



Universidad de Valladolid

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Segmentación de imágenes urbanas de un dron mediante Deep Learning

Alumno: Carlos Martín Sanz

Tutor: Teodoro Calonge Cano

Agradecimientos

En primer lugar, me gustaría dar las gracias a Teodoro Calonge por tutorizar y guiarme en este trabajo fin de grado y también al resto de profesores que han participado en mi formación académica durante esta etapa de mi vida.

A mis compañeros y amigos, Alejandro Pulido y Héctor Toribio, me gustaría expresar mi profundo agradecimiento por su compañía y colaboración desde el inicio de este camino, así como por su constante apoyo durante la elaboración de este trabajo.

Gracias a mi pareja, Paula, y mis amigos, por haber sido un apoyo incondicional en los momentos más difíciles; sin ellos, probablemente no habría llegado hasta aquí.

Por último, no tengo palabras suficientes para expresar mi gratitud a mi familia, por haber confiado en mí y haberme apoyado en los momentos más difíciles. Quería hacer una mención especial a mi hermana, Natalia, capaz de hacerme sacar siempre una sonrisa y distraerme en situaciones complicadas.

Resumen

El campo de la Inteligencia Artificial está adquiriendo una relevancia considerable en diversos ámbitos y sectores de la sociedad. Uno de ellos donde la Inteligencia Artificial ha destacado significativamente, es en la clasificación de imágenes y detección de objetos dentro de las mismas. Gracias a estos avances se ha conseguido desarrollar aplicaciones tan dispares como: conducción autónoma, diagnóstico médico o automatización industrial.

Este trabajo explora el área de la Visión por Computador para la detección de zonas de aterrizaje seguras para drones empleando, para ello, la segmentación de imágenes urbanas. Uno de los propósitos de este trabajo es presentar, tanto el marco teórico, como práctico, para desarrollar este modelo de segmentación de imágenes. Para este fin, se emplean técnicas de Deep Learning (Aprendizaje Profundo), en particular, Redes Neuronales Convolucionales, bajo la arquitectura conocida como UNET. A pesar de que los resultados obtenidos alcanzan una alta precisión, considero adecuado mencionar, que no se puede asumir como una solución única y definitiva.

Finalmente, el trabajo se complementa con una aplicación web, que permite utilizar el sistema de detección de zonas de aterrizaje de forma eficiente y sencilla. Con ello, una persona puede supervisar y analizar los resultados obtenidos por la aplicación y proporcionar una cierta explicación de los mismos.

Abstract

Today, Artificial Intelligence (AI) applications are growing up day by day reaching almost everything in our life. In particular, it becomes more popular in Artificial Vision problems as image classification or semantic segmentation, where the Deep Learning has been widely used. Consequently, several intelligence tasks as autonomous driving, medical diagnosis or industrial automation have experimented a very important successes.

This work is focused on drone safe landing zones detection in a satellite (or done) photo. It can be considered as one more image segmentation example, if the corresponding ground truth is available as it taken place in practice. To address that problem, a certain Artificial Vision algorithm will be used. More precisely, a Convolutional Neural Network under a UNET architecture has been selected.

First of all, a theoretical study for understanding that AI model is introduced. After that, an experimental framework will be chosen, explained and implemented. The next step will show the results obtained. In spite they reach a high accuracy level, it doesn't imply that system proposed will be considered as definitive, since another models should be experimented to discard them or to taken into consideration. Unfortunately, that task can not be affordable in this work.

Finally, an application web based has been made to manage the AI model trained. Once the photo was entered, this program outputs the drone safety landing zones estimated by the system, as well as four more kinds of zones depicted in this text.

As conclusion, this project has showed one more time again, that UNET architecture could be used as the basic algorithm to image semantic segmentation offering a very good results.

Índice general

Agradecimientos	3
Resumen	5
Abstract	7
Lista de figuras	13
Lista de tablas	17
1. Introducción	1
1.1. Contexto actual	1
1.2. Origen del proyecto	1
1.3. Motivación	1
1.4. Objetivos del proyecto	2
1.5. Estructura	3
2. Planificación del proyecto	5
2.1. Metodología de trabajo	5
2.2. Entregables	7
2.3. Planificación inicial	8
2.4. Variaciones sobre la planificación inicial	11
2.5. Plan de riesgos	11
2.6. Plan de costes	16
2.6.1. Coste hardware	16
2.6.2. Coste software	16
2.6.3. Coste de recursos humanos	16
2.6.4. Coste total	17
3. Marco Teórico	19
3.1. Introducción Redes Neuronales	19
3.2. Redes Neuronales Artificiales	19
3.2.1. Modelo de McCulloch y Pitts	19
3.2.2. Aprendizaje Profundo	21
3.3. Redes Neuronales Convolucionales	22

3.3.1.	Visión artificial	23
3.3.2.	Convolución	23
3.3.3.	Desplazamiento del kernel	28
3.3.4.	Downsampling	30
3.3.5.	Upsampling	32
3.4.	Arquitectura de una UNET	37
3.4.1.	Propuesta original	37
3.5.	Otros conceptos importantes	39
3.5.1.	Función de activación	40
3.5.2.	Función de pérdida	43
3.5.3.	Optimizadores	44
3.5.4.	Inicialización de pesos	47
3.5.5.	Técnicas de regularización	48
3.6.	Problemas en el entrenamiento	50
3.6.1.	Evanescencia del gradiente	50
3.6.2.	Sobreajuste (Overfitting)	52
3.6.3.	Infrajuste (Underfitting)	52
4.	Conjunto de datos	55
4.1.	Descripción del conjunto	55
4.1.1.	Origen	55
4.1.2.	Formato	55
4.1.3.	Etiquetado y distribución	56
4.2.	Tratamiento sobre el conjunto	57
4.2.1.	Preprocesamiento	58
4.2.2.	Data Augmentation	60
4.3.	Carga de datos	61
5.	Desarrollo completo del modelo	65
5.1.	Construcción	65
5.1.1.	Downsampling - Fase de extracción	65
5.1.2.	Upsampling - Fase de expansión	66
5.1.3.	Arquitectura de la red	67
5.1.4.	Implementación de la red	69
5.2.	Entrenamiento	71
5.2.1.	<code>torch.nn.init.kaiming_uniform_</code>	72
5.2.2.	<code>torch.nn.CrossEntropyLoss</code>	72
5.2.3.	<code>torch.optim.Adam</code>	72
5.2.4.	<code>torch.optim.lr_scheduler.ReduceLROnPlateau</code>	72
5.3.	Bucle de entrenamiento	73
6.	Implementación y herramientas	79

6.1. Tecnologías utilizadas	79
6.1.1. Python	79
6.1.2. Anaconda	80
6.1.3. Jupyter Notebook	80
6.1.4. PyTorch	80
6.1.5. PyTorch vs Keras vs TensorFlow	80
6.1.6. Overleaf	81
6.1.7. Flask	81
6.1.8. GitHub	82
6.1.9. GanttProject	82
6.1.10. Astah Professional	82
7. Resultados	83
7.1. Comparativa fase de entrenamiento	83
7.2. Evolución del modelo	84
7.3. Evaluación	86
7.3.1. Evaluación del caso binario	86
7.4. Interpretación de los resultados	93
7.4.1. Generada por el modelo	93
7.4.2. Binarización	93
7.4.3. Resultado final	94
8. Aplicación web	97
8.1. Introducción	97
8.2. Análisis	97
8.2.1. Análisis de requisitos	97
8.2.2. Casos de uso	99
8.2.3. Diagrama de clases inicial	101
8.3. Diseño	102
8.3.1. Decisiones sobre la implementación	102
8.3.2. Patrón de diseño	102
8.3.3. Arquitectura	103
8.3.4. Diagramas de secuencia	104
8.4. Seguridad	108
8.5. Pruebas	109
9. Conclusiones y líneas futuras	111
9.1. Aprendizaje	111
9.2. Líneas futuras	112
A. Manual de instalación	113
B. Manual de usuario	115

C. Contenidos del CD-ROM	119
D. Apéndice sobre PyTorch	121
D.1. torch.nn.Module	121
D.2. Downsampling - Fase de extracción	121
D.2.1. torch.nn.Conv2d	121
D.2.2. torch.nn.BatchNorm2d	122
D.2.3. torch.nn.LeakyRelu	122
D.2.4. torch.nn.MaxPool2d	122
D.3. Upsampling - Fase de expansión	123
D.3.1. torch.nn.ConvTranspose2d	123
D.3.2. torch.nn.Dropout	123
D.3.3. torch.cat	123
D.3.4. torch.optim.lr_scheduler.ReduceLROnPlateau	123
Bibliografía	125

Índice de figuras

2.1. Metodología CRISP-DM. Fuente: [8]	7
2.2. Inicio del proyecto y gestión de datos.	8
2.3. Modelo, pruebas y conclusiones.	9
2.4. Aplicación, despliegue y elaboración de la memoria.	9
2.5. Diagrama de Gantt: Planificación Inicial.	10
3.1. Partes de una neurona biológica. Fuente: [16]	20
3.2. Neurona Artificial McCulloch y Pitts. Fuente: [17]	20
3.3. Aprendizaje Automático y Profundo subconjuntos de la Inteligencia Artificial. Fuente: [20]	21
3.4. Perceptrón Multicapa. Fuente: [21]	22
3.5. Conectividad dispersa vs conectividad densa. Fuente: [30]	24
3.6. Compartición de parámetros. Fuente: [30]	24
3.7. Primera iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]	26
3.8. Segunda iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]	26
3.9. Última iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]	26
3.10. Extracción de características. Fuente: [33]	27
3.11. Ejemplo de imagen RGB y los tres canales de información. Fuente: [34]	27
3.12. Convolución imagen RGB Fuente: [36]	28
3.13. Convolución con parámetro stride de valor 1. Fuente: [38]	29
3.14. Convolución con parámetro stride de valor 2. Fuente: [38]	29
3.15. Convolución tras aplicar <i>same padding</i> . Fuente: [40]	30
3.16. Max Pooling con filtro 2x2. Fuente: [42]	31
3.17. Min Pooling con filtro 2x2.	31
3.18. Average Pooling con filtro 2x2.	31
3.19. Unpooling: Método Nearest Neighbor. Fuente: [43]	33
3.20. Unpooling: Método Bed of Nails.	33
3.21. Unpooling: Método Max Unpooling. Fuente: [44]	33
3.22. Convolución normal vs transpuesta. Fuente: [45]	34
3.23. Ejemplo aplicación convolución transpuesta.	34
3.24. Interpolación bilineal.	35
3.25. Ejemplo interpolación bilineal en escala de grises.	35
3.26. Arquitectura U-Net. Fuente: [50]	38
3.27. Ruta de contacción de un modelo U-Net.	38

3.28. Cuello de botella de un modelo U-Net.	39
3.29. Ruta de expansión de un modelo U-Net.	39
3.30. Representación gráfica de la función sigmoide. Fuente: [53]	40
3.31. Representación gráfica de la función tangente hiperbólica. Fuente: [55]	41
3.32. Representación gráfica de ReLU. Fuente: [56]	42
3.33. Representación gráfica Leaky ReLU. Fuente: [57]	42
3.34. Conexiones de entrada y salida de una neurona. Fuente: [61]	48
3.35. Aplicación dropout. Fuente: [65]	49
3.36. Ejemplo conexión residual. Fuente: [67]	51
3.37. Ejemplo Depthwise separable convolution. Fuente: [68]	51
4.1. Imagen original.	56
4.2. Ground Truth.	56
4.3. Imagen segmentada.	56
4.4. Contenido de los directorios del conjunto de datos.	56
4.5. Distribución de clases del conjunto de datos completo	56
4.6. Distribución de clases del conjunto de entrenamiento	57
4.7. Distribución de clases del conjunto de prueba	57
5.1. Resumen del modelo. Parte 1	68
5.2. Resumen del modelo. Parte 2	69
7.1. Aprendizaje del modelo y Early Stopping.	84
7.2. Evolución de las pérdidas por iteraciones.	85
7.3. Momento atascado en un mínimo local. Fuente: [82]	85
7.4. Evolución tasa de acierto.	85
7.5. Precisión tras efecto del planificador.	86
7.6. Evolución de la Tasa de Aprendizaje.	86
7.7. Matriz de confusión. Fuente: [84].	87
7.8. Curva ROC. Clase: Obstáculos.	88
7.9. Curva ROC. Clase: Agua.	89
7.10. Curva ROC. Clase: Vegetación.	90
7.11. Curva ROC. Clase: Movimiento.	91
7.12. Curva ROC. Clase: Zona Aterrizable.	92
7.13. Probabilidades predichas sobre una imagen del conjunto de test.	93
7.14. Predicción vs Ground Truth coloreada.	94
7.15. Píxeles en negro en la predicción.	95
8.1. Diagrama de Casos de Uso	99
8.2. Diagrama de Clases inicial	101
8.3. Diagrama Uses Style General.	103
8.4. Diagrama Modules and Uses Style. Paquete: Plantilla.	103
8.5. Diagrama Modules and Uses Style. Paquete: Vista.	104

8.6. Diagrama Modules and Uses Style. Paquete: Modelo.	104
8.7. Diagrama de secuencia. CU-01: Subir imagen.	105
8.8. Uses Style CU-02. Obtener predicción.	106
8.9. Diagrama de secuencia. CU-02. Obtener predicción.	107
8.10. Diagrama de secuencia. CU-02. Mostrar resultados	108
8.11. Diagrama de secuencia. CU-03: Cancelar subida.	108
A.1. Versión de Python.	113
B.1. Inicio de la aplicación.	115
B.2. Previsualización de la imagen subida.	116
B.3. Visualización de los resultados.	116
B.4. Error cuando no se introduce ninguna imagen.	116
B.5. Formato incorrecto del archivo cargado.	117
C.1. Contenido del CD-ROM.	119

Índice de tablas

2.1. Riesgo R01	12
2.2. Riesgo R02	12
2.3. Riesgo R03	13
2.4. Riesgo R04	13
2.5. Riesgo R05	14
2.6. Riesgo R06	14
2.7. Riesgo R07	15
2.8. Riesgo R08	15
2.9. Desglose detallado del plan de costes	17
3.1. Ventajas e inconvenientes descenso del gradiente estocástico	44
3.2. Ventajas e inconvenientes descenso del gradiente mini batch	45
3.3. Ventajas e inconvenientes de la técnica Momento	45
3.4. Ventajas e inconvenientes de la técnica Adagrad	46
3.5. Ventajas e inconvenientes de la técnica RMSProp	46
3.6. Ventajas e inconvenientes de la técnica RMSProp	47
6.1. Tabla comparativa.	81
7.1. Resultados conseguidos en los distintos entrenamientos.	83
7.2. Porcentaje de precisión sobre los conjuntos.	86
7.3. Métricas de evaluacion. Clase: Obstáculos.	88
7.4. Métricas de evaluacion. Clase: Agua.	89
7.5. Métricas de evaluacion. Clase: Vegetación.	90
7.6. Métricas de evaluación. Clase: Movimiento.	91
7.7. Métricas de evaluación. Clase: Zona Aterrizable.	92
7.8. Tresholds óptimos para cada clase.	92
7.9. Gama de colores RGB para cada clase.	94
8.1. Requisitos funcionales de la aplicación	98
8.2. Requisitos no funcionales de la aplicación	98
8.3. Requisitos funcionales de información de la aplicación	98
8.4. Descripción del CU-01: Subir imagen	99
8.5. Descripción del CU-02: Obtener predicción	100

8.6. Descripción del CU-03: Cancelar subida	101
---	-----

Capítulo 1

Introducción

1.1. Contexto actual

La Inteligencia Artificial (IA) se ha convertido en uno de los campos de mayor interés en la sociedad en los últimos años, debido a los avances significativos y la revolución tecnológica que ha causado en nuestras vidas. Su impacto se ha hecho especialmente evidente en diversos sectores, que van desde la atención médica hasta la industria automotriz, transformando la forma habitual a la que estábamos acostumbrados.

El auge de la IA y transversalidad, trae consigo dos sentimientos opuestos. Por un lado, el temor a que esto pueda modificar de forma irreversible la vida tradicional, tal y como la conocemos. Por otro lado, la expectación del alcance y los notables avances que puede implicar. A pesar de los temores, totalmente comprensibles, la IA puede ser sumamente beneficiosa y valiosa en numerosos campos, debido a su inmenso potencial.

Actualmente, la IA nos ha brindado una pequeña demostración de sus capacidades, instándonos a explorar sus aplicaciones en múltiples áreas, con el principal cometido de lograr un impacto positivo para la sociedad y solucionar de forma eficaz y más sencilla muchos problemas, tanto en ámbitos industriales, como en nuestra vida cotidiana.

1.2. Origen del proyecto

El punto de partida de este Trabajo de Fin de Grado (TFG) se encuentra en un reto/desafío que surgió en la plataforma Kaggle [1], que coincidió con mi idea original de desarrollar una Red Neuronal capaz de identificar diversos objetos a partir de imágenes. Dicho desafío se centra en la creación de un modelo de Deep Learning, que utiliza la segmentación de imágenes, para detectar y distinguir zonas de aterrizaje seguras para drones, de otros objetos y obstáculos.

Para lograr dicho objetivo, Kaggle proporciona un conjunto de datos, que incluye imágenes reales capturadas por un dron, así como su correspondiente mapa (Ground-truth) que marcan de manera explícita las diferentes zonas, que se pretenden diferenciar. Este conjunto de datos ha servido como base para el desarrollo del modelo, lo que ha permitido explorar de manera práctica el potencial de ciertos sistemas de Deep Learning en tareas de segmentación de imágenes.

A lo largo de este TFG, se explorarán los detalles del proceso y los resultados obtenidos a partir de este desafío inicial, en el contexto de la Visión por Computadora y el Aprendizaje Profundo.

1.3. Motivación

Cada vez con mayor frecuencia [2], los drones son empleados en diversos sectores, desempeñando tareas o labores que resultarían complicadas para el ser humano u otras máquinas.

Aunque, en un principio, los drones estaban diseñados con fines meramente recreativos, la posibilidad de integrar elementos externos como cámaras, sensores y sistemas GPS, entre otros, los convierte en herramientas ideales para desempeñar otras funciones. Algunas de estas tareas son la búsqueda y rescate de personas, periodismo,

monitorización de instalaciones, cuidado de cultivos y la lucha contra los incendios forestales, entre otras.

La Inteligencia Artificial, que está estrechamente vinculada a las funciones que hemos mencionado anteriormente, amplía significativamente las aplicaciones posibles y su utilidad, cuando se combina de manera efectiva. Esto permite la automatización de tareas y la toma de decisiones inteligentes en tiempo real.

En este sentido, el Trabajo Fin de Grado propuesto tiene como objetivo principal identificar posibles zonas de aterrizaje seguro, distinguiendo éstas de otros elementos presentes en la imagen; entre ellos, objetos en movimiento, agua, vegetación u obstáculos. Asimismo, se podría utilizar la identificación del resto de objetos para desempeñar otras tareas y, así, aprovechar más eficientemente el modelo desarrollado, como veremos en el **Capítulo 9**.

Identificar y establecer zonas de aterrizaje seguras tiene una importancia crucial para drones que desempeñan labores críticas, como las comentadas anteriormente. En la ejecución de estas tareas, la seguridad y la integridad del dron son factores de relevancia. Los drones, por lo general, representan una inversión significativa, por lo que un aterrizaje en áreas no apropiadas podría conllevar daños en su estructura, que implicarían costosas reparaciones, o incluso interrumpir la tarea que se esté llevando a cabo. Garantizar zonas de aterrizaje seguras supondría prevenir posibles accidentes, tanto en términos de la seguridad de los propios drones, como de la protección de personas y propiedades. En este contexto, la identificación y la gestión eficaz de zonas de aterrizaje seguras constituyen un componente crítico para el éxito, viabilidad y coste de estas misiones.

El uso de un modelo de Deep Learning para el problema anteriormente expuesto, puede ser una solución innovadora y efectiva para los sectores que integran drones en la ejecución de muchas de sus tareas. En particular, el modelo de Red Neuronal Convolutiva propuesto, permite realizar la segmentación y obtener como salida un plano diferente por cada uno de los distintos objetos identificados, pudiendo distinguir con claridad los distintos componentes diferenciados por el modelo, alcanzando así el objetivo principal propuesto en este trabajo. Precisamente, esta capacidad nos brinda un amplio espectro de oportunidades ya que, aunque el Trabajo de Fin de Grado (TFG) se enfoca en la identificación de zonas de aterrizaje seguras, las demás categorías, como objetos en movimiento, vegetación, agua u obstáculos, podrían emplearse con diferentes propósitos adicionales.

1.4. Objetivos del proyecto

Este proyecto se centra en la investigación y posterior desarrollo de un modelo de Deep Learning para la segmentación de imágenes. Dada la limitada experiencia previa en el campo, se llevará a cabo una importante etapa de comprensión teórica y adquisición de conocimientos relacionados con las Redes Neuronales Convolucionales, especialmente en lo que respecta al modelo elegido UNET. En este contexto, se han establecido los siguientes objetivos:

- Análisis exhaustivo del problema comprendiéndolo a fondo, conociendo qué pretendemos abordar y la utilidad ante situaciones reales, así como familiarizarnos con el conjunto de datos disponible para este propósito.
- Análisis y procesamiento de los datos, adecuando las imágenes del conjunto de datos y permitiendo que sirvan como entrada al modelo propuesto.
- Diseño y desarrollo de un modelo basado en Redes Neuronales Convolucionales, con la implementación del modelo UNET y la definición de sus parámetros, incluyendo el reajuste de ellos para mejorar los resultados finales.
- Realización de pruebas, análisis de los resultados y comparaciones. Se pretende probar el modelo desarrollado, el cual no sólo debe superar una alta tasa de clasificación correcta, sino también demostrar robustez. Con el fin de justificar las decisiones en la configuración del entrenamiento del modelo, se realizarán comparaciones entre algunas de las mejores opciones, aportando así un análisis fundamentado en el rendimiento y la eficiencia de distintas alternativas.
- Diseño e implementación de una aplicación web, con la que probar el modelo desarrollado. El principal cometido de dicha aplicación, será proporcionar al usuario una interfaz sencilla e interactiva, que permita la carga de una imagen. La aplicación nos ofrecerá una visualización de la imagen segmentada coloreada y la imagen original, facilitando la interpretación y comparación de la clasificación realizada.

1.5. Estructura

En esta Sección, se describe la estructura y un breve resumen del contenido de cada uno de los capítulos que constituyen la memoria de este Trabajo Fin de Grado:

- **Capítulo 1: Introducción.** Se presenta el contexto actual, la motivación para la realización del proyecto, así como los objetivos que se pretenden alcanzar.
- **Capítulo 2: Planificación del proyecto.** Se describe la planificación del proyecto y organización de las distintas tareas, así como la estimación de tiempo para cada una de ellas, un plan de riesgos y su gestión. Por último, un plan que abarque los costes del proyecto.
- **Capítulo 3: Marco teórico.** Se presentan los fundamentos teóricos esenciales que resultan necesarios para comprender el proyecto y su funcionamiento, poniendo un énfasis particular en las Redes Neuronales Convolucionales y en el modelo concreto elegido.
- **Capítulo 4: Conjunto de datos.** Se presenta el origen, características, descripción y tratamiento de los datos utilizados para el entrenamiento y evaluación del modelo desarrollado.
- **Capítulo 5: Desarrollo completo del modelo.** Se detalla su diseño y su arquitectura utilizado, para ello, Redes Neuronales Convolucionales.
- **Capítulo 6: Implementación y herramientas.** Se exponen las herramientas utilizadas para el progreso del proyecto.
- **Capítulo 7: Resultados.** Se muestran los resultados obtenidos para distintas configuraciones, una evaluación de la efectividad y acierto del modelo definitivo desarrollado.
- **Capítulo 8: Aplicación web.** Se aborda todo lo relativo al análisis, diseño, funcionalidad, desarrollo e integración de la aplicación web del proyecto.
- **Capítulo 9: Conclusiones y líneas futuras.** Se lleva a cabo una retrospectiva, recopilando las lecciones aprendidas a lo largo de este TFG y su relación con las asignaturas del grado. Finalmente, se proponen líneas futuras de trabajo, que podrían mejorar y complementar este proyecto.

Capítulo 2

Planificación del proyecto

Dada la envergadura y complejidad de este Trabajo Fin de Grado, considero esencial llevar a cabo una adecuada gestión, planificación de las tareas y riesgos para cumplir con los objetivos establecidos. En este capítulo, se expondrá la metodología empleada, la planificación inicial desarrollada, las variaciones finales que se han producido frente a la planificación inicial, los principales hitos, un plan de gestión de riesgos y uno de costes.

2.1. Metodología de trabajo

Para seleccionar la metodología más adecuada para este TFG, es fundamental considerar el contexto en el que se encuentra enmarcado. En particular, estaría dentro del campo de la Minería de Datos y el Aprendizaje Profundo. Su objetivo es utilizar datos previamente procesados como entrada para un modelo de Aprendizaje Profundo, con el propósito de extraer conocimiento y patrones, lo que permitirá identificar diversos objetos. Las metodologías que mejor se adaptan a proyectos de esta naturaleza, son las siguientes: KDD (Knowledge Discovery in Databases) [3], SEMMA (Sample, Explore, Modify, Model, Assess) [4] y CRISP-DM (Cross-Industry Standard Process for Data Mining) [5]. Dentro de las tres metodologías anteriormente mencionadas, CRISP-DM se alinea de manera más consistente con el desarrollo de nuestro proyecto.

No obstante, a pesar de lo anteriormente mencionado, considero apropiado aplicar una metodología híbrida, porque los objetivos y requisitos están claramente definidos, sobretodo, porque ciertas fases o etapas deben llevarse a cabo de manera secuencial, lo que se alinea con un enfoque de tipo cascada (Waterfall). En vista de esto, la combinación que más se adecúa a todos los ámbitos del proyecto, implica adoptar una metodología híbrida, incorporando elementos de ambas Waterfall y CRISP-DM.

La forma óptima de combinar ambas implicaría desarrollar una metodología Waterfall para la fase inicial, englobando la definición de requerimientos, objetivos y diseño del proyecto, es decir, la etapa antes de comenzar con la implementación. La metodología CRISP-DM vendría en la fase de Minería de Datos, mucho más iterativa y flexible. Englobará tareas como el análisis de los datos, construcción del modelo, resultados, etc. Con todo esto, se pretende proporcionar la metodología más adecuada en cada momento, aportando un enfoque secuencial en la parte inicial y optar por un planteamiento más flexible al iniciar la etapa más práctica.

Waterfall

También conocida como cascada [6], es una metodología para la gestión de proyectos. Presenta un enfoque secuencial que divide el proyecto en distintas fases. En este flujo de fases, ninguna comienza hasta que se haya terminado la anterior.

Se emplea típicamente en proyectos con poca variabilidad, donde cada una de las etapas y los objetivos se encuentran bien definidos. En términos generales, se representa mediante diagramas de flujo o de Gantt, en los cuales, tras completar un conjunto de tareas, se progresa a la siguiente fase de manera secuencial. Este enfoque metodológico se caracteriza por seguir un proceso de seis fases dispuestas en un orden preestablecido, lo que promueve una ejecución lineal y ordenada.

1. Requisitos.

Se recopila la información necesaria con el fin de tratar de garantizar el éxito del proyecto. Tras esta etapa,

tenemos que tener claro los recursos necesarios, una estimación realista de los tiempos empleados en cada tarea y el alcance del proyecto.

2. Diseño.

Se puede subdividir a su vez en dos: diseño de alto nivel y diseño detallado. En la primera, se lleva a cabo la tarea de *brainstorming*, o lluvia de ideas, y posibles soluciones a ellas. En la segunda, se definen detalles particulares del diseño de alto nivel.

3. Implementación.

Se lleva a cabo la codificación, tomando la información de las dos etapas anteriores, para la construcción de un 'producto' funcional.

4. Pruebas.

El objetivo de esta fase es detectar posibles errores y garantizar que el producto cumple con los requisitos especificados. En caso de que se encuentre algún error, éste podrá ser corregido antes de pasar a la siguiente fase.

5. Desarrollo.

Se entrega al cliente el producto completo para que sea revisado y se cerciore de que los requisitos expuestos, se cumplen.

6. Mantenimiento.

Una vez que el producto ha sido entregado, el cliente estará utilizándolo, por lo que es posible que surjan problemas, que hay que solucionar mediante actualizaciones.

Como se puede apreciar en la **Figura 2.5**, se ha realizado un diagrama de Gantt, que detalla la secuencia de las tareas, junto a una estimación temporal realista sobre la duración de ellas. Aunque este proyecto se relaciona más con el ámbito de la Minería de Datos, que con uno típico de Ingeniería de Software, se ha optado por emplear la metodología en cascada en ciertas fases, en particular, es aplicable a la fase inicial, que abarca la definición de requisitos, planificación y diseño del modelo. Las etapas de pruebas y mantenimiento también demandan este enfoque, ya que el propósito principal de este proyecto radica en la implementación de una aplicación de segmentación de imágenes, la cual debe someterse a rigurosas pruebas de funcionamiento, tanto en el modelo, como en la aplicación, así como contar con un soporte para posibles fallos o errores. No obstante, para la fase más práctica relacionada con la Minería de Datos, se ha preferido utilizar una metodología que se ajusta de manera óptima a este tipo de proyectos como es CRISP-DM.

CRISP-DM

Es una metodología [7] adaptable y flexible diseñada específicamente para proyectos en el campo de la Minería de Datos, es decir, aquellos proyectos cuyo propósito principal es extraer conocimiento de los datos. Esta metodología, como se ilustra en la **Figura 2.1**, sigue una estructura cíclica. En el dominio de la Minería de Datos y, en general, en el ámbito de la Ciencia de Datos, la naturaleza cíclica desempeña un papel fundamental, ya que a través del refinamiento y las iteraciones se logra mejorar los resultados obtenidos.

Podemos distinguir 6 fases principales dentro del esquema de CRISP-DM:

1. Comprensión del negocio.

Se centra en delimitar claramente los requisitos y objetivos del proyecto, a través de un enfoque empresarial, para traducirlo a nuestro problema de Minería de Datos.

2. Comprensión de los datos.

Engloba tareas relacionadas con los datos, entre las que se encuentra la elección inicial de conjunto de datos, familiarizarse con ellos e identificar posibles problemas o transformaciones que puedan provenir de ellos.

3. Preparación de los datos.

Abarca todas las tareas necesarias para construir el conjunto de datos final, el que recibirá como entrada el modelo implementado. En esta fase, se llevan a cabo las tareas de preprocesamiento de los datos y transformaciones necesarias para que puedan ser utilizados como entrada.

4. Modelado.

En esta etapa se aplican técnicas de modelado y se calibran los parámetros de la red, con el fin de obtener valores óptimos.

5. Evaluación.

Se realizan pruebas de forma exhaustiva, con el propósito de revisar que cumpla todos los objetivos propuestos.

6. Despliegue.

Hasta ahora tenemos el modelo creado y probado, pero falta crear una aplicación, para presentarse y que pueda probarlo el usuario final.

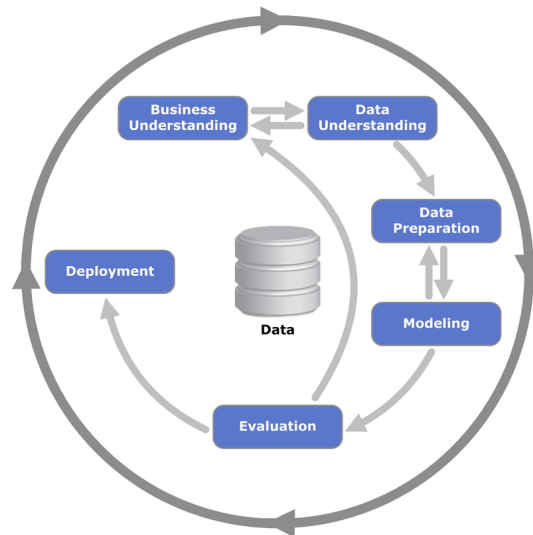


Figura 2.1: Metodología CRISP-DM. Fuente: [8]

En la fase práctica de este proyecto, vemos en el diagrama de Gantt tareas definidas como *la gestión de datos, desarrollo del modelo, pruebas y conclusiones, despliegue y aplicación*, que están estrechamente vinculadas a la Minería de Datos y que se ajustan de manera significativa a la metodología CRISP-DM.

2.2. Entregables

Relacionado con el enfoque secuencial de la metodología, se han establecido una serie de hitos que actuarán como entregables. Estos se definen como puntos clave en el avance y progreso del proyecto, ya que representan etapas fundamentales que deben completarse antes de avanzar a la siguiente fase. Esta estructura asegura un control y seguimiento constante sobre el proyecto, lo que permite llevar a cabo revisiones y mejoras de forma eficiente y parcial.

Así pues, se han establecido los siguientes hitos:

- **Hito 1:** Estudio y adquisición de conocimientos previos, búsqueda de conjunto de datos y comprensión del problema planteado
- **Hito 2:** Comprensión de los datos, carga y procesado.
- **Hito 3:** Desarrollo de un modelo de segmentación básico y funcional.
- **Hito 4:** Realización de pruebas, análisis de los resultados y refinamiento del modelo, buscando alcanzar su rendimiento óptimo.
- **Hito 5:** Desarrollo, despliegue y montaje del modelo desarrollado en una aplicación web implementada en un entorno Docker
- **Hito 6:** Redacción de la memoria.

A pesar de que el Hito 1 no produce ningún entregable, se considera uno más por su gran relevancia en las tareas que se desempeñan en esta fase, ya que proporcionan los fundamentos, sobre todo, teóricos, para iniciar la realización de este proyecto.

2.3. Planificación inicial

Esta sección presenta un primer esquema para la ejecución y éxito del proyecto. Abarca una estimación temporal de las tareas involucradas, así como una fecha de inicio y fin estimada para cada una de ellas.

El plan de trabajo propuesto se ajusta a la guía docente [9], que establece que el TFG tiene una carga de 12 ECTS, es decir, de aproximadamente 300 horas. Inicialmente, y con la idea de adaptarnos a lo anteriormente mencionado, se ha planteado dedicar cuatro horas al proyecto diariamente. Aunque no se ha cumplido estrictamente todos los días, de forma general sí se ha respetado. Con el objetivo de mantener un seguimiento constante del proyecto, se ha acordado mantener reuniones quincenales con el tutor, mayoritariamente de forma presencial pero, en caso de ser necesario, se ha considerado también el formato online. Durante estas reuniones, se revisa el trabajo y se establecen objetivos o metas, que se deben alcanzar para el próximo encuentro. En caso de surgir algún contratiempo o riesgo de los que veremos en la **Sección 2.5**, esta reunión se aplazaría un tiempo máximo de una semana.

A pesar de que el contacto con el tutor empezó a mediados de Septiembre de este 2023, proyecto ha tenido comienzo el 29 de Septiembre de 2023 y se estima que finalizará el 14 de Enero de 2024. Esta fecha no tiene en cuenta diversos factores que podrían ocasionar retrasos en los plazos previstos.

El diagrama de Gantt de la **Figura 2.5** muestra el orden de realización de las tareas, así como una estimación del tiempo que se empleará en la realización de cada una de ellas. Aunque en esta figura obtenemos una visión general de la planificación, no podemos observar con detalle las fechas y duración prevista para cada una de las tareas. Por ello, voy a realizar un desglose en las **Figuras 2.2,2.3,2.4**.

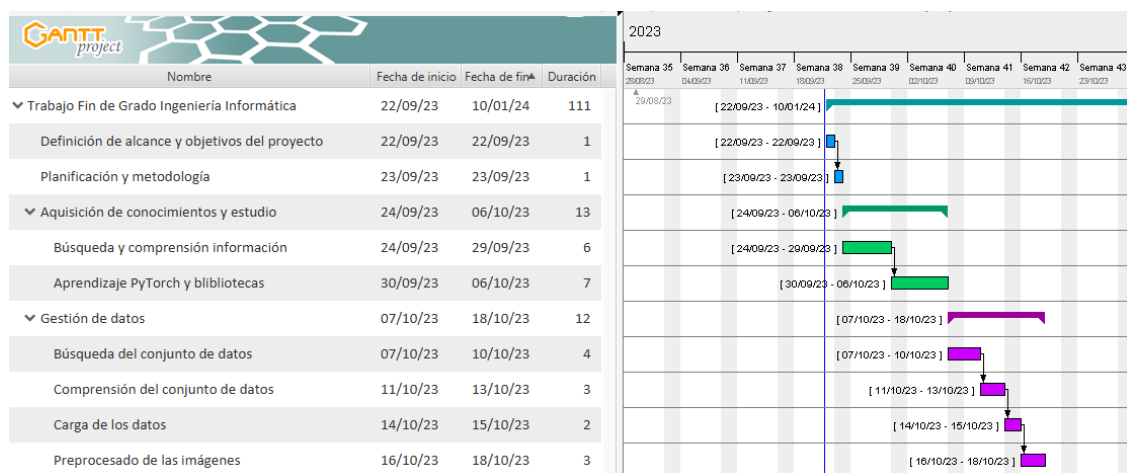


Figura 2.2: Inicio del proyecto y gestión de datos.

En el tramo inicial del proyecto, se observa que las tareas se centran en la planificación y organización del mismo. Se incluyen actividades de adquisición de conocimientos y un estudio previo antes de iniciar la fase práctica del trabajo.

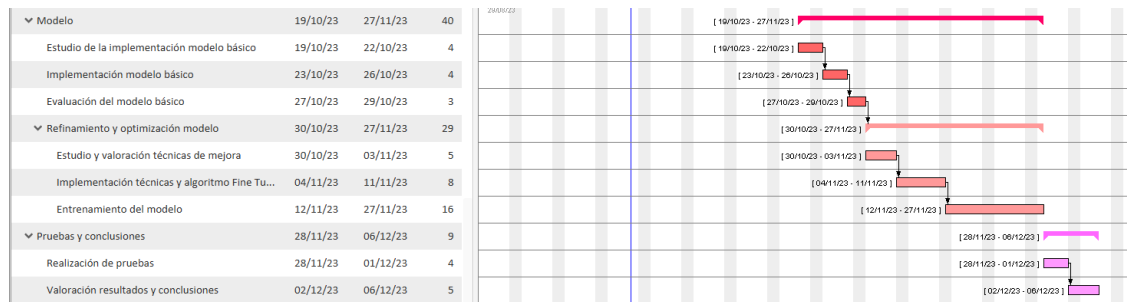


Figura 2.3: Modelo, pruebas y conclusiones.

En el tramo intermedio del proyecto, se desarrollan tareas relacionadas con el desarrollo e implementación del modelo, pruebas y conclusiones. La fase de pruebas es crucial para evaluar la efectividad del modelo propuesto, lo que nos permite presentar posteriormente las conclusiones.

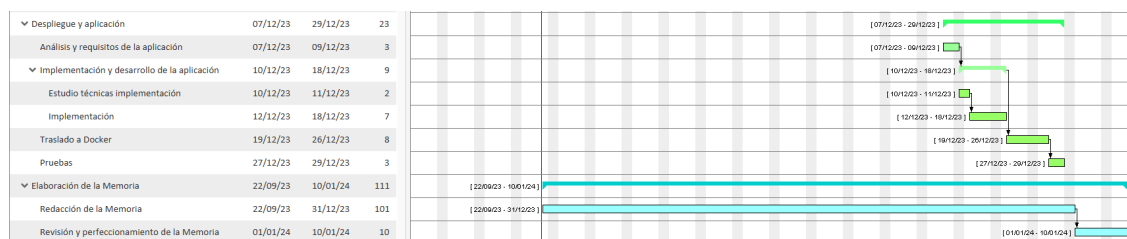


Figura 2.4: Aplicación, despliegue y elaboración de la memoria.

La etapa final del proyecto se enfoca en la implementación aplicación y el posterior despliegue del modelo.

Como se puede apreciar en la **Figura 2.5**, la tarea de mayor duración es la *Elaboración de la Memoria*, la cual fue planificada para extenderse a lo largo de todo el proyecto desde su inicio. Excluyendo esta tarea, el diseño, desarrollo, optimización, refinamiento y entrenamiento del modelo constituyen las siguientes tareas de mayor duración que se engloban en una única denominada *Modelo*, con una estimación de duración de 40 días. Esta fase se considera fundamental, por lo que se ha asignado el tiempo necesario para llevarla a cabo de manera adecuada. Aunque su estimación se ha realizado con cierta incertidumbre, dado que la duración del entrenamiento del modelo es algo que no podemos conocer exactamente durante la planificación, únicamente disponemos de información sobre el tamaño del conjunto de datos y sobre el modelo que se pretende aplicar. Con todo esto, he considerado la estimación más realista posible.

En la planificación inicial presentada, no se han incluido explícitamente las reuniones periódicas mantenidas con el tutor, que se han llevado a cabo de manera regular, con frecuencia quincenal, a excepción de aquellas en las que se registraron avances mínimos en el proyecto. Estas interacciones han proporcionado un marco fundamental para la organización y seguimiento efectivo del proyecto.

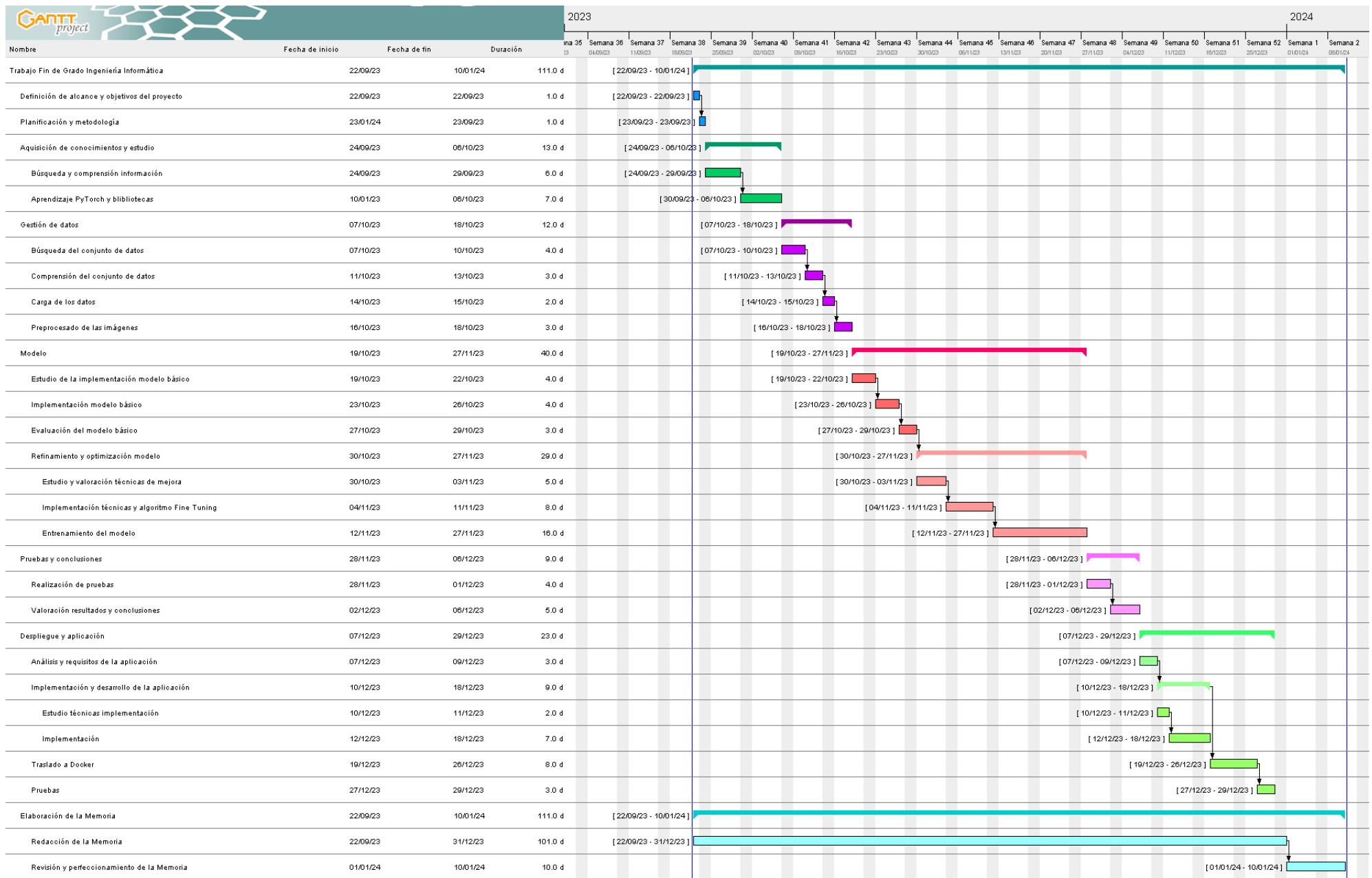


Figura 2.5: Diagrama de Gantt: Planificación Inicial.

2.4. Variaciones sobre la planificación inicial

A pesar de haber procurado realizar una planificación inicial lo más realista posible, se han presentado ligeras variaciones en los plazos establecidos durante el transcurso del proyecto.

Dejando de lado las discrepancias menores en la estimación del tiempo para algunas tareas, donde tomaron más tiempo del previsto, pero fueron compensadas por otras que se completaron más rápidamente, han surgido tres cambios significativos o desviaciones respecto a la planificación inicial.

- La carga de datos estaba originalmente programada para la segunda quincena de octubre. Sin embargo, debido a un problema externo, no pudimos acceder a la máquina virtual hasta casi finales de ese mes. Afortunadamente, esto no provocó un retraso en el progreso, ya que empleé ese tiempo para avanzar en la redacción de la parte técnica de la memoria del proyecto. Durante este periodo, también estuve realizando pruebas sobre la carga de datos en mi máquina personal.
- Inicialmente se planteó la implementación de un Algoritmo Fine-Tuning, que permitiera adaptar la tasa de aprendizaje durante la fase de entrenamiento del modelo. No obstante, durante la configuración del entrenamiento, al consultar la documentación de PyTorch, descubrí la existencia de planificadores ya programados que realizaban esta tarea. En consecuencia, se decidió utilizar una de estas opciones en lugar de desarrollar un nuevo algoritmo. El tiempo asignado para esta tarea se empleó en el refinamiento del modelo y la realización de diversas pruebas.
- El comienzo del desarrollo de la aplicación web coincidió con los períodos de entrenamiento del modelo, ya que eran momentos en los que no se podían realizar cambios en el código, debido a que el entrenamiento ya habría concluido. Para aprovechar al máximo la productividad durante estos períodos, se avanzó en la aplicación, logrando una versión básica a principios de diciembre. El tiempo inicialmente asignado para la aplicación se destinó al refinamiento y mejora de la redacción de la memoria.

