 0.7543
Epoch 11/15
168/168 [=====] - 287s 2s/step - loss: 0.6201 - accuracy: 0.8133 - val_loss: 0.9389 - val_accuracy: 0.7624
```

Figura 36. Entrenamiento modelo 2 de clasificación multiclase.

5.2.2.1.1 Visualizaciones

Se lleva a cabo una representación de los resultados obtenidos, que se mostrarán a partir de gráfica donde se visualizan las funciones de pérdida y las métricas analizadas, en este caso el *accuracy* del modelo.

Comenzando por las funciones de pérdida, se observa cómo la gráfica de *loss* disminuye su valor paulatinamente hasta valores cada vez más bajos, mientras que la función de *val_loss* hace lo propio, pero hasta alcanzar un mínimo absoluto, producida en la época 9, valor 8 de la x. Los dos sucesivos valores mostrados corresponderán a los resultados ofrecidos por las diferentes gráficas hasta que la paciencia establecida se agotase.

Haciendo una descripción seguidamente sobre las gráficas de *accuracy*, se puede presenciar que ambas toman valores muy parecidos durante todo el recorrido posible marcado por las épocas, simplemente diferenciándose al principio, donde para épocas primeras el *val_accuracy* muestra resultados mejores que su métrica respectiva de entrenamiento, pero que, a partir de la época 5 valor de x igual a 6, se comprueba cómo se produce un cambio de tendencia y la gráfica de *accuracy* es la que impone mejorando los resultados frente al conjunto de validación.

Como se puede contemplar, las tendencias obtenidas son similares a las del método anterior, ya que estas tendencias son similares para todos los tipos de problemas que presenten este tipo de características.

El gráfico obtenido es el que se presenta inmediatamente.

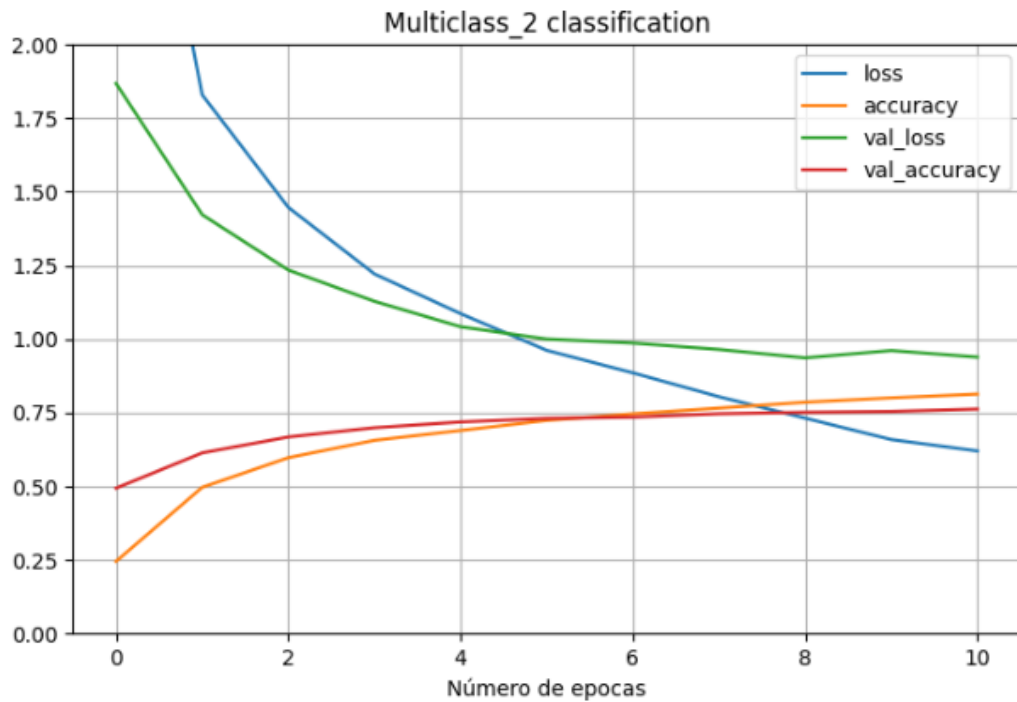


Figura 37. Visualización del modelo 2 de clasificación multiclase.

5.2.2.2 Test y predicciones

Nuevamente, se debe realizar un apartado de predicciones que indiquen si el modelo está realmente clasificando de forma correcta las imágenes de cada clase que recibe como input.

De esta forma, se procede a llevar a cabo la clasificación de una imagen aleatoria de cada una de las clases disponibles en el modelo, donde se muestra de una forma visual si el plato predicho es correcto, marcando de color verde la satisfacción de su clasificación, además de agregándole la etiqueta de *correct*; y se establece un color rojo, en signo de advertencia, si el plato clasificado no coincide con su valor real. Además, de producirse este caso, se añadiría al título la clase correcta que tendría que haber sido clasificada.

Para simplificar, se adjunta en la memoria la predicción de las primeras 12 clases, por tema puramente estético y visual.



Figura 38. Predicciones del modelo 2 de clasificación multiclase.

Aparte de realizar la clasificación de una imagen aleatoria por cada una de las clases, se debe determinar las métricas para todo el conjunto de prueba, ya que los resultados anteriores pueden depender del azar y no atribuir una realidad empírica a nuestro modelo de clasificación

Por tanto, se realiza el estudio de una serie de métricas que proporcionan unos valores sobre todo el conjunto de test. Las métricas en las que se realizan un estudio son las siguientes.

Métrica	Valor
Accuracy	0.7514
Recall	0.7514
F-score	0.7517

Tabla 13. Resultados de las métricas del modelo 2 de clasificación multiclase

- **Accuracy.** Medida mostrada directamente por el modelo por decisión nuestra, pero calculada también para comprobar su veracidad, produce u valor de 0.7514, el cual se marca por encima del valor prefijado como modelo aceptado (75% de precisión).
- **Recall.** Métrica que ha sido calculada mediante la función de *metrics*, y que devuelve un valor de 0.7514, coincidiendo con el valor de la medida de *accuracy*.
- Por último, se hace un análisis de la métrica **F1-Score**, calculada a su vez mediante la función *metrics*, y devolviendo un valor de 0,7517.

Se muestra la salida de nuestra función *metrics* para verificar la obtención de dichas métricas.

Accuracy: 0.7514285714285714
Recall: 0.7514285714285714
F1-score: 0.7517651879733925

Figura 39. Resultados de las métricas del modelo 2 de clasificación multiclase

El análisis de estas métricas deja ver valores muy similares en accuracy, recall y F1-score, indicando que el modelo está obteniendo un rendimiento equilibrado en la predicción de las clases positivas y negativas. Esto significa que el modelo tiene una buena capacidad para identificar correctamente las muestras positivas y negativas, sin sesgos significativos hacia ninguna clase y que el modelo está logrando un buen rendimiento en la clasificación de las clases y que no se está enfocando excesivamente en una métrica específica a expensas de las demás.

5.2.3 Un modelo de clasificación multiclase donde la estructura de la red neuronal convolucional es creada por nuestra parte.

El último modelo presentado de clasificación multiclase será el reflejo de una implementación completa por nuestra parte, en la que se decide diseñar una estructura de CNN adecuada para conseguir valores que iguales o superen al modelo anterior.

5.2.3.1 Implementación

Con este modelo, se pasa a evaluar el reto que suponía crear una red neuronal convolucional autónomamente, para deducir si, bajo una estructura de red definida por nuestra parte, se es capaz de realizar un acercamiento a los rendimientos obtenidos por una red pre entrenada o incluso poder llegar a superarlos. Para ello, el modelo de clasificación seguido es el mismo que en los casos anterior, una clasificación multiclase.

Tras una serie de pruebas para identificar cuáles son los parámetros de número de clases y número de imágenes por clase que hacen que el modelo sea lo más óptimo posible, siempre ciñéndose las condiciones de recursos disponibles, se han obtenido los siguientes resultados.

Número de clases N	Número de imágenes	Rendimiento	Tiempo de duración
20	600	0.2054	+1h
10	700	0.5096	1h

Tabla 14. Elección de los parámetros para seleccionar el modelo 3 de clasificación multiclase

- La primera prueba supuso algo bastante sorprendente ya que se esperaba un rendimiento más bajo, pero no el acontecido. Se propuso mantener el número de imágenes igual que el modelo anterior, establecido hasta ahora como el mejor, pero se

optó por bajar el número de clases a 20 ya que se preveía que los resultados propuestos podrían ser peores. Sin embargo, estos resultados fueron realmente pésimos ya que, tras más de una hora de tiempo de duración, el accuracy máximo marcado sólo pudo alcanzar el valor de 0,2054.

- Para evadir las dudas de si la estructura de la red era tan mala como la comprobada en el apartado anterior, se decidió realizar una bajada brusca de clases para evaluar este suceso, aumentando ligeramente las imágenes. Se estableció un número de clases igual a 10 y el número de imágenes por clase igual a 700, y se obtuvo la conclusión de que los resultados realmente iban en dirección hacia una buena clasificación, pero que el proceso era tan sumamente lento que escapa de los marcos de tiempo establecidos.

Por tanto, se concluye que una red neuronal creada de manera autónoma puede llegar a producir resultados óptimos de clasificación para el ejemplo que se está estudiando, pero que donde reside la diferencia es en la capacidad que presenta una red previamente entrenada en la detección velozmente de las características que proponen una clasificación rápida y eficaz.

Para explicar cómo ha sido la construcción de este modelo creado por nuestra parte, se siguen unas directrices que se comentan a continuación.

- **Se establecen los valores para la división holdout**, que dividirán el volumen total de datos en conjunto de entrenamiento y conjunto de prueba. Los valores que se han atribuido a esta separación han sido un 80% del total para llevar a cabo la parte de entrenamiento, mientras que el 20% restante se ha dedicado para pruebas. El conjunto de train será utilizado para el entrenamiento de nuestro modelo, así como el conjunto de test servirá para la detección del sobreajuste y su parada antes de que este vaya a más.
- **Se realiza el preprocesamiento correspondiente de las imágenes** para incluirlas en los diferentes conjuntos establecidos, aplicando una adaptación de las dimensiones común establecida de 240x240. A su vez, se incluyen como etiquetas la lista de ingredientes correspondiente a cada plato de comida, escogidos del archivo 'ingredients.txt', cuyos elementos son atributos categóricos.
- **Se realiza la implementación propia.** Para ello, se decide simular una estructura parecida a la propuesta por VGG-16, además de otras como Inceptionv3. Esta sería la estructura que se ha seguido.
 - En primer lugar, contextualizar qué tipos existen para la creación de una red neuronal convolucional CNN.
 - **Enfoque secuencial.** El enfoque secuencial es una forma sencilla y lineal de construir modelos en TensorFlow. Se basa en la clase Sequential de TensorFlow, donde las capas se agregan secuencialmente una tras otra. Cada capa se agrega utilizando el método `add()` y se conecta

automáticamente a la capa anterior. Este enfoque es adecuado para modelos de CNN relativamente simples y lineales.

- **Enfoque funcional.** El enfoque funcional es más flexible y permite construir modelos más complejos con múltiples ramificaciones o conexiones no lineales. Se basa en la API funcional de TensorFlow, donde se crea un grafo dirigido de capas conectadas explícitamente. Cada capa se define como una función que toma una o varias capas de entrada y devuelve una capa de salida. Esto permite conexiones más complejas entre capas y la reutilización de capas en diferentes partes del modelo.

El enfoque utilizado será el tipo secuencial.

- Además, antes de empezar con la explicación de la estructura de nuestro modelo, se debe definir cuáles son los parámetros que conformarán cada una de las capas implementadas.

- Para la capa de convolución, se cuentan con los siguientes parámetros.
 - **Filters.** Este parámetro contempla el número de filtros que se utilizarán en la capa convolucional, donde cada filtro representa una matriz de pesos que se convoluciona con la imagen de entrada para extraer características específicas. Cuanto mayor sea el número de filtros, más características diferentes podrá aprender la capa.
 - **Kernel_size.** Especifica el tamaño del kernel (ventana) que se utilizará para la convolución. El kernel se desliza por la imagen de entrada y realiza operaciones de multiplicación y suma para generar un mapa de características. Un tamaño de kernel más grande captura características más grandes, mientras que un tamaño de kernel más pequeño captura características más pequeñas y detalladas.
 - **Input_shape.** Este parámetro se relaciona con la forma de los datos de entrada a la capa convolucional. Debe ser una tupla que indique el tamaño de la imagen de entrada y el número de canales. Por ejemplo, en una imagen en color RGB, el tamaño de la imagen sería (alto, ancho) y el número de canales sería 3.
 - **Padding.** Este parámetro determina si se aplicará relleno a la imagen de entrada antes de la convolución, donde este puede ser "valid" (sin relleno) o "same" (con relleno). En "valid", no se agrega ningún relleno y la salida será más pequeña que la entrada. En "same", se agrega relleno para que la salida tenga el mismo tamaño que la entrada.

- **Activation.** Corresponde con la función de activación que se aplicará a la salida de la capa convolucional. La función de activación introduce no linealidad en la red y permite que la CNN aprenda relaciones más complejas en los datos. Algunas funciones de activación comunes incluyen ReLU, sigmoid y tanh.
 - **Name.** Se aplica al nombre que se le otorgará a la capa convolucional.
- Para la capa de pooling, se establece el siguiente parámetro.
 - **Stride.** Se utiliza en la capa de pooling para especificar el número de píxeles que el filtro de pooling se desplaza horizontal y verticalmente en cada paso durante el muestreo. Con este parámetro, dependiendo de su valor, es como se controla la reducción de la dimensionalidad del mapa de características.
- Una vez explicados todos estos conceptos básicos para el entendimiento de la implementación de una red neuronal convolucional, se pasa a describir la estructura que se ha decidido plantear.
 - La CNN comienza con una capa convolucional, denominada 'conv1', con 64 filtros de tamaño 3x3, cuya función de activación es Relu. Esta capa marca las características dimensionales que deben tener todas las imágenes que vayan a ser clasificadas. Las dimensiones adoptadas han sido de 240x240, por 3 canales debido a ser imágenes a color.
 - Seguidamente, se aplican dos capas convolucionales más con las mismas características de parámetros para conseguir el aprendizaje más complejas, ya que cada capa de convolución aprende diferentes características de la imagen de entrada.
 - Posteriormente, se añade una capa de pooling para reducir la dimensionalidad del mapa de características, estableciendo un stride igual a 2. La capa de pooling utilizada es la de MaxPooling2D.
 - Este patrón, de 3 capas convolucionales idénticas y una capa de pooling que las sucede, será el patrón establecido durante toda la arquitectura de la red, ya que se busca obtener características más complejas a medida que se van creando capas convolucionales, y después se pasa a llevar a cabo su reducción con las capas de pooling.
 - La siguiente secuencia mostrará 3 capas convolucionales, en el que el número de características a extraer, o *filters*, hace que aumente al valor de 256.

- Posteriormente, se vuelve a implementar una capa de pooling del tipo MaxPooling2D, al tratarse de imágenes.
- La última secuencia que se implementa está compuesta por 3 capas convolucionales de 512 *filters* cada una, por lo que se espera de esta manera haber aumentado suficientemente la capacidad de extracción de características en esta última capa, última capa de convolución en la fase convolucional de la red.
- La última capa de pooling también contará con un *stride* igual a 2 y su tipo será MaxPooling2D.
- De esta forma, se termina la fase convolucional, por lo que ahora se necesita transformar mis datos bidimensionales en unidimensionales para poder adaptarlos a las neuronas de salida de la última capa.
- Por tanto, se lleva a cabo la implementación de la capa GlobalAveragePooling2D, la cual realiza la labor anteriormente mencionada, por lo que es perfecta para su uso en este momento de la construcción de la CNN.
- Seguidamente, se monta una capa densa oculta e interconectada, con un número de 256 neuronas, ya que en la red pre entrenada vgg16 funcionó bien. Con una función de activación también de relu, queda establecida la penúltima capa de nuestra CNN, ya que se añadirá antes de implementar la capa de salida una capa de Drop Out para combatir el sobreajuste.
- La última capa montará tantas neuronas como clases totales se esté clasificando en ese momento. La función de activación será softmax, ya que se busca una clasificación de la clase que más probabilidad tiene de ser la correcta.

De esta forma, la red quedaría totalmente implementada y se pasaría a comprobar la eficiencia de la misma. A continuación, se muestra una imagen de su estructura.

