
Análisis de diferentes técnicas de segmentación semántica
Analysis of Different Semantic Segmentation Techniques



Trabajo de Fin de Máster
Curso 2022–2023

Autor
Alejandro Aizel Boto

Director
Gonzalo Pajares Martinsanz

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Análisis de diferentes técnicas de segmentación semántica

Analysis of Different Semantic Segmentation Techniques

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de Ingeniería del Software e Inteligencia Artificial

Autor
Alejandro Aizel Boto

Director
Gonzalo Pajares Martinsanz

Convocatoria: Septiembre 2023
Calificación:

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

4 de septiembre de 2023

Autorización de difusión

El abajo firmante, matriculado en el Máster en Ingeniería en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: “Análisis de diferentes técnicas de segmentación semántica”, realizado durante el curso académico 2022-2023 bajo la dirección de Gonzalo Pajares Martinsanz en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Alejandro Aizel Boto

4 de septiembre de 2023

Dedicatoria

Me gustaría dedicar este trabajo a mi familia por ser un gran apoyo durante todos estos años y por estar siempre ahí. Este logro es también vuestro.

Gracias por todo.

Agradecimientos

A mi tutor Gonzalo Pajares Martinsanz por ayudarme a elegir el tema del trabajo y a buscar una solución cuando no conseguía avanzar. Por ayudarme a elegir el contenido para la memoria y solventar todas las dudas sobre la teoría y la organización de la misma. Muchas gracias.

Resumen

Análisis de diferentes técnicas de segmentación semántica

La IA ha evolucionado durante todos estos años hasta convertirse en una herramienta prácticamente indispensable en nuestro día a día. Esta evolución ha permitido avances en campos tales como la medicina, la conducción o la agricultura, por citar solo algunos, que han cambiado la forma en la que los humanos interaccionan con la tecnología.

Este proyecto nace con el objetivo de estudiar y entender los conceptos teóricos que subyacen tras el paradigma del Deep Learnign (DL) y concretamente en el ámbito de lo que se conoce como segmentación semántica. La segmentación semántica es una técnica que permite clasificar a nivel de pixel una imagen en sus diferentes componentes, diferenciando distintas áreas pertenecientes a una misma clase.

En el proyecto se realiza la implementación de una Red Neuronal Convolutional (RNC) que permite detectar diferentes zonas de imágenes agrícolas para su clasificación. Con ello se estudian varios modelos de redes de segmentación semántica como SegNet, Unet y DeepLab3 que se comparan entre sí para elegir la que mejor resultado ofrece y sobre ella se realizan las mejoras pertinentes con el fin de poder ajustarla y mejorar los resultados iniciales.

Palabras clave

Inteligencia Artificial, Redes neuronales, Aprendizaje Automático, Segmentación Semántica, Redes Neuronales Convolucionales, Codificador-Decodificador, Matlab.

Abstract

Analysis of Different Semantic Segmentation Techniques

AI has evolved over the years to become an almost indispensable tool in our daily lives. This evolution has enabled advances in fields such as medicine, transportation and agriculture, to name just a few, which have changed the way humans interact with technology.

This project aims to study and understand the theoretical concepts underlying the paradigm of DL, specifically in the field known as semantic segmentation. Semantic segmentation is a technique that allows pixel-level classification of an image, distinguishing different areas belonging to the same class.

The project implements a Convolutional Neural Network (CNN) that detects different regions in agricultural images for classification. Several models of semantic segmentation networks such as SegNet, Unet, and DeepLab3 are compared to choose the one that provides the best results, and necessary improvements are made to fine-tune it and enhance the initial results.

Keywords

Artificial Intelligence, Neural Networks, Machine Learning, Semantic Segmentation, Convolutional Neural Networks, Encoder-Decoder, Matlab.

Índice

1. Introducción	1
1.1. Motivación	2
1.2. Objetivos	5
1.3. Organización de la memoria	6
1. Introduction	7
1.1. Motivation	8
1.2. Objectives	11
1.3. Thesis Organization	12
2. Aprendizaje Automático: bases del Aprendizaje Profundo	13
2.1. Inteligencia Artificial	14
2.2. Deep Learning	14
2.2.1. Redes neuronales biológica	15
2.2.2. Redes neuronales artificiales	16
2.3. Aprendizaje de las Red Neuronal (RN)	18
2.3.1. Función de coste	19
2.3.2. Método del gradiente descendente	21
2.3.3. Retro-propagación: backpropagation	23

2.4. Validación de la RN	26
2.4.1. Elección de los conjuntos de datos	26
2.4.2. Generalización de la RN	27
2.4.3. Hiperparámetros	28
3. Redes Neuronales Convolucionales	31
3.1. Estructura de las Redes Neuronales Convolucionales	33
3.1.1. Capa de convolución	34
3.1.2. Convolución en volúmenes	40
3.1.3. Pooling	41
3.1.4. Sobreajuste, <i>weight decay</i> y <i>dropout</i>	43
3.1.5. La función softmax	44
4. Segmentación Semántica: modelos de RN	47
4.1. Modelo Encoder-Decoder	48
4.1.1. Autoencoder neuronal	48
4.1.2. Autoencoder convolucional	49
4.1.3. Funciones de pérdida y activación	51
4.2. Modelos de redes neuronales para segmentación semántica	53
4.2.1. Modelo Encoder-Decoder en segmentación semántica	53
4.2.2. Modelo DeepLab3	54
4.2.3. Modelo SegNet	56
4.2.4. Modelo U-Net	57
5. Herramientas de desarrollo, tecnologías y metodología de trabajo	59
5.1. Herramientas de desarrollo	59
5.2. Tecnologías	60
5.2.1. Python	60

5.2.2. Matlab	61
6. Análisis de Resultados	63
6.1. Resultados obtenidos	63
6.1.1. DeepLab3	64
6.1.2. U-Net	70
6.1.3. SegNet	74
6.2. Análisis y comparación de resultados	78
6.2.1. Mejora de resultados	79
6.2.2. Resultado final	81
7. Conclusiones y Trabajo Futuro	83
7. Conclusions and Future Work	85
Bibliografía	87
A. Código y guía de instalación	93

Índice de figuras

1.1. Ejemplo de conducción automática. Imagen bajo licencia Creative Commons, Fuente: Gold (2023)	3
1.2. Ejemplo de tomografía cerebral. Imagen bajo licencia Creative Commons, Fuente: Asnaebsa (2022)	3
1.3. Ejemplo de tomografía cerebral. Imagen bajo licencia Creative Commons, Fuente: Campi (2016)	4
1.4. Ejemplo ilustrativo de segmentación semántica	4
1.1. Example of autonomous driving. Image under a Creative Commons license, Source: Gold (2023)	9
1.2. Example of brain tomography. Image under a Creative Commons license, Source: Asnaebsa (2022)	9
1.3. Example of crop analysis. Image under a Creative Commons license, Source: Campi (2016)	10
1.4. Illustrative example of semantic segmentation	10
2.1. Esquema visual de los paradigmas IA, ML y DL	15
2.2. Neurona con sus principales partes etiquetadas	16
2.3. Neurona con función de activación logística	17
2.4. Ejemplo de red neuronal sencilla de tres capas	18
2.5. Ejemplo de gráfico del precio de una vivienda por m^2	20
2.6. Ejemplo de gráfico del precio de una vivienda por m^2 con la función de hipótesis	20

2.7. Gráfico 3D de la función de gradiente descendente, Fuente: Andrew (2017)	21
2.8. Ejemplo de red neuronal	24
2.9. Ejemplo de red neuronal	24
2.10. Visualización de subajuste, generalización y sobreajuste	27
3.1. Ejemplo de imagen de gato de 64x64 píxeles. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)	31
3.2. Ejemplo de imagen de gato de 1000x1000 píxeles. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)	32
3.3. Visualización de red neuronal para 3000000 características.	33
3.4. Clasificación de una imagen. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)	33
3.5. Estructura de una RNC, Fuente: Aizel et al. (2021)	34
3.6. Funciones rectángulo	34
3.7. Funciones rectángulo generando un área común	35
3.8. Ejemplo de convolución	36
3.9. Ejemplo de convolución, Fuente: Pajares et al. (2021)	36
3.10. Ejemplo completo de convolución, Fuente: Pajares et al. (2021)	37
3.11. Ejemplo de padding, Fuente: Pajares et al. (2021)	38
3.12. Ejemplo de stride, Fuente: Pajares et al. (2021)	39
3.13. Ejemplo de stride, Fuente: Pajares et al. (2021)	39
3.14. Ejemplo de no realización de la convolución cuando hay desbordamiento	40
3.15. Ejemplo de convolución en volúmenes, Fuente: Andrew (2017)	40
3.16. Ejemplo de aplicación de varios filtros, Fuente: Pajares et al. (2021)	41
3.17. Ejemplo de pooling	42
3.18. Ejemplo de pooling	42
3.19. Cancelación de neuronas en base a un porcentaje	44
3.20. Ejemplo de dropout con pooling, Fuente: Pajares et al. (2021)	44

4.1.	Proceso completo de un autoencoder, Fuente: Pajares et al. (2021)	49
4.2.	Autoencoder convolucional:, Fuente: Pajares et al. (2021)	50
4.3.	Ejemplo de upsampling, Fuente: Pajares et al. (2021)	50
4.4.	Operación de Convolución traspuesta, Fuente: Pajares et al. (2021)	51
4.5.	Variación de error de las funciones de pérdida, Fuente: Pajares et al. (2021)	52
4.6.	Modelo encoder-decoder, Fuente: Pajares et al. (2021)	53
4.7.	Ejemplo convolución dilatada, Fuente: Pajares et al. (2021)	54
4.8.	Esquema de funcionamiento de DeepLab, Fuente: Pajares et al. (2021) . . .	55
4.9.	Propuesta de Chen y col., Fuente: Pajares et al. (2021)	56
4.10.	Esquema en paralelo con convoluciones <i>atrous</i> , Fuente: Pajares et al. (2021)	56
4.11.	Modelo SegNet, Fuente: Tsang (2019)	57
4.12.	Modelo U-Net, Fuente: Pajares et al. (2021)	57
5.1.	Interfaz de usuario de Matlab	61
6.1.	Proceso de entrenamiento de DeepLab 3 con resnet50	65
6.2.	Matriz de confusión normalizada de DeepLab 3 con resnet50	66
6.3.	Proceso de entrenamiento de DeepLab 3 con xception	67
6.4.	Matriz de confusión normalizada de DeepLab 3 con xception	68
6.5.	Proceso de entrenamiento de DeepLab 3 con inceptionresnetv2	69
6.6.	Matriz de confusión normalizada de DeepLab 3 con inceptionresnetv2 . . .	70
6.7.	Proceso de entrenamiento de U-Net con DeepStride 2	71
6.8.	Matriz de confusión normalizada de U-Net con DeepStride de 2	72
6.9.	Proceso de entrenamiento de U-Net con DeepStride 4	73
6.10.	Matriz de confusión normalizada de U-net con DeepStride de 4	74
6.11.	Proceso de entrenamiento de SegNet con VGG16	75
6.12.	Matriz de confusión normalizada de SegNet con VGG16	76
6.13.	Proceso de entrenamiento de SegNet con VGG19	77

6.14. Matriz de confusión normalizada de SegNet con VGG19	78
6.15. Matriz de confusión normalizada resultante del ajuste final	80
6.16. Ejemplo 1	81
6.17. Ejemplo 2	81
6.18. Ejemplo 3	82
A.1. Ruta de acceso a las imágenes	94
A.2. Ruta de acceso a las etiquetas	94
A.3. Elección de la red base	94
A.4. Creación de la red	94
A.5. Bloque donde se establecen los hiperparámetros	95
A.6. Bloque de entrenamiento del modelo	95

Índice de tablas

2.1.	Precio de una vivienda según sus m^2	19
2.2.	Asignación aproximada por conjunto de entrenamiento	27
2.3.	Asignación aproximada por conjunto sin validación	27
6.1.	Modelos utilizados con cada una de sus arquitecturas	63
6.2.	Datos de entrenamiento de DeepLab3 con resnet50	65
6.3.	Datos de entrenamiento de DeepLab3 con xception	67
6.4.	Datos de entrenamiento de DeepLab3 con inceptionresnetv2	69
6.5.	Datos de entrenamiento de U-Net con DeepStride de 2	71
6.6.	Datos de entrenamiento de U-Net con DeepStride de 2	73
6.7.	Datos de entrenamiento de SegNet con VGG16	75
6.8.	Datos de entrenamiento de SegNet con VGG19	77
6.9.	Porcentaje de precisión con cada método de optimización	79
6.10.	Porcentaje de precisión de cada prueba	79
6.11.	Valores de los hiperparámetros	80

Introducción

“The people who are crazy enough to think they can change the world are the ones who do”
— Steve Jobs

¿Pueden pensar las máquinas? (Turing, 1950) Esta idea de que las máquinas sean capaces de pensar surge durante la Segunda Guerra Mundial, momento en el que Alan Turing desarrolló una máquina que era capaz de descifrar los mensajes en clave escritos por los alemanes (Scotto, 2018). Se podría decir que este suceso supuso el comienzo de lo que hoy conocemos como Inteligencia Artificial (IA), idea que llevaba ya varios años introduciéndose en la cultura popular de la época. Un ejemplo notable es “El mago de Oz” de L. Frank Baum, donde un hombre de hojalata anhela una mente y un corazón, lo que nos sugiere la dotación de emociones y conciencia a las máquinas (Anyoha, 2017).

Sin embargo, esta idea no quedó solamente en la mente de los literarios y los artistas, sino que trascendió a los matemáticos y científicos de la época. En 1943, McCulloch y Pitts (1943) presentaron un modelo matemático de neuronas artificiales tratando de aproximarse al comportamiento del cerebro humano. Este modelo, que contaba con una red de neuronas que se activaban o desactivaban en función de la presencia o ausencia de estímulos, se considera el primer intento en el campo de la IA, pese a que no se hablaba de IA todavía en aquella época (Withey y Koles, 2008).

La IA desde entonces ha ido evolucionando hasta convertirse en una herramienta muy utilizada actualmente en muchos campos como la banca, el marketing, el entretenimiento, la agricultura y por supuesto la robótica donde la percepción computacional juega un papel importante. Así, encontramos herramientas que nos permiten detectar cuántas personas aparecen en una imagen, preguntar algo a un asistente virtual y obtener una respuesta coherente, hacer una recomendación de una película de interés o calcular la ruta más rápida a un determinado lugar. Como se puede ver, existen multitud de aplicaciones interesantes en los que la IA juega un papel fundamental. Este trabajo se centra precisamente en el campo de la Visión por Computador (VC) o Visión Artificial (VA) cuyo objetivo principal consiste en el reconocimiento del entorno.

1.1. Motivación

La IA ha cambiado de forma radical nuestra forma de vivir, aprender, transportarnos, trabajar e incluso de relacionarnos con los demás. Permite realizar tareas más rápido y de forma más eficiente, aumentando nuestra productividad; mejorar las recomendaciones obtenidas en diferentes plataformas; diagnosticar enfermedades mucho antes; descubrir nuevos tratamientos; etc. En general, es una potente ayuda para vivir mejor, y es aquí donde radica su importancia, en el potencial que posee para mejorar nuestras vidas.

A veces explicar el concepto de la IA a un público general se convierte en una ardua tarea por lo abstracto y complicado que resulta simplificar algo tan complejo. Para mucha gente esta tecnología podría asemejarse a la “magia”, pero nada más lejos de la realidad. En cada una de las recomendaciones que hace Google cuando se realiza una búsqueda, o en cada película mostrada por Netflix en la sección “Para ti”, hay detrás todo un equipo de ingenieros, informáticos, matemáticos y físicos que han estado años diseñando, ajustando y entrenando un modelo de RN que permite obtener estos resultados.

Como se ha mencionado anteriormente, la IA no deja de evolucionar, y con esta evolución surgen nuevas propuestas de uso y aplicación. Una de estas nuevas propuestas es la VC. Los primeros experimentos en VC comenzaron en 1959 (Marr, 1982), con un grupo de neurofisiólogos que analizaron cómo un gato se comportaba ante una serie de imágenes. Observaron que este respondía antes a las formas simples como los bordes o las líneas, lo que significaba que el procesamiento de imágenes debía empezar por aquí. No obstante, desde esos experimentos de 1959 la VC ha ido evolucionado notablemente, de forma que en la actualidad permite realizar cosas como identificar una imagen perteneciente a una clase determinada, detectar objetos o personas presentes en la imagen, agrupar todos los elementos de la imagen en una clase, y mucho más. Las aplicaciones y formas de utilizar la VC son infinitas. No obstante, este trabajo se centra en lo que se conoce como segmentación semántica, uno de los campos de interés en la VC. La segmentación semántica permite analizar una imagen a nivel de píxel, generando zonas o regiones agrupadas de píxeles que se identifican con una determinada clase, previamente identificada. Por ejemplo, en una escena de tráfico urbano, dichas agrupaciones se corresponden con calzadas, vehículos, peatones, aceras, árboles y cielo, que constituyen las clases de interés (IBM, 2023b).

A continuación se muestran algunas de las aplicaciones más interesantes de la segmentación semántica, ilustrándolas mediante imágenes distribuidas bajo licencia Creative Commons:

- Conducción automática: uno de los casos más difundidos últimamente y en el que la segmentación semántica funciona muy bien es su aplicación a la conducción automática. La cámara instalada en un vehículo permite distinguir las distintas zonas de la calle identificando los diferentes elementos presentes, a la vez que los agrupa en diferentes clases, como se ha mencionado previamente. En el ejemplo de la Figura 1.1, se observa cómo se ha identificado la carretera pintada de ● rojo, los demás coches se han coloreado de ● azul y por último los peatones se han pintado de color ● verde. De esta forma se puede identificar exactamente dónde se encuentra cada objeto para así tomar decisiones.



Figura 1.1: Ejemplo de conducción automática. Imagen bajo licencia Creative Commons, Fuente: Gold (2023)

- Medicina: en medicina la segmentación semántica se utiliza para multitud de aplicaciones. Las técnicas como la Tomografía Computarizada (TC) y la Resonancia Magnética (RM) utilizan la segmentación semántica para reconocer y examinar regiones que sean de interés, lo que les permite a los médicos identificar y diagnosticar enfermedades de forma más rápida y efectiva. Como se puede observar en la Figura 1.2 se ve cómo la radiografía ha sido clasificada mostrando las diferentes zonas que se marcan en distintos colores.

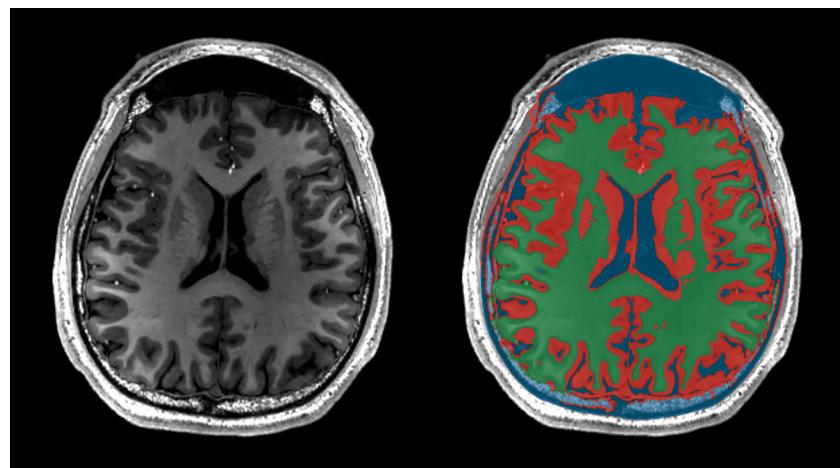


Figura 1.2: Ejemplo de tomografía cerebral. Imagen bajo licencia Creative Commons, Fuente: Asnaebsa (2022)

- Cosechas: la segmentación semántica ha revolucionado la agricultura permitiendo identificar las cosechas, plagas y suelos, y de esta forma prevenir las enfermedades y la pérdida de los cultivos. Al igual que con la anterior figura, en la Figura 1.3 se puede ver un campo de cultivo donde se marcan en diferentes colores las diferentes regiones de la imagen.

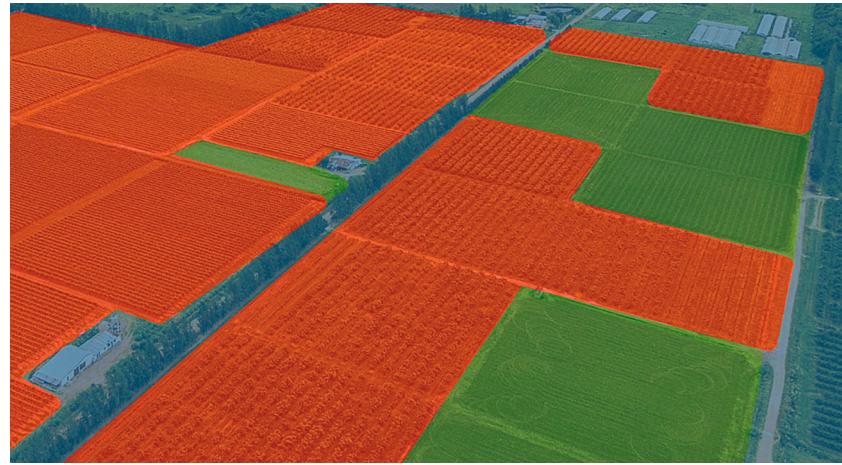


Figura 1.3: Ejemplo de tomografía cerebral. Imagen bajo licencia Creative Commons, Fuente: Campi (2016)

Es por ello por lo que el principal interés por este trabajo reside en conseguir entender de forma teórica y práctica los conceptos fundamentales bajo el paradigma de Deep Learning (DL), esto es Aprendizaje Automático (AP) (Pajares et al., 2021), aplicado a la VC/VA y profundizar en la segmentación semántica, analizando varios de los modelos de Redes Neuronales Convolucionales (RNC) más utilizados para tal fin, comparándolos entre sí para determinar cuál de ellos proporciona mejores resultados, para finalmente ajustar dicho modelo con el fin de conseguir posibles mejoras. En el presente trabajo se utilizan imágenes agrícolas para identificar dos tipos de estructuras a nivel de píxel: suelo y cultivo.

En la Figura 1.4 se puede ver un ejemplo de entrenamiento, de los 828 ejemplos que forman el conjunto de datos, formado por la imagen y sus etiquetas.



Figura 1.4: Ejemplo ilustrativo de segmentación semántica

1.2. Objetivos

El objetivo de este trabajo consiste en estudiar los conceptos teóricos relacionados con la segmentación semántica en imágenes dentro del paradigma de DL, y comparar distintos modelos de segmentación semántica utilizando un dataset de agricultura para determinar qué modelo ofrece mejores resultados. No se pretende el desarrollo de un modelo específico sino el análisis de los resultados que ofrecen los modelos bajo estudio. Para lograr esto se utilizarán distintas herramientas que se explicarán en el capítulo cuatro de este documento.

El resultado del proyecto consiste en un repositorio online en el que se dispondrá de todo el código desarrollado junto con la memoria. Para la consecución del objetivo general se plantean los subobjetivos que se indican a continuación, bajo los dos aspectos que se mencionan de investigación y desarrollo.

Investigación

Antes de comenzar a hacer ningún desarrollo es importante conocer la tecnología con la que se va a trabajar, así como la metodología seguida, que se concreta en los siguientes puntos:

- Estudio teórico-práctico: consistente en la realización de un curso sobre DL en la plataforma Coursera que permite obtener las bases para poder comenzar. En este curso se explica la teoría sobre VC y las bases para poder trabajar con segmentación semántica. Una vez obtenidas las bases se realizó la búsqueda de información para aprender los conceptos fundamentales de la segmentación semántica y finalmente se investigó sobre sus posibles aplicaciones.
- Elección de la tecnología: una de las cosas más importantes a la hora de comenzar un nuevo proyecto es decidir las tecnologías con las que se van a trabajar. En este caso la tecnología/herramienta elegida fue Matlab (2023). Además, Matlab está disponible bajo licencia para los usuarios de la Universidad Complutense de Madrid (UCM) y posee gran documentación y multitud de ejemplos ya hechos que permiten avanzar muy rápido y obtener resultados mucho antes.
- Elección del tema: a la hora de elegir el tema de la aplicación se estudiaron diversas posibilidades. Se comenzó con un dataset de conducción, pero el gran tamaño de las clases y número de imágenes hizo que finalmente se optase por un dataset mucho más sencillo con menos imágenes y únicamente dos clases, que permitían avanzar mucho más rápido.
- Elección del hardware: el problema cuando se quiere desarrollar una aplicación de IA es la gran exigencia en poder computacional que se requiere a la hora de realizar el entrenamiento. Por las limitaciones de los recursos hardware propio se analizó la posibilidad de contratar un *cluster* de forma que ofreciera recursos suficientes para poder hacer viable el proyecto, pero finalmente, dado el dataset elegido se optó por el computador de sobremesa propio.

Desarrollo

Una vez definida la forma de trabajar se comenzó con el desarrollo, para lo cual se optó por lo siguiente, siempre bajo la guía de la metodología de trabajo:

- Elección de modelos RNC: Lo primero fue buscar modelos de RN de interés en segmentación semántica. Al haber decidido trabajar con Matlab, esta tarea fue breve en el tiempo gracias a la cantidad de documentación y ejemplos disponibles. Finalmente se eligieron tres modelos de redes de este tipo, que son las que han permitido realizar las comparaciones de desempeño correspondientes.
- Adaptación de los ejemplos: al ya tener un ejemplo de ejecución para cada una, el siguiente paso consistió en preparar el entorno y adaptar el código a los ejemplos disponibles.
- Entrenamiento de las redes: finalmente se comenzó con el entrenamiento guardando todos los resultados de ejecución y eficiencia de cada una de ellas.
- Estudio final: una vez realizado todos los entrenamientos se compararon los resultados obtenidos para determinar qué modelo obtenía mejores resultados. De esta forma fue posible seleccionar los mejores y comenzar a realizar ajustes para ver si se conseguía mejorar los resultados.
- Realización de la memoria: en la que se refleja todo el proceso, se exponen los conceptos y se obtienen los resultados pertinentes, a la vez que se analizan.

1.3. Organización de la memoria

La memoria se ha organizado en los siguientes capítulos:

1. Introducción: Explicación genérica sobre lo que se va a tratar en este documento junto con la motivación y objetivos.
2. Aprendizaje Automático: bases del Aprendizaje Profundo: Explicación teórica sobre los conceptos del DL y la VC.
3. Redes Neuronales Convolucionales: Explicación teórica sobre los conceptos fundamentales de las RNC.
4. Segmentación Semántica: Modelos de Redes Neuronales: Explicación teórica sobre la segmentación semántica y análisis de los modelos utilizados.
5. Herramientas de desarrollo, tecnologías y metodología de trabajo: Explicación de las herramientas que se utilizan a la hora de realizar este proyecto.
6. Análisis de resultados: Explicación de los resultados obtenidos, comparación de los modelos entre sí y posibles mejoras.
7. Conclusión y trabajo a futuro: Análisis de lo aprendido y posibles evoluciones a futuro.

Chapter 1

Introduction

“The people who are crazy enough to think they can change the world are the ones who do”
— Steve Jobs

Can machines think? (Turing, 1950) The idea that machines can think emerged during World War II when Alan Turing developed a machine capable of deciphering encrypted messages sent by the Germans (Scotto, 2018). It could be said that this event marked the beginning of what we now know as Artificial Intelligence (AI), an idea that had been gaining ground in popular culture for several years. A notable example is L. Frank Baum’s “The Wizard of Oz” where a tin man longs for a mind and a heart, suggesting the endowment of emotions and consciousness of machines (Anyoha, 2017).

However, this idea was not confined to the minds of writers and artists but also extended to mathematicians and scientists of the time. In 1943, McCulloch y Pitts (1943) presented a mathematical model of artificial neurons, attempting to approximate the behavior of the human brain. This model, which consisted of a network of neurons that activated or deactivated based on the presence or absence of stimuli, is considered the first attempt in the field of Artificial Intelligence (AI), even though the term AI was not yet used at that time (Withey y Koles, 2008).

AI has since evolved to become a widely used tool in various fields such as banking, marketing, entertainment, agriculture, and, of course, robotics, where computer vision plays a significant role. We find tools that allow us to detect how many people appear in an image, ask a virtual assistant a question and receive a coherent response, receive movie recommendations of interest, or calculate the fastest route to a specific location. As can be seen, there are numerous interesting applications where AI plays a fundamental role. This work focuses precisely on the field of Computer Vision (CV), whose main goal is environmental recognition.

1.1. Motivation

AI has radically changed the way we live, learn, commute, work, and even interact with others. It enables us to perform tasks faster and more efficiently, increasing our productivity; improve recommendations on various platforms; diagnose diseases much earlier; discover new treatments; and more. In general, it is a powerful aid for a better life, and this is where its importance lies, in the potential it has to improve our lives.

Explaining the concept of AI to a general audience can sometimes be a daunting task due to how abstract and complex it can be. For many people, this technology might seem like “magic,” but far from it. Behind every recommendation that Google makes when you perform a search or every movie shown by Netflix in the “For You” section, there is a team of engineers, computer scientists, mathematicians, and physicists who have spent years designing, fine-tuning and training a Neural Network (NN) model that delivers these results.