A pesar de las desviaciones o cambios mencionados, no se ha producido un retraso en el plazo general de finalización del proyecto. Debido a la similitud con la planificación inicial, se ha decidido abstenerse de modificar el diagrama de Gantt inicial, optando en su lugar por proporcionar un breve comentario sobre estos cambios en esta sección.

2.5. Plan de riesgos

Aparte de elaborar una planificación detallada, es importante desarrollar un plan de riesgos, porque permite enfrentarnos a posibles contratiempos, puesto que siempre existe incertidumbre y variabilidad. En este contexto, la planificación y gestión de los riesgos pueden contribuir significativamente a lograr los objetivos, sin que conlleve retrasos significativos en el tiempo establecido.

El plan de riesgos [10] contempla su identificación y su análisis, a partir cuatro elementos clave que son fundamentales para clasificarlos y evaluarlos de forma adecuada.

- **Probabilidad.** Se refiere a la frecuencia con la que una incidencia podría suceder, etiquetándose de baja, media o alta.
- **Impacto.** Se refiere a la magnitud del efecto que tendría dicho riesgo, si llega a materializarse, también categorizándose en bajo, medio o alto.
- **Mitigación.** Acciones que se llevan a cabo, con el fin de reducir la probabilidad de que dicho riesgo suceda.
- **Contingencia.** Medidas a implementar, una vez el riesgo se materializa, con el fin de reducir su impacto.

El propósito del plan de gestión de riesgos elaborado es asegurar el éxito del proyecto, lo que permite anticipar y prevenir de manera efectiva los posibles contratiempos, así como de diseñar estrategias de contingencia, en caso de que éstos se materialicen.

Se presentan los principales riesgos identificados en las Tablas **2.1** a **2.8**.

Riesgo R01	Falta de disponibilidad
Descripción	Debido a la extensa duración del proyecto, es posible que existan problemas de disponibilidad por parte del estudiante. Podría requerir cambios en la planificación quincenal para adecuarse a las circunstancias.
Probabilidad	Media
Impacto	Medio
Mitigación	■ Equilibrar las responsabilidades, vida personal y gestión del proyecto para prevenir el cansancio y el estrés.
Contingencia	■ Tareas u objetivos que no ha dado tiempo a completar, en la quincena prevista distribuirlos para realizar la quincena siguiente. ■ Planificar con flexibilidad y margen.

Tabla 2.1: Riesgo R01

Riesgo R02	Falta de experiencia en el Aprendizaje Profundo
Descripción	El estudiante carece de experiencia en Aprendizaje Profundo, y no está familiarizado con las herramientas. Podría haber dificultades en la implementación, ajuste y optimización del modelo de segmentación.
Probabilidad	Alta
Impacto	Medio
Mitigación	■ Adquirir conocimiento previo sobre las herramientas y el modelo a desarrollar en el proyecto, antes de comenzar, mediante cursos, tutoriales y documentación entre otros.
Contingencia	■ Buscar herramientas o alternativas factibles ya conocidas por el estudiante, en lugar de recurrir a nuevas herramientas. ■ Buscar asesoría de un experto o alguien que tenga conocimientos en la herramienta, si el conocimiento adquirido es insuficiente. ■ Consultar documentación o tutoriales que expliquen el uso de las herramientas.

Tabla 2.2: Riesgo R02

Riesgo R03	Error en la estimación del tiempo previsto en tareas
Descripción	La estimación de tiempo prevista para alguna de las tareas planificadas, no se ajusta a la realidad, lo que puede producir retrasos o adelantos en la finalización del proyecto.
Probabilidad	Alta
Impacto	Medio
Mitigación	■ Realizar un análisis exhaustivo de cada tarea para poder obtener la estimación más realista posible. ■ Contar con un margen o periodo extra de tiempo al final del proyecto, de forma que si se producen atrasos este tiempo extra sirva para cubrirlos.
Contingencia	■ Usar ese margen de tiempo para completar fases del proyecto. ■ Reducir el alcance del proyecto en la medida de lo posible.

Tabla 2.3: Riesgo R03

Riesgo R04	Complejidad del modelo
Descripción	El modelo de segmentación puede resultar más complejo de lo esperado, lo que podría implicar retrasos en la implementación y dificultades en su comprensión.
Probabilidad	Alta
Impacto	Alto
Mitigación	■ Realizar un análisis detallado de la complejidad del modelo antes de comenzar. ■ Dividir el proyecto en etapas y definir hitos intermedios para el seguimiento.
Contingencia	■ Si el modelo óptimo planteado es demasiado complejo, considerar simplificar el modelo planteado. ■ Consultar y pedir ayuda a alguien experto o con conocimientos en el modelo planteado.

Tabla 2.4: Riesgo R04

Riesgo R05	Dificultades en la interpretación de resultados
Descripción	La interpretación de los resultados del modelo puede ser complicada, lo que podría dificultar la toma de decisiones y la obtención de conclusiones basadas en los resultados obtenidos por el modelo.
Probabilidad	Media
Impacto	Medio
Mitigación	■ Obtener conocimientos previos en la visualización de datos. ■ Trabajar en la visualización de resultados y en la interpretación de las predicciones del modelo.
Contingencia	■ Buscar herramientas de visualización. ■ Si los datos están visualizados pero no conseguimos obtener conclusiones sobre ellos, buscar asesoramiento una persona con conocimientos en interpretación de resultados.

Tabla 2.5: Riesgo R05

Riesgo R06	Modificación de los requisitos/alcance del proyecto
Descripción	A lo largo del proyecto, pueden surgir cambios que afecten a los requisitos y alcance del proyecto. Afectando a la planificación inicial o el alcance final.
Probabilidad	Baja
Impacto	Medio
Mitigación	■ Desde el inicio definir y priorizar las funcionalidades esenciales e importantes, evitando los cambios en estas, y manteniendo la estabilidad del proyecto. ■ Mantener retroalimentación continua con el tutor, para validar el progreso y hacer ajustes o mejoras que impliquen las menos perturbaciones posibles.
Contingencia	■ Si afecta a una parte esencial o prioritaria del proyecto, hacer uso del tiempo extra planificado para completarlo adecuadamente. ■ Si no es esencial, considerar la posibilidad de realizar cambios, siempre que sean de menor magnitud y no produzcan retrasos considerables en el resto de tareas del proyecto. ■ Si no es esencial, e implica cambios significativos, dejarlo fuera de la planificación y plantearlo como línea futura.

Tabla 2.6: Riesgo R06

Riesgo R07	Dificultades en la evaluación del modelo
Descripción	Evaluar la eficacia del modelo de segmentación puede ser complicado y subjetivo, lo que podría llevar a resultados ambiguos.
Probabilidad	Media
Impacto	Medio
Mitigación	■ Definir métricas de evaluación claras y objetivas antes de comenzar el proyecto. ■ Realizar varias pruebas y obtener opiniones externas sobre la evaluación.
Contingencia	■ Considerar la posibilidad de consultar a expertos en el campo de Visión por Computadora y Machine Learning.

Tabla 2.7: Riesgo R07

Riesgo R08	Recursos de hardware insuficientes
Descripción	El entrenamiento del modelo pueden requerir recursos de hardware significativos, y la falta de acceso a estos recursos podría ser un obstáculo.
Probabilidad	Alta
Impacto	Alto
Mitigación	■ Planificar con anticipación los requisitos de hardware y buscar acceso a recursos como GPU de alto rendimiento o servicios en la nube.
Contingencia	■ En el caso de que los recursos sean insuficientes solicitar utilizar una máquina virtual proporcionada por la Facultad de Ingeniería Informática. ■ Solicitar a alguien, la cesión de una máquina con mejores recursos. ■ Buscar el acceso a servicios en la nube. ■ Realizar una labor de optimización del modelo.

Tabla 2.8: Riesgo R08

2.6. Plan de costes

Para concluir con la planificación de este proyecto, considero fundamental elaborar un presupuesto general, que abarque los costes de las diversas áreas esenciales, como son: hardware, software y recursos humanos. Elementos de coste adicionales o infraestructura, como la electricidad, desgaste de los componentes o el lugar de trabajo, no se han incluido en este plan global de costes. La elaboración de un presupuesto, o plan de costes, es una etapa crítica en todos los proyectos, ya que ofrece una estimación del desembolso económico que implica su consecución.

Este proyecto tiene prevista una duración de aproximadamente 5 meses: desde mediados de septiembre de 2023, hasta mediados de enero de 2024. Por lo tanto, este plan abarcará los costes totales a lo largo de todo el período estimado.

2.6.1. Coste hardware

Engloba el coste estimado de todos los dispositivos y máquinas empleadas en la realización del proyecto. Los dispositivos utilizados son:

- Ordenador portátil Asus TUF Gaming FX505DTFX505DT (16GB de RAM, procesador AMD Ryzen 7 3750H y tarjeta gráfica NVIDIA GeForce RTX 1650). Actualmente tiene un coste de 715,00€ en la página web de PcComponentes [11]. Aunque no es exactamente el mismo ordenador, presenta características similares y un precio parecido al pagado en su momento. Suponiendo que la vida útil del mismo es de 5 años, y que la duración estimada del proyecto son 5 meses, el coste a tener en cuenta tras realizar los cálculos pertinentes es de 59,58€.
- Máquina virtual: prestada por el Departamento de Informática (ATC, CCIA y LSI) de la Universidad de Valladolid. (16GB de RAM, 8 cores Intel(r) Xeon(r) Gold 6326 cpu @ 2.90ghz). El precio estimado de cada uno de estos cores en el mercado, está entorno a los 1520€, según la página oficial de Intel [12]. Asimismo estimamos un precio a la RAM entorno a los 50€ según PcComponentes [13]. No tendremos en cuenta el costo de la tarjeta gráfica, que ha sido utilizada en momentos puntuales del trabajo, cuyo coste es difícil estimar. Tras realizar las operaciones necesarias, el precio total de la máquina virtual se estima en 12200€. Si tenemos en cuenta que su vida útil es de unos 10 años aproximadamente, el coste de uso de esta máquina en el proyecto es de unos 508,33€.
- Monitor LG HD 60Hz 32" con un coste aproximado de 199,00€ y una vida útil de unos 10 años. Realizando los cálculos pertinentes, el coste del uso de este monitor es de unos 8,29€.

El coste total de hardware es de 582,87€.

2.6.2. Coste software

Para la realización de este proyecto, se han utilizado programas software de uso gratuito. Al ser desarrollado por un estudiante de la Universidad de Valladolid, ésta proporciona licencias de forma gratuita para algunos de los programas, por lo que el uso de ellos no ha supuesto desembolso alguno. Sin embargo, el desarrollo de este mismo proyecto en otro entorno, como el empresarial, hubiese supuesto un coste adicional, ya que se habrían tenido en cuenta el precio de las licencias de los programas software empleados.

2.6.3. Coste de recursos humanos

Para este proyecto, desarrollado en un ámbito académico, el coste de recursos humanos no se aplica. Si estuviésemos en un ámbito empresarial, sí que sería necesario tener en cuenta este coste. La duración estimada de este TFG por la guía docente de la asignatura [9] es aproximadamente 300 horas. En España, el salario bruto de un desarrollador perfil junior por hora, según *talent.com*, es de unos 13.83€ aproximadamente, por lo que el montante aproximado sería de unos 4150€.

2.6.4. Coste total

Para poder calcularlo, se han sumado las cantidades parciales de cada uno de los aspectos que hemos tenido en cuenta para este plan de costes, al que se ha añadido un margen de 20 % extra, que pueda cubrir pequeños desfases en la estimación económica realizada. En la **Tabla 2.9** se muestra un desglose de todo esto.

Concepto	Coste
Coste hardware	576,18€
Coste software	No se aplica a nuestro proyecto (0€)
Coste de recursos humanos	4150€
Coste total	4726,18€
Coste total + margen	5671,416€

Tabla 2.9: Desglose detallado del plan de costes

Capítulo 3

Marco Teórico

En este capítulo, se presentan los conocimientos fundamentales para comprender el posterior desarrollo de este Trabajo de Fin de Grado. Su comprensión constituye un pilar sólido y esencial, ya que conforman la base necesaria para entender con éxito el trabajo experimental de este proyecto. Se abordan los principios teóricos que sustentan las Redes Neuronales en general, las Convolucionales en particular, y un enfoque para el análisis detallado de la arquitectura UNET. Dado que este modelo no ha sido previamente explorado durante el transcurso del Grado, comprender su funcionamiento y la capacidad de diseñar un modelo de este tipo, se perfilan como objetivos fundamentales de este trabajo.

3.1. Introducción Redes Neuronales

Se utilizan en una gran variedad de aplicaciones, como el reconocimiento de patrones, la clasificación de datos, el procesamiento de Lenguaje Natural, la Visión por Computadora y muchas otras áreas de la Inteligencia Artificial. Su capacidad para modelar relaciones complejas y extraer características significativas de los datos, las convierte en poderosas herramientas para la resolución de problemas del mundo real.

Para entender su funcionamiento, es necesario comprender y familiarizarse con algunos de los conceptos importantes que se van a describir a lo largo de este Capítulo.

3.2. Redes Neuronales Artificiales

Presenta un modelo computacional y matemático basado en el complejo funcionamiento del sistema nervioso biológico presente en los seres humanos. En su estructura [14], estas redes se componen de elementos denominados *neuronas artificiales*, que emulan el comportamiento de las biológicas. Están dispuestas en una topología interconectada, permitiéndoles transmitir señales y procesar información para generar una salida específica.

En este contexto de las Redes Neuronales Artificiales, es esencial entender y comprender el modelo que subyace en su funcionamiento. Por ello la **Subsección 3.2.1** presentará las bases de cómo las neuronas artificiales procesan la información y, al final, toman decisiones.

3.2.1. Modelo de McCulloch y Pitts

El modelo de McCulloch y Pitts [15], propuesto por *Warren McCulloch* y *Walter Pitts* en 1943, establece cierta analogía entre el comportamiento de la neurona artificial y la biológica. Con ello, se hizo hincapié en que el poder computacional no reside exclusivamente en el funcionamiento individual de las neuronas, sino en la extensa conectividad que existe entre ellas y en su capacidad de funcionar en paralelo.

Para comprender el modelo propuesto, comenzaremos por comentar de forma general el funcionamiento de una neurona biológica [14]. Acto seguido, analizaremos las similitudes que existen entre este proceso y el modelo propuesto por McCulloch y Pitts.

Como podemos observar en la **Figura 3.1**, las neuronas biológicas están formadas por:

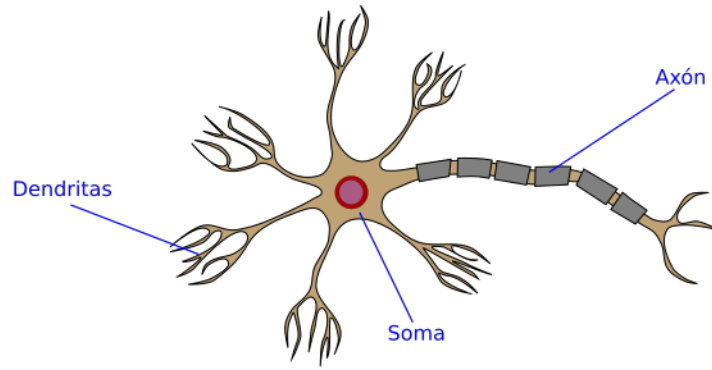


Figura 3.1: Partes de una neurona biológica. Fuente: [16]

- **Dendritas:** extensiones ramificadas compuestas por fibras. Su principal función es recibir señales eléctricas de otras neuronas y transmitirlas desde el exterior hasta el soma.
- **Soma:** contiene el núcleo, recibe las señales eléctricas de las dendritas, las procesa, para decidir en qué medida la neurona debe transmitir dicha señal hacia el axón.
- **Axón:** prolongación larga y delgada. Su función principal es conducir los impulsos eléctricos desde el soma a las neuronas objetivo.
- **Sinapsis:** conexiones especializadas entre el axón de una neurona y las dendritas o cuerpos celulares de otras neuronas.

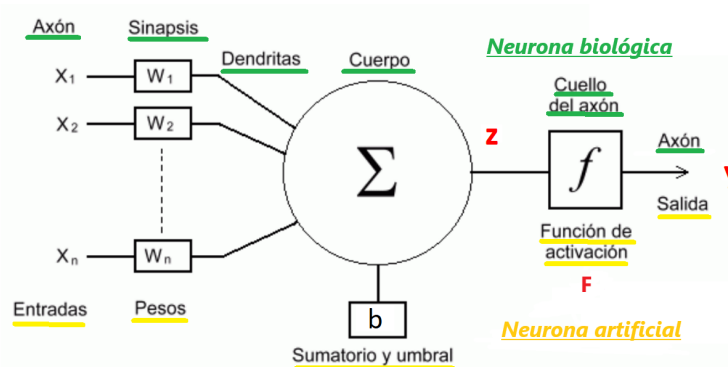


Figura 3.2: Neurona Artificial McCulloch y Pitts. Fuente: [17]

En la **Figura 3.2**, podemos observar la estructura de una Neurona Artificial, según el modelo planteado por McCulloch y Pitts [17]. Estas son algunas de las similitudes con la neurona biológica:

- **Entrada:** representada por un vector, cuyas componentes son $x_1, x_2, \dots, x_n$ que serían captadas por el equivalente a las dendritas.
- **Pesos:** cada entrada tiene un peso asociado $w_1, w_2, \dots, w_n$. Estos se emplean posteriormente para replicar el proceso de la sinapsis.
- **Cuerpo:** viene dado por la suma de todas las sinapsis, obteniéndose lo que se denomina *salida analógica* (z).

$$z = w_0 + \sum_{j=1}^n w_j \cdot x_j \quad (3.1)$$

- **Cuello del axón:** la salida (z), se somete a una transformación mediante la aplicación de la función de activación ($F(z)$), que simula el proceso que ocurre en el extremo del axón. Obtenemos, así, la *salida definitiva* (y).

$$y = F(z) \quad (3.2)$$

En este modelo, se emplea un enfoque binario, en el que tanto las entradas, como las salidas, se representan como valores binarios, es decir, 0 ó 1. Cada entrada tiene asociado un peso, que refleja su influencia en la neurona. El proceso se inicia con la suma ponderada de las entradas multiplicadas por sus respectivos pesos. Esta operación proporciona una medida de la activación neta de la neurona. Posteriormente, se aplica una función de activación, que determina en qué medida la neurona se activa o se inhibe. El funcionamiento es muy sencillo; si la suma ponderada supera el umbral de la función de activación aplicada, emite una señal de salida de activación 1, de lo contrario, emite señal inactiva, 0.

El objetivo central del modelo propuesto por *McCulloch y Pitts*, y del resto de modelos neuronales posteriores, es ajustar los pesos ($w_1, w_2, \dots, w_n$), de manera que la salida de la red se aproxime lo máximo posible a la salida deseada. En este modelo inicial, la información fluye en una única dirección, desde las capas iniciales de la red, hasta las finales, por lo que los pesos de cada neurona se determinan exclusivamente por las salidas generadas por las neuronas previas.

En 1974 [18], *Paul Werbos* presentó en su tesis doctoral la técnica de la retropropagación que marcó un avance significativo en el desarrollo de las Redes Neuronales Artificiales. Esto permitió ajustar los pesos de las neuronas en función del error de salida, haciendo que la red se entrene aprendiendo de sus errores anteriores. Esta técnica llevó a entrenar las Redes Neuronales de forma más eficiente otorgando una mayor capacidad de aprendizaje y adaptación al modelo.

3.2.2. Aprendizaje Profundo

O en inglés *Deep Learning* [19], es una rama del Aprendizaje Automático tal y como se muestra en la siguiente figura.

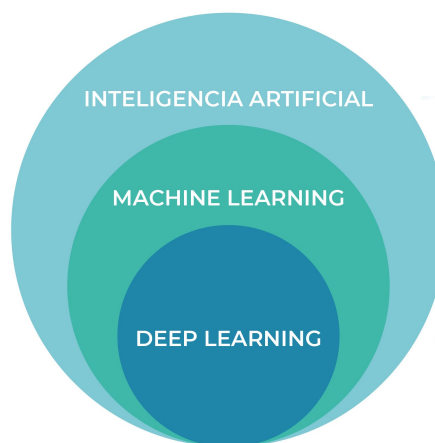


Figura 3.3: Aprendizaje Automático y Profundo subconjuntos de la Inteligencia Artificial. Fuente: [20]

Como puede verse, el Aprendizaje Profundo se centra en el entrenamiento de Redes Neuronales Artificiales de muchas capas, permitiendo que éstas realicen tareas complejas. Se utiliza el término *profundo*, porque las capas operan de manera secuencial lo que, bajo una representación vertical, daría lugar a hablar de profundidad de la red y, por extensión, de cada capa.

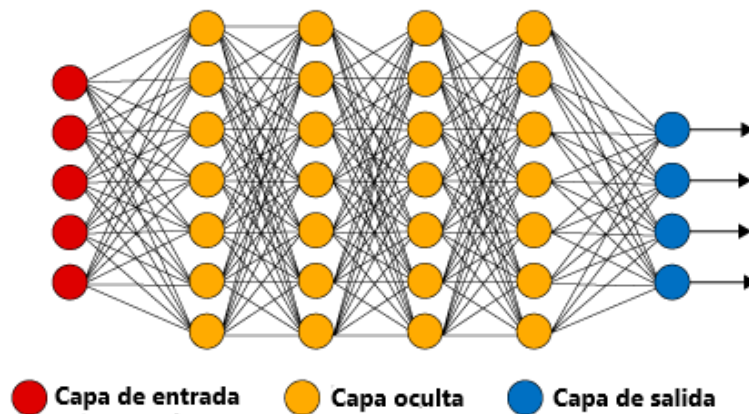


Figura 3.4: Perceptrón Multicapa. Fuente: [21]

La **Figura 3.4** representa un Perceptrón Multicapa [22], también conocido como MLP: siglas en inglés de "Multilayer Perceptron". Está formado por múltiples capas, pero como mínimo debe haber una de entrada, una de salida y una entre ellas denominada oculta. En algunos contextos, el Perceptrón Multicapa se considera como uno de los modelos más simples dentro del ámbito del Aprendizaje Profundo, llegando en ocasiones a no ser considerado como un modelo de Deep Learning, porque este término habitualmente se emplea para referirse a arquitecturas que presentan muchas capas ocultas. Las Redes Neuronales tradicionales contienen dos o tres capas ocultas, sin embargo un modelo de Deep Learning puede tener hasta 150 [23]. Siendo N el número de capas ocultas, a medida que el valor de N aumenta, se incrementa la profundidad de la red y, con ello, el número de conexiones. Eso se traduce en mayor capacidad de representación, y aprendizaje de patrones y características complejas. Por ello, las Redes Neuronales Profundas, tienen un gran potencial en tareas de procesamiento de imágenes o de lenguaje natural.

En las primeras capas de la red, se aprenden elementos simples, como pueden ser bordes o colores. Conforme se avanza hacia las capas más profundas del modelo, éste combina las características simples ya extraídas para detectar patrones más complejos, como pueden ser las partes de un objeto. Así se consigue que, según se avanza hacia las capas cercanas a la salida del modelo, las características detectadas se vuelvan más específicas.

En nuestro caso, el objetivo es identificar diferentes objetos en imágenes, por lo que el aprendizaje de características es esencial. El enfoque que nos ofrece el Deep Learning es la elección más adecuada, gracias a su capacidad para aprender automáticamente representaciones jerárquicas y patrones. En la siguiente sección, investigaremos las Redes Neuronales Convolucionales (CNN), que son un tipo de modelo de Aprendizaje Profundo. Analizaremos cómo operan y examinaremos su relevancia en el contexto del Reconocimiento de Imágenes y el Procesamiento del Lenguaje Natural.

3.3. Redes Neuronales Convolucionales

Conocidas por CNN, su acrónimo en inglés: Convolutional Neural Networks. Están específicamente diseñadas para procesar datos estructurados en forma de matriz o cuadrícula, como imágenes y vídeos. Se caracterizan por la gran efectividad para capturar patrones, debido a la alta capacidad para extraer y aprender características jerárquicas. En consecuencia, las CNN desempeñan tareas fundamentales dentro del campo de la Visión Artificial por computadora, incluyendo reconocimiento de objetos y segmentación de imágenes.

Las Redes Neuronales Convolucionales tienen sus raíces en el *Neocognitron* [24], una innovadora propuesta presentada por *Kunihiko Fukushima* en 1980 [25]. El *Neocognitron* es una red jerárquica multicapa, que presenta conexiones variables entre las células de capas contiguas, que fue construido con el objetivo de reconocer números manuscritos en japonés. Este modelo inicial sentó las bases en la detección de patrones invariantes en el espacio, lo que ha sido fundamental para el posterior desarrollo de las CNN, hasta como las conocemos hoy en día.

En 1998 *Yann André LeCun* [26] fue el encargado de mejorar significativamente el modelo inicial, para ello introdujo un enfoque de aprendizaje, que se fundamenta en la retropropagación, lo que permitía un entrenamiento mucho más eficaz de la red. Desde aquel momento, las Redes Neuronales Convolucionales han experimentado una serie de avances significativos. Uno de los más destacados fue llevado a cabo por *Dan Criesan* y su equipo, quienes lograron optimizar estas redes para su ejecución en unidades de procesamiento gráfico (GPU). Esta adaptación

produjo mejoras notables, tanto en términos de resultados, como de eficiencia en el procesamiento de datos.

El modelo inicial propuesto, junto con las mejoras continuas que se han aplicado a lo largo del tiempo, ha elevado significativamente la importancia y la eficacia de las CNN, lo que las ha llevado a ser una alternativa sumamente valiosa y eficaz en el ámbito de la Visión por Computadora.

3.3.1. Visión artificial

Actualmente [27], representa una disciplina de la Inteligencia Artificial, cuyo enfoque radica en el análisis, procesamiento y extracción automatizada de información a partir de imágenes, con el propósito de generar datos que puedan ser interpretados y manipulados por una computadora. Para lograr entrenar una máquina, para que ésta sea capaz de interpretar la información por sí sola, necesitamos una gran cantidad de datos, los suficientes, para que sea capaz de aprender las diferencias y parecidos entre los diferentes patrones.

Para conseguirlo, se emplean dos tecnologías que están estrechamente relacionadas: el Deep Learning y una Red Neuronal Convolutiva. Ambas permiten analizar y procesar imágenes de forma automática. Para ello, se aplican técnicas de preprocesamiento de imágenes con el fin de resaltar ciertos aspectos específicos, consiguiendo así hacerla más adecuada para la extracción de información. La última fase se enfoca en la extracción del conocimiento de los datos, para que pueda ser utilizado en diversos campos y aplicaciones tan variadas como la conducción automática, la orientación de robots y la exploración automática de áreas.

Con el fin de entender el funcionamiento completo de las Redes Neuronales Convolutivas, así como su relación con la Visión Artificial y sus objetivos, tenemos que explorar tres conceptos fundamentales [28]: la convolución, los métodos de *downsampling* y *upsampling*, que se explican en las **Subsecciones 3.3.2, 3.3.4 y 3.3.5**, respectivamente.

3.3.2. Convolución

Es una operación lineal matemática [29] entre dos funciones, que puede ser vista como que una de ellas es un filtro de lectura de la otra.

Las operaciones de convolución aplicadas en las Redes Neuronales Convolutivas presentan una serie de ventajas frente a la multiplicación habitual de matrices, que se realiza en las Redes Neuronales tradicionales [30]:

- **Conectividad dispersa:** En las Redes Neuronales tradicionales, cada neurona tiene como entrada, la salida de cada una de las neuronas de la capa anterior, lo que se conoce como *conectividad densa*. Esto implica una gran cantidad de parámetros y coste computacional. Sin embargo, la convolución opera de manera localizada, ya que emplea un kernel de menor tamaño, que se aplica repetidamente a través de la imagen. Gracias a esto, se consigue una reducción considerable del número de parámetros y una mejora significativa en la eficiencia del modelo.

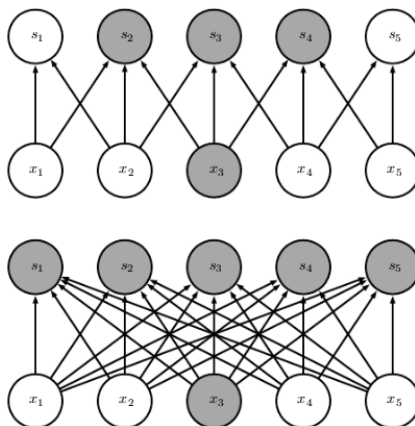


Figura 3.5: Conectividad dispersa vs conectividad densa. Fuente: [30]

- **Compartición de parámetros:** Las Redes Neuronales Convolucionales, utilizan los mismos parámetros (pesos) para diferentes partes de la imagen, es decir, en cada operación de convolución. Esto no sucede en las redes tradicionales, donde cada peso se emplea una única vez. Con esto se consigue reducir la cantidad de parámetros y desarrollar la capacidad de generalización, a la vez que podría identificar una misma característica en distintas ubicaciones de la imagen.

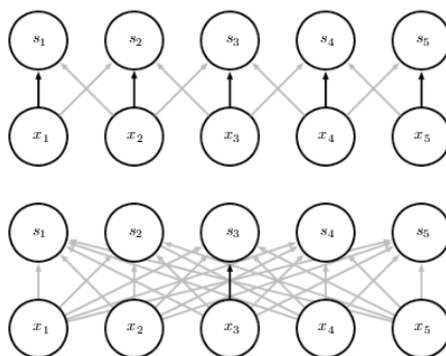


Figura 3.6: Compartición de parámetros. Fuente: [30]

En la **Figura 3.6**, podemos ver la compartición de parámetros en ambos modelos, con conectividad densa y dispersa. La flecha negra representa las conexiones que utilizan un parámetro concreto. Como podemos ver en la imagen, el modelo de conectividad dispersa (CNN), gracias a la compartición de parámetros que utiliza, se usa en todas las ubicaciones de entrada. Por su parte, en el modelo completamente conectado, el parámetro se utiliza sólo una vez, ya que no presenta compartición.

- **Representaciones equivalentes.** Esta propiedad, conocida como *Equivarianza a la Traslación*, es fundamental en las Redes Neuronales Convolucionales (CNN). Permite reconocer patrones en una imagen independientemente de su ubicación. En otras palabras, si la imagen de entrada sufre una transformación de traslación, la capa de salida experimentará la misma transformación, lo que equivale a que la característica no haya sufrido ningún desplazamiento. Esta propiedad es aplicable en el caso de transformaciones de traslación debido a la equivarianza en las operaciones de convolución utilizadas en las CNN. Sin embargo, es importante destacar, que esta propiedad no se aplica de manera similar a otras transformaciones, como el escalado o la rotación.

Una vez que hemos establecido las ventajas de la convolución, en comparación con la multiplicación tradicional de matrices en un modelo de conexión densa, es pertinente examinar cómo se lleva a cabo y en qué consiste.

La fórmula matemática para la convolución de dos funciones es:

$$R(x) = \int_{-\infty}^{\infty} I(z) \cdot K(x - z) dz \quad (3.3)$$

donde:

- **I**: son las imágenes que componen los datos de entrada. Un array normalmente multidimensional.
- **K**: es el kernel que se aplica sobre I. De nuevo un array multidimensional y normalmente de dimensiones inferiores a I.
- **R**: es el resultado obtenido tras la operación de convolución, que genera una nueva función.

En la **Fórmula 3.3** vemos la expresión en la forma continua, pero también podemos utilizar su forma discreta, que la emplearemos en el resto de la sección:

$$R(x) = \sum_{m=-\infty}^{\infty} I(m) \cdot K(x - m) \quad (3.4)$$

En nuestro caso, disponemos de imágenes como entrada, por lo que cada imagen de I es una matriz bidimensional sobre la que se aplicará la convolución. Para poder efectuar la convolución, disponemos de, un kernel (K), que de nuevo es otra matriz bidimensional. Aplicando la fórmula de la convolución discreta en nuestro input y kernel bidimensionales, obtenemos como resultado otra matriz de dos dimensiones R [31]:

$$R(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n) \quad (3.5)$$

Generalmente, el tamaño del kernel es más pequeño que el de la imagen. Se suele aplicar la propiedad conmutativa en la fórmula (3.6) porque se ve con más facilidad el número reducido de combinaciones válidas para los índices m y n . Lo que disminuye la cantidad de cálculos requeridos.

$$R(i, j) = \sum_m \sum_n I(i - m, j - n) \cdot K(m, n) \quad (3.6)$$

En la fórmula (3.7), se aplica la técnica de correlación cruzada, más sencilla de calcular en comparación con la convolución, ya que no requiere voltear el kernel para realizar los cálculos. Aunque la correlación cruzada y la convolución no son idénticas, en el contexto práctico de las Redes Neuronales Convolucionales, sus comportamientos son muy similares. Asimismo, es importante destacar que en muchas bibliotecas, el término *convolución* se utiliza de manera genérica, incluso cuando se está realizando una correlación cruzada.

$$R(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n) \quad (3.7)$$

Las **Figuras 3.7, 3.8 y 3.9** proporcionan un ejemplo numérico, que facilita la comprensión del proceso. El kernel actúa como una ventana móvil, recorriendo la matriz bidimensional de la imagen de arriba a abajo y de izquierda a derecha. En cada iteración, se multiplica el valor de la celda del kernel por el valor del píxel asociado de la matriz de la imagen y, después, se suman todos. Como resultado de esta operación, obtenemos la matriz o tensor R.

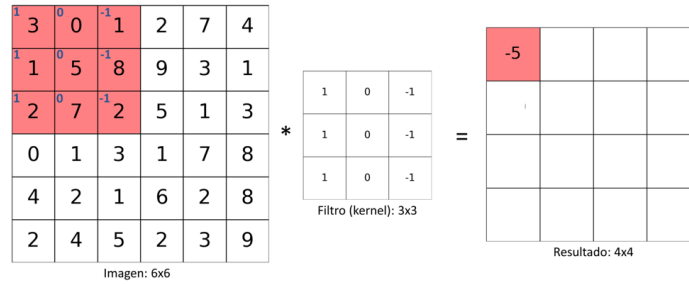


Figura 3.7: Primera iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]

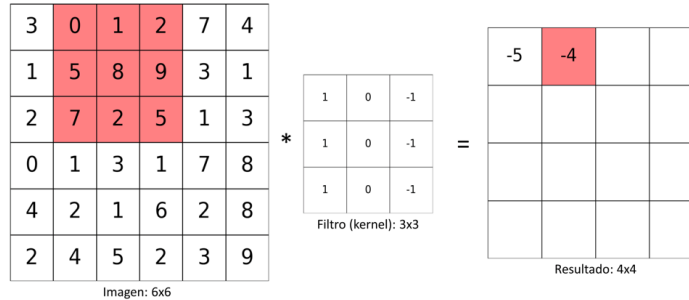


Figura 3.8: Segunda iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]

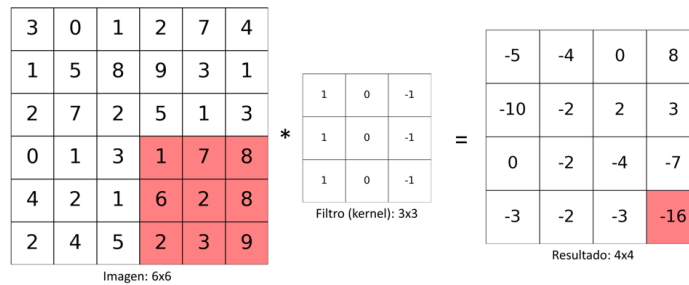


Figura 3.9: Última iteración en una convolución 2D, aplicada a un ejemplo numérico. Fuente: [32]

En el ejemplo expuesto anteriormente, se emplea una matriz de imagen de 6x6 como punto de partida y se aplica un filtro de dimensiones 3x3. El resultado de esta operación es un tensor resultante o *Mapa de Características* con dimensiones reducidas a 4x4. Teniendo en cuenta las dimensiones de este ejemplo, llegamos a la conclusión de que para una entrada de dimensiones $m \cdot n$ y un kernel o filtro de dimensiones $u \cdot v$, obtenemos un tensor de dimensiones $m-u+1$ (filas) x $n-v+1$ (columnas).

El resultado, tras la operación de convolución, se denomina *Mapa de Características* o en inglés *Feature Map*. Al aplicar el filtro a una imagen, se genera otra y en función del kernel aplicado, se verán resaltados unos aspectos u otros, relacionado con ciertas características específicas de la imagen original. En el ejemplo de la **Figura 3.7**, el filtro aplicado es uno de los conocidos como *filtros de Sobel* ($[1,0,-1], [1,0,-1], [1,0,-1]$), que se emplea para la detección de bordes verticales. Por otro lado, existe la versión horizontal de este filtro ($[1,1,1], [0,0,0], [-1,-1,-1]$), que nos permite identificar bordes y transiciones horizontales. En el contexto del presente TFG, es importante destacar que, más allá de los ampliamente difundidos y utilizados filtros de Sobel, existe una diversidad prácticamente infinita de filtros, cada uno de los cuales se aplica con el propósito específico de identificar una característica particular.

El desplazamiento del kernel en el proceso de convolución no es una constante, varía dependiendo de dos parámetros esenciales: el *padding* y el *stride*. Estos dos factores juegan un papel crucial en la forma en que el filtro se desplaza a lo largo de la imagen de entrada, lo que determinará el tamaño del tensor resultante. En la **Sección 3.3.3** se detallan qué son y cómo afectan estos parámetros a la convolución.

Estas Redes Neuronales normalmente realizan múltiples convoluciones en cascada, permitiendo cada vez extraer características de complejidad creciente. Los Mapas de Características generados por las capas iniciales de la red

permiten identificar patrones o características simples, como líneas rectas o diagonales. Sin embargo, a medida que avanzamos hacia las capas profundas de la red, vemos como se reconocen patrones y características más complejas. Este concepto se ilustra en la **Figura 3.10**, donde la imagen de entrada es la cara de una persona y se pueden ver las distintas características extraídas según los distintos niveles de abstracción. En los más bajos se aprecian líneas horizontales, verticales y diagonales principalmente. En el nivel medio de abstracción, ya podemos identificar partes de la cara como los ojos, boca y nariz entre otros. Por último, en las capas más profundas de la red, podemos observar que ya se identifican caras completas, combinando los patrones y características identificadas por las capas previas.

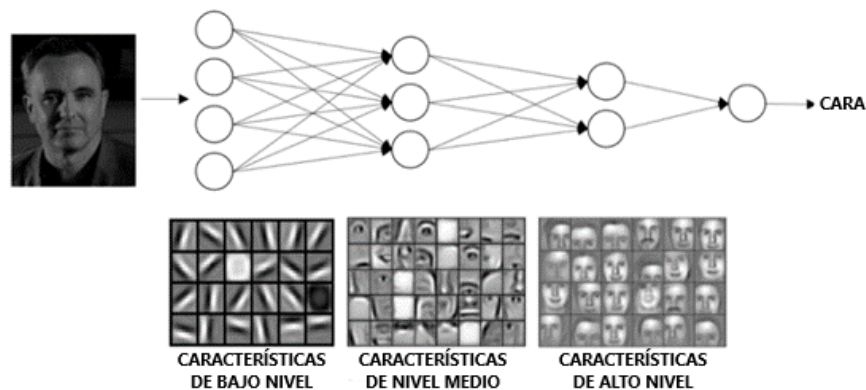


Figura 3.10: Extracción de características. Fuente: [33]

Convolución de imágenes a color (RGB)

En el contexto previamente presentado, hemos abordado la representación de imágenes como matrices de píxeles, que contienen un único canal de información, es decir, imágenes en escala de grises. Sin embargo, en la realidad y en el marco de nuestro proyecto, nos enfrentaremos al procesamiento de imágenes a color, las cuales incorporan tres canales de información, reciben el nombre de imágenes RGB.



Figura 3.11: Ejemplo de imagen RGB y los tres canales de información. Fuente: [34]

En la **Figura 3.11** vemos una imagen RGB [35], ésta presenta para cada píxel, tres canales de información: Rojo (R), Verde (G) y Azul (B). De esta forma cada píxel se representa como una combinación de estos tres colores y se almacena como un tensor de $m \cdot n$ con tres dimensiones para representar el color.

Aunque, la operación de convolución es esencialmente la misma, en el caso de imágenes RGB se realiza una convolución independiente para cada canal de entrada. Por último, las salidas obtenidas de cada canal se suman para crear un único mapa de características combinado.

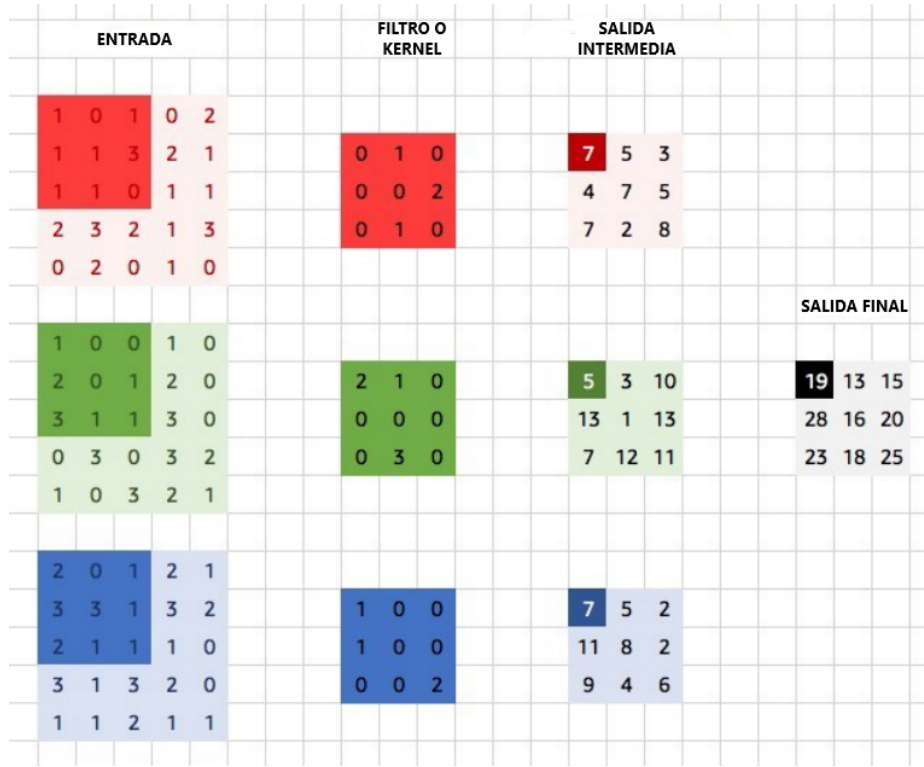


Figura 3.12: Convolución imagen RGB Fuente: [36]

En la **Figura 3.12** vemos que ahora existen tres filtros o kernels, uno para canal de información. Por tanto, cada uno de ellos puede presentar valores diferentes. Para cada canal se genera un *mapa de características intermedio*, o salida intermedia. La respuesta final se obtiene al realizar la suma de cada una de las salidas intermedias.

Para adecuarnos al modelo con imágenes RGB, tenemos que modificar también la formulación anteriormente propuesta. Ahora se añade un nuevo parámetro c , que representa el canal de información, lo que podría verse como un filtro de una dimensión más.

$$R_{c,i,j} = \sum_{m=1} \sum_{n=1} I_{c,i+m-1,j+n-1} \cdot K_{c,m,n} \quad (3.8)$$

donde:

- **I**: son las imágenes de entrada, $I_{c,i,j}$, (i,j) son los elementos del canal de información c .
- **K**: es el tensor tridimensional que representa el kernel. $K_{c,m,n}$ (m,n) se emplea para localizar el valor dentro del canal c .
- **R**: es cada una de las salidas intermedias y presenta el mismo formato que la entrada I .

3.3.3. Desplazamiento del kernel

El desplazamiento del kernel en una operación de convolución no es una constante, sino que depende de dos términos fundamentales: el stride y el padding. A pesar de esta variabilidad, el kernel sigue el patrón de recorrer la imagen por completo, de arriba a abajo y de izquierda a derecha. Aunque el objetivo y el funcionamiento es el mismo, estos términos tienen una influencia importante, ya que modifican como se lleva a cabo este desplazamiento, y afecta a la extracción de características.

Stride

El *stride*, o salto [37], es un parámetro del kernel de la Red Neuronal que representa la magnitud del avance del núcleo al recorrer la imagen. En todos los ejemplos de convoluciones expuestos hasta ahora, el kernel se ha desplazado un píxel hacia la derecha o hacia abajo en cada una de las iteraciones. En estos casos, el valor del stride utilizado es igual a 1, como en la **Figura 3.13**.

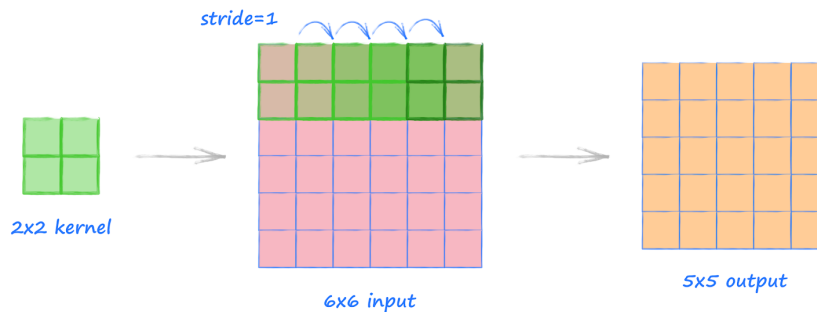


Figura 3.13: Convolución con parámetro stride de valor 1. Fuente: [38]

Este valor puede ajustarse, permitiendo que la operación de convolución pueda llevarse a cabo con valores de stride mayores, lo que produce imágenes de salida de menor tamaño. La reducción de esta dimensión es útil, ya que disminuye significativamente la cantidad de datos que se procesan entre capas sucesivas de la red, produciendo mayor eficiencia computacional. Igualmente, puede facilitar el aprendizaje y la identificación de patrones en conjuntos de datos más compactos, lo que resulta en un mejor rendimiento del modelo. En la **Figura 3.14**, podemos ver una operación de convolución con valor de stride igual a 2.