```

vgg_model_multiclass_3 = keras.Sequential()

vgg_model_multiclass_3.add(keras.layers.Conv2D(filters = 64, kernel_size = (3,3),
        input_shape = (240,240,3), padding = 'same',
        activation = 'relu', name = 'conv1'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(64, 3, padding = 'same', activation = 'relu', name = 'conv2'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(64, 3, padding = 'same', activation = 'relu', name = 'conv3'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.Conv2D(256, 3, padding = 'same', activation = 'relu', name = 'conv4'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(256, 3, padding = 'same', activation = 'relu', name = 'conv5'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(256, 3, padding = 'same', activation = 'relu', name = 'conv6'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.Conv2D(512, 3, padding = 'same', activation = 'relu', name = 'conv7'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(512, 3, padding = 'same', activation = 'relu', name = 'conv8'))
vgg_model_multiclass_3.add(keras.layers.Conv2D(512, 3, padding = 'same', activation = 'relu', name = 'conv9'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.GlobalAveragePooling2D())

vgg_model_multiclass_3.add(keras.layers.Dense(units=256, activation='relu'))
vgg_model_multiclass_3.add(keras.layers.Dropout(0.2))
vgg_model_multiclass_3.add(keras.layers.Dense(units=len(CLASSES_mc3), activation='softmax'))

```

Figura 40. CNN creada por nuestra parte.

- **Se aplican las técnicas para combatir sobreajuste.** En primer lugar, se añade la capa de Drop Out con un valor de 0.2, recomendado para este tipo de situaciones, antes de la capa de salida, y se lleva a cabo la alteración aleatoria de las imágenes del conjunto de entrenamiento, técnica denominada Data Augmentation, que proporcionará una mayor diversidad de imágenes que propiciará un mejor entrenamiento en la búsqueda de patrones que ayuden a la distinción y detección de los diferentes ingredientes por clase.
- **Se crean los generadores** para los diferentes conjuntos, entrenamiento y pruebas, y se aprovecha su funcionalidad para incluir las modificaciones aportadas por la técnica de Data Augmentation en el conjunto de train, así como el target size que tendrían ambas divisiones.
- **Se implementa**, como en modelos anteriores, el **optimizador Adam** con un *learning rate* de 0.0001, el cual ofrece los mejores resultados según las pruebas que se han ido haciendo.

Una vez seguidos toda esta serie de directrices, se pasa al entrenamiento del modelo para determinar los resultados que este ofrece en los datos, mostrados a continuación.