As mentioned earlier, AI continues to evolve, and with this evolution come new proposals for use and application. One of these new proposals is Computer Vision. The first experiments in Computer Vision (CV) began in 1959 (Marr, 1982), with a group of neurophysiologists who analyzed how a cat reacted to a series of images. They observed that the cat responded more to simple shapes like edges or lines, which meant that image processing should start here. However, since those experiments in 1959, CV has evolved significantly, allowing tasks such as identifying an image belonging to a specific class, detecting objects or people in the image, grouping all elements of the image into a class, and much more. The applications and uses of CV are limitless. Nevertheless, this work focuses on what is known as semantic segmentation, one of the areas of interest in CV. Semantic segmentation allows the analysis of an image at pixel level, generating regions or groups of pixels identified with a particular pre-defined class. For example, in an urban traffic scene, these groupings correspond to roads, vehicles, pedestrians, sidewalks, trees, and the sky, which are the classes of interest (IBM, 2023b).

Here are some of the most interesting applications of semantic segmentation, illustrated with images distributed under a Creative Commons license:

- Autonomous Driving: One of the most widely discussed applications recently is its use in autonomous driving. The camera installed in a vehicle allows distinguishing different areas of the road by identifying different elements present while categorizing them into different classes, as mentioned earlier. In the example in Figure 1.1, can be seen how the road painted in ● red has been identified, other cars are colored ● blue, and pedestrians are painted ● green. This allows for precise identification of the location of each object to make decisions accordingly.



Figure 1.1: Example of autonomous driving. Image under a Creative Commons license, Source: Gold (2023)

- Medicine: In medicine, semantic segmentation is used for a multitude of applications. Techniques like Computerized Tomography (CT) and Magnetic Resonance Imaging (MRI) use semantic segmentation to recognize and examine regions of interest, allowing doctors to identify and diagnose diseases more quickly and effectively. As seen in Figure 1.2, the X-ray image has been classified, showing different marked areas in different colors.

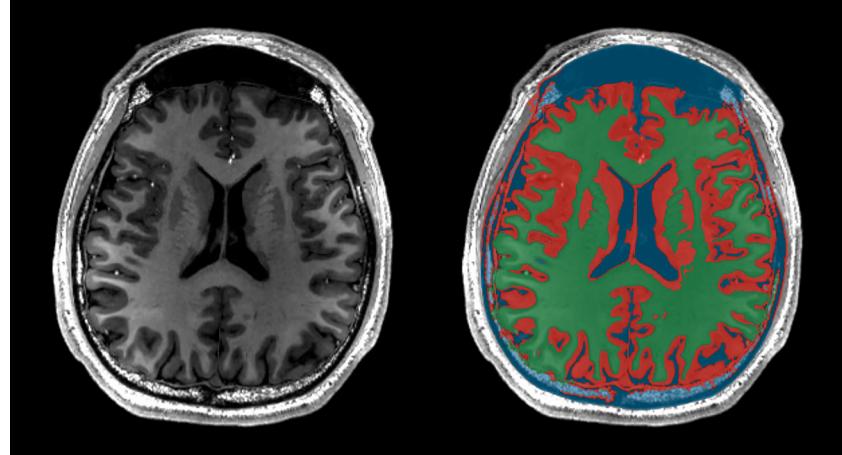


Figure 1.2: Example of brain tomography. Image under a Creative Commons license, Source: Asnaebsa (2022)

- Agriculture: Semantic segmentation has revolutionized agriculture by allowing the identification of crops, pests, and soils to prevent diseases and crop loss. Similar to the previous figure, in Figure 1.3, can be seen a crop field where different regions of the image are marked in different colors.

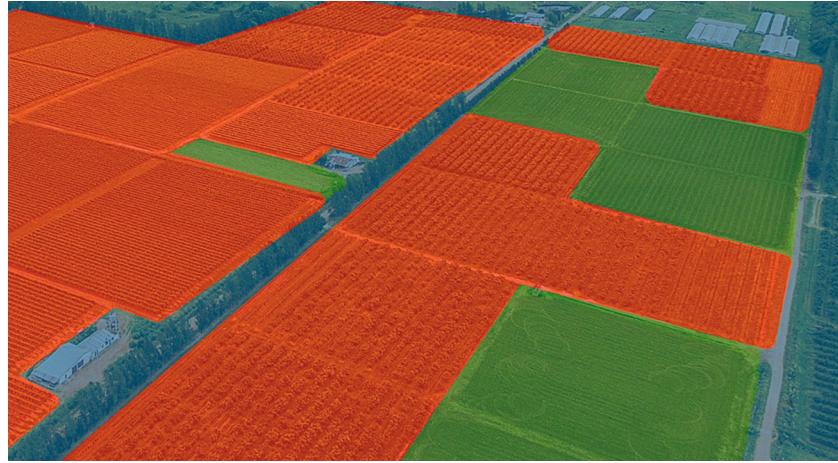


Figure 1.3: Example of crop analysis. Image under a Creative Commons license, Source: Campi (2016)

This is why the main interest in this work lies in gaining a theoretical and practical understanding of the fundamental concepts under the Deep Learning (DL) paradigm, which is Machine Learning (ML) (Pajares et al., 2021) applied to CV, and go deeper into semantic segmentation by analyzing several of the most commonly used Convolutional Neural Network (CNN) models for this purpose, comparing them to determine which one provides better results, and finally adjusting the chosen model to achieve possible improvements. This work uses agricultural images to identify two types of pixel-level structures: soil and crops.

In Figure 1.4, can be seen a training example out of the 828 examples that make up the dataset, consisting of an image and its labels.



Figure 1.4: Illustrative example of semantic segmentation

1.2. Objectives

The objective of this work is to study the theoretical concepts related to semantic segmentation in images within the DL paradigm and compare different semantic segmentation models using an agriculture dataset to determine which model offers better results. The goal is not to develop a specific model but to analyze the results provided by the models under study. Various tools used in this project will be explained in Chapter Four of this document.

The result of the project consists of an online repository containing all the developed code along with the thesis. To achieve the general objective, the following sub-objectives are outlined under the two aspects of research and development.

Research

Before starting any development, it is important to understand the technology to be used, as well as the methodology to be followed, which is specified in the following points:

- Theoretical-Practical Study: Involves taking a course on DL on the Coursera platform to obtain the basics to start. This course explains the theory of CV and the basics for working with semantic segmentation. Once the basics were obtained, a search for information was conducted to learn the fundamental concepts of semantic segmentation and explore its potential applications.
- Technology Selection: One of the most important aspects when starting a new project is deciding which technologies to work with. In this case, the chosen technology/tool was Matlab (2023). Additionally, Matlab is available under license for UCM users and has extensive documentation and numerous pre-made examples, allowing for rapid progress and earlier results.
- Topic Selection: When choosing the application topic, various possibilities were considered. It started with a driving dataset, but the large class size and number of images led to the selection of a much simpler dataset with fewer images and only two classes, which allowed for faster progress.
- Hardware Selection: The challenge when developing an AI application is the high computational power required for training. Due to limitations in personal hardware resources, the possibility of renting a cluster with sufficient resources to make the project viable was explored. However, due to the selected dataset, the decision was made to use the personal desktop computer resources.

Development

Once the work approach was defined, development began, for which the following steps were taken, always following the work methodology:

- Selection of CNN Models: The first step was to search for CNN models of interest in semantic segmentation. Since Matlab was chosen as the development environment, this task was relatively quick thanks to the extensive documentation and examples available. Three models of such networks were finally chosen, which allowed for performance comparisons.
- Adaptation of Examples: Having an execution example for each model, the next step was to prepare the environment and adapt the code to the available examples.
- Network Training: Finally, training of the networks commenced, saving all execution results and efficiency for each of them.
- Final Analysis: After completing all the training, the results obtained were compared to determine which model achieved better results. This allowed for the selection of the best-performing model and further adjustments to see if improvements could be made.
- Report Writing: This phase involved documenting the entire process, explaining the concepts, and presenting the relevant results for analysis.

1.3. Thesis Organization

The thesis is organized into the following chapters:

1. Introduction: Generic explanation of what will be covered in this document along with motivation and objectives.
2. Machine Learning Basics: Introduction to the basics of Deep Learning, explaining the concepts of DL and CV.
3. Convolutional Neural Networks: Theoretical explanation of the fundamental concepts of CNNs.
4. Semantic Segmentation: Neural Network Models: Theoretical explanation of semantic segmentation and analysis of the models used.
5. Development Tools, Technologies, and Work Methodology: Explanation of the tools used in this project along with the methodology followed.
6. Results Analysis: Explanation of the obtained results, comparison of models, and potential improvements.
7. Conclusion and Future Work: Analysis of what has been learned and possible future developments.

Capítulo 2

Aprendizaje Automático: bases del Aprendizaje Profundo

Cuando se resuelven problemas, muchas veces uno se centra más en la solución que en el camino. ¿Cómo puedo resolver este problema? ¿Cuál sería la mejor herramienta para poder resolverlo? ¿Existen tecnologías que me permitan hacer lo que quiero resolver? Estas preguntas a la hora de desarrollar una aplicación compleja son muy importantes ya que pueden suponer una diferencia significativa en la calidad del resultado final obtenido. Es importante saber cómo afrontar un problema y cuáles son las mejores herramientas para ello.

Realizar un proyecto de estas características no es tarea fácil y se necesitan herramientas potentes como el DL para poder conseguirlo. A la hora de identificar el problema se podría decir algo como: “Necesito dividir las fotos de mi galería entre suelo y cultivo”. Y para lograr la mejor solución posible se trata de encontrar la herramienta correcta.

Una persona en cuestión de segundos es capaz de dibujar en la imagen la división perfecta de lo que es suelo y de lo que es cultivo, pero parece que lograr encontrar un algoritmo perfecto que sea capaz de identificar esto en primera instancia resultaría muy complejo e incluso en algunos casos imposible. Es en este tipo de tareas donde el DL tiene todo su potencial, ya que se le puede enseñar a identificar de forma precisa qué partes de una imagen son suelo y cuales cultivo, casi al igual que lo haría una persona.

En este capítulo se explican los fundamentos del DL, posteriormente se describen aspectos relativos sobre la VC, para finalmente pasar a explicar en detalle la segmentación semántica, uno de los campos de interés dentro de DL. En lo que sigue tanto para el presente capítulo como el siguiente, se utiliza como referencia base Andrew (2011), junto con las referencias asociadas para cada uno de los conceptos que se exponen, tal como Pajares et al. (2021).

2.1. Inteligencia Artificial

El término IA está muy arraigado en la cultura popular de los dos últimos siglos. Se encuentran referencias a la misma en películas, comics, libros, series, etc. Aunque esta idea de las máquinas inteligentes lleva acuñado a la ciencia ficción desde hace siglos no fue hasta 1956 donde se comenzó a hablar formalmente de IA, año en el que se acuñó el término durante la conferencia “Dartmouth Summer Research Project on Artificial Intelligence” de John McCarthy. Fue en esta conferencia donde se anunciaron los objetivos y la visión que se tenía con la IA. Pocos años más tarde, en 1959, Arthur Samuel acuñó el término Machine Learning (ML) durante su trabajo como ingeniero en IBM (Scientest, 2022).

Durante esta época la gran expectativa que había por la IA generó mucho interés, interés que se fue disolviendo en los siguientes años hasta la década de los 90 debido a varios factores como los resultados tan malos obtenidos, las pérdidas de financiación que se produjeron y la baja potencia de los ordenadores de la época que impedía realizar cálculos tan complejos como se requieren en IA, y en especial en DL. Aunque a partir de los 90 del siglo pasado comenzó a resurgir el interés por la IA, no fue hasta comienzos de la década pasada cuando el creciente interés por parte de los investigadores y de las empresas ocasionó un aumento dramático en la financiación y la inversión (of Data Science, 2021).

Se podría decir que la IA es la inteligencia llevada a cabo por las máquinas. Hoy en día se encuentra la IA en prácticamente cualquier área de la vida, implementada de muchas formas diferentes y utilizando diferentes tecnologías. Una de las áreas más populares actualmente es el ML, o la versión más potente de esta, el DL, concepto que se va a estudiar en la próxima sección.

2.2. Deep Learning

Cuando se habla de ML una de las primeras cosas en las que se piensa es en las RN, y aunque son su principal implementación, el ML no deja de ser un conjunto de algoritmos que construyen modelos basados en unos datos de muestra que permiten realizar predicciones sobre otros datos nunca antes vistos por este. Algunas de las implementaciones del ML son la regresión lineal, las máquinas de Vectores de Soporte (VS), bosques aleatorios y, por supuesto las RN (IBM, 2023a).

Como ya se ha comentado anteriormente, el DL es un subconjunto del ML, una evolución sofisticada y matemáticamente compleja de los algoritmos de aprendizaje automático. Son algoritmos que intentan simular la forma en la que un ser humano sacaría conclusiones, y que se asemejan mucho a la estructura del cerebro humano. En la Figura 2.1 se puede ver un esquema visual del lugar que ocupa cada uno de estos conceptos introducidos.

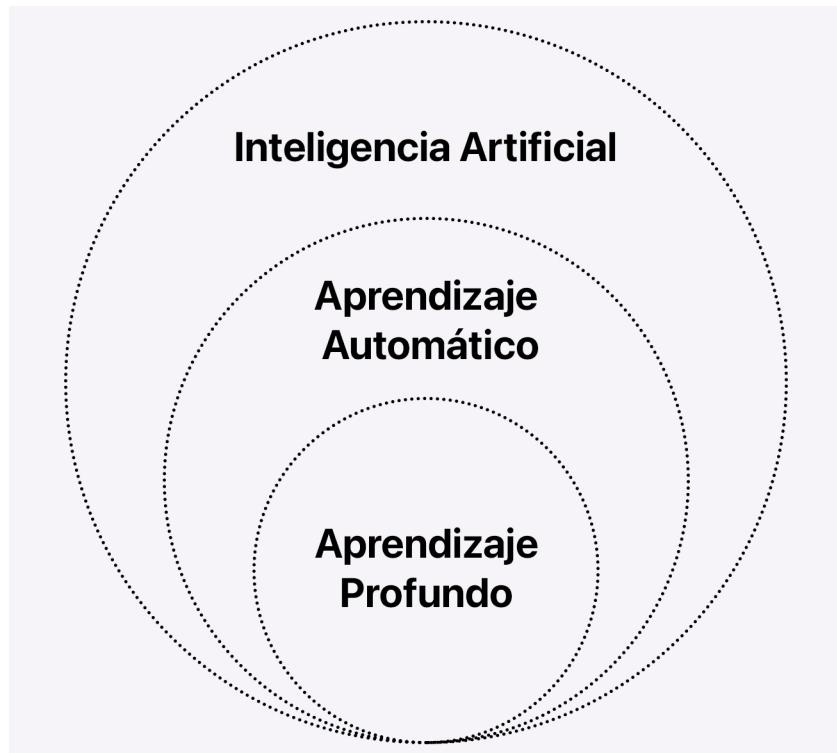


Figura 2.1: Esquema visual de los paradigmas IA, ML y DL

Por dejar los conceptos claros se podría decir que:

- IA: teoría y desarrollo de sistemas informáticos capaces de realizar tareas que normalmente requieren inteligencia humana.
- ML: desarrollo de algoritmos que otorgan a un ordenador “la capacidad de aprender sin estar explícitamente programados para ello”.
- DL: algoritmos de aprendizaje automático más complejos con estructura lógica similar a un cerebro humano mediante RN artificiales u otras estructuras complejas.

Como puede deducirse fácilmente, una de las estructuras claves son las RNs. Se plantea así la pregunta ¿qué es una RN?, cuya respuesta se concreta en lo que sigue.

2.2.1. Redes neuronales biológica

En biología se podría definir una neurona como “el tipo de célula que recibe y envía mensajes entre el cuerpo y el encéfalo” (NIH, 2023). Es la unidad más básica que forma el sistema nervioso. En conjunto son las encargadas de transmitir información eléctrica y química para que se pueda caminar, pensar, interactuar y sentir (Ruiz, 2021). Su estructura es similar a la de una estrella y su función básica es la de recibir y transmitir información en forma de impulsos eléctricos a través de una compleja red de neuronas que constituyen el sistema nervioso (Montagud, 2020). Por ejemplo, si queremos mover un brazo, las neuronas

encargadas de esto mandarán un impulso eléctrico a los músculos para poder hacerlo; o si algún sensor como los ojos quiere enviar información a nuestro cerebro de lo que está viendo, se mandan pulsos eléctricos a las neuronas para que podamos visualizarlo.

Las neuronas tienen tres partes bien diferenciadas (Ruiz, 2021):

1. Soma: Es el cuerpo de la neurona y el lugar donde se encuentra el núcleo. Es desde aquí de donde salen las otras partes de la neurona, las dendritas y el axón.
2. Dendritas: Son extensiones de la neurona que permiten el intercambio de información para que la pueda procesar. Se podría decir que estos son los cables de entrada de la neurona, los que reciben la información que posteriormente procesa el núcleo.
3. Axón: Consta de una única prolongación que lleva la información desde la neurona hacia otra. Se podría decir que estos son los cables de salida, encargados de conectar unas neuronas con otras. Transmiten la información ya transformada por el núcleo.

En la Figura 2.2 se pueden ver cada una de estas partes.

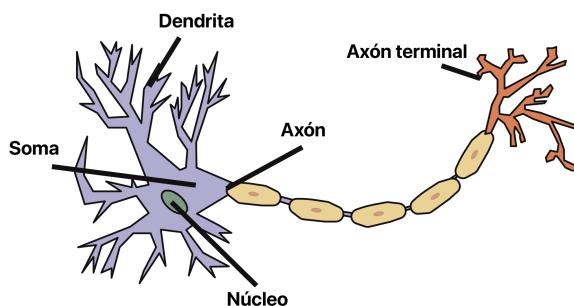


Figura 2.2: Neurona con sus principales partes etiquetadas

Cuando una neurona recibe la información a través de las dendritas, esta se modifica incrementando, disminuyendo o regulando la producción de un potencial de acción dependiendo si la señal es excitatoria, inhibidora o moduladora. Una vez terminado, se transmite una señal a través del axón para que llegue a otras neuronas y de esta forma alcanzar al sistema nervioso central. La comunicación entre dos neuronas se produce transformando este potencial de acción en una señal química a través de un pequeño espacio denominado sinapsis (Facts, 2018).

Como se explica a continuación, las RN artificiales no están muy alejados de este concepto y se puede ver claramente la amplia inspiración que ha tenido este proceso a la hora de diseñarlas.

2.2.2. Redes neuronales artificiales

Las RN artificiales, o simplemente RN, son un modelo que pretenden imitar el funcionamiento del cerebro humano. En estas redes en lugar de neuronas se encuentran nodos, que se conectan entre sí para trasmitir información de uno a otro.

Cada uno de estos nodos representan un modelo muy simplista de lo que hace una neurona. En la Figura 2.3 se muestra la imagen de una neurona con una función de activación logística.

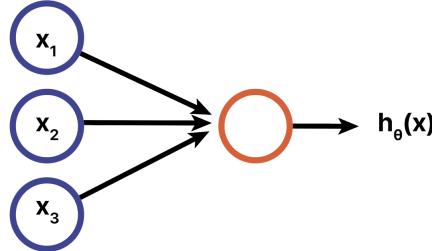


Figura 2.3: Neurona con función de activación logística

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \quad (2.1)$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}} \quad g(z) = \frac{1}{1 + e^{-z}} \quad (2.2)$$

Como se puede ver, la similitud que hay entre el modelo natural y el artificial es bastante claro. Existen conexiones de entrada que podrían ser las dendritas. Estas conexiones de entrada enlazan cada neurona con un número de neuronas de la capa anterior. Cada una de las entradas (x) tiene asociado otro parámetro llamado peso o parámetro (θ) que indica la importancia que ese valor tiene en esa parte de la red. Todos estos valores se suman multiplicados con sus correspondientes pesos y pasan por una función de activación ($g(z)$), enviando este resultado por la salida, que sería el equivalente al axón.

Por simplificar, se podría decir que una RN es simplemente un modelo de ecuaciones matemáticas, donde se toma una variable de entrada y , tras realizar una serie de operaciones, da como resultado una o más variables de salida (Shin, 2020).

Las RN están formadas por varias capas. La RN más simple es la que está formada por tres capas, una capa de entrada, una capa oculta y una capa de salida. No obstante, como se verá más adelante, en DL los modelos de redes constan de multitud de capas ocultas que se encargarán de realizar todo el aprendizaje de la red.

- Capa de entrada: la capa de entrada consta de una o más variables o características.
- Capas ocultas: son las capas intermedias de la red. Realizan operaciones que permiten transformar los datos de entrada de la red para formar una salida.
- Capa de salida: devuelve el resultado de aplicar todas las operaciones a través de la red. Normalmente devuelve el resultado como porcentaje de acierto y puede tener

tantas neuronas como clases tenga la RN. Cada resultado indicará el porcentaje de acierto de que el *input* introducido pertenezca a dicha clase.

En la Figura 2.4 se observa un ejemplo de una RN formada por las tres capas mencionadas, y a continuación las ecuaciones que determinan su funcionamiento.

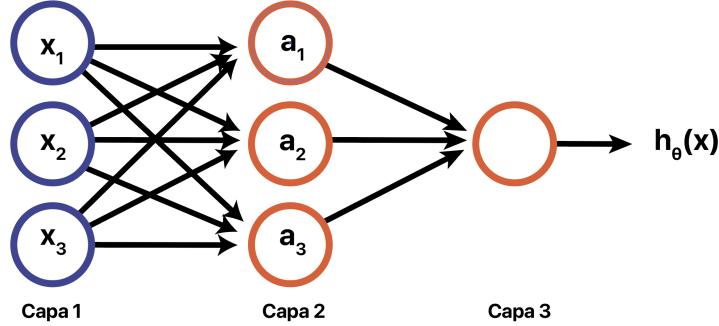


Figura 2.4: Ejemplo de red neuronal sencilla de tres capas

$$a_i^{(j)} = \text{activación de la unidad } i \text{ en la capa } j \quad (2.3)$$

$$\Theta^{(j)} = \text{matriz de pesos de la capa } j \text{ a la capa } j + 1 \quad (2.4)$$

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \end{aligned} \quad (2.5)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Se puede decir por lo tanto que si la RN tiene s_j unidades en la capa j , s_{j+1} unidades en la capa $j+1$, entonces $\Theta^{(j)}$ tendrá las dimensiones de $s_{j+1} \times (s_j + 1)$. A este proceso de ir hacia adelante calculando los resultados de cada capa de la RN hasta obtener la salida se conoce como propagación hacia delante.

2.3. Aprendizaje de las RN

Ya se ha visto lo que son las neuronas artificiales, su similitud con las neuronas del cerebro y cómo se conectan entre sí para formar la arquitectura de la RN artificial. Se han determinado también los pasos que realiza una RN para pasar de unos datos de entrada a una salida, pero queda por ver lo más importante que es cómo son capaces de aprender estas redes.

Por lo general, cuando se habla de aprendizaje en RN se puede hacer referencia a dos tipos de aprendizaje (Delua, 2021):

- Aprendizaje supervisado: este tipo de aprendizaje se define por utilizar datos etiquetados. Para cada uno de los ejemplos de entrenamiento se dispone de su correspondiente etiqueta. Este tipo de aprendizaje sigue la idea de que los datos de entrada y los de salida están relacionados. De esta forma, cuando el tipo de aprendizaje que se requiera sea supervisado se hablará de problemas de regresión o de clasificación. Un ejemplo de uso práctico sería la predicción del precio de una vivienda según variables tales como el número de habitaciones, el tamaño, etc.
- Aprendizaje no supervisado: se utiliza cuando no se tiene mucha idea de cómo se deberían mostrar los resultados, ya que no se dispone de etiquetas que permitan clasificar los ejemplos de entrenamiento disponibles. Consiste en analizar y agrupar conjuntos de datos que tengan características similares y de esta forma descubrir grupos y patrones ocultos. Es muy utilizado en problemas de agrupación en clústeres.

Aunque es interesante conocer ambos tipos de aprendizajes el que se utiliza en este trabajo es el supervisado por lo que a partir de ahora se omitirá la especificación supervisado cuando se hable de aprendizaje.

2.3.1. Función de coste

Para hacer que una RN aprenda es necesario entrenarla. Anteriormente se ha hablado de los pesos de la RN que son unos valores que se van multiplicando capa por capa hasta dar como resultado un resultado final y obtener la clasificación. El entrenamiento consistirá en ajustar estos pesos para conseguir obtener unos resultados aceptables que sirvan para predecir con certeza un nuevo valor en el futuro. Para ello se va a utilizar el concepto de función de pérdida o *Loss Function* en inglés. A continuación, se muestra un ejemplo en regresión lineal para ilustrar intuitivamente el concepto

Se plantea un conjunto de datos sobre el tamaño en m^2 de una casa y su precio en euros que se puede visualizar en la tabla 2.1.

Tamaño en m^2 (x)	Precio en miles de euros (y)
2104	460
1216	232
1534	315
853	178
...	...

Tabla 2.1: Precio de una vivienda según sus m^2

Si se representa esta tabla en un gráfico se obtendría la representación que se muestra en la Figura 2.6.

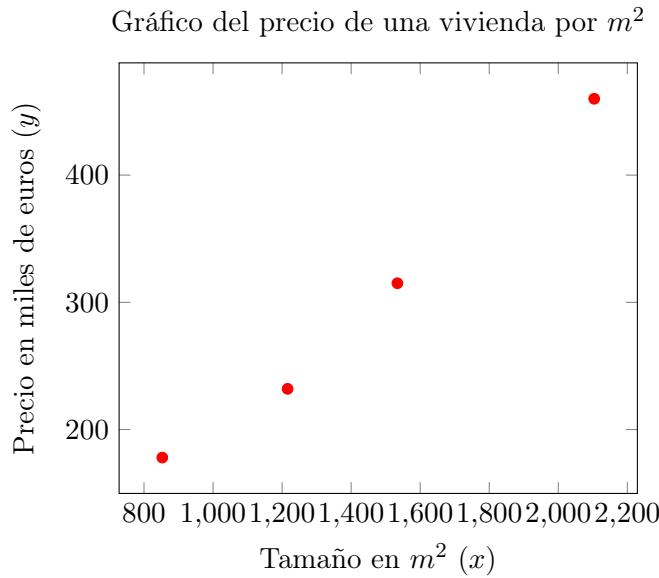


Figura 2.5: Ejemplo de gráfico del precio de una vivienda por m^2

Observando la representación de la figura 2.1 se puede ver que la gráfica que mejor representaría este conjunto de datos sería una función lineal que pase por medio de la mayor cantidad de puntos posibles.

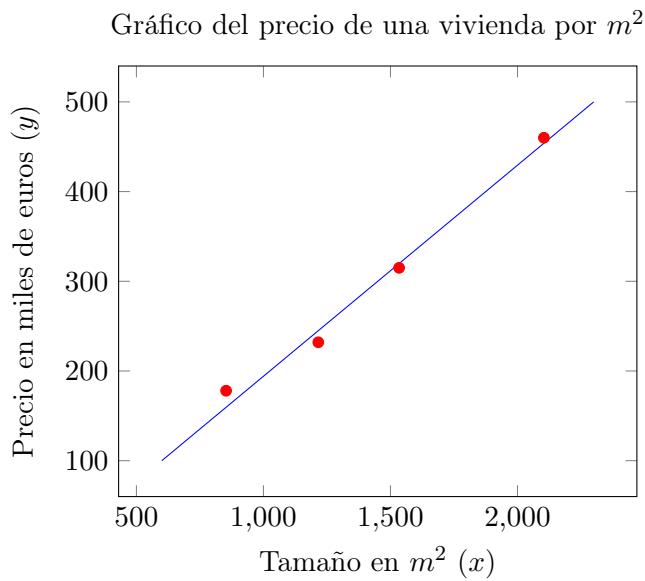


Figura 2.6: Ejemplo de gráfico del precio de una vivienda por m^2 con la función de hipótesis

Esta función, dado un valor cualquiera de m^2 , permite calcular de forma precisa el precio en miles de euros de la vivienda. Por lo tanto, lo que se tiene que encontrar son los valores de esta función lineal. Se define por lo tanto la hipótesis:

$$h_{\theta}(x) = \theta_1 x + \theta_0 \quad (2.6)$$

Esta hipótesis representa la función objetivo a la que se quiere llegar. Lo único que queda por hacer es encontrar los valores de θ_0 y θ_1 .

Por este motivo parece lógico que lo que se quiera conseguir es que para cada dato de entrenamiento la diferencia entre el resultado de aplicar la función de hipótesis y el valor real $(h_{\theta}(x_m) - y_m)^2$ sea lo más pequeño posible para todos los ejemplos de entrenamiento. Es necesario encontrar los valores de θ_1 y θ_0 que minimicen la siguiente función, denominada función de coste:

$$\underset{\theta_0, \theta_1}{\text{minimizar}}, \quad \mathcal{J}(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^i) - y^i)^2 \quad (2.7)$$

Esta función de coste es conocida como función de coste cuadrática y es una de las más utilizadas. Existen otras como la entropía cruzada, el coeficiente Dice (Milletari et al., 2016) y divergencia de Kullback-Leibler (Kullback y Leibler, 1951), que son ampliamente utilizadas.