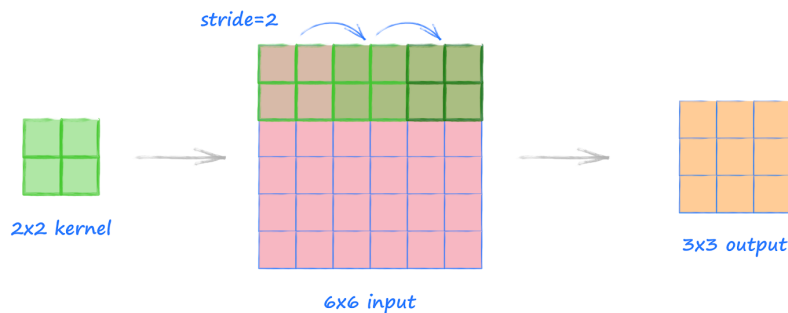


Figura 3.14: Convolución con parámetro stride de valor 2. Fuente: [38]

Padding

El *padding* es una técnica fundamental en las operaciones de convolución. Consiste en añadir píxeles transparentes o de valor cero en la imagen de entrada original antes de aplicar la convolución, para que se ajuste a las especificaciones del modelo.

Existen cuatro tipos principales de padding [39]:

- **Same padding:** es el tipo más comúnmente utilizado. Su propósito principal es asegurar que la imagen resultante de la convolución tenga el mismo tamaño que la imagen de entrada, añadiendo los píxeles en los bordes de la imagen original. Permite extraer características más específicas de las imágenes y mejora la capacidad de la red para reconocer patrones complejos. Aunque no se utiliza en todas las capas de la red, su aplicación estratégica en ciertas capas es fundamental para mantener el tamaño de la imagen entre ellas.
- **Valid padding:** esta variante no realiza padding alguno. El kernel se aplica únicamente en las zonas donde el kernel y la imagen se superponen totalmente. Esto implica que la imagen resultante es de menor tamaño que la imagen de entrada, ya que se busca reducir las dimensiones de la misma a medida que se realizan las convoluciones.

- **Full padding:** es una técnica de padding más avanzada, se añaden tantas filas y columnas alrededor de la imagen de entrada como sean necesarias, para que el filtro puede desplazarse completamente por toda la imagen. De esta forma, evitamos que se produzca pérdida de información en los bordes.
- **Causal padding:** este padding añade elementos al inicio sobre los datos, facilitando la predicción de valores al inicio. Adquiere importancia en aplicaciones donde se prevén eventos futuros, basándose en información pasada únicamente, como por ejemplo, en tareas de procesamiento de Lenguaje Natural.

A continuación, ilustraremos el concepto de same padding, puesto que es el tipo más utilizado:

Si realizamos una operación de convolución sin padding, tomando una imagen de dimensiones $n \cdot n$, sobre la que se aplica un kernel de dimensión $k \cdot k$, el mapa de características resultante tendría unas dimensiones $n - k + 1 \cdot n - k + 1$. Sin embargo, tras aplicar same padding de anchura p las dimensiones del mapa resultado son $n - k + 2p + 1 \cdot n - k + 2p + 1$.

En ejemplo de la **Figura 3.15**, podemos ver una imagen de entrada de tamaño $n = 6$, el kernel aplicado de tamaño $k = 3$ y el padding tamaño $p = 1$, ya que se añade una única fila de nuevos píxeles. Sustituyendo en la formula anterior $6 - 3 + 2 \times 1 + 1 = 6$, obtenemos $n=6$, por lo que el mapa de características resultante mantiene las dimensiones de la entrada.

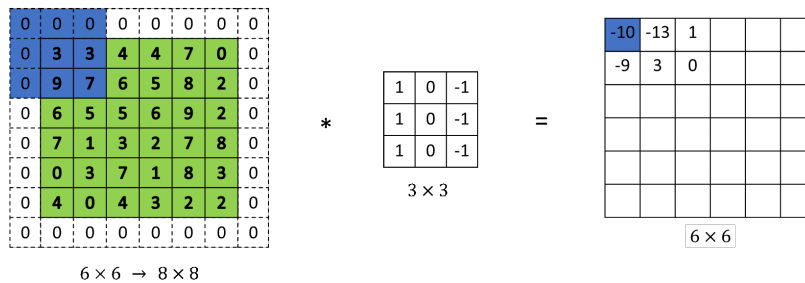


Figura 3.15: Convolución tras aplicar *same padding*. Fuente: [40]

Además de la operación de convolución, existen otras dos técnicas que se emplean en las Redes Neuronales Convolucionales: el *downsampling* o submuestreo de características, y el *upsampling*.

3.3.4. Downsampling

El *downsampling* es una técnica esencial en el ámbito de la Visión por Computadora y el procesamiento de imágenes. Su propósito fundamental es la reducción de la resolución en imágenes para obtener una representación más compacta. En la **Sección 3.4**, veremos cómo en el contexto de las UNET, el downsampling se emplea en la fase de contracción o extracción de características, permitiendo la detección de características clave en una representación de imagen más reducida.

Existen diversas técnicas para llevar a cabo este proceso:

1. **Pooling.** Aplica un filtro sobre la imagen original, seleccionando un valor en función del tipo de pooling aplicado. Dado que es una de las técnicas más comunes y ampliamente utilizadas en el contexto de las Redes Neuronales Convolucionales, considero oportuno dedicar el resto de la sección a explorar en detalle su objetivo, sus diversos tipos y las ventajas inherentes a la aplicación de esta técnica en el marco de nuestro modelo.
2. **Stride Convolution.** Se aplican filtros mediante intervalos regulares, omitiendo algunos de los píxeles en el proceso.
3. **Submuestreo bilineal y bicúbico.** utilizan un algoritmo bilineal o bicúbico, respectivamente, y calculan un valor interpolado para los píxeles de la imagen resultante.

Pooling

Operación fundamental que se emplea para reducir las dimensiones de los mapas de características. Esta técnica implica analizar una imagen por regiones, lo que permite extraer la información más representativa de la misma.

Se introducen capas de pooling o agrupación entre las convolucionales, para conseguir compactar los mapas de características generados.

El funcionamiento del pooling es muy similar a la convolución de las Redes Neuronales Convolucionales (CNN). Utiliza un filtro que se desplaza como una ventana móvil a lo largo del mapa de características de la salida de la capa convolucional anterior. El proceso concluye cuando se ha recorrido todo el mapa. En general, se emplea un filtro de tamaño 2×2 y se desliza utilizando un stride de dos, lo que significa que se desplaza dos píxeles a la derecha o hacia abajo en cada iteración. En un ejemplo sencillo, si tenemos un mapa de características de 4×4 y aplicamos pooling con un filtro 2×2 , la salida resultante tendrá la mitad de tamaño en ambas dimensiones, generando así un mapa de características resultante de 2×2 .

Existen cuatro tipos principales de pooling [41], cada uno de estos tipos tiene sus propias aplicaciones y ventajas en la extracción de características relevantes de las imágenes.

- **Max Pooling.** Resume las características en una región representada por su valor máximo de esa región. Resulta adecuado para imágenes que presentan un fondo oscuro, ya que selecciona los píxeles más brillantes de cada región.

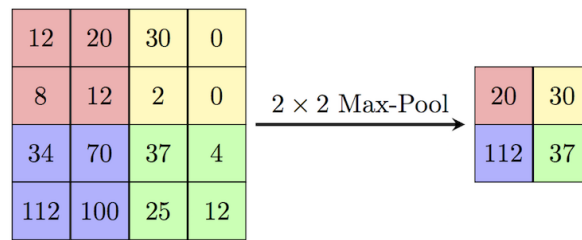


Figura 3.16: Max Pooling con filtro 2×2 . Fuente: [42]

- **Min Pooling.** Compacta las características de una región representada por su valor mínimo en esa región. Al contrario que el anterior, se utiliza en imágenes que presentan un fondo claro, ya que selecciona los píxeles más oscuros.

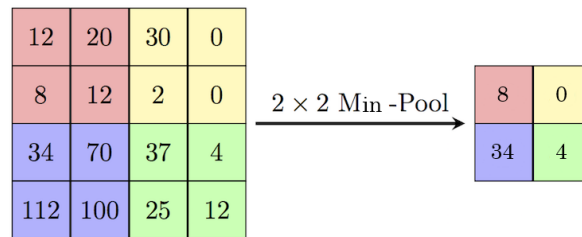


Figura 3.17: Min Pooling con filtro 2×2 .

- **Average Pooling.** Se sustituyen las características en una región representada por su promedio. Con esta técnica, los bordes radicales de una imagen se suavizan.

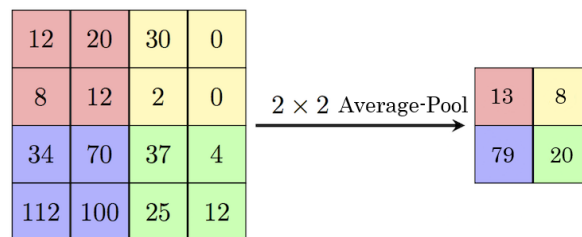


Figura 3.18: Average Pooling con filtro 2×2 .

- **Global Pooling.** Esta técnica reduce el mapa de características a un único valor. Para conseguirlo, las dimensiones del filtro son las mismas que las del mapa.

Independiente del tipo de pool aplicado al modelo, añadir capas de pooling o agrupación al modelo entre las capas convolucionales, trae consigo una serie de ventajas:

- **Reducción de la dimensionalidad:** el pooling reduce el tamaño de los mapas de características y, por consiguiente, también disminuye el número de parámetros del modelo. Así conseguimos reducir la complejidad computacional, aceleramos el entrenamiento y evitamos que se produzca sobreajuste, lo que implica un mejor rendimiento ante conjuntos de datos de prueba no conocidos. Todo esto tiene un límite, puesto que esta reducción conlleva una pérdida de conocimiento/información.
- **Invarianza ante pequeñas traslaciones:** dota a la Red Neuronal de capacidad para identificar características de una imagen independientemente de su localización en la imagen de entrada. Si un objeto se desplaza levemente a otro punto de la imagen, la red sigue teniendo capacidad de detectarlo, ya que la principal característica de éste se encuentra en la misma región.
- **Menor complejidad:** permite que la red sea más eficiente, si hablamos de consumo de memoria y recursos de cómputo, debido a que simplifica la representación de las características.
- **Mayor abstracción de características:** resume la información en regiones más pequeñas, descartando así los detalles innecesarios y centrándose en las características de interés. La red desarrolla una mejor capacidad de generalización y la dotamos de robustez ante variaciones en los datos de entrada

Es importante destacar que la elección de la técnica de downsampling, puede variar según el contexto de la aplicación específica que se pretende desarrollar y las características de las imágenes de entrada. El tipo de pooling más extendido en los modelos convolucionales es el Max Pooling. Proporciona resultados generalmente buenos en la mayoría de los casos, y es el utilizado en la arquitectura original propuesta de la UNET. En última instancia, la elección de la técnica adecuada dependerá de la naturaleza de los datos y los objetivos de procesamiento específicos de cada aplicación.

3.3.5. Upsampling

El proceso de upsampling presenta como objetivo principal reorganizar la información previamente compactada a través de las características detectadas en la fase de extracción anterior (downsampling). De esta manera, el upsampling posibilita la reconstrucción de una imagen segmentada, en la que se identifican con claridad cada una de las categorías o clases que se pretenden reconocer. Este proceso es esencial para obtener resultados precisos y detallados en tareas de segmentación de imágenes.

Existen diversas técnicas para realizar este proceso:

■ Unpooling.

El objetivo es realizar la labor opuesta al pooling, es decir, trata de devolver el tamaño original al mapa de características que ha recibido como entrada. Se puede realizar de tres formas diferentes, y para explicarlo supondremos que el mapa de características original presenta unas dimensiones de 4×4 , y que tras aplicar la operación de pooling estas se ven reducidas a la mitad, 2×2 . Existen diversos métodos para realizar el unpooling [43].

Nearest Neighbor.

Es el método más sencillo para realizar el unpooling: copia el valor de un píxel del mapa de características que recibe como entrada, a todos los píxeles de la región correspondiente del mapa de salida. El fundamento de este sistema es replicar el valor en una región más grande aumentando, así, la resolución de la imagen.

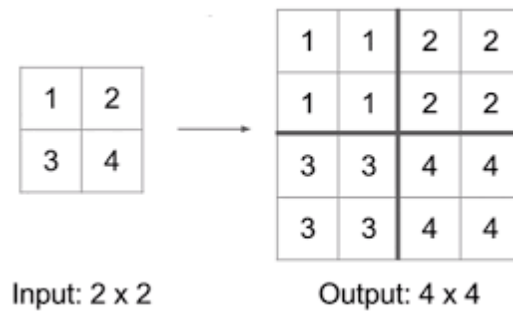


Figura 3.19: Unpooling: Método Nearest Neighbor. Fuente: [43]

Este enfoque debido a su simpleza presenta un problema: el mapa de características de salida se asemeja a una subdivisión en bloques, ya que todos píxeles de una región tienen el mismo valor.

Bed of Nails.

Este método coloca, en la esquina superior izquierda de la región, el valor correspondiente del píxel del mapa de características que recibe como entrada; completa el resto de los elementos de la región con ceros.

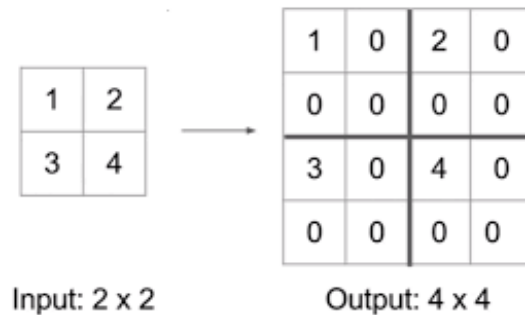


Figura 3.20: Unpooling: Método Bed of Nails.

Al aplicar este enfoque obtenemos una estructura de salida detallada, pero los elementos muestreados se encuentran siempre en la misma ubicación, la esquina superior izquierda del mapa de características obtenido como salida.

Max Unpooling.

Este método soluciona el problema presentado por el método *Bed of Nails*. Realiza el proceso de upooling de forma similar, pero tiene la capacidad de recordar de dónde provienen los valores de los píxeles más altos antes de que se efectuase el *Max Pooling*. De esta forma, cuando realiza esta operación se van colocando los valores en las posiciones de cada subregión donde estaban ubicados.

Debido a esta capacidad, se realiza una reconstrucción más precisa de la información.

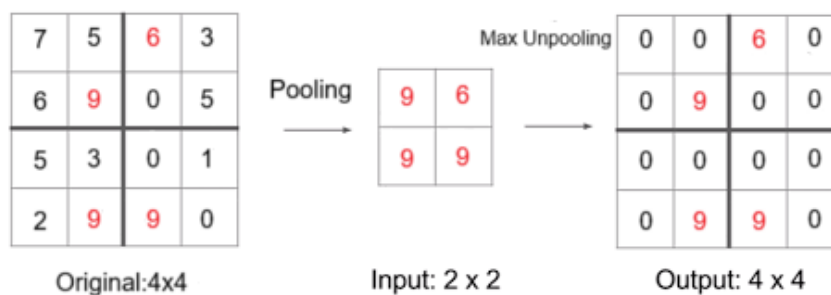


Figura 3.21: Unpooling: Método Max Unpooling. Fuente: [44]

■ Convolución transpuesta.

De nuevo, el objetivo principal es aumentar las dimensiones del mapa de características que se recibe como entrada [45]. Al igual la convolución normal, la transpuesta se basa en dos parámetros clave: el stride y el padding. En la **Figura 3.22** vemos una comparación de la salida obtenida en la convolución normal y la transpuesta.

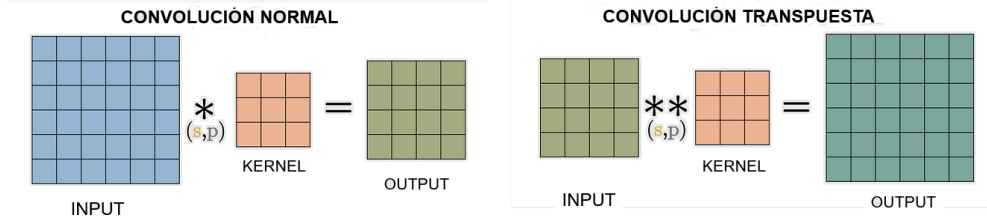


Figura 3.22: Convolución normal vs transpuesta. Fuente: [45]

Para llevar a cabo la implementación de la convolución transpuesta, es importante tener en cuenta varios parámetros clave. En primer lugar, consideramos el tamaño de la imagen de entrada, de dimensiones $i \times i$, así como el tamaño del kernel, de tamaño $k \times k$. Los parámetros fundamentales de stride y padding representados por s y p respectivamente, desempeñan un papel fundamental en este proceso. Con estos elementos definidos, podemos abordar la implementación de la convolución transpuesta en cuatro pasos fundamentales.

1. Calculamos los parámetros z y p' , aplicando las siguientes fórmulas:

$$z = s - 1 \quad (3.9)$$

$$p' = k - p - 1 \quad (3.10)$$

2. Entre cada fila y columna de la imagen de entrada, añadir tantos ceros como valor de z . Así conseguimos aumentar las dimensiones de la entrada a $(2 * i - 1) \times (2 * i - 1)$.
3. Añadimos en los bordes de la imagen modificada tantas capas de ceros como el valor de p' .
4. Aplicamos una convolución normal a la imagen que obtenemos tras la ejecución del paso 3 con un filtro de stride de 1.

En la siguiente figura podemos ver la combinación de los cuatro pasos en orden y cómo afectan estas transformaciones a la entrada. Asimismo, en la referencia [46] hay una serie de gifs que ayudan a comprender el funcionamiento de la convolución transpuesta.

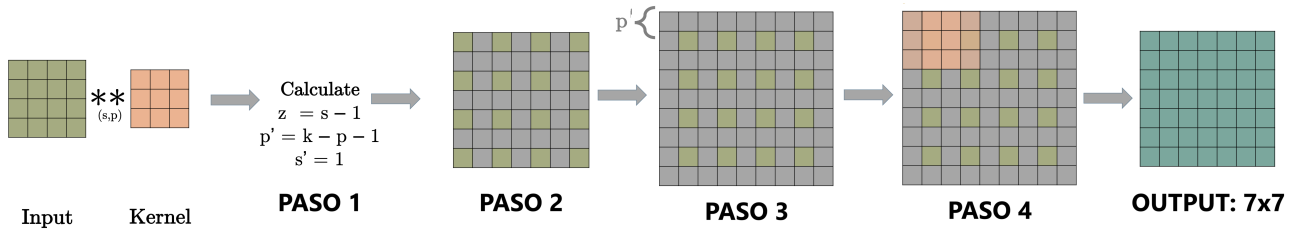


Figura 3.23: Ejemplo aplicación convolución transpuesta.

■ Interpolación bilineal.

Originalmente desarrollada para su uso en el ámbito de las matemáticas [47], también ha adquirido importancia en el campo del procesamiento de imágenes y la visión por computadora.

Realizando este aumento de dimensiones existen agujeros, píxeles vacíos o sin valores, debido a que no se les puede asignar una intensidad y colores adecuados. Dichos píxeles necesitan tener valores RGB o escala de grises según el caso. Para ello, empleamos la interpolación bilineal. Para cada uno de estos píxeles vacíos se toman los cuatro píxeles conocidos más próximos que conforman un rectángulo. A partir de ellos, mediante la interpolación se halla un valor adecuado para el píxel que estamos interpolando.

El cálculo del nuevo valor para el píxel vacío se puede resumir en dos pasos:

1. Hallamos la media ponderada de los valores que toman píxeles de las esquinas superiores e inferiores del rectángulo. Para ello, tenemos en cuenta la posición del píxel que estamos interpolando respecto a la posición de los píxeles conocidos.
2. Calculamos la media ponderada de los resultados obtenidos en el paso anterior, teniendo en cuenta la posición del píxel de referencia.

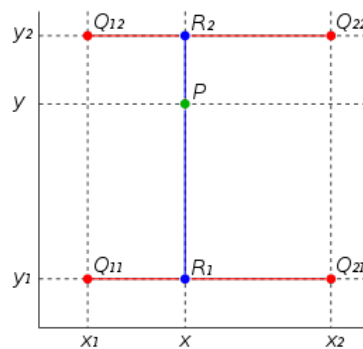


Figura 3.24: Interpolación bilineal.

En la imagen anterior podemos ver los píxeles que conforman el rectángulo de color rojo y, en verde, el píxel que queremos interpolar.

Con el fin de ilustrar los pasos anteriores y facilitar la comprensión de la **Figura 3.24**, en la siguiente imagen se expone un ejemplo:

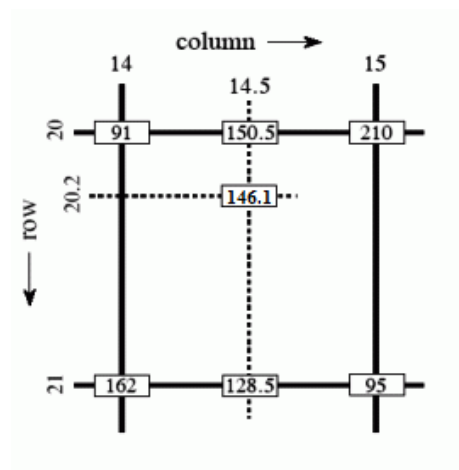


Figura 3.25: Ejemplo interpolación bilineal en escala de grises.

En esta figura vemos los píxeles conocidos que conforman el rectángulo, los cuales se encuentran ubicados en las coordenadas (20,14), (20,15), (21,14), (21,15). La primera componente hace referencia a la fila y la segunda a la columna. Se trata de un ejemplo en escala de grises por lo que presentan un único valor, 91, 210, 162

y 95 respectivamente. El píxel que queremos interpolar está situado en (20.2,14.5). Con toda esta información:

PASO 1

Aplicamos las fórmulas de la interpolación lineal en la dirección X:

$$f_{(x,y_1)} = \frac{x_2 - x}{x_2 - x_1} \cdot f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} \cdot f(Q_{21}) \quad (3.11)$$

$$f_{(x,y_2)} = \frac{x_2 - x}{x_2 - x_1} \cdot f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} \cdot f(Q_{22}) \quad (3.12)$$

Sustituyendo por los valores adecuados según las esquinas del rectángulo obtenemos:

Píxeles de las esquinas inferiores

$$I_{(21,14'5)} = \frac{15 - 14,5}{15 - 14} \cdot 162 + \frac{14,5 - 14}{15 - 14} \cdot 95 = 128,5 \quad (3.13)$$

Píxeles de las esquinas superiores

$$I_{(20,14'5)} = \frac{15 - 14,5}{15 - 14} \cdot 91 + \frac{14,5 - 14}{15 - 14} \cdot 210 = 150,5 \quad (3.14)$$

PASO 2

Con los valores $I_{(20,14'5)}$ y $I_{(21,14'5)}$ anteriormente calculados ahora podemos interpolar en la dirección Y:

$$f_{(x,y)} = \frac{y_2 - y}{y_2 - y_1} \cdot f(x, y_1) + \frac{y - y_1}{y_2 - y_1} \cdot f(x, y_2) \quad (3.15)$$

En la que sustituyendo por los valores concretos de nuestro problema tenemos:

$$f_{(20'2,14'5)} = \frac{21 - 20,2}{21 - 20} \cdot 150,5 + \frac{20,2 - 20}{21 - 20} \cdot 128,5 = 146,1 \quad (3.16)$$

Tras aplicar la interpolación bilineal al píxel de ubicación (20.2, 14.5), logramos obtener un valor en la escala de grises de 146.1. De esta manera, hemos conseguido asignar un valor adecuado a un píxel, cuyo valor era previamente desconocido, partiendo del conocimiento previo sobre el valor de píxeles circundantes. Existen otras técnicas relacionadas con la interpolación, como la bicúbica, trilineal o spline, que también se utilizan con el mismo objetivo de rellenar los huecos en datos de imágenes.

A lo largo de la **Sección 3.3**, hemos explorado conceptos y términos fundamentales sobre las Redes Neuronales Convolucionales y su eficiencia en tareas de Visión por Computadora, detección de objetos, segmentación y clasificación de imágenes. Sin embargo, para abordar con éxito un problema o un proyecto concreto, necesitamos definir una arquitectura de Red Neuronal Convolutiva específica, que permita adaptarnos a nuestros objetivos y necesidades. Para el desarrollo de este proyecto, se aplicará una arquitectura conocida como UNET. Este modelo destaca en aplicaciones donde se desempeñan tareas de segmentación de imágenes, en las que se necesita una alta precisión en la delimitación de objetos y regiones de interés dentro de estas imágenes.

En la siguiente sección, profundizaremos en esta arquitectura y conoceremos su funcionamiento en detalle.

3.4. Arquitectura de una UNET

El modelo de Red Convolutiva conocido como UNET [48], es una arquitectura de Aprendizaje Profundo desarrollada por *Olef Ronneberger, Philipp Fischer y Thomas Brox* en 2015. Fue propuesta en el artículo *U-Net: Convolutional Networks for Biomedical Image Segmentation*, como una mejora en la segmentación de imágenes biomédicas.

Esta arquitectura deriva de las Redes Totalmente Convolucionales (FCN), introducidas en 2014 por *Long, Shalhamer y Darrel*. Las FCN se emplean en tareas de clasificación de imágenes. En ellas, se asigna una clase o categoría a una imagen en su totalidad. En el ámbito de su origen, la biomedicina, surgió la necesidad de detectar a partir de una imagen otros problemas de salud, tumores o área de anormalidad. Con el fin de abordar este objetivo, se desarrolló la arquitectura UNET, que permite asignar una clasificación a cada píxel de la imagen, permitiendo así la identificación precisa de patrones y estructuras.

Esta nueva arquitectura supuso una solución más efectiva en las tareas de segmentación de imágenes, ya que logró superar ciertas limitaciones y asociadas a las Redes Totalmente Convolucionales (FCN) [49]:

- **Resolución reducida:** en las FCN según se avanza en las capas de convolución se reduce la resolución de la imagen, mientras que la arquitectura UNET gracias a las capas de convolución transpuesta y a la conexiones de salto, mantienen la resolución original. Esto permite mantener una alta precisión en la localización detalles pequeños y concretos.
- **Ineficiencia en datos desequilibrados:** las FCN pueden presentar problemas en caso de que haya desequilibrio en la distribución de clases, ya que tienen a fijarse en las regiones más grandes y comunes. La arquitectura UNET permite centrarse en regiones específicas sin perder información.
- **Segmentación:** las arquitecturas FCN tenían como objetivo clasificar una imagen entera dentro de una sola etiqueta o clase. Sin embargo, la arquitectura UNET clasifica cada uno de los píxeles de la imagen, además, gracias a las conexiones de salto y arquitectura de copia, nos permite realizar una segmentación más precisa.

3.4.1. Propuesta original

En esta subsección, se realiza un análisis y se proporciona una detallada explicación de la estructura general de una UNET. Para ello, usaremos como base el esquema proporcionado por los propios creadores.

El nombre *UNET* proviene de la distintiva y simétrica forma de "U" de su estructura, que podemos observar en la **Figura 3.26**. En ella distinguimos 3 partes [50]:

- **Ruta de contracción:** constituye el proceso convolutivo. En ella, se aplican de forma repetida convoluciones, seguido de la función de activación ReLu y, por último, operaciones de agrupación *Max Pooling*. El objetivo es capturar las principales características de las imágenes de entrada. Se corresponde con la parte izquierda del esquema de la **Figura 3.26**
- **Puente o cuello de botella:** su función es conectar ambas rutas, contracción y expansión. En la **Figura 3.26**, constituye la parte inferior central.
- **Ruta de expansión:** sería el proceso de deconvolución o upsampling, donde se emplean convoluciones transpuestas. En esta fase se aumenta la resolución de los mapas de características, reconstruyendo así un mapa de características denso. Debido a su forma y su simetría, se identifica con la parte derecha del esquema.

Además de estas tres partes apreciables en el esquema de la estructura, existen las *Conexiones de salto*, representadas en la **Figura 3.26** como flechas grises. Su objetivo es conectar capas coincidentes, de la ruta codificadora en la decodificadora (gracias a su estructura simétrica). De esta forma, mejora la precisión de la segmentación y evita que se produzca el fenómeno de Evanesencia del Gradiente, ya que toma una copia de los mapas de características obtenidos en capas anteriores y los concatena en la ruta decodificadora.

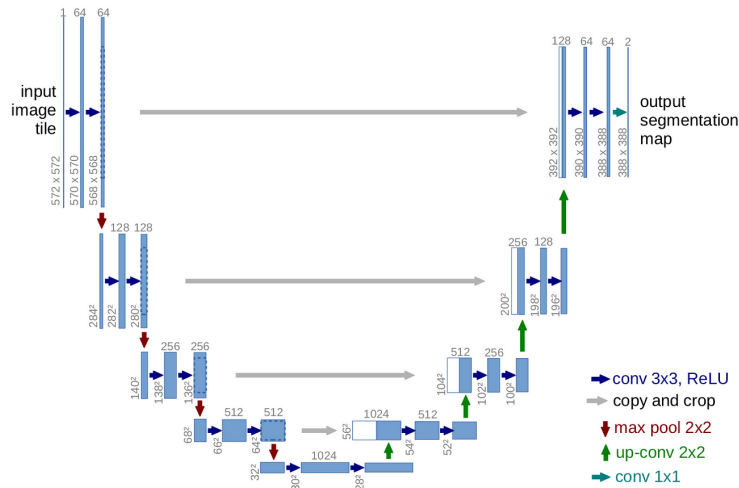


Figura 3.26: Arquitectura U-Net. Fuente: [50]

Después de tener una visión general de la estructura general de la UNET, vamos a ver el funcionamiento de cada una de sus partes de forma detallada [51].

Ruta de contracción

Constituye la parte convolucional del proceso. Su objetivo es llevar a cabo la tarea de *downsample*. Esta etapa se compone de capas convolucionales formadas por dos convoluciones 3x3, donde después de cada una de ellas se aplica la función de activación ReLu elemento a elemento de cada una de las características. Antes de pasar a la siguiente capa de convolución, se realiza la operación de max pooling de tamaño 2x2 y stride de 2. En este esquema, conseguimos, a la vez que reducimos las dimensiones de los mapas de características, aumentar progresivamente la cantidad de características extraídas ya que, entre cada capa convolucional, duplicamos los canales de información.

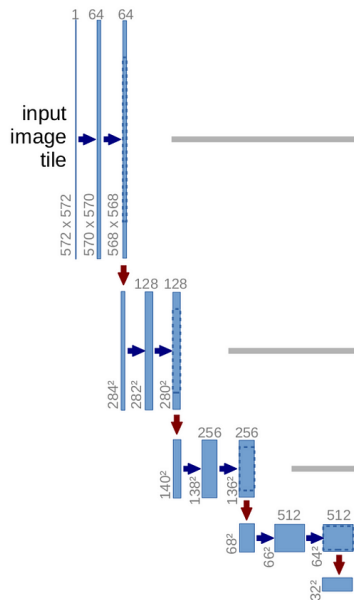


Figura 3.27: Ruta de contracción de un modelo U-Net.

Cuello de botella

Sirve de puente o unión entre la parte codificadora y decodificadora de la red. Las convoluciones ocurren de la misma manera, pero ahora la salida obtenida se somete a una operación de *upsampling*. En el planteamiento

original, el proceso de upsampling utiliza una convolución transpuesta de 2x2, con lo que conseguimos duplicar las dimensiones del mapa de características, reduciendo a la mitad los canales de información para la siguiente etapa.

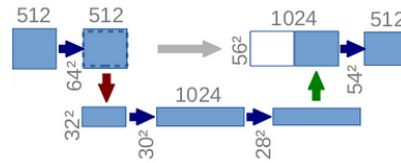


Figura 3.28: Cuello de botella de un modelo U-Net.

Ruta de expansión

Realiza la operación contraria a la parte codificadora o ruta de contracción. Esta etapa comienza tomando como entrada la salida del cuello de botella. De nuevo, consta de capas convolucionales de dos convoluciones 3x3, donde se aplica la función de activación ReLu. Ahora se realiza un proceso de upsampling en lugar de downsampling, esta es la principal diferencia. Se realizan convoluciones transpuestas de 2x2 entre cada capa de convolución; así conseguimos duplicar el tamaño de los mapas de características y reducir a la mitad los canales de información. La última convolución de la capa final es 1x1. Su función es mapear el vector de características de 64 componentes al número de clases que tenemos: 2 en este contexto.

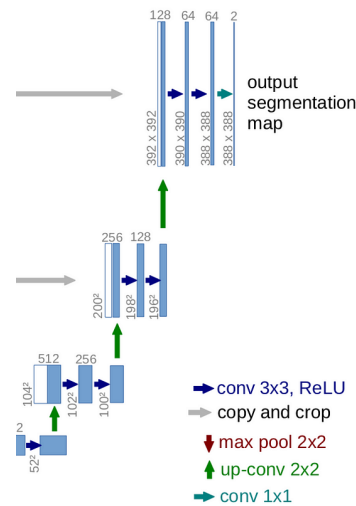


Figura 3.29: Ruta de expansión de un modelo U-Net.

Generalmente, una UNET se divide en dos fases principales: la extracción de patrones y características (ruta de contracción), seguida de la reconstrucción de la imagen original (ruta de expansión), las cuales están unidas por un cuello de botella. En la fase de extracción, la imagen de entrada se procesa gradualmente a través de capas convolucionales para extraer características relevantes, reduciendo la resolución espacial en el proceso. Una vez se alcanza el final de esta fase, se inicia la fase de reconstrucción utilizando técnicas de upsampling para aumentar la resolución y reorganizar la información extraída. Finalmente, la última capa de convolución en la fase de reconstrucción mapea las características a un número específico de clases, lo que resulta en la generación de la imagen final segmentada. Esta última convolución genera, por tanto, tantas máscaras como clases queremos segmentar.

3.5. Otros conceptos importantes

Para construir un modelo neuronal sólido y completo, además de la comprensión de los conceptos vistos en la sección anterior, existen otros que son de gran importancia. Estos conceptos y técnicas amplían nuestra comprensión,

habilidad en el diseño y entrenamiento de Redes Neuronales, permitiendo alcanzar un mejor ajuste a los objetivos deseados. Asimismo, contribuyen a la construcción de un modelo más robusto y efectivo.

3.5.1. Función de activación

La función de activación [52] constituye una parte fundamental de la Red Neuronal. Se encarga de generar la salida de una neurona a partir del valor de entrada recibido.

Estas funciones de activación se dividen en dos tipos principales:

■ Lineales.

La función tiene una ecuación del tipo $f(x) = x$ y la salida no está acotada por un rango. Oscila en un rango $(-\infty, \infty)$. Estas funciones presentan un problema principal: siempre tienen una salida lineal, por lo que las capas sucesivas también, es decir, genera un comportamiento lineal a lo largo de toda la Red Neuronal. Otro contrapunto es que no permiten llevar a cabo la retropropagación debido a su ecuación, cuya derivada siempre es una constante, por lo que, es independiente de la entrada.

■ No lineales.

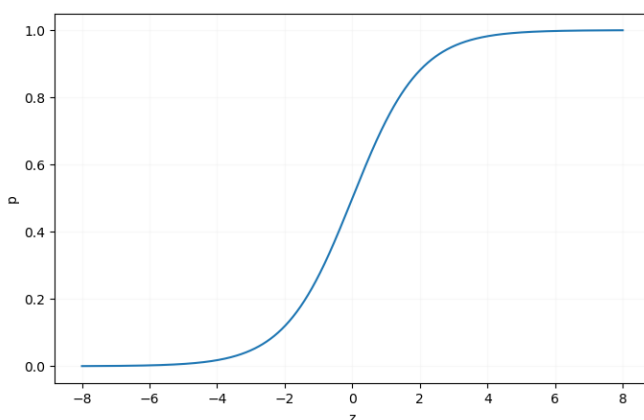
Son las funciones de activación más utilizadas en los modelos de Red Neuronal. Debido a su no linealidad, permiten al modelo adaptarse mejor a los datos y facilita la generalización del mismo.

Las funciones de activación no lineales desempeñan un papel fundamental en el contexto del Aprendizaje Profundo y se erigen como las más ampliamente utilizadas en este dominio, precisamente por su capacidad para superar las dos problemáticas inherentes a las funciones de activación lineales. Por ello, vamos a estudiar los diferentes tipos de funciones no lineales y sus ventajas.

Antes de adentrarnos en la exposición de los diferentes tipos de funciones de activación, es fundamental comprender el concepto de gradiente, ya que desempeña un papel central en nuestro estudio. El gradiente indica la dirección de máximo aumento o disminución de una función, dependiendo de si estamos buscando maximizar o minimizar dicha función. En el contexto del Aprendizaje Automático y las Redes Neuronales, el gradiente se refiere a las derivadas parciales de la función de pérdida con respecto a los pesos de la red. Estos valores guían el proceso de ajuste de los pesos durante el entrenamiento, permitiendo que la red se adapte y mejore su rendimiento en la tarea deseada.

Función sigmoide

Transforma la entrada a una salida que oscila en el rango $(0,1)$, y se utiliza en modelos donde el objetivo es predecir la probabilidad en un problema de clasificación binario. Si el valor de salida es alto, se aproxima de manera asintótica a 1, si por el contrario, es bajo, se aproxima a 0.



Expresión:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3.17)$$

Presenta una serie de características:

- Converge lentamente.
- Mata el gradiente.
- Acotada $(0,1)$.
- Diferenciable, podemos encontrar pendiente en dos puntos cualesquiera.
- Monótona, creciente en todo su dominio.

Figura 3.30: Representación gráfica de la función sigmoide. Fuente: [53]

Función SOFTMAX

Función similar a la sigmoide [54], cuya salida generada oscila de nuevo en el rango (0,1). Se utiliza habitualmente en problemas de clasificación múltiples, ya que nos devuelve una distribución de probabilidad de pertenecer cada clase, de forma que las componentes de la salida suman 1.

Expresión:

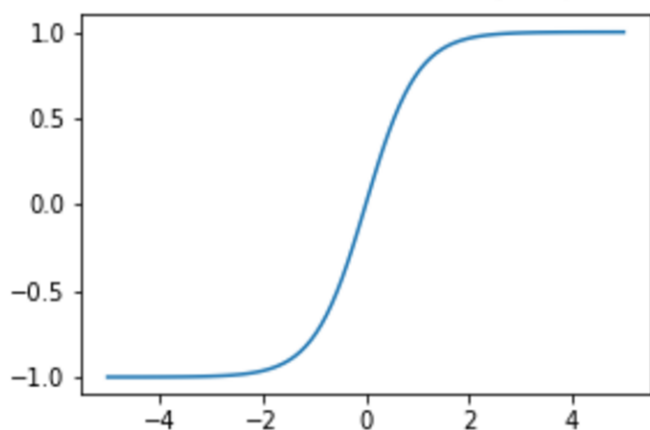
$$f(z)_j = \frac{e^z_j}{\sum_{k=1}^K e^{z_k}} \quad (3.18)$$

Presenta una serie de características:

- Acotada (0,1).
- Diferenciable, podemos encontrar pendiente en dos puntos cualesquiera.
- Buen rendimiento últimas capas.
- Problemas de clasificación multiclase.

Función tangente hiperbólica

Función muy similar a la sigmoide, pero ahora genera una salida que oscila entre (-1,1). Ahora los valores altos se aproximan asintóticamente a 1 y, los bajos, a -1. Se emplea habitualmente en problemas de clasificación binaria.



Expresión:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (3.19)$$

Presenta una serie de características:

- Converge lentamente.
- Disminuye el gradiente.
- Acotada (-1,1).
- Diferenciable.
- Monótona creciente en todo su dominio.

Figura 3.31: Representación gráfica de la función tangente hiperbólica. Fuente: [55]

Función de activación ReLU

En inglés *Rectified Lineal Unit* es la función de activación más utilizada. Transforma los valores anulando los negativos asignándoles el valor 0, y dejando los positivos con el mismo valor que entran.

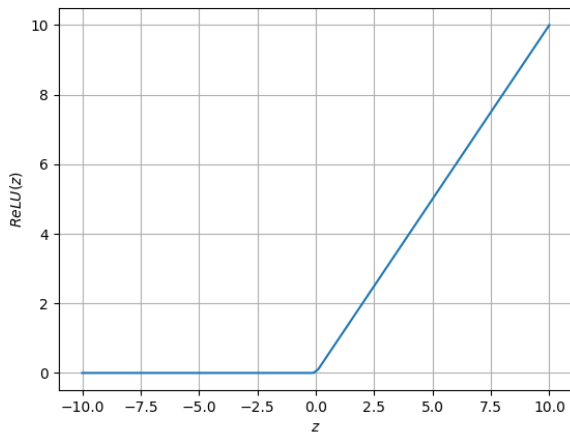


Figura 3.32: Representación gráfica de ReLU. Fuente: [56]

Es una de las funciones de activación más utilizadas desde el año 2000, debido a que presenta una serie de ventajas respecto al resto de las funciones. La ReLU es capaz de reducir el efecto *Evanescencia del Gradiente*, que veremos en la **Sección 3.6.1**. Los modelos que implementan ReLU convergen a una mayor velocidad, en comparación con modelos que utilizan otras funciones, como la sigmoide o tangente hiperbólica, gracias a la facilidad de optimización que presentan esos modelos y la simplicidad de sus operaciones.

A pesar de su amplia utilización en Redes Neuronales, la ReLU presenta dos puntos en contra importantes. En primer lugar, no es derivable en el punto $x=0$. Esto puede dificultar el proceso de entrenamiento de la red en algunos casos y complicar el uso de ciertas técnicas de optimización que dependen de las derivadas. El otro inconveniente importante es que convierte todos los valores negativos en cero, lo que se traduce en la pérdida de la capacidad del modelo para adaptarse adecuadamente a patrones de datos donde los valores negativos pueden ser significativos.

Función Leaky ReLU

Muy similar a la función de activación ReLU, pero resuelve uno de sus principales problemas. Transforma los valores de forma que los negativos los multiplica por un coeficiente rectificativo [57], y los positivos no reciben ninguna modificación. Normalmente, este coeficiente rectificador toma el valor de 0.01 o valores similares, en caso de ser distinto la función pasa a denominarse *Randomized ReLU*.

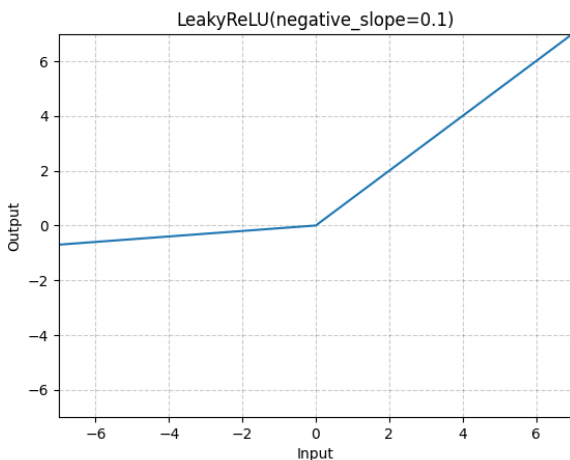


Figura 3.33: Representación gráfica Leaky ReLU. Fuente: [57]

Expresión:

$$f(x) = \max(0, x) \quad (3.20)$$

Presenta una serie de características:

- No está acotada $(0, \infty)$.
- Mayor velocidad de convergencia.
- Solo se activa con valores positivos.
- Monótona creciente en todo su dominio.

La función a trozos $f(x)$ se define de la siguiente manera, siendo a el coeficiente rectificador:

$$f(x) = \begin{cases} a \cdot x & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \quad (3.21)$$

Presenta una serie de características:

- No está acotada $(-\infty, \infty)$.
- Penaliza los valores negativos.
- Buen desempeño con imágenes.
- Monótona creciente en todo su dominio.

3.5.2. Función de pérdida

En el contexto de las Redes Neuronales [58], la función de pérdida se utiliza para evaluar el ajuste del modelo a los datos del conjunto de entrenamiento. Su objetivo es medir la similitud entre las predicciones y los valores reales. Es una forma de cuantificar el *error* a lo largo del proceso de entrenamiento, lo que permite que durante este proceso, se ajusten gradualmente ciertos parámetros con el fin de reducir las diferencias entre la predicción y los valores reales consiguiendo, así, mejorar el rendimiento.

En el campo de las funciones de pérdida, existe una gran variedad de ellas diseñadas para abordar objetivos específicos y mejorar en el entrenamiento del modelo. Podemos categorizar estas funciones en dos grupos distintos: aquellas diseñadas para tareas de regresión y las ideadas para problemas de clasificación. En el contexto de este TFG, no vamos a realizar una distinción concreta de los tipos de funciones de pérdida; únicamente comentaremos dos técnicas, que corresponden con las más utilizadas en modelos de Aprendizaje Automático.

MSE (Mean Square Error)

El error cuadrático medio se define como la media de las diferencias al cuadrado entre la predicción y los valores reales.

Donde:

- n : es el tamaño del conjunto de datos.
- y_i : valor real.
- $\hat{y}_i$: predicción realizada por el modelo.

Formulación:

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n}$$

Aunque es la función de pérdida más utilizada, penaliza los errores grandes de forma significativa debido al cuadrado de la fórmula, por lo que es sensible ante valores extremos, también conocidos como outliers.

Cross Entropy Loss

La pérdida de entropía cruzada es una función de pérdida muy empleada en problemas de clasificación. Mide la cantidad de información que se pierde al comparar la predicción de la clase con la etiqueta real. La entropía cruzada aumenta a medida que la predicción difiere de la etiqueta real, por lo que el objetivo es su minimización o, lo que es lo mismo, hacer que la predicción se asemeje lo más posible al valor real.