```

Epoch 1/15
70/70 [=====] - 261s 4s/step - loss: 2.2681 - accuracy: 0.1336 - val_loss: 2.2144 - val_accuracy: 0.1632
Epoch 2/15
70/70 [=====] - 268s 4s/step - loss: 2.1710 - accuracy: 0.1882 - val_loss: 2.1213 - val_accuracy: 0.2088
Epoch 3/15
70/70 [=====] - 269s 4s/step - loss: 2.0597 - accuracy: 0.2402 - val_loss: 2.0148 - val_accuracy: 0.2456
Epoch 4/15
70/70 [=====] - 268s 4s/step - loss: 2.0290 - accuracy: 0.2430 - val_loss: 1.9645 - val_accuracy: 0.2676
Epoch 5/15
70/70 [=====] - 269s 4s/step - loss: 1.9810 - accuracy: 0.2654 - val_loss: 1.9412 - val_accuracy: 0.2904
Epoch 6/15
70/70 [=====] - 271s 4s/step - loss: 1.9317 - accuracy: 0.2866 - val_loss: 1.8650 - val_accuracy: 0.3206
Epoch 7/15
70/70 [=====] - 269s 4s/step - loss: 1.8894 - accuracy: 0.3121 - val_loss: 1.8176 - val_accuracy: 0.3676
Epoch 8/15
70/70 [=====] - 272s 4s/step - loss: 1.8255 - accuracy: 0.3395 - val_loss: 1.7991 - val_accuracy: 0.3713
Epoch 9/15
70/70 [=====] - 275s 4s/step - loss: 1.7841 - accuracy: 0.3589 - val_loss: 1.7160 - val_accuracy: 0.3904
Epoch 10/15
70/70 [=====] - 269s 4s/step - loss: 1.7113 - accuracy: 0.3957 - val_loss: 1.7347 - val_accuracy: 0.4110
Epoch 11/15
70/70 [=====] - 270s 4s/step - loss: 1.6689 - accuracy: 0.4127 - val_loss: 1.6985 - val_accuracy: 0.3853
Epoch 12/15
70/70 [=====] - 269s 4s/step - loss: 1.6648 - accuracy: 0.4141 - val_loss: 1.6633 - val_accuracy: 0.4324
Epoch 13/15
70/70 [=====] - 269s 4s/step - loss: 1.5697 - accuracy: 0.4520 - val_loss: 1.6509 - val_accuracy: 0.4294
Epoch 14/15
70/70 [=====] - 269s 4s/step - loss: 1.5242 - accuracy: 0.4759 - val_loss: 1.5647 - val_accuracy: 0.4654
Epoch 15/15
70/70 [=====] - 269s 4s/step - loss: 1.4692 - accuracy: 0.4938 - val_loss: 1.4634 - val_accuracy: 0.5096

```

Figura 41. Entrenamiento del modelo 3 de clasificación multiclase.

Como se puede apreciar, los valores de las funciones de pérdida disminuyen muy lentamente, y esto provoca que el accuracy aumente de una forma continua pero muy leve. Por tanto, esto indica que el rango de mejora del modelo seguramente sea muy amplio, ya que las funciones de pérdida tras 15 épocas siguen estando en valores superiores a 1, mientras que en la red pre entrenada VGG-16 esto sólo pasaba en apenas las primeras épocas. Por tanto, seguramente de tener más recursos, se podría observar un resultado óptimo de esta implementación

5.2.3.2 Test y predicciones

Aunque se observa claramente el mal funcionamiento del modelo en el tiempo marcado, se refleja a continuación la gráfica de resultados, así como una evaluación de las predicciones.

En la siguiente gráfica se pasa a analizar los resultados del entrenamiento.

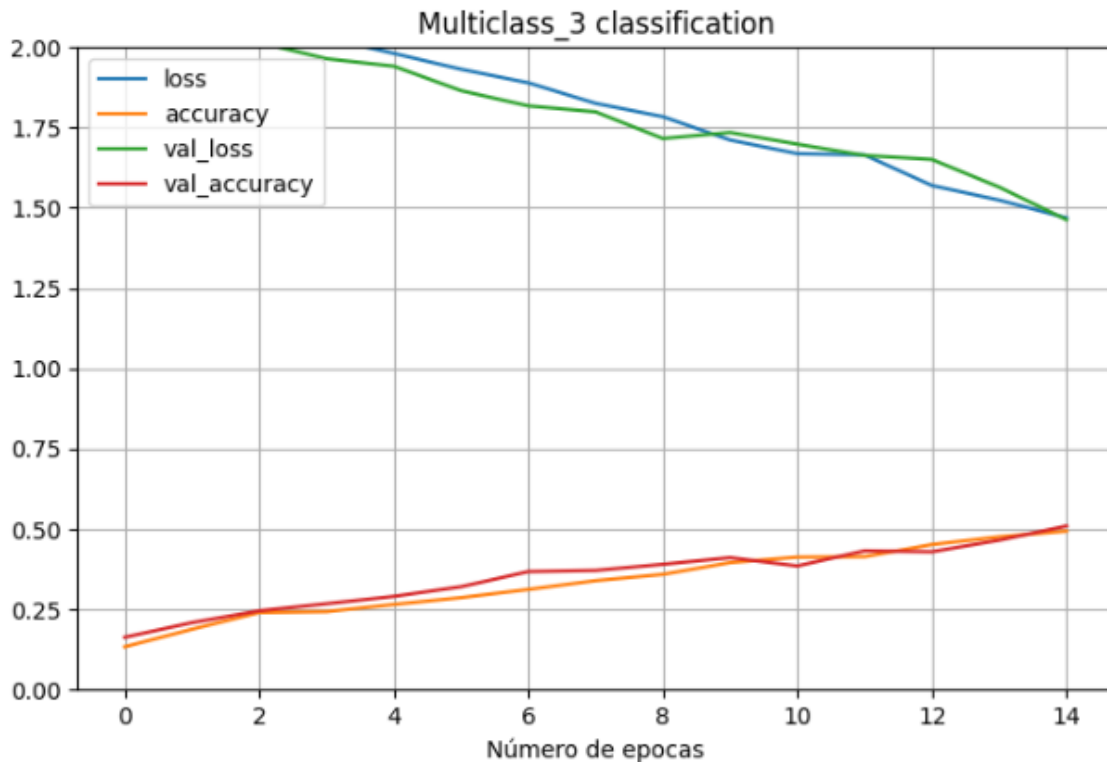


Figura 42. Gráfica de resultados del modelo 3 de clasificación multiclase

Se observa la tendencia anteriormente explicada, donde las gráficas correspondientes a las funciones de pérdida siguen disminuyendo ambas, y los accuracy de los diferentes conjuntos aumentan, de ahí la conclusión de que, en un período más prolongado de tiempo, el resultado de clasificación llegaría a ser aceptable en cuanto a rendimiento.

Se realiza la parte de predicciones de una imagen aleatoria por cada plato de comida, y como se suponía, ante este bajo número de épocas, los resultados son nefastos.



Figura 43. Predicciones del modelo 3 de clasificación multiclase

Las métricas evaluadas de este modelo serían las siguientes. Como ya se preveía, estas métricas reflejan unos resultados ineficientes.

Métrica	Valor
Accuracy	0.5096
Recall	0.4107
F-score	0.3935

Tabla 15. Métricas del 3 modelo de clasificación multiclase

5.2.3.3 Segundo modelo implementado por nuestra parte

Para determinar si el bajo nivel de resultados se debe a una mala confección de la estructura de la CNN, se pasa a proponer un modelo alternativo donde comprobar si estos resultados son producto de un mal diseño de la red o es un ámbito más generalizado.

Para ello, se propone una nueva estructura centrada en la sencillez, para determinar si la complejidad relativa del otro modelo puede verse mejorada con un número mucho menor de capas convolucionales y de pooling. Esta se basa en 3 transiciones de capas convolucionales y capas de pooling, donde cada vez se va aumentando más el número de kernels o características, iniciado en 32 y terminado en 96. Las capas de pooling son del tipo MaxPooling, y la capa que precede a la parte de perceptrón multicapa sería una GlobalAveragePooling. Por último, se añade una capa densa de 256 neuronas y la capa de Drop out, finalizando con la capa de salida con tantas neuronas como clases de salida y con la función de activación softmax.

Así quedaría su estructura.

```
vgg_model_multiclass_3 = keras.Sequential()

vgg_model_multiclass_3.add(keras.layers.Conv2D(filters = 32, kernel_size = (3,3),
        input_shape = (240,240,3), padding = 'same',
        activation = 'relu', name = 'conv1'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.Conv2D(64, 3, padding = 'same', activation = 'relu', name = 'conv2'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.Conv2D(96, 3, padding = 'same', activation = 'relu', name = 'conv3'))

vgg_model_multiclass_3.add(keras.layers.MaxPooling2D(strides = (2,2)))

vgg_model_multiclass_3.add(keras.layers.GlobalAveragePooling2D())

vgg_model_multiclass_3.add(keras.layers.Dense(units=256, activation='relu'))
vgg_model_multiclass_3.add(keras.layers.Dropout(0.2))
vgg_model_multiclass_3.add(keras.layers.Dense(units=len(CLASSES_mc3), activation='softmax'))
```

Figura 44. Estructura de la segunda CNN implementada en clasificación multiclase

Y este ha sido su resultado para los mismos parámetros que la red anterior: 10 clases y 700 imágenes por clase.