2.3.2. Método del gradiente descendente

Una vez definida la función de coste queda encontrar la forma de ajustar los pesos en las distintas capas de la red para conseguir reducir el coste. Si se observa un gráfico en 3D de la función de coste mencionada previamente con los dos parámetros θ_1 y θ_0 , se obtiene una gráfica como la siguiente, 2.7:

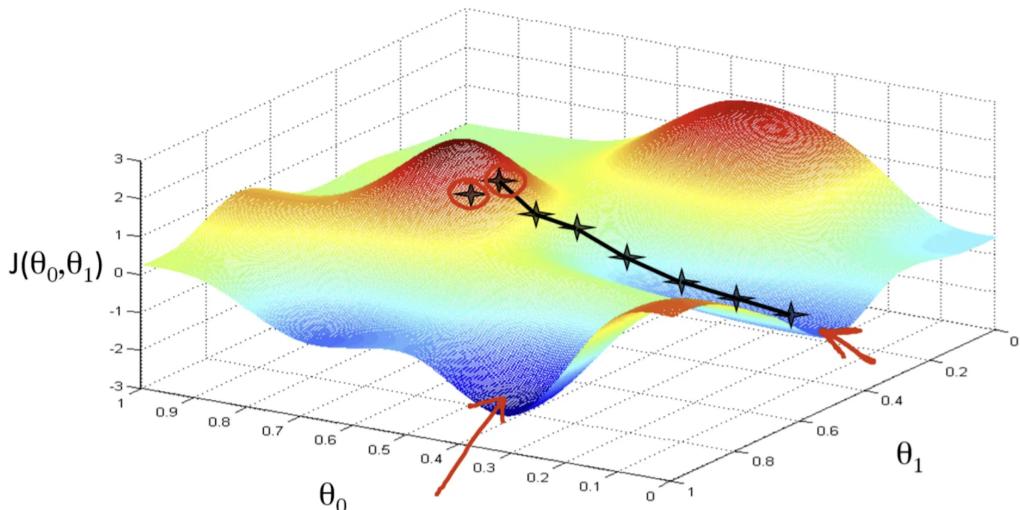


Figura 2.7: Gráfico 3D de la función de gradiente descendente, Fuente: Andrew (2017)

Como se muestra en la Figura 2.7, se tienen distintas zonas donde la función logra

valores mínimos. Esto es lo que se conoce como mínimo local, y el más menor de todos, mínimo global. El objetivo es encontrar la forma de ir iterando para conseguir llegar hasta uno de estos puntos, idealmente al menor.

Para lograr esto matemáticamente se puede recurrir al concepto de la derivada en una función. La derivada de una función en un punto da la pendiente de esa función en dicho punto, lo que permite saber “hacia dónde” hay que dirigirse. De esta forma se puede ir aplicando la derivada a la función hasta conseguir llegar a un mínimo local, momento en el que se habrá obtenido un valor aceptable para el problema. Esto es lo que se conoce como el algoritmo del gradiente descendente:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{J}(\theta_0, \theta_1), \quad \text{para } j = 0 \text{ y } j = 1 \quad (2.8)$$

Se omite la definición de la derivada parcial por no ser el objetivo de este trabajo.

Se puede observar en la expresión 2.8 el coeficiente α . Este valor se llama coeficiente de aprendizaje o tasa de aprendizaje, *learning rate* en inglés. Este coeficiente de aprendizaje ayuda a controlar en cada iteración el tamaño del paso que se da, es decir, cuánto se avanza “hacia adelante”. Es importante a la hora de diseñar un algoritmo elegir un buen coeficiente de aprendizaje ya que dependiendo de si es muy grande un muy pequeño los pasos serán de distinta naturaleza en cada iteración.

Cabe destacar que el símbolo $:=$ no se trata de una igualdad sino de una asignación, es decir, que el valor θ_j pasará a tener en cada iteración el nuevo valor generado al aplicar la expresión una vez. En cada iteración se actualizan a la vez los valores de θ_0 y θ_1 :

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} \mathcal{J}(\theta_0, \theta_1) \quad (2.9)$$

$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} \mathcal{J}(\theta_0, \theta_1) \quad (2.10)$$

$$\theta_0 := \text{temp0} \quad (2.11)$$

$$\theta_1 := \text{temp1} \quad (2.12)$$

El proceso anterior se repite un número determinado de veces hasta conseguir encontrar un mínimo, es decir, cuando la diferencia entre la iteración anterior y la actual sea mínima, momento en el que se dice que la función ha convergido. No obstante, esto no asegura haber encontrado la mejor solución al problema. Se ha de tener en cuenta el concepto de mínimos locales y globales mencionados anteriormente. Un mínimo o máximo se define como el punto en una función cuyo valor es mínimo o máximo, respectivamente, con respecto a todos los puntos vecinos. Conseguir encontrar el mínimo global a veces no es tan fácil y en ocasiones depende de los valores de θ escogidos al principio. Existen algunas técnicas que pueden ayudar a evitar este problema:

- Realizar varias ejecuciones del algoritmo inicializando los valores de θ con diferentes valores en cada ejecución. De esta forma se puede tener más posibilidades de encontrar el máximo global y no quedarnos en el primer valor encontrado.
- Variar la tasa de aprendizaje. Cuanto más grande sea la tasa de aprendizaje más rápido se avanzará, pero existe la posibilidad de que el algoritmo se quede oscilando entre mínimos globales. Por otro lado, tener una tasa de aprendizaje demasiado pequeña puede hacer que se quede atascado en mínimos locales y ralentizar mucho el aprendizaje. Lo ideal es encontrar un valor intermedio adaptado a nuestro problema.

La función de coste de las RN es algo distinta a la vista anteriormente para regresión lineal y se trata de una generalización de la función de coste de regresión logística, tal y como aparece en la expresión 2.13:

$$\mathcal{J}(0) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))_k \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_j^{(l)})^2 \quad (2.13)$$

2.3.3. Retro-propagación: backpropagation

Una vez presentados los conceptos básicos de las RN y definida la función de coste de las RN, se comenta el procedimiento para tratar de minimizar la función de coste llamado Backpropagation (retro-propagación) (Pajares y de la Cruz, 2007).

Como antes el objetivo es minimizar la función de coste siguiente:

$$\mathcal{J}(0) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))_k \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_j^{(l)})^2 \quad (2.14)$$

$$\min_{\Theta} \mathcal{J}(\Theta) \quad (2.15)$$

Y para ello es necesario calcular las derivadas parciales de la mencionada función de coste:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} \mathcal{J}(\Theta) \quad (2.16)$$

A continuación se explica este procedimiento mediante un ejemplo práctico, basado en la red neuronal de la Figura 2.8 (Andrew, 2011).

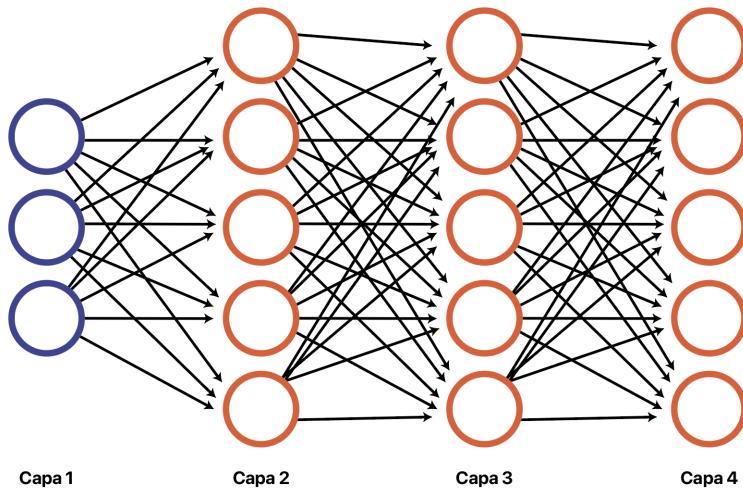


Figura 2.8: Ejemplo de red neuronal

Lo primero que se debe hacer es la propagación hacia adelante en la red neuronal de la Figura 2.9, cuyos cálculos se observan a continuación:

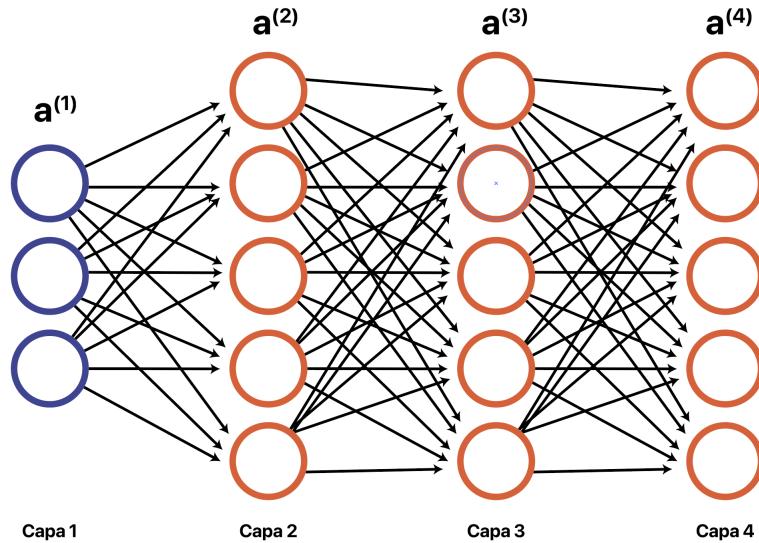


Figura 2.9: Ejemplo de red neuronal

$$\begin{aligned}
 a^{(1)} &= x \\
 z^{(2)} &= \Theta^{(1)} a^{(1)} \\
 a^{(2)} &= g(z^{(2)}) \\
 z^{(3)} &= \Theta^{(2)} a^{(2)} \\
 a^{(3)} &= g(z^{(3)})
 \end{aligned}$$

$$\begin{aligned} z^{(4)} &= \Theta^{(3)} a^{(3)} \\ a^{(4)} &= h_{\Theta}(x) = g(z^{(4)}) \end{aligned}$$

Una vez hecho eso, el paso que se debe realizar es el cálculo de las derivadas. La intuición con este método es que para cada nodo se va a calcular el término $\delta_j^{(l)}$ que va a representar el error del nodo j en la capa l .

Como se vio anteriormente en esta sección, el error es la diferencia en valor absoluto entre lo “predicho” por la RN y el valor real. Como el valor de a en la última capa es el valor final, la predicción del algoritmo para $L = 4$ es:

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad (2.17)$$

Una vez obtenido el error de la última capa hay que calcular el error de las capas anteriores (propagación hacia atrás):

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} * g'(z^{(3)}) \quad (2.18)$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} * g'(z^{(2)}) \quad (2.19)$$

Finalmente queda calcular las derivadas. La derivada parcial vista anteriormente, se puede representar de la siguiente forma sin detallar su complejidad.

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} \mathcal{J}(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \quad (2.20)$$

Para terminar, a continuación se resumen todos los pasos necesarios en la retro-propagación. Se define Δ como un acumulador de la derivada parcial en el que se puede ir actualizando el valor de la derivada parcial en cada iteración. Los pasos son los siguientes:

Para un conjunto de entrenamiento: $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$

1. Establecer $\Delta_{ij}^{(l)} = 0 \quad \forall l, i, j$.
2. Para $i = 1$ a m :
 - a) Establecer $a^{(1)} = x^{(i)}$
 - b) Realizar la propagación hacia adelante para calcular $a^{(l)}$ para todas las capas $l = 2, 3, \dots, L$
 - c) Con el vector de resultados de y^i , calcular $\delta^L = a^{(L)} - y^{(i)}$
 - d) Calcular $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$
 - e) Calcular $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
3. Calcular $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$

Se concluye ya que:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} \mathcal{J}(\Theta) = D_{ij}^{(l)} \quad (2.21)$$

2.4. Validación de la RN

Ahora que ya se conocen todos los conceptos básicos de las RN, queda comprender cómo evaluar si una RN está haciendo bien su trabajo.

2.4.1. Elección de los conjuntos de datos

Lo esencial para conseguir un buen resultado es partir de un buen conjunto de datos (dataset) de entrenamiento, con la calidad suficiente para que el aprendizaje sea adecuado. Un dataset claro y sin errores puede acelerar el proceso notablemente.

Cuando se entrena hay que tener en cuenta que este se realiza sobre el mismo conjunto de datos (los datos de entrada proporcionados a la RN). Esto conlleva el hecho de que la red aprenda patrones específicos para esos datos, por lo que es necesario que sea lo más amplio y representativo posible para que la red pueda generalizar convenientemente y realizar inferencias lo mejor posible sobre datos nunca vistos. Por eso, el hecho de que una RN dé muy buenos resultados con los datos utilizados para entrenar no puede ser un indicativo de calidad, ya que se le debería proporcionar unos datos que nunca antes haya visto para poder determinar su eficiencia. Es por eso que normalmente se realiza una división de los datos de entrenamiento en tres clases (Andrew, 2011), (d'Archimbaud, 2023):

- Conjunto de entrenamiento (training set): es el conjunto que utiliza la RN para para ajustar los parámetros. Será con estos datos con los que la RN evolucionará hasta conseguir la convergencia.
- Conjunto de validación (validation set): es un conjunto de datos que se utiliza para ver cómo se está comportando la RN y conseguir mejores ajustes según los ejemplos disponibles.
- Conjunto de pruebas (testing set): es el conjunto de datos con el que se evalúa la RN al terminar y ver cómo de bien ha aprendido a generalizar.

A la hora de realizar un entrenamiento la cantidad de datos es limitada, por lo que es interesante conocer qué porcentaje de los datos disponibles asignar a cada set. Además, en ocasiones, el set de validación se omite por completo reduciendo los conjuntos a un único *test* de entrenamiento y otro de *test*. Por lo general, una buena división es la que se encuentra en la tabla 2.2 o la tabla 2.3, aunque todo dependerá del proyecto en cuestión y la cantidad de datos disponibles:

Conjunto	Porcentaje (%)
Entrenamiento	70 %
Validación	15 %
Pruebas	15 %

Tabla 2.2: Asignación aproximada por conjunto de entrenamiento

Conjunto	Porcentaje (%)
Entrenamiento	80 %
Pruebas	20 %

Tabla 2.3: Asignación aproximada por conjunto sin validación

2.4.2. Generalización de la RN

Uno de los problemas más comprometidos a la hora de desarrollar una RN es no conseguir que el modelo llegue a la generalización, que no sea capaz de conseguir que la red aprenda obteniendo un porcentaje bajo de aciertos y resultados indeseados para imágenes distintas a las utilizadas en el entrenamiento. Dependiendo de los resultados obtenidos se puede hablar de tres posibles situaciones: subajuste, sobreajuste y generalización (Andrew, 2011). En la Figura 2.10 se puede observar cada una de estas tres situaciones en un problema de regresión lineal, y cómo afectaría cada uno de estos problemas a la predicción.

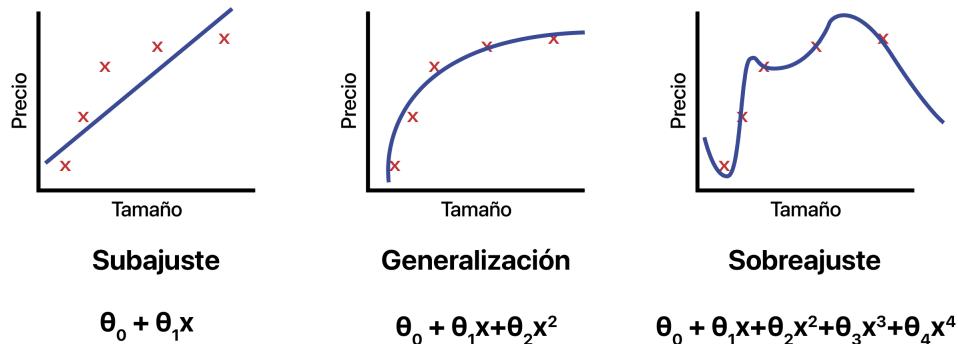


Figura 2.10: Visualización de subajuste, generalización y sobreajuste

- Subajuste o Underfitting: se produce cuando la función objetivo no consigue ajustarse bien al problema, ni siquiera con los datos de entrenamiento. En el ejemplo anterior se puede ver cómo una función lineal no consigue representar correctamente la tendencia de la gráfica. En una situación como esta se puede hablar que el procedimiento tiene un sesgo (*bias*) alto.
- Generalización: se produce cuando la función objetivo consigue ajustarse al problema y predecir de forma precisa valores futuros. Esta es la situación a la que se aspira a la hora de diseñar un algoritmo de DL.
- Sobreajuste u Overfitting: se produce cuando la función objetivo se ajusta demasiado al problema, produciendo un error bajo con los datos de entrada pero muy alto con

otros datos no utilizados para el entrenamiento. En el ejemplo se puede ver cómo la función pasa por todos los puntos, pero hace una gráfica muy artificial que no representa de forma realista la tendencia de la función. En una situación como esta se puede decir que el ajuste tiene un variabilidad alta.

A la hora de abordar este problema existen multitud de soluciones que pueden utilizarse para intentar mejorar el resultado de la RN y conseguir que converja de la mejor manera posible evitando el subajuste o el sobreajuste. Algunas ideas que se proponen con tal propósito son:

- Elegir el número de características correcto: parece lógico que utilizar como única variable a la hora de predecir el precio de una casa la superficie en m^2 va a hacer que no haya una predicción muy precisa en el precio (underfitting). Al mismo tiempo, tener demasiadas características en consideración con muy pocos datos de entrenamiento haría que se obtuviesen valores desproporcionados sin llegar a una correcta generalización (overfitting). Lo ideal sería elegir de forma adecuada qué características seleccionar para realizar el entrenamiento y de esta forma reducir el número de características disponibles para conseguir el balance apropiado y diseñar un modelo que sea capaz de predecir con exactitud el precio.
- A veces, aunque se tengan demasiadas características, cada una tiene una importancia diferente a la hora de predecir el resultado de la RN. En estos casos se puede aplicar lo que se conoce como regularización, donde se mantienen todas las características pero se reduce la magnitud del valor de los parámetros de θ_j . Esto consiste en penalizar determinados valores de θ_j para que su coste aumente y por lo tanto su “influencia” se reduzca. Para ello se utiliza el término λ , conocido como coeficiente de regularización:

$$\mathcal{J}(\theta) = \frac{1}{2m} \left[\sum_{i=0}^m (h_\theta(x^i) - y^i)^2 + \lambda \sum_{j=1}^n \theta_j^2 \right] \quad (2.22)$$

- Aumento de datos: otra de las opciones es aumentar la cantidad de datos de entrenamiento. A veces esto no es posible, o resultaría demasiado costoso, así que lo que se suele hacer en estos casos es generar nuevos datos de entrenamiento utilizando los ya existentes. Por ejemplo, al trabajar con un clasificador de imágenes, se podría voltear, recortar, realzar, etc. una imagen y de esta forma generar variaciones de la misma clase que se pueden utilizar para entrenar.

Como se ha comentado previamente, estos son algunos ejemplos de lo que se podría hacer para intentar conseguir la generalización del modelo de red. También se puede revisar los datos de entrada para cerciorarse de que no hay errores y revisar la estructura de la RN añadiendo o quitando parámetros y capas. Otra opción viable consiste en ajustar los hiperparámetros.

2.4.3. Hiperparámetros

A lo largo de la explicación se ha visto cómo iban apareciendo elementos que permiten definir y modificar el proceso de entrenamiento de la RN. De entre todos ellos son los

hiperparámetros que aparecen en el proceso de entrenamiento, los que determinan en gran medida la evolución de este. Es importante ajustar estos parámetros para aumentar la eficiencia del modelo, reduciendo tiempos de entrenamiento y mejorar su capacidad de predicción, evitando el problema del sub y sobreajuste mencionados previamente. Existen diversos hiperparámetros que se pueden ajustar para mejorar el entrenamiento de la red. Algunos de los más importantes son (Andrew, 2011):

- *Learning rate* o tasa de aprendizaje: controla la velocidad a la que se avanza en el ajuste de los pesos en cada iteración y se ajusta la función de pérdida. Una tasa de aprendizaje muy pequeña puede ralentizar mucho el aprendizaje de la red y hacer que se quede estancada dentro de un mínimo local. Por el contrario, un valor demasiado grande podría hacer que oscile alrededor un mínimo y no consiga llegar nunca al valor óptimo.
- Número de épocas: se refiere al número de veces que la RN repite el proceso de entrenamiento completo en todo el conjunto de datos. Una época se considera completada cuando todos los datos de entrenamiento han pasado por la RN.
- *Batch size* o tamaño del lote: determina cuantos ejemplos de entrenamiento se utilizan en cada iteración. Existen diferentes tipos de *batch size* dependiendo del número de ejemplos de entrenamiento que se utilicen como por ejemplo *batch size* completo, donde se utilizan todos los datos de entrenamiento como un único lote; *mini batch size*, donde se divide el conjunto de entrenamiento en mini lotes con un tamaño fijo y se actualizan los pesos después de cada mini lote; o incluso *batch size* variable, ya que algunos modelos de entrenamiento permiten utilizar lotes de datos variables donde el tamaño del lote se va modificando durante el entrenamiento. Un *mini batch size* más pequeño puede ayudar en los casos en los que se disponga de menos memoria y permite realizar una mejor búsqueda de los mínimos locales ya que los pesos se actualizan con mayor frecuencia, mientras que con un *batch size* más grande se puede obtener una convergencia más rápida y más estabilidad ya que estima a partir de más ejemplos de entrenamiento en cada paso, además de mayor velocidad de entrenamiento.
- Regularización: Como ya se ha visto anteriormente se trata de una técnica utilizada para evitar el sobreajuste de una RN. Existen numerosas técnicas de regularización tales como *Dropout*, L1, L2, etc. que pueden ayudar a resolver este problema. Siempre es necesario elegir el mejor valor de λ para determinar la cantidad de penalización a aplicar durante la actualización de los pesos del modelo.

Capítulo **3**

Redes Neuronales Convolucionales

Los modelos de RN vistos hasta ahora funcionan muy bien en los casos en los que se disponen de datos estructurados como bases de datos, tablas, características, etc. Son valores claramente definidos que se pueden introducir en la RN como *input* y obtener una respuesta apropiada. Sin embargo, cuando se trata de datos no estructurados como imágenes, videos, audio, etc. utilizar una RN convencional no parece ser la mejor de las opciones. Se muestra como ejemplo de la clasificación de una imagen de $64 \times 64 \times 3$ píxeles de un gato (64 píxeles de alto, 64 píxeles de ancho y 3 canales RGB) como se puede ver en la Figura 3.2.



Figura 3.1: Ejemplo de imagen de gato de 64×64 píxeles. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)

El número de características a introducir a la RN sería de del siguiente orden, si estas son los valores de los píxeles:

$$64 \times 64 \times 3 = 12288 \text{ características} \quad (3.1)$$

Con este tamaño de imagen se habría de 12.288 características, que aunque son mu-

chas, todavía es asumible por la mayoría de ordenadores convencionales. No obstante, una imagen así tiene muy poca resolución, y trabajar con resoluciones tan bajas hace que los resultados no sean tan precisos como es de esperar. Si se pone de ejemplo otra imagen con 1 megapixel de resolución, es decir, $1000 \times 1000 \times 3$ como en la Figura 3.2:



Figura 3.2: Ejemplo de imagen de gato de 1000×1000 píxeles. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)

El número de características que se tendrían que introducir a la RN sería de:

$$1000 \times 1000 \times 3 = 3000000 \text{ características} \quad (3.2)$$

La cantidad de características es demasiado grande, y si se tienen por ejemplo 1000 unidades en la primera capa oculta, la matriz de pesos tendría $3,000,000 \times 1000 = 3,000,000,000$ características lo cual es excesivamente grande y se necesitaría una gran cantidad de datos para evitar el problema del sobreajuste al igual que un ordenador muy potente para poder entrenar. Visualmente la red resultante se podría asemejar a la de la Figura 3.3.

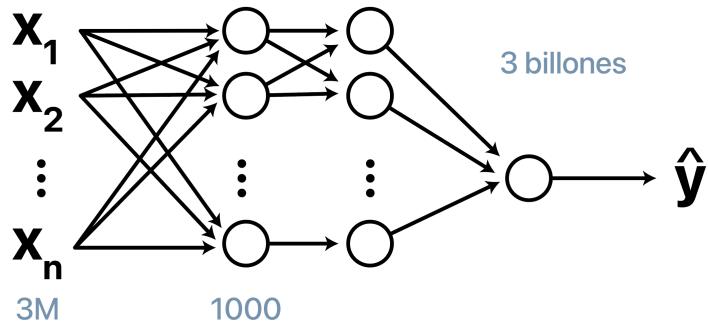


Figura 3.3: Visualización de red neuronal para 3000000 características.

Para aplicaciones de la vida real, trabajar con imágenes con tan poca resolución sería un impedimento a la hora de obtener unos buenos resultados. Es aquí donde entran en juego las denominadas Redes Neuronales Convolucionales, RNC (Convolutional Neural Network, CNN). Las convoluciones constituyen uno de los bloques fundamentales de este tipo de RN, que permiten manejar la información mediante consideraciones locales como son las operaciones de vecindad (convoluciones) involucradas, cuyo objetivo es poder manejar imágenes y clasificarlas, detectar personas u objetos, sustituir un estilo gráfico por otro y dividir la imagen en zonas pertenecientes a distintas clases. En la Figura 3.4 se observa un ejemplo ilustrativo de una clasificación de la imagen de un gato.

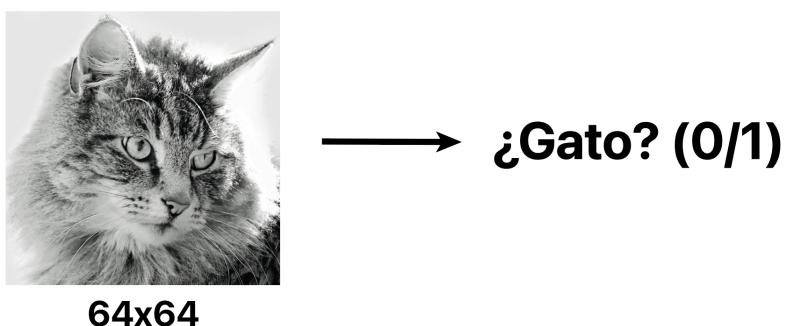


Figura 3.4: Clasificación de una imagen. Imagen bajo licencia Creative Commons, Fuente: Hodan (2022)

3.1. Estructura de las Redes Neuronales Convolucionales

Una diferencia notable en términos de arquitectura de las CNN es que, mientras una RN convencional está formada únicamente por capas de neuronas totalmente conectadas, una CNN está formada por al menos una capa de convolución. Estas capas de convolución son las que permiten extraer características utilizando lo que se conocen como filtros o *kernels*, a diferencia de las redes convencionales que utilizan la multiplicación de matrices. En la Figura 3.5 se deja un esquema de cómo se estructura una CNN esquemáticamente.

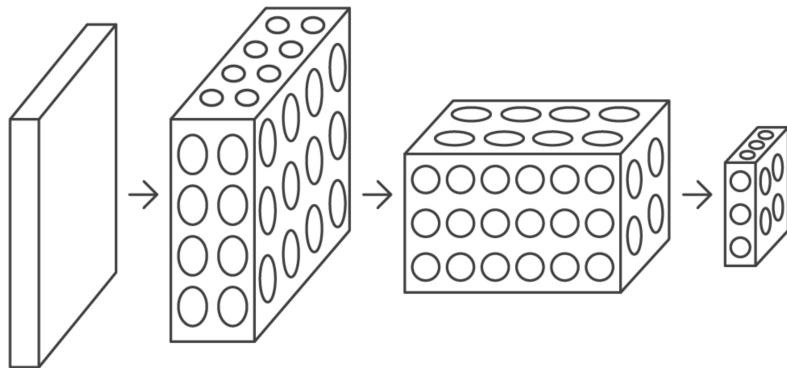


Figura 3.5: Estructura de una RNC, Fuente: Aizel et al. (2021)

En esta sección se analiza el funcionamiento de las CNN, las distintas capas que posee y el proceso de generación de resultados.

3.1.1. Capa de convolución

Como su propio nombre indica el término convolucional se refiere a la operación matemática de la convolución, aunque en términos de RN la definición no coincide exactamente con la definición matemática pura.

Pero, ¿para qué se utiliza la convolución? Por norma general la operación de convolución requiere de dos funciones con valores reales como argumento. Para hacerlo más sencillo se podría imaginar que ambas funciones son las funciones rectángulo. Esta función posee valores que son la unidad en un rango determinado, y cero en el resto. En la Figura 3.6 se muestran dos funciones rectángulo.