Existen dos formulaciones: una para problemas de clasificación binaria y otra para multiclase:

Clasificación Binaria

Donde:

- y : es el valor de la etiqueta real, 0 o 1, al tratarse de un problema binario.
- p : es la probabilidad de que el ejemplo tomado pertenezca a la clase 1 predicha por el modelo.

Formulación:

$$CE(y, p) = -(y \log(p) + (1-y) \log(1-p))$$

Clasificación Multiclase

Donde:

- y : vector one-hot, que contiene el valor de la clase real. Elemento que vale 1 es el que toma el valor de la clase real y , el resto, valen 0.
- p : vector de probabilidades predichas por el modelo.
- K : número de clases.

Formulación:

$$CE(y, p) = -\sum_{i=1}^K y_i \log(p_i)$$

En esta sección, hemos explorado dos funciones de pérdida fundamentales en el Aprendizaje Automático: el Error Cuadrático Medio (MSE) y la Entropía Cruzada (Cross-Entropy Loss). Estas funciones se emplean habitualmente en problemas de regresión y clasificación respectivamente. Ahora examinaremos con detalle los optimizadores: otro componente clave en el proceso de entrenamiento de Redes Neuronales. Adquieren gran importancia en el ajuste de los parámetros del modelo, porque nos sirven para minimizar el valor de estas funciones de pérdida para lograr un rendimiento óptimo.

3.5.3. Optimizadores

Son algoritmos o métodos [59] que se emplean en el contexto de las Redes Neuronales para ajustar ciertos parámetros del modelo, como pueden ser los pesos o tasa de aprendizaje, con el fin de minimizar o maximizar el valor de la función de pérdida en el proceso de entrenamiento. El objetivo principal es guiar la actualización de pesos, sesgos y tasas de aprendizaje de la red, con el fin de reducir las pérdidas y obtener resultados más precisos.

Dentro del amplio campo de los optimizadores para Redes Neuronales, existen numerosas variantes de la técnica de descenso de gradiente. Sin embargo, en este contexto, nos centraremos en las dos principales y más ampliamente reconocidas, que han demostrado obtener resultados excepcionales y han sido ampliamente adoptadas a nivel mundial:

■ Descenso del gradiente estocástico.

Es una variante del descenso del gradiente, donde se impone que los lotes se escojan aleatoriamente, para evitar que sean todas muestras de dichos lotes de una misma clase o que predomine una de ellas. Si no hubiese lotes, el carácter estocástico se obtiene haciendo una permutación aleatoria al inicio de cada época.

Donde:

- θ : representa el vector de parámetros que se están optimizando.
- α : tasa de aprendizaje.
- $\nabla J(\theta; x^{(i)}, y^{(i)})$: gradiente de la función de pérdida $J(\theta)$, respecto a los parámetros θ para una única instancia de entrenamiento con características $x^{(i)}$ y label $y^{(i)}$.

Algoritmo:

$$\theta = \theta - \alpha \nabla J(\theta; x^{(i)}, y^{(i)})$$

Presenta una serie de ventajas e inconvenientes que quedan recogidas en la siguiente tabla:

Ventajas	Inconvenientes
Converge en menos tiempo	Alta variación en parámetros y valor de la función
Requiere menos memoria	Necesita reducir lentamente el valor de la tasa de aprendizaje

Tabla 3.1: Ventajas e inconvenientes descenso del gradiente estocástico

■ Descenso del gradiente mini batch.

Es la mejor variante de todas las presentes en el algoritmo del Descenso de Gradiente. El conjunto de datos se divide en lotes, y los parámetros se actualizan después de cada lote consiguiendo, así, reducir la varianza significativamente.

Donde:

- θ : representa el vector de parámetros que se están optimizando.
- α : tasa de aprendizaje.
- $\nabla J(\theta; B(i))$: gradiente de la función de pérdida $J(\theta)$, respecto a los parámetros θ , de un $B(i)$ que son los lotes de ejemplo de entrenamiento con n instancias.

Algoritmo:

$$\theta = \theta - \alpha \nabla J(\theta; B(i))$$

Ventajas	Inconvenientes
Presenta menos variaciones, ya que actualiza por cada lote	Elección de tasa de aprendizaje óptima
Requiere una cantidad media de memoria	Se puede perder en mínimos locales

Tabla 3.2: Ventajas e inconvenientes descenso del gradiente mini batch

■ Momento.

El término *momento* fue inventado para incorporarlo en los algoritmos de optimización para conseguir mejorar la convergencia y estabilidad en el entrenamiento del modelo. Este método consigue reducir la varianza del descenso del gradiente, a través de un nuevo hiper-parámetro denominado "momento": γ . Con ello, se ayuda a estabilizar la convergencia del modelo hacia la dirección de interés, reduciendo posibles fluctuaciones hacia direcciones que perjudican la convergencia de éste.

Donde:

- θ : representa el vector de parámetros que se están optimizando.
- α : tasa de aprendizaje.
- $v(t)$ y $v(t-1)$: vector de momento en la iteración t y $t-1$, respectivamente.
- γ : hiper-parámetro momento. Habitualmente toma el valor 0.9, o similar.
- $\nabla J(\theta; B(i))$: gradiente de la función de pérdida $J(\theta)$, respecto a los parámetros θ .

Algoritmo:

$$V(t) = \gamma V(t-1) + (\alpha) \nabla J(\theta)$$

Nuevos parámetros::

$$\theta = \theta - \alpha V(t)$$

Ventajas	Inconvenientes
Reduce oscilaciones y varianza	Añade un nuevo hiper-parámetro cuyo valor debemos elegir con precisión
Converge más rápido	

Tabla 3.3: Ventajas e inconvenientes de la técnica Momento

■ Adagrad.

Adagrad (Adaptive Gradient Descent) mitiga uno de los principales problemas de los optimizadores vistos hasta ahora, en los que la tasa de aprendizaje es la misma para todas las iteraciones. Para evitar esto, modifica la tasa de aprendizaje η para cada parámetro y en cada iteración t , en función del historial particular de gradientes.

Donde:

- $\theta_{t+1,i}$, $\theta_{t,i}$: valor actualizado del parámetro i en la iteración $t+1$, t , respectivamente.
- η : tasa de aprendizaje. Se modifica en función de un parámetro dado $\theta(i)$, en un momento dado t y basándose en los gradientes calculados previos de ese parámetro.
- ϵ : factor que evita la división por cero. Habitualmente del orden de 10^{-8} .
- $G_{t,ii}$: matriz diagonal que almacena la suma acumulativa de los cuadrados de los gradientes del parámetro i en la iteración t .
- $g_{t,i}$: gradiente de la función de pérdida respecto al parámetro i en la iteración t .

Algoritmo:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Ventajas	Inconvenientes
Se ajusta automáticamente la tasa de aprendizaje	Computacionalmente costoso
Funciona adecuadamente con datos escasos	Entrenamiento del modelo lento

Tabla 3.4: Ventajas e inconvenientes de la técnica Adagrad

■ RMSProp.

Es una extensión del método anterior que soluciona el problema principal que éste presenta: su lentitud. RMSProp en lugar de acumular todos los antiguos gradientes, como hace Adagrad, limita el número de ellos usando el concepto de ventana (de tamaño w). De esta forma, selecciona sólo los gradientes más recientes. Aplica una media exponencial ponderada para suavizar los cambios que se realizan sobre los parámetros, consiguiendo reducir cambios bruscos.

Donde:

- η : tasa de aprendizaje.
- $E[g^2]_t$: representa la media acumulativa de los cuadrados de los gradientes.
- γ : hiper-parámetro que controla la media anterior, es decir, la contribución de $E[g^2]_{t-1}$ en $E[g^2]_t$. Habitualmente toma el valor de 0.9.
- $(dC/dW)^2$: el cuadrado del gradiente de la función de pérdida C , con respecto a los parámetros w en la iteración t .

Algoritmo:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) \left(\frac{dC}{dW} \right)^2$$

Nuevos parámetros:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t}} \left(\frac{dC}{dW} \right)$$

Ventajas	Inconvenientes
Evitamos la decadencia de la tasa de aprendizaje	Alto coste computacional

Tabla 3.5: Ventajas e inconvenientes de la técnica RMSProp

■ Adam.

Adam (Adaptive Moment Estimation) [60] combina dos de las técnicas anteriormente mencionadas, por lo que es uno de los algoritmos más utilizados debido a su eficiencia y rendimiento en una amplia variedad de problemas. Utiliza las ventajas proporcionadas por la técnica del Momento y RMSProp, empleando la estimación del momento y la magnitud de los gradientes anteriores para actualizar el valor de los parámetros en cada iteración. Para lograr una mayor eficiencia y ajuste en el entrenamiento, Adam no emplea una tasa de aprendizaje constante, sino que adapta esta individualmente para cada parámetro.

Donde:

- $\hat{m}_t$: la media del momento de los gradientes en la iteración t .
- $\hat{v}_t$: la media del segundo momento de los gradientes en la iteración t .
- η : tasa de aprendizaje.
- ε : factor que evita la división por cero. Habitualmente del orden de 10^{-8} .
- w_t, w_{t-1} : pesos del modelo en las iteraciones t y $t-1$.
- β_1, β_2 : parámetros de suavizado, que controlan los valores $\hat{m}$ y $\hat{v}$. Habitualmente toman valores 0.9 y 0.999 respectivamente.

Algoritmo:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Actualización parámetros:

$$w_t = w_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

Ventajas	Inconvenientes
Converge rápidamente	Alto coste computacional
Corrige el descenso de la tasa de aprendizaje y las altas variaciones	

Tabla 3.6: Ventajas e inconvenientes de la técnica RMSProp

3.5.4. Inicialización de pesos

La inicialización de pesos constituye un aspecto crítico en el desarrollo y creación de un modelo de Deep Learning, ya que define la estructura inicial y, en consecuencia, su capacidad para representar y aprender de los datos. Una elección adecuada de los pesos iniciales puede prevenir la aparición de problemas comunes, como la Evanescencia del Gradiente. Por el contrario, una incorrecta elección inicial de los pesos, puede provocar obstáculos en el proceso de entrenamiento del modelo produciendo, así, una lenta convergencia o, en el peor de los casos, que nunca converja.

No existe una única estrategia de inicialización de pesos que funcione para todos los escenarios, sino que existen diversos métodos de inicialización. Algunos de los más comunes son [61]:

1. Inicialización a cero.

La técnica de inicialización a cero, consiste en establecer todos los pesos de la red en cero al inicio del entrenamiento. Es la estrategia más simple para preparar una red para su entrenamiento. Debido a su simplicidad, presenta una serie de limitaciones. En primer lugar, esta técnica resulta muy ineficaz, porque todos los pesos de la red son iguales, por lo que todas las neuronas producen la misma salida y se actualizan de igual manera en cada iteración. Este fenómeno se conoce como *simetría*. Lo que limita la capacidad de la red para aprender patrones y representaciones en los datos. Esta problemática ocurre siempre que se lleva a cabo una inicialización constante de pesos, por lo que se opta por otro tipo de inicializaciones.

2. Inicialización aleatoria.

Esta técnica trata de solventar las limitaciones de la inicialización a cero. Para ello, asigna valores aleatorios (excepto ceros) a los pesos de las neuronas. Al asignar valores aleatorios es posible que estos adquieran valores muy altos o demasiado bajos, provocando Sobreajuste y Evanescencia del Gradiente respectivamente. La probabilidad de que se asignen valores extremos depende, en cierta medida, de la distribución de la inicialización y la escala de valores.

Relacionado con esto último, existen dos tipos de inicialización aleatoria:

- **Normal aleatoria:** los pesos se inicializan tomando valores de una distribución normal de media cero y desviación típica determinada. Es técnica puede funcionar bien en arquitecturas profundas, porque ayuda a evitar la saturación de neuronas.
- **Uniforme aleatorio:** los pesos se inicializan tomando valores de una distribución uniforme en un rango específico: son seleccionados aleatoriamente con igual probabilidad. De esta manera, limitamos la magnitud de los pesos iniciales y tenemos un control más preciso sobre el rango de valores. Su principal desventaja es que puede ser menos útil en arquitecturas profundas.

3. Inicialización de Xavier.

En la inicialización de Xavier, también conocida como la *inicialización de Glorot*, tiene como objetivo abordar el problema de la inicialización de pesos de forma más efectiva. La idea principal es ajustar los valores iniciales, de manera que la varianza de las salidas obtenidas se mantenga aproximadamente constante a medida que se avanza en la capas de la red.

La distribución uniforme se toma en el rango $[-a, a]$, donde a se calcula de la siguiente manera:

$$a = \sqrt{\frac{6}{f_{an_in} + f_{an_out}}} \quad (3.22)$$

Donde:

- **fan.in**: representa las conexiones de entrada de una neurona.
- **fan.out**: representa las conexiones de salida de una neurona.

En la **Figura 3.34** se ilustran los conceptos anteriormente explicados, donde $w1$, $w2$, $w3$ son conexiones de entrada y $W1$, $W2$ son conexiones de salida.

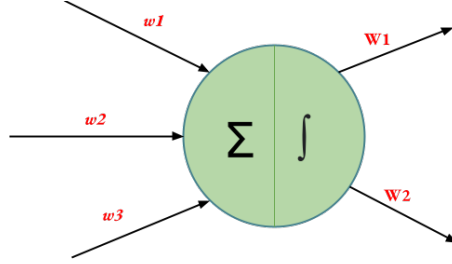


Figura 3.34: Conexiones de entrada y salida de una neurona. Fuente: [61]

4. Inicialización normalizada de Xavier.

Similar a la inicialización de Xavier original, pero ahora los valores siguen una distribución normal de media 0 y desviación típica a :

$$w_i \sim N(0, a) \quad (3.23)$$

Donde a se calcula de la misma forma que en la inicialización de Xavier:

$$a = \sqrt{\frac{6}{\text{fan.in} + \text{fan.out}}} \quad (3.24)$$

5. Inicialización He.

Recibe este nombre en honor a Keiming He [62], autor del artículo que popularizó esta técnica en 2015, que trata de abordar el problema de la Evanescencia del Gradiente en Redes Profundas. Consigue resolver este problema, gracias a que la varianza se escala en función del número de conexiones de entrada permitiendo, así, que la información fluya a través de la red. Esta técnica funciona especialmente bien en redes donde se aplican funciones de activación como la ReLU y variantes, es decir, funciones de activación rectificadas.

Sigue una distribución normal donde la media es cero y la varianza $\frac{2}{\text{fan.in}}$:

$$w_i \sim N(0, \sqrt{\frac{2}{\text{fan.in}}}) \quad (3.25)$$

3.5.5. Técnicas de regularización

Las técnicas de regularización son un elemento fundamental en el entrenamiento de modelos de Deep Learning. Su principal objetivo es evitar el sobreajuste del modelo. Su aplicación busca conseguir que el modelo aprenda patrones y características generales, permitiendo que se pueda aplicar a cualquier conjunto de datos, mejorando su capacidad de generalización y fortaleciendo la robustez del modelo.

Existen tres principales técnicas de regularización [63]:

■ Regularización L1 (Lasso)

El objetivo de esta regularización es forzar a algunos pesos de la red a volverse exactamente cero y así eliminar características que pueden ser irrelevantes en el modelo. La función que vemos en la **Ecuación 3.27**, solo considera los pesos absolutos y además introduce un parámetro de ajuste λ , que se encarga de controlar cuánto se penalizan los valores absolutos de los coeficientes de las características de la función de pérdida.

$$L1 = \sum_{j=1}^m (Y_i - W_o - \sum_{i=1}^n W_i X_{j,i})^2 + \lambda \sum_{i=1}^n |W_i| \quad (3.26)$$

λ en Lasso es un hiper-parámetro crucial que controla la selección de características y el grado de regularización. Es importante ajustar el valor, de forma que equilibre el ajuste a los datos de entrenamiento con la capacidad de generalización. El comportamiento de λ , según el valor que tome, son los siguientes [64]:

- **$\lambda = 0$:** el modelo se comporta como una regresión lineal convencional, sin penalización en los valores absolutos de los coeficientes. En este caso no se elimina ninguna característica y se utilizan todas las disponibles.
- **Valor de λ pequeño:** el valor va aumentando desde 0 y se aplica una pequeña penalización a los valores absolutos de los coeficientes. Es una reducción ligera, por lo que no se llegan a establecer a cero: simplemente reduce la importancia de las características menos relevantes, pero sigue utilizando todas.
- **Valor de λ grande:** aplica una reducción significativa a los valores absolutos de los coeficientes. Esta reducción se considera importante: algunos coeficientes toman el valor cero y se eliminan de las características, manteniendo únicamente las que son relevantes para el modelo.

Esta técnica es especialmente útil cuando existen características de entrada que no son relevantes para el problema; con ello, se busca reducir la dimensionalidad del problema.

■ Regularización L2 (Ridge)

En esta operación añadimos la suma de los cuadrados de los pesos a la función de pérdida. Conseguimos penalizar, así, los pesos más grandes y favorecer los pesos pequeños. El objetivo principal es reducir todos los pesos de la red, pero a diferencia de la *regularización L1*, no es necesario que éstos alcancen forzosamente el valor cero. El comportamiento del hiper-parámetro λ es el mismo que en la regularización L1.

$$L2 = \sum_{j=1}^m (Y_i - W_o - \sum_{i=1}^n W_i X_{j,i})^2 + \lambda \sum_{i=1}^n W_i^2 \quad (3.27)$$

Habitualmente se emplea en problemas de regresión y clasificación. Normalmente ayuda a mantener los pesos controlados, evitando que tomen valores extremos y el sobreajuste.

■ Dropout

Se emplea habitualmente en las capas totalmente conectadas, donde existe una mayor posibilidad de que ocurra el sobreajuste. El dropout simplemente *ignora* un conjunto de neuronas seleccionadas de manera aleatoria en algunas fases del entrenamiento, desactivando así conexiones entre ellas. Tras aplicar dropout como se ve en la **Figura 3.35**, se reduce el número de neuronas de la red cuyos pesos se actualizan. La probabilidad de que una neurona sea excluida de la red es p , o lo que es lo mismo, la probabilidad de que contribuya en el entrenamiento es $1 - p$.

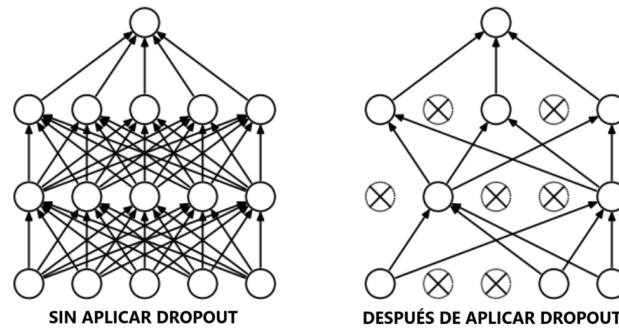


Figura 3.35: Aplicación dropout. Fuente: [65]

Gracias a esta técnica, conseguimos que disminuya el sobreajuste del modelo, porque evitamos el entrenamiento de todas las neuronas en el conjunto de entrenamiento de una vez. Asimismo, al reducir el número

de éstas mejoramos la velocidad de entrenamiento y permitimos que el modelo aprenda funciones internas más robustas, lo que suele implicar la generalización mejor ante datos nuevos. Presenta un inconveniente principal: si realizamos dropout normalmente implica un entrenamiento con un mayor número de épocas, en comparación al entrenamiento sin dropout.

3.6. Problemas en el entrenamiento

Durante el proceso de entrenamiento de modelos de Deep Learning, además de la selección de numerosos hiper-parámetros e implementación de diversas técnicas a lo largo de las distintas fases del aprendizaje, también debemos tener en cuenta una serie de desafíos y obstáculos que pueden aparecer durante el desarrollo y aprendizaje del modelo, en particular, la Evanescencia del Gradiente, problemas de sobreajuste o infrajuste, entre otros. Aunque estos conceptos ya han aparecido a lo largo de este Capítulo, trataremos de analizar y explorar cada uno de estos problemas con más detalle.

3.6.1. Evanescencia del gradiente

Este fenómeno [66] ha aparecido en numerosas ocasiones hasta ahora, suele ocurrir por lo general, en modelos muy profundos que emplean métodos basados en el gradiente y retropropagación para su aprendizaje. El problema consiste en que el gradiente disminuye significativamente a medida que se realiza la retropropagación a través de las capas más profundas, volviéndose demasiado pequeño y provocando que las neuronas de las capas profundas no realicen apenas actualización de sus pesos.

Este problema suele manifestarse en modelos de Redes Neuronales Profundas que emplean funciones de activación tradicionales, como la sigmoide o la tangente hiperbólica. Para comprender mejor este fenómeno, consideramos un ejemplo sencillo. Supongamos que tenemos un modelo de Red Neuronal con solo tres capas ocultas, donde utilizamos la función sigmoide, cuyas derivadas se sitúan en el rango $(0, 0.25]$. Durante el proceso de entrenamiento, el gradiente se calcula mediante retropropagación y las derivadas de las funciones de activación sigmoide se multiplican en cada capa.

Imaginemos que, en un momento dado del entrenamiento, llegamos a la última capa con un gradiente igual a 4.0. Ahora, veamos cómo este gradiente se reduce exponencialmente al retroceder en las capas ocultas:

- En la segunda capa, calculamos el gradiente multiplicando el gradiente de la última capa por la derivada de la función sigmoide en esa capa, que asumimos es 0,15. Como resultado, el gradiente en la segunda capa se convierte en $4,0 * 0,15 = 0,6$.

- Continuando con este proceso, en la primera capa repetimos el cálculo anterior. Por lo tanto, el gradiente en esta capa se reduce a $0,6 * 0,15 = 0,09$.

Este ejemplo simple nos muestra cómo el gradiente disminuye exponencialmente a medida que retrocedemos a través de las capas. En modelos de Redes Neuronales Profundas con numerosas capas ocultas, este fenómeno puede llevar a gradientes extremadamente pequeños, implicando así actualizaciones de peso prácticamente nulas. Como ya sabemos, este fenómeno dificulta considerablemente el proceso de ajuste y entrenamiento de la red.

Para evitar este problema, a parte de elegir valores adecuados para ciertos hiper-parámetros del modelo, existen técnicas diseñadas con el fin de abordar este problema:

1. Batch Normalization

La normalización se implementa a través de una serie de operaciones, y tiene como objetivo amplificar las diferencias entre muestras con valores muy similares. En particular, adapta y normaliza los datos de los lotes durante el proceso de entrenamiento. Este procedimiento implica el cálculo de la media y la varianza de un lote de datos, seguido de la normalización de las salidas mediante la sustracción de la media y la división por la desviación estándar. Posteriormente, se aplica una transformación lineal y una escala. Este proceso asegura que las salidas de las neuronas se mantengan dentro de un rango útil, $(0,1)$.

No obstante, en el libro *"Deep Learning with Python"* de François Chollet [67], se presenta una afirmación intrigante: 'No se sabe realmente con certeza por qué la normalización por lotes ayuda' en relación a la cuestión de por qué esta técnica es eficaz en la mitigación del problema de la desaparición del gradiente.

2. Conexiones residuales

Las conexiones residuales permiten a las capas aprender una diferencia residual entre la entrada y la salida. Nos permiten *saltarnos* una o más capas, de forma que el gradiente se suma a la capa de salida en lugar de ser modificado por cada una de las capas. Tiene una ventaja principal, si la conexión residual salta demasiadas capas se puede eliminar o reducir el número de capas que afecta. Gracias a este salto, se consigue que la información fluya fácilmente a través de la red mitigando así la Evanescencia del Gradiente.

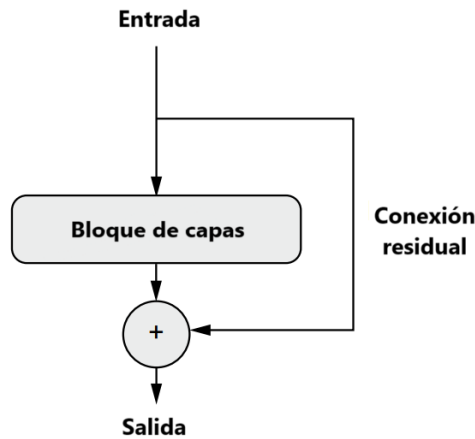


Figura 3.36: Ejemplo conexión residual. Fuente: [67]

3. Depthwise separable convolution

Es una variante de la convolución original [68], pero presenta una diferencia principal: en vez de aplicar una única convolución a todos los canales de información, ahora se realiza una convolución separada para cada canal. Esta técnica se basa en la suposición de que los diferentes canales de entrada son independientes y las ubicaciones espaciales de las salidas intermedias están muy correlacionadas. La ventaja principal frente a una convolución normal, es que requiere significativamente menos parámetros y realizar menos cálculos, por lo que converge más rápido y son menos propensas a la Evanescencia del Gradiente y el Sobreajuste.

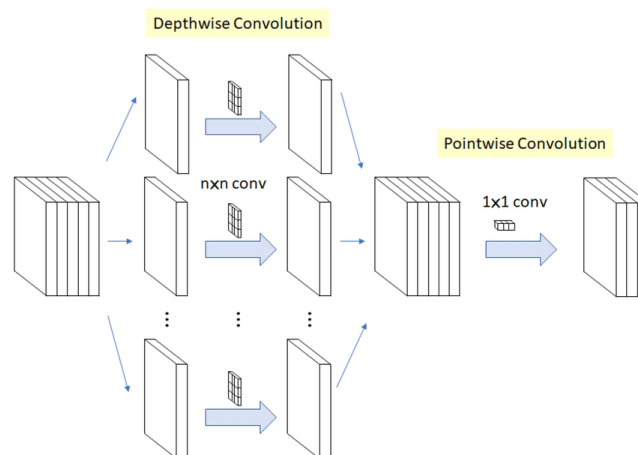


Figura 3.37: Ejemplo Depthwise separable convolution. Fuente: [68]

Como vemos en la **Figura 3.37** se realiza una convolución para cada canal de entrada de forma independiente, consiguiendo extraer características concretas de cada canal. Se combinan las salidas, y después se realiza una convolución 1x1, para fusionar y refinar las características identificada antes de pasar a la siguiente capa.

Existen otras técnicas como el dropout y de regularización que ya han sido comentadas en detalle en la **Sección 3.5.5**, por lo que se ha omitido su explicación en esta, pero que también pueden ser utilizada como técnicas de mitigación de este problema.

3.6.2. Sobreajuste (Overfitting)

El sobreajuste [69] es un fenómeno, por el cual el modelo pasa de aprender, a memorizar los ejemplos de entrenamiento, obteniendo un comportamiento deficiente ante nuevos datos. En otras palabras, el modelo no desarrolla la capacidad de generalizar y únicamente se ajusta al conjunto de entrenamiento proporcionado.

El sobreajuste puede darse por diversos factores:

- **Conjunto de entrenamiento pequeño:** al ser demasiado pequeño no presenta instancias que representen todos los posibles valores de entrada. Esto implica que el modelo se ajusta a los ejemplos específicos proporcionado por los datos.
- **Ruido en los datos:** los datos tienen información errónea o irrelevante que el modelo intenta aprender.
- **Demasiado entrenamiento:** si el entrenamiento se realiza durante muchas épocas o iteraciones, puede que el modelo memorice el conjunto de entrenamiento. Esto ocurre cuando sigue aprendiendo a pesar de haber alcanzado ya un rendimiento adecuado para los datos proporcionados.
- **Hiper-parámetros inadecuados:** elección de valores inapropiados de algunos parámetros fundamentales de la red.
- **Modelo complejo:** elaborar un modelo excesivamente complejo, puede hacer que éste aprenda del ruido de los datos, en lugar de hacerlo sobre la información de relevante.

Para prevenir el sobreajuste, es esencial realizar tareas de escalado y diversificación de los datos de entrenamiento. Sin embargo, además de estas estrategias generales, existen enfoques específicos para mitigar el riesgo de sobreajuste. Entre estas técnicas, se encuentran la detección de ruido en los datos, la selección cuidadosa de características relevantes, el empleo de técnicas de regularización y la aplicación de técnicas de Data Augmentation sobre los datos, entre otras. Es importante destacar, que también se puede dar la situación contraria, conocida como infrajuste, que se abordará en la siguiente sección.

3.6.3. Infrajuste (Underfitting)

El infrajuste [70] representa el fenómeno contrario al sobreajuste. En este caso el modelo no se ajusta de forma adecuada a los datos de entrenamiento, y tampoco desarrolla la capacidad de generalización para obtener buenos resultados ante datos no vistos previamente.

El infrajuste puede darse por diversos factores:

- **Simplicidad del modelo:** debido a la simplicidad de éste, no es capaz de extraer las características complejas de los datos.
- **Conjunto de entrenamiento pequeño:** el conjunto no es suficientemente grande para que el modelo pueda desarrollar la capacidad de generalización.
- **Falta de escalado:** es conveniente realizar previamente un ajuste de todas las características o entradas a una escala común, de forma que todas tomen valores dentro de un rango similar.

Así como ocurre con el sobreajuste, el proceso de prevención del infrajuste, también requiere realizar tareas de preprocesamiento de los datos. Adicionalmente, existen enfoques específicos para mitigar el riesgo de infrajuste, que enfoques incluyen el aumento de la complejidad del modelo, la ampliación del conjunto de datos de entrenamiento recopilando más ejemplos o mediante técnicas de Data Augmentation, así como la aplicación de técnicas de regularización y la selección cuidadosa de hiper-parámetros óptimos.

Esta sección ha proporcionado los cimientos necesarios para emprender con éxito el desarrollo de este Trabajo de Fin de Grado. Se han explorado conceptos fundamentales en el campo de las Redes Neuronales, incluyendo

su arquitectura, componentes, diversas técnicas y operaciones relevantes, así como posibles desafíos durante el proceso de entrenamiento de los modelos. Estos conocimientos establecen una sólida base de la que partiremos para comprender, en capítulos posteriores, cómo hemos diseñado, desarrollado y entrenado nuestro propio modelo.

En la siguiente sección, se abordará detalladamente el conjunto de datos elegido, el origen de éste y las transformaciones que se van a llevar a cabo antes de ser introducido en el modelo.

Capítulo 4

Conjunto de datos

Este capítulo presenta un análisis exhaustivo del conjunto de datos seleccionado para este Trabajo Fin de Grado. En primer lugar, se proporciona una descripción detallada de su contenido y su origen. Asimismo, en la primera parte de este Capítulo se describen aspectos relacionados con el formato y las características específicas de los datos, incluyendo detalles sobre la estructura de etiquetado y la distribución de las muestras. En la segunda parte, se presentan las distintas transformaciones y adecuaciones realizadas sobre el conjunto para su posterior utilización.

4.1. Descripción del conjunto

El conjunto de datos seleccionado para este Trabajo de Fin de Grado se conoce como *Semantic Segmentation Drone Dataset*, el cual consta de un total de 400 imágenes urbanas capturadas por drones en formato JPG. Asimismo, cada imagen está acompañada por su correspondiente mapa Ground Truth, en formato PNG. Este conjunto de datos ofrece dos opciones distintas para su uso: una pensada para desarrollar un proyecto de segmentación binaria y otra para multicategoría. En nuestro caso, emplearemos exclusivamente el segundo conjunto, que permite la identificación de los siguientes cinco macrogrupos: obstáculos, agua, vegetación, objetos en movimiento y zonas consideradas aptas para el aterrizaje. Me refiero a los macrogrupos en lugar de las clases, ya que inicialmente las imágenes de referencia (Ground Truth) se concibieron para la identificación de 24 clases distintas. No obstante, el reto planteado en la plataforma Kaggle, se ha enfocado en la categorización de estas imágenes en 5 macrogrupos más amplios. En la **Sección 4.2.1**, se abordará el proceso de mapeo de píxeles que ha sido necesario realizar, para ajustarse a esta consideración, facilitando la transición de la clasificación original de 24 clases a la nueva de 5 macrogrupos anteriormente expuesta.

4.1.1. Origen

El dataset ha sido extraído de un reto de Kaggle [1], que a su vez es una extensión de uno del *Institute of Computer Graphics and Vision at the Graz University of Technology* [71]. Kaggle es una plataforma en línea donde perfiles del campo de la ciencia, como científicos de datos, ingenieros etc. colaboran, aprenden y participan en competiciones dando solución a diversos retos propuestos en la plataforma. De forma resumida, es un entorno colaborativo, donde se proponen retos, se aportan soluciones y se comparten hallazgos entre su comunidad.

4.1.2. Formato

El conjunto de datos sigue una arquitectura de carpetas. En la denominada *binary_dataset*, se encuentra el conjunto de datos para la segmentación binaria. La etiquetada con *classes_dataset* contiene las imágenes segmentadas, así como las imágenes originales redimensionadas. La última, *semantic-drone-dataset*, que contiene el conjunto de datos multiclase original. A su vez, ésta se divide en dos directorios: *label_images_semantic* que almacena las imágenes Ground Truth asociadas a las imágenes tomadas por el dron. Por su parte, el segundo directorio, *original_images*; cuyo contenido son las imágenes urbanas reales en la resolución original, en formato png y jpg respectivamente. Ambos tipos de imagen presentan unas dimensiones originales de 6000×4000 píxeles. Asimismo, se proporciona un archivo *Excel*, que consiste en una leyenda con los colores utilizados en las imágenes segmentadas y un archivo *.txt* con el mapeo de los píxeles correspondiente..

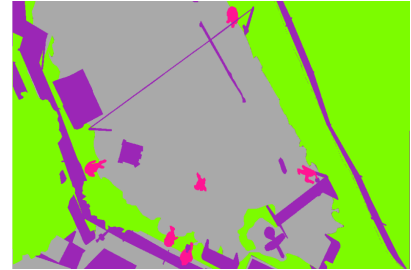


Figura 4.1: Imagen original.

Figura 4.2: Ground Truth.

Figura 4.3: Imagen segmentada.

Figura 4.4: Contenido de los directorios del conjunto de datos.

4.1.3. Etiquetado y distribución

Para contextualizar la información presentada en esta sección sobre los datos y su distribución, es crucial tener en consideración la transformación llevada a cabo previamente. Esta consiste en la conversión de las 24 clases de imágenes originales a los 5 macrogrupos definidos. Los resultados que se exponen a continuación se corresponden a estas 5 clases resultantes de dicha transformación.

El conjunto de datos consta originalmente de 400 imágenes, tras la aplicación de técnicas de aumento de datos, obtenemos 12.800, de las cuales 9.600 se destinan al conjunto de entrenamiento y 3.200 al de prueba. La distribución de clases se mantiene consistente, ya que se aplican las mismas transformaciones a todas las imágenes y sus correspondientes Ground Truth de manera uniforme.

He generado un gráfico en Python que visualiza de manera clara la distribución de las clases en el conjunto de datos utilizado. Con el propósito de minimizar posibles irregularidades provocadas por el conteo de píxeles aislados con valores incorrectos, se ha optado por incrementar en uno la cuenta correspondiente a una clase, en particular, cuando la imagen en consideración presente al menos 200 píxeles de dicha clase. Esta estrategia busca tener en cuenta regiones representativas en el conjunto de datos, en las que el modelo pueda identificar patrones y características relevantes.

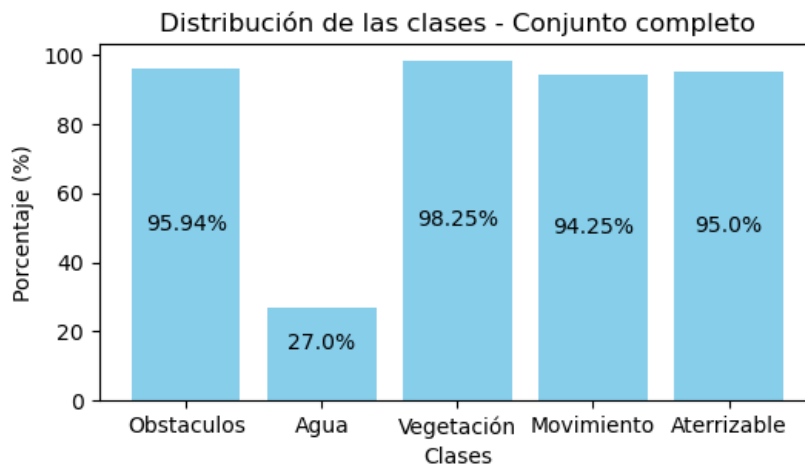


Figura 4.5: Distribución de clases del conjunto de datos completo

Los resultados revelan una distribución equitativa en la mayoría de las clases. Cabe destacar que, aunque la clase Agua se exhibe en un menor número de imágenes en comparación con otras clases, tal disparidad no compromete la capacidad del modelo para identificar y caracterizar dichas zonas. No se considera necesario aplicar correcciones adicionales, ya que la representatividad de la clase Agua en el conjunto de datos es suficiente para ser identificada con precisión por el modelo.

De igual manera, se han generado dos gráficos (**Figuras 4.6 4.7**) para visualizar la distribución de clases en los conjuntos de entrenamiento (train) y prueba (test) respectivamente. Esta representación gráfica nos permite

verificar el equilibrio en la distribución tras la aleatoriedad introducida en la generación de ambos conjuntos. En particular, se busca prevenir el desequilibrio significativo entre clases, lo que podría afectar negativamente a la capacidad de generalización del modelo.

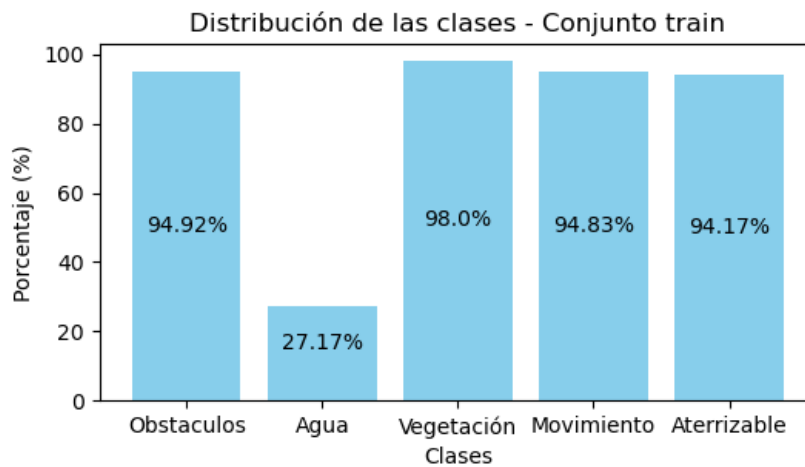


Figura 4.6: Distribución de clases del conjunto de entrenamiento

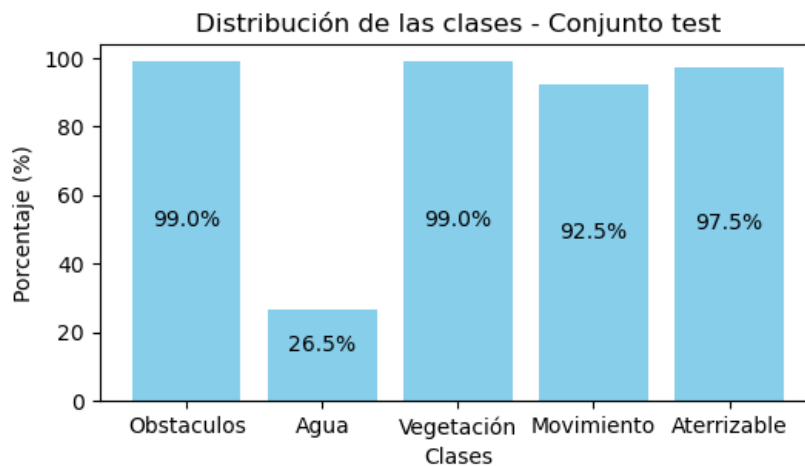


Figura 4.7: Distribución de clases del conjunto de prueba

Según la representación gráfica anterior, podemos observar que las proporciones en los conjuntos de entrenamiento y prueba, muestran una consistencia notoria con la distribución presente en el conjunto de datos completo. Esto sugiere que la variabilidad introducida por la aleatoriedad en la generación de ambos conjuntos, no afecta de manera significativa al entrenamiento del modelo, ya que las proporciones se mantienen de manera estable y coherente.

4.2. Tratamiento sobre el conjunto

En esta sección se exponen las distintas transformaciones aplicadas al conjunto de datos original. Este proceso se divide en dos etapas distintas: una fase inicial de preprocesamiento y una posterior de aumento del número de imágenes mediante técnicas de Data Augmentation.

El propósito de estas transformaciones radica en la adaptación del conjunto de datos a nuestras necesidades específicas, lo que resulta esencial para asegurar la eficacia y la calidad del proceso de aprendizaje del modelo.

4.2.1. Preprocesamiento

En la fase de preprocesamiento, se llevan a cabo una serie de tareas destinadas a la modificación de las imágenes originales, que incluyen el proceso de renombrado de las imágenes, que resulta fundamental para facilitar la posterior división del conjunto de datos en conjuntos de entrenamiento y prueba, así como la redimensión de las mismas a un tamaño óptimo para que sirvan como entradas de la Red Neuronal.

Renombrado del conjunto de datos

Inicialmente, las imágenes del conjunto se encontraban nombradas con números no consecutivos siguiendo el patrón 'xxx.png' o 'xxx.jpg', dependiendo de si se trataba de un Ground Truth o una imagen respectivamente. Donde 'xxx' representa el número de la imagen. Dado que el conjunto original constaba de 400 imágenes, se procedió a renombrarlas desde '000.png' hasta '399.png'. Este proceso se llevó a cabo con el propósito de establecer una secuencia numérica consecutiva en los nombres de las imágenes, lo que iba a ayudar significativamente a simplificar y aleatorizar el posterior proceso de separación entre los conjuntos de entrenamiento y prueba.

A pesar de que las imágenes no tenían una nomenclatura numérica continua, es importante destacar que estaban ordenadas de manera ascendente, lo que facilitó el proceso de renombrado. Como se puede observar en el **Código 4.1**, se inició el proceso creando una lista que contenía ordenados numéricamente todos los archivos del directorio proporcionado como parámetro.

Para llevar a cabo el renombrado, se implementó un contador que se incrementa en una unidad en cada iteración al procesar una imagen. Utilizando este contador, se generó el nuevo nombre de archivo siguiendo el patrón anterior y posteriormente se copió de la imagen en la carpeta destino.

La decisión de realizar este proceso en una carpeta distinta a la original se tomó con la finalidad de preservar los datos originales en todo momento, evitando cualquier riesgo de sobrescribirlos y asegurando la disponibilidad de una copia de respaldo. Por lo tanto, se adoptó la práctica de trabajar exclusivamente con directorios distintos al de origen durante todo el proceso de preprocesamiento de los datos.

```
archivos = sorted(os.listdir(carpeta_origen))

# Contador para el nuevo nombre
contador = 0

# Iterar sobre los archivos y renombrarlos, de forma consecutiva
for archivo in archivos:
    if archivo.endswith((".jpg", ".png")):
        nuevo_nombre = f"{contador:03d}.png"
        ruta_origen = os.path.join(carpeta_origen, archivo)
        ruta_destino = os.path.join(carpeta_destino, nuevo_nombre)
        shutil.copy(ruta_origen, ruta_destino)
        contador += 1
```

Código 4.1: Función renombrado

Conjunto de entrenamiento, test, mapeo y redimensión

Se describe el proceso de generación del conjunto de entrenamiento y prueba, aprovechando la transformación de renombrado. Se ha optado por dividir el conjunto de datos en un 75 % para entrenamiento y un 25 % para prueba. Aplicando estos porcentajes a las 400 imágenes originales, se obtienen 300 para el conjunto de entrenamiento y 100 para el conjunto de prueba.

Para lograr una división aleatoria, se han generado dos conjuntos de índices disjuntos, con valores aleatorios en el rango de 0 a 399. 300 índices de ellos para el conjunto de entrenamiento, y 100 para el de prueba. En el fragmento de **Código 4.2** podemos observar cómo se generan ambos conjuntos. La creación de estos se lleva a cabo extrayendo el número de cada una de las imágenes y verificando en qué conjunto de índices se encuentra, lo que permite generar ambas particiones de manera aleatoria y equitativa.

```
total = 400
train_len = int(total*0.75)
test_len = int(total*0.25)
```

```
all_indices = list(range(400))
idx_train = random.sample(all_indices, train_len)

for idx in idx_train:
    all_indices.remove(idx)

idx_test = random.sample(all_indices, test_len)
```

Código 4.2: Creación índices para los conjuntos

Tras la generación de las listas de índices disjuntas, llamamos a la función que se encarga de almacenar las imágenes en la carpeta correspondiente realizando la distinción entre train o test, la redimensión de éstas y el mapeo correspondiente de los píxeles de las imágenes Ground Truth.

El siguiente código es el encargado de realizar el proceso de separación entre conjunto de entrenamiento y prueba. Según sea en el que se encuentre el índice, se asigna una ruta destino u otra; de esta manera, se consigue que las imágenes tengan como destino el directorio correcto.

```
for i in range(400):
    if i in idx_train:
        dest_images = dest_train_images
        dest_labels = dest_train_label
    elif i in idx_test:
        dest_images = dest_test_images
        dest_labels = dest_test_label
    else:
        continue
```

Código 4.3: Separación conjuntos

Una vez determinadas las rutas de destino para las imágenes, se procede a la redimensión de las mismas. Inicialmente presentan una resolución de 6000x4000 píxeles, pero se redimensionan y pasan a tener una resolución de 256x256 píxeles. Este cambio de tamaño permite que las imágenes sirvan como entrada al modelo, así como un manejo más eficiente de las mismas durante el proceso de entrenamiento y prueba del modelo de Aprendizaje Automático.