```
Epoch 1/15
70/70 [=====] - 91s 1s/step - loss: 2.2979 - accuracy: 0.1125 - val_loss: 2.2910 - val_accuracy: 0.1625
Epoch 2/15
70/70 [=====] - 88s 1s/step - loss: 2.2777 - accuracy: 0.1580 - val_loss: 2.2559 - val_accuracy: 0.1669
Epoch 3/15
70/70 [=====] - 89s 1s/step - loss: 2.2333 - accuracy: 0.1811 - val_loss: 2.2075 - val_accuracy: 0.1897
Epoch 4/15
70/70 [=====] - 89s 1s/step - loss: 2.1945 - accuracy: 0.1954 - val_loss: 2.1899 - val_accuracy: 0.1838
Epoch 5/15
70/70 [=====] - 89s 1s/step - loss: 2.1683 - accuracy: 0.2032 - val_loss: 2.1737 - val_accuracy: 0.2051
Epoch 6/15
70/70 [=====] - 89s 1s/step - loss: 2.1472 - accuracy: 0.2161 - val_loss: 2.1387 - val_accuracy: 0.2294
Epoch 7/15
70/70 [=====] - 89s 1s/step - loss: 2.1184 - accuracy: 0.2459 - val_loss: 2.0986 - val_accuracy: 0.2610
Epoch 8/15
70/70 [=====] - 88s 1s/step - loss: 2.0906 - accuracy: 0.2496 - val_loss: 2.0770 - val_accuracy: 0.2625
Epoch 9/15
70/70 [=====] - 89s 1s/step - loss: 2.0749 - accuracy: 0.2559 - val_loss: 2.0443 - val_accuracy: 0.2809
Epoch 10/15
70/70 [=====] - 88s 1s/step - loss: 2.0538 - accuracy: 0.2646 - val_loss: 2.0355 - val_accuracy: 0.2787
Epoch 11/15
70/70 [=====] - 89s 1s/step - loss: 2.0322 - accuracy: 0.2754 - val_loss: 2.0313 - val_accuracy: 0.2846
Epoch 12/15
70/70 [=====] - 89s 1s/step - loss: 2.0108 - accuracy: 0.2786 - val_loss: 2.0342 - val_accuracy: 0.2801
Epoch 13/15
70/70 [=====] - 89s 1s/step - loss: 1.9946 - accuracy: 0.2946 - val_loss: 2.0329 - val_accuracy: 0.2757
```

Como se puede ver, los resultados son peores que los mostrados con la arquitectura de red implementada anteriormente, por lo que garantiza que se necesita un modelo más complejo de CNN para la correcta clasificación, así como más recursos temporales para su desarrollo.

Las métricas que devolvería el modelo se muestran seguidamente.

Métrica	Valor
Accuracy	0.2757
Recall	0.2503
F-score	0.2540

Tabla 16. Métricas del 4 modelo de clasificación multiclase

5.2.4 Conclusión obtenida

La conclusión que se puede adoptar de estas dos estrategias es que, reduciendo un porcentaje del número total de imágenes por plato, se consigue clasificar un número mayor de clases diferentes que si directamente se aplicara un sesgo en la cifra de comidas a clasificar, suponiendo esto un valor de mejora de hasta un 33%.

Por tanto, se opta por el segundo de los métodos para ser trasladado a la clasificación multilabel de ingredientes, el cual es el fin de este proyecto.

Sin embargo, en cuanto a la conclusión ofrecida en la comparativa entre la diferencia en la efectividad de resultados obtenidos por una red pre entrenada y las dos implementaciones seguidas para intentar hacer frente a estos valores, se muestra el gran mérito de las redes entrenadas y por qué se utilizan con tanta frecuencia hoy en día, ya que su desempeño de

efectividad y tiempo de desarrollo produce que el empleo de este tipo de redes garantice un buen resultado en ejemplos de clasificación en un entorno similar al que fueron empleadas.

A continuación, se muestra una tabla con toda la información obtenida durante este proceso de clasificación.

Ejemplo nº	Tipo	Accuracy	Recall	F1-score
1	Reducción de clases	0.7514	0.7514	0.7517
2	Reducción de clases y de imágenes por clase	0.8388	0.7895	0.7872
3	Red neuronal creada por nuestra parte.	0.5096	0.4107	0.3935
4	Red neuronal creada por nuestra parte, simplificada respecto a la anterior	0.2757	0.2503	0.2540

Tabla 17. Comparativa final de los resultados en clasificación multiclase

5.3 Desarrollo de los diferentes modelos. Clasificación multilabel

Una vez llevado a cabo la clasificación multiclase de los platos de comida, se realiza un cambio en el tipo de clasificación, pasándose a desarrollar el tipo multilabel, aplicándose a la detección de los diferentes ingredientes que componen los platos de comida.

Para ello, se comenzará por llevar a cabo esta clasificación escogiendo como modelo una red pre entrenada, en este caso VGG-16, para después comprobar los resultados frente a una red implementada entera por nuestra parte. De esta forma, se observará cuáles de ellas obtiene unos valores de métricas más altos y si esta diferencia es significativamente grande.

5.3.1 Un modelo de clasificación multilabel de los ingredientes mediante una red neuronal convolucional pre entrenada.

El primer método de este tipo, como se ha comentado en el apartado anterior, propone utilizar como modelo la red previamente entrenada VGG-16, en la que se presupone unos resultados cuya eficiencia pueda estar garantizada.

5.3.1.1 Implementación

Siguiendo una línea similar a la establecida en la clasificación multiclase realizada para los dos modelos anteriores, se pasa a citar toda aquella serie de pasos e indicaciones necesarias para el correcto entrenamiento de nuestro modelo.

5.3.1.1.1 Número de clases seleccionadas

El primer modelo que se opta por desarrollar en este nuevo tipo de clasificación, la clasificación multilabel o multi etiqueta, tomará una serie de parámetros para adaptar su viabilidad de entrenamiento frente al volumen de datos que se disponen y los recursos que se tienen en posesión.

Para ello, gracias a los modelos de clasificación multiclase desarrollados anteriormente, se determina que son dos los parámetros que determinarán la efectividad del modelo. El primero corresponde al número total de platos de comida diferentes que, dentro del marco temporal y computacional disponibles, se puede clasificar, descartando el resto debido a una incompatibilidad de su selección por el límite de tamaño. El segundo parámetro, cuya eficacia se vio demostrada tras la segunda estrategia seguida en el tipo de clasificación anterior, corresponderá con el número total de imágenes por plato de comida, garantizando una reducción del total de fotografías por clase, sin llevar a cabo una decadencia de rendimiento significativa.

Al igual que en el caso anterior, se prefija una serie de umbrales que marcarán cuáles son los criterios para establecer la efectividad de nuestro modelo, acorde a las circunstancias en las que se envuelve.

El modelo es implementado en su totalidad y se pasará a su descripción. Sin embargo, los bajos e irregulares resultados ofrecidos han provocado que la importancia de su desarrollo se haya visto mermada en el trabajo.

5.3.1.1.2 Método holdout

Se ha considerado óptimo una división de 80-20, referida a un 80% del total de datos que pasarán a formar parte del conjunto de train, mientras que el 20% restante se utilizarán para llevar a cabo el conjunto de pruebas. En este caso, a diferencia de los modelos anteriores, no se creará una serie de directorios para guardar las imágenes de cada respectivo conjunto en ellos ya que, en este caso, para la creación de subdirectorios, se tendría que establecer como nombre el conjunto entero de ingredientes por cada plato seleccionado, ya que la técnica de creación de subdirectorios, denominada *sparse*, otorga como clase al nombre raíz de este.

Debido a que esta estrategia carece de sentido en el tipo de clasificación multilabel que ahora se está desarrollando, se adoptará la postura de almacenar en listas los valores tanto de las imágenes como de su clasificación real, los diferentes ingredientes o etiquetas de dicha clase.

Para ello, además de la función de selección de clases para el modelo, se debe implementar una nueva función que asegure la obtención de los ingredientes por plato, siendo estos siempre los mismos para la misma clase seleccionada. La solución que se aporta al respecto es la de, una vez leídos y extraídos los ingredientes del archivo 'ingredients.txt', crear de un diccionario que almacene como clave el plato de comida y atribuirle como valor el conjunto de

ingredientes correspondientes que lo forman. De esta manera, se consigue tantas claves como clases permita la clasificación del modelo, y la lista correspondiente de ingredientes por cada comida en la que se encuentran.

Para el procesamiento de las imágenes, se lleva a cabo una lectura donde se encuentra cada fotografía, realizando un cambio de canal si este fuera necesario al método base RGB, estableciendo un tamaño de imagen de 240 x 240 y transformándolas a un formato matricial de array para su mejor procesamiento posteriormente.

El código implementado para llevar a cabo el método holdout con lo explicado anteriormente se muestra a continuación.

