TRABAJO FIN DE GRADO

GRADO EN ESTADÍSTICA

Estudio e implementación de redes neuronales convolucionales para la segmentación de imágenes

Autora
Julia García Vega

Tutores
Prof. Dra. Ana Belén Nieto Librero
Prof. Dr. José Luis Vicente Villardón

Facultad de ciencias

—

Salamanca, Julio de 2024

ESTUDIO E IMPLEMENTACIÓN DE REDES NEURONALES CONVOLUCIONALES PARA LA SEGMENTACIÓN DE IMÁGENES

Trabajo de Fin de Grado

Grado en Estadística

Curso académico 2023-2024

GARCIA VEGA JULIA - 70961775J

Firmado digitalmente por GARCIA VEGA JULIA - 70961775J
Fecha: 2024.06.30 21:03:04 +02'00'

Autora: Julia García Vega

Tutora: Ana Belén Nieto Librero

Tutor: José Luis Vicente Villardón

Certificado de los tutores TFG

D. José Luis Vicente Villardón, profesor del Departamento de Estadística de la Universidad de Salamanca y Dña. Ana Belén Nieto Librero, profesora del Departamento de Estadística de la Universidad de Salamanca,

HACEN CONSTAR:

Que el trabajo titulado "Estudio e implementación de redes neuronales convolucionales para la segmentación de imágenes", que se presenta, ha sido realizado por Julia García Vega, con DNI 70961775J y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado en Estadística en esta Universidad.

Salamanca, 3 de julio de 2024

VICENTE VILLARDON JOSE LUIS - 07839395Y

Firmado digitalmente por VICENTE VILLARDON JOSE LUIS - 07839395Y
Fecha: 2024.07.01 10:42:23 +02'00'

Fdo.: José Luis Vicente Villardón

NIETO LIBRERO ANA BELEN - 11966438K

Firmado digitalmente por NIETO LIBRERO ANA BELEN - 11966438K
Fecha: 2024.07.01 11:04:07 +02'00'

Fdo.: Ana Belén Nieto Librero

García Vega, J. (2024). *Estudio e implementación de redes neuronales convolucionales para la segmentación de imágenes*. Trabajo de Fin de Grado. Salamanca: Universidad de Salamanca.

RESUMEN

Las redes neuronales convolucionales se han convertido en la principal técnica para la segmentación de imágenes quedando los métodos anteriores obsoletos. Desde que se presentó la primera arquitectura convolucional adaptada para la segmentación, numerosas arquitecturas han sido propuestas por investigadores. Entre estas arquitecturas destacan SegNet, U-Net y DeepLab. Este trabajo presenta un estudio sobre la implementación y optimización de estas tres arquitecturas de redes neuronales convolucionales para la segmentación de imágenes del entorno urbano. El objetivo de este trabajo es lograr un modelo de segmentación entrenado que sea óptimo en la segmentación de imágenes de la vía urbana desde la perspectiva de un peatón para un posible futuro uso en dispositivos de asistencia en la navegación de personas con discapacidad visual. Para ello se utilizó herramientas como Python, Google Colaboratory y TensorFlow y se evaluaron los modelos en base a diversas métricas de precisión. Los resultados muestran que la arquitectura U-Net con hiperparámetros optimizados se ajusta mejor a los datos frente a las otras dos arquitecturas ofreciendo una segmentación con un porcentaje de clasificación correcta de los pixeles de un 90%. Esta alta precisión deja abierta la posibilidad de una implementación en dispositivos de asistencia para personas con discapacidad visual para una navegación más autónoma y segura por la vía urbana.

Palabras claves: redes neuronales convolucionales, segmentación de imágenes, U-Net, SegNet, DeepLab, discapacidad visual, entorno urbano

García Vega, J. (2024). *Study and implementation of convolutional neural networks for image segmentation*. Bachelor's Degree Final Project. Salamanca: University of Salamanca.

ABSTRACT

Convolutional neural networks have become the primary technique for image segmentation, rendering previous methods obsolete. Since the introduction of