Asimismo, las imágenes de Ground Truth experimentan un mapeo de píxeles, como ya hemos comentado. Este proceso se realiza a través de dos diccionarios. El primero, denominado intermedio, se encarga de convertir todos los valores de píxeles de una misma clase a un valor único y distinto. Posteriormente, este valor se mapea al valor real de la clase mediante el segundo diccionario. Este valor de clase final está en el rango de 0 a 4, inclusive.

```
map_pixels_intermediate = {
    0: 24, 6: 24, 10: 24, 11: 24, 12: 24, 13: 24, 14: 24, 21: 24, 22: 24, 23: 24,
    5: 25, 7: 25,
    2: 26, 3: 26, 8: 26, 19: 26, 20: 26,
    15: 27, 16: 27, 17: 27, 18: 27,
    1: 28, 4: 28, 9: 28
}

map_pixels_final = {
    24: 0,
    25: 1,
    26: 2,
    27: 3,
    28: 4
}
```

Código 4.4: Diccionarios de mapeo

```
img = Image.open(os.path.join(origen_images, f"{i:03d}.png"))
img = img.resize((256, 256), Image.NEAREST)
```

```

img.save(os.path.join(dest_images, f"{i:03d}.png"))

mask = Image.open(os.path.join(origen_label, f"{i:03d}.png"))
mask_np = np.array(mask)

# Mapea las imagenes Ground Truth
for idx, j in map_pixels_intermediate.items():
    # Sustituimos por los valores
    mask_np[mask_np==idx] = j

for idx, j in map_pixels_final.items():
    # Sustituimos por los valores
    mask_np[mask_np==idx] = j

# Converitmos de array a imagen PIL
mask_remap = Image.fromarray(mask_np)
mask_remap = mask_remap.resize((256,256), Image.NEAREST)
mask_remap.save(os.path.join(dest_labels, f"{i:03d}.png"))

```

Código 4.5: Redimensión y mapeo Ground Truth

Ahora, con todas las imágenes preprocesadas y listas, aplicamos las técnicas de Data Augmentation que nos permiten incrementar el tamaño de nuestro conjunto de imágenes.

4.2.2. Data Augmentation

El proceso de aumento de datos nos permite transformar un conjunto inicial de 400 imágenes en un total de 12.800. Para lograr esto, emplearemos técnicas de Data Augmentation, como la creación de imágenes espejo, rotaciones y la introducción de variabilidad en el contraste y el brillo de las imágenes. Esto lo conseguimos haciendo uso de la biblioteca OpenCV (cv2), que nos brinda un conjunto de herramientas para la manipulación de imágenes. En el siguiente fragmento de código podemos ver cómo se han aplicado las mencionadas transformaciones para obtener el conjunto final de imágenes.

```

# Imagen espejo
img_mirror = cv2.flip(img, 1)
cv2.imwrite(os.path.join(image_dir, f"{number}_mirror.png"), img_mirror)

# Rotaciones
for angle in angles:
    if is_mask:
        interpolation = cv2.INTER_NEAREST
    else:
        interpolation = cv2.INTER_LINEAR

    M = cv2.getRotationMatrix2D((int(img.shape[1]/2), int(img.shape[0]/2)), angle, 1)
    img_rot = cv2.warpAffine(img, M, (img.shape[1], img.shape[0]),
        flags=interpolation, borderMode=cv2.BORDER_REFLECT)
    cv2.imwrite(os.path.join(image_dir, f"{number}_rot{angle}.png"), img_rot)

# Variabilidad imagen
if is_mask == False:
    img_bc = cv2.convertScaleAbs(img, alpha=1.3, beta=10)
elif is_mask == True:
    img_bc = img

```

Código 4.6: Redimensión y mapeo Ground Truth

Estas operaciones nos permiten generar 32 imágenes distintas a partir de una original. Para ser más precisos, se crea una imagen espejo y se generan dos más con variaciones en el contraste y el brillo, una para la original y, otra, para la imagen espejo. Esto nos da un total de 4 imágenes a partir de una sola. Adicionalmente, para cada una de estas 4 imágenes, aplicamos 7 rotaciones en incrementos de 45° , lo que resulta en 28 imágenes rotadas en total. Sumandolas todas llegamos a un total de 32 imágenes por cada original.

En la generación de las imágenes rotadas, aplicamos dos métodos diferentes de interpolación: `INTER_NEAREST` para las imágenes Ground Truth y `INTER_LINEAR` para las imágenes originales. El primero aplica el método del vecino más cercano, evitando así que se generen diferentes y nuevos valores para los píxeles, asegurándonos que los valores de los píxeles de la imagen Ground Truth siguen siendo valores de clase. Sin embargo, `INTER_LINEAR` aplica una interpolación bilineal, que es más lenta, lo que se traduce en una menor pérdida de información, consiguiendo mantener así los detalles en la imagen original. De esta forma, mantenemos los valores en las imágenes Ground Truth y reducimos la pérdida de información, conservando los detalles de la original.

Al realizar rotaciones en ciertos ángulos sobre las imágenes, es importante considerar que, en las esquinas, perdemos información valiosa de los valores de los píxeles. Para abordar este problema y garantizar que estas áreas no queden en negro generando píxeles vacíos, empleamos una estrategia eficaz en nuestro proceso: la opción `borderMode=cv2.BORDER_REFLECT` de la biblioteca OpenCV.

Esta configuración especial nos permite aplicar un efecto de espejo en los píxeles de las esquinas durante la rotación. De esta manera, los píxeles en las esquinas reflejan los valores de los píxeles correspondientes en la imagen original. Al hacerlo, conseguimos asignar valores apropiados a estas áreas, lo que contribuye a enriquecer el aprendizaje de nuestro modelo. Asimismo, esta técnica mantiene la correlación esencial con la información de Ground Truth de nuestras imágenes.

A modo de resumen, aplicando estas técnicas a las 400 imágenes originales, generamos un total de 12.800; que se encuentran desglosadas de la siguiente manera: 400 imágenes originales, más 1.600 imágenes espejo y variadas. A cada una de estas últimas, se les aplican 7 rotaciones, lo que da como resultado 11.200 imágenes adicionales. Al sumar todas estas categorías, obtenemos 12.800 imágenes para utilizar en nuestro conjunto.

4.3. Carga de datos

Carga Total

Una opción para la carga de datos en el entrenamiento de modelos de Deep Learning es intentar tener todas las imágenes simultáneamente en la memoria. Con este enfoque, evitamos que se realicen lecturas de disco, que son más lentas, lo que se traduce directamente en una aceleración en el proceso de entrenamiento. Sin embargo, hay varios factores que hacen que este método no sea viable en la práctica.

El principal de ellos, que por sí sólo hace inviable esta técnica es la limitación de memoria RAM. El conjunto de datos no es el único recurso que consume memoria durante el entrenamiento, también hay que tener en cuenta los parámetros del modelo, gradientes de la red y otros procesos que se están ejecutando simultáneamente en la máquina, como el navegador.

Incluso con el uso de Data Loaders, que veremos en el siguiente punto, si no se designa un tamaño de lote minuciosamente, el proceso de entrenamiento puede llegar a detenerse por falta de memoria. Por lo tanto, la carga total de datos en memoria no resulta viable y se opta por el uso de Data Loaders.

A pesar de que el tiempo de entrenamiento del modelo puede representar una restricción dentro de un contexto académico y para un Trabajo de Fin de Grado, en algunas etapas del proceso he tenido acceso a una GPU de alto rendimiento. Esto me ha permitido realizar pruebas y generar el modelo definitivo de manera mucho más rápida que en una CPU. Por lo tanto, aunque el uso de Data Loaders puede producir algún retraso en el tiempo de entrenamiento, no es considerable, ya que se compensa con el aumento de velocidad proporcionado por la GPU.

Data Loader

Para asegurar que nuestro modelo pueda acceder a las imágenes almacenadas en el disco de la máquina virtual, es necesario transferirlas de la memoria de almacenamiento al espacio de trabajo en RAM. Dado que los conjuntos de datos utilizados en el Aprendizaje Automático suelen ser pesados, no podemos cargar todas las imágenes de forma simultánea, como se ha comentado en el punto anterior. Para abordar este desafío de manera eficiente, utilizamos un `DataLoader` de PyTorch.

Un `DataLoader` de PyTorch es un objeto iterable, que permite realizar una carga por lotes para el entrenamiento de un modelo, proporcionando así flexibilidad y eficiencia para la carga de los datos. Este `DataLoader` se crea a partir de un objeto `Dataset`, que gestiona el conjunto de datos y proporciona una interfaz para acceder a él. El `DataLoader`, por otro lado, toma este `Dataset` y se encarga de generar lotes de datos para su procesamiento.

Adicionalmente, ofrece funcionalidades adicionales, como la aleatorización de los datos y la posibilidad de aplicar una cadena de transformaciones a cada lote de datos cargado.

Antes de comentar cómo se realiza la carga de los datos utilizando `DataLoader`, voy a hablar de clase `UNetDataset`, que hereda de la clase `Dataset` de PyTorch. Esta clase se utiliza para cargar y procesar las imágenes y Ground Truth correspondientes para nuestro modelo.

```
train_dataset = UNetDataset(train_img_folder, train_label_folder)
test_dataset = UNetDataset(test_img_folder, test_label_folder)

class UNetDataset(Dataset):
    def __init__(self, img_dir, mask_dir):
        self.img_dir = img_dir
        self.mask_dir = mask_dir
        self.to_tensor = ToTensor()

        # Lista con todos los nombres de los archivos
        # Los nombres de las imagenes y de Ground Truth coinciden
        self.images = os.listdir(img_dir)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        # Generamos la ruta exacta a cada una de las imagenes
        img_path = os.path.join(self.img_dir, self.images[idx])
        mask_path = os.path.join(self.mask_dir, self.images[idx])

        # Abrimos las imagenes
        image = np.array(Image.open(img_path).convert("RGB")) / 255.0
        mask = np.array(Image.open(mask_path).convert("L"), dtype=np.int32)

        # Aplicamos la transformaci n ToTensor para convertir en tensores
        image = self.to_tensor(image)
        mask = self.to_tensor(mask)

        image = image.permute(1, 2, 0)
        mask = mask.squeeze(0)

        return image, mask
```

Código 4.7: Clase UNetDataset

En el código anterior, se observan una serie de transformaciones. La principal de ellas es la conversión a tensor, necesaria para que las imágenes puedan ser procesadas por la red.

Asimismo, se realiza una normalización de los valores de los píxeles de las imágenes originales. Esto se logra dividiendo cada valor de píxel entre 255, lo que escala los valores al rango [0,1]. Este paso es crucial para asegurar que la Red Neuronal pueda procesar eficientemente los datos de la imagen. Es importante destacar, que esta normalización no se aplica a las imágenes de Ground Truth, puesto que queremos mantener los valores originales de los píxeles, ya que son valores asociados con una clase. Por lo tanto, alterar estos valores podría afectar la precisión de nuestro modelo.

El **Código 4.8** muestra cómo se crean los `DataLoader`, que toman como entrada la salida de la clase `UNetDataset`, donde es importante destacar una serie de parámetros:

- **batch_size:** tamaño del lote en cada una de las iteraciones. Hemos elegido uno adecuado, para que las imágenes puedan cargarse en la memoria RAM, y se puedan obtener buenos resultados en el rendimiento y generalización del modelo.
- **shuffle = True:** permite que los datos no se carguen de manera secuencial, siguiendo un orden predefinido, dotando de aleatoriedad a la carga de las mismas. Esta característica es altamente beneficiosa en nuestro caso, ya que después de aplicar las transformaciones, las primeras 32 imágenes son variaciones ligeras de la

misma imagen. Si no se aleatorizaran los datos, existiría poca variación en dos lotes consecutivos, puesto que las 16 imágenes del lote serían variaciones de la misma imagen, lo que perjudicaría considerablemente el entrenamiento del modelo. Gracias a esta permutación de los datos, podemos garantizar una mayor variación en las imágenes de cada lote, lo que contribuye a un entrenamiento más efectivo y a un modelo más robusto.

- **num_workers:** constituye el número de procesos que se emplean en la carga de los datos a memoria. He fijado su valor a ocho, para que coincida con el número de procesadores de la máquina virtual de la que disponemos.

```
train_loader = DataLoader(train_dataset , batch_size=tam_lote , shuffle=True , num_workers= 8)
test_loader = DataLoader(test_dataset , batch_size=tam_lote , shuffle=True , num_workers= 8)
```

Código 4.8: Creación de los DataLoaders

Dado que la cantidad de imágenes no es excesivamente elevada, como puede ser en otros proyectos de Deep Learning, podemos establecer un tamaño de lote adecuado y no demasiado grande. A pesar de haber elegido uno relativamente pequeño, gracias a disponer de una GPU de alto rendimiento y un conjunto de datos ajustado, hemos podido llevar a cabo pruebas de diferentes entrenamientos en un plazo razonable de tiempo.

Después de haber detallado minuciosamente las transformaciones aplicadas al conjunto de datos inicial en este capítulo, hemos logrado preparar un conjunto de datos organizado en lotes, dispuesto para su implementación en la siguiente fase del proyecto. Este capítulo ha desempeñado un papel crucial en la etapa experimental, ya que la preparación y adecuación de los datos son elementos fundamentales para optimizar el rendimiento y la generalización de los modelos de Aprendizaje Profundo.

En el próximo Capítulo, nos adentraremos en la construcción, arquitectura, entrenamiento y evaluación del modelo diseñado. Este paso es fundamental para comprender cómo los datos, previamente transformados y estructurados, convergen en la creación de un modelo robusto.

Capítulo 5

Desarrollo completo del modelo

En este Capítulo realizaremos un análisis exhaustivo de las decisiones que conforman la estructura final de nuestro modelo y el razonamiento detrás de estas elecciones. Para ello, nos vamos a centrar en la utilización de PyTorch, en particular, su paquete *nn*, el cuál nos proporciona múltiples opciones para la construcción de la red. Examinaremos de forma detallada la arquitectura de nuestro modelo, los procedimientos para su entrenamiento, optimización y las clases empleadas.

5.1. Construcción

En esta sección, proporcionamos una explicación detallada de las clases empleadas y los parámetros ajustados durante la construcción del modelo, sin adentrarnos en detalles técnicos sobre las operaciones específicas de cada clase, ya que esa información se encuentra en el **Apéndice D**.

Con el fin de ofrecer un desglose más estructurado, dividiremos este apartado en dos subsecciones: *Downsampling* y *Upsampling*. En la primera subsección, cubrimos las clases utilizadas y los parámetros modificados en la fase de reducción de dimensiones de la red. Mientras que, en la segunda subsección, nos enfocaremos en las clases y parámetros relevantes para la etapa de aumento de dimensiones.

5.1.1. Downsampling - Fase de extracción

5.1.1.1. `torch.nn.Conv2d`

Lleva a cabo una convolución 2D en una entrada que consta de múltiples canales. Los parámetros empleados en la construcción del modelo toman los siguientes valores:

- **in_channels y out_channels.** Dependen de la profundidad de la capa. Inicialmente toma el valor 3, dado que las imágenes tienen 3 canales de información, e incrementa tomando como valor potencias de 2, hasta alcanzar 1024, en el punto más profundo de la red.
- **kernel_size.** El valor que utilizamos es 3, puesto que queremos mantener las convoluciones 3x3.
- **stride.** Para preservar la información espacial y aumentar la capacidad de detección de patrones complejos, usamos un stride de 1.
- **padding.** Aplicamos same padding para preservar los detalles, dando el valor "same" al parámetro.

5.1.1.2. `torch.nn.BatchNorm2d`

Realiza normalización por lotes a una entrada mini-lote de entradas 2D. Únicamente emplea un parámetro *out_channels* que indica cuántos canales tiene la salida de la capa anterior, y esta capa de normalización por lotes normalizará estos canales durante el entrenamiento.

5.1.1.3. `torch.nn.LeakyRelu`

Aplica la función de activación LeakyReLU. En el modelo desarrollado se aplica después de realizar la convolución y la normalización por lotes.

Modificamos dos parámetros:

- **negative_slope.** El valor del coeficiente rectificador que utilizamos es 0.01, mantenemos el que viene por defecto.
- **inplace.** Habilitamos este parámetro booleano, **true**. Ahora la operación se realiza directamente en el tensor de entrada sin crear un nuevo tensor para almacenar el resultado. De esta forma, ahorramos memoria durante el entrenamiento del modelo.

5.1.1.4. `torch.nn.MaxPool2d`

Implementa la operación Max Pooling (**Sección 3.3.4**). Se ha optado por emplear esta técnica en la capa de agrupación, porque permite extraer las características más dominantes dentro de la imagen.

Los valores elegidos para algunos de los parámetros son:

- **kernel_size y stride.** Ambos parámetros se han fijado en el valor de 2. Esta elección ha sido realizada con el propósito de disminuir a la mitad las dimensiones de los mapas de características y de dotar al modelo una mayor robustez proporcionando invarianza frente a pequeñas traslaciones.
- **padding.** No añadimos padding, para mantener la reducción de dimensiones lograda con el tamaño del kernel.
- **return_indices.** Anulamos el retorno de los índices, ya que no necesitamos conocer las posiciones de los valores a la hora de realizar el Upsampling.

5.1.2. Upsampling - Fase de expansión

5.1.2.1. `torch.nn.ConvTranspose2d`

Se ha optado por implementar la convolución transpuesta como técnica de aumento de resolución en la fase de Upsampling del modelo.

Los parámetros importantes son muy similares a los de la convolución normal, a excepción de dos:

- **kernel_size y stride:** asignamos el valor 2 a ambos parámetros. Al usar los mismos valores en la capa de agrupación y en la de expansión mantenemos la correspondencia y la simetría, características clave de la arquitectura UNET.

5.1.2.2. `torch.nn.Dropout`

Antes de la última capa convolucional en nuestro modelo, hemos incorporado la técnica de dropout. Esta consiste en suprimir la actualización de pesos en un porcentaje específico de neuronas, seleccionadas aleatoriamente, durante el proceso de entrenamiento. El propósito principal de esta estrategia es mitigar el sobreajuste y acelerar el tiempo de entrenamiento del modelo.

El parámetro modificado es la probabilidad de inhibición, para encontrar valor óptimo se realizaron múltiples entrenamientos como veremos en la **Sección 7.1**. Al final se seleccionó 0.3 como valor final, lo que quiere decir que únicamente el 70 % de las neuronas del modelo actualizan sus pesos en la última capa.

5.1.2.3. `torch.cat`

Permite concatenar una secuencia de tensores, se ha utilizado para implementar las conexiones de salto, permitiendo fusionar la salida de la convolución transpuesta con la salida correspondiente del bloque de submuestreo descendente.

Los parámetros modificados son:

- **tensors:** secuencia `[x, skip_input]`, donde la `x` representa la salida de la convolución transpuesta y `skip_input`, la salida del bloque de contracción.
- **dim:** toma el valor 1, para que concatene la primera dimensión de los tensores.

5.1.3. Arquitectura de la red

La arquitectura elegida para el modelo se inspira fielmente en la arquitectura original presentada en la **Sección 3.4.1**. Antes de comenzar con la arquitectura elegida, voy a exponer las diferencias y similitudes de nuestro modelo con el original.

Al tratarse de un modelo U-Net, mantenemos la estructura codificador-decodificador. En la ruta de contracción construimos cuatro bloques convolucionales, de la misma forma que ocurre en la estructura original. Cada bloque de muestreo descendente consta de dos capas convolucionales seguidas de una capa de agrupación máxima. Estas capas convolucionales aplican kernels de tamaño 3x3, un stride de 1, y same-padding, como hemos visto. Después de cada capa convolucional se realiza, normalización por lotes y se aplica la función de activación Leaky Relu. Tras el bloque convolucional se aplica una capa de agrupación máxima que reduce las dimensiones de los mapas de características de entrada a la mitad.

La base de la U está constituida por el 'cuello de botella', formado por dos capas convolucionales. Esta parte de la red nos permite extraer características en la resolución más baja, antes de la reconstrucción, por lo que he considerado importante mantenerla.

En la ruta de expansión vemos de nuevo cuatro bloques de muestreo ascendente, manteniendo así la forma de 'U' y siguiendo la correspondencia con la fase de submuestreo. Se emplea la convolución transpuesta (Punto 3.3.5) como técnica de aumento en las dimensiones espaciales de la entrada. Se ha optado por utilizar la convolución transpuesta debido a su eficacia en el incremento de la resolución espacial de la imagen, ya que mantiene detalles y efectúa una reconstrucción precisa de la misma.

Para implementar las 'conexiones de salto' concatenamos la salida de esta convolución transpuesta con la salida correspondiente del bloque de submuestreo descendente.

Finalmente, el modelo termina aplicando una capa convolucional que nos permite generar tantos planos de salida como clases hay que identificar, realizando la tarea de segmentación planteada.

A pesar de la similitud que presenta respecto al modelo original, hemos añadido la normalización por lotes después de cada capa convolucional ya que mejora el rendimiento. Asimismo, justo antes de la última convolución se realiza un dropout del 30 % de las neuronas.

Tras esta breve explicación sobre las decisiones tomadas en cada una de las partes del modelo, podemos visualizar la configuración del modelo con la función `summary()`, del módulo `torchsummary`, que nos ofrece un resumen de las diferentes capas, salidas y parámetros de la red. Al tratar con un modelo UNET con un gran número de capas, he dividido la salida obtenida en dos. En la **Figura 5.2** podemos ver el resumen para la ruta de contracción y cuello de botella.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 256, 256]	1,792
BatchNorm2d-2	[-1, 64, 256, 256]	128
LeakyReLU-3	[-1, 64, 256, 256]	0
Conv2d-4	[-1, 64, 256, 256]	36,928
BatchNorm2d-5	[-1, 64, 256, 256]	128
LeakyReLU-6	[-1, 64, 256, 256]	0
BlockConv-7	[-1, 64, 256, 256]	0
MaxPool2d-8	[-1, 64, 128, 128]	0
DownsamplingBlock-9	[[[-1, 64, 128, 128], [-1, 64, 256, 256]]]	
Conv2d-10	[-1, 128, 128, 128]	73,856
BatchNorm2d-11	[-1, 128, 128, 128]	256
LeakyReLU-12	[-1, 128, 128, 128]	0
Conv2d-13	[-1, 128, 128, 128]	147,584
BatchNorm2d-14	[-1, 128, 128, 128]	256
LeakyReLU-15	[-1, 128, 128, 128]	0
BlockConv-16	[-1, 128, 128, 128]	0
MaxPool2d-17	[-1, 128, 64, 64]	0
DownsamplingBlock-18	[[[-1, 128, 64, 64], [-1, 128, 128, 128]]]	
Conv2d-19	[-1, 256, 64, 64]	295,168
BatchNorm2d-20	[-1, 256, 64, 64]	512
LeakyReLU-21	[-1, 256, 64, 64]	0
Conv2d-22	[-1, 256, 64, 64]	590,080
BatchNorm2d-23	[-1, 256, 64, 64]	512
LeakyReLU-24	[-1, 256, 64, 64]	0
BlockConv-25	[-1, 256, 64, 64]	0
MaxPool2d-26	[-1, 256, 32, 32]	0
DownsamplingBlock-27	[[[-1, 256, 32, 32], [-1, 256, 64, 64]]]	
Conv2d-28	[-1, 512, 32, 32]	1,180,160
BatchNorm2d-29	[-1, 512, 32, 32]	1,024
LeakyReLU-30	[-1, 512, 32, 32]	0
Conv2d-31	[-1, 512, 32, 32]	2,359,808
BatchNorm2d-32	[-1, 512, 32, 32]	1,024
LeakyReLU-33	[-1, 512, 32, 32]	0
BlockConv-34	[-1, 512, 32, 32]	0
MaxPool2d-35	[-1, 512, 16, 16]	0
DownsamplingBlock-36	[[[-1, 512, 16, 16], [-1, 512, 32, 32]]]	
Conv2d-37	[-1, 1024, 16, 16]	4,719,616
BatchNorm2d-38	[-1, 1024, 16, 16]	2,048
LeakyReLU-39	[-1, 1024, 16, 16]	0
Conv2d-40	[-1, 1024, 16, 16]	9,438,208
BatchNorm2d-41	[-1, 1024, 16, 16]	2,048
LeakyReLU-42	[-1, 1024, 16, 16]	0
BlockConv-43	[-1, 1024, 16, 16]	0
ConvTranspose2d-44	[-1, 1024, 32, 32]	4,195,328
Conv2d-45	[-1, 512, 32, 32]	7,078,400
BatchNorm2d-46	[-1, 512, 32, 32]	1,024
LeakyReLU-47	[-1, 512, 32, 32]	0
Conv2d-48	[-1, 512, 32, 32]	2,359,808
BatchNorm2d-49	[-1, 512, 32, 32]	1,024
LeakyReLU-50	[-1, 512, 32, 32]	0

Figura 5.1: Resumen del modelo. Parte 1

La parte correspondiente a la ruta de expansión y la convolución final se ve en la siguiente figura:

LeakyReLU-50	[-1, 512, 32, 32]	0
BlockConv-51	[-1, 512, 32, 32]	0
UpsamplingBlock-52	[-1, 512, 32, 32]	0
ConvTranspose2d-53	[-1, 512, 64, 64]	1,049,088
Conv2d-54	[-1, 256, 64, 64]	1,769,728
BatchNorm2d-55	[-1, 256, 64, 64]	512
LeakyReLU-56	[-1, 256, 64, 64]	0
Conv2d-57	[-1, 256, 64, 64]	590,080
BatchNorm2d-58	[-1, 256, 64, 64]	512
LeakyReLU-59	[-1, 256, 64, 64]	0
BlockConv-60	[-1, 256, 64, 64]	0
UpsamplingBlock-61	[-1, 256, 64, 64]	0
ConvTranspose2d-62	[-1, 256, 128, 128]	262,400
Conv2d-63	[-1, 128, 128, 128]	442,496
BatchNorm2d-64	[-1, 128, 128, 128]	256
LeakyReLU-65	[-1, 128, 128, 128]	0
Conv2d-66	[-1, 128, 128, 128]	147,584
BatchNorm2d-67	[-1, 128, 128, 128]	256
LeakyReLU-68	[-1, 128, 128, 128]	0
BlockConv-69	[-1, 128, 128, 128]	0
UpsamplingBlock-70	[-1, 128, 128, 128]	0
ConvTranspose2d-71	[-1, 128, 256, 256]	65,664
Conv2d-72	[-1, 64, 256, 256]	110,656
BatchNorm2d-73	[-1, 64, 256, 256]	128
LeakyReLU-74	[-1, 64, 256, 256]	0
Conv2d-75	[-1, 64, 256, 256]	36,928
BatchNorm2d-76	[-1, 64, 256, 256]	128
LeakyReLU-77	[-1, 64, 256, 256]	0
BlockConv-78	[-1, 64, 256, 256]	0
UpsamplingBlock-79	[-1, 64, 256, 256]	0
Dropout-80	[-1, 64, 256, 256]	0
Conv2d-81	[-1, 5, 256, 256]	325
=====		
Total params: 36,963,461		
Trainable params: 36,963,461		
Non-trainable params: 0		

Figura 5.2: Resumen del modelo. Parte 2

En esta segunda parte del resumen, podemos ver que `torchsummary` nos proporciona el número total de parámetros de la red, aproximadamente 37 millones. Para evaluar la adecuación de nuestro modelo, es esencial considerar la proporción entre los parámetros de la Red Neuronal y la cantidad total de píxeles procesados. En nuestro caso, el resultado de esta proporción es 0.01470, indicando que la red cuenta con un número de parámetros razonable en relación con la cantidad de píxeles que se manejan.

Tras haber abordado las decisiones fundamentales relativas a la arquitectura del modelo, así como las justificaciones subyacentes a cada elección, procederemos, en la siguiente subsección, a detallar la implementación concreta de dichas decisiones.

5.1.4. Implementación de la red

Se han definido cuatro clases que colaboran para crear la estructura de nuestra U-Net. En el **Código 5.1**, se presenta la clase `BlockConv`, que hereda de `nn.Module` y representa un bloque convolucional en nuestro modelo. Cada bloque está compuesto por dos capas convolucionales, después, como ya comenté en la sección anterior, se realiza la normalización por lotes y se aplica la función de activación.

```
class BlockConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(BlockConv, self).__init__()
        self.blockConv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding="same"),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(negative_slope=0.01, inplace=True),
        )

    def forward(self, x):
        return self.blockConv(x)
```

Código 5.1: Clase `BlockConv`

Hemos definido dos clases: `DownsamplingBlock` y `UpsamplingBlock`, ambas heredan nuevamente de `nn.Module`. Éstas se encargan de abordar la fase de submuestreo de características y la de reconstrucción de la imagen respectivamente.

```
class DownsamplingBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DownsamplingBlock, self).__init__()
        self.blockConv = BlockConv(in_channels, out_channels)
        self.down_sample = nn.MaxPool2d(kernel_size=2, stride=2, padding=0,
return_indices=False)

    def forward(self, x):
        skip_out = self.blockConv(x)
        down_out = self.down_sample(skip_out)

        return (down_out, skip_out)
```

Código 5.2: Clase DownsamplingBlock

Como se observa en el código previo, después aplicar un bloque convolucional, se utiliza una capa de agrupación máxima haciendo uso de la clase `nn.MaxPool2d()`. La función `forward()` tras realizar las operaciones devuelve una tupla (`down_out`, `skip_out`), donde `down_out` es la salida después de aplicar el MaxPooling, que se utilizará como entrada para el siguiente bloque convolucional, y `skip_out` es la salida antes de la aplicación del MaxPooling. Esta última, nos permite implementar las conexiones de salto en la fase de upsampling.

La clase `UpsamplingBlock` se encarga de abordar la fase de upsampling. Se introducen dos modificaciones significativas, en comparación con la fase de downsampling. En primer lugar, se realiza la expansión haciendo uso de la clase `nn.ConvTranspose2d` antes de llamar nuevamente el bloque convolucional. En segundo lugar, en la función `forward()`, se implementan conexiones de salto mediante la concatenación entre la salida del bloque de upsampling anterior y el `skip_input` correspondiente de la fase de downsampling.

```
class UpsamplingBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(UpsamplingBlock, self).__init__()

        self.upsamplingBlock = nn.ConvTranspose2d(in_channels-out_channels, in_channels-out_channels, kernel_size=2, stride=2)
        self.double_conv = BlockConv(in_channels, out_channels)

    def forward(self, down_input, skip_input):
        x = self.upsamplingBlock(down_input)
        x = torch.cat([x, skip_input], dim=1)
        return self.double_conv(x)
```

Código 5.3: Clase UpsamplingBlock

La clase que sirve como unión de las anteriormente descritas y que da forma a la arquitectura del modelo se denomina `UNet`. En esta clase, se identifican cuatro componentes clave:

- **Etapas de Downsampling:** compuesta por cuatro bloques convolucionales y sus capas de agrupación.
- **Cuello de botella:** únicamente se realizan dos convoluciones y no se lleva a cabo la agrupación, se hace uso de la clase `BlockConv`, únicamente.
- **Etapas de Upsampling:** nuevamente por cuatro bloques convolucionales para mantener la correspondencia y forma del modelo U, donde previamente se realiza un aumento de la resolución de la imagen.
- **Última convolución:** se realiza una convolución 1×1 , con el objetivo de obtener un vector de características para cada clase que queremos segmentar.

La función `forward()` define cómo se procesan los datos a través de la red, pasando las entradas correspondientes a cada una de las etapas. Asimismo, la clase recibe un único parámetro `'out_classes'`, que, como su nombre indica representa el número de clases a segmentar.

```
class UNet(nn.Module):
    def __init__(self, out_classes):
        super(UNet, self).__init__()

        # Downsampling
        # 3 canales de informacion —> imagenes RGB
        self.down_1 = DownsamplingBlock(3, 64)
        self.down_2 = DownsamplingBlock(64, 128)
        self.down_3 = DownsamplingBlock(128, 256)
        self.down_4 = DownsamplingBlock(256, 512)

        # Cuello de botella
        self.neck_conv = BlockConv(512, 1024)

        # Upsampling
        self.up_4 = UpsamplingBlock(512 + 1024, 512)
        self.up_3 = UpsamplingBlock(256 + 512, 256)
        self.up_2 = UpsamplingBlock(128 + 256, 128)
        self.up_1 = UpsamplingBlock(64 + 128, 64)

        # Dropout
        self.dropout = nn.Dropout(0.3)

        # Final Convolution
        # Genera tantos planos como clases tenemos que identificar
        self.last_conv = nn.Conv2d(64, out_classes, kernel_size=1)

        # La funcion forward describe el flujo de los datos a traves de la red
        # Capas de downsampling —> Cuello de botella —> Capas de upsampling —> Salida final
        def forward(self, x):
            x, skip1_out = self.down_1(x)
            x, skip2_out = self.down_2(x)
            x, skip3_out = self.down_3(x)
            x, skip4_out = self.down_4(x)

            x = self.neck_conv(x)

            x = self.up_4(x, skip4_out)
            x = self.up_3(x, skip3_out)
            x = self.up_2(x, skip2_out)
            x = self.up_1(x, skip1_out)

            x = self.dropout(x)

            x = self.last_conv(x)

            return x
```

Código 5.4: Clase UNet

5.2. Entrenamiento

En esta sección, se detallan los aspectos cruciales relacionados con la configuración, parámetros y el proceso de entrenamiento del modelo propuesto. Constituye una fase importante, ya que el éxito del proyecto depende en gran medida de cómo se configure y entrene el modelo final.

5.2.1. `torch.nn.init.kaiming_uniform_`

En el contexto del entrenamiento de la Red Neuronal, la inicialización de los pesos desempeña un papel fundamental. Antes de abordar aspectos directamente relacionados con el proceso de entrenamiento, cabe destacar la importancia de establecer adecuadamente los pesos iniciales, dado que este paso inicial puede afectar de manera significativa en el rendimiento global del modelo.

En nuestra implementación, hemos optado por emplear la clase `torch.nn.init.kaiming_uniform_` para llevar a cabo la inicialización Kaiming o He, cuya explicación se encuentra en la **Sección 5**. Esta elección se fundamenta en la eficacia demostrada de esta inicialización en modelos que hacen uso de funciones de activación tipo ReLU o sus variantes, como en nuestro caso particular con la utilización de LeakyReLU. Cabe destacar que la inicialización Kaiming contribuye a obtener resultados más efectivos al prevenir fenómenos de desvanecimiento o explosión de gradientes durante el proceso de entrenamiento.

Esta inicialización se aplica de manera consistente en todas las capas donde se llevan a cabo operaciones de convolución o convolución transpuesta, capas `Conv2d` y `ConvTranspose2d`, respectivamente.

La función `init_weights`, ha sido diseñada con el propósito de implementar esta inicialización en todas las capas pertinentes de la red.

```
def init_weights(m):
    if type(m) == nn.Conv2d or type(m) == nn.ConvTranspose2d:
        torch.nn.init.kaiming_uniform_(m.weight, nonlinearity='leaky_relu')

    if m.bias is not None:
        torch.nn.init.zeros_(m.bias)
```

Código 5.5: Función para la inicialización de pesos

5.2.2. `torch.nn.CrossEntropyLoss`

Respecto a la función de pérdida, hemos optado por la `CrossEntropyLoss`, pérdida de la entropía cruzada, ya que es realmente útil en problemas de clasificación multiclase. Ésta emplea dos funciones: `nn.LogSoftmax` y `nn.NLLLoss`, con el fin de hallar la pérdida de manera más estable [72].

En problemas de segmentación multiclase, como es el caso de nuestro TFG, es común aplicar la función `Softmax` en la capa final del modelo. Esto nos permite obtener un plano por clase, donde el valor de cada píxel representa la probabilidad de pertenecer a esa clase específica. En nuestro modelo no aplicamos esta capa final para evaluar su precisión, sino que durante la validación comparamos píxel a píxel con la máscara Ground Truth y posteriormente, calculamos el porcentaje de píxeles correctamente clasificados.

En nuestro caso, aplicamos la función `Softmax` para llevar a cabo la binarización de los planos predichos por el modelo. Posteriormente, aplicamos un umbral óptimo dependiente de la clase a estos planos, lo que nos permite realizar la binarización deseada.

5.2.3. `torch.optim.Adam`

Adam es una técnica de optimización ampliamente utilizada y como se comentó en la sección de optimizadores **Sección 3.5.3**, Adam combina dos algoritmos de optimización efectivos y tiene la capacidad de ajustar la tasa de aprendizaje para cada parámetro individualmente. Estas características lo convierten en un optimizador eficiente y eficaz, mostrando mejoras significativas en la velocidad de convergencia y destacando por su eficiencia computacional y uso eficaz de la memoria.

5.2.4. `torch.optim.lr_scheduler.ReduceLROnPlateau`

Con el objetivo de refinar la tasa de aprendizaje, se ha incorporado este método de ajuste. Para establecer el valor del parámetro principal de esta clase, `patience`, se han llevado a cabo pruebas con distintos valores, generalmente en un rango cercano a 2-3. Estas pruebas se han orientado a determinar si la reducción de la tasa de aprendizaje produce mejoras sustanciales en el entrenamiento del modelo. En muchas de estas pruebas, el sistema

no aplicó la reducción por dos motivos. El primero es porque el entrenamiento concluyó tras alcanzar el número máximo de épocas. El segundo es por la activación del **Early Stopping**, que emplea la precisión de clasificación sobre el conjunto de pérdida, mientras que el *scheduler*, utiliza la pérdida media de la época, por ello se activaba antes que el planificador y detenía el entrenamiento.

Sin embargo, en el modelo definitivo, como veremos en el **Capítulo 7**, si que se ha aplicado el planificador refinando ligeramente el entrenamiento.

5.3. Bucle de entrenamiento

Antes de empezar con el bucle principal de entrenamiento, establecemos diversos valores y parámetros esenciales para la ejecución del mismo, que nos permiten configurar el comportamiento del mismo. Esto incluye la instanciación del modelo y otros elementos mencionados en la sección anterior, como son la función de pérdida, la selección del optimizador, la inicialización de pesos y la configuración de parámetros específicos para el entrenamiento, entre otras.

- **device:** representa el dispositivo disponible para la ejecución del modelo, pudiendo adquirir el valor 'cuda' en el caso de contar con una GPU, o 'CPU' en ausencia de la misma. Aunque pude solicitar el uso de la GPU durante todo el desarrollo del proyecto, se hizo uso de ella en ciertos momentos, que facilitaron la realización de pruebas y experimentos en el entrenamiento del modelo. Cuando ambas opciones están disponibles, se da preferencia a la GPU debido al rendimiento superior que ofrece en términos de velocidad de entrenamiento y realización de cálculos.
- **unet_model:** crea la instancia del modelo UNet, especificando el número de clases de salida a través del parámetro `out_classes`.

```
device = torch.device('cuda' if torch.cuda.is_available() else "cpu")

out_classes = 5

# Instancia de la U-Net
unet_model = UNet(out_classes)

# Inicialización de pesos al modelo
unet_model.apply(init_weights)
unet_model = unet_model.to(device) # Si disponemos de GPU pasamos a la GPU
```

Código 5.6: Instanciación del modelo y device

Otras destinadas al ajuste del entrenamiento específicamente:

- **tam_lote:** tamaño del lote.
- **epochs:** número de épocas de entrenamiento.
- **stop:** límite máximo para el número de épocas consecutivas sin observar mejoras en la precisión sobre el conjunto de prueba. Esta medida nos habilita para implementar la técnica de *Early Stopping*. En caso de que no se registren mejoras en la precisión sobre el conjunto de prueba durante un número específico de épocas consecutivas, definido como **stop**, se detiene el proceso de entrenamiento del modelo. Nos permite prevenir el sobreajuste y optimizar el rendimiento.
- **patience:** número de épocas consecutivas sin que se produzca mejora en la pérdida media de la época. Si se alcanza este número el learning rate se reducirá.
- **learning_rate:** tasa de aprendizaje.
- **loss_crit:** función de pérdida.
- **optm:** el optimizador.
- **scheduler:** el planificador que adapta la tasa de aprendizaje a medida que las épocas avanzan.

```
tam_lote = 16
epochs = 40
patience = 2
stop = 8
learning_rate = 0.001
loss_crit = nn.CrossEntropyLoss()
optm = optim.Adam(unet_model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler
                .ReduceLROnPlateau(optm, patience=patience, verbose=True)
```

Código 5.7: Configuración de parámetros del entrenamiento

Por último, variables que almacenan información sobre el entrenamiento del modelo:

- **train_acc** y **test_acc**: contienen las tasas de acierto de entrenamiento y prueba, respectivamente, para cada época.
- **learning_rates**: almacena los distintos valores que ha tomado la tasa de aprendizaje en cada uno de las épocas.
- **total_losses**: contiene las pérdidas obtenidas en cada una de las iteraciones del entrenamiento.
- **best_acc_val**: contiene la mayor precisión alcanzada hasta el momento en el entrenamiento.
- **epoch_worst**: número de épocas consecutivas sin que se produzca una mejora en la precisión.
- **cont_epochs**: número de épocas ejecutadas en el entrenamiento.

```
train_acc = []
test_acc = []
learning_rates = []
total_losses = []

best_acc_val = 0
epoch_worst = 0
cont_epochs = 0
```

Código 5.8: Configuración de parámetros del entrenamiento

Ya tenemos definida la configuración del entrenamiento del modelo, por lo que podemos hacer uso de la función **train_unet**. Esta función ejecuta el entrenamiento de una época y nos devuelve las pérdidas y la tasa de acierto en la clasificación de los píxeles para esa época. En el **Código 5.9** vemos la función **train_unet**, de la que podemos destacar:

- Inicialmente, se crea una lista para almacenar las pérdidas de la época y dos variables, una que contabiliza el número de píxeles clasificados correctamente y otra que contiene el número de píxeles totales. Asimismo, se establece el modelo en modo de entrenamiento, haciendo uso de la instrucción **model.train()**; habilitando el dropout y la normalización batch.
- Posteriormente, para cada lote del conjunto de entrenamiento, llevamos a cabo las siguientes acciones:
 1. Permutamos las dimensiones de las imágenes para que cumplan con el formato (N, C, H, W) .
 2. Las enviamos al dispositivo de cálculo **device**.
 3. El modelo realiza la predicción sobre las imágenes, y después se calcula la pérdida entre la predicción y la imagen Ground Truth. Esto se realiza con la instrucción **loss = loss_criterion(outputs, masks)**.
 4. Limpiamos los gradientes acumulados en el optimizador (**optm.zero_grad()**).
 5. Realizamos la retropropagación del error para recalcular los gradientes (**loss.backward()**) y actualizamos los pesos de la red aplicando el paso del optimizador (**optm.step()**).

6. Para calcular la tasa de acierto, generamos un tensor que contiene las clases que se predicen con mayor probabilidad para cada uno de los píxeles, de la siguiente manera: `predicted = torch.max(outputs.data, 1)`. Después hallamos el número total de píxeles y el número de los clasificados correctamente, comparando el tensor `predicted` con los valores de la imagen Ground Truth.
 7. Liberamos la memoria empleada en las imágenes y Ground Truth que ya han sido procesadas.
 8. Guardamos la pérdida del lote en la lista de pérdidas de la época y calculamos el porcentaje de los píxeles bien clasificados.
- Finalmente, la función devuelve, como se ha mencionado previamente, una lista que contiene las pérdidas y el porcentaje de píxeles bien clasificados para una época.

```
def train_unet(model, train_dataloader, optm, loss_criterion, device):  
  
    train_loss = []  
  
    correct = 0  
    total = 0  
  
    model.train()  
  
    for images, masks in train_dataloader:  
        images = images.permute(0, 3, 1, 2)  
  
        images, masks = images.to(device, dtype=torch.float),  
                           masks.to(device, dtype=torch.long)  
  
        outputs = model(images)  
  
        loss = loss_criterion(outputs, masks)  
  
        optm.zero_grad()  
  
        loss.backward()  
        optm.step()  
  
        _, predicted = torch.max(outputs.data, 1)  
        total += masks.nelement()  
        correct += predicted.eq(masks.data).sum().item()  
  
        del images, masks  
        gc.collect()  
  
        train_loss.append(loss.item())  
  
    accuracy = round((correct / total)*100,2)  
  
    return train_loss, accuracy
```

Código 5.9: Función para el entrenamiento de una época: `train_unet`

Se ha implementado una función adicional que se encarga de procesar el conjunto de datos de prueba. Esta función al igual que `train_unet` calcula la cantidad de píxeles que se clasifican correctamente en la predicción, comparándolo con su correspondiente imagen Ground Truth, que contiene la clasificación idónea de estos. De esta manera, se obtiene el porcentaje de píxeles correctamente clasificados en relación con el número total de píxeles en la imagen, proporcionando una evaluación cuantitativa del rendimiento del modelo en términos de precisión de la clasificación de píxeles sobre el conjunto de prueba. Dicha función podemos observarla en el **Código 5.10**.

Presenta una estructura muy similar al método de entrenamiento, por lo que voy a comentar la principal diferencia:

- **model.eval():** Permite poner el modelo en modo evaluación, ya que presenta comportamientos diferentes en ciertas capas como dropout y las de normalización batch.

- Con el propósito de simplificar la función, se han excluido las referencias a la pérdida obtenida, enfocándose exclusivamente en la devolución del porcentaje de acierto en la clasificación de los píxeles. De esta forma, damos a la función un enfoque directo en la métrica principal del mismo.

```
def validate_unet(model, test_dataloader, loss_criterion, device):
    model.eval()

    correct = 0
    total = 0

    with torch.no_grad():
        for images, masks in test_dataloader:

            images = images.permute(0, 3, 1, 2)
            images, masks = images.to(device, dtype=torch.float), masks.to(device, dtype=torch.long)

            outputs = model(images)

            _, predicted = torch.max(outputs.data, 1)
            total += masks.nelement()
            correct += predicted.eq(masks.data).sum().item()

    accuracy = round((correct / total)*100,2)
    return accuracy
```

Código 5.10: Función para la validación: validate_unet

Las dos funciones presentadas en los fragmentos de **Código 5.9** y **5.10**, se emplean de manera conjunta con el propósito de entrenar el modelo, mediante un bucle que abarca todas las épocas, tal como se evidencia en el fragmento de **Código 5.11**.

- Inicializamos un temporizador con la función `time.time()` que nos va a permitir conocer el tiempo total de entrenamiento del modelo.
- Definimos un número máximo de épocas, con esto conseguimos que el entrenamiento no se prolongue demasiado en el tiempo. De forma habitual, la técnica de Early Stopping implementada hace que este se detenga antes, pero en caso de que no ocurriera, disponemos de un límite de épocas de entrenamiento.
- Para cada una de las épocas realizamos lo siguiente:
 1. Empleando la función `train_unet` obtenemos una lista que contiene las pérdidas de cada época, así como los valores de la tasa de acierto de esa época.
 2. En caso de que se den las condiciones para ello, modificamos el valor del learning rate con el scheduler elegido, en función de la pérdida media.
 3. Evaluamos la capacidad de generalización del modelo utilizando el conjunto de prueba, haciendo uso de la función `validate_unet`.
 4. Comprobamos si la precisión de validación conseguida en la época actual mejora el anterior registro hasta ese momento. Si lo mejora, guardamos una copia del estado del modelo, con la instrucción `torch.save()` y reiniciamos la variable `epoch_worst`.
Si por el contrario, no mejora la precisión aumentamos en una unidad la variable `epoch_worst`, que almacena el conteo de épocas consecutivas sin que se produzca mejora en la precisión.
 5. El mecanismo de Early Stopping, comprueba si el número de épocas consecutivas sin que se produzca mejora alcanza el límite establecido, en caso afirmativo detenemos el bucle de entrenamiento.