```
CLASSES_m11 = selected_classes(N)
dicc_labels_m11 = selected_labels(CLASSES_m11)

base_dir = './content/Food101/food-101/images'

# TRAIN/VAL/TEST: 80/20%
SPLIT_RATIO_TEST=0.80

train_images = []
train_labels = []

test_images = []
test_labels = []
```

```

for cl in CLASSES_m11:
    # path de las imagenes de la clase cl
    img_path = os.path.join(base_dir, cl)

    # obtenemos la lista de todas las imagenes
    images = glob.glob(img_path + '/*.jpg')

    # TRAIN & TEST (80%-20%)
    # determinamos cuantas imagenes son el 80%
    num_train = int(round(IMAGES_NUMBER*SPLIT_RATIO_TEST))

    # separamos las imagenes en dos listas (train, test)
    train, test = images[:num_train], images[num_train:IMAGES_NUMBER]

    # añadimos las imagenes
    for t in train:
        img = cv.imread(t)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        img = cv.resize(img, (240, 240))
        img = img_to_array(img)

        train_images.append(img)
        train_labels.append(dicc_labels_m11[cl])

    for tst in test:
        img = cv.imread(tst)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        img = cv.resize(img, (240, 240))
        img = img_to_array(img)

        test_images.append(img)
        test_labels.append(dicc_labels_m11[cl])

```

Figura 45. Método holdout modelo 1 de clasificación multilabel.

5.3.1.1.3 Implementación del modelo. Estructura y reducción del overfitting

Seguidamente, tras convertir las imágenes y *labels* en arrays y llevar a cabo una normalización de los valores de los píxeles de las imágenes, se procede a la selección del modelo.

Antes de implementar el modelo, se necesita hacer una transformación de nuestras etiquetas ya que se tratan de elementos categóricos y se deben tratar como conjuntos numéricos. Para ello, se pasará a una contextualización de los elementos que intervienen.

- **One-hot encoding.** Aunque no implicado directamente en este caso, es necesaria su explicación. One-hot encoding es una técnica utilizada para representar variables categóricas en forma de vectores binarios. Consiste en asignar un valor binario de 1 a la categoría correspondiente y 0 a las demás categorías. Por ejemplo, si se tienen tres categorías, se crearían vectores binarios de longitud 3, de modo que cada vector tendría un único valor de 1 en la posición correspondiente a la categoría y 0 en todas las demás posiciones. Esta representación se utiliza comúnmente en problemas de clasificación multiclase, donde se requiere convertir las etiquetas categóricas en una forma numérica adecuada para el entrenamiento de modelos de aprendizaje automático.

- **MultiLabelBinarizer.** Es una técnica utilizada para representar etiquetas multiclase en forma de vectores binarios, utilizada cuando se tiene más de una etiqueta posible para cada instancia de datos. La codificación se realiza asignando un valor binario de 1 a las etiquetas presentes y 0 a las etiquetas ausentes en cada instancia. Sin embargo, la alternativa que propone frente a la codificación one-hot, es que esta representación es útil en problemas de clasificación multilabel, donde cada instancia puede pertenecer a múltiples clases al mismo tiempo.

Una vez explicadas las técnicas de codificación de los valores categóricos en valores numéricos, se centra el caso al modelo. Para ello, se hace uso de la función `MultiLabelBinarizer`, de la librería `scikit-learn`, la cual realizará una transformación one-hot de las etiquetas pero que, debido a estar preparado para clasificación multilabel, salva la diversidad de tamaño que presentan cada una de las etiquetas, siendo este irregular. Es decir, se necesita transformar los ingredientes de cada plato, donde cada comida tendrá un número diferente de ingredientes que componen ese plato, en valores numéricos para que el clasificador los pueda tratar. Además, esto ayudará a saber el conjunto total de etiquetas, o ingredientes, que se está manejando en el modelo, para establecer de esta forma el número total de neuronas en la capa de salida.

Otorgados los valores numéricos a nuestras etiquetas, se pasa a la implementación del modelo. Debido a los buenos resultados en los modelos de clasificación anterior y su familiarización debido al desarrollo de estos, implicando búsqueda de documentación asociada, se decide seleccionar la red pre entrenada VGG-16 nuevamente, descrita en su totalidad en apartados anteriores.

Se aplica la técnica de Fine-tuning también al modelo, debido a las buenas prestaciones que otorgó en los anteriores modelos y a las potenciales ventajas que esta técnica aporta, mediante la detección de las características más relevantes para una más efectiva clasificación de los diferentes ingredientes que componen cada plato de comida. Se seleccionan las últimas dos capas para descongelarlas y extraer estas características.

Complementando la parte multicapa de la red neuronal convolucional, cabe mencionar que se ha seguido una estructura similar a la ya mostrada, mediante una capa de pooling denominada `GlobalAverage2D()`, la cual facilita la transición bidimensional en la que estaba siendo procesada la imagen a una estructura unidimensional, idónea para la capa de salida. A su vez, se añade una capa densa con un total de 256 neuronas, que produzca un efecto de tránsito entre la capa anterior y la capa de salida, esta última formada por la cantidad total de ingredientes anteriormente analizados que componen nuestra clasificación, suponiendo una neurona por ingrediente de nuestro conjunto. La activación de las neuronas de esta última capa es aplicada por la función sigmoide, ya que se necesita una clasificación binaria que identifique si dichos ingredientes se encuentran en nuestra imagen analizada o no. Además, cabe mencionar la inclusión de una capa de Drop Out anterior a la capa de salida para forzar a la red neuronal a un sobre esfuerzo de aprendizaje, función que favorecerá el retraso en la aparición del sobreajuste.

Por tanto, la estructura completa del primer modelo de clasificación multilabel quedaría de la siguiente manera.

```

vgg_multilabel_1 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\_58889256/58889256 [=====] - 0s 0us/step

# We freeze the layers of the pretrained model and apply Fine-tuning.
for layer in vgg_multilabel_1.layers[:-2]:
    layer.trainable = False

# We add to the model a GlobalAveragePooling2D layer and two Denses(128,len(labels_m1_size))
x = GlobalAveragePooling2D()(vgg_multilabel_1.output)
x = Dense(256, activation='relu')(x)
# DROP OUT
x = Dropout(0.2)(x)
output = Dense(labels_m1_size, activation='sigmoid')(x)

vgg_multilabel_model = Model(inputs=vgg_multilabel_1.input, outputs=output)

```

Figura 46. Estructura CNN del modelo 1 de clasificación multilabel.

A su vez, se incluye la otra técnica para combatir el sobreajuste, denominada Data Augmentation, referida a la modificación aleatoria de las imágenes del conjunto de entrenamiento mediante transformaciones indicadas por el desarrollador.

Las alteraciones presentadas serían:

- Alteración en el grado de rotación
- Cambio en el ancho de la imagen
- Cambio en la altura de la imagen
- Aplicación de zoom
- Volteo horizontal

5.3.1.1.4 Generador y optimizador

A continuación, se desarrollan los diferentes generadores que guardarán las imágenes tanto de train como de test. Además, se aprovecha su funcionalidad para incluir los parámetros establecido con la técnica de Data Augmentation y se añade el preprocesado base aplicado por la red VGG-16.

Posteriormente, se pasa a la implementación del optimizador que se utilizará en el modelo. Se selecciona nuevamente el optimizador Adam, adaptando su *learning rate* al valor de 0.0001, y se establece como función de pérdida *binary_crossentropy*, que como ya se pudo ver, se utiliza para la clasificación binaria establecida para nuestras neuronas de salida. Se centra la atención en la medida *accuracy* para que en cada iteración o epoch se vaya calculando, a la par que lo hace con la función de pérdida.

5.3.1.2 Entrenamiento del modelo

Finalmente, tras establecer toda la serie de pasos mencionada anteriormente, se pasa al entrenamiento del modelo, mediante el método *fit()*.

Para ello, es necesario dotarlo de los siguientes parámetros.

- **Generador de train.** Se utiliza la funcionalidad *flow* para atribuir como conjunto *x* las imágenes constituyen el conjunto de entrenamiento, así como las etiquetas codificadas correspondientes. Además, debido al uso de *batch_size*, se añade como tamaño de lote el valor atribuido al tamaño de batch.
- **Steps_per_epochs.** Como en los modelos anteriores, se incluye cuantos son los steps realizados en cada epoch, siendo su valor el tamaño total de mi conjunto de imágenes de train dividido entre en tamaño de batch.
- **Epochs.** Se indica el número de epoch que se quiere establecer en el modelo. En este caso, se establece una cifra de 20 épocas.
- **Callbacks.** Se añade callback para imponer una frenada en el entrenamiento si se localiza el inicio de sobreajuste y en 2 iteraciones de paciencia no ha conseguido solventarse. Además, se recuperan los pesos de la mejor iteración.
- **Generador de test.** Por último, se establece el parámetro de *validation_data*, donde se incluye el conjunto de prueba, siendo la *x* las imágenes correspondientes al conjunto de test y las etiquetas los respectivos valores codificados.

La implementación quedaría de la siguiente forma.