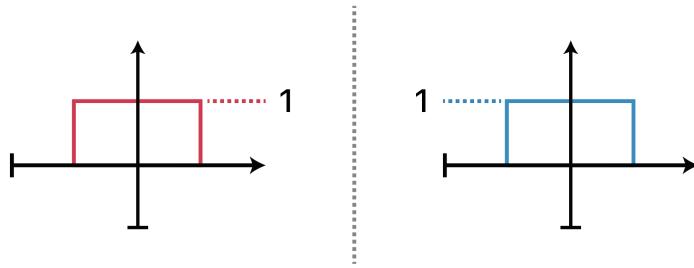


Figura 3.6: Funciones rectángulo

Si se desliza la primera función superponiéndola sobre la segunda, se observa que a partir de un cierto valor las funciones coinciden generando un área común como se puede ver en la Figura 3.7

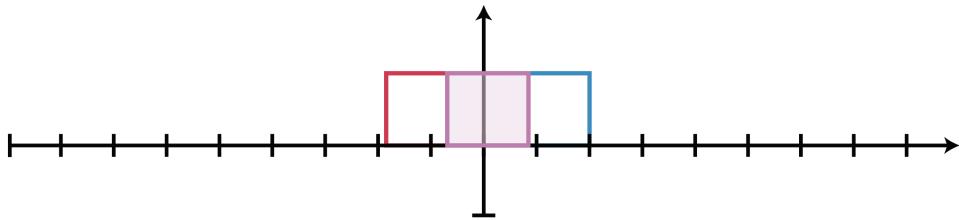


Figura 3.7: Funciones rectángulo generando un área común

Si se van haciendo estos desplazamientos infinitesimales y se anota el valor en una segunda gráfica con respecto al valor de desplazamiento de la primera función se puede ver la obtención de una línea continua. Esto es lo que se conoce como convolución, formalmente definida como:

$$s(\tau) = (f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) g(\tau - t) d\tau \quad (3.3)$$

Una vez introducida la definición matemática de la convolución, es necesario definir su aplicación en el contexto de las RN.

En el ámbito de las CNN, el primer argumento f de la convolución se conoce como entrada o *input*, el segundo g , como núcleo o *kernel* y la salida s como mapa de características o *feature map*. La entrada puede estar formada por un vector o una matriz multidimensional de datos y el núcleo puede estar formado por un vector o una matriz multidimensional de pesos, que se ajustan mediante el proceso de aprendizaje. Esta estructura es la que se conoce como tensor. En el caso de la VC y en el manejo de imágenes tanto la entrada como el núcleo serán matrices. Es por esto por lo que la convolución sobre una imagen I se realiza sobre más de un eje a la vez ya que se tratan de estructuras de datos bidimensionales que serán tratadas como un núcleo bidimensional:

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n) \quad (3.4)$$

Aplicando el mismo criterio se puede extender esta expresión utilizando tensores y núcleos de convolución de dimensión mayor que dos. Para tres dimensiones se tiene la siguiente expresión:

$$S(i, j, k) = (I * K)(i, j, k) = \sum_m \sum_n \sum_p I(m, n, p) K(i - m, j - n, k - p) \quad (3.5)$$

En la Figura 3.8 se muestra un ejemplo ilustrativo de cómo se aplica la convolución utilizando una imagen de 6×6 con un núcleo de 3×3 . En ella se representan los píxeles de la imagen con valores numéricos, que bien podrían representar los valores de intensidad en una imagen de escala de grises.

$$\begin{array}{|c|c|c|c|c|c|} \hline
 3 & 0 & 1 & 2 & 7 & 4 \\ \hline
 1 & 5 & 0 & 9 & 3 & 1 \\ \hline
 2 & 7 & 2 & 5 & 1 & 3 \\ \hline
 0 & 1 & 3 & 1 & 7 & 8 \\ \hline
 4 & 2 & 1 & 6 & 2 & 8 \\ \hline
 2 & 4 & 5 & 2 & 3 & 9 \\ \hline
 \end{array}
 \quad
 \begin{matrix} * \\ 6 \times 6 \end{matrix}
 \quad
 \begin{array}{|c|c|c|} \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 1 & 0 & -1 \\ \hline
 \end{array}
 \quad
 \begin{matrix} 3 \times 3 \\ = \\ 4 \times 4 \end{matrix}
 \quad
 \begin{array}{|c|c|c|c|} \hline
 -5 & -4 & 0 & 8 \\ \hline
 -10 & -2 & 3 & 3 \\ \hline
 3 & 3 & 3 & 3 \\ \hline
 3 & 3 & 3 & 3 \\ \hline
 \end{array}$$

Figura 3.8: Ejemplo de convolución

Al igual que en el ejemplo anterior en la Figura 3.9 se muestra la convolución con el solapamiento total del núcleo junto con sus operaciones.

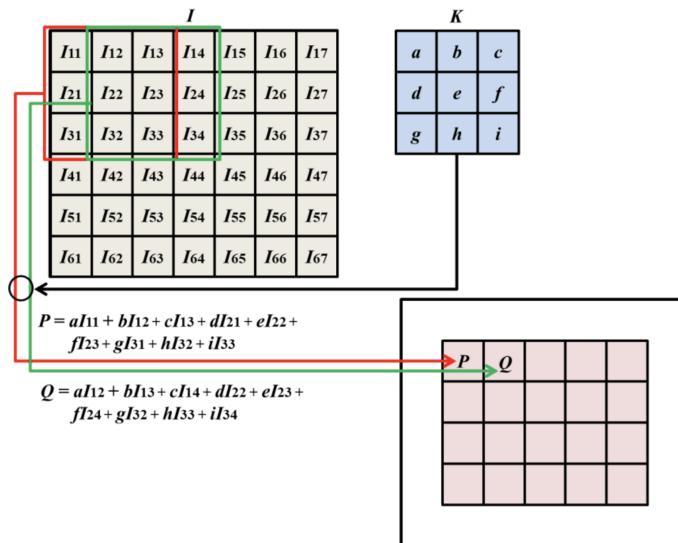


Figura 3.9: Ejemplo de convolución, Fuente: Pajares et al. (2021)

Como se puede ver en la Figura 3.9, la ventana de convolución se va desplazando, y en cada parte se calcula el valor que resulta de sumar cada una de las multiplicaciones de los valores de cada uno de los píxeles de la imagen con el valor correspondiente al núcleo. Dichas operaciones se pueden ver a continuación.

$$P = aI_{11} + bI_{12} + cI_{13} + dI_{21} + eI_{22} + fI_{23} + gI_{31} + hI_{32} + iI_{33} \quad (3.6)$$

$$Q = aI_{12} + bI_{13} + cI_{14} + dI_{22} + eI_{23} + fI_{24} + gI_{32} + hI_{33} + iI_{34} \quad (3.7)$$

Tras realizar todas las operaciones convolucionales, se obtiene una imagen como resultado, que generalmente es de menor dimensión que la original. En el caso del ejemplo se ha pasado de tener una imagen 6×6 a tener una imagen 4×4 . Esto es debido a que el tamaño del núcleo es de 3×3 el cual únicamente tiene 4×4 posibles posiciones para poder

realizar la operación de convolución. Se define que si tenemos una imagen de dimensiones $i \times j$ y se realiza una convolución con núcleo de dimensiones $n \times m$ el tamaño de la imagen resultante será de:

$$|I_{res}| = i - n + 1 \times j - m + 1 \quad (3.8)$$

Parece lógico pensar que si se continúa realizando la operación de convolución llegará un momento en el que la imagen se habrá reducido a un único píxel. Para esto existen distintas técnicas que permiten evitar la pérdida de la resolución a la hora de aplicar la operación de convolución.

En la Figura 3.10 se muestra un ejemplo ilustrativo completo sobre el proceso de la convolución utilizando una imagen y núcleo similar al del ejemplo explicativo anterior.

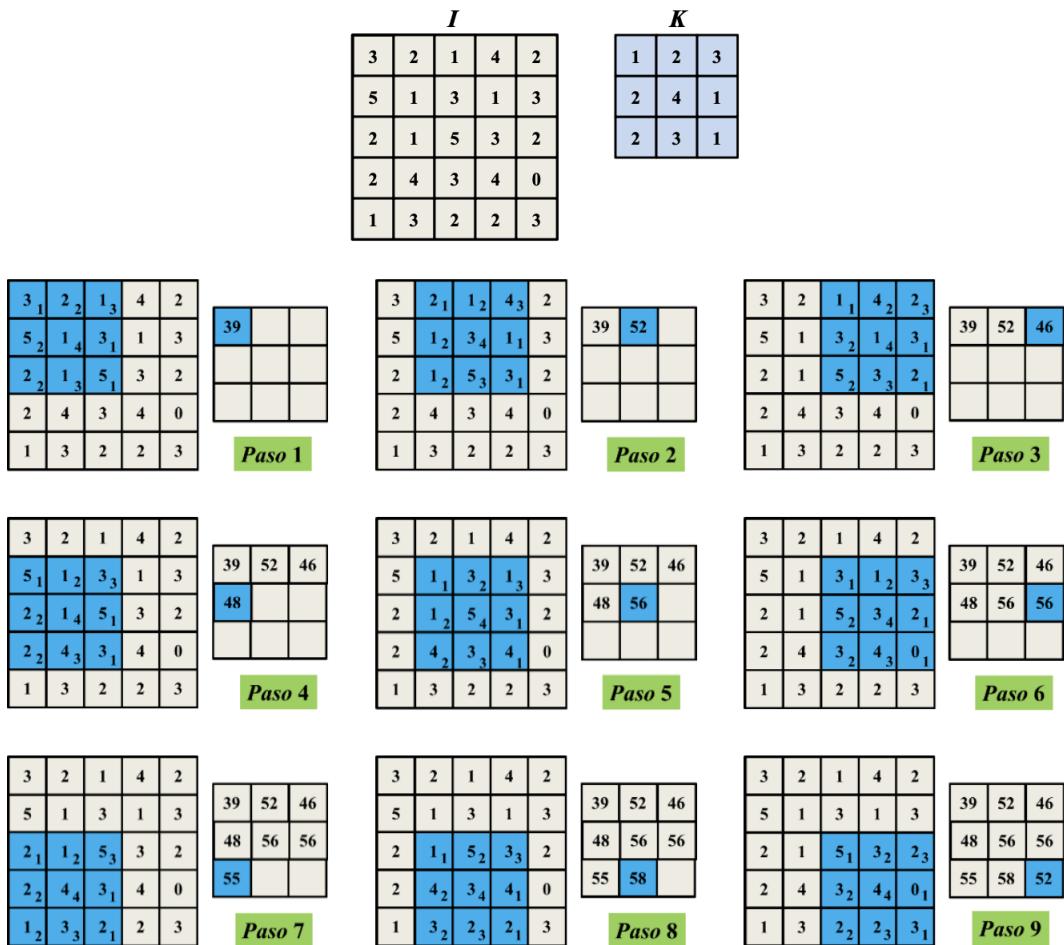


Figura 3.10: Ejemplo completo de convolución, Fuente: Pajares et al. (2021)

3.1.1.1. Padding

Como se ha comentado previamente, cada vez que se aplica la operación de convolución sobre una imagen, esta disminuye de tamaño. Del mismo modo, cada uno de los píxeles de la imagen participan de distinta forma en el resultado final de la operación de convolución. Por ejemplo, si se consideran cualquiera de las 4 esquinas de la imagen, estas participan una sola vez en la convolución y únicamente van a tener representación en las 4 esquinas de la imagen resultante, mientras que un píxel del centro de la imagen participará tantas veces como $n \times m$ tamaño tenga el núcleo. Esto ocasiona que se pierda mucha información en los bordes de la imagen.

Para solventar estos dos problemas, antes de realizar la operación convolucional se puede aplicar lo que se conoce como *padding* (extensión) a la imagen de un número determinado de píxeles y rellenarlo de ceros, para generar así una imagen con nuevas dimensiones como se puede ver en la Figura 3.11.

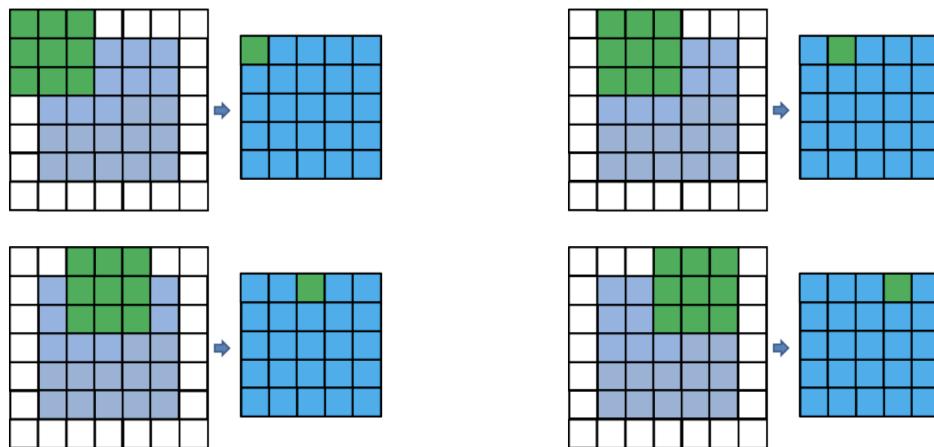


Figura 3.11: Ejemplo de padding, Fuente: Pajares et al. (2021)

De esta forma, la imagen original pasa de tener dimensión 5×5 a tener dimensión 7×7 , y con un núcleo de 3×3 y aplicando la función mencionada anteriormente $|I_{res}| = i - n + 1 \times j - m + 1$ se puede ver que la imagen resultante tendrá dimensión $|I_{res}| = 7 - 3 + 1 \times 7 - 3 + 1 = 5 \times 5$, es decir, se mantiene el tamaño de la imagen original.

Esto es lo que se conoce como *zero-padding*, que consiste en llenar con ceros la zona de ampliación aplicada. Si bien, puede optarse por no aplicar este tipo de extensión, en cuyo caso, se dice que la operación que se realiza es no *zero-padding*. Desde el punto de vista de la implementación, cuando se utiliza este tipo de *padding* se suele indicar como “*same*” y en caso contrario como “*valid*”.

3.1.1.2. Stride

Otro de los conceptos fundamentales en las redes convolucionales es lo que se conoce como desplazamiento o *stride*. Este concepto se define como el número de píxeles que

se desplaza el núcleo en cada convolución a medida que se aplica sobre la imagen de entrada. Es decir, indica el número de posiciones que se saltan antes de aplicar la siguiente convolución como se puede ver en la Figura 3.12.



Figura 3.12: Ejemplo de stride, Fuente: Pajares et al. (2021)

Manteniendo el mismo ejemplo anterior pero en este caso aplicando un *stride* de 2 ($s = 2$), se puede ver cómo del paso 1 al paso 2 se han hecho dos desplazamientos en lugar de uno. En caso de llegar al final de una fila, el *stride* también se aplica al desplazamiento vertical como se puede ver en la Figura 3.13.



Figura 3.13: Ejemplo de stride, Fuente: Pajares et al. (2021)

En este caso la ecuación para calcular el tamaño de la imagen resultante I_{res} para una imagen de tamaño $i \times j$ con *padding* p y un núcleo de tamaño $n \times m$ y *stride* s es:

$$|I_{res}| = \left(\frac{i + 2p - n}{s} + 1 \right) \times \left(\frac{j + 2p - m}{s} + 1 \right) \quad (3.9)$$

Cabe destacar que en caso en el que el filtro de convolución sobresalga de la imagen debido al *stride* no se realizará la convolución por convención como se puede ver en la Figura 3.14.

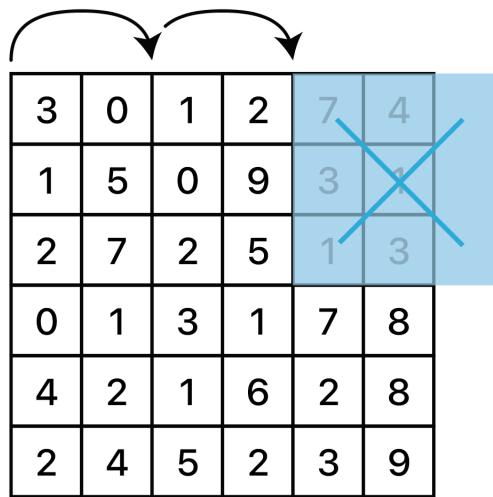


Figura 3.14: Ejemplo de no realización de la convolución cuando hay desbordamiento

3.1.2. Convolución en volúmenes

Como resulta lógico, las operaciones de convolución no se limitan únicamente a los espacios en dos dimensiones, sino que se pueden aplicar en los espacios tridimensionales, lo que resulta imprescindible cuando se trabaja con imágenes ya que normalmente se trabaja con 3 canales de color RGB.

Tomando como ejemplo el de la Figura 3.15, la imagen de 5×5 pasará a tener tamaño 5×3 sobre la que se aplica una convolución con un filtro de tamaño $3 \times 3 \times 3$. Una forma de verlo es pensar que se tiene una pila de 3 imágenes junto con otra pila de 3 filtros de convolución que se van ejecutando de forma simultánea.

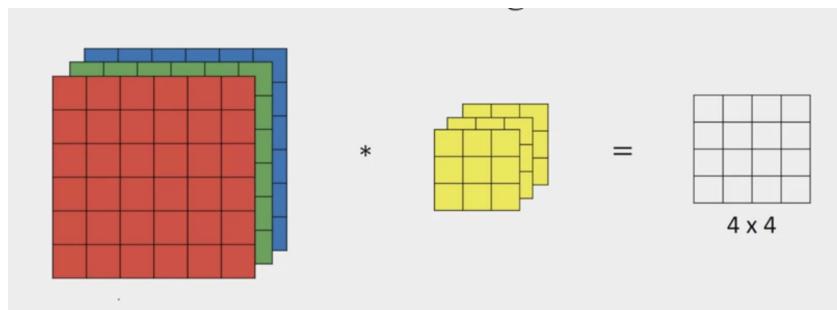


Figura 3.15: Ejemplo de convolución en volúmenes, Fuente: Andrew (2017)

La convolución se realiza como en el caso anterior con la única diferencia de que después de cada convolución se suma el resultado de cada una de las capas para generar la salida resultante como se puede observar en la Figura 3.15.

Por último, como se puede apreciar, el resultado obtenido es de una dimensión, y esto es porque solo se ha aplicado un filtro. No obstante, puede darse el caso en el que se quieran

detectar varias características de la imagen y que se quiera aplicar diferentes filtros. Se puede seguir esta idea realizando varias convoluciones con filtros diferentes y apilando estos resultados como si fuese un volumen igual que la imagen de entrada como se puede ver en la Figura 3.16.

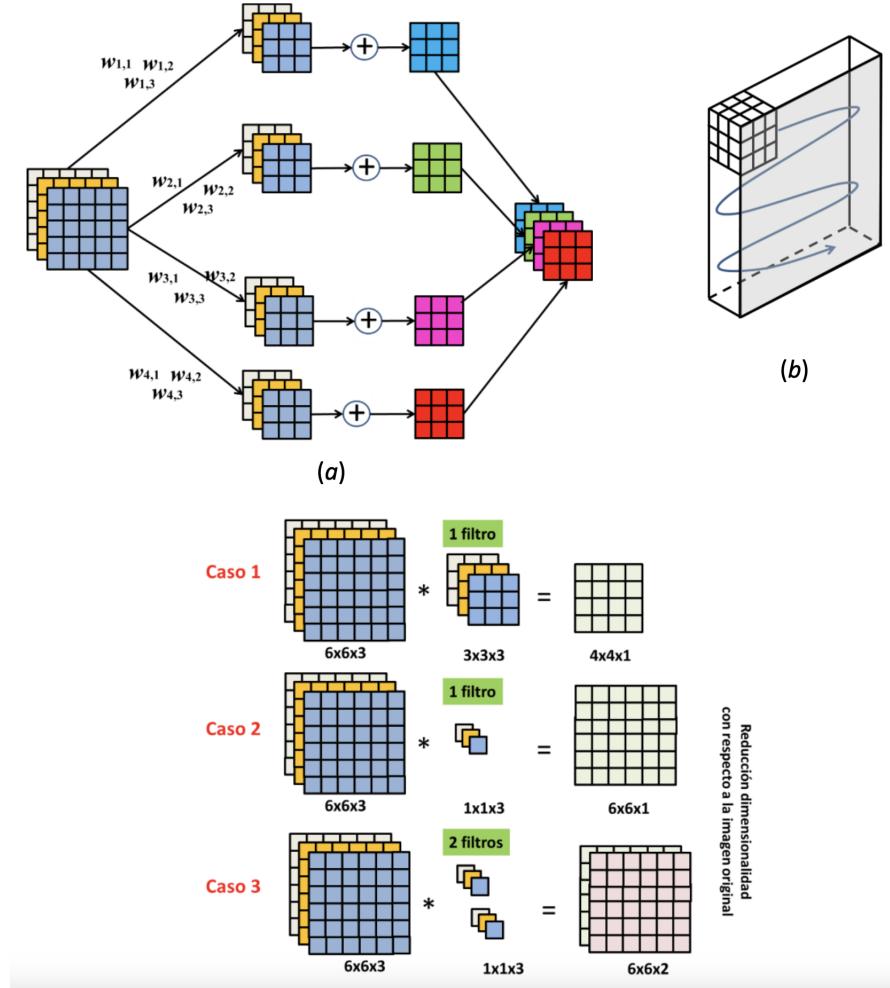


Figura 3.16: Ejemplo de aplicación de varios filtros, Fuente: Pajares et al. (2021)

La ecuación para determinar el tamaño de la salida es muy similar a la vista anteriormente, pero añadiendo como ultima dimensión el número de canales o filtros (n_c') que se utilicen:

$$|I_{res}| = \left(\frac{i + 2p - n}{s} + 1 \right) \times \left(\frac{j + 2p - m}{s} + 1 \right) \times n_c' \quad (3.10)$$

3.1.3. Pooling

Además de las capas convolucionales, en las CNN se encuentran otro tipo de capas denominadas de agrupamiento o de *pooling* encargadas de reducir el tamaño de la repre-

sentación con el fin de poder acelerar el cálculo y permitir que sean algo más robusta a la hora de detectar características. A diferencia de las capas convolucionales estas no disponen de ningún *kernel* y tampoco tendrán parámetros para aprender.

Lo primero que procede hacer es dividir la imagen o el mapa de características de entrada en varias regiones o ventanas de un tamaño determinado $N \times N$ como se ve en la Figura 3.17.

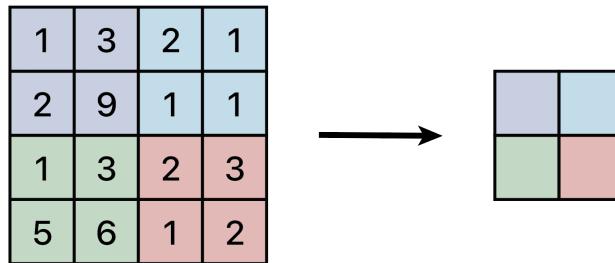


Figura 3.17: Ejemplo de pooling

Posteriormente se debe realizar sobre cada una de estas regiones una operación que devolverá un valor que pasará a ser el valor de la correspondiente operación aplicada sobre la región. Entre las operaciones que se pueden realizar se encuentran las siguientes, siendo las dos últimas las más utilizadas:

- *Min Pooling*: Consiste en elegir de todos los valores de la ventana el valor más pequeño.
- *Max Pooling*: Consiste en elegir de todos los valores de la ventana el valor más grande.
- *Average Pooling*: Consiste en hacer una media de todos los valores.

Para cada región se obtiene el valor resultante según la operación aplicada. A continuación, se muestra el ejemplo anterior aplicando *Max Pooling* en la Figura 3.18.

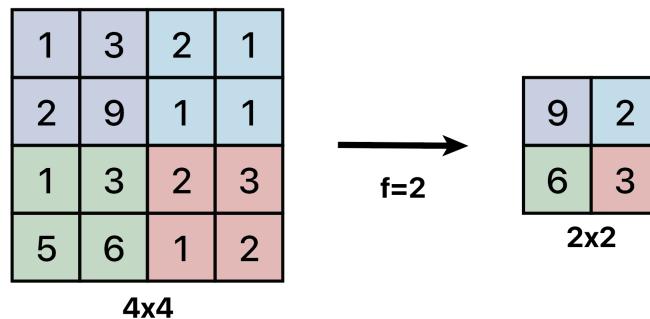


Figura 3.18: Ejemplo de pooling

Por lo tanto, esta operación es como si se aplicase un filtro de tamaño dos ($f = 2$) con un *stride* de dos ($s = 2$), siendo estos los hiperparámetros a fijar para cada capa de *pooling*.

f : tamaño del filtro
 s : *stride* o desplazamiento
 tipo de *pooling*

En ocasiones cuando se posiciona la ventana en las proximidades de los bordes algunos de los elementos de la ventana pueden quedar fuera de los elementos de entrada. Para poder obtener representación también de estos elementos se puede extender con valores de cero aplicando *padding* como se ha visto anteriormente (*zero-padding*).

3.1.4. Sobreajuste, *weight decay* y *dropout*

Ya se ha mencionado en el capítulo anterior el problema que surgen en las RN cuando se ajusta un número elevado de pesos en numerosas neuronas. Se produce lo que se conoce como sobreajuste. Un método muy utilizado para solventar este problema es la normalización, que como se vio anteriormente, es una técnica que permite penalizar determinados valores para hacer que aumente el coste y por lo tanto reducir su influencia.

Uno de los tipos de regularización que más se utilizan es la regularización L2. Para implementarlo se tiene que incorporar a la función de error la magnitud cuadrada de los pesos de la red, es decir, para cada uno de los pesos w se agrega $0,5\lambda w^2$ a la función de error. Esta regularización penaliza fuertemente a los pesos con valores altos y menos a los que tienen valores más discretos, haciendo que la red tenga en cuenta de alguna forma a todas sus entradas en lugar de priorizar solo alguna de ellas. Durante la actualización de los pesos en el proceso de gradiente descendente, el uso de esta regularización significa que cada uno de los pesos tiende a cero, y es debido a esto por lo que la regularización L_2 se conoce comúnmente como disminución de peso o *weight decay* en inglés.

No obstante, esta no es la única forma que existe para paliar el sobreajuste. Existe otra técnica llamada *dropout* que ayuda a tratar este problema y mejora el rendimiento del modelo. El *dropout* consiste en “apagar” un número determinado de neuronas de forma aleatoria durante el entrenamiento. Esto hace que las neuronas sean unas más independientes evitando que el modelo se especialice en ciertos subconjuntos de características.

La elección de neuronas que se van a desactivar se hace de forma aleatoria. Se puede establecer un hiperparámetro que les asigne una probabilidad de supervivencia durante la fase de entrenamiento a cada capa, como se puede observar en la Figura 3.19.

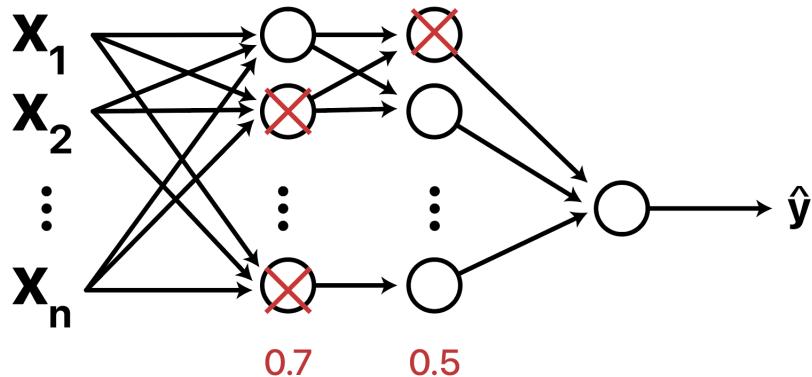


Figura 3.19: Cancelación de neuronas en base a un porcentaje

El *dropout* funciona porque cuando se tienen redes muy profundas, al actualizar los pesos de forma conjunta puede darse el caso que algunas conexiones tengan mayor capacidad predictiva que otras. Si en cada iteración se hace que solo una parte de estas neuronas se actualicen, consigue un mejor ajuste de los pesos. En las siguientes iteraciones las neuronas activas y desactivadas cambiarán y podrán seguir ajustándose hasta terminar el entrenamiento. A esta técnica se la conoce como co-adaptación.

En las CNN el *dropout* se puede aplicar tras la salida de una capa y tras un agrupamiento (*pooling*) como se ve en el ejemplo de la Figura 3.20.

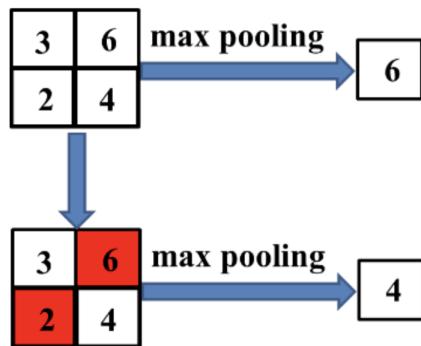


Figura 3.20: Ejemplo de dropout con pooling, Fuente: Pajares et al. (2021)

3.1.5. La función softmax

La función softmax o también llamada función exponencial normalizada se aplica en una de las capas que generalmente aparece al final de la RNC. Se utiliza principalmente para transformar un vector de valores reales x de n -dimensiones en otro vector $\text{softmax}(x)$ de n -dimensiones formado por valores reales en el rango $[0, 1]$, es decir, su propósito principal es el de transformar un vector de valores reales en un vector de probabilidades, donde cada elemento del vector representa la probabilidad de pertenecer a una clase determinada. Se define de la siguiente forma:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} \quad \forall i = (1, 2, \dots, n) \quad y \quad x = (x_1, x_2, \dots, x_n) \in \mathbb{R} \quad (3.11)$$

Capítulo 4

Segmentación Semántica: modelos de RN

La segmentación semántica es una rama mixta del DL y la VC que se encarga de etiquetar regiones de una imagen que forman parte de la misma clase. Su objetivo es clasificar cada uno de los píxeles (o una parte de ellos) como perteneciente a una clase o categoría. Esta tarea se conoce comúnmente como predicción densa.