A lo largo del proceso de entrenamiento, se registran y almacenan diversos valores que facilitarán la visualización y comprensión de la evolución de nuestro modelo. Entre estos parámetros se encuentran la precisión alcanzada en cada época durante el entrenamiento y la evaluación, los puntos en los cuales se produce una mejora significativa, produciendo un guardado de un nuevo estado del modelo, así como los momentos en los que se realiza una detención anticipada mediante la técnica de Early Stopping.

```
t_ini = time.time()

for epoch in range(epochs):

    # Entrenamiento
    train_loss, acc_train = train_unet(unet_model, train_loader, optm, loss_crit, device)

    lr_previo = optm.param_groups[0]['lr']

    perdida_media = sum(train_loss)/len(train_loss)
    print(f"La perdida media de la epoca {epoch+1} es: {perdida_media}")
    scheduler.step(perdida_media)

    # Imprime el learning rate actual
    lr = scheduler.get_last_lr()[0]
    learning_rates.append(lr)

    if lr_previo != lr :
        print(f" ~~ Adaptando Learning Rate ~~ Modificacion de: {lr_previo - lr}")

    print(f"Epoca {epoch + 1}, Learning Rate: {lr:.6f}")

    # Fusionamos en un unico array las perdidas obtenidas
    total_losses.extend(train_loss)
    train_acc.append(acc_train)

    print(f"Epoca {epoch + 1}, Precision entrenamiento: {acc:.2f}")

    # Validacion
    test_loss, acc_test = validate_unet(unet_model, test_loader, loss_crit, device)
    test_acc.append(acc_test)
    print(f"Epoca {epoch + 1}, Precision validacion: {acc_test:.2f}")

    # Si el modelo mejora lo almacenamos
    if acc_test > best_acc_val :
        best_acc_val = acc_test
        epoch_worst = 0
        torch.save(unet_model, "models/better_model_results.pth")
        print(" ~ GUARDANDO MODELO ~ Obtenemos una precision = ", best_acc_val, " %.")
    else:
        epoch_worst += 1

    cont_epochs += 1

    # Parada preventiva
    if epoch_worst >= stop:
        print(" ~~ DETENCION DEL ENTRENAMIENTO ~~ Van ", epoch_worst, " epocas sin
obtener mejora.")
        break

t_fin = time.time()
print("\nEl tiempo de entrenamiento del modelo son: ", ((t_fin-t_ini)/60)/60, " horas")
```

Código 5.11: Bucle para el entrenamiento y validación del modelo captionpos

Capítulo 6

Implementación y herramientas

6.1. Tecnologías utilizadas

En esta sección, voy a justificar de forma detallada las tecnologías seleccionadas para este TFG y cómo se han aplicado en sus diferentes fases.

La elección de estas herramientas se basa en su eficacia para abordar los objetivos y requerimientos específicos del proyecto planteado, conocimientos previos sobre ellas y preferencias personales.

A lo largo de esta sección, se explorará el razonamiento detrás de esta elección y una visión general de su aplicación en el proyecto.

6.1.1. Python

Python [73] es un lenguaje de programación interpretado, se caracteriza por su versatilidad y flexibilidad en comparación a otros lenguajes. Es el lenguaje más familiarizado dentro del Aprendizaje Automático, consecuencia de su uso en asignaturas como *Técnicas de Aprendizaje Automático* y *Minería de Datos*.

Entre las principales ventajas de Python frente a otros lenguajes de programación, se encuentran las siguientes:

- **Simplicidad y coherencia:** Python permite desarrollar código de manera sencilla y legible, gracias a su estructura que es relativamente cercana a los lenguajes naturales.
- **Amplias opciones de visualización:** Python ofrece múltiples posibilidades para visualizar los resultados obtenidos, lo que facilita la interpretación y presentación de los datos.
- **Bibliotecas:** Python cuenta con un amplio catálogo de bibliotecas, como NumPy, Pandas, PyTorch, OpenCV y Matplotlib, cuyas funciones son de gran utilidad para las tareas realizadas en este TFG.

A pesar de las numerosas ventajas de Python, existe una desventaja principal: la velocidad. Al ser un lenguaje interpretado, su ejecución es más lenta en comparación con los lenguajes compilados. Sin embargo, en el contexto del Aprendizaje Automático, si la velocidad no es un obstáculo insuperable, Python supera con creces a sus competidores.

En este sentido, es importante destacar que disponemos de un préstamo de una máquina virtual con cuatro cores de procesamiento y bajo petición, una GPU de alto rendimiento, por parte del Departamento de Informática (ATC, CCIA y LSI) de la Universidad de Valladolid, lo que nos permite acelerar la ejecución de los procesos. Esta infraestructura nos permite acelerar la ejecución de los procesos, lo que hace que las limitaciones de velocidad de Python no sean un impedimento insuperable.

Por lo tanto, a pesar de la desventaja de la velocidad, las ventajas y comodidades que ofrece Python en este contexto hacen que sea la elección preferida para este TFG.

6.1.2. Anaconda

Anaconda [74] es una plataforma de código abierto que facilita la gestión y distribución de paquetes para Python y otros lenguajes de programación orientados a la ciencia, como R.

Algunas de las ventajas que ofrece esta plataforma son:

- **Administración y distribución de paquetes:** facilita la gestión, manejo de paquetes y bibliotecas de Python, lo que facilita tareas como instalar, actualizar o desinstalar.
- **Entornos virtuales:** permite la creación de entornos virtuales, ofreciendo una mayor encapsulación, separando paquetes y versiones entre los distintos proyectos, evitando así posibles conflictos. Un ejemplo destacado de este proyecto es la utilización de Jupyter Notebook.
- **Conjunto herramientas:** adicionalmente incluye herramientas y paquetes preinstalados. Los más utilizados en la Ciencia de Datos, entre ellos: Numpy, Pandas o SciPy.

6.1.3. Jupyter Notebook

Jupyter Notebook [75] es un entorno de trabajo de código abierto desarrollado por Proyecto Jupyter. Se basa en un sistema de notebooks o cuadernos, que se ejecutan a través de un navegador web. Cada cuaderno está dividido en celdas que se pueden ejecutar de forma independiente. Este formato permite combinar código Python con otros lenguajes de programación. Asimismo, entre código y código es posible incluir cajas con explicaciones de texto formateado bajo formato Markdown, $\text{\LaTeX}$ o HTML. Así como, permitir modificar el tipo de celda a otros formatos o agregar distintos elementos, como títulos o encabezados entre otros.

6.1.4. PyTorch

PyTorch [76] es una biblioteca de código abierto, cuyo objetivo principal es la implementación, desarrollo y entrenamientos de modelo de Aprendizaje Profundo, de una forma eficiente y sencilla. En la **Sección 6.1.5**, se analizará el por qué se optó por PyTorch entre las diversas opciones disponibles en el ecosistema de Python.

6.1.5. PyTorch vs Keras vs TensorFlow

Esta sección proporciona una justificación de la elección de PyTorch en lugar de otras bibliotecas similares, como puede ser Keras o TensorFlow.

A continuación, llevaré a cabo una comparación en diversos ámbitos entre estas tres [77]:

- **Facilidad de uso:** Keras y PyTorch proporcionan una experiencia inicial de uso más sencilla y amigable que TensorFlow, que presenta una curva de aprendizaje más empinada. Keras destaca por la rapidez en la construcción y entrenamiento de modelos. PyTorch, por su parte, por su simpleza, flexibilidad y control sobre el modelo.
- **Nivel de abstracción:** Keras suministra un marco de trabajo de alto nivel, con múltiples capas y modelos ya contruidos. A partir de aquí, se crean otros más complejos conectando bloques de construcción configurables elaborando encapsulación de capas y operaciones, para producir un código más legible. Su inconveniente principal es la escasa flexibilidad, ya que presenta más limitaciones a la hora de modificar los hiper-parámetros de la red y otros más internos.

Por contra, PyTorch y Tensorflow presentan muchos niveles de abstracción, lo que les hace menos legibles y más difíciles de comprender. A cambio, son tremendamente flexibles, propiedad muy apreciada para un uso profesional y personalizado del Aprendizaje Profundo.

- **Depuración:** Keras, como hemos visto, destaca por su sencillez, por lo que a menudo no necesita depuración. Sin embargo, PyTorch y Tensorflow, sí. Comparativamente, PyTorch ofrece mejores capacidades de depurabilidad que TensorFlow, aunque nada más sea, porque su sintaxis es mucho más cercana a Python.
- **Datasets:** Keras es la mejor opción cuando se trabajan con conjuntos de datos pequeños, mientras que TensorFlow y PyTorch obtienen mejores rendimientos con grandes conjuntos de datos.

- **Rendimiento y velocidad:** Este aspecto guarda una estrecha relación con el punto anterior. PyTorch y TensorFlow destacan por su eficiencia en el uso y gestión de la memoria, lo que se traduce en rapidez y alto rendimiento. Por otro lado, Keras, en comparación, es más lento y menos eficiente en términos de rendimiento.

En la siguiente tabla se resume los pros y contras de lo que acaba de exponer.

Aspecto	Keras	TensorFlow	PyTorch
Facilidad uso	Simple, conciso y legible	Complejo	Sencillo pero menos legible
Niveles de Abstracción	Alto	Bajo	Bajo
Datasets	Pequeños conjuntos de datos	Grandes conjuntos de datos	Grandes conjuntos de datos
Depuración	Simple, a veces no necesita	Compleja	Buenas capacidades de depuración
Rendimiento y velocidad	Lento, bajo rendimiento	Rápido, alto rendimiento	Rápido, alto rendimiento

Tabla 6.1: Tabla comparativa.

Así pues, Keras parece una opción más simple y cómoda frente a PyTorch y TensorFlow, pero menos flexible. Por el contrario, PyTorch y TensorFlow nos permiten construir Redes Convolucionales personalizadas, en particular, la arquitectura elegida UNET, lo que constituye uno de los objetivos de este trabajo.

A pesar de que TensorFlow ofrece un alto rendimiento en el manejo de conjuntos de datos grandes, tiene como contrapunto que no proporciona la misma flexibilidad a la hora de modificar la arquitectura de una Red Neuronal UNET original que PyTorch, que es más sencillo de aprender. Esto es algo que ha de valorarse para alguien con poca experiencia en el campo, ya que en asignaturas como *Minería de Datos y Técnicas de Aprendizaje Automático* se ha visto de forma superficial.

6.1.6. Overleaf

Overleaf [78] es un editor $\text{\LaTeX}$ que, a través de su aplicación web, permite la redacción colaborativa online y agiliza significativamente el proceso de creación y edición de artículos científicos, técnicos, informes y tesis, entre muchos otros documentos.

He seleccionado esta aplicación de edición de texto, en lugar de otras, por las siguientes razones:

- **Experiencia previa con la herramienta:** ya había utilizado Overleaf con anterioridad en otras asignaturas del Grado, por lo que conocía su funcionamiento y estaba familiarizado con su interfaz.
- **Plataforma web:** no requiere instalación de ningún elemento, ni actualizaciones, esto me permite acceder desde distintos sistemas operativos y/o máquinas accediendo a la última versión del documento.
- **Restaura versiones anteriores:** recupera antiguas versiones del documento de forma rápida, evitando así posibles cambios o pérdidas de información no deseadas.

Las ventajas mencionadas ofrecen un flujo de trabajo más cómodo y mayor seguridad en la gestión del documento. Por lo tanto, la redacción de esta memoria en Overleaf ha contribuido significativamente a mejorar la eficiencia y la calidad de este trabajo.

6.1.7. Flask

Flask [79] es un framework escrito en Python, diseñado para simplificar y facilitar la creaciones de aplicaciones web.

Optar por él se basó en las siguientes ventajas:

- **Integración:** está escrito en Python y diseñado específicamente para ser utilizado con este lenguaje. De esta forma, aprovechamos la flexibilidad y potencia de Python para desarrollar la aplicación web de forma efectiva y sencilla.

- **‘Micro’ Framework:** es considerado un "micro"framework, lo cual lo convierte la opción ideal para desarrollar una aplicación básica. Esto se alinea perfectamente con nuestro objetivo de proporcionar una interfaz simple y sencilla, ya que el objetivo principal de nuestro proyecto no es este.
- **Servidor web:** para realizar el despliegue de la aplicación, cuenta con un servidor web sencillo.
- **Depurador:** identifica si tenemos algún error en el código, pudiendo arreglarlo, así como ver el valor de las variables en todo momento.

6.1.8. GitHub

GitHub [80] es repositorio online gratuito para mantener un sistema de control de versiones del código y administrar de forma eficiente proyectos. Es uno de los repositorios más utilizados a nivel mundial, ampliamente utilizado en múltiples asignaturas durante la realización del grado.

6.1.9. GanttProject

GanttProject [81] es una herramienta de escritorio gratuita y compatible con múltiples plataformas que facilita la creación de diagramas de Gantt para la planificación de proyectos y la gestión de recursos.

Esta aplicación ha sido fundamental en la etapa inicial del proyecto, en particular, en el proceso de planificación, permitiéndonos crear de manera intuitiva, eficiente y rápida los diagramas de Gantt necesarios.

6.1.10. Astah Professional

Astah Professional es una herramienta que se emplea en la Ingeniería de Software, usada aquí para el modelado visual y diseño de software.

Se ha optado por esta herramienta para desarrollar la fase de Ingeniería de Software de la aplicación, porque ya había sido utilizada previamente en otras asignaturas como *Planificación y Diseño de Sistemas Computacionales*. Asimismo, Astah ofrece una interfaz más sencilla e intuitiva, con la que crear diagramas de forma más dinámica, comparado con su principal oponente: *Visual Paradigm*.

Capítulo 7

Resultados

En este capítulo, se lleva a cabo la evaluación del rendimiento del modelo implementado. Para alcanzar su versión final, se han ejecutado múltiples entrenamientos, variando sus hiper-parámetros y empleando diversas técnicas de inicialización de pesos y probando diferentes planificadores, con el objetivo de determinar cuáles de estas opciones maximizan su rendimiento. Tras un período de pruebas y ajustes, se seleccionó aquel que logra la mayor tasa de acierto en la clasificación de píxeles sobre el conjunto de prueba.

En las secciones subsiguientes, se presentará una concisa comparativa entre los resultados obtenidos durante la fase de entrenamiento y los generados por el modelo elegido.

7.1. Comparativa fase de entrenamiento

El proceso de entrenamiento de la red tiene una duración aproximada de 2-4h dependiendo del número de épocas, ajuste de los hiper-parámetros y el dropout realizado. El conjunto de datos final consta de 12800 imágenes, que ha sido dividido de la siguiente manera:

- **Entrenamiento:** 75 %, 9600 imágenes.
- **Validación:** 25 %, 3200 imágenes.

Ambos conjuntos de datos fueron generados al inicio del proyecto de manera única, asegurando así que la comparativa presentada en la **Tabla 7.1** utilizara el mismo conjunto de entrenamiento y prueba. Esta elección se realizó con el propósito de evitar posibles variaciones en el rendimiento del modelo originadas por diferencias en los datos, en lugar de ser atribuibles a las configuraciones de hiper-parámetros.

Nombre	Batch	Epochs	Optimizador	Scheduler	Pesos I	Dropout	T. Train	Acc Train (%)	Acc Test (%)
adagrad_01	16	40	Adagrad	RLROP	xavier	0.45	3 h	96.45	92.23
adadelta_01	16	50	Adadelta	RLROP	kaiming	0.50	3.48 h	97.02	93.57
adadelta_02	16	40	Adadelta	RLROP	xavier	0.20	3 h	97.83	93.71
adam_01	32	25	Adam	-	xavier	0.35	1.46 h	95.25	91.57
adam_02	16	45	Adam	RLROP	kaiming	0.2	3.60 h	97.03	94.02
adam_03	16	45	Adam	RLROP	kaiming	0.1	3.50 h	97.18	93.83
adam_04	64	40	Adam	RLROP	kaiming	0.5	2 h	96.18	93.43
adam_05	32	45	Adam	RLROP	xavier	0.1	2.75 h	96.21	93.3
adam_06	16	45	Adam	RLROP	kaiming	0.7	2.88 h	96.02	93.51
best_model.res	16	50	Adam	RLROP	kaiming	0.3	3.48 h	97.45	94.07

Tabla 7.1: Resultados conseguidos en los distintos entrenamientos.

Debido a las limitaciones temporales impuestas por la disponibilidad de la GPU, la cual ha estado a nuestra

disposición durante un período de tres días, hemos llevado a cabo un conjunto de 10 configuraciones de entrenamiento. Durante este proceso, hemos observado que un tamaño de lote de 16 produce resultados óptimos, frente a 32 ó 64. Asimismo, hemos identificado que el uso del optimizador Adam junto con el planificador ReduceOnPlateau, conduce a un rendimiento superior.

Es relevante destacar que la inicialización de pesos utilizando la técnica He, como se describe en la **Sección 5**, ha demostrado obtener mejores resultados en comparación con la inicialización de Xavier. Esta preferencia se fundamenta en la utilización de la función de activación ReLU, específicamente LeakyReLU, donde la inicialización obtiene mejores resultados de forma general. Aunque habitualmente los dropouts cercanos al 50 % suelen ofrecer mejor rendimiento, hemos constatado que, para nuestro escenario particular, el valor óptimo se sitúa en torno al 30 %.

Finalmente, el modelo seleccionado, denominado `best_model_results`, ha mostrado ser la elección más acertada al alcanzar la tasa de acierto más elevada en el conjunto de prueba.

7.2. Evolución del modelo

Como se detalló en el **Capítulo 5**, se ha implementado una estrategia donde, al observarse una mejora en la precisión sobre el conjunto de prueba durante una determinada época, se almacena dicho modelo. De este modo, se registra el modelo que consigue los resultados más sobresalientes. Esta dinámica descrita, combinada con la técnica de *Early Stopping*, han conseguido finalizar el entrenamiento del modelo antes de alcanzar el número máximo de épocas.

```
La pérdida media de la época 37 es: 0.0667619841111203
Época 37, Learning Rate: 0.000100
Época 37, Precisión entrenamiento: 97.45
Época 37, Precisión validación: 94.07
~ GUARDANDO MODELO ~ Obtenemos una precisión = 94.07 %.
La pérdida media de la época 38 es: 0.0655569733551182
Época 38, Learning Rate: 0.000100
Época 38, Precisión entrenamiento: 97.49
Época 38, Precisión validación: 94.06
La pérdida media de la época 39 es: 0.06471487421542406
Época 39, Learning Rate: 0.000100
Época 39, Precisión entrenamiento: 97.52
Época 39, Precisión validación: 94.06
La pérdida media de la época 40 es: 0.06383536961550514
Época 40, Learning Rate: 0.000100
Época 40, Precisión entrenamiento: 97.55
Época 40, Precisión validación: 94.04
La pérdida media de la época 41 es: 0.06331725212434927
Época 41, Learning Rate: 0.000100
Época 41, Precisión entrenamiento: 97.56
Época 41, Precisión validación: 94.04
La pérdida media de la época 42 es: 0.06257383175194263
Época 42, Learning Rate: 0.000100
Época 42, Precisión entrenamiento: 97.59
Época 42, Precisión validación: 94.06
La pérdida media de la época 43 es: 0.06211642915382981
Época 43, Learning Rate: 0.000100
Época 43, Precisión entrenamiento: 97.61
Época 43, Precisión validación: 94.05
La pérdida media de la época 44 es: 0.06160693881412347
Época 44, Learning Rate: 0.000100
Época 44, Precisión entrenamiento: 97.62
Época 44, Precisión validación: 94.04
La pérdida media de la época 45 es: 0.06102826389173666
Época 45, Learning Rate: 0.000100
Época 45, Precisión entrenamiento: 97.64
Época 45, Precisión validación: 94.04
~~ DETENCIÓN DEL ENTRENAMIENTO ~~ Van 8 épocas sin obtener mejora.
```

Figura 7.1: Aprendizaje del modelo y Early Stopping.

Con el propósito de obtener una comprensión detallada de la progresión del entrenamiento y validación, se han elaborado dos gráficos. En primer lugar **Figura 7.2**, se muestra la evolución de las pérdidas en cada iteración. Se observa una tendencia general de disminución constante, a medida que avanzan las iteraciones, aunque se identifican picos donde la pérdida aumenta en un conjunto de iteraciones.

Este suceso lo provoca el algoritmo, que ocasionalmente queda atrapado en mínimos locales, en lugar de continuar su búsqueda hacia el mínimo global. Durante ciertas iteraciones, el momento del algoritmo adquiere valores más altos intentando así escapar de este mínimo local. Este comportamiento se refleja en la **Figura 7.3** pero, aplicado en nuestro caso, a un mínimo local, lo que explica así la presencia de valores que no siguen la tendencia general de descenso constante en la pérdida, lo que se manifiesta como picos en ciertos conjuntos de iteraciones.

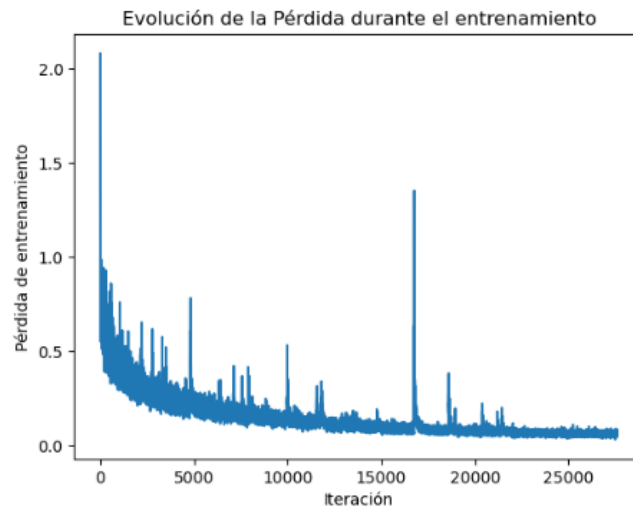


Figura 7.2: Evolución de las pérdidas por iteraciones.

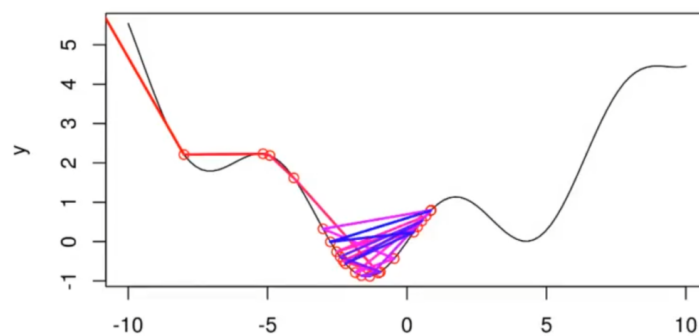


Figura 7.3: Momento atascado en un mínimo local. Fuente: [82]

En la imagen podemos observar que el suceso expuesto anteriormente ocurre de forma más habitual en las primeras iteraciones del aprendizaje y, a medida que las iteraciones aumentan, se vuelve más esporádico.

El segundo gráfico (**Figura 7.4**) presenta la evolución de la precisión en cada época, permitiendo la comparación entre la precisión lograda en el conjunto de entrenamiento y en el de validación. En términos generales, se constata que la tasa de aciertos en el de entrenamiento tiende a ser superior a la validación, dado que el modelo, en estas últimas, realiza predicciones sobre imágenes que no ha encontrado previamente.

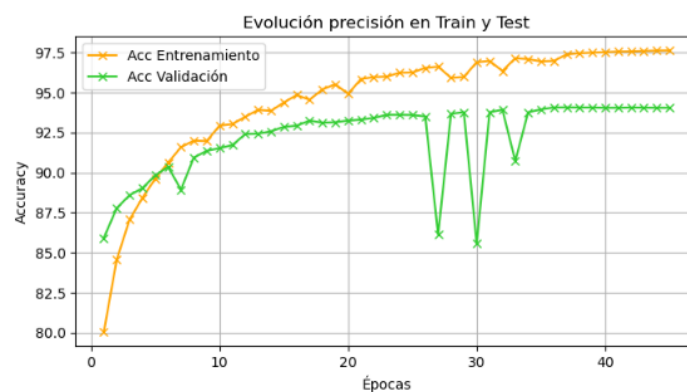


Figura 7.4: Evolución tasa de acierto.

Durante el entrenamiento del modelo seleccionado, en la época 35 se aplica el planificador implementado, el cuál reduce a 0.0009 la tasa de aprendizaje. Tras esta disminución, se observa en las épocas siguientes, una mejora de la precisión. Asimismo, se ve cómo se estabilizan los resultados de la precisión (ver Figura 7.4).

```
La pérdida media de la época 35 es: 0.08061284617210428
~ Adaptando Learning Rate ~ Modificación de: 0.0009
Época 35, Learning Rate: 0.000100
Época 35, Precisión entrenamiento: 96.97
Época 35, Precisión validación: 93.94
~ GUARDANDO MODELO ~ Obtenemos una precisión = 93.94 %.
La pérdida media de la época 36 es: 0.06887653121103843
Época 36, Learning Rate: 0.000100
Época 36, Precisión entrenamiento: 97.38
Época 36, Precisión validación: 94.06
~ GUARDANDO MODELO ~ Obtenemos una precisión = 94.06 %.
La pérdida media de la época 37 es: 0.0667619841111203
Época 37, Learning Rate: 0.000100
Época 37, Precisión entrenamiento: 97.45
Época 37, Precisión validación: 94.07
~ GUARDANDO MODELO ~ Obtenemos una precisión = 94.07 %.
```

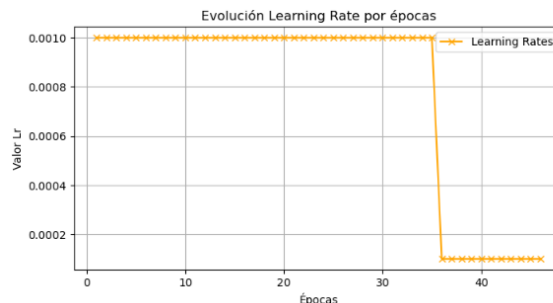


Figura 7.5: Precisión tras efecto del planificador.

Figura 7.6: Evolución de la Tasa de Aprendizaje.

7.3. Evaluación

Como se detalla en la **Sección 7.1**, la elección de dividir el conjunto de datos exclusivamente en dos partes, entrenamiento y prueba, responde a una cuidadosa consideración. La adopción de esta estrategia se fundamenta en la premisa de enfocar la evaluación del modelo en dos conjuntos representativos: el conjunto de entrenamiento, que facilita el ajuste de los parámetros del modelo mediante la retroalimentación directa de los datos utilizados durante el entrenamiento y, el conjunto de prueba, que se reserva para evaluar el rendimiento del modelo en datos no vistos previamente. Esta metodología de evaluación con dos conjuntos se ha preferido en lugar de una división en tres, (entrenamiento, validación y prueba), para optimizar la eficiencia computacional y garantizar que el modelo se evalúe en un conjunto de prueba independiente y relevante, dejando una mayor cantidad de datos para el entrenamiento. La decisión de no incorporar un conjunto de validación adicional se basa en la observación de que la monitorización del rendimiento durante el entrenamiento, a través de técnicas como el ajuste dinámico de la tasa de aprendizaje y el *Early Stopping*, proporciona una evaluación efectiva y suficientemente rigurosa del modelo, sin la necesidad de un conjunto de validación independiente. Igualmente, en la mayoría de casos, la precisión obtenida sobre el conjunto de validación es muy similar a la obtenida sobre el de prueba.

La evolución de las precisiones obtenidas se puede observar en la **Figura 7.4**, pero en esta tabla mostramos sólo las alcanzadas en el modelo seleccionado:

Conjunto	Precisión
Entrenamiento	97.45
Test	94.07

Tabla 7.2: Porcentaje de precisión sobre los conjuntos.

7.3.1. Evaluación del caso binario

La salida generada por el modelo puede interpretarse como imágenes, en las cuales el valor de cada píxel refleja la probabilidad de pertenencia a la clase correspondiente. En este contexto, si la probabilidad excede un umbral predefinido, se clasifica como una instancia positiva, indicando que pertenece a la clase en cuestión; de lo contrario, se clasifica como una instancia negativa. Con el propósito de determinar los umbrales óptimos para cada una de las clases, nos proponemos emplear métricas fundamentales para la evaluación de este tipo de modelos [83]:

- **True Positive (TP):** tasa de instancias positivas, aplicado a nuestro caso, píxeles clasificados correctamente.
- **False Positive (FP):** tasa de instancias clasificadas como positivas cuando realmente son negativas. Píxeles clasificados incorrectamente dentro una clase.
- **False Negative (FN):** tasa de instancias clasificadas incorrectamente como negativas. En nuestro trabajo son los píxeles que pertenecen a una categoría pero que no se cuentan como tal.

- **True Negative (TN):** tasa de instancias negativas. Píxeles que no pertenecen a una clase y que son clasificados correctamente, como no pertenecientes a dicha clase.

Estas cuatro métricas conforman la matriz de confusión. La diagonal formada por TP y TN representa las clasificaciones correctas y, la otra diagonal, las clasificaciones incorrectas, conformando una matriz similar a la de la **Figura 7.7**.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Figura 7.7: Matriz de confusión. Fuente: [84].

Derivadas de las cuatro métricas previamente mencionadas, se derivan otras tres de naturaleza más avanzada, las cuales posibilitan una evaluación más detallada y exhaustiva del rendimiento del modelo.

- **Sensibilidad:** Representa la capacidad del modelo para identificar correctamente las instancias positivas. Un valor alto indica que el modelo es efectivo en la detección de la clase.

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \quad (7.1)$$

- **Especificidad:** Describe la habilidad del modelo para identificar correctamente las instancias negativas. Una cantidad grande refleja la capacidad del modelo para evitar falsos positivos.

$$\text{Especificidad} = \frac{TN}{TN + FP} \quad (7.2)$$

- **Precisión:** Evalúa la exactitud de las instancias clasificadas como positivas. Una alta precisión indica la fiabilidad de las predicciones positivas realizadas por el modelo.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (7.3)$$

Hemos determinado los valores de las métricas previas para diversos umbrales en el rango de 0 a 1, con variaciones de 0.1 en 0.1. Estas métricas no sólo nos brindan una evaluación detallada del rendimiento del modelo, sino que también posibilitan la construcción de las *Curvas ROC*. Estas representaciones gráficas son fundamentales para la evaluación del clasificador y nos asisten en la elección del umbral óptimo. Para la elaboración de estas curvas, se exhiben las tasas de Verdaderos Positivos (*True Positive Rate*) frente a las tasas de Falsos Positivos (*False Positive Rate*).

Se han generado, para cada una de las clases, los resultados correspondientes a las métricas mencionadas, así como las respectivas Curvas ROC.

Clase Obstáculos

En **Tabla 7.3** podemos observar, cómo a partir del umbral 0.1, se produce un incremento muy elevado en la especificidad y precisión. No podemos observar tan buenos resultados en la métrica sensibilidad a medida que se aumenta el umbral. Esto implica que el modelo presenta dificultades para identificar los píxeles que realmente sí pertenecen a esta clase. Es en el umbral 0.5, donde podemos observar un mayor equilibrio entre especificidad y sensibilidad, así como un valor de precisión bastante adecuado también.

Treshold	Sensibilidad	Especificidad	Precisión
0.0	1.00000	0.00000	0.09047
0.1	0.82293	0.95740	0.65633
0.2	0.78199	0.96938	0.71605
0.3	0.74954	0.97598	0.75478
0.4	0.71972	0.98054	0.78467
0.5	0.69040	0.98400	0.80938
0.6	0.65996	0.98681	0.83105
0.7	0.62608	0.98927	0.85135
0.8	0.58452	0.99157	0.87172
0.9	0.52309	0.99399	0.89483
1.0	0.00000	1.00000	0.00000

Tabla 7.3: Métricas de evaluacion. Clase: Obstáculos.

El umbral óptimo seleccionado para esta clase es 0.5. Observamos que la precisión no alcanza valores muy elevados para niveles altos de sensibilidad. Esta circunstancia se debe a que estas zonas no suelen estar representadas por áreas consistentes de píxeles. En muchos casos, se trata de regiones que presentan otras clases en su entorno, caracterizadas por líneas finas de píxeles y patrones diversos, lo que puede propiciar confusiones por parte del modelo con relativa facilidad. Esto se puede observar también en su curva ROC:

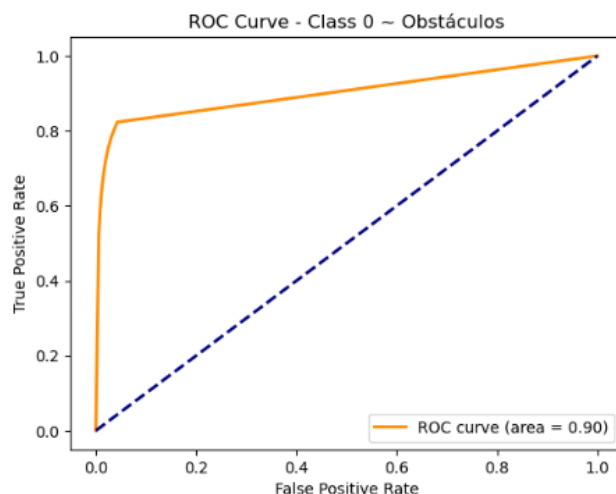


Figura 7.8: Curva ROC. Clase: Obstáculos.

Clase Agua

Para esta clase, vemos que a partir del umbral 0.1, se obtienen valores para las tres métricas muy altos casi siempre superiores a 0.95. Nuevamente es en el umbral 0.5, donde podemos observar un mayor equilibrio entre especificidad, sensibilidad y precisión, por lo que éste será el umbral elegido.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	0.99000	0.00000	0.03021
0.1	0.97373	0.99760	0.91152
0.2	0.96875	0.99812	0.92688
0.3	0.96403	0.99844	0.93620
0.4	0.95905	0.99868	0.94338
0.5	0.95347	0.99888	0.94950
0.6	0.94680	0.99906	0.95502
0.7	0.93843	0.99922	0.96011
0.8	0.92697	0.99938	0.96503
0.9	0.90760	0.99956	0.97058
1.0	0.00000	1.00000	0.00000

Tabla 7.4: Métricas de evaluacion. Clase: Agua.

Vemos que los valores para las métricas son bastante altos, algo que puede resaltar, puesto que de esta clase existen un menor número de imágenes, como vimos en la **Sección 4.1.3**. Esto se debe a que las zonas de agua, en la mayoría de imágenes, presentan una forma y colores similares. Suelen estar aisladas de otros objetos o zonas, lo que permite al modelo identificarlas con sencillez.

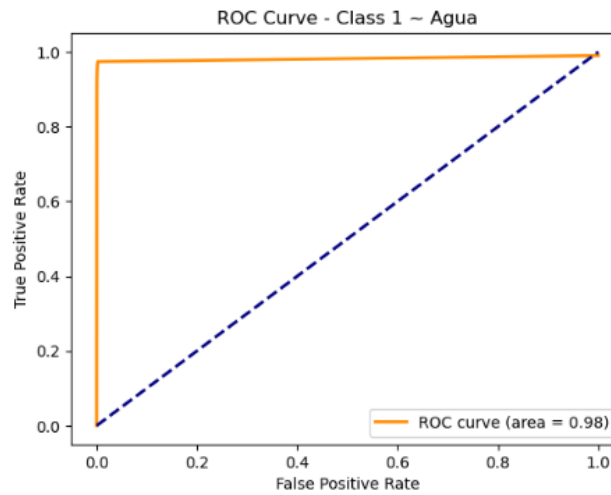


Figura 7.9: Curva ROC. Clase: Agua.

Clase Vegetación

En este caso observamos unos valores muy similares a los obtenidos para la clase Agua. A partir del umbral 0.1 valores superiores a 0.9 en las tres métricas. Se trata, junto con Agua, de la clase mejor clasificada por la Red Neuronal.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	1.00000	0.00000	0.35783
0.1	0.98800	0.95128	0.91744
0.2	0.98295	0.96240	0.93475
0.3	0.97816	0.96912	0.94553
0.4	0.97311	0.97404	0.95360
0.5	0.96742	0.97802	0.96025
0.6	0.96070	0.98149	0.96609
0.7	0.95226	0.98465	0.97150
0.8	0.94081	0.98773	0.97681
0.9	0.92191	0.99108	0.98269
1.0	0.00000	1.00000	0.00000

Tabla 7.5: Métricas de evaluación. Clase: Vegetación.

En el caso, tanto de la clase 'Vegetación', como de la de 'Agua', el modelo exhibe un desempeño excepcional. Este fenómeno se atribuye nuevamente a las características distintivas de los patrones y colores asociados con estas clases específicas. De manera general, las zonas que presentan tonalidades azules y verdes, suelen corresponder a cuerpos de agua y vegetación respectivamente. Este reconocimiento facilita al modelo la identificación y diferenciación más efectiva de estas clases en comparación con otras.

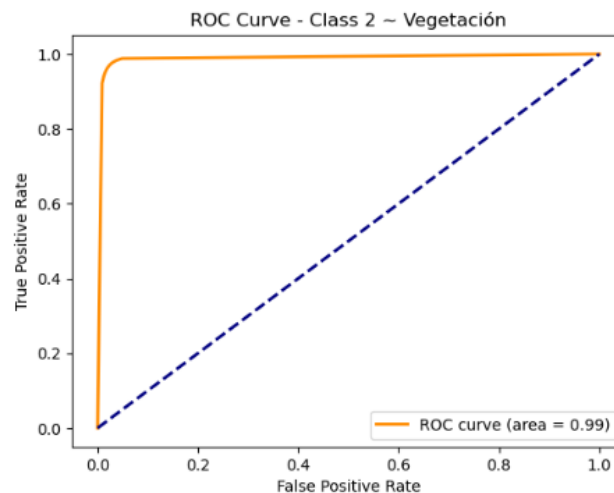


Figura 7.10: Curva ROC. Clase: Vegetación.

Movimiento

Los resultados obtenidos para esta clase, son similares a los mostrados en la clase Obstáculos. Se consigue, para todos los umbrales superiores a 0.1, valores de especificidad muy altos, mientras que no tanto para la sensibilidad. Esto implica que el modelo es muy eficaz identificando los píxeles que no pertenecen a esta clase, pero no tanto para identificar los que sí pertenecen a ella.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	1.00000	0.00000	0.01921
0.1	0.85450	0.99398	0.72260
0.2	0.83051	0.99555	0.77317
0.3	0.80909	0.99649	0.80707
0.4	0.78752	0.99716	0.83387
0.5	0.76410	0.99770	0.85698
0.6	0.73700	0.99816	0.87791
0.7	0.70409	0.99858	0.89797
0.8	0.66142	0.99896	0.91820
0.9	0.59596	0.99936	0.94123
1.0	0.00000	1.00000	0.00000

Tabla 7.6: Métricas de evaluación. Clase: Movimiento.

La Curva ROC correspondiente a esta clase exhibe un área bajo la curva menor, en comparación con las clases en las cuales el modelo demuestra un rendimiento superior. La forma de la curva es similar a la obtenida para la clase Obstáculos. Esta tendencia se atribuye a que, aunque las imágenes contienen muestras de esta clase, estas áreas suelen caracterizarse por un menor número de píxeles, que son más escasas en comparación con otras clases. En muchos casos, incluso al examinar visualmente la imagen proporcionada al modelo, se observa una similitud en colores y patrones con otras clases, como Zona Aterrizable u Obstáculos, lo que plantea dificultades para que el modelo diferencie efectivamente entre estas clases, de ahí los resultados obtenidos.

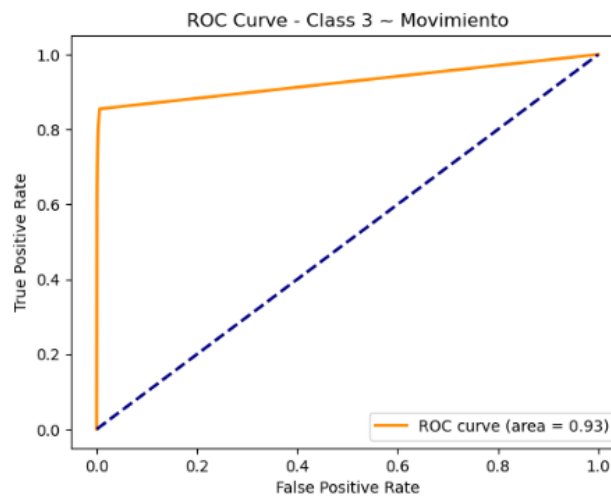


Figura 7.11: Curva ROC. Clase: Movimiento.

Zona Aterrizable

En esta clase, se observa un buen rendimiento, con resultados que guardan similitud con las clases Agua y Vegetación. A partir de un umbral de 0.1, se registran valores superiores a 0.9 en las tres métricas evaluadas. Sin embargo, al comparar con las otras dos clases, se constata que se obtienen valores de especificidad ligeramente inferiores. Esto indica que, aunque el modelo es capaz de identificar correctamente la mayoría de los píxeles que no pertenecen a la clase Zona Aterrizable, existe una proporción de píxeles que, a pesar de pertenecer a dicha clase, no son clasificados correctamente por el modelo. Esto puede deberse a la similitud visual con otras clases.

Threshold	Sensibilidad	Especificidad	Precisión
0.0	1.00000	0.00000	0.50228
0.1	0.98667	0.90052	0.91034
0.2	0.98180	0.92002	0.92636
0.3	0.97736	0.93202	0.93648
0.4	0.97274	0.94109	0.94427
0.5	0.96758	0.94861	0.95081
0.6	0.96156	0.95525	0.95666
0.7	0.95400	0.96143	0.96216
0.8	0.94357	0.96764	0.96774
0.9	0.92594	0.97481	0.97426
1.0	0.00000	1.00000	0.00000

Tabla 7.7: Métricas de evaluación. Clase: Zona Aterrizable.

Debido al alto rendimiento del modelo en la clasificación de esta clase, la curva ROC muestra un área bajo la curva (AUC) cercana a 1. Esto indica que el modelo tiene una alta tasa de verdaderos positivos y una baja tasa de falsos positivos, lo que se traduce en una buena capacidad clasificación de la clase.

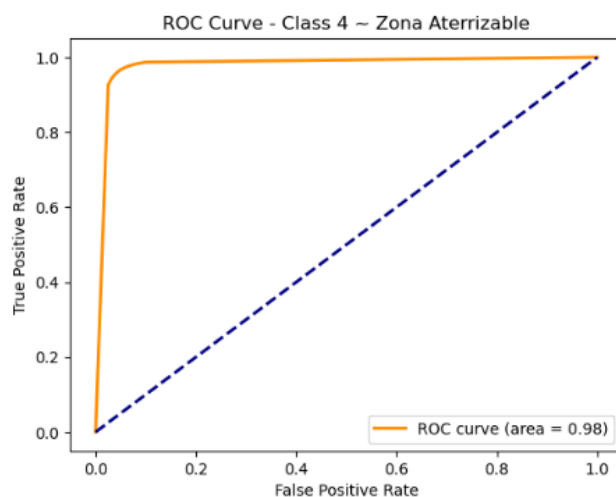


Figura 7.12: Curva ROC. Clase: Zona Aterrizable.

En términos generales, el modelo demuestra un rendimiento de clasificación sólido en todas las clases, con un rendimiento destacado en las clases Agua, Vegetación y Zona Aterrizable. Sin embargo, se observa un rendimiento ligeramente inferior en las clases Obstáculos y Movimiento. Esta discrepancia puede estar asociada a similitudes visuales entre estas clases, lo que podría dificultar la capacidad del modelo para diferenciarlas con precisión. A pesar de esto, el sistema es capaz de identificar todas las cinco clases posibles, sin mostrar una capacidad de reconocimiento nula para ninguna de ellas.

En lo que respecta a la selección de los umbrales óptimos para cada clase, se ha buscado un equilibrio entre maximizar el número de píxeles clasificados y asegurar un nivel mínimo de certeza en la clasificación correcta de cada píxel. Para lograr este objetivo, se ha optado por seleccionar el umbral que proporciona el mayor equilibrio entre las tres métricas evaluadas. Curiosamente, este umbral resultó ser el mismo para todas las clases, 0.5.

Clase	Threshold
Obstáculos	0.5
Agua	0.5
Vegetación	0.5
Movimiento	0.5
Zona Aterrizable	0.5

Tabla 7.8: Thresholds óptimos para cada clase.

7.4. Interpretación de los resultados

En esta sección, se abordarán las transformaciones llevadas a cabo en las salidas, partiendo de las predicciones generadas por el modelo, hasta alcanzar la representación visual final presentada al usuario a través de la aplicación.

7.4.1. Generada por el modelo

Inicialmente, los resultados generados por el modelo consisten en cinco planos: uno por cada clase, donde los valores de los píxeles representan las puntuaciones o valores 'logit' de pertenencia a la clase correspondiente. Posteriormente, se aplica una función softmax a estas salidas, transformándolas en probabilidades. De esta manera, obtenemos cinco planos, donde las cantidades del primer plano representan las probabilidades de que cada píxel pertenezca a esa clase y, así, sucesivamente. En la **Figura 7.14**, se muestran estas probabilidades utilizando la máscara 'hot'. En este esquema de color, los valores altos se representan con colores claros y los bajos con colores oscuros, proporcionando una representación visual intuitiva de los resultados obtenidos.