```
history_multilabel = vgg_multilabel_model.fit(  
    train_datagen_mlb1.flow(train_images, train_labels_encoded, batch_size=100),  
    steps_per_epoch=len(train_images) // 100,  
    epochs=20,  
    callbacks=[es_callback],  
    validation_data=test_datagen_mlb1.flow(test_images, test_labels_encoded)  
)
```

Figura 47. Método *fit()* modelo 1 de clasificación multilabel.

Se puede observar cómo tanto la función de pérdida de entrenamiento como la función de pérdida del conjunto de validación parece actuar de forma acorde a las clasificaciones establecidas anteriormente. Sin embargo, el deterioro viene dado por la métrica de accuracy para ambos conjuntos, ya que sus valores excesivamente bajos e irregulares muestran el reflejo de fallos en la implementación.

Tras un análisis del modelo y de toda su constitución, se lleva a cabo la hipótesis de que quizás no se ha conseguido un buen preprocesado de las imágenes a la hora de incluirlas en las respectivas listas, o que no se ha conseguido relacionar de alguna manera

las imágenes con sus etiquetas o *labels* correspondientes. Por eso, tras la realización de múltiples pruebas y búsqueda incesable de los fallos que se habrían producido, se toma la decisión de prescindir de esta clasificación.

De todas formas, se muestra el esfuerzo realizado y los resultados obtenidos.