Algo importante a tener en cuenta es que no se están detectando objetos. Si, por ejemplo, se tiene una imagen de un árbol; el tronco y las hojas son dos categorías diferentes pertenecientes al mismo objeto, en este caso el árbol. Y del mismo modo, si se tienen, por ejemplo, dos plantas en una foto con tierra, ambas tierras pertenecerán a la misma categoría, aunque sean de objetos diferentes. Es decir, lo importante es la clasificación a nivel de píxel, identificar a qué clase pertenece cada uno independientemente del objeto del que formen parte, Jordan (2018).

Para medir la bondad de la clasificación se utiliza frecuentemente el coeficiente de Jaccard (1901), que determina la similitud de dos conjuntos A y B (también conocido como intersección sobre la unión, *intersection over union* en inglés) y definido de la siguiente forma:

$$J(A, B) = |A \cap B| / |A \cup B| \quad (4.1)$$

Con A siendo el conjunto de píxeles de la misma categoría relativo a la segmentación predicha y B el correspondiente al *ground truth*.

En este capítulo se estudia la estructura del tipo *encoder-decoder* en la que se basan los modelos DeepLab V3, SegNet y Unet que han sido los utilizados para realizar este trabajo. En lo que sigue, se utiliza como referencia base Pajares et al. (2021), junto con las referencias asociadas para cada uno de los conceptos que se exponen.

4.1. Modelo Encoder-Decoder

Un modelo de *encoder-decoder*, también denominado *autoencoder* o autocodificador se encarga de transformar una secuencia de entrada en otra secuencia de salida lo más parecida posible a la del ejemplo de entrada, lo que significa que las dimensiones del vector de salida y las dimensiones del vector de entrada tienen que ser las mismas.

Como su propio nombre indica un modelo de *encoder-decoder* está formado por dos partes bien diferenciadas. El *encoder* es el responsable de tomar la secuencia de entrada y comprimirla en una representación más compacta y significativa que se manda al *decoder*. Tras esto, el *decoder* lo transforma en la secuencia de salida deseada. Por ejemplo, si se quiere traducir una frase de español a inglés el *encoder* capturaría el significado de la oración de origen y el *decoder* utilizaría esta información para generar la frase traducida al inglés.

Por lo tanto, los *autoencoder* se alejan de la idea de clasificación tradicional de las RN ya que la salida no va a ser una categoría, clase o etiqueta sino que va a ser algo mucho más complejo.

Existen dos tipos de *autoencoders* que se diferencian principalmente por el tipo de arquitectura sobre el que están construidos. Estos son los *autoencoders* neuronales y los *autoencoder* convolucionales.

4.1.1. Autoencoder neuronal

En las RN vistas anteriormente conviene recordar que el objetivo del problema es el de clasificación, es decir, que la entrada y salida no van a tener el mismo número de neuronas, ya que las neuronas de entrada dependerán del tamaño de la entrada que se quiera permitir y las neuronas de salida dependerán del número de clases a identificar. En el caso de los *autoencoders* neuronales esto no es así, porque hay que recordar que tienen que tener el mismo número de neuronas de entrada como de salida.

Por ejemplo, si se habla de una imagen, cada uno de los píxeles de la imagen de entrada tiene asociado una variable propia a la salida de la red. Por esto no resulta adecuado directamente el procedimiento de reducción del tamaño de los datos en las capas porque haría que el número de neuronas de entrada y salida no fuese el mismo.

Para solucionar esto una vez reducido el tamaño de entrada se añade otra etapa que incrementa la resolución de los datos. En la Figura 4.1, se muestra el proceso completo de un *autoencoder* que como se puede observar tiene forma de reloj de arena y simétrica respecto a la capa oculta central, llamada a veces cuello de botella.

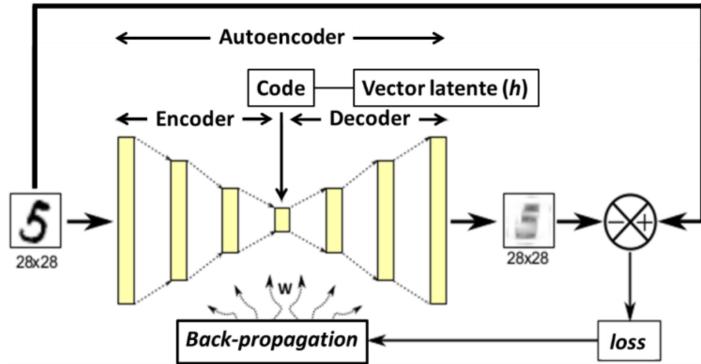


Figura 4.1: Proceso completo de un autoencoder, Fuente: Pajares et al. (2021)

Como se ha adelantado previamente, las primeras capas del *autoencoder* constituyen el codificador (*encoder*) encargadas de reducir el tamaño de la entrada para obtener una representación más compacta y significativa que se conoce como vector latente o code (*h*). Las capas siguientes constituyen el decodificador (*decoder*) que están encargadas de reconstruir la información más relevante de los datos originales tomando como *input* el vector latente.

4.1.2. Autoencoder convolucional

El *autoencoder* convolucional tiene algunas características que lo diferencian notablemente del *autoencoder* neuronal.

Anteriormente, se ha explicado cómo se aplican las operaciones de agrupamiento (*pooling*) en la reconstrucción de RNC. Estas operaciones, realizadas a través de capas de agrupamiento, reducen gradualmente el tamaño de los datos de entrada, con respecto a la imagen original. En este tipo de redes, tras completar la etapa convolucional, se obtiene un conjunto de imágenes (datos) de baja resolución que se serializan en un vector que sirve de entrada para una RN densa (totalmente conectada), donde se lleva a cabo la tarea de clasificación.

Si se elimina la RN densa final de este proceso, se puede construir la etapa de codificación de un *autoencoder* convolucional. En este caso, se estaría transformando una imagen en un vector que, si tiene las dimensiones adecuadas, podría representar la representación latente de la imagen original.

Para poder construir correctamente el *encoder* y decoder, hay que tener en cuenta una cosa. Cuando se realizan operaciones de agrupamiento en una red convolucional, por ejemplo una operación de *max pool*, se pierden $N \times N - 1$ píxeles con una función no invertible. Entonces, ¿Cuál es el procedimiento que utiliza el decodificador para pasar de 1 píxel a $N \times N$ píxeles?

4.1.2.1. Reconstrucción mediante upsampling y convolución transpuesta

Para solventar el problema recién introducido se pueden realizar dos operaciones conocidas como *upsampling* y convolución transpuesta.

Si analizamos un ejemplo similar al anteriormente visto a la hora de construir el modelo *eencoder-decoder*, se ve que se representa un *autoencoder* convolucional con dos etapas convolucionales enfrente en el cuello de botella.

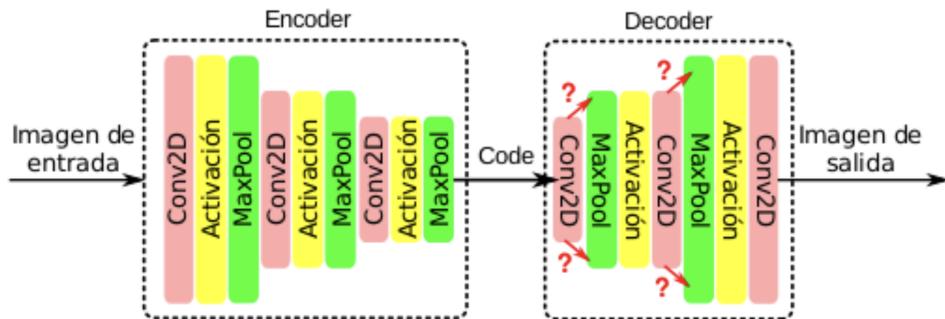


Figura 4.2: Autoencoder convolucional; Fuente: Pajares et al. (2021)

No obstante, como se puede ver en la Figura 4.2, en esta segunda etapa se observa una contradicción ya que tras realizar la capa de agrupamiento de *max pool* se ve que aumenta el tamaño de la imagen. Esto es porque en la etapa de decodificación se sustituye la operación de reducción por una etapa de reconstrucción, cuyo objetivo es construir a partir de 1 píxel, sus vecinos de $N \times N$ píxeles. Este es el método que se conoce como *upsampling* (sobremuestreo). En esta etapa por lo tanto se introducen dos nuevos hiperparámetros que son el tamaño de la vecindad y el procedimiento para llenar los valores de los $N \times N$ píxeles a partir del valor de 1 único píxel.

Algunos de los procedimientos que se utilizan para realizar el *upsampling* son:

- En el primero lo que se hace es copiar el valor de cada píxel de la imagen original en la esquina superior izquierda, rellenando los píxeles restantes con ceros.
- Mientras que en el segundo lo que se hace es copiar el valor del píxel original en todos los píxeles de la nueva imagen.

<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>2</td></tr> <tr><td>3</td><td>4</td></tr> </table>	1	2	3	4	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>0</td><td>2</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> <tr><td>3</td><td>0</td><td>4</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	1	0	2	0	0	0	0	0	3	0	4	0	0	0	0	0	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr><td>1</td><td>1</td><td>2</td><td>2</td></tr> <tr><td>1</td><td>1</td><td>2</td><td>2</td></tr> <tr><td>3</td><td>3</td><td>4</td><td>4</td></tr> <tr><td>3</td><td>3</td><td>4</td><td>4</td></tr> </table>	1	1	2	2	1	1	2	2	3	3	4	4	3	3	4	4
1	2																																					
3	4																																					
1	0	2	0																																			
0	0	0	0																																			
3	0	4	0																																			
0	0	0	0																																			
1	1	2	2																																			
1	1	2	2																																			
3	3	4	4																																			
3	3	4	4																																			

Figura 4.3: Ejemplo de upsampling, Fuente: Pajares et al. (2021)

La segunda operación posible para poder realizar el proceso de reconstrucción es la convolución transpuesta. Para comprender esta operación, es útil recordar que, en la convolución convencional, el tamaño de la imagen resultante dependía del relleno utilizado. Además, la convolución de una imagen con un filtro puede expresarse como una multiplicación matricial, donde el filtro de convolución se convierte en una matriz de convolución. Tanto la imagen original como la imagen resultante se serializan en un vector final.

Para aumentar la resolución y revertir el efecto de la convolución, se requiere una nueva matriz de convolución cuyas dimensiones sean las transpuestas de la matriz anterior. Además, los valores de esta matriz de convolución transpuesta se pueden aprender de la misma manera que los valores de la matriz de convolución convencional. Durante el proceso de aprendizaje de los parámetros del *autoencoder*, se puede aprender la mejor forma de reconstruir los píxeles de una etapa a partir de los píxeles de una etapa de resolución más baja. En la Figura 4.4 se puede ver un ejemplo de convolución traspuesta.

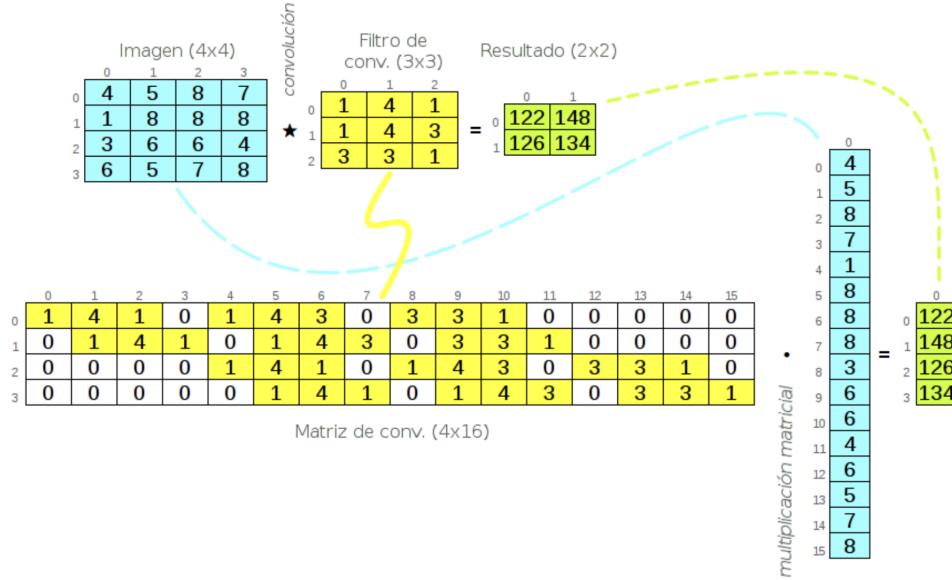


Figura 4.4: Operación de Convolución traspuesta, Fuente: Pajares et al. (2021)

4.1.3. Funciones de pérdida y activación

Para poder retropropagar el error durante el entrenamiento, es necesario utilizar una función de pérdida que permita cuantificar la discrepancia entre las predicciones del modelo y los valores reales de los datos. Del mismo modo, es importante seleccionar una función de activación para las neuronas. En esta sección se expone una pequeña descripción de ambas desde la perspectiva de los *autoencoders*, utilizando los conceptos previamente mencionados en secciones anteriores.

4.1.3.1. Función de pérdida

El objetivo del *autoencoder* es reconstruir el ejemplo de entrada proporcionado, y como los valores de la salida en la etapa final de la decodificación se comparan con los valores de dicho ejemplo, el problema de aprendizaje de los parámetros de la red se puede formular como el ajuste de una regresión múltiple con tantas funciones de pérdida como atributos (variables de entrada) tenga cada ejemplo.

En el caso de una imagen, sería con tantas funciones de pérdida como píxeles tenga la imagen original. Para combinar en una única función todas esas funciones de pérdida, se puede calcular su valor promedio.

Como función de pérdida de un *autoencoder* se utilizan habitualmente algunas de las funciones de pérdida más apropiadas y populares para regresión:

- Error cuadrático medio (ECM), también conocido por pérdida cuadrática, L_2 o MSE (Mean Square Error), a veces con regularización.
- Error absoluto medio (EAM), también conocido por pérdida L_1 o MAE (Mean Absolute Error).
- Pérdida Log-Cosh, que es una versión suavizada del error absoluto medio.

En la Figura 4.5 se puede ver como cada una de estas funciones de pérdida varían con el error, obtenido como la diferencia entre el valor de entrada y de salida. Conviene reseñar que para el caso concreto de un *autoencoder* convolucional que se use para un problema de VA, se podrían utilizar otras funciones de pérdida, tales como medidas de comparación de imágenes o de su densidad espectral, entre otras.

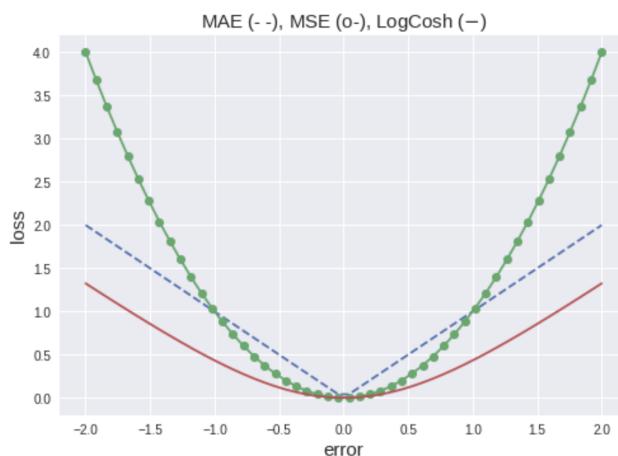


Figura 4.5: Variación de error de las funciones de pérdida, Fuente: Pajares et al. (2021)

4.1.3.2. Función de activación

Como el problema se trata como una regresión, la activación en la última capa de las neuronas tiene que ser lineal. Para el resto es conveniente utilizar funciones de activación

que eviten el desvanecimiento del gradiente como con la función ReLU. Esta función es una buena elección para el *autoencoder*, ya que la profundidad total de un *autoencoder* es el doble que el de la RN o convolucional asociada a su etapa de codificación.

4.2. Modelos de redes neuronales para segmentación semántica

Una vez explicado cómo funcionan las RNC y el modelo de *encoder-decoder* se explican varios modelos dentro del campo de la segmentación semántica. Se comienza explicando la estructura general de un modelo de segmentación semántica y posteriormente se explican los tres modelos utilizados y enfrentados en este proyecto que son: DeepLab v3, SegNet y U-net.

4.2.1. Modelo Encoder-Decoder en segmentación semántica

Una estructura típica de *encoder-decoder* proporciona un esquema general de un modelo de segmentación semántica. El planteamiento de esta red aplicado a la segmentación semántica consiste en construir una secuencia de capas convolucionales que realizan un submuestreo y sobremuestreo, de forma que, en primer lugar se reducen las resoluciones espaciales de forma progresiva mediante operaciones convolucionales y de agrupamiento, para luego incrementar las resoluciones en el orden inverso con operaciones de deconvolución y *unpooling*. En definitiva, se trata de un planteamiento multiescala que codifica la información contextual a través de una serie de filtros u operaciones de *pooling*, mientras las capas posteriores capturan características más finas tales como detalles y bordes existentes en los objetos.

En la Figura 4.6 se puede ver la estructura típica de un *autoencoder* aplicado a segmentación semántica con una imagen de entrada y salida que tiene la misma resolución, pero con la segunda imagen ya clasificada. Para esto se necesita un conjunto de datos ya etiquetados, ya que durante el entrenamiento la salida de la red se compara con la imagen segmentada.

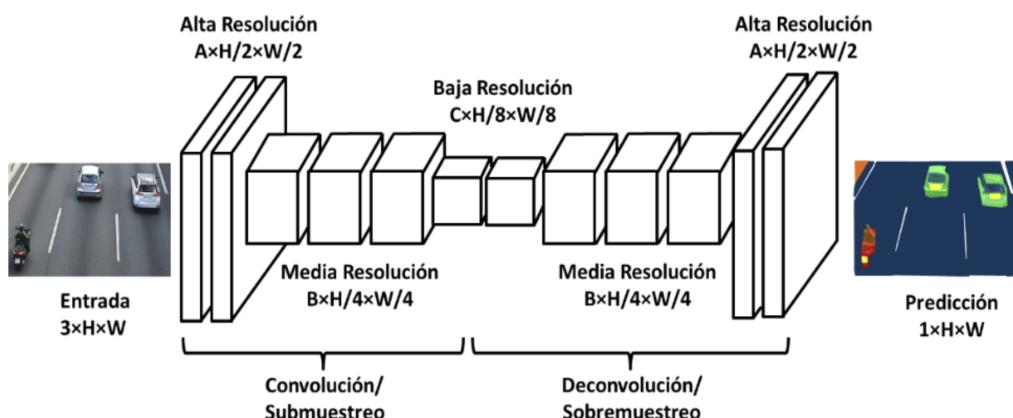


Figura 4.6: Modelo encoder-decoder, Fuente: Pajares et al. (2021)

4.2.2. Modelo DeepLab3

Antes de explicar el Modelo de DeepLab3 conviene explicar el modelo originario DeepLab. Este modelo surge en 2018 cuando Chen et al. (2017) plantean aplicar la convolución dilatada, conocida también como *atrous*, para segmentación semántica. La convolución dilatada consiste en dilatar el núcleo insertando espacios entre sus elementos. La razón de dilatación se controla mediante el hiperparámetro d . La convolución dilatada se define de la siguiente forma:

$$o(x) = \sum_{k=1}^K i[x + d * k] w[k] \quad (4.2)$$

Las convoluciones dilatadas se utilizan para incrementar el campo receptivo de las unidades de salida sin incrementar la dimensión del núcleo, lo que resulta ciertamente efectivo. Si se tiene en cuenta la razón d , la dimensión de salida o y un núcleo de dimensión k , se puede establecer la nueva dimensión del núcleo de la siguiente forma:

$$l = k + (k - 1)(d - 1) \quad (4.3)$$

En la Figura 4.7 se puede ver un ejemplo de convolución dilatada para $i = 7$, $k = 3$, $d = 2$, $p = 0$ y $s = 1$, resultando una salida de $o = 3$.

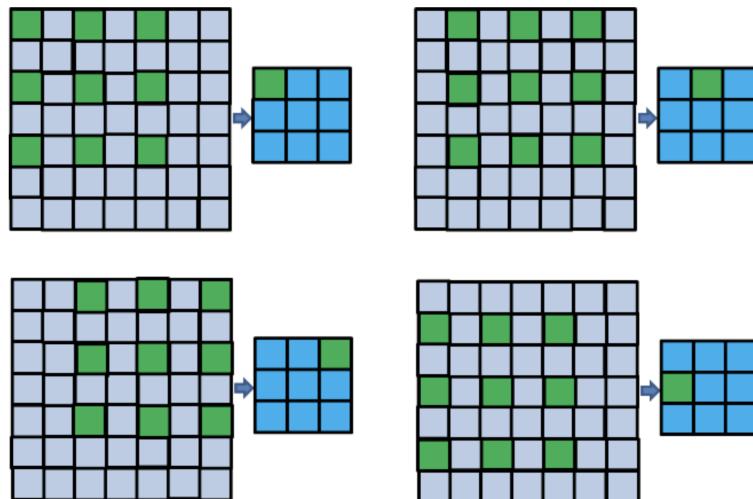


Figura 4.7: Ejemplo convolución dilatada, Fuente: Pajares et al. (2021)

La aplicación de la convolución dilatada permite extender el campo de visión de los filtros al incorporar información contextual global de las imágenes sin incrementar el número de parámetros ni el coste computacional.

Por otro lado, Chen y col. (2017) proponen también un esquema piramidal bajo el esquema denominado Agrupamiento Piramidal Espacial Dilatado (ASPP, *Atrous Spatial*

Pyramid Pooling). Este esquema permite capturar objetos e información contextual a diferentes escalas, gracias a los filtros aplicados a diferentes resoluciones. Por último, mejoran la localización de los límites de los objetos combinando métodos basados en CNN y modelos de gráficos probabilísticos. Combinando operaciones de *pooling* de máximo (*maxPooling*) y submuestreo en las CNN se logra invariancia respecto a la localización pero a costa de la precisión. Para superar este inconveniente, los autores proponen combinar las respuestas de la capa final de la CNN con una totalmente conectada.

Se propone como CNN las arquitecturas VGG-16 o ResNet-101, en las que se transforman todas las capas totalmente conectadas a capas convolucionales. Este esquema se puede ver en la Figura 4.8.

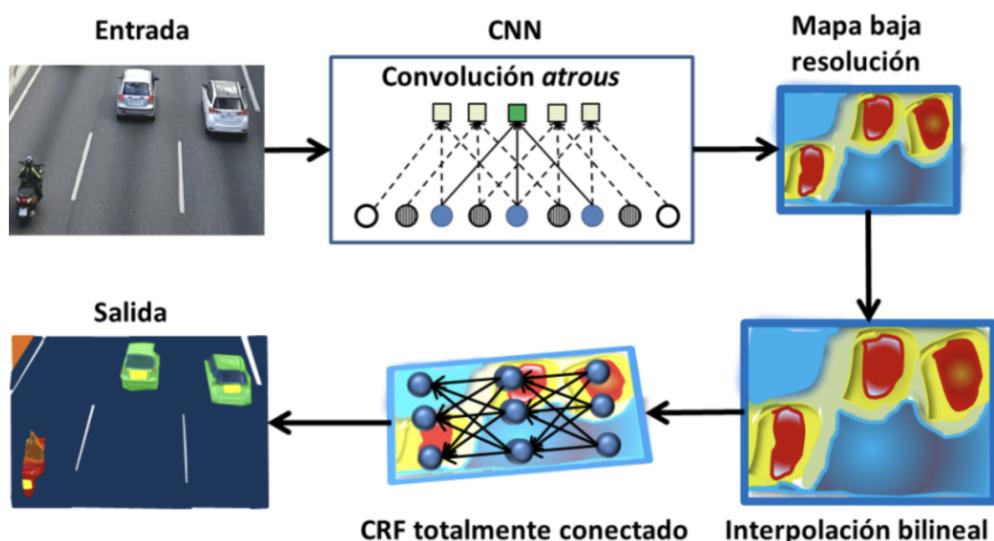


Figura 4.8: Esquema de funcionamiento de DeepLab, Fuente: Pajares et al. (2021)

Tomando como base la propuesta de DeepLab, Chen y col. (2017) crean DeepLab3 combinando módulos en cascada y en paralelo de convoluciones dilatadas (*atrous*). Para ello modifican una arquitectura ResNet, concretamente la ResNet-101, para mantener mapas de características de alta resolución en los bloques o niveles más profundos utilizando las mencionadas convoluciones *atrous*. También se utilizan otros modelos como ResNet-50.

Chen y col. proponen partir de un diseño de bloques ResNet, de forma que a partir del bloque 4 se aplican las convoluciones *atrous* en cascada, concretamente de tipo 3×3 como se muestra en la Figura 4.9. El objetivo es mantener mapas de características de alta resolución en los bloques más profundos.

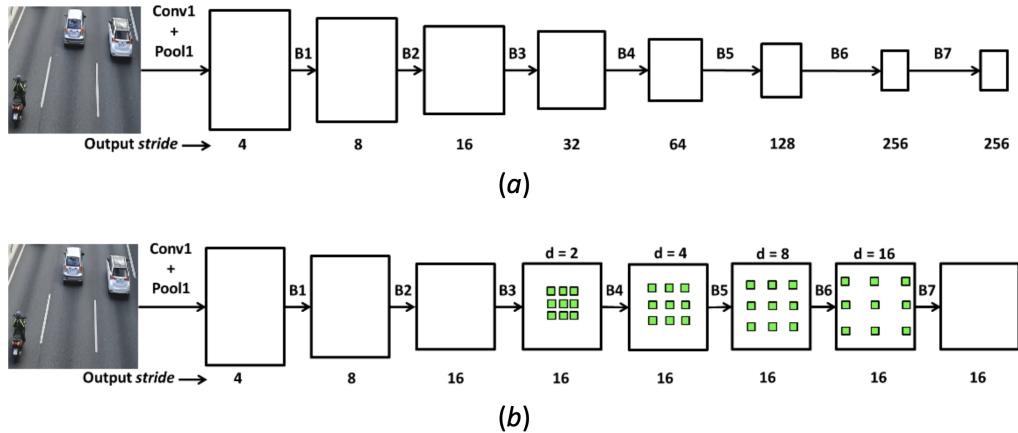


Figura 4.9: Propuesta de Chen y col., Fuente: Pajares et al. (2021)

A continuación, se muestra el esquema en paralelo con convoluciones *atrous*, que se agrupan en un módulo de tipo ASPP.

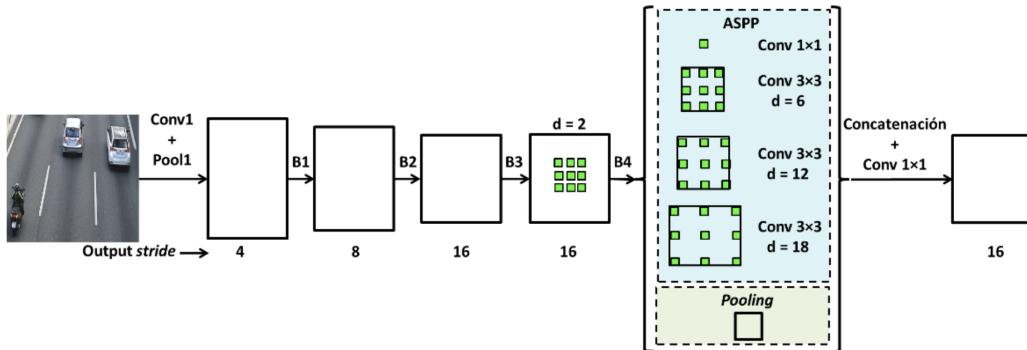


Figura 4.10: Esquema en paralelo con convoluciones *atrous*, Fuente: Pajares et al. (2021)

En este módulo se incluye una convolución de tamaño 1×1 y normalización por lotes. Las salidas así producidas se concatenan y procesan mediante otra convolución 1×1 para crear la salida final con la función *logits* (generalmente softmax) para cada píxel.

4.2.3. Modelo SegNet

SegNet fue desarrollada por Badrinarayanan y col. (2017), quienes formaron parte del Departamento de Ingeniería de la Universidad de Cambridge en ese momento, Tsang (2019).

Segnet utiliza un modelo *encoder-decoder* seguido de una capa final de clasificación por píxeles. En la parte del codificador se realizan convoluciones y agrupaciones de *pooling*. En total, sin contar las capas totalmente conectadas existen 13 capas convolucionales de VGG-16. Las capas de *pooling* se realizan con agrupaciones máximas de tamaño 2×2 . Por otro lado, en el decodificador, se realiza el muestreo y las convoluciones para que finalmente se clasifique utilizando softmax cada uno de los píxeles como se puede ver en la Figura

4.11.