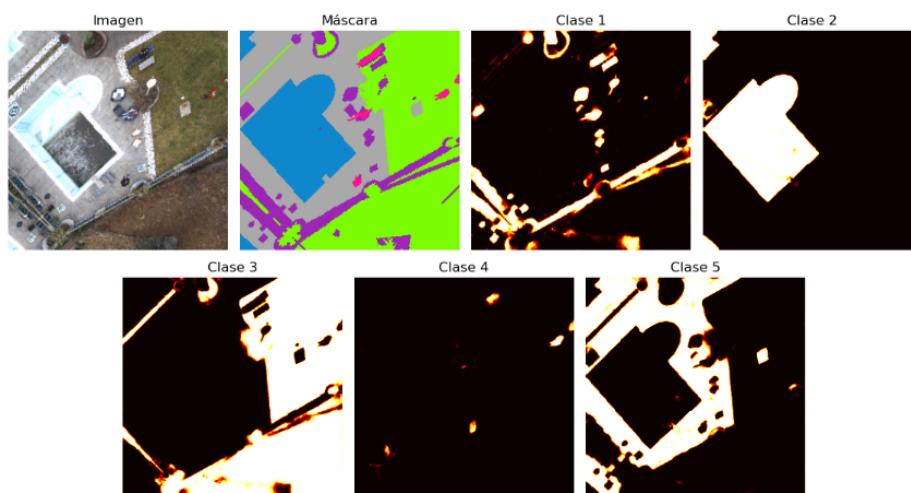


Figura 7.13: Probabilidades predichas sobre una imagen del conjunto de test.

7.4.2. Binarización

Posteriormente, se implementa esta operación a cada uno de los planos de probabilidades derivados del modelo de segmentación. Este proceso implica la aplicación de un umbral óptimo, determinado a partir de las curvas ROC, junto con las métricas asociadas a estas curvas.

En este contexto, cuando el valor de un píxel supera el umbral establecido, se asigna el valor 1; de lo contrario, se asigna el valor 0. Este umbral desempeña un papel crucial al resaltar la presencia de la clase específica en cada plano, definiendo de manera nítida los límites de las regiones de interés y eliminando las tonalidades rojizas presentes en la imagen de mapa de calor de la **Figura 7.14**. Dichas tonalidades representan píxeles en los cuales el modelo no logra clasificar con certeza: probabilidad inferior al 45 %-50 %. Estos píxeles, en su mayoría, se encuentran en los bordes, donde se produce un cambio de clase. Aquí, el modelo no es capaz de detectar a la perfección donde finaliza una clase y comienza la otra.

Tras la binarización de todos los planos, si no se han clasificado dentro de ninguna clase, serán representados en la imagen definitiva de color negro.

Con el propósito de mejorar la precisión del modelo, es posible que algunos píxeles presenten una asignación multiclase, es decir, superen el umbral establecido para más de una clase. Para abordar esta situación, se adopta la estrategia de asignar al píxel el valor correspondiente a la clase con la probabilidad más alta. Este enfoque asegura que la clasificación refleje la clase, para la cual el modelo muestra mayor confianza.

Tras la ejecución del siguiente fragmento de código, `binary_maps` contiene los planos binarizados:

```

# Umbrales ptimos
thresholds = [0.5, 0.5, 0.5, 0.5, 0.5]

# Tensor de las mismas dimensiones que la salida del modelo
binary_maps = torch.zeros_like(outputs)

# Tensor para almacenar las probabilidades máximas
max_probs, max_classes = torch.max(outputs, dim=1)

for i in range(outputs.size(1)):
    binary_map = (outputs[:, i, :, :] > thresholds[i]).float()

    # Nos permite aplicar la clase de mayor probabilidad
    binary_maps[:, i, :, :] = binary_map * (max_classes == i).float()

```

Código 7.1: Binarización aplicada a la salida del modelo

7.4.3. Resultado final

Finalmente, se fusionan todos los planos binarizados en uno único para dar la imagen resultante. En este contexto, no se produce superposición de píxeles, ya que se presentan dos casos distintos: el píxel adopta el valor correspondiente a la clase que lo clasificaba con la probabilidad más elevada, o asume el valor 0, al no superar el umbral óptimo para ninguna de las clases.

Con el propósito de facilitar una comparación visual intuitiva, se ha seguido la paleta de colores definida en las imágenes Ground Truth coloreadas y proporcionadas en el conjunto de datos. Los códigos RGB asignados a cada clase son los siguientes:

Clase	Color (RGB)
Obstáculos	(155, 38, 182)
Agua	(14, 135, 204)
Vegetación	(124, 252, 0)
Movimiento	(255, 20, 147)
Zona Aterrizable	(169, 169, 169)

Tabla 7.9: Gama de colores RGB para cada clase.

Se ha decidido seguir la misma coloración, porque nos permite ver una gran similitud de la imagen predicha comparada con su imagen Ground Truth a simple vista:

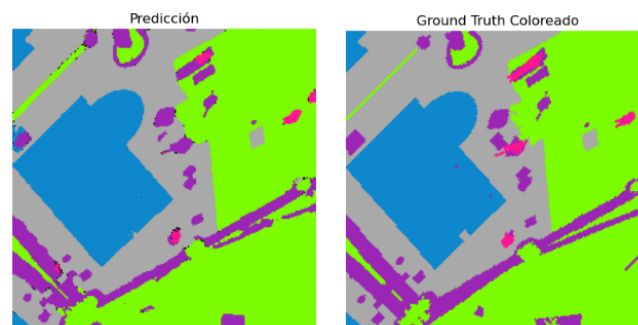


Figura 7.14: Predicción vs Ground Truth coloreada.

Dado que la imagen seleccionada cuenta con su correspondiente Ground Truth, se procedió al cálculo de la precisión, obteniendo un valor del 94.78 %, ligeramente superior a la precisión global alcanzada por el modelo. En este caso específico, se observa que un 0.27 % de los píxeles (176) no ha superado el umbral establecido para

ninguna de las clases, indicando que el modelo no ha logrado clasificarlos con un nivel mínimo de precisión. Éstos se representan en negro en la **Figura 7.15**.



Figura 7.15: Píxeles en negro en la predicción.

Como se ha mencionado anteriormente, estos píxeles suelen ubicarse comúnmente en los bordes y límites donde se produce una transición entre clases. No obstante, gracias a la elección de umbrales relativamente bajos, se logra mantener un equilibrio adecuado entre precisión y la presencia de píxeles sin clasificar. De esta manera, garantizamos la clasificación de la mayoría de los píxeles, manteniendo una alta precisión sin incrementar de manera significativa la cantidad de píxeles sin clasificar. Gracias a este enfoque equilibrado, conseguimos una representación visual de la predicción coherente y precisa.

Capítulo 8

Aplicación web

Este capítulo presenta un breve proyecto de Ingeniería de Software, que consiste en la creación de una aplicación web destinada a servir como herramienta que proporcione al usuario una interfaz simple e intuitiva. La finalidad de esta herramienta es posibilitar la interacción del usuario con el modelo elaborado, permitiéndole cargar imágenes y obteniendo como resultado las predicciones realizadas por el mismo. La aplicación proporciona información complementaria, como la cantidad de píxeles que no han sido clasificados con certeza y el porcentaje que representan con respecto al total.

Esta herramienta de software se presenta como un componente adicional para facilitar la utilización y obtención de resultados al usuario tras la implementación del modelo desarrollado. A pesar de que el núcleo central de este Trabajo de Fin de Grado se concentra en la investigación y aplicación de Redes Neuronales Convolucionales para la segmentación de imágenes urbanas, se ha estimado pertinente la incorporación de esta aplicación.

En este sentido, este capítulo aborda las fases principales de análisis y diseño propias de un proyecto de software. Cabe destacar que se adapta a la simplicidad de nuestra aplicación, sin entrar en detalles específicos, ya que éste no es el objetivo principal del proyecto. No obstante, se ha realizado para proporcionar un contexto general, que permita comprender cómo se ha desarrollado esta herramienta.

8.1. Introducción

La aplicación desarrollada en el marco de este proyecto recibe el nombre de LandDetectIA. Este nombre refleja de manera concisa el propósito central de la herramienta, que es la detección de zonas de aterrizaje seguras para drones mediante el uso de un modelo de Deep Learning. LandDetectIA, a través de operaciones como la carga de imágenes y la visualización de resultados, busca facilitar el proceso de obtención de predicciones, complementando así la investigación principal realizada en el ámbito de las Redes Neuronales Convolucionales.

8.2. Análisis

8.2.1. Análisis de requisitos

En esta sección, se aborda el análisis de requisitos de la aplicación, un paso fundamental para el desarrollo eficiente y el establecimiento de las bases necesarias para su implementación exitosa.

8.2.1.1. Requisitos funcionales

Establecemos las principales operaciones que nuestra aplicación debe permitir realizar al usuario.

ID	Nombre	Descripción
RF-01	Carga de imagen	El sistema deberá permitir al usuario cargar una imagen.
RF-02	Segmentación	El sistema deberá ser capaz de ofrecer una imagen segmentada, diferenciando las clases posibles: Obstáculos, Agua, Vegetación, Movimiento y Zona Aterrizable a partir de la imagen proporcionada.
RF-03	Resumen de segmentación	El sistema deberá proporcionar un resumen detallado de los resultados de la segmentación.
RF-04	Visualización de resultados	El sistema deberá permitir al usuario visualizar tanto la imagen segmentada como el resumen de la segmentación.
RF-05	Cancelación de carga de imagen	El sistema deberá permitir al usuario cancelar la subida de una imagen en cualquier momento.

Tabla 8.1: Requisitos funcionales de la aplicación

8.2.1.2. Requisitos no funcionales

Definimos las características de rendimiento, usabilidad y otros aspectos del sistema, que son esenciales para garantizar el buen funcionamiento y la eficacia global de la aplicación.

ID	Nombre	Descripción
RNF-01	Facilidad de uso	El sistema deberá permitir a un usuario con conocimientos medios o básicos en el uso de Internet obtener su primera predicción de manera autónoma, sin requerir asistencia adicional. El tiempo máximo, que un usuario promedio deberá invertir en el proceso, no deberá exceder un minuto.
RNF-02	Tiempo	El sistema deberá ofrecer el resultado de la predicción en un tiempo inferior a un minuto.
RNF-03	Accesibilidad	El sistema deberá permitir acceder a él a través de un navegador web.
RNF-04	Formatos de imagen	El sistema deberá permitir la carga de imágenes en formatos JPG, JPEG y PNG.
RNF-05	Portabilidad	El sistema deberá ser ejecutable en cualquier entorno que presente Docker instalado.
RNF-06	Compatibilidad	El sistema deberá ser desarrollado en un lenguaje de programación que ofrezca máxima compatibilidad con PyTorch.

Tabla 8.2: Requisitos no funcionales de la aplicación

8.2.1.3. Requisitos de información

ID	Nombre	Descripción
RDI-01	Modelo	El sistema deberá salvar el modelo que realiza la segmentación.
RDI-02	Imagen	El sistema deberá almacenar la imagen cargada.
RDI-03	Predicción	El sistema deberá almacenar la imagen segmentada.
RDI-04	Procesada	El sistema deberá almacenar la imagen procesada que se genere a partir de la original cargada.
RDI-05	Resumen	El sistema deberá almacenar la información sobre la predicción hasta que se visualice.

Tabla 8.3: Requisitos funcionales de información de la aplicación

8.2.2. Casos de uso

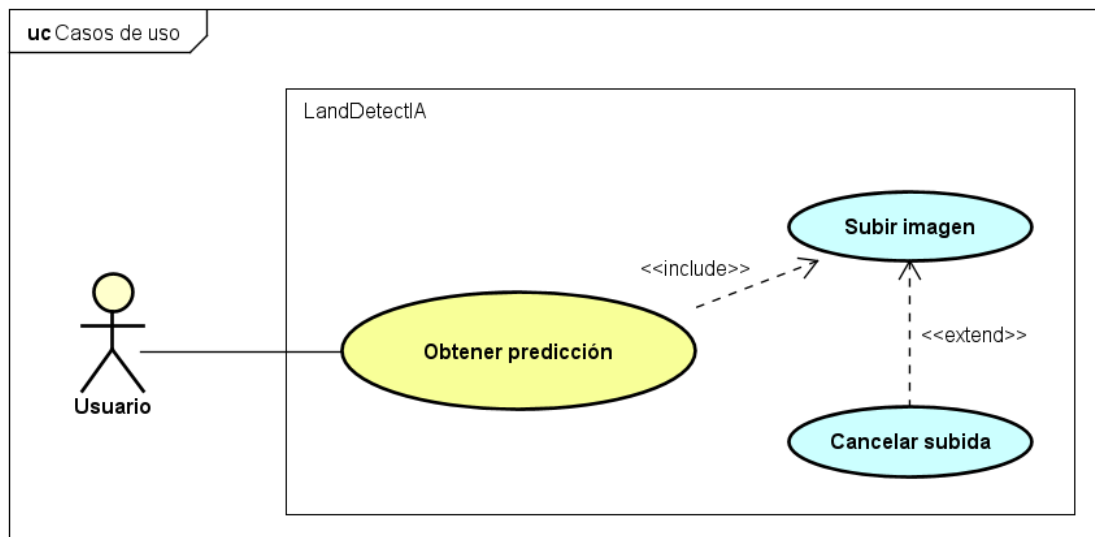


Figura 8.1: Diagrama de Casos de Uso

Descripciones de los Casos de Uso

CU-01	Subir imagen
Actor	Usuario
Descripción	El usuario sube una imagen para que sea procesada por el modelo.
Precondiciones	El usuario ha iniciado el sistema y se encuentra en la interfaz de carga de imágenes.
Postcondiciones	El sistema ha almacenado la imagen subida.
Secuencia normal	1. El usuario sube una imagen en el sistema. 2. El sistema comprueba el formato de la imagen. 3. El sistema ofrece una previsualización de la imagen subida.
Secuencia alternativa	1.a El usuario envía sin cargar una imagen previamente. 1.1.a El sistema detecta que no se ha cargado ninguna imagen, muestra un mensaje de error y vuelve al paso 1. 2.a El sistema detecta que el formato de la imagen no es compatible, muestra un mensaje de error y vuelve al paso 1. 3.a El usuario desea modificar la imagen subida, tiene efecto el caso de uso <Cancelar Subida>.

Tabla 8.4: Descripción del CU-01: Subir imagen

CU-02	Obtener predicción
Actor	Usuario
Descripción	El sistema ofrece la segmentación realizada sobre la imagen cargada.
Precondiciones	El usuario ha subido en el sistema la imagen que quiere segmentar.
Postcondiciones	El sistema ha almacenado la predicción realizada. El sistema ofrece una visualización del resultado obtenido.
Secuencia normal	1. El usuario solicita la predicción enviando la imagen cargada. 2. El sistema almacena la imagen cargada. 3. El sistema preprocesa la imagen 4. El sistema almacena la imagen procesada. 5. El sistema carga el modelo. 6. El sistema obtiene la predicción. 7. El sistema optimiza la salida obtenida. 8. El sistema ofrece una visualización de la imagen procesada y los obtenidos tras la segmentación.
Secuencia alternativa	2.a Se produce un error durante el almacenado de la imagen cargada. 3.a Se produce un error durante el preprocesado de la imagen. El sistema muestra un mensaje de error. 4.a Se produce un error durante el almacenado de la imagen procesada. El sistema muestra un mensaje de error. 5.a Se produce un error durante la carga del modelo. El sistema muestra un mensaje de error. 6.a Se produce un error durante la predicción del modelo. El sistema muestra un mensaje de error. 7.a Se produce un error durante la optimización de la salida. El sistema muestra un mensaje de error. 8.a El usuario desea realizar otra predicción. Se realiza el caso de uso <Subir imagen>

Tabla 8.5: Descripción del CU-02: Obtener predicción

CU-03	Cancelar subida
Actor	Usuario
Descripción	El sistema anula la imagen cargada y permite subir una imagen nueva.
Precondiciones	El usuario ha subido en el sistema la imagen que quiere segmentar.
Postcondiciones	El sistema elimina la imagen cargada por el usuario. El sistema elimina la previsualización de la imagen cargada. El sistema está listo para que se suba una nueva imagen.
Secuencia normal	1. El usuario hace click en cancelar. 2. El sistema elimina la previsualización de la imagen. 3. El sistema permite al usuario subir una nueva imagen.
Secuencia alternativa	2.a Se produce un error eliminando la previsualización de la imagen. El sistema muestra un mensaje de error.

Tabla 8.6: Descripción del CU-03: Cancelar subida

8.2.3. Diagrama de clases inicial

En este diagrama, se ha planteado la necesidad de la creación de dos clases. La clase UNet representa la arquitectura de la Red Neuronal utilizada, en este caso, modelo U-Net. En ella, se plasma, el método 'obtenerPredicción', que se encarga de encapsular el proceso de inferencia y predicción del modelo, este se toma como entrada una imagen y devuelve los planos segmentados.

Por otro lado, la clase Best_Model es una instancia específica de UNet, que ha sido entrenada y optimizada para alcanzar la máxima precisión posible. Esta clase no sólo hereda las propiedades y métodos de UNet, sino que también almacena la configuración específica que fue determinada durante el proceso de entrenamiento. Best_Model es un objeto concreto derivado de la clase UNet.

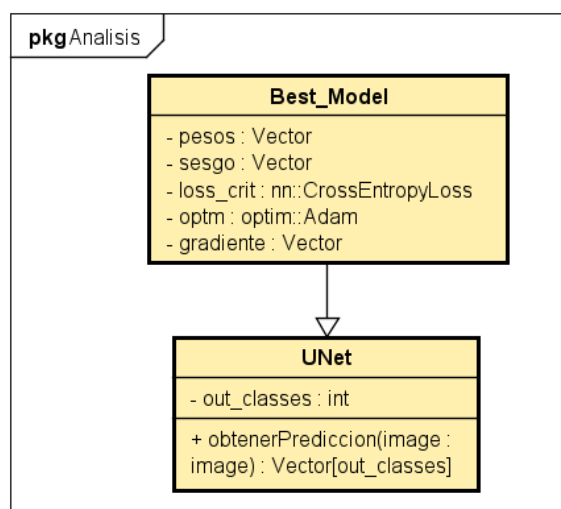


Figura 8.2: Diagrama de Clases inicial

8.3. Diseño

8.3.1. Decisiones sobre la implementación

Tras el análisis del sistema, en concordancia con lo establecido al inicio de este capítulo, la solución que satisface la mayoría de los requisitos de manera eficaz, es el desarrollo de una aplicación web.

Con el objetivo de cumplir con uno de los requisitos no funcionales de mayor prioridad, RNF-06, se ha optado por desarrollar la aplicación en Python, utilizando el microframework Flask [79]. Como se mencionó en la **Sección 6.1.7**, permite el desarrollo de aplicaciones web ofreciendo máxima compatibilidad con Python. Destaca por su sencillez y por presentar una curva de aprendizaje moderada, aspectos que se alinean con los plazos y restricciones de tiempo asociados al desarrollo de este TFG.

Optar por una aplicación web aborda otros requisitos no funcionales, como son los requisitos RNF-01, RNF-03 y RNF-04, relacionados con la facilidad de uso, accesibilidad y el cumplimiento de los formatos de imagen. Se ha decidido desarrollar una interfaz de usuario simple e intuitiva pero que, a su vez, implemente comprobaciones para garantizar la coherencia de los datos ingresados y así evitar la implementación código adicional dedicado a estas verificaciones. Para el desarrollo de esta interfaz, se hará uso de código HTML, CSS y JavaScript. Estas interfaces de usuario estarán basadas en las empleadas en un TFG previo [85].

Finalmente, para abordar el requisito de portabilidad, RNF-05, se ha elegido el uso de Docker para empaquetar la aplicación y sus dependencias en un contenedor. Esta elección proporciona un entorno consistente a la aplicación, permitiendo que el sistema sea portable y pueda ser desplegado en cualquier entorno que tenga Docker instalado.

Todas estas decisiones de diseño se han tomado con el objetivo de cumplir exitosamente con la mayoría de los requisitos previamente expuestos durante la fase de análisis.

8.3.2. Patrón de diseño

8.3.2.1. Patrón MVT

El patrón elegido para el desarrollo de la aplicación es el Modelo-Vista-Plantilla (MVT), conocido como Model-View-Templates en inglés. Este patrón se utiliza comúnmente en el desarrollo de aplicaciones web con herramientas como Flask o Django.

El objetivo principal de aplicar este patrón concreto es que nos permite estructurar la lógica de aplicación de manera efectiva. En particular, en tres componentes que están interconectados, asignando así responsabilidades concretas y facilitando la modularidad.

Este patrón MVT presenta semejanzas y diferencias respecto al conocido patrón MVC (Modelo-Vista-Controlador):

- **Modelo:** es el encargado del manejo, y gestión de los datos, en una aplicación convencional. Definiría la estructura de estos y manejaría las consultas a la base de datos, entre otras muchas tareas. En nuestro caso concreto, se corresponde con el modelo óptimo entrenado que se encuentra dentro del directorio *model/* y la clase sobre la que se instancia este modelo, contenida en el archivo *unet.py*. Este archivo tiene las cuatro clases que generan la arquitectura general de nuestro modelo. Está en el directorio *notebooks/Model/*, es decir, nuestro modelo está compuesto por dos paquetes *model* y *notebooks/Model*, que contienen el óptimo y la clase que permite la instanciación de éste respectivamente.
- **Vista:** es el componente que asume las responsabilidades de un controlador en un MVC convencional. A parte de gestionar la salida de la información, es decir, cómo se muestran y presentan los datos resultantes al usuario, también maneja la lógica y navegabilidad de la aplicación. Gestiona las interacciones de la interfaz, maneja las solicitudes HTTP y está conectado con el modelo, para enviar, recuperar o manipular los datos, para después presentarlos. En nuestra aplicación, el encargado de desempeñar estas funciones es el propio código Python de *app.py*.
- **Plantilla:** este último componente se encarga presentar los datos al usuario en el formato adecuado. Adquiere las competencias de la Vista del patrón MVC. En nuestro caso, este componente consiste en las interfaces HTML que se encuentran en el directorio *templates/*.

8.3.3. Arquitectura

En esta sección, se expone de manera detallada la arquitectura de la aplicación desarrollada. Para ello, se emplean los diagramas *Uses Style*, así como el diagrama específico *ModulesAndUsesStyle* de cada paquete, en el que se incluyen las clases y cómo se relacionan. La confección de estos diagramas de manera específica, para cada paquete, posibilitará obtener una visión más detallada acerca del diseño y la organización estructural de la aplicación.

La elaboración de estos diagramas, y la realización de la fase de Ingeniería de Software, no constituyen una tarea prioritaria en la *Mención en Computación*. En este contexto, me he inspirado para toda esta fase, incluido la realización de los diagramas, en otro TFG [86].

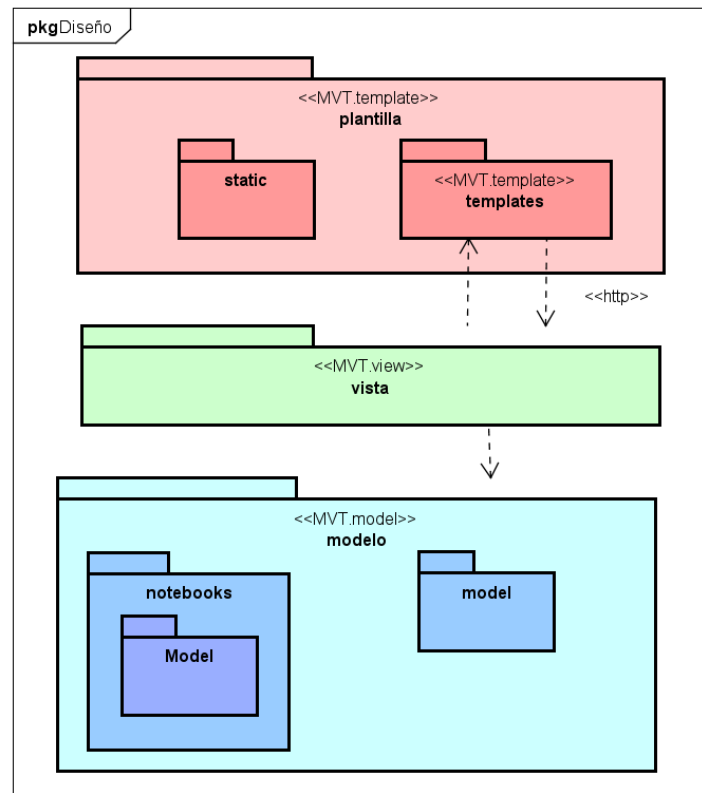


Figura 8.3: Diagrama Uses Style General.

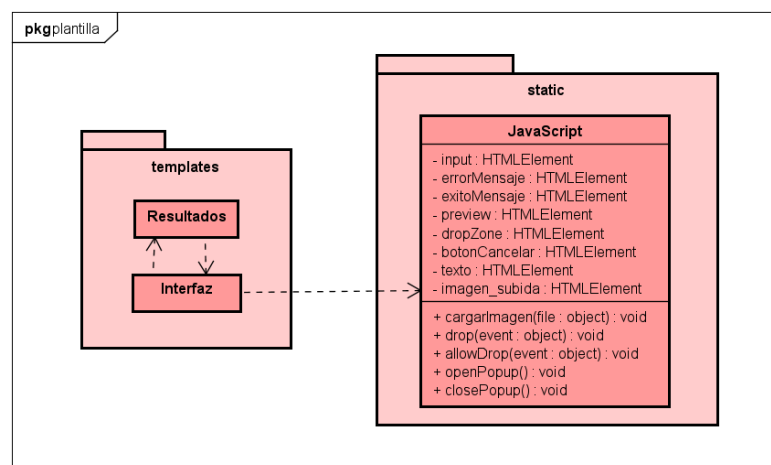


Figura 8.4: Diagrama Modules and Uses Style. Paquete: Plantilla.

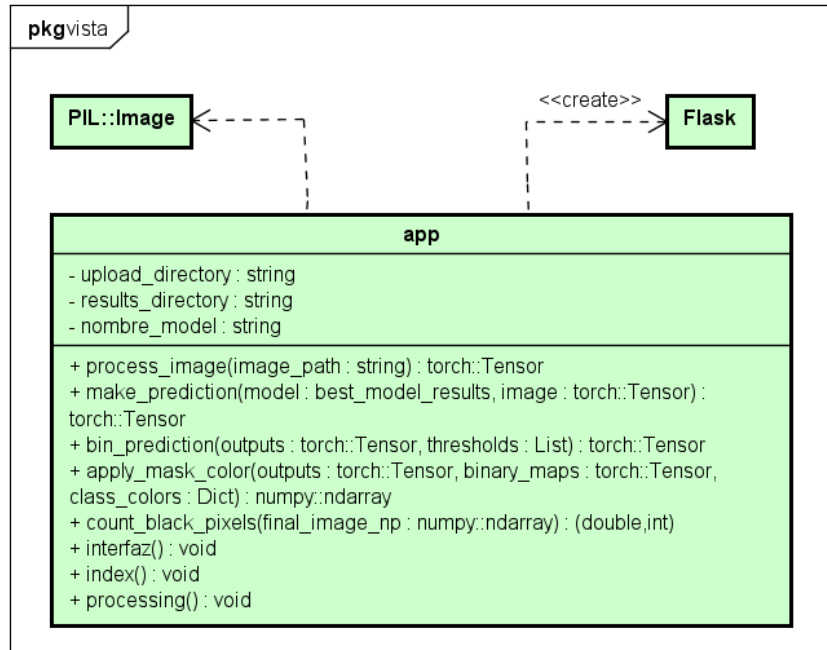


Figura 8.5: Diagrama Modules and Uses Style. Paquete: Vista.

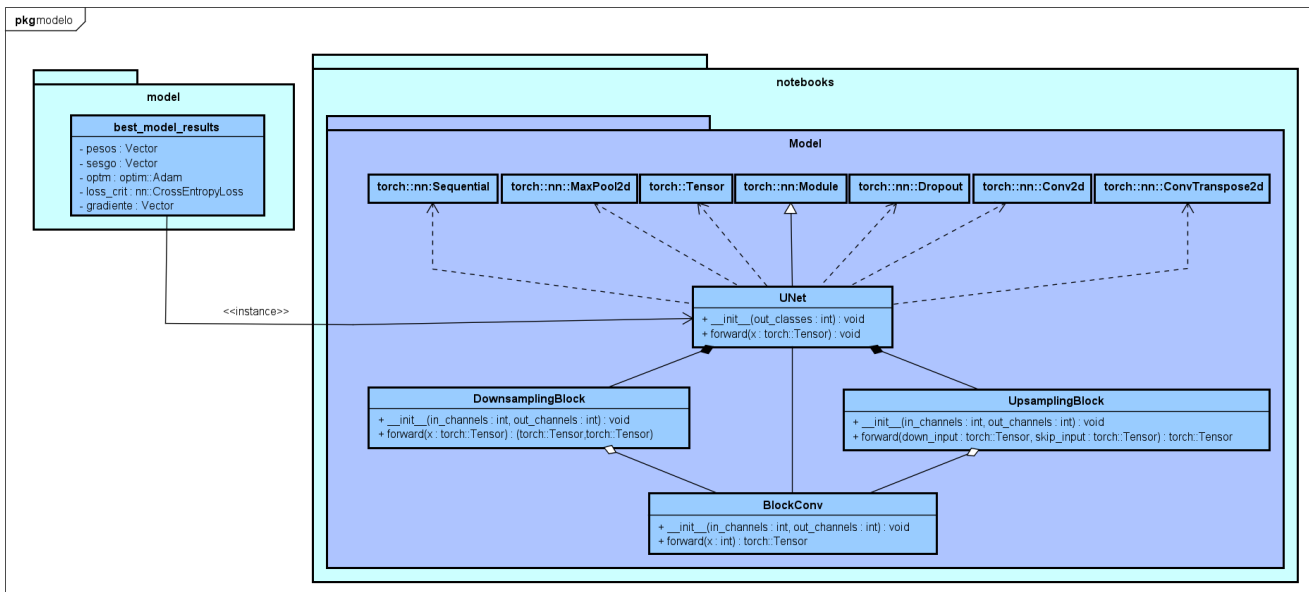


Figura 8.6: Diagrama Modules and Uses Style. Paquete: Modelo.

8.3.4. Diagramas de secuencia

En esta subsección, se presentan los diagramas de secuencia correspondientes a los casos de uso detallados expuestos en **Sección 8.2.2**, con el propósito de facilitar la comprensión y análisis de las interacciones entre los distintos elementos del sistema. Asimismo, permite plasmar de forma más clara los flujos de funcionamiento en cada uno de los escenarios.

Adicionalmente, incluimos, para el caso de uso principal, *Obtener predicción* y el diagrama *Uses Style*. De esta manera, detallamos de forma específica las dependencias y relaciones de la arquitectura en la realización del mismo.

Diagrama de secuencia: CU-01. Subir imagen

El CU-01 comprende, desde que el usuario accede a la interfaz principal, hasta el momento exacto en el que se hace click en el botón de enviar. Por ello, en este caso de uso, se cubre el caso alternativo de cancelar la subida de la imagen.

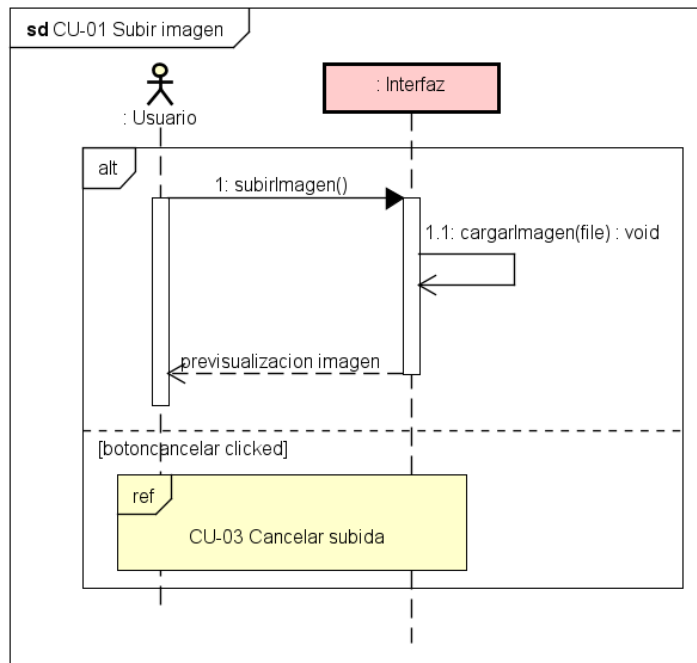


Figura 8.7: Diagrama de secuencia. CU-01: Subir imagen.

Uses Style: CU-02. Obtener predicción

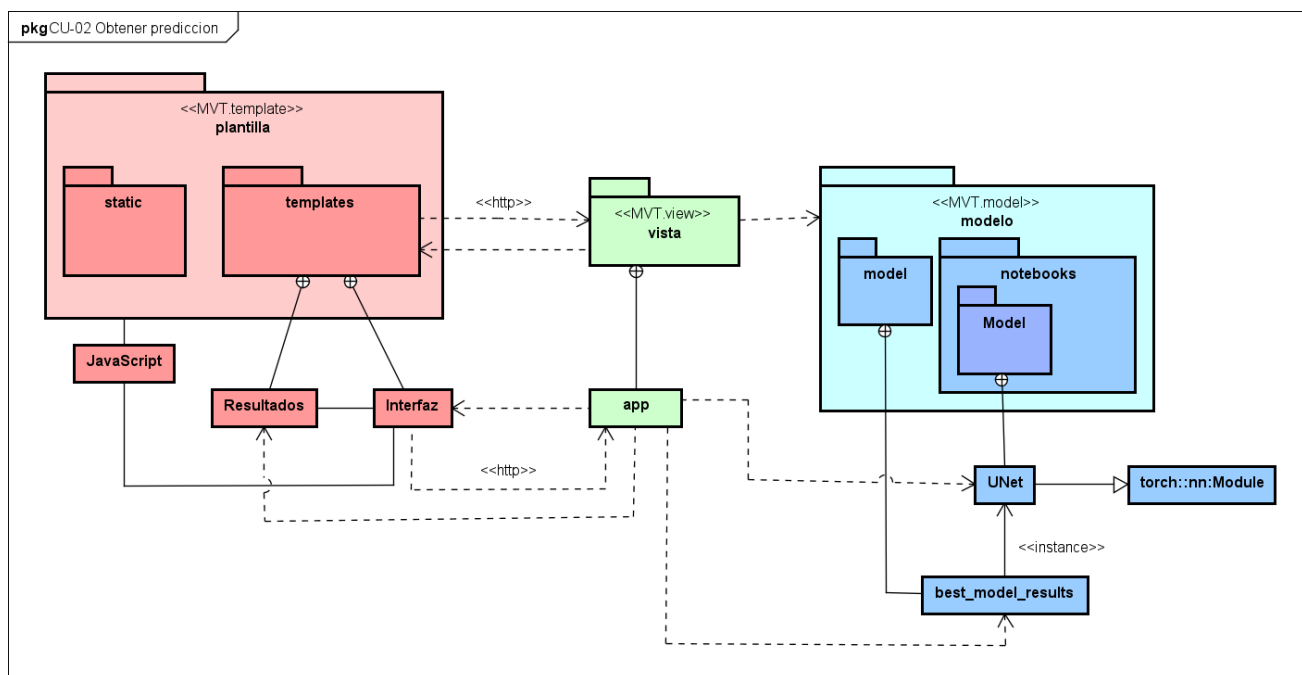


Figura 8.8: Uses Style CU-02. Obtener predicción.

Diagrama de secuencia: CU-02. Obtener predicción

Se ha optado por desglosar este caso de uso en dos diagramas de secuencia, facilitando el entendimiento de este y simplificando el diagrama de secuencia principal.

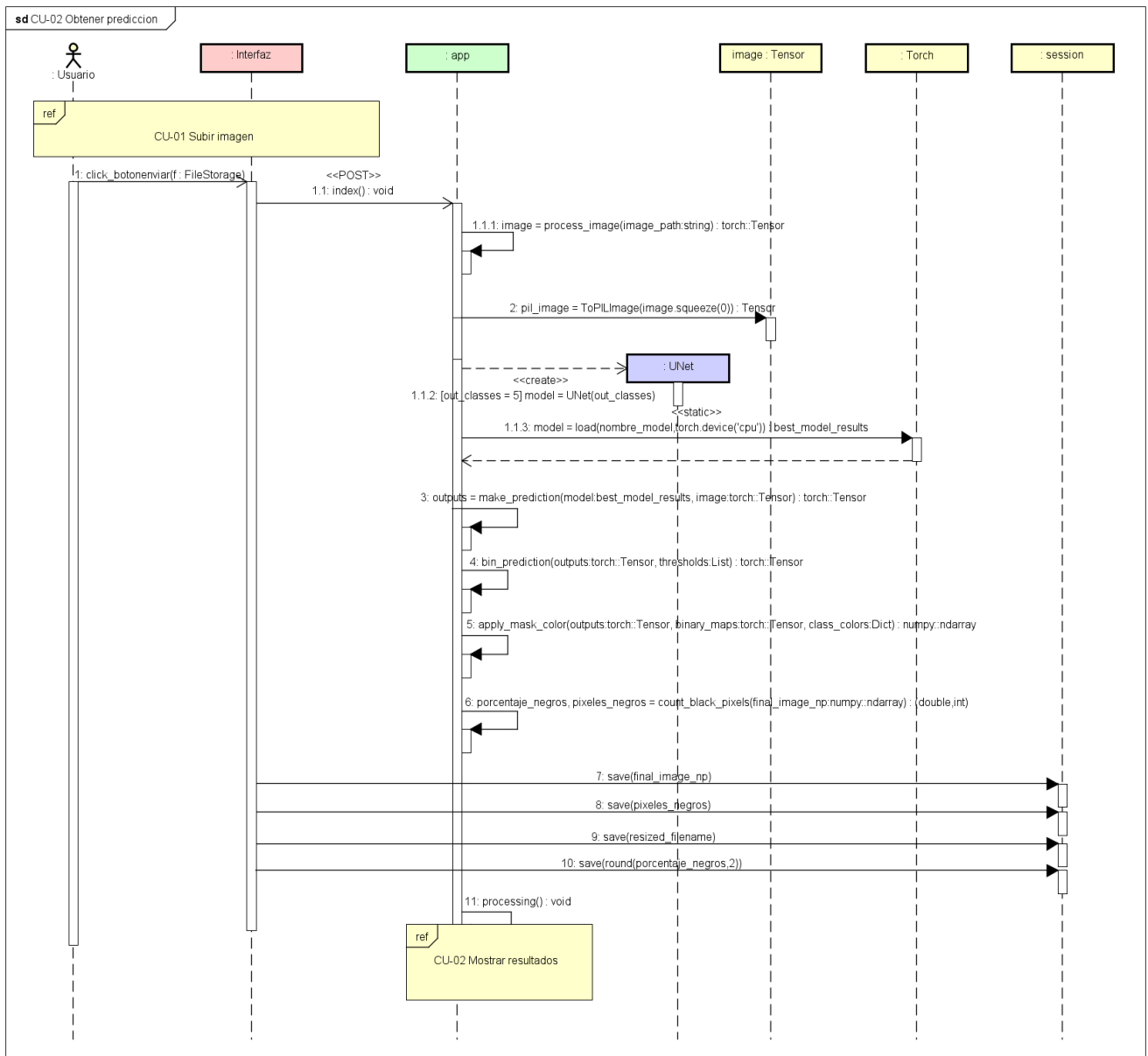


Figura 8.9: Diagrama de secuencia. CU-02. Obtener predicción.

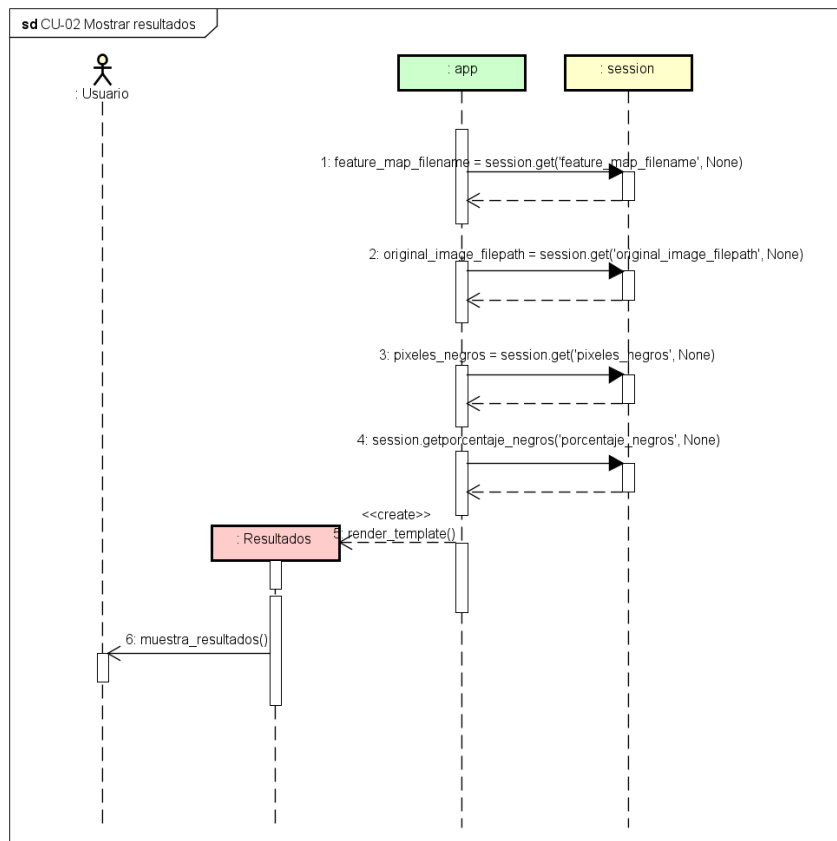


Figura 8.10: Diagrama de secuencia. CU-02. Mostrar resultados

Diagrama de secuencia: CU-03. Cancelar subida

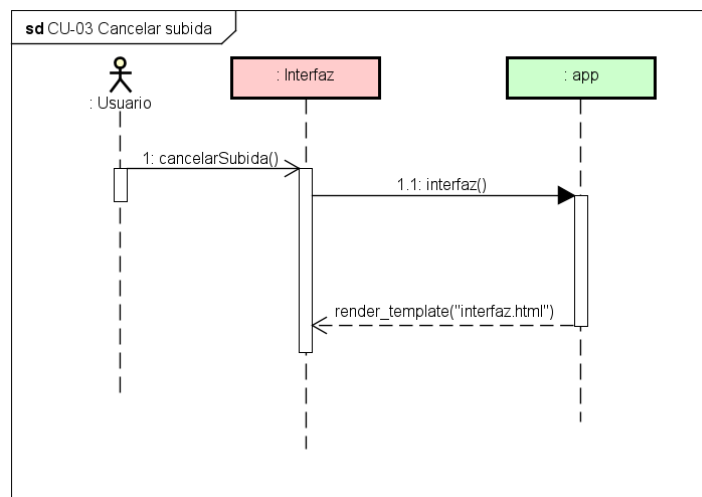


Figura 8.11: Diagrama de secuencia. CU-03: Cancelar subida.

8.4. Seguridad

En el desarrollo de aplicaciones web, la gestión de la seguridad y el tratamiento seguro de los datos son aspectos cruciales, especialmente si la aplicación se va a utilizar en un entorno real. En el TFG presente, la implementación

de medidas de seguridad avanzadas queda fuera del alcance del proyecto. El objetivo principal es desarrollar una aplicación, que permita a los usuarios realizar predicciones, no se va a emplear en un entorno real. Como medida de seguridad principal, se ha implementado una comprobación de la extensión del archivo cargado a través de código HTML, asegurando que sólo se acepten formatos de archivo adecuados para el procesamiento previsto.

8.5. Pruebas

Para verificar el correcto funcionamiento de la aplicación, no se ha desarrollado una batería de pruebas. En su lugar, dada la simplicidad de la aplicación, se han realizado pruebas manuales de los flujos de trabajo principales y alternativos descritos en los casos de uso detallados en 8.2.2. Como los resultados obtenidos y la experiencia de navegación coinciden con los comportamientos esperados, se considera que la aplicación funciona correctamente.

Capítulo 9

Conclusiones y líneas futuras

9.1. Aprendizaje

A lo largo de este TFG, que ha representado un desafío personal y un reto académico, he adquirido una considerable cantidad de conocimientos, principalmente en el ámbito del *Aprendizaje Automático* y la *Minería de Datos*. Éstos han permitido aumentar mi comprensión y mis habilidades en estas disciplinas, que han constituido la base fundamental para el desarrollo y progreso de este proyecto.

Debido a la amplitud de este TFG, he podido profundizar en diversas áreas y reforzar algunas de las habilidades y disciplinas de la Informática, que ya había desarrollado previamente durante mi grado. No solo he ampliado mi conocimiento teórico, sino que también he fortalecido mi capacidad para aplicar estos conceptos en la práctica, lo que considero un logro significativo. Algunas de las asignaturas del Grado relacionadas con el desempeño de estas habilidades son:

- **Planificación y diseño de sistemas computacionales.** Especialmente útil en el desarrollo de la planificación inicial del proyecto, así como en la elección del patrón de diseño, modelos y diagramas relativos a la Ingeniería de Software para el desarrollo de la aplicación.
- **Servicios y Sistemas Web** me ha permitido elaborar el desarrollo de la aplicación web final, de una forma mucho más intuitiva y sencilla.
- **Profesión y Sociedad.** Gracias a la realización de varios informes en esta asignatura, se ha fortalecido mi habilidad para expresar de manera efectiva ideas complejas y analizar información de manera sistemática. Esta destreza ha desempeñado un papel fundamental en la redacción precisa y estructurada de la presente memoria.
- **Fundamentos de Ingeniería de Software.** Junto a Planificación y Diseño de Sistemas Computacionales, las principales asignaturas donde he podido adquirir conocimiento sobre Ingeniería de Software. En conjunto, componen la base de todo lo relacionado con Ingeniería de Software en este TFG.

Si bien he resaltado la importancia de seis asignaturas específicas, es pertinente destacar que todas las materias cursadas, aunque no se mencionen de manera explícita, han contribuido ya sea, en mayor o menor medida, en la consolidación de unas bases fundamentales, tanto prácticas, como teóricas. Por tanto, todos los conocimientos adquiridos a lo largo de mi formación académica ha posibilitado con éxito el desarrollo integral de este TFG.