```
Epoch 1/20
24/24 [=====] - 66s 2s/step - loss: 0.7177 - accuracy: 0.0896 - val_loss: 0.6640 - val_accuracy: 0.0383
Epoch 2/20
24/24 [=====] - 33s 1s/step - loss: 0.6572 - accuracy: 0.1688 - val_loss: 0.6390 - val_accuracy: 0.0017
Epoch 3/20
24/24 [=====] - 34s 1s/step - loss: 0.6469 - accuracy: 0.1117 - val_loss: 0.6366 - val_accuracy: 0.3283
Epoch 4/20
24/24 [=====] - 33s 1s/step - loss: 0.6424 - accuracy: 0.1146 - val_loss: 0.6351 - val_accuracy: 0.1133
Epoch 5/20
24/24 [=====] - 34s 1s/step - loss: 0.6413 - accuracy: 0.1171 - val_loss: 0.6342 - val_accuracy: 0.2633
Epoch 6/20
24/24 [=====] - 34s 1s/step - loss: 0.6399 - accuracy: 0.1058 - val_loss: 0.6334 - val_accuracy: 0.0133
Epoch 7/20
24/24 [=====] - 33s 1s/step - loss: 0.6379 - accuracy: 0.1146 - val_loss: 0.6325 - val_accuracy: 0.2783
Epoch 8/20
24/24 [=====] - 33s 1s/step - loss: 0.6371 - accuracy: 0.1304 - val_loss: 0.6310 - val_accuracy: 0.2200
Epoch 9/20
24/24 [=====] - 33s 1s/step - loss: 0.6353 - accuracy: 0.1092 - val_loss: 0.6311 - val_accuracy: 0.0467
Epoch 10/20
24/24 [=====] - 33s 1s/step - loss: 0.6340 - accuracy: 0.1242 - val_loss: 0.6269 - val_accuracy: 0.0383
Epoch 11/20
24/24 [=====] - 33s 1s/step - loss: 0.6307 - accuracy: 0.1192 - val_loss: 0.6251 - val_accuracy: 0.2133
Epoch 12/20
24/24 [=====] - 34s 1s/step - loss: 0.6292 - accuracy: 0.1021 - val_loss: 0.6239 - val_accuracy: 0.1250
Epoch 13/20
24/24 [=====] - 34s 1s/step - loss: 0.6291 - accuracy: 0.1117 - val_loss: 0.6212 - val_accuracy: 0.1533
Epoch 14/20
24/24 [=====] - 34s 1s/step - loss: 0.6282 - accuracy: 0.1000 - val_loss: 0.6201 - val_accuracy: 0.0400
Epoch 15/20
24/24 [=====] - 33s 1s/step - loss: 0.6287 - accuracy: 0.1067 - val_loss: 0.6232 - val_accuracy: 0.0533
```

Figura 48. Resultados de entrenamiento del modelo de clasificación multilabel

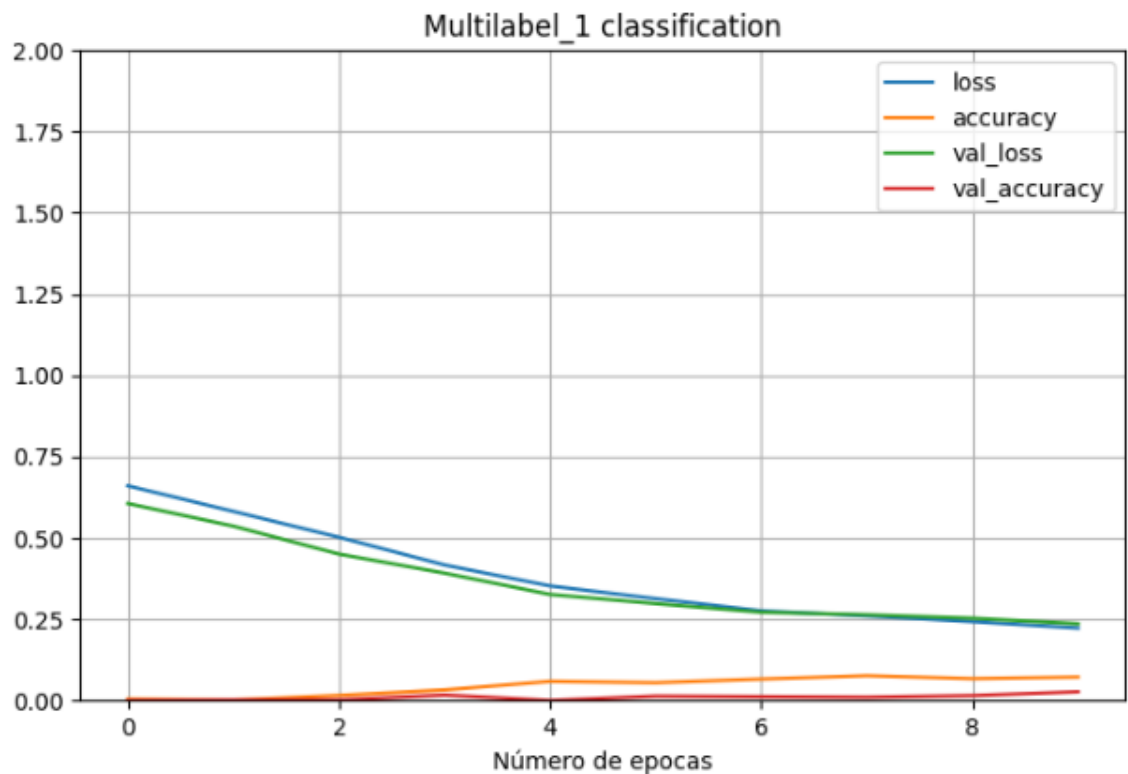


Figura 49. Visualización del modelo de clasificación multilabel.

El gráfico, a su vez, refleja una tendencia esperada en las diferentes funciones de pérdida, pero la deficiencia de la subida de la gráfica de la métrica accuracy provoca que el modelo no pueda ser correctamente evaluado.

5.3.2 Conclusión obtenida

El resultado de la clasificación multilabel ha supuesto un reto, en el que se ha hecho un trabajo y se ha llevado a cabo una documentación preparada para ser capaces de sacar adelante, debido a ser un paso que se había propuesto y que dotaba de un sentido completo al proyecto. Sin embargo, debido a fallos que se desconocen, no ha sido posible la creación de un modelo que presentase resultados óptimos que favoreciesen la idea de realizar una detección de ingredientes. Por tanto, la implementación de una red neuronal convolucional cuya estructura dependiera de nuestra parte carecía de sentido ya que no se conseguiría una mejora de los resultados.

Aun así, se ha decidido incorporar como parte del proyecto ya que, en la mayoría de casos, la parte visible que se expone en un trabajo son el conjunto de éxitos conseguidos en su desarrollo, ocultando o evadiendo los fallos y adversidades que se producen por el camino. Por eso, se ha considerado importante mostrar también la parte que representa las dificultades y obstáculos encontrados, ya que suponen un tiempo de investigación y desarrollo igual o en casos como este hasta superior que partes realizadas con resultados conseguidos.

6 Problemas encontrados

Durante el discurso del proyecto, han sido varios los problemas que han supuesto un peligro real para la realización de este dentro de los marcos temporales establecidos, para alcanzar su grado de aceptabilidad de cara a la fecha final del proyecto.

A continuación, se detallan las principales adversidades que ha ido sufriendo el proyecto a lo largo de su desarrollo, así como la solución que se han propuesto para ir solventando los mismos.

- Se ha contemplado como los **riesgos planteados al inicio** en la identificación de riesgos han salido todos a la luz.
 - **El dataset que se disponía era muy amplio**, por lo que se tuvieron que crear diferentes estrategias para poder hacer frente y llevar a cabo un análisis de los datos con la mejor precisión y alcance posible.
 - **La incertidumbre de los parámetros de elección** a lo largo de los diferentes modelos de clasificación ha supuesto una incógnita a desvelar mediante pruebas y tiempos largos de ejecución, ya que cada modelo probado marcaba un tiempo de entre 45/50 minutos.
 - **La creación de la clasificación multilabel**. Sin duda, uno de los problemas más densos del proyecto, ya que la documentación que se ha tenido que desarrollar acerca de este tipo de clasificación, así como los limitados ejemplos similares que se ha podido encontrar, ha supuesto que el proyecto se viese envuelto en la decisión de prescindir de esta parte, ya que su implementación no estaba garantizada para poder ser incluida dentro de la fecha fin del proyecto. Finalmente, se ha optado por su inclusión debido al alto porcentaje de tiempo empleado en este desarrollo.
 - **Incumplimiento de los tiempos marcados**. Se han tenido que hacer modificaciones con respecto a la planificación inicial establecida, aumentando o disminuyendo las fases futuras que sufrían esta modificación.

Sin embargo, gracias al plan de contingencia marcado inicialmente, se ha sabido sobreponerse a estas adversidades de una forma más eficiente que aquella que se hubiese tenido que emplear de no haber hecho un estudio previo de los posibles riesgos.

- **Fuera de los riesgos que a priori se pudieron establecer**, se ha sufrido la aparición de otro tipo de problemas inesperados.
 - **La imposibilidad de subir el dataset** si no era por medio de la herramienta de Google Drive. La subida de un elemento tan masivo como el dataset, incluso en estado comprimido, suponía una barrera para relacionarlo con Google Colab, debido a que imposibilitaba su subida debido a exceder el tiempo de *upload*. Por tanto, para poder gestionar el espacio disponible en Google Drive para

posibilitar el almacenamiento del dataset, se tuvo que hacer una gran limpieza de archivos y directorios para su cabida.

- **La poca disponibilidad de acceso a las GPUs** por parte de Google Colab en el último mes. Desconociendo exactamente la razón, se ha notado un gran deterioro en la disponibilidad que la herramienta de Google ha brindado para poder acceder a sus recursos de computación en el marco temporal del mes de junio. Finalmente, se tuvo que llevar a cabo la suscripción Pro para poder hacer frente a este desafío.
- **Problemas al emparejar las imágenes con las clases correspondientes** en la clasificación multiclase. Debido a que se almacenaban las imágenes en subdirectorios, donde el nombre de la carpeta servía como nombre de la clase al hacer uso del método *sparse*, se ha contemplado el error de que, el orden en el que se creaban los directorios no era el mismo que luego se reflejaba en la posición de las carpetas, siendo este último ordenado alfabéticamente. Por tanto, las predicciones mostraban un error que realmente no existía no por culpa del modelo, sino de la relación que existía entre datos y clases.
- **Cálculo de las métricas.** También se ha sufrido en la propia piel cómo los `test_generator` creados para los conjuntos de pruebas eran ineficientes a la hora de calcular las predicciones de todo el modelo, ya que al tomarse muestras aleatorias por tamaño de batch, no conseguía relacionar las predicciones con las clasificaciones reales. Para solventar este error, se tuvo que recorrer manualmente el generador para conseguir el orden apropiado para realizar la correcta relación entre predicciones y valores reales que atribuirían el resultado de las métricas del modelo.

7 Conclusiones

Varias son las conclusiones finales y globales que se pueden obtener a raíz de la completa elaboración de este proyecto.

La primera, indica que se ha sabido sobreponerse al conjunto excesivamente grande de datos que comprendía el trabajo, marcando una serie de pautas, como los diferentes modelos aportados y las diferentes estrategias utilizadas en cada uno, que han supuesto un camino hacia el éxito en este apartado. Además, se ha hecho la comprobación de la capacidad de efectividad que muestra una red neuronal entrenada con un número muy alto de imágenes con respecto a las redes que por nuestra parte se han propuesto para intentar hacer frente al excelente rendimiento que, en este caso, VGG-16 ha presentado.

La segunda, se refiere a la parte menos visible del proyecto, la clasificación multilabel de los ingredientes, en la que el esfuerzo empleado para alcanzar la satisfacción de un grado de rendimiento en el proyecto aceptable ha sido frustrado por un fallo indetectable que ha ocasionado girar el foco y la atención a la primera parte de la clasificación.

Por último, como conclusión global, se expone que ha sido un proyecto que, a nivel personal, ha supuesto un reto con el que no se había tenido todavía la oportunidad de realizar un enfrentamiento ante este, y aunque el fin último no ha sido el que se estableció en un principio, todo el camino hacia dicho fin ha sido marcado por un aprendizaje continuo que ha despertado el interés por seguir trabajando en aspectos relacionados con esta temática. Además, ha constituido una forma de enfocar los obstáculos atravesados durante esta travesía para que, aunque la meta no haya podido ser alcanzada al nivel que se hubiese querido llegar, el aprendizaje y esfuerzo durante el tránsito provocan una satisfacción personal recompensada.

8 Proyectos futuros

Tras el desarrollo completo del proyecto, varios han sido los atenuantes que han incitado a proponer nuevas propuestas y nuevos planes futuros con los que poder llevar el reconocimiento de ingredientes en un plato de comida a su máxima expresión.

Debido a que la intención fundamental en este proyecto se convirtió en brindar una ayuda para todo ese tipo de personas que sufrieran de algún trastorno alimenticio, personas cuya salud se ve realmente afectada por situaciones del día a día donde pedir un simple plato de comida en un restaurante pueda suponer un desafío para ellas, se realizó una inmersión en la concienciación que se tiene que tener ante este tipo de ciudadanía, ya que resultan un conjunto de población mucho más grande que el que se piensa.

Además, la vivencia personal vivida en el extranjero, donde no se está acostumbrado a los diferentes platos de comida que sirven en esa región, supone una dificultad extra para este tipo de personas que deben tener una especial atención en todos los ingredientes que puede contener su plato, para no sufrir un susto que podría acarrearles, en casos más extremos, la hospitalización. Y todo esto podría verse enormemente reducido mediante el reconocimiento adecuado de los ingredientes.

Obviamente no es tarea sencilla poder llegar a desarrollar un clasificador que pueda determinar los ingredientes de un plato de comida, ya que un plato está compuesto por un número generalmente elevado de alimentos, y lo que es aún más difícil, es normal que algunos de ellos no sean apreciables a simple vista o estén escondidos bajos otro tipo de alimentos que componen el plato o bajo alguna de las salsas que complementan la comida.

Aun así, la ambición que se presenta radica en una posibilidad de poder llegar a un acercamiento de la motivante idea de conseguir, mediante los planes de futuro que ahora pasará a desarrollar, una detección eficaz de un grupo de ingredientes reducido, lo que producen con más frecuencia algún tipo de intolerancia en la ciudadanía, y centrar la atención en su detección en una serie de comidas, quizás reducidas también o regionales.

Para ello, se proponen una serie de mejoras a implementar en mi proyecto para conseguir alcanzar estas metas que se tienen marcadas.