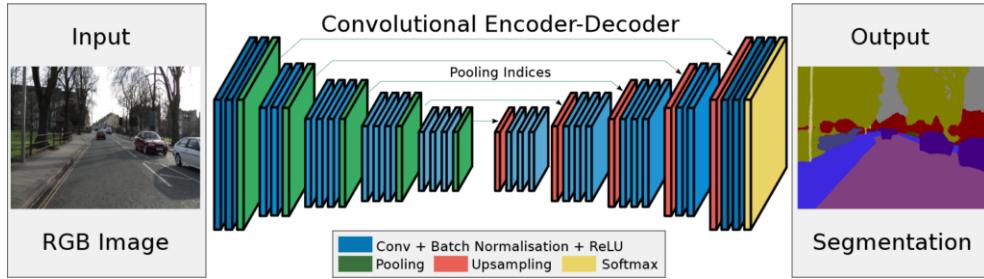


Figura 4.11: Modelo SegNet, Fuente: Tsang (2019)

4.2.4. Modelo U-Net

Este modelo fue propuesto por Ronneberger et al. (2015) para imágenes microscópicas biológicas y se basa en el esquema FCN de Long et al. (2015), así como el encoder-decoder. Se compone de dos partes de submuestreo y sobremuestreo, tal como se puede observar en la Figura 4.12. Una modificación importante en relación con las arquitecturas anteriores estriba en la parte de sobremuestreo o expansión, donde aparecen una serie de operaciones y mapas de características intermedias, que permiten la propagación de información contextual de la imagen hacia las capas de mayor resolución. Como consecuencia, la ruta expansiva es más o menos simétrica a la ruta de contracción y produce una arquitectura en forma de U. La red carece de capas completamente conectadas y solo usa la parte válida de cada convolución, es decir, solo la parte del mapa de segmentación que contiene los píxeles para los cuales el contexto completo está disponible en la imagen de entrada.

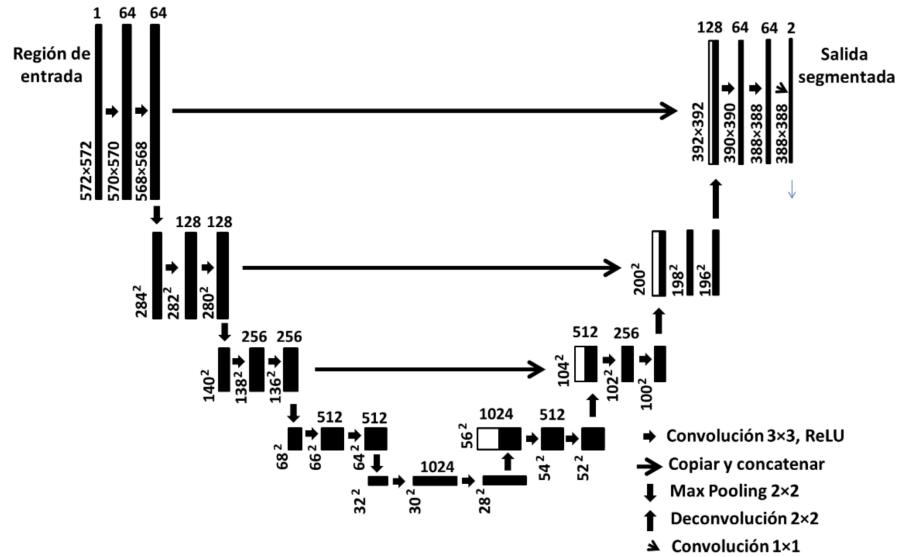


Figura 4.12: Modelo U-Net, Fuente: Pajares et al. (2021)

La ruta de concentración sigue la arquitectura típica de una red convolucional. Consiste en aplicar de forma repetida dos convoluciones de dimensión 3×3 sin padding y cada una

seguida de una ReLU y una operación de *pooling* de máximo con dimensión 2×2 y con desplazamiento (*stride*) de paso 2 con el fin de disminuir la resolución. En cada paso de reducción se duplica el número de mapas de características. Cada paso en la ruta expansiva consiste en un sobremuestreo del mapa de características seguido de una deconvolución de tipo 2×2 que reduce a la mitad el número de canales de características, una concatenación con el mapa de características recortado que se corresponde con el mapa de la ruta de contracción, los recuadros de color blanco en el ejemplo anterior, y dos convoluciones de tamaño 3×3 , cada una seguida de una ReLU. En la capa final, se utiliza una convolución de tamaño 1×1 para asignar cada vector de características de 64 componentes al número deseado de clases. En total, la red tiene 23 capas convolucionales. Se utiliza como método de optimización el gradiente estocástico descendente con momentum de valor 0.99 y la entropía cruzada como función para la optimización. Debido a las convoluciones sin *padding* la imagen de salida es de menor tamaño que la de entrada.

Capítulo 5

Herramientas de desarrollo, tecnologías y metodología de trabajo

A la hora de realizar un trabajo de estas características es imprescindible conocer las herramientas con las que desarrollarlo. En este capítulo se explican y enumeran las herramientas, aplicaciones y tecnologías utilizadas para el desarrollo del proyecto.

5.1. Herramientas de desarrollo

Durante el comienzo del proyecto se barajaron varias posibilidades a la hora de elegir las herramientas. A continuación, se explican de forma resumida las utilizadas en el presente proyecto:

- GitHub: Repositorio en línea muy utilizado para almacenar y administrar proyectos software, así como hacer un seguimiento de los cambios realizados en el código durante el desarrollo del proyecto (Lutkevich, 2023). Permite mediante el uso de ramas dividir el alcance del proyecto y llevar un control de lo que se va desarrollando. En este caso GitHub se ha utilizado para organizar todo el proyecto, desde esta memoria hasta el código.
- Visual Studio Code: Se trata de un editor de código desarrollado por Microsoft y muy popular debido a su gran cantidad de extensiones y a que es de código abierto. Se ha utilizado para poder escribir esta memoria.
- Google Meet: entorno de videollamadas en línea desarrollado por Google que permite comunicarse a distancia, compartir la pantalla, chat y grabación de reuniones. Utilizado en diversas comunicaciones con el director del trabajo.
- Matlab: Se trata de un plataforma de programación, y de un lenguaje de programación, desarrollado por TheMathWorks (2023) muy utilizado por ingenieros, científicos y matemáticos, que permite realizar de forma muy eficiente análisis de datos, desa-

rrollar algoritmos y crear modelos y aplicaciones (MathWorks, 2023a). Se ha utilizado para realizar los entrenamientos de las RN.

- Latex: Es un sistema de composición de texto muy utilizado en el ámbito académico y científico para crear documentos técnicos como artículos de investigación, tesis y libros. A diferencia de un procesador de textos convencional, que utilizan un enfoque visual, LaTeX se basa en comandos de texto y estructura lógica. Se ha utilizado para escribir esta memoria en su totalidad.
- ClickUp: Plataforma online de gestión de proyectos. Es una herramienta que permite a los equipos organizar, colaborar y realizar un seguimiento de sus actividades y proyectos de manera eficiente. Se utilizó para realizar toda la estimación de tareas, tiempos y planificación así como hacer un seguimiento del tiempo de cada una de las tareas para compararlo con el tiempo estimado y conocer retrasos.

5.2. Tecnologías

Como se ha comentado previamente, la elección de la tecnología es una parte crucial en cualquier proyecto. Cada tecnología ofrece unas ventajas sobre la otra y es imprescindible determinar cuál elegir para poder obtener el éxito. A la hora de trabajar con RN las posibilidades son casi infinitas. Se estudiaron cuáles eran los lenguajes de programación más utilizados y tras conocer sus ventajas e inconvenientes finalmente se decidió utilizar Matlab. Aunque lenguajes como C++, Java, etc. son muy utilizados, se decidió descartarlos por la complejidad que supondría realizar este proyecto en esos lenguajes. Se optó por investigar opciones más acordes, tales como Python y Matlab por su rapidez de implementación y su facilidad de uso. A continuación, se explican estas dos tecnologías y por qué se decidió finalmente utilizar Matlab.

5.2.1. Python

Python es un lenguaje de programación interpretado, es decir, que no se compila antes de ejecutarse sino que es un intérprete el que lo traduce (Camp, 2020); orientado a objetos, es decir que permite crear y definir clases y objetos, y dispone de herramientas como la herencia y el polimorfismo (Powell, 2021), es de alto nivel, esto es, con una sintaxis clara y no tan dependiente de la arquitectura del procesador, lo que hace que sea fácil de leer y escribir (Rouse, 2017); y con semántica dinámica, lo que significa que el tipo de una variable se determina en tiempo de ejecución y no es necesario que se declare de forma explícita (Ostrich, 2023). Todas estas características lo hacen muy atractivo para desarrollar aplicaciones de forma sencilla y rápida y con su gran cantidad de librerías, módulos y paquetes se pueden realizar acciones complejas en pocas líneas de código (Python, 2023).

Una de las librerías ampliamente utilizadas a la hora de realizar un modelo de DL con Python es la librería Keras. Se trata de una librería de aprendizaje profundo de alto nivel desarrollada por Google que permite implementar, modificar y entrenar numerosos modelos de RN ya existentes o crear los propios de forma muy fácil (Simplilearn, 2023).

El objetivo de esta biblioteca es aumentar la eficiencia y rapidez a la hora de crear

modelos de RN. Para ello, en lugar de funcionar como un framework independiente, Keras funciona como una Application Programming Interface (API) que permite acceder a varios frameworks de ML y desarrollarlos (Aizel et al., 2021).

Entre todas las ventajas que ofrece el lenguaje de programación Python y lo útil que es disponer de herramientas como Keras para acelerar el trabajo de entrenamiento de las RN, esta fue una opción muy determinante a la hora de elegir la tecnología aplicada a este proyecto.

5.2.2. Matlab

Como se ha comentado previamente Matlab no es solo un lenguaje de programación sino que se trata también de un entorno de programación muy potente para realizar operaciones complejas y manejo de matrices y vectores (MathWorks, 2023a). Se trata de un lenguaje de matriz/matriz de alto nivel con declaraciones de flujo de control, funciones, estructuras de datos, entrada/salida y características de programación orientada a objetos. Permite tanto “programar en lo pequeño” para crear rápidamente programas de “usar y tirar”, como “programar en lo grande” para crear programas de aplicación completos, grandes y complejos.

Matlab integra la computación, la visualización y la programación en un entorno fácil de usar donde los problemas y las soluciones se expresan en una notación matemática familiar. Algunos de los campos en los que Matlab es más utilizado son matemáticas y computación; desarrollo de algoritmos; modelado, simulación y creación de prototipos; análisis, exploración y visualización de datos; gráficos científicos y de ingeniería; etc. (CIMSS, 2023). En la Figura 5.1 se muestra un ejemplo de interfaz de Matlab.

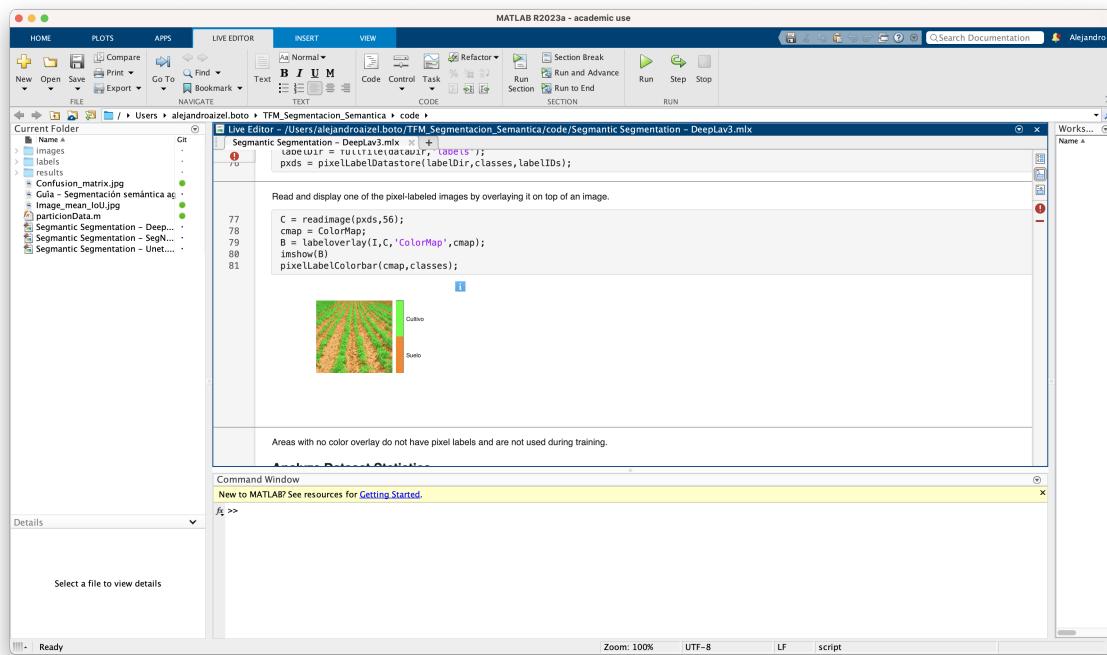


Figura 5.1: Interfaz de usuario de Matlab

Las tareas de DL se simplifican mucho con Matlab. Es uno de los mejores entornos para usar ML porque tiene las funciones y herramientas que permiten manejar grandes cantidades de datos, así como aplicaciones que simplifican su uso (MathWorks, 2023b).

Como ya se ha adelantado en la sección anterior, la tecnología que finalmente se eligió fue Matlab. Pese a la mayor familiaridad con el lenguaje Python, Matlab ofrece varias ventajas frente a Python que lo hacen una mejor opción para este proyecto. Además, también supuso un reto aprender un lenguaje de programación nuevo, ya que ofrece mucho valor en términos de ampliación de habilidades y capacidades como desarrollador para aplicarlos en futuros proyectos. Algunas de estas ventajas son las que se enumeran a continuación.

- Este proyecto no busca el desarrollo de una aplicación web, una api o un sistema con el que interactuar, por lo que en este aspecto Python no iba a ofrecer ninguna utilidad más que el entrenamiento de las RN.
- Matlab ofrece numerosos recursos y ejemplos oficiales que se pueden modificar y adaptar a un proyecto. En este caso se utilizaron tres ejemplos de RN ya escritos por Matlab que solamente se tuvieron que modificar y ajustar a los datos para poder realizar el entrenamiento. Esto ha permitido ahorrar tiempo que se pudo dedicar al estudio de los resultados y a realizar más pruebas.
- Como ya se ha dicho anteriormente, Matlab tiene una capacidad de cálculo matemático muy fuerte cosa que en Python es difícil de hacer. Python no tiene soporte de matrices, aunque se puede conseguir con la biblioteca NumPy, pero en Matlab existe por defecto, lo que ya es una ventaja de por si. Matlab es particularmente bueno en el procesamiento de señales e imágenes, cosa que Python no y su rendimiento es mucho peor. En general Matlab está diseñado para las tareas que se necesita realizar y es mucho más rápido, no solo en ejecución sino en desarrollo (Vadapalli, 2023).

Análisis de Resultados

En este capítulo se muestran y analizan los resultados obtenidos con los modelos de redes implementados, tanto desde el punto de vista del entrenamiento como de la clasificación semántica. Se va a analizar para cada uno la precisión global obtenida, así como su matriz de confusión que darán una idea general de la eficiencia del modelo.

Los diferentes experimentos se han llevado a cabo utilizando el dataset disponible de forma pública en Di Cicco et al. (2017), consistente en 1252 imágenes RGB agrícolas de dimensión 360×480 píxeles. Las imágenes de entrenamiento, validación y pruebas han sido convenientemente etiquetadas en las dos categorías mencionadas (suelo y cultivo).

6.1. Resultados obtenidos

Como se ha comentado en la introducción el conjunto de datos está formado por imágenes de agricultura ya etiquetadas en las que se distinguen dos clases diferentes: suelo y cultivo. Los modelos utilizados son los que se muestran en la Tabla 6.1. Cabe mencionar que el *encoder* está implementado por los modelos base que se indican.

Modelo de segmentación semántica	Modelo de red base o backbone
DeepLab3	resnet50 xception inceptionresnetv2
SegNet	vgg16 vgg19
Unet	deepStride: 2 deepStride: 4

Tabla 6.1: Modelos utilizados con cada una de sus arquitecturas

Por lo general, a la hora de hacer el entrenamiento se siguen los siguientes pasos:

1. Lectura de los datos de entrada sin etiquetar y su almacenamiento en la correspon-

diente variable.

2. Lectura de los datos etiquetados correspondientes a los datos de entrada y su almacenamiento en la correspondiente variable.
3. Creación de una variable en la que se indican las clases que se van a utilizar. En este caso suelo y cultivo.
4. Preparación de los datos de entrenamiento. Una vez se leen los datos de entrada etiquetados y sin etiquetar se dividen en los tres conjuntos mencionados en la teoría *train*, *val* y *test*. En este caso se ha hecho una diferenciación del 20 % para validación, 20 % para *test* y el restante 60 % para *train*.
5. Creación de la RN. Es ahora donde, utilizando las librerías proporcionadas por Matlab, se crea el modelo indicando la RN a utilizar.
6. Elección de los hiperparámetros. Como se ha indicado en la teoría, existen una serie de hiperparámetros como el *learning rate*, momentum, regularización, etc. que se tienen que ajustar.
7. Aumento de datos. Como se ha comentado en teoría un método para mejorar el entrenamiento es hacer *data augmentation*. En este paso se crean nuevos datos de entrenamiento utilizando los ya disponibles y sin necesidad de aumentar el número de ejemplos de entrenamiento. Las operaciones que se realizan en este proyecto son una reflexión aleatoria izquierda/derecha y una traslación aleatoria x o y de 10 píxeles o -10 píxeles. Estas operaciones se realizan tanto a las imágenes como a sus etiquetas.
8. Entrenamiento de la RN. Finalmente, una vez configurado el modelo se procede a entrenar la RN, obteniendo los pesos resultantes del ajuste, que definen el modelo entrenado.

Una vez entrenada la RN es necesario comprobar el comportamiento con nuevos datos. Para ello se utilizan las métricas apropiadas para determinar la precisión y la sensibilidad de cada uno de los modelos. Las métricas utilizadas son:

- Precisión global (*Global accuracy*): Es un porcentaje que nos indica el nivel de acierto global del modelo.
- Matriz de confusión normalizada. Se trata de una matriz de $N \times N$ siendo N el número de clases que tiene el modelo que nos permite ver el impacto de cada clase en el rendimiento global. Resume las predicciones realizadas por el modelo (eje X) en relación con las clases reales de los datos (eje Y).

6.1.1. DeepLab3

El modelo DeepLab utiliza como modelos base, en la fase de Encoder las siguientes redes pre-entrenadas: resnet18, resnet50, mobilenetv2 y xception. En este caso se ha decidido seleccionar tres de ellas para realizar el entrenamiento, estas son resnet50, xception, inceptionresnetv2 por su uso generalizado en estos modelos.

El *encoder* que crea esta RN ha sido entrenado con ImageNet (2009), y en este caso se han adaptado los pesos utilizando las imágenes de agricultura.

6.1.1.1. Entrenamiento con resnet50

A continuación, en la Figura 6.1, se muestra el proceso de entrenamiento de con la red resnet50.

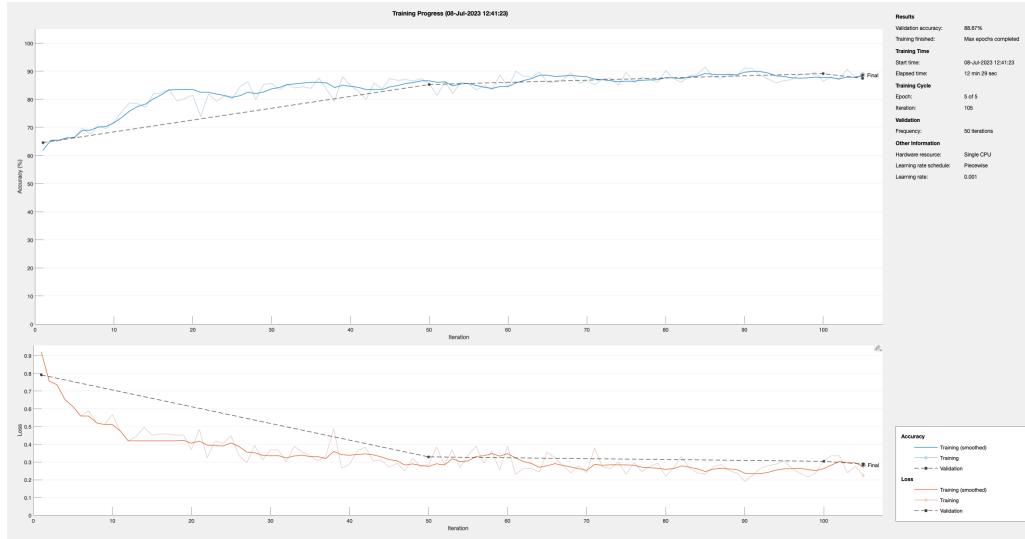


Figura 6.1: Proceso de entrenamiento de DeepLab 3 con resnet50

De este proceso se pueden sacar algunos datos que se reflejan en la tabla 6.2.

Tiempo de ejecución	12 minutos 29 segundos
Número de épocas	5
accuracy	88.93 %

Tabla 6.2: Datos de entrenamiento de DeepLab3 con resnet50

Se puede obtener la matriz de confusión normalizada en la Figura 6.2:

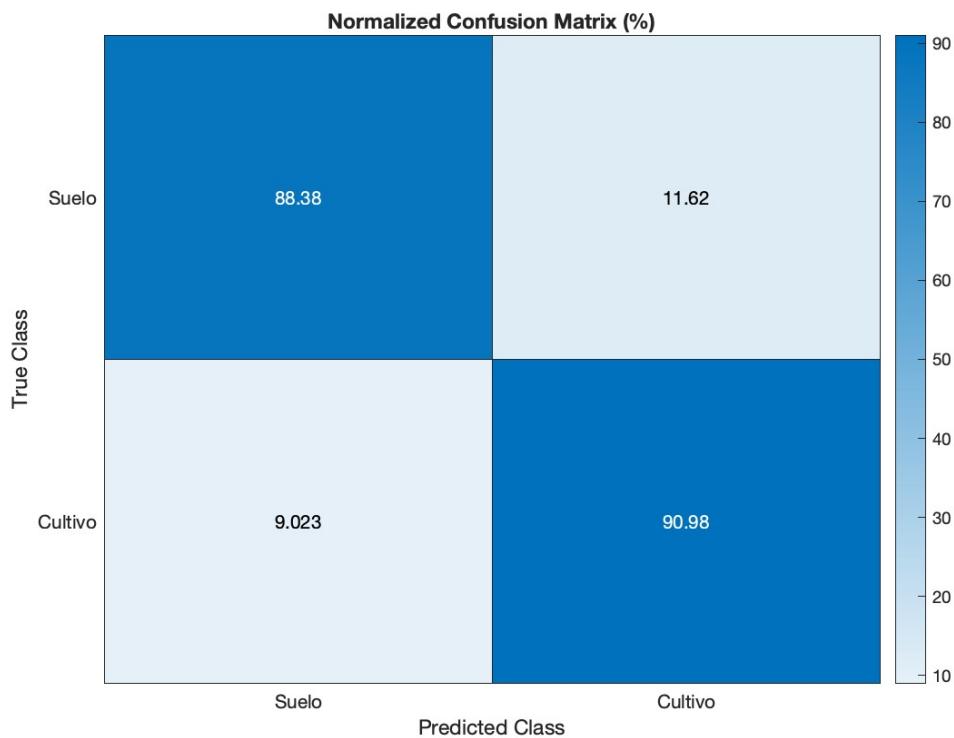


Figura 6.2: Matriz de confusión normalizada de DeepLab 3 con resnet50

Al examinar la matriz de confusión, se observa que la red tiene una buena capacidad para predecir la clase “suelo” (88.38 % de las veces correcta), aunque se cometan algunos errores en esta predicción (11.62 % de las veces se clasifica como “cultivo” cuando es “suelo”). Por otro lado, la red muestra una mayor precisión al predecir la clase “cultivo” (90.98 % de las veces correcta), pero nuevamente hay algunos errores en esta clasificación (9.02 % de las veces se clasifica como “suelo” cuando es “cultivo”).

En general, estos resultados indican que la red ha aprendido a distinguir entre las clases de “suelo” y “cultivo” con una precisión aceptable. Sin embargo, es importante tener en cuenta los errores de clasificación existentes.

6.1.1.2. Entrenamiento con xception

A continuación, en la Figura 6.3 se muestra el proceso de entrenamiento de con la red xception.

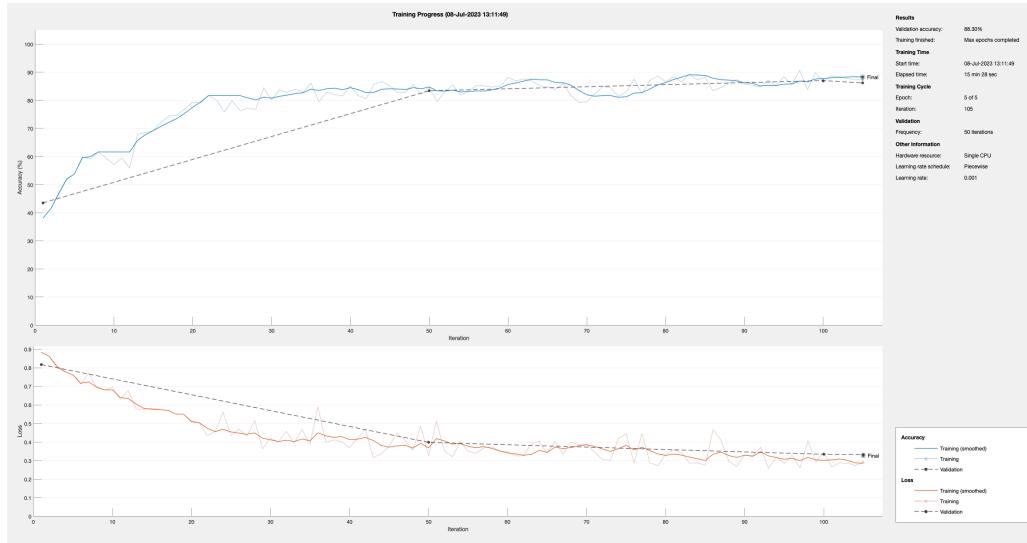


Figura 6.3: Proceso de entrenamiento de DeepLab 3 con xception

De la ejecución de este proceso se pueden extraer algunos resultados que se ven en la tabla 6.3.

Tiempo de ejecución	15 minutos 28 segundos
Número de épocas	5
accuracy	88.74 %

Tabla 6.3: Datos de entrenamiento de DeepLab3 con xception

La matriz de confusión normalizada es como sigue, Figura 6.4.

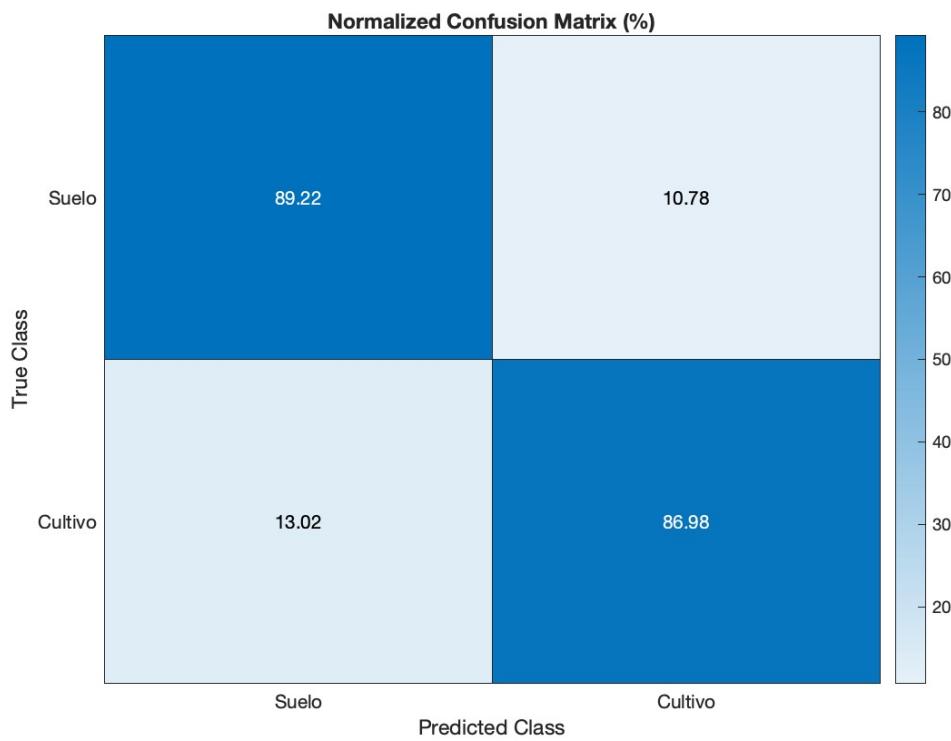


Figura 6.4: Matriz de confusión normalizada de DeepLab 3 con xception

Si examinamos la matriz de confusión, se obtiene que esta red también tiene una buena capacidad para predecir la clase “suelo” (89.22 % de las veces correcta), aunque se cometan algunos errores en esta predicción (10.78 % de las veces se clasifica como “cultivo” cuando es “suelo”). En este caso la red es capaz de predecir ligeramente mejor la clase suelo que la anterior, pero baja mucho la precisión del cultivo con respecto a la anterior (86.98 % de las veces correcta), habiendo errores (9.02 % de las veces se clasifica como “suelo” cuando es “cultivo”).

Teniendo en cuenta que esta red ha necesitado más tiempo en la ejecución que mejora solo ligeramente la predicción del suelo con respecto a la anterior y que la predicción del cultivo baja considerablemente, se puede afirmar que es una peor opción en este caso. No obstante, los resultados en términos generales son correctos y se puede decir que la red ha aprendido a distinguir entre las clases de “suelo” y “cultivo” con una precisión adecuada.

6.1.1.3. Entrenamiento con inceptionresnetv2

A continuación se muestran los resultados respecto del proceso de entrenamiento con la red base inceptionresnetv2 en la Figura 6.5

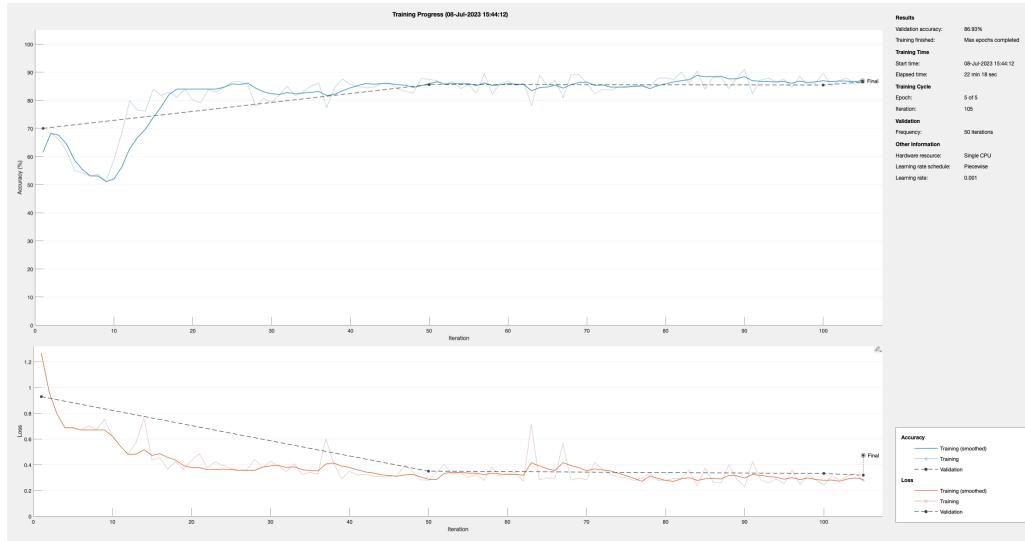


Figura 6.5: Proceso de entrenamiento de DeepLab 3 con inceptionresnetv2

De este proceso se pueden sacar algunos datos que se visualizan en la tabla 6.4.

Tiempo de ejecución	22 minutos 18 segundos
Número de épocas	5
accuracy	87.82 %

Tabla 6.4: Datos de entrenamiento de DeepLab3 con inceptionresnetv2

En la Figura 6.6 se observa la matriz de confusión normalizada:

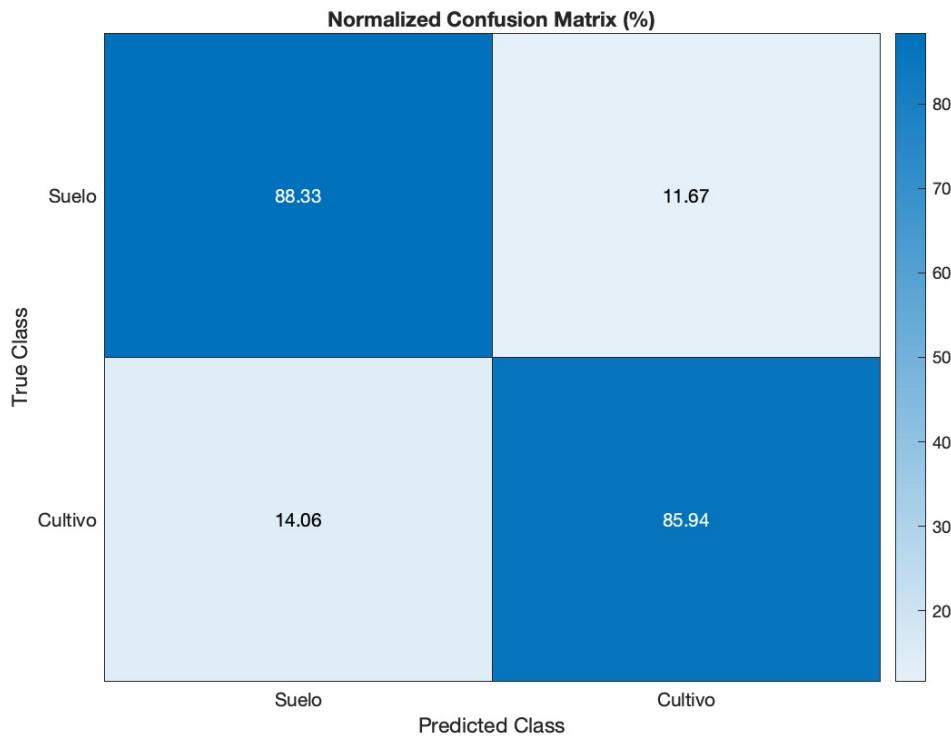


Figura 6.6: Matriz de confusión normalizada de DeepLab 3 con inceptionresnetv2

Al analizar la matriz de confusión, se ve que la red muestra una precisión ligeramente menor en comparación con las anteriores. Pese a que la predicción de la clase suelo se ha mantenido más o menos estable en 88.33 % (con un 11.67 % de clasificación errónea como “cultivo”), la precisión en la clasificación de la clase “cultivo” ha disminuido considerablemente a 85.94 % (con un 14.06 % de clasificación errónea como “suelo”).

En comparación con las redes anteriores, la red InceptionResNetV2 muestra una disminución general en la precisión global y en la capacidad de distinguir entre las clases “suelo” y “cultivo”. Aunque todas las redes logran una precisión razonable, el modelo con la red ResNet50 ha obtenido los mejores resultados y muestra una precisión global y rendimiento ligeramente superior al ser la red que mostró un mejor rendimiento en la clasificación de ambas clases y que tardó menos tiempo en ser entrenada.

6.1.2. U-Net

La diferencia de U-Net con respecto al anterior modelo DeeLabv3 está en que no utiliza modelos preentrenados, sino que se crea el modelo desde cero. Para ello existe el hiperparámetro *encoderDepth* para asignar un valor múltiplo del tamaño de imagen, pero a mayor número más costoso resulta ser el entrenamiento. En este caso se ha tenido que realizar el entrenamiento para *encoderDepth* = 2 y *encoderDepth* = 4, ya que con un valor de 8 se queda sin memoria.

6.1.2.1. DeepStride 2

A continuación, en la Figura 6.7, se muestra el proceso de entrenamiento de la red:

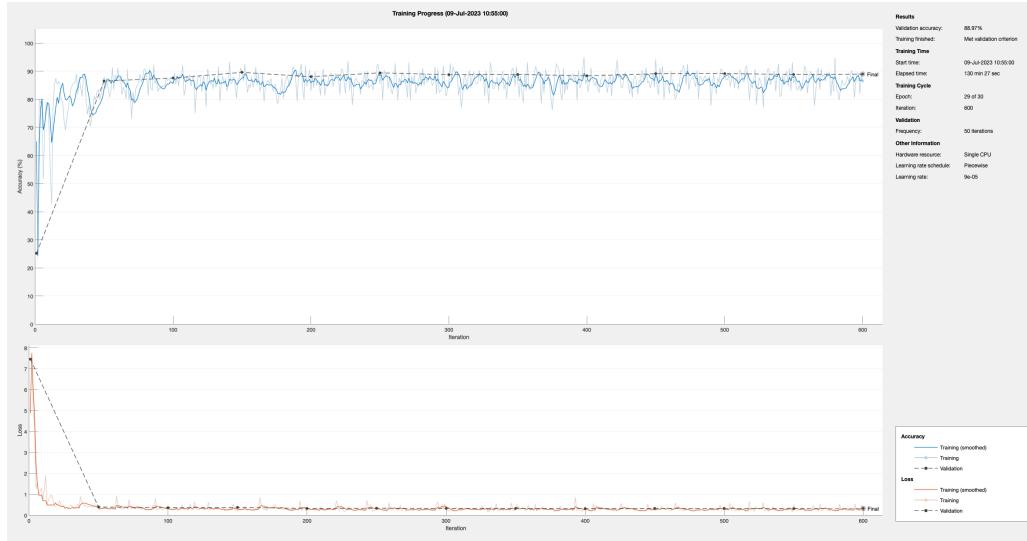


Figura 6.7: Proceso de entrenamiento de U-Net con DeepStride 2

De este proceso se pueden inferir algunos resultados que se muestran en la tabla 6.5:

Tiempo de ejecución	130 minutos 27 segundos
Número de épocas	30
accuracy	86.96 %

Tabla 6.5: Datos de entrenamiento de U-Net con DeepStride de 2

Si se representa la matriz de confusión normalizada, Figura 6.8, se obtiene:

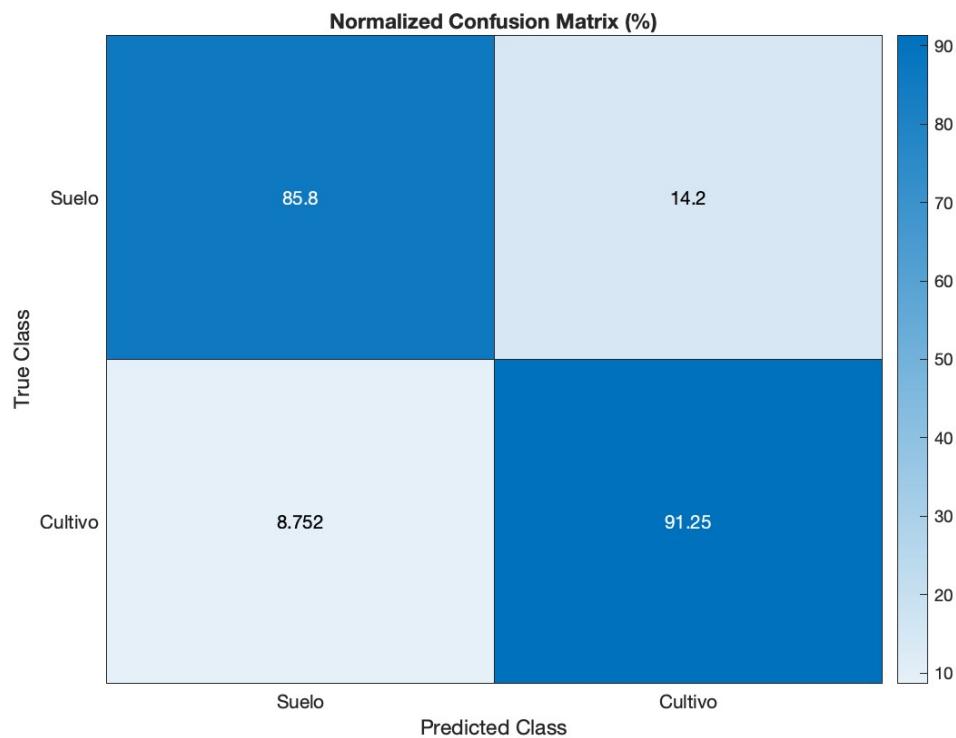


Figura 6.8: Matriz de confusión normalizada de U-Net con DeepStride de 2

Al analizar la matriz de confusión, se observa que la red muestra una precisión similar a las redes anteriores. La predicción de la clase “suelo” es del 85.8% (con un 14.2% de clasificación errónea como “cultivo”), mientras que la precisión en la clasificación de la clase “cultivo” es del 91.25% (con un 8.75% de clasificación errónea como “suelo”).

Como se puede observar, los valores no se alejan de lo que se ha conseguido hasta el momento. Sin embargo, el resultado es ligeramente peor que los anteriores con un tiempo de entrenamiento considerablemente mayor para el tipo de datos con el que se está entrenando.

6.1.2.2. DeepStride 4

A continuación se muestra el proceso de entrenamiento de la red en la Figura 6.9:

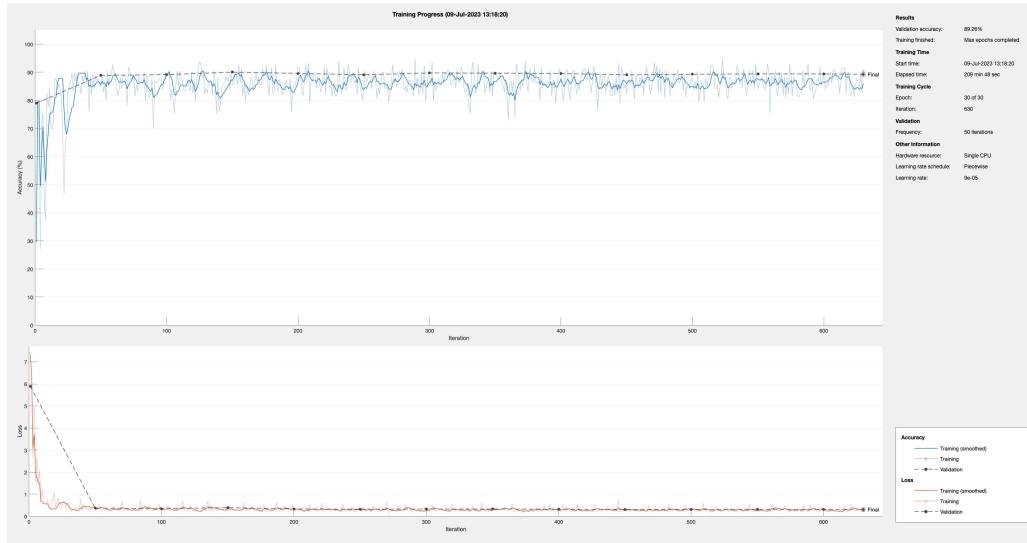


Figura 6.9: Proceso de entrenamiento de U-Net con DeepStride 4

De este proceso se pueden extraer sacar algunos datos en la tabla 6.6:

Tiempo de ejecución	209 minutos 48 segundos
Número de épocas	30
accuracy	87.29 %

Tabla 6.6: Datos de entrenamiento de U-Net con DeepStride de 2

Si se saca la matriz de confusión normalizada en la Figura 6.10, se obtiene:

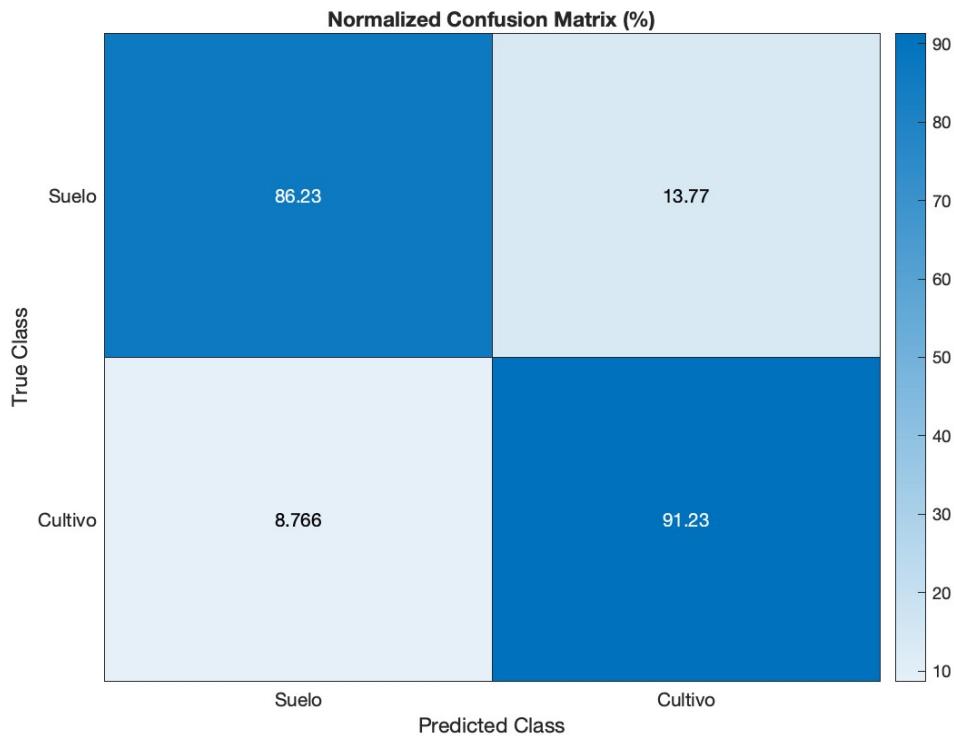


Figura 6.10: Matriz de confusión normalizada de U-net con DeepStride de 4

Al analizar la matriz de confusión, se puede ver que la red muestra una precisión casi identica a la anterior con *deep stride* 2. La predicción de la clase “suelo” es del 86.23 % (con un 13.77 % de clasificación errónea como “cultivo”), mientras que la precisión en la clasificación de la clase “cultivo” es del 91.23 % (con un 8.76 % de clasificación errónea como “suelo”).

Como se ve los valores son prácticamente idénticos a los anteriores por lo que no se presenta ninguna mejoría con un *stride* mayor y el tiempo tan alto de ejecución hace que carezca de interés a la hora de ser elegida candidata para realizar el entrenamiento final.

6.1.3. SegNet

Este es un modelo similar a U-Net, con la diferencia de que además del *encoder* pueden utilizarse los modelos pre-entrenados vgg16 y vgg19. Para realizar el estudio de este modelo se han realizado dos entrenamientos utilizando estas dos redes.

6.1.3.1. VGG16

A continuación se muestra el proceso de entrenamiento de la red en la Figura 6.11:

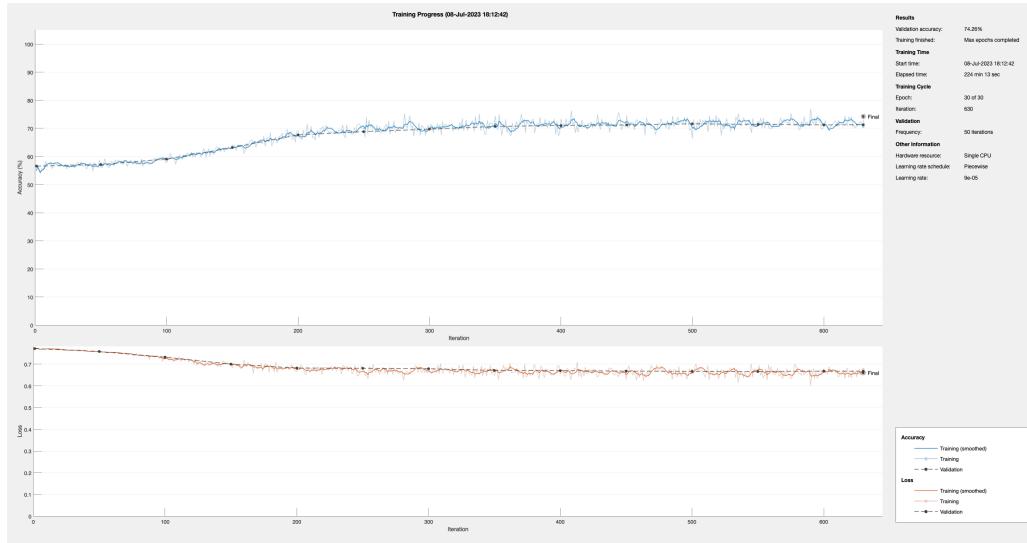


Figura 6.11: Proceso de entrenamiento de SegNet con VGG16

De este proceso se pueden inferir algunos datos que se plasman en la tabla 6.7:

Tiempo de ejecución	224 minutos 13 segundos
Número de épocas	30
accuracy	75.96 %

Tabla 6.7: Datos de entrenamiento de SegNet con VGG16

Si se muestra la matriz de confusión normalizada en la Figura 6.13, se puede deducir lo siguiente:

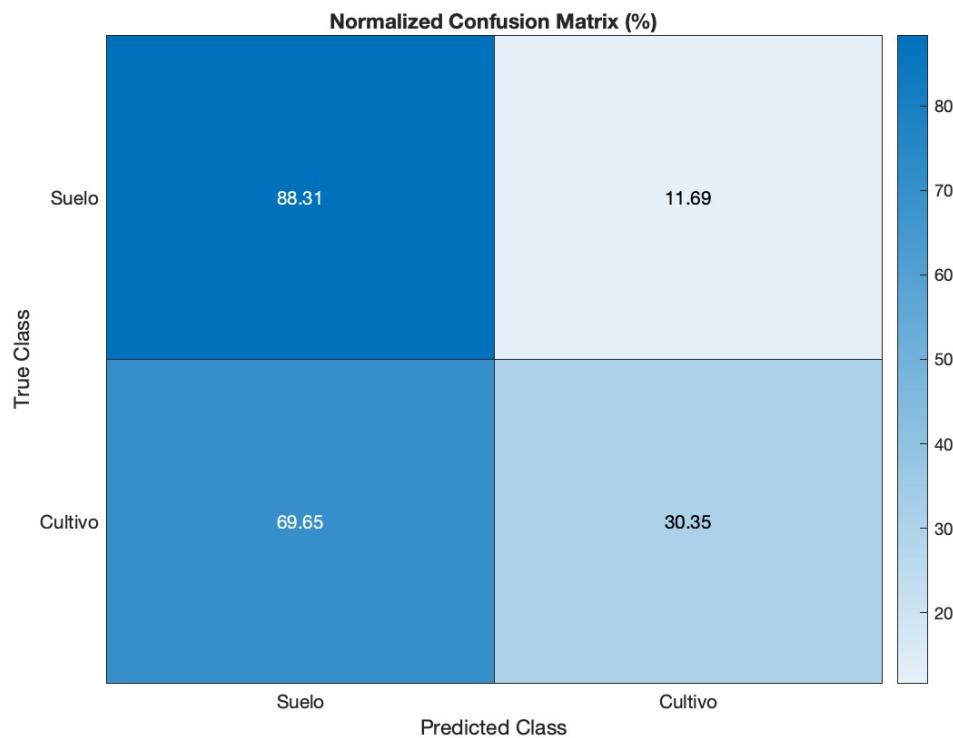


Figura 6.12: Matriz de confusión normalizada de SegNet con VGG16

Al analizar la matriz de confusión, se puede ver que la red muestra una precisión bastante inferior con los datos que han ido obteniendo con los otros modelos. La predicción de la clase “suelo” es del 88.31 % (con un 11.69 % de clasificación errónea como “cultivo”) lo cual no se aleja de lo que se ha ido obteniendo. No obstante, la precisión en la clasificación de la clase “cultivo” es del 30.35 % (con un 69.65 % de clasificación errónea como “suelo”) siendo estos valores inadmisibles para ser utilizados como clasificador.

Al tratarse de los peores valores obtenidos hasta el momento y considerando la cantidad de tiempo que ha tardado en ejecutarse, no merece la pena utilizar este modelo para la clasificación de los resultados disponibles.

6.1.3.2. VGG19

A continuación se muestra el proceso de entrenamiento de la red en la Figura 6.13 con este modelo:

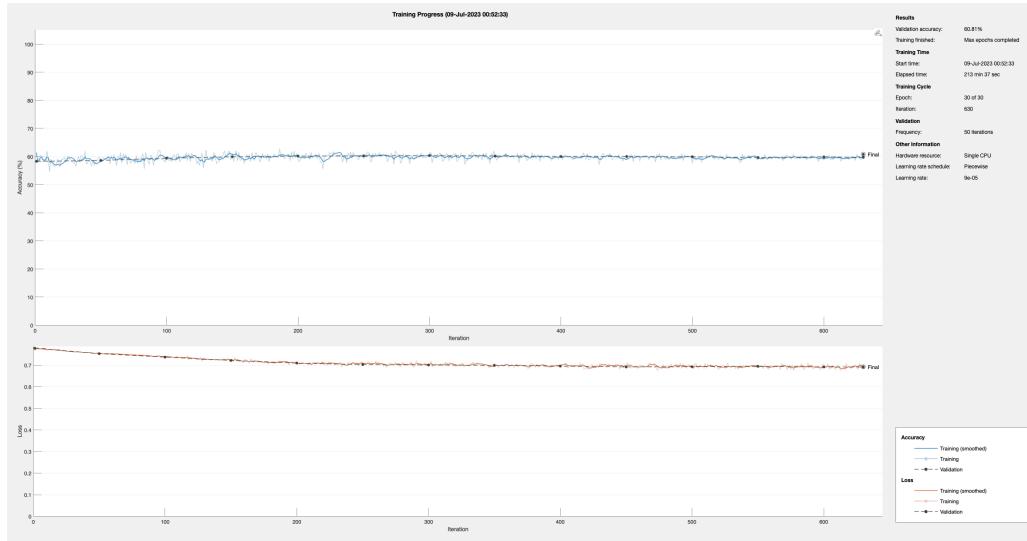


Figura 6.13: Proceso de entrenamiento de SegNet con VGG19

De este proceso se pueden sacar algunos datos en la tabla 6.8:

Tiempo de ejecución	213 minutos 37 segundos
Número de épocas	30
accuracy	60.83 %

Tabla 6.8: Datos de entrenamiento de SegNet con VGG19

Si se representa la matriz de confusión normalizada de la Figura 6.14 se obtiene:

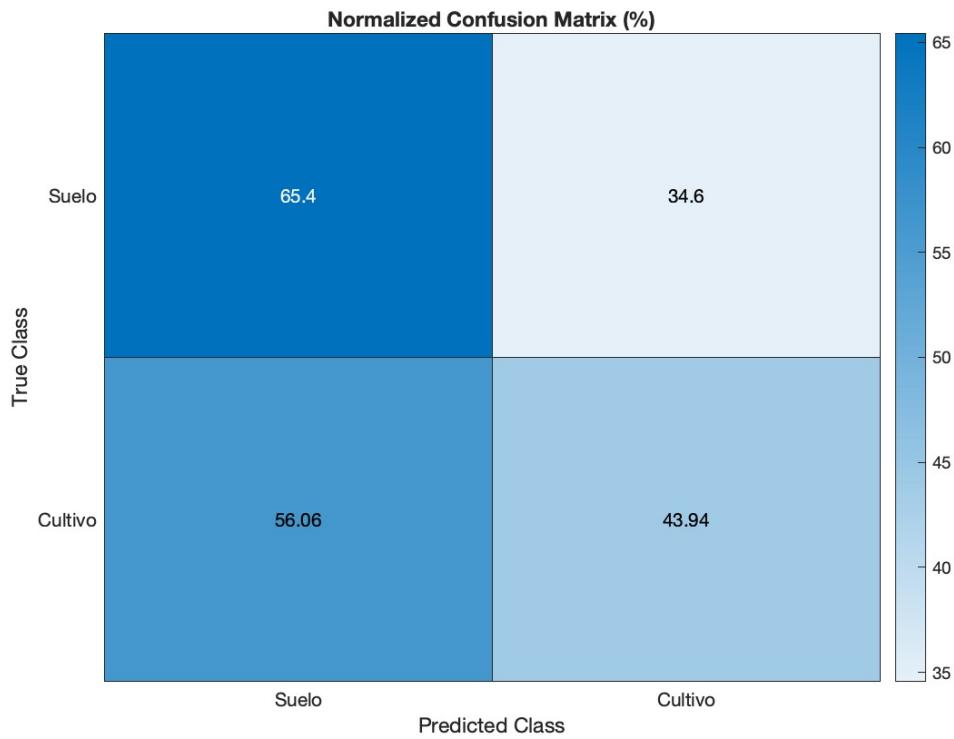


Figura 6.14: Matriz de confusión normalizada de SegNet con VGG19

Al analizar la matriz de confusión, se puede observar lo siguiente. La predicción de la clase “suelo” es de un 65.4 % (con un 34.6 % de clasificación errónea como “cultivo”) el peor dato obtenido hasta el momento. Por otro lado, la precisión en la clasificación de la clase “cultivo” es del 56.06 % (con un 43.94 % de clasificación errónea como “suelo”) algo mejor que el modelo entrenado con VGG16 pero muy alejado de unos valores óptimos.

Como se ve, se trata de los peores datos obtenidos, obteniendo una precisión global que supera ligeramente el 60 % de aciertos, un valor muy malo.

6.2. Análisis y comparación de resultados

A continuación se analizan de forma conjunta los resultados obtenidos. Por un lado, se tienen dos redes que han sido capaces de generalizar correctamente y derivar unos resultados bastante aceptables, concretamente las redes DeepLab3 y U-Net. Por otro lado, la red SegNet ha sido incapaz de obtener buenos resultados de predicción tanto para VGG16 como para VGG19, con unos tiempos de entrenamiento bastante altos para el tipo de datos utilizados. Por ello se ha decidido descartar esta arquitectura, centrando el estudio las otras dos.

Con las dos seleccionadas, se procede a elegir la más prometedora. En este caso se ha seleccionado DeepLab3 con la red base resnet50 debido a la gran eficiencia que muestra con pocas épocas y los valores tan altos que obtiene. Se han realizado reentrenamientos para in-

tentar mejorar sus resultados modificando los hiperparámetros anteriormente mencionados que se pasan a explicar a continuación.

6.2.1. Mejora de resultados

Como se ha indicado previamente, se selecciona la red DeepLab3 junto con resnet50 por su mejor comportamiento frente al resto para intentar reajustarla y ver si se consigue mejorar ligeramente la RN. Debido a que los datos de entrenamiento no son excesivamente elevados y que el porcentaje de aciertos es bastante alto, el margen de mejora no se espera que sea muy grande. A este respecto conviene recordar que el porcentaje de precisión del modelo anterior es de 88.93 %.

Lo primero que se ha realizado fueron varias pruebas modificando el método de optimización. Aunque no se ha mencionado anteriormente, el modelo fue entrenado con el método *Stochastic Gradient Descent* con *momentum* (sgdm). Para ver cómo se comportan estas redes con otros métodos de optimización se ha decidido reentrenarla utilizando rmsprop y adam obteniendo los siguientes resultados en la tabla 6.9:

Método de optimización	Porcentaje de precisión global (%)
sgdm	88.93
rmsprop	82.39
adam	88.95

Tabla 6.9: Porcentaje de precisión con cada método de optimización

Como se puede deducir, tanto sgdm como adam consiguen unos valores de precisión muy altos mientras que rmsprop no. Por conveniencia y por facilitar las pruebas se ha decidido seguir ajustando el modelo utilizando adam. Para ello se han realizado dos entrenamientos más modificando algunos hiperparámetros como el número de épocas, el *mini batch size* y el *learning rate* obteniendo los siguientes resultados en la tabla 6.10:

Método de optimización	Porcentaje de precisión global (%)
Prueba 1	89.49
Prueba 2	90.51

Tabla 6.10: Porcentaje de precisión de cada prueba

Como puede inferirse se ha conseguido mejorar en un 2 % la precisión del modelo con la segunda prueba que se ha realizado. Los valores de los hiperparámetros utilizados para conseguir esta precisión fueron los que aparecen en la tabla 6.11:

Hiperparámetro	Valor
LearnRateSchedule	piecewise
LearnRateDropPeriod	5
LearnRateDropFactor	0.2
InitialLearnRate	1e-4
L2Regularization	0.01
ValidationData	dsVal
MaxEpochs	10
MiniBatchSize	16
Shuffle	every-epoch
CheckpointPath	tempdir
VerboseFrequency	2
Plots	training-progress

Tabla 6.11: Valores de los hiperparámetros

Se obtiene también la siguiente matriz de confusión en la Figura 6.15, donde se han obtenido valores bastante superiores a los obtenidos anteriormente:

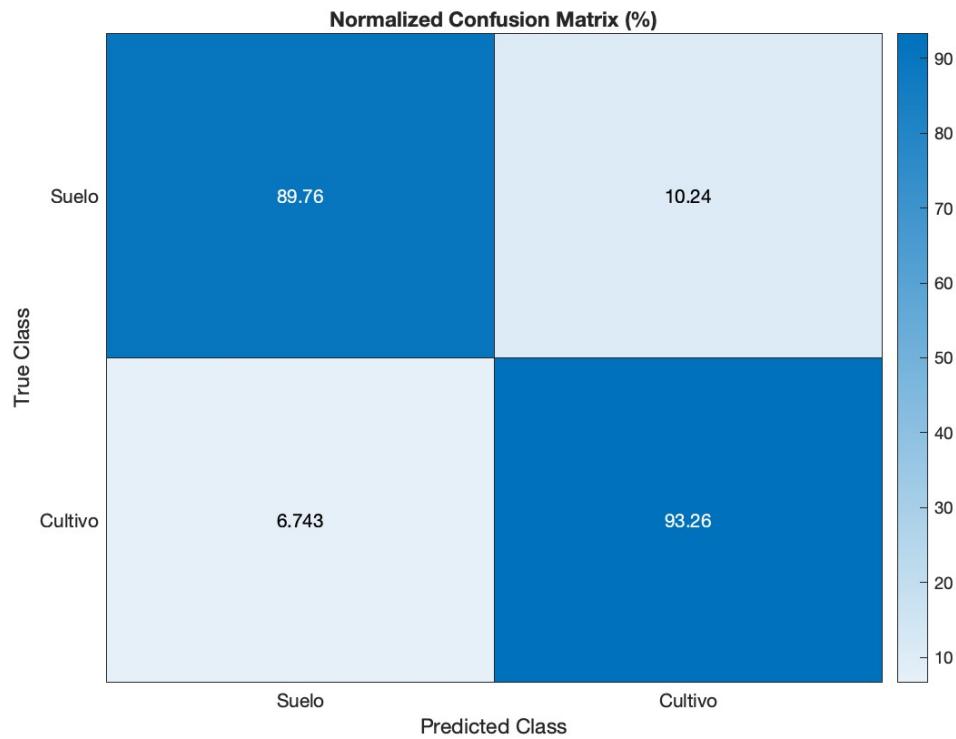


Figura 6.15: Matriz de confusión normalizada resultante del ajuste final

6.2.2. Resultado final

Habiendo identificado ya la red que proporciona los mejores resultados, el siguiente paso consiste en verificar su comportamiento durante la fase de inferencia o clasificación. A continuación, se presentan algunos ejemplos donde se observa cómo la red ha predicho los resultados correspondientes sobre una serie de imágenes.

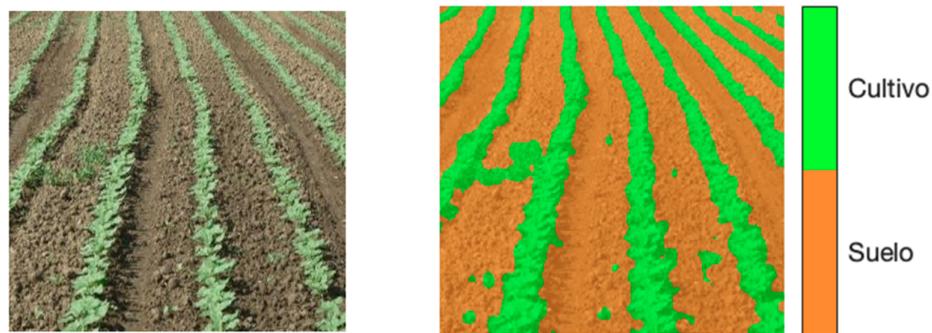


Figura 6.16: Ejemplo 1

Como se puede observar en la Figura 6.17 la predicción realizada es aceptable. Ha sabido diferenciar de forma muy precisa lo que es cultivo de la tierra, incluso en aquellas áreas donde cuesta distinguirlo por parte del experto humano. Si se extraen los porcentajes de acierto de este ejemplo de “test” se observa que para la clase suelo se ha conseguido un porcentaje de acierto del 88,7% y para la clase cultivo un porcentaje del 85,3%, lo que suponen unos valores de acierto muy altos.

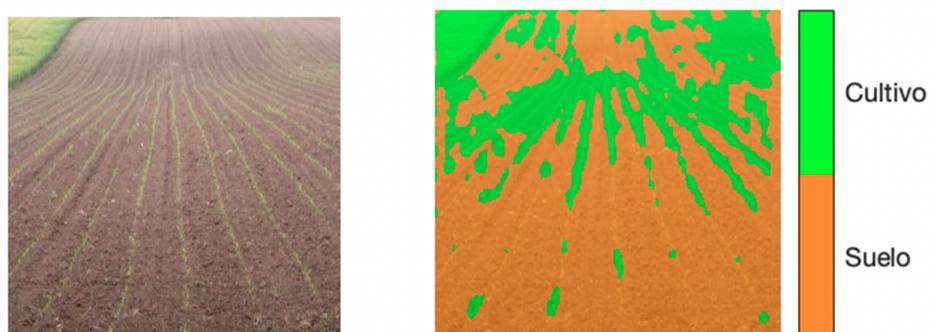


Figura 6.17: Ejemplo 2

Como se puede deducir en el ejemplo de la Figura 6.18, el modelo está lejos de obtener una precisión del 100% y se puede observar en este ejemplo, donde le cuesta detectar correctamente las finas líneas de cultivo. Para este ejemplo los resultados son algo peores como muestran sus porcentajes de acierto a cada una de las clase. Para la clase suelo se ha obtenido un 75,2% de acierto para la clase suelo pero un porcentaje del 20,2% para la clase cultivo, unos valores algo inferiores al anterior ejemplo. Pese a estos resultados, en general ha conseguido identificar los patrones objetivo.

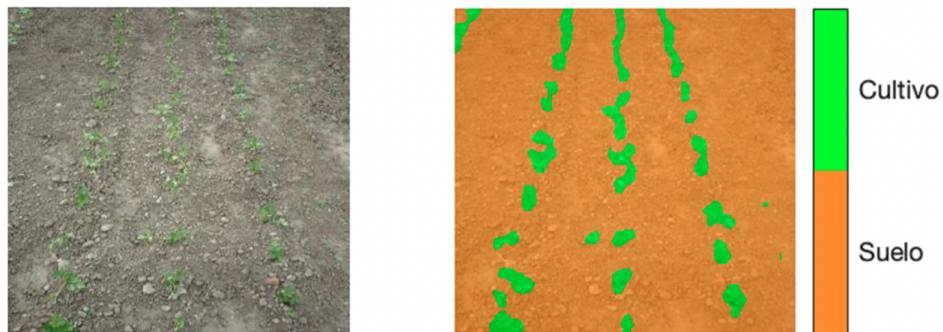


Figura 6.18: Ejemplo 3

Por último, en imágenes como la de la Figura 6.18 donde el cultivo no resulta ser del todo claro, estando prácticamente mimetizado con el suelo, el modelo ha conseguido identificar correctamente distinguiendo todas las zonas de cultivo. Los porcentajes de acierto con respecto a su imagen etiquetada es del 96,8 % de acierto para la clase suelo y 75,5 % de acierto para la clase cultivo, unos resultados aceptables considerando la imagen de prueba escogida.

Como se puede observar los resultados obtenidos son bastante prometedores. El modelo es capaz de identificar de forma correcta las zonas de cultivo y suelo, y con las mejoras realizadas al final se ha conseguido obtener un modelo que no solo es capaz de identificar las zonas, sino que lo hace de forma bastante precisa.

Conclusiones y Trabajo Futuro

En este trabajo se han estudiado tres métodos de Segmentación Semántica en imágenes bajo el paradigma DL (DeepLabv3, SegNet y U-Net). Se comenzó con el estudio de los modelos de redes básicos para pasar al estudio de los modelos CNN, siguiendo por las arquitecturas *encoder-decoder* que constituyen el fundamento de los modelos de segmentación semántica estudiados.

Se pudo determinar cómo las CNN son capaces de aprender diferentes características y predecir de forma más o menos exitosa las categorías en nuevas imágenes nunca antes vistas. En base a los resultados obtenidos, se puede concluir que la segmentación semántica utilizando RN es una técnica prometedora para el análisis de imágenes y que ha sido capaz de clasificar correctamente las imágenes agrícolas proporcionadas. Los modelos de RN evaluados, como DeepLab3 y U-Net, han demostrado su capacidad para distinguir entre el suelo y el cultivo con una precisión global significativa.

Además, durante la etapa de ajuste de la red se ha visto cómo modificando los hiperparámetros es posible hacer que la red se comporte de forma diferente mejorando sus resultados, eficiencia computacional o tiempo de entrenamiento, que son aspectos críticos a tener en cuenta, especialmente cuando se trabaja con conjuntos de datos más grandes o en entornos en tiempo real.

En términos generales se han cumplido los objetivos propuestos en este trabajo que consistían en el estudio y en la implementación de diferentes modelos de RN con el fin de clasificar a nivel de pixel las imágenes de agricultura.

No obstante, en base a los análisis realizados, es posible abrir algunas líneas de mejora que se enumeran a continuación:

- Como se comentó en la introducción hubo que recurrir a un *dataset* relativamente reducido para poder entrenarlo utilizando los recursos disponibles. Quizás utilizando un *dataset* mucho más grande y con mejor resolución se puedan obtener resultados más precisos.
- Por otro lado, este trabajo muestra un primer contacto con los modelos de segmen-

tación semántica en DL. En este sentido, se podrían investigar nuevos modelos y técnicas más actuales para ampliar el análisis comparativo de modelos.

- Utilizar los modelos de segmentación semántica en una aplicación real, tras la elección de un escenario de interés.

Chapter 7

Conclusions and Future Work

In this thesis, three Semantic Segmentation methods in images under the DL paradigm (DeepLabv3, SegNet, and U-Net) have been studied. The study began with the examination of basic network models, followed by an investigation into CNN models, and further exploration of the encoder-decoder architectures that form the foundation of the studied semantic segmentation models.

It could be determined how CNNs are capable of learning different features and predicting the categories of new, previously unseen images to varying degrees of success. Based on the results obtained, it can be concluded that semantic segmentation using RN is a promising technique for image analysis, and it has successfully classified the provided agricultural images. Evaluated RN models, such as DeepLab3 and U-Net, have demonstrated their ability to distinguish between soil and crops with significant overall accuracy.

Furthermore, during the network fine-tuning phase, it was observed that by modifying hyperparameters, it is possible to make the network behave differently, improving its results, computational efficiency, or training time, which are critical aspects to consider, especially when working with larger datasets or real-time environments.

In general terms, the objectives set in this work, which consisted of studying and implementing different RN models in order to classify agricultural images at pixel level, have been achieved.

However, based on the conducted analyses, it is possible to open some lines of improvement:

- As mentioned in the introduction, a relatively small dataset had to be used due to resource constraints. Perhaps, by using a much larger dataset with higher resolution, more accurate results could be obtained.
- Additionally, this work represents an initial exploration of semantic segmentation models in DL. In this regard, further research into newer models and techniques could expand the comparative analysis of models.

- The application of semantic segmentation models in a real-world scenario, following the selection of a specific area of interest.

Bibliografía

- AIZEL, A., PORTILLO, R. y SANZ, D. *tracción de preguntas a partir de imágenes para personas con problemas de memoria mediante técnicas de DeepLearning*, Disponible on-line <https://docta.ucm.es/entities/publication/5bdb37bb-c249-45bb-b8b9-f7f2074541d2>. 2021.
- ANDREW, N. *Machine Learning*, Disponible on-line <https://www.coursera.org/learn/machine-learning-course/home/week/1>. 2011.
- ANDREW, N. *Convolutional Neural Networks*, Disponible on-line <https://www.coursera.org/learn/convolutional-neural-networks>. 2017.
- ANYOHA, R. *The History of Artificial Intelligence*, Disponible on-line <https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/>. 2017.
- ASNAEBSA. *Cross-sectional T1-weighted MRI of a healthy human brain produced at a ultra high-field MR of 7 Tesla*, Disponible on-line, https://commons.wikimedia.org/wiki/File:MRI_of_Human_Brain.jpg. 2022.
- CAMP, F. C. *Interpreted vs Compiled Programming Languages: What's the Difference?*, Disponible on-line <https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/>. 2020.
- CAMPI, M. *The Harvest*, Disponible on-line, [https://commons.wikimedia.org/wiki/File:The_Harvest_\(182814841\).jpeg](https://commons.wikimedia.org/wiki/File:The_Harvest_(182814841).jpeg). 2016.
- CHEN, L., PAPANDREOU, G., SCHROFF, F. y ADAM, H. *Rethinking Atrous Convolution for Semantic Image Segmentation..* arXiv:1706.05587 [cs.CV], 2017.
- CIMSS. *What Is MATLAB?*, Disponible on-line <https://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/what-is-matlab.html>. 2023.
- DÁRCHIMBAUD, E. *Training, Validation and Test Sets: How To Split Machine Learning Data*, Disponible on-line <https://kili-technology.com/training-data/training-validation-and-test-sets-how-to-split-machine-learning-data>. 2023.
- OF DATA SCIENCE, H. *AI Winter: The Highs and Lows of Artificial Intelligence*, Disponible on-line <https://www.historyofdatascience.com/ai-winter-the-highs-and-lows-of-artificial-intelligence/>. 2021.

- DELUA, J. *Supervised vs. Unsupervised Learning: What's the Difference?*, Disponible on-line <https://www.ibm.com/blog/supervised-vs-unsupervised-learning/>. 2021.
- DI CICCO, M., POTENA, C., GRISETTI, G. y PRETTO, A. *Automatic model based dataset generation for fast and accurate crop and weeds detection*, in: *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 5188-5195. doi:10.1109/IROS.2017.8206408, 2017.
- FACTS, B. *Github*, Disponible on-line <https://www.brainfacts.org/core-concepts/how-neurons-communicate>. 2018.
- GOLD, D. *Cars Urban Street Free Stock Photo*, Disponible on-line, <https://negativespace.co/cars-urban-street/>. 2023.
- HODAN, G. *Cat - Blanco y Negro*, Disponible on-line <https://www.publicdomaininpictures.net/es/view-image.php?image=32743&picture=cat-blanco-y-negro>. 2022.
- IBM. *What is deep learning?*, Disponible on-line <https://www.ibm.com/topics/deep-learning>. 2023a.
- IBM. *¿Qué es la visión artificial?*, Disponible on-line <https://www.ibm.com/es-es/topics/computer-vision>. 2023b.
- IMAGENET. *ImageNet*, Disponible on-line, <https://www.image-net.org/>. 2009.
- JACCARD, P. *Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines*. *Bulletin de la Société Vaudoise des Sciences Naturelles* 37. 241-272, 1901.
- JORDAN, J. *An overview of semantic image segmentation*, Disponible on-line <https://www.jeremyjordan.me/semantic-segmentation/>. 2018.
- KULLBACK, S. y LEIBLER, R. *On Information and Sufficiency*. *Annals of Mathematical Statistics* 22. 79-86, 1951.
- LONG, J., SHELHAMER, E. y DARRELL, T. *Fully Convolutional Networks for Semantic segmentation*. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR'15)*. Boston, MA, USA.. 3431-3440, 2015.
- LUTKEVICH, B. *GitHub*, Disponible on-line <https://www.techtarget.com/searchitoperations/definition/GitHub>. 2023.
- MARR, D. *Vision*. W.H.Freeman and Co., San Francisco, CA. 1982.
- MATHWORKS. *What Is MATLAB?*, Disponible on-line <https://es.mathworks.com/discovery/what-is-matlab.html>. 2023a.
- MATHWORKS. *¿Qué es Machine Learning?*, Disponible on-line <https://es.mathworks.com/discovery/machine-learning.html>. 2023b.
- MATLAB. *The Mathworks*, Disponible on-line <https://es.mathworks.com/>. 2023.
- MCCULLOCH, W. y PITTS, W. *A Logical Calculus of the Ideas Immanent in nervous Activity*. *Bulletin of Mathematical Biophysics* 5. 115-133, 1943.

- MILLETARI, F., NAVAB., N. y AHMADI, S. *V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.* Proc. Fourth International Conference on 3D Vision (3DV). Stanfورد, CA. 565-571, 2016.
- MONTAGUD, N. *¿Cómo funcionan las neuronas?, Disponible on-line* <https://psicologiyamente.com/neurociencias/como-funcionan-neuronas>. 2020.
- NIH. *Neurona,* Disponible on-line <https://www.cancer.gov/espanol/publicaciones/diccionarios/diccionario-cancer/def/neurona>. 2023.
- OSTRICH, A. *Python: An Introduction,* Disponible on-line <https://generalassembly.com/blog/what-is-python-a-beginners-guide>. 2023.
- PAJARES, G. y DE LA CRUZ, J. *Visión por Computador: Imágenes Digitales y Aplicaciones.* RA-MA, Madrid. 2007.
- PAJARES, G., P.J., H. y BESADA, E. *Aprendizaje Profundo.* RC-Libros, Madrid.. 2021.
- POWELL, R. *Functional vs object-oriented programming,* Disponible on-line <https://circleci.com/blog/functional-vs-object-oriented-programming/>. 2021.
- PYTHON. *What is Python? Executive Summary,* Disponible on-line <https://www.python.org/doc/essays/blurb/>. 2023.
- RONNEBERGER, O., FISCHER, P. y BROX, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation..* arXiv:1505.04597 [cs.CV], 2015.
- ROUSE, M. *High-Level Language,* Disponible on-line <https://www.techopedia.com/definition/3925/high-level-language-hll>. 2017.
- RUIZ, L. *Neurona: qué es y cuáles son sus partes,* Disponible on-line <https://www.psyciencia.com/neurona-que-es-y-cuales-son-sus-partes/>. 2021.
- SCIENTEST, D. *Inteligencia artificial : definición, historia, usos, peligros,* Disponible on-line <https://datascientest.com/es/inteligencia-artificial-definicion>. 2022.
- SCOTTO, S. *Inteligencia Artificial. ¿Son las máquinas capaces de pensar?,* Disponible on-line <https://sites.miis.edu/ondaglobal/2018/09/13/los-animales-piensan-los-humanos-piensan-son-las-maquinas-capaces-de-pensar-tambien>. 2018.
- SHIN, T. *A Beginner-Friendly Explanation of How Neural Networks Work,* Disponible on-line <https://towardsdatascience.com/a-beginner-friendly-explanation-of-how-neural-networks-work-55064db60df4>. 2020.
- SIMPLILEARN. *What Is Keras: The Best Introductory Guide To Keras,* Disponible on-line <https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras>. 2023.
- THEMATHWORKS. *The Mathworks,* Disponible on-line <https://es.mathworks.com/>. 2023.
- TSANG, S. *Review: SegNet (Semantic Segmentation),* Disponible on-line <https://towardsdatascience.com/review-segnet-semantic-segmentation-e66f2e30fb96>. 2019.
- TURING, A. *Computing Machinery and Intelligence.* Mind 49: 433-460, 1950.

VADAPALLI, P. *MATLAB Vs Python: Difference Between Matlab and Python*, Disponible on-line <https://www.upgrad.com/blog/matlab-vs-python/>. 2023.

WITHEY, D. y KOLES, Z. J. A review of medical image segmentation: Methods and available software, disponible on-line <https://api.semanticscholar.org/corpusid:565586>. 2008.

Glosario

AI Artificial Intelligence. 7, 8, 11

API Application Programming Interface. 60

CT Computerized Tomography. 9

CV Computer Vision. 8, 10, 11

DL Deep Learning. xi, xiii, 5, 6, 13–15, 17, 27, 47, 60, 61, 83–85

IA Inteligencia Artificial. 1, 2, 5, 14, 15

ML Machine Learning. 14, 15, 60, 61

MRI Magnetic Resonance Imaging. 9

RM Resonancia Magnética. 3

RN Red Neuronal. xv, xvi, 2, 6, 14–19, 21, 23, 25–29, 31–35, 43, 47–50, 52–54, 56, 58–62, 64, 65, 79, 83, 85

RNC Red Neuronal Convolutional. xi, xx, 6, 33, 34, 44, 49, 53

TC Tomografía Computarizada. 3

UCM Universidad Complutense de Madrid. 5, 11

VA Visión Artificial. 1, 52

VC Visión por Computador. 1, 2, 5, 6, 13, 35, 47

VS Vectores de Soporte. 14

Apéndice A

Código y guía de instalación

Para poder acceder al código que se ha utilizado en este proyecto se puede acceder al repositorio online <https://github.com/alejandroaizel/TFMcode> donde se podrá descargar en su totalidad el código utilizado.

Este repositorio cuenta con 3 archivos de MatLab con cada uno de los modelos utilizados (DeepLabv3, SegNet y Unet). Además se proporcionan las imágenes y etiquetas utilizadas para poder realizar el entrenamiento. A continuación se explican los pasos para poder ejecutar el código.

Lo primero que se debe hacer es instalar el programa Matlab o acceder vía online con una cuenta habilitada para ello. Una vez dentro de la interfaz de Matlab se podrán importar todos los archivos y carpetas del repositorio para poder comenzar con su ejecución. En caso de realizar la ejecución de forma local se deberán de instalar los correspondiente módulos para poder cargar los modelos y realizar el entrenamiento. Estos módulos se conocerán al realizar la ejecución del código ya que el propio programa mostrará un error indicand el módulo necesario a instalar. Si se ha decidido por la herramienta online no es necesario realizar ningún paso adicional.

Antes de poder ejecutar el código se deberán modificar las rutas de las carpetas del dataset de imágenes. Para ello simplemente se deberán modificar las correspondientes rutas en las partes que aparecen en la Figura A.1 y Figura A.2.

```

46 % imagesCamVid = 'C:\Trabajo\Clara\CamVid\RealData\CamVidImages';
47 % imgDir = fullfile(imagesCamVid, '701_StillsRaw_full');
48 % imds = imageDatastore(imgDir);
49
50 dataDir = '/MATLAB Drive/TFMcode';
51 imDir = fullfile(dataDir,'images');
52 imds = imageDatastore(imDir);

Display one of the images.

53 I = readimage(imds,56);
54 I = histeq(I);
55 imshow(I)

```

Figura A.1: Ruta de acceso a las imágenes

```

74 Use the classes and label IDs to create the pixelLabelDatastore.
75
76 labels = '/MATLAB Drive/TFMcode';
labelDir = fullfile(dataDir,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);

```

Figura A.2: Ruta de acceso a las etiquetas

Posteriormente se deberá elegir el backbone que se utilizará. Para ello se deberá descomentar las correspondientes líneas que se encuentran en el bloque de la Figura A.3 y Figura A.4.

```

23 shows you how to train a semantic segmentation network using transfer learning.
24
25 Train a Semantic Segmentation Network
26 This example trains a DeepLab v3+ network with weights initialized from a pre-trained ResNet-18 network.
27 ResNet-18 is an efficient network that is well suited for applications with limited processing power.
28 Depending on application requirements. For more details, see Pretrained Deep Neural Networks.
29 To get a pretrained Resnet-18, install Deep Learning Toolbox™ Model for Resnet-18 Network. After installation is complete, run the following code to verify that the installation is correct.
30
31 %resnet18();
32 %resnet50();
33 %mobilenetv2();
34 %ception();
35 %inceptionresnetv2();

Download CamVid Dataset

```

Figura A.3: Elección de la red base

```

95 Create the Network
96 Use the deepLabv3plusLayers function to create a DeepLab v3+ network based on ResNet-18. Choosing the best network for your application depends on your specific needs. You can use ResNet-18, ResNet-50 or MobileNet v2, or you can try other semantic segmentation network architectures such as SegNet, fully convolutional networks, or U-Net.
97
98 % Specify the network image size. This is typically the same as the training image sizes.
99 imageSize = [304 320 3];
100
101 % Specify the number of classes.
102 numClasses = numel(classes);
103
104 % Create DeepLab v3+.
105 lgraph = deepLabv3plusLayers(imageSize, numClasses, "resnet18");
106 %lgraph = deepLabv3plusLayers(imageSize, numClasses, "resnet50");
107 %lgraph = deepLabv3plusLayers(imageSize, numClasses, "mobilenetv2");
108 %lgraph = deepLabv3plusLayers(imageSize, numClasses, "xception");
109 %lgraph = deepLabv3plusLayers(imageSize, numClasses, "inceptionresnetv2");
110
111 plot(lgraph)

```

Figura A.4: Creación de la red

Antes de comenzar con el entrenamiento se podrán modificar valores como los hiperparámetros como se puede ver en la Figura A.5.



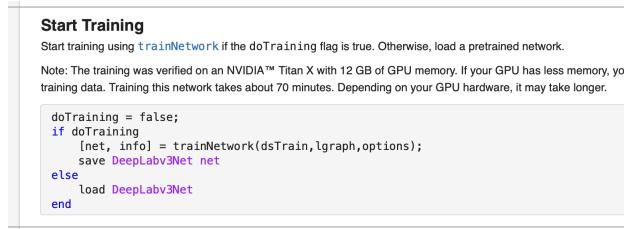
```

114 % Define validation data.
115 %dsVal = combine(imdsVal,pxdsVal);
116 dsVal = pixelLabelImage datastore(imdsVal,pxdsVal);
117
118 % Define training options.
119 options = trainingOptions('adam',...
120     'LearnRateSchedule', 'piecewise',...
121     'LearnRateDropPeriod', 5, ...
122     'LearnRateDropFactor', 0.2, ...
123     'InitialLearnRate', 1e-4, ...
124     'L2Regularization', 0.01, ...
125     'ValidationData', dsVal, ...
126     'MaxEpochs', 20, ...
127     'MiniBatchSize', 32, ...
128     'Shuffle', 'every-epoch', ...
129     'CheckpointPath', tempdir, ...
130     'VerboseFrequency', 2, ...
131     'Plots', 'training-progress');
132 %'VaabelsIdationPatience', 4);

```

Figura A.5: Bloque donde se establecen los hiperparámetros

Por último, solamente quedará dar al botón de *play* para que comience la ejecución de la red neuronal. El entrenamiento se realiza en la parte que se puede ver en la Figura A.6. Una vez finalizado el entrenamiento se guardará un fichero con los pesos ya ajustados que se podrá cargar de nuevo modificando la variable *doTraining* para evitar que se reentrene de nuevo.



```

Start Training
Start training using trainNetwork if the doTraining flag is true. Otherwise, load a pretrained network.

Note: The training was verified on an NVIDIA™ Titan X with 12 GB of GPU memory. If your GPU has less memory, you
training data. Training this network takes about 70 minutes. Depending on your GPU hardware, it may take longer.

doTraining = false;
if doTraining
    [net, info] = trainNetwork(dsTrain,lgraph,options);
    save DeepLabv3Net net
else
    load DeepLabv3Net
end

```

Figura A.6: Bloque de entrenamiento del modelo

De esta forma se habrá entrenado correctamente la red y se podrán hacer las pertinentes pruebas con el fin de analizar su eficiencia con las imágenes del conjunto de pruebas. Estas pruebas son las que aparecen justo a continuación del bloque de entrenamiento.