Asimismo, es relevante destacar las habilidades que he desarrollado de manera autónoma durante la realización de este trabajo. Desde sus inicios, este proyecto se planteó como una oportunidad para explorar a fondo la Visión por Computadora mediante el Aprendizaje Automático. A pesar de haber optado por la Mención en Computación y excluyendo las dos asignaturas troncales, debido a la amplitud del área y la limitación de tiempo, no se ha podido indagar demasiado en las Redes Neuronales Convolucionales. Son abordadas únicamente de manera práctica en la última parte de la asignatura de Minería de Datos, donde fueron tratadas con Keras en lugar de PyTorch, por su simplicidad. Esto ha implicado un reto importante, puesto que hemos necesitado conocer el funcionamiento de esta nueva biblioteca.

Dejando de lado parte central de este proyecto, la integración del modelo en la aplicación web y la creación de la misma, también representaron un desafío. Esto no fue tanto por el desarrollo de las interfaces o el estilo, sino por

la encapsulación dentro de un contenedor Docker, un aspecto que no se había abordado en la carrera. La creación de la aplicación con Flask, a pesar de su simplicidad, marcó mi primera experiencia en el desarrollo de aplicaciones web enmarcadas en un entorno Python.

9.2. Líneas futuras

Tras abordar con éxito los principales objetivos propuestos en la planificación inicial, la finalización de este TFG abre nuevos caminos, que permiten completar y mejorar el trabajo realizado, que no han podido abordarse debido a la limitación temporal de un TFG.

En este sentido cabría apuntar:

- La elaboración de un algoritmo con la capacidad de generar una área de aterrizaje segura y delimitada, ajustándose a las medidas de seguridad pertinentes. Este enfoque posibilitaría, conociendo las dimensiones del dron, la definición de áreas de aterrizaje con una precisión prácticamente infalible, garantizando un 100 % de seguridad. Se contempla la incorporación de un margen de seguridad entre las distintas clases detectables, en caso de ser viable. Asimismo, en situaciones que presenten múltiples opciones, se otorgará al usuario la capacidad de elegir la ubicación preferida para el aterrizaje.
- La creación de una herramienta web más completa, para que pudiese ser utilizada en un entorno real. Con un software que permitiera recibir imágenes en tiempo real de múltiples drones simultáneamente, combinado en el punto anteriormente expuesto. Esto permitiría efectuar el aterrizaje de los drones prácticamente de forma autónoma.
- Considerar otras arquitecturas y configuraciones de la red U-Net, tratando de obtener mejoras en los resultados obtenidos, sobre todo en la identificación de algunas de las clases como Movimiento u Obstáculos ya que, en ellas, obtenemos peores resultados.
- Aprovechar el reconocimiento de elementos adicionales en las imágenes, de forma que no se limite exclusivamente a garantizar un aterrizaje seguro. Otras zonas o elementos como la vegetación, o agua, abren perspectivas hacia diversas aplicaciones o tareas. Esta capacidad puede ser integrada en un software más robusto y completo, generando así una herramienta versátil que va más allá de la mera optimización del aterrizaje de drones. La identificación precisa de estos elementos proporciona una base sólida para la ejecución de diversas tareas, transformando la captura de imágenes en una herramienta multifuncional capaz de facilitar la realización eficiente de diferentes operaciones en entornos diversos.
- Disponer de una base de datos y un amplio equipo de trabajo. De esta forma podríamos disponer de un mayor número de imágenes y, con ello, generar un modelo mucho más apto y funcional para casos reales. En el conjunto de datos empleado, disponemos de los mapas Ground Truth, si queremos aumentar el tamaño del conjunto con nuevas imágenes, deberíamos disponer del equipo anteriormente mencionado que realizase con precisión y a mano cada uno de los Ground Truth para las nuevas imágenes. El coste de todo lo anterior se sale del contexto académico y sería muy elevado. Sin embargo, permitiría la creación de una aplicación mucho más realista y funcional. Este enfoque, aunque es costoso, podría proporcionar resultados significativamente mejores y más precisos, lo que a su vez podría conducir a avances significativos en el campo.

Apéndice A

Manual de instalación

En este anexo, se presentan los diferentes pasos, comandos y fases que hay que seguir para realizar la instalación y configuración en la máquina virtual, de las herramientas necesarias para desarrollar la aplicación de la Red Neuronal y la aplicación web.

Los objetivos básicos, que han de quedar cubiertos en la instalación, son:

- Disponer de un entorno de trabajo en Python, en particular Jupyter, como se ha comentado en la **Sección 6.1.3**.
- Instalar todas las librerías que se emplean en la construcción y desarrollo del modelo propuesto.
- Instalación de librerías que nos permitan llevar a cabo el despliegue web del modelo.
- Pasos para realizar la instalación del Docker.

Para poder instalar el entorno de Python que deseamos utilizaremos Anaconda. Que se puede instalar a través del siguiente enlace:

<https://docs.anaconda.com/free/anaconda/install/linux/>

En la máquina virtual cedida por la Escuela de Ingeniería Informática, esta herramienta ya está instalada, con la versión 23.7.4. Al igual que el lenguaje de programación Python, en su versión 3.11.5:

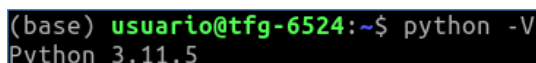
A terminal window with a black background and green text. The prompt is '(base) usuario@tfg-6524:~\$' and the command 'python -V' has been executed, resulting in the output 'Python 3.11.5'.

Figura A.1: Versión de Python.

Para la instalación del resto de bibliotecas utilizadas así como sus respectivas dependencias, utilizaremos la distribución Anaconda, para ello habrá que ejecutar los siguientes comandos:

- Pytorch (versión 2.0.1):
conda install pytorch
- Torchvision (versión 0.15.2)
conda install torchvision
- tqdm (versión 4.65.0)
conda install tqdm
- OpenCV (versión 4.8.1.78)
pip install opencv-contrib-python

Instalación Docker

Para habilitar el despliegue de la aplicación en un entorno Docker, primero debemos instalarlo en la máquina virtual. La instalación se llevó a cabo siguiendo la Documentación Oficial de Docker [87], así como otros recursos útiles, como la guía de DigitalOcean [88].

La versión de Docker instalada es la 24.0.7. Se incluyeron las versiones 0.11.2 de Docker Buildx y 2.21.0 de Docker Compose, como los principales plugins. Éstos son los pasos detallados para la instalación:

1. Primero actualizamos el sistema antes de instalar ningún nuevo paquete, ejecutando el comando `sudo apt-get update`
2. Instalamos Docker, a través del comando `sudo apt-get install docker-ce`. Importante tener en cuenta la versión de sistema operativo.
3. Instalación de los plugings. `sudo apt-get install docker-buildx-plugin docker-compose-plugin`.

Ahora, con Docker listo, procedemos a configurar el entorno para desplegar nuestra aplicación. Dentro de la carpeta que contiene todos los archivos de la aplicación, creamos un Dockerfile. Este archivo contiene instrucciones para construir automáticamente una imagen, que incluye todo lo necesario para ejecutar la aplicación. Ejecutamos el siguiente comando para construir la imagen:

```
sudo docker build -t app_landdetectia .
```

Con la imagen creada, podemos instanciar un contenedor, que es una instancia en ejecución de la imagen, para lanzar nuestra aplicación. Utilizamos el siguiente comando:

```
sudo docker run -it --name contenedor-app app_landdetectia.
```

Siguiendo estos pasos, podremos conseguir que la aplicación se ejecute de manera eficiente en un entorno Docker.

Apéndice B

Manual de usuario

Aquí se expone una guía de uso de la aplicación desarrollada. Tras la ejecución del comando que nos permite el acceso a la aplicación, vemos la pestaña principal de la aplicación, que se ve en la **Figura B.1**.

```
sudo docker run -it --name contenedor-app prueba_app
```

Llegados a este punto, se van a exponer los pasos para obtener una predicción sobre una imagen:

- Arrastrando una imagen al cuadro Drag & Drop, o bien haciendo click en el recuadro, abrirá un explorador de archivos que permitirá la selección de una imagen. Tras la selección de la imagen, se mostrará en este recuadro una previsualización de la imagen cargada, como se ve en la **Figura B.2**.
 - Posteriormente el usuario presiona el botón 'Enviar'. Ahora la imagen está siendo procesada por el modelo. Tardará apenas unos segundos en generar la predicción.
 - El usuario también tiene la posibilidad de presionar el botón 'Cancelar'. Con esto se anula la subida de la imagen proporcionada y permite al usuario cargar una nueva imagen.
- Finalmente, cuando ha generado la predicción definitiva, muestra los resultados en pantalla gracias a otra interfaz html, como vemos en la **Figura B.3**. En esta misma interfaz, se ofrece la posibilidad de realizar otra predicción haciendo uso del botón 'Volver'.

En la **Figura B.4** y **B.5**, vemos que haciendo un uso incorrecto de la aplicación se muestra un mensaje informando sobre el error.

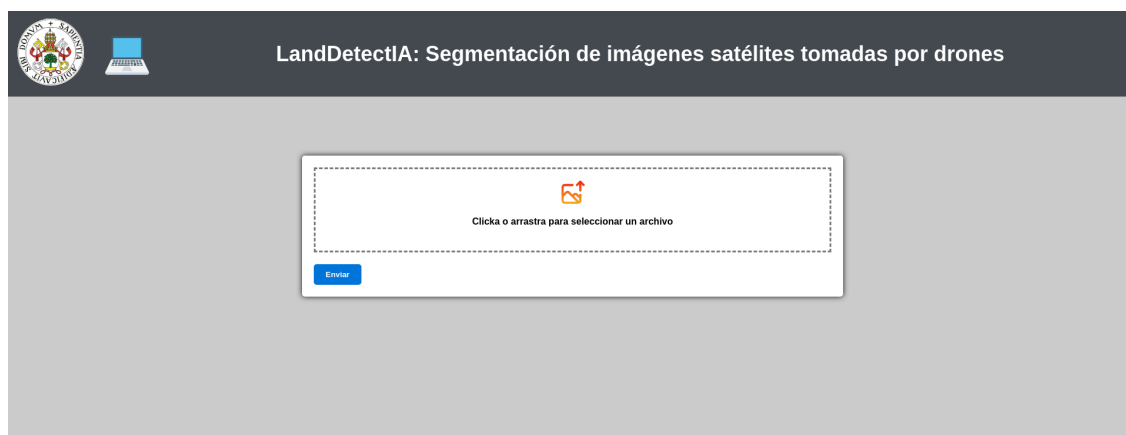


Figura B.1: Inicio de la aplicación.



Figura B.2: Previsualización de la imagen subida.

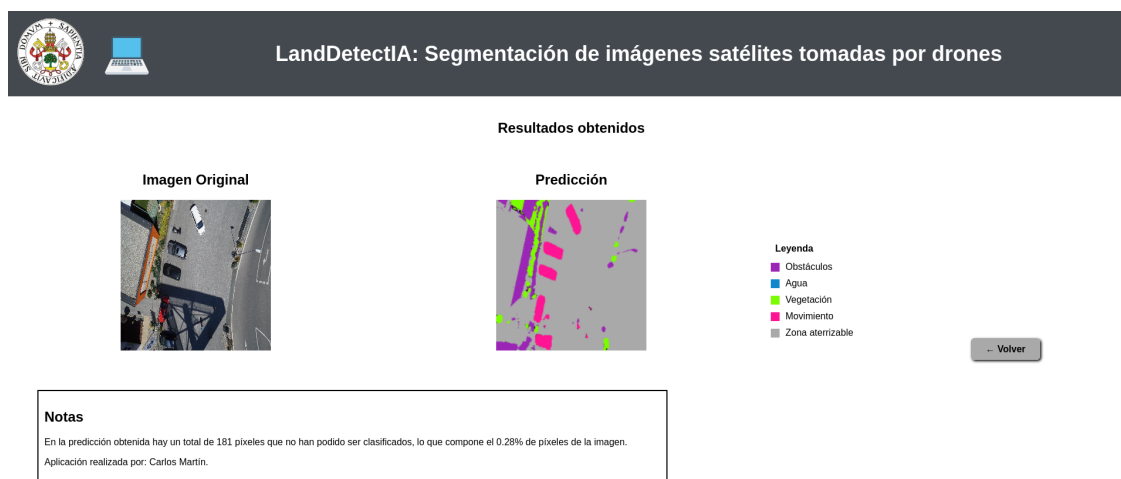


Figura B.3: Visualización de los resultados.

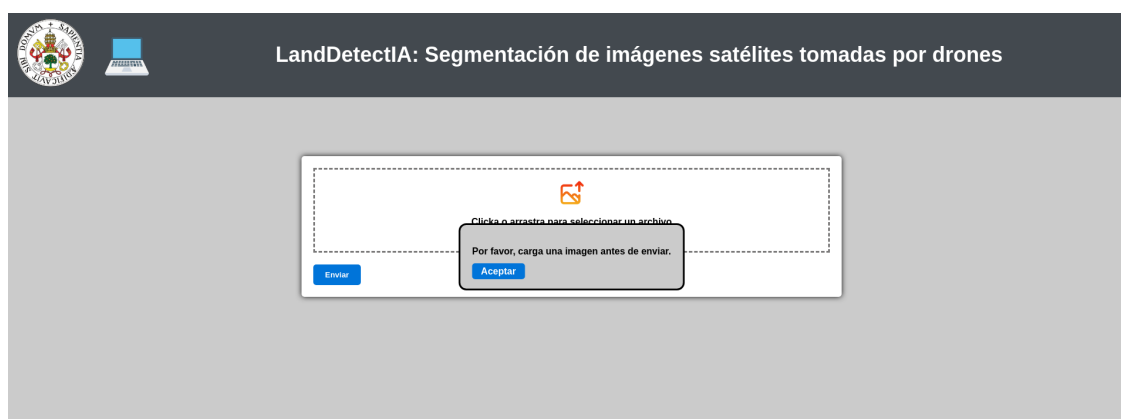


Figura B.4: Error cuando no se introduce ninguna imagen.

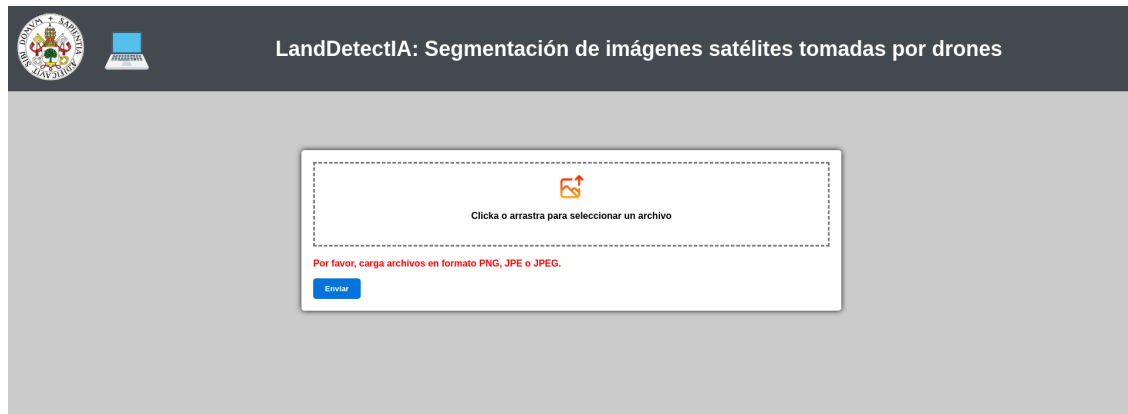


Figura B.5: Formato incorrecto del archivo cargado.

Apéndice C

Contenidos del CD-ROM

El objetivo de este apéndice es explicar el contenido del CD-ROM entregado. Asimismo, estos ficheros se encuentran en repositorio de la siguiente referencia [89]. En la siguiente imagen podemos ver un esquema, que nos plasma el contenido del mismo.

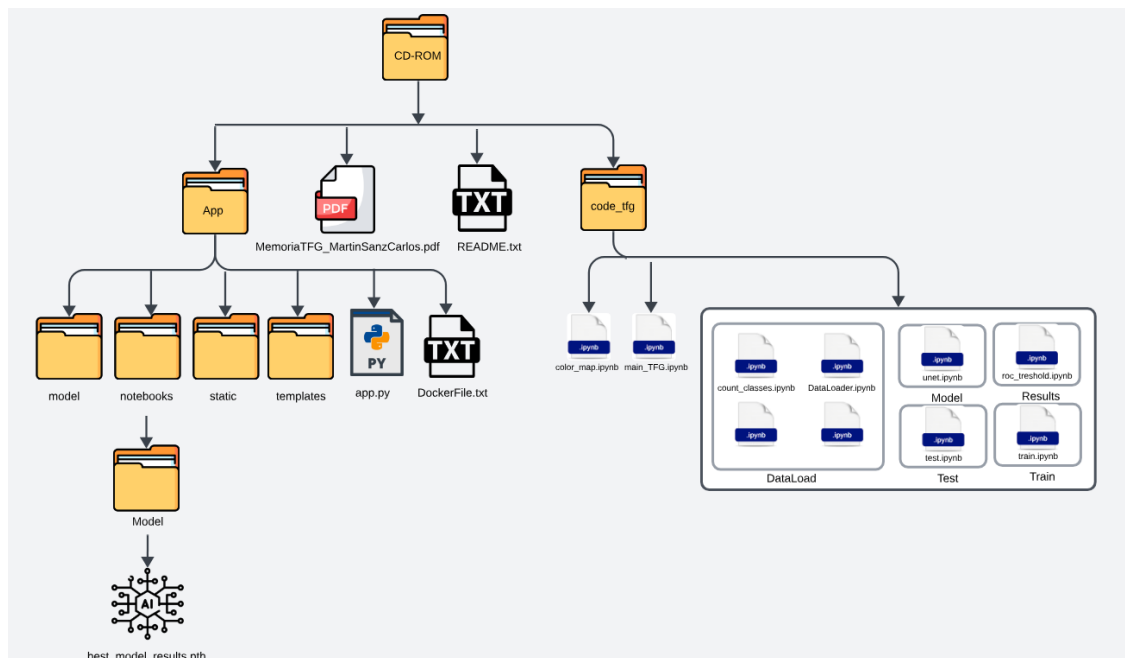


Figura C.1: Contenido del CD-ROM.

En el CD encontramos un README, con una breve descripción del proyecto y su contenido, esta memoria, y dos directorios, `App/` y `code_tfg/`.

El primero de estos directorios contiene a su vez otros cuatro:

- `model/`. Contiene el modelo preentrenado que consigue una mayor precisión de clasificación.
- `notebooks/Model/`. Contiene la clase UNET, que define la arquitectura del modelo.
- `static/`. Contiene las imágenes estáticas, el fichero de estilo y el código Javascript.
- `templates/`. Contiene las plantillas HTML de la aplicación.

Asimismo, tiene dos ficheros, `app.py` y `DockerFile.txt`, que constituyen el grueso funcional de la aplicación y el fichero de configuración del Docker.

El otro directorio contiene dos ficheros `colormap.ipynb` y `main_TFG.ipynb`. El primer cuaderno Jupyter define dos variables, para que puedan ser utilizadas en el resto de cuadernos. El segundo es el cuaderno sobre el que se han realizado las pruebas de entrenamientos, obtención de métricas y resultados.

El resto de cuadernos se encuentran organizados en directorios, cada `.ipynb` en uno. Para no sobrecargar el esquema de la **Figura C.1**, se han representado los directorios con un rectángulo gris. Vemos cinco directorios:

- **DataLoad**: cuatro ficheros `.ipynb`, para el preprocesamiento y preparación de los datos.
- **Model**: la clase que define la arquitectura del modelo.
- **Train**: el cuaderno `.ipynb` con la función que realiza el entrenamiento de una época.
- **Test**: el cuaderno `.ipynb` con la función que realiza la validación de una época.
- **Results**: un cuaderno `.ipynb` con las funciones que nos permiten obtener los resultados y métricas del modelo.

Apéndice D

Apéndice sobre PyTorch

Este es un breve apéndice, donde se recoge la información técnica sobre las clases empleadas en el desarrollo del modelo final del proyecto.

Con el fin de obtener una estructura más clara que nos permita entender mejor, cuándo se utiliza cada clase y en qué fase específica del desarrollo, voy a dividir la estructura de este apéndice en *Downsampling* y *Upsampling*.

D.1. torch.nn.Module

La clase `Module` [90] del paquete `torch.nn` constituye la base de la mayoría de los modelos de Redes Neuronales desarrollados con PyTorch. Proporciona una estructura limpia, sencilla y flexible para crear y gestionar modelos, facilitando así la implementación, el entrenamiento y otras tareas.

Como mínimo, esta clase implementa dos métodos, `__init__()` y `forward()`:

- `__init__()`: es el método constructor de la clase que se emplea en Python. En este contexto, se utiliza para configurar la inicialización del modelo. Se ejecuta una vez creada una instancia de nuestro modelo. Este método se emplea para definir las capas, parámetros y módulos de la red, además de poder especificar parámetros y configuraciones iniciales.
- `forward()`: este método establece cómo se realizan los cálculos a través del modelo, es decir, cómo se encuentra la salida a partir de la entrada. Se especifica la secuencia de operaciones, esto es, cómo avanza la entrada a través de las distintas capas y/o módulos de la red definidos en `__init__()`.

D.2. Downsampling - Fase de extracción

En este apartado, veremos las clases del paquete `nn`, que han sido empleadas para la realización de la etapa de extracción del modelo desarrollado.

D.2.1. torch.nn.Conv2d

La clase `Conv2d` [91] nos permite llevar a cabo una convolución 2D de un lote de imágenes que constan de múltiples canales. Para una capa con una entrada de tamaño (N, C_{in}, H, W) y una salida de dimensión $(N, C_{out}, H_{out}, W_{out})$, la respuesta de la capa se calcula de la siguiente manera:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) * \text{input}(N_i, k) \quad (\text{D.1})$$

Donde **N** es el tamaño del lote, **C** es el número de canales de entrada, **H** es la altura en píxeles de las imágenes de entrada, **W** es la anchura nuevamente en píxeles y ***** representa la operación de correlación cruzada en 2D.

Algunos de los principales parámetros de la función, que han sido utilizados en la creación de nuestro modelo con el propósito de mejorar su funcionamiento son:

- **in_channels**: número de canales de información de la imagen de entrada.
- **out_channels**: número de canales producidos por la convolución.
- **kernel_size**: tamaño del filtro.
- **stride**: tamaño del stride.
- **padding**: el tipo de padding que se añade en los bordes de la imagen de entrada.

D.2.2. torch.nn.BatchNorm2d

`BatchNorm2d` aplica normalización por lotes [92] a una entrada mini-lote de entradas 2D, que presentan una dimensión adicional de canales. En definitiva, es la clase del paquete `nn` que se encarga de realizar la normalización al lote de entrada. En la **Sección 3.6.1** se explica su funcionamiento de forma más generalizada.

La fórmula que utiliza para ello, es la siguiente:

$$\hat{y} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta \quad (D.2)$$

Donde $E[x]$ es la media y $Var[x]$ es la varianza de la entrada, γ y β son vectores de parámetros de aprendizaje del mismo tamaño que la entrada.

Algunos de los parámetros importantes son:

- **num_features**: dimensión de la entrada, representado como la C de una entrada (N, C, H, W) .
- **eps**: parámetro que proporciona estabilidad.
- **momentum**: permite, o no, hallar la varianza o media móvil acumulativa.

D.2.3. torch.nn.LeakyRelu

Esta clase nos permite aplicar función de activación LeakyReLU [93] a nuestro modelo, que se suele aplicar después de realizar la convolución y la normalización por lotes.

Esta clase tiene dos argumentos:

- **negative_slope**: representa el valor del coeficiente rectificador.
- **inplace**: parámetro booleano, en caso de valor `true`, permite almacenar el resultado de la operación en el mismo tensor, sin la necesidad de crear uno nuevo.

D.2.4. torch.nn.MaxPool2d

`MaxPool2d` [94] aplica Max Pooling (Sección 3.3.4) sobre una entrada que consta de múltiples canales. Para una capa con entrada de tamaño (N, C, H, W) , salida (N, C, H_{out}, W_{out}) y un filtro de tamaño (kH, kW) , la salida se puede describir como:

$$\text{out}(N_i, C_j) = \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n) \quad (D.3)$$

Donde N_i representa el ejemplo del lote, C_j el canal de salida, **stride[0]** y **stride[1]** los strides del eje vertical y horizontal respectivamente, **h** y **w** serían posiciones en el mapa de características de entrada. Por último, **m** y **n** son valores que se suman a las coordenadas calculadas según los strides.

Algunos parámetros principales son:

- **kernel_size**: tamaño del filtro sobre el que hay que extraer el valor máximo.
- **stride**: stride empleado por el kernel.
- **padding**: relleno con valores de $-\infty$ en ambos lados.

D.3. Upsampling - Fase de expansión

D.3.1. `torch.nn.ConvTranspose2d`

La clase `ConvTranspose2d` [95] nos proporciona la técnica de la convolución transpuesta para aplicar a nuestro modelo durante la fase de upsampling.

Los parámetros importantes son muy similares a los de la convolución normal, a excepción de dos:

- **output_padding:** tamaño del padding que se añade.
- **dilation:** espacio que se deja entre los elementos del filtro.

D.3.2. `torch.nn.Dropout`

Nos permite aplicar dropout [96], que como ya se vió en la **Sección 3.5.5**, es una técnica de regularización. Durante el entrenamiento establece de forma aleatoria a cero algunos elementos del tensor de entrada, con probabilidad p , consiguiendo prevenir la co-adaptación de neuronas y favoreciendo la regularización de la red.

El parámetro principal es p , que establece la probabilidad de que un elemento sea anulado.

D.3.3. `torch.cat`

Es la única clase [97] que se ha empleado en la creación del modelo que no pertenece al paquete `nn`. Permite concatenar una secuencia de tensores en la dimensión proporcionada. Para ello, todos los tensores han de tener, o la misma forma, o estar vacíos.

Los parámetros principales son:

- **tensors:** representa la secuencia de tensores que queremos concatenar.
- **dim:** es la dimensión sobre la que se concatenan los tensores.

D.3.4. `torch.optim.lr_scheduler.ReduceLROnPlateau`

Implementa una técnica de *Learning Rate Scheduler*, con el fin de refinar el ajuste de la tasa de aprendizaje. Específicamente, el método seleccionado reduce la tasa de aprendizaje en un factor variable entre 2 y 10 cuando la métrica de validación deja de mejorar.

Los principales parámetros de este planificador son:

- **optimizer:** donde proporcionamos el optimizador que se ve afectado.
- **factor:** es la reducción que se aplica en el learning rate cuando entra el planificador.
- **patience:** número consecutivo de épocas sin observar mejoras en la métrica establecida, para que se aplique la reducción.
- **threshold:** umbral que se utiliza para medir el nuevo óptimo y centrarse solamente en los cambios significativos.

Bibliografía

- [1] Arturo Ghinassi. *Semantic Segmentation Drone Dataset*. Kaggle. 2023. URL: <https://www.kaggle.com/datasets/santurini/semantic-segmentation-drone-dataset>. Last accessed: 10/10/2023.
- [2] EagleDron. *Los 10 usos principales de los drones*. <https://tinyurl.com/2x3pwfxl>. Last accessed: 24/10/2023.
- [3] GeeksforGeeks. *KDD Process in Data Mining*. URL: <https://www.geeksforgeeks.org/kdd-process-in-data-mining/>. Last accessed: 28/10/2023.
- [4] Jorge Romero. *Metodologías de Minería de Datos*. Jun, 2019. URL: <https://jorgeromero.net/metodologias-de-mineria-de-datos/>. Last accessed: 09/11/2023.
- [5] Sngular. *Data Science: CRISP-DM Metodología*. <https://www.sngular.com/es/data-science-crisp-dm-metodologia/>. Last accessed: 09/11/2023.
- [6] Asana. *Waterfall Project Management Methodology*. <https://asana.com/es/resources/waterfall-project-management-methodology>. 2022. Last accessed: 09/11/2023.
- [7] Pete Chapman et al. *CRISP-DM 1.0: Step-by-step data mining guide*. Inf. téc. NCR, SPSS, DaimlerChrysler, 2000. URL: <https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2012-13/kdd/files/CRISPWP-0800.pdf>. Last accessed: 09/11/2023.
- [8] Wikipedia. *Source of CRISP-DM Methodology Image*. 29/09/2023. URL: https://es.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining. Last accessed: 09/11/2023.
- [9] Universidad de Valladolid. *Guía docente del trabajo de fin de grado (Mención en Computación)*. URL: https://apps.stic.uva.es/guias_docentes/uploads/2023/545/46978/1/Documento.pdf. Last accessed: 10/11/2023.
- [10] SoftExpert. *Guía Práctica para la Gestión de Riesgos - 12 Etapas*. URL: <https://blog.softexpert.com/es/guia-practica-plan-riesgos-12-etapas/>. Last accessed: 12/11/2023.
- [11] PC Componentes. *Asus TUF Gaming F15 FX506LHB-HN359 Intel Core i5-10300H 16GB 512GB SSD GTX 1650 15.6"*. URL: <https://www.pccomponentes.com/asus-tuf-gaming-f15-fx506lhb-hn359-intel-core-i5-10300h-16gb-512gb-ssd-gtx-1650-156>. Last accessed: 12/11/2023.
- [12] Intel. *Intel Xeon Gold 6326 Processor 24M Cache 2.90 GHz - Specifications*. URL: <https://www.intel.la/content/www/xl/es/products/sku/215274/intel-xeon-gold-6326-processor-24m-cache-2-90-ghz/specifications.html>. Last accessed: 02/12/2023.
- [13] PC Componentes. *Memorias RAM DDR4*. 2023. URL: https://www.pccomponentes.com/memorias-ram-ddr4?campaigntype=dsa&campaignchannel=busqueda&gad_source=1&gclid=CjwKCAiA6byqBhAWEiwAnGCA4LsbCxdj_kYAImlmn47xVvZuoX1qKJt2_xoCunkQAvD_BwE. Last accessed: 10/11/2023.
- [14] Wikipedia. *Neurona de McCulloch-Pitts*. URL: https://es.wikipedia.org/wiki/Neurona_de_McCulloch-Pitts. Last accessed: 28/11/2023.
- [15] Carmelo Bonilla Carrión. *Trabajo Fin de Grado: Redes Convolucionales*. Jun 2020. URL: <https://idus.us.es/bitstream/handle/11441/115221/TFG%20DGM%20Bonilla%20Carri%C3%B3n%20Carmelo.pdf?sequence=1&isAllowed=y>. Last accessed: 25/11/2023.
- [16] Juan Mellado. *Imagen de Neurona Biológica*. Aug 2019. URL: <https://inmensia.com/blog/20190826/redes-neuronales-1/>. Last accessed: 22/11/2023.
- [17] *Imagen modelo McCulloch-Pitts*. 2021. URL: https://dcain.etsin.upm.es/~carlos/bookAA/05.1_RedNeuralesIntroduccion.html. Last accessed: 22/11/2023.
- [18] Wikipedia. *Paul Werbos*. March 2021. URL: https://es.wikipedia.org/wiki/Paul_Werbos. Last accessed: 02/12/2023.

- [19] IT Masters. *En qué consiste el Deep Learning*. Oct 2023. URL: <https://www.itmastersmag.com/noticias-analisis/en-que-consiste-el-deep-learning/>. Last accessed: 01/12/2023.
- [20] Hiberus. *Machine Learning*. URL: <https://www.hiberus.com/tecnologias-diferenciales/machine-learning>. Last accessed: 25/11/2023.
- [21] *Imagen relacionada con artículo en LinkedIn*. URL: https://media.licdn.com/dms/image/C4E12AQHqA1bxq5Hgrg/article-cover_image-shrink_600_2000/0/1572047824059?e=2147483647&v=beta&t=JgVYcrpHUozpgLhjg8cfvKlY0z2aJqA3vtLM0. Last accessed: 25/11/2023.
- [22] Anirudh Edpuganti. *Multi-Layer Perceptron*. URL: <https://iq.opengenus.org/multi-layer-perceptron/>. Last accessed: 26/11/2023.
- [23] *Deep Learning*. URL: <https://es.mathworks.com/discovery/deep-learning.html>. Last accessed: 26/11/2023.
- [24] Kunihiro Fukushima. *Neocognitron: A hierarchical neural network capable of visual pattern recognition*. URL: <https://www.sciencedirect.com/science/article/abs/pii/0893608088900147>. Last accessed: 27/11/2023.
- [25] *Red Neuronal Convolutiva*. Jan, 2023. URL: https://es.wikipedia.org/wiki/Red_neuronal_convolutiva. Last accessed: 27/11/2023.
- [26] Wikipedia. *Yann LeCun*. Sep, 2023. URL: https://es.wikipedia.org/wiki/Yann_LeCun. Last accessed: 27/11/2023.
- [27] IBM. *Computer Vision*. URL: <https://www.ibm.com/es-es/topics/computer-vision>. Last accessed: 27/11/2023.
- [28] IBM. *Convolutional Neural Networks*. URL: <https://www.ibm.com/es-es/topics/convolutional-neural-networks>. Last accessed: 27/11/2023.
- [29] *Convolución*. URL: <https://glossary.slb.com/es/terms/c/convolution>. Last accessed: 28/11/2022.
- [30] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Convolutional Networks*. URL: <https://www.deeplearningbook.org/>. Last accessed: 04/12/2023.
- [31] Wikipedia. *Convolución*. Mar, 2023. URL: <https://es.wikipedia.org/wiki/Convoluci%C3%B3n>. Last accessed: 27/11/2023.
- [32] Miguel Sotaquirá. *Convolución en Redes Convolucionales*. Mar, 2019. URL: <https://www.codificandobits.com/blog/convolucion-redes-convolucionales/>. Last accessed: 28/11/2023.
- [33] Rodrigo Cabello. *Vision Transformer*. Feb, 2022. URL: <https://www.plainconcepts.com/es/vision-transformer/>. Last accessed: 28/11/2023.
- [34] Pablo Soto. *Presentación de Pablo Soto*. URL: https://www.tec.ac.cr/sites/default/files/media/uploads/presentacion_pablosoto.pdf. Last accessed: 15/11/2023.
- [35] MathWorks. *Tipos de Imágenes*. URL: https://es.mathworks.com/help/matlab/creating_plots/image-types.html. Last accessed: 15/11/2023.
- [36] Thom Lane. *Multi-Channel Convolutions explained with... MS Excel!* Medium. Oct, 2018. URL: <https://medium.com/apache-mxnet/multi-channel-convolutions-explained-with-ms-excel-9bbf8eb77108>. Last accessed: 04/12/2023.
- [37] Miguel Sotaquirá. *Padding, Strides, Maxpooling y Stacking en Redes Convolucionales*. Apr, 2019. URL: <https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/>. Last accessed: 20/11/2023.
- [38] Make Your Own Neural Network. *Calculating Output Size of Convolutions*. Feb, 2020. URL: <http://makeyourownneuralnetwork.blogspot.com/2020/02/calculating-output-size-of-convolutions.html>. Last accessed: 20/11/2023.
- [39] Georgios Nanos. *Padding in Deep Neural Networks*. Jun, 2023. URL: <https://www.baeldung.com/cs/deep-neural-networks-padding#:~:text=Padding%20is%20a%20technique%20employed,of%20the%20deep%20learning%20model..> Last accessed: 28/11/2023.
- [40] DataHacker.rs. *What Is Padding in Convolutional Neural Networks?* Nov, 2018. URL: <https://datahacker.rs/what-is-padding-cnn/>. Last accessed: 28/11/2023.
- [41] Devashree Madhugiri. *Pooling Layer*. Sep, 2023. URL: <https://www.knowledgehut.com/blog/data-science/pooling-layer>. Last accessed: 28/11/2023.

- [42] Papers with Code. *Max Pooling*. URL: <https://paperswithcode.com/method/max-pooling>. Last accessed: 28/11/2023.
- [43] Jun94 DevPblog. *Unsampling, Unpooling, and Transpose Convolution*. Dec, 2020. URL: <https://medium.com/jun94-devpblog/d1-12-unsampling-unpooling-and-transpose-convolution-831dc53687ce>. Last accessed: 29/11/2023.
- [44] *Example of an Unpooling Process: Indices of Max Pooling Are Kept Up and Reused To...* URL: https://www.researchgate.net/figure/Example-of-an-unpooling-process-Indices-of-max-pooling-are-kept-up-and-reused-to_fig4_335754645. Last accessed: 30/11/2023.
- [45] Aqeel Anwar. *What Is a Transposed Convolutional Layer?* Mar, 2020. URL: <https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11>. Last accessed: 29/11/2023.
- [46] vdumoulin. *A Guide to Convolution Arithmetic for Deep Learning*. Apr, 2019. URL: https://github.com/vdumoulin/conv_arithmetic. Last accessed: 29/11/2023.
- [47] Wikipedia. *Bilinear Interpolation*. Aug, 2023. URL: https://en.wikipedia.org/wiki/Bilinear_interpolation. Last accessed: 30/11/2023.
- [48] Thomas Brox Olaf Ronneberger Philipp Fischer. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. May, 2015. URL: <https://arxiv.org/abs/1505.04597>. Last accessed: 01/12/2023.
- [49] Dr. Cem Gazioglu. *Comparison of Fully Convolutional Networks (FCN) and U-Net for Road Segmentation from High Resolution Imageries*. Dec, 2020. URL: <https://dergipark.org.tr/tr/download/article-file/1105454>. Last accessed: 30/11/2023.
- [50] Jeremy Zhang. *U-Net: Line by Line Explanation*. Oct, 2019. URL: <https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5>. Last accessed: 01/12/2023.
- [51] rupert ai. *The U-Net (actually) explained in 10 minutes*. 2023. URL: https://www.youtube.com/watch?v=NhdzGfB1q74&ab_channel=rupertai. Last accessed: 21/11/2023.
- [52] Diego Calvo. *Función de Activación en Redes Neuronales*. Dec, 2018. URL: <https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>. Last accessed: 01/12/2023.
- [53] *La Función Sigmoide*. URL: <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/la-funcion-sigmoide>. Last accessed: 01/12/2023.
- [54] *Regresión Softmax*. URL: <https://interactivechaos.com/es/manual/tutorial-de-machine-learning/regresion-softmax>. Last accessed: 01/12/2023.
- [55] Jorge Calvo. *Crear una Red Neuronal desde las Matemáticas*. URL: <https://www.europeanvalley.es/noticias/crear-red-neuronal-desde-las-matematicas/>. Last accessed: 01/12/2023.
- [56] Miguel Sotaquirá. *Funciones de Activación en Redes Neuronales*. Sep, 2018. URL: <https://www.codificandobits.com/blog/funcion-de-activacion/>. Last accessed: 01/12/2023.
- [57] PyTorch Contributors. *PyTorch Documentation - LeakyReLU*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html>. Last accessed: 03/12/2023.
- [58] Ravindra Parmar. *Common Loss Functions in Machine Learning*. Sep, 2018. URL: <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23>. Last accessed: 02/12/2023.
- [59] Sanket Doshi. *Optimizers for Training Neural Networks*. Jan, 2019. URL: <https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6>. Last accessed: 02/12/2023.
- [60] Vitaly Bushaev. *Adam: Latest Trends in Deep Learning Optimization*. Oct, 2018. URL: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>. Last accessed: 02/12/2023.
- [61] suvratara06. *Weight Initialization Techniques for Deep Neural Networks*. URL: <https://www.geeksforgeeks.org/weight-initialization-techniques-for-deep-neural-networks/>. Last accessed: 02/12/2023.
- [62] *Kaiming He's Personal Website*. May, 2009. URL: <https://kaiminghe.github.io/>. Last accessed: 02/12/2023.
- [63] Anand Borad. *Regularization: Make your Machine Learning Algorithms Learn, Not Memorize*. eInfoChips. Oct, 2022. URL: <https://www.einfochips.com/blog/regularization-make-your-machine-learning-algorithms-learn-not-memorize/#:~:text=There%20are%20three%20main%20regularization,Dropout>. Last accessed: 18/11/2023.
- [64] Statistics How To. *Tuning Parameter: Simple Definition, Examples*. 2023. URL: <https://www.statisticshowto.com/tuning-parameter/>. Last accessed: 18/11/2023.

- [65] Amar Budhiraja. *Learning Less to Learn Better: Dropout in Deep Machine Learning*. Dec, 2016. URL: <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>. Last accessed: 04/12/2023.
- [66] Wikipedia. *Vanishing gradient problem*. Sep, 2023. URL: https://en.wikipedia.org/wiki/Vanishing_gradient_problem. Last accessed: 18/11/2023.
- [67] François Chollet. *Deep Learning with Python*. Manning Publications, 2021, págs. 251-259. Last accessed: 18/11/2023.
- [68] Sumant Hegde. *An Introduction to Separable Convolutions*. Analytics Vidhya. Nov, 2021. URL: <https://www.analyticsvidhya.com/blog/2021/11/an-introduction-to-separable-convolutions/>. Last accessed: 19/11/2023.
- [69] Amazon Web Services (AWS). *¿Qué es el sobreajuste?* URL: <https://aws.amazon.com/es/what-is/overfitting/#:~:text=El%20sobreajuste%20es%20un%20comportamiento,no%20para%20los%20datos%20nuevos..> Last accessed: 19/11/2023.
- [70] dewangNautiyal. *ML — Underfitting and Overfitting*. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>. Last accessed: 19/11/2023.
- [71] TU Graz. *ICG - DroneDataset*. URL: <https://www.tugraz.at/index.php?id=22387>. Last accessed: 17/11/2023.
- [72] Shittu Olumide Ayodeji. *What is Cross Entropy Loss in PyTorch? - Educative*. educative.io. URL: <https://www.educative.io/answers/what-is-cross-entropy-loss-in-pytorch>. Last accessed: 04/12/2023.
- [73] Ligdi González. *¿Por qué usar Python para Machine Learning?* Aprende IA. Apr, 2022. URL: <https://aprendeia.com/por-que-python-para-machine-learning/>. Last accessed: 21/11/2023.
- [74] Wikipedia. *Anaconda (distribución de Python)*. Jun, 2023. URL: [https://es.wikipedia.org/wiki/Anaconda_\(distribuci%C3%B3n_de_Python\)](https://es.wikipedia.org/wiki/Anaconda_(distribuci%C3%B3n_de_Python)). Last accessed: 18/10/2023.
- [75] Roberto Díaz. *Aprende a utilizar Jupyter Notebook para tus proyectos de Ciencia de Datos*. The Machine Learners. URL: <https://www.themachinelearners.com/jupyter-notebook/>. Last accessed: 18/10/2023.
- [76] Oracle DevLive. *¿Qué es PyTorch: una guía completa*. May, 2022. URL: <https://developer.oracle.com/es/learn/technical-articles/what-is-pytorch>. Last accessed: 18/10/2023.
- [77] John Terra. *Keras vs TensorFlow vs PyTorch: Key Differences Among Deep Learning*. Simplilearn. Aug, 2023. URL: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#what_is_pytorch. Last accessed: 18/10/2023.
- [78] Adri Nerja. *¿Qué es Overleaf? Ventajas y para qué sirve*. URL: <https://www.adrinerja.com/que-es-overleaf/>. Last accessed: 18/10/2023.
- [79] José Domingo Muñoz. *¿Qué es Flask y qué ventajas ofrece?* OpenWebinars. Nov, 2017. URL: <https://openwebinars.net/blog/que-es-flask/>. Last accessed: 18/10/2023.
- [80] Yúbal Fernández. *¿Qué es Github y qué es lo que le ofrece a los desarrolladores?* Xataka. Oct, 2019. URL: <https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores>. Last accessed: 18/10/2023.
- [81] Wikipedia. *GanttProject*. Nov, 2023. URL: <https://es.wikipedia.org/wiki/GanttProject>. Last accessed: 18/10/2023.
- [82] Redacción KeepCoding. *Mínimos en data mining: mínimo local y global*. Accessed: 15/12/2023. URL: <https://keepcoding.io/blog/minimos-en-data-mining-global-y-local/>.
- [83] Gowtham S R. *Confusion Matrix to No Confusion Matrix in Just 5mins*. Jun, 2022. URL: <https://pub.towardsai.net/confusion-matrix-179b9c758b55>. Last accessed: 05/12/2023.
- [84] Fernando Izco. *Base de datos corporativa de personas - Métricas*. Nov, 2018. URL: https://bookdown.org/f_izco/BDC-POC/metricas.html. Last accessed: 05/12/2023.
- [85] Jorge San José Lorza. *DocScan: Deep Learning para la segmentación de documentos escaneados*. Director o Tutor: Calonge Cano, Teodoro. 2021. URL: <https://uvadoc.uva.es/handle/10324/50455>. Last accessed: 05/12/2023.
- [86] Mario Izquierdo Álvarez. “Redes convolucionales 2D en Pytorch: clasificación de imágenes de TAC de retina (OCT)”. Accessed: 12/12/2023. Trabajo de Fin de Grado. Valladolid, España: Universidad de Valladolid, Escuela de Ingeniería Informática de Valladolid, 2023. URL: <https://uvadoc.uva.es/handle/10324/62935>.

- [87] Docker Documentation Team. *Install Docker Engine on Ubuntu*. 2023. URL: <https://docs.docker.com/engine/install/ubuntu/>. Last accessed: 24/11/2023.
- [88] Brian Hogan. *Cómo instalar y utilizar Docker en Ubuntu 20.04*. 2020. URL: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04-es>. Last accessed: 24/11/2023.
- [89] Carlos Martín Sanz. *Trabajo de Fin de Grado*. Accessed: 23/02/2024. URL: <https://github.com/CarlosM2807/TrabajoFindeGrado2023-2024>.
- [90] *Module — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.Module.html>. Last accessed: 03/12/2023.
- [91] *Conv2d — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html>. Last accessed: 03/12/2023.
- [92] *BatchNorm2d — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm2d.html>. Last accessed: 03/12/2023.
- [93] *LeakyReLU — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html>. Last accessed: 03/12/2023.
- [94] *MaxPool2d — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html>. Last accessed: 03/12/2023.
- [95] *ConvTranspose2d — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html>. Last accessed: 03/12/2023.
- [96] *Dropout — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html>. Last accessed: 03/12/2023.
- [97] *torch.cat — PyTorch 2.1 documentation*. URL: <https://pytorch.org/docs/stable/generated/torch.cat.html>. Last accessed: 03/12/2023.