- **Aumento de los recursos.** Todo esto queda en vano si no se consigue un mayor equipo que proporcione el número suficiente de recursos para poder dar un paso al frente e ir más allá de una simple clasificación multilabel de ingredientes. Para ello, hay múltiples ideas que se pueden seguir para lograr este primer punto
 - **Utilizar servicios de computación en la nube.** Existe la posibilidad de aprovechar los servicios de plataformas en la nube como Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform. Estas plataformas ofrecen una amplia gama de servicios escalables, como instancias de cómputo, almacenamiento y bases de datos, adaptados según tus necesidades. Esto permitirá acceder a recursos de alta capacidad y ajustarlos fácilmente para satisfacer los requisitos del proyecto.
 - **Utilizar servicios de computación distribuida.** Debido a que el proyecto implicará tareas intensivas en computación, como entrenamiento de modelos de aprendizaje automático a gran escala o simulaciones complejas, una opción perfecta es utilizar servicios de computación distribuida. Ejemplos populares

incluyen Apache Hadoop y Apache Spark, que permiten distribuir el procesamiento de datos y aprovechar clústeres de computadoras para acelerar las tareas.

- **Configurar clústeres de computadoras locales.** Otra opción sería, de disponer de acceso a múltiples computadoras, configurar un clúster local para distribuir la carga de trabajo. Esto implica interconectar las computadoras y utilizar herramientas de administración de clústeres para dividir las tareas entre los nodos disponibles. Al compartir la carga de trabajo, obtener un mayor rendimiento y recursos de computación adicionales se convierte en una tarea más simple.
- **Utilizar hardware especializado.** Otra opción, quizás algo más limitada, sería la de pasarse al plan más avanzado de Google Colab, para obtener toda la capacidad de recursos de GPUs y TPUs, así como de memoria RAM disponible. Una opción alternativa sería adquirir ese material comprándolo directamente en una tienda, creando un dispositivo a las necesidades que el proyecto.
- **Aumento en el volumen de datos.** Barajando las opciones que supondría crear un clasificador que pudiera clasificar los ingredientes de multitud de comidas que pudieran encontrar sustancias alérgicas, se es consciente del increíble volumen de datos que esto supondría, ya que para la propia clasificación necesitaría un número elevado de imágenes, para conseguir detectar con la mayor precisión posible los ingredientes, y a parte del número, necesitaría contar con una gran variedad de platos para poder ayudar en un sentido real a una persona que fuese a un restaurante y tuviese dudas de la composición de su comida. Para ello, también se proponen una serie de alternativas.
 - **Varios datasets provinciales.** La idea sería segmentar al máximo posible el volumen de datos, y esto se lograría reduciendo el número de platos diferentes a la gastronomía más demandada por cada región. De esta forma, se tendrían varios datasets, uno por provincia, y un clasificador para cada uno de ellos, por lo que podría aliviar la computación de datos.
 - **Datasets a nivel de comunidad autónoma.** Debido a lo tedioso que supondría llevar a cabo tantos dataset y tantos clasificadores como provincias haya, interesaría dar un salto hacia la gastronomía típica de cada comunidad autónoma, donde el conjunto de datos aumentaría, y se debería contar con un equipo especializado para ello.
 - **Un dataset por país.** De ser una opción viable, cosa que realmente no se cree que pudiera estar en nuestra mano, la gran idea de nuestro proyecto sería poder predecir los ingredientes que se encuentran en las comidas más típicas de todo un país entero, en este caso España. Con ello, se podría ampliar la visión a nivel estatal y soñar con una posible internacionalización del proyecto con datasets que contemplasen la gastronomía de otros países.

Con todo esto, ya se podría dar forma a un nuevo proyecto futuro. Sin embargo, se necesita que el usuario pueda interaccionar con el proyecto para adaptarlo a sus necesidades, tanto alérgicas como gastronómicas, por lo que se propondría una app móvil para facilitar esta labor.

Esta aplicación consistiría en una herramienta donde se pudiera realizar una fotografía con un dispositivo móvil del plato de comida y obtener una clasificación y detección de sus ingredientes en un tiempo razonable para poder ser realmente una herramienta útil. El usuario podría interactuar libremente con ella y configurarla acorde a sus requisitos personales.

La aplicación funcionaría de forma que el usuario tomase una fotografía de la comida que en ese momento va a degustar. La imagen sería procesada de la forma apropiada al input que tendría marcados en nuestro programa para poder realizar su lectura. Una vez procesada, se llevaría a cabo una clasificación y predicción con el modelo correspondiente que el usuario hubiera marcado como seleccionado.

A partir de ahí, el modelo realizaría la predicción del plato de comida, añadiendo una posible advertencia si no detecta los ingredientes establecidos por el individuo como alérgicos y que comúnmente se encontrasen en ese plato. Una vez terminado todo este proceso, se mostraría, mediante una interfaz agradable, los diferentes ingredientes que se habrían detectado.

Estos serían los principales puntos con los que se dotaría a la aplicación.

- **Capacidad de selección del dataset.** El usuario podría elegir la provincia/comunidad/país que decidiese para adaptar la gastronomía de cada una de estas regiones a sus intereses, para poder facilitar una variedad de platos más común en su dieta diaria.
- **Capacidad de selección de los ingredientes.** El usuario debería poder seleccionar aquellos ingredientes que le provocan una intolerancia, para así poder identificar los platos más comunes donde estos se encuentran y prestar una especial atención en los mismos, sin descartar su existencia en otro tipo de comidas donde se podría ubicar también.
- **Capacidad de interactuar con la cámara.** Por supuesto, habría que desarrollar los pluggins necesarios para poder gestionar los permisos de acceso a la cámara de forma que el usuario pudiera realizar una instantánea de su comida y recibir la composición de ingredientes en un corto tiempo de plazo.
- **La posibilidad de recuperar la información** de platos previamente clasificados, para que el usuario pudiese volver a acceder a esta información sin necesidad de volver a crear otra fotografía.

Con todo este proceso queda reflejado la humilde intención de proporcionar un pensamiento más allá del trabajo desarrollado, para que este proyecto no haya quedado en la formación como un mero trabajo más a realizar, y poder dotarlo de una nueva dimensión que haya producido pensar en todas las vías posibles que podrían acompañar a proyecto en un producto que podría ser lanzado al mercado, y del que pudieran nutrirse el mayor número de personas posibles para mejorar y garantizar sus estados de salud.

9. Bibliografía

- Amazon. (s.f.). *Amazon Web Services AWS*. Obtenido de <https://aws.amazon.com/es/what-is/distributed-computing/>
- Arce, J. I. (26 de Julio de 2019). *Health Big Data*. Obtenido de <https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/>
- Arrabales, R. (2016). *Xataka*. Obtenido de <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial>
- Avinash. (2019). *Kaggle - Multiclass Food Classification using TensorFlow*. Obtenido de <https://www.kaggle.com/code/theimgclist/multiclass-food-classification-using-tensorflow>
- Awan, A. A. (2022). *Datacamp*. Obtenido de <https://www.datacamp.com/tutorial/complete-guide-data-augmentation>
- Bengi, C. (s.f.). *Hash Dork*. Obtenido de <https://hashdork.com/es/gpu-vs-tpu/>
- Bolaños, M. (2017). *COMPUTER VISION AND MACHINE LEARNING AT THE UNIVERSITY OF BARCELONA (CVMLUB)*. Obtenido de Dataset Recipes5k: <http://www.ub.edu/cvub/recipes5k/>
- Bolaños, M. (2017). *COMPUTER VISION AND MACHINE LEARNING AT THE UNIVERSITY OF BARCELONA (CVMLUB)*. Obtenido de <http://www.ub.edu/cvub/ingredients101/>
- Brownlee, J. (2018). *Machine Learning Mastery*. Obtenido de <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>
- Calvo, D. (8 de Diciembre de 2018). *Perceptrón Multicapa – Red Neuronal*. Obtenido de <https://www.diegocalvo.es/perceptron-multicapa/>
- Chaos, I. (s.f.). *Interactive chaos*. Obtenido de <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam>
- Chaos, I. (s.f.). *Interactive Chaos*. Obtenido de <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/metodo-de-retencion>
- Cloud, G. (s.f.). *¿Qué es la inteligencia artificial o IA?* Obtenido de <https://cloud.google.com/learn/what-is-artificial-intelligence?hl=es-419>
- Coding, R. K. (2023). *Keep Coding*. Obtenido de <https://keepcoding.io/blog/transfer-learning-fine-tuning/>
- Coding, R. K. (13 de Enero de 2023). *Keep Coding*. Obtenido de <https://keepcoding.io/blog/tipos-capas-red-neuronal-convolucional/>
- Daniel. (s.f.). *DataScientest*. Obtenido de <https://datascientest.com/es/vgg-que-es-este-modelo-daniel-te-lo-cuenta-todo>

Editions ENI. (s.f.). Obtenido de <https://www.ediciones-eni.com/open/mediabook.aspx?idR=8dd2ca32769cb24b49648b15ef8e777e>

El libro de python. (2023). Obtenido de <https://ellibrodepython.com/yield-python>

González, M. L. (2015). *PLAN DE ATENCIÓN A LAS ALERGIAS E INTOLERANCIAS ALIMENTARIAS EN CANTABRIA*. Cantabria: Gobierno de Cantabria. Obtenido de http://saludcantabria.es/uploads/pdf/ciudadania/xPAAI_ALIMENTARIAS.pdf

Google. (s.f.). *Google Colab*. Obtenido de <https://colab.research.google.com/?hl=es>

IBM. (s.f.). *¿Qué son las redes neuronales?* Obtenido de <https://www.ibm.com/es-es/topics/neural-networks>

IBM. (13 de 12 de 2023). *IBM*. Obtenido de <https://www.ibm.com/docs/es/was-zos/9.0.5?topic=servers-introduction-clusters>

IBM. (s.f.). *IBM*. Obtenido de <https://www.ibm.com/es-es/topics/cloud-computing>

IBM. (s.f.). *Redes Neuronales Convolucionales*. Obtenido de <https://www.ibm.com/es-es/topics/convolutional-neural-networks>

Indeed. (2023). Obtenido de <https://es.indeed.com/>

Isma. (2023). *Kaggle - Model new*. Obtenido de <https://www.kaggle.com/code/ismasafitri1312/model-new>

Keras. (s.f.). *Keras API References*. Obtenido de <https://keras.io/api/>

Lerch, D. (2018). *Deep Learning con redes pre-entrenadas en ImageNet*. Obtenido de <https://medium.com/neuron4/redes-pre-entrenadas-en-imagenet-30d858c37b1f>

Matplotlib. (s.f.). *Matplotlib Pyplot*. Obtenido de https://www.w3schools.com/python/matplotlib_pyplot.asp

ME(X), D. (2019). *Kaggle - Brain Tumor Detection v1.0 || CNN, VGG-16*. Obtenido de <https://www.kaggle.com/code/ruslankl/brain-tumor-detection-v1-0-cnn-vgg-16>

MÉDICA, R. (2019). *Redacción médica*. Obtenido de <https://www.redaccionmedica.com/autonomias/aragon/una-de-cada-cuatro-personas-sufre-algun-tipo-de-intolerancia-alimentaria-7046>

Mehrotra, R. (2018). *Flower Recognition CNN Keras*. Obtenido de <https://www.kaggle.com/code/rajmehra03/flower-recognition-cnn-keras>

Numpy. (s.f.). *Numpy Documentation*. Obtenido de <https://numpy.org/doc/stable/index.html>

OpenCv. (s.f.). *Introducing OpenCV University*. Obtenido de <https://opencv.org/>

Os. (s.f.). *Interfaces misceláneas del sistema operativo*. Obtenido de <https://docs.python.org/es/3.10/library/os.html>

Pandas. (s.f.). *Pandas documentation*. Obtenido de <https://pandas.pydata.org/docs/>

Scikit-learn. (s.f.). *Scikit-learn. Machine Learning*. Obtenido de <https://scikit-learn.org/stable/>

TensorFlow. (s.f.). *Crea modelos de aprendizaje automático de nivel de producción con TensorFlow*. Obtenido de <https://www.tensorflow.org/?hl=es-419>

Velasco, L. (Abril de 2020). *Velascoluis*. Obtenido de <https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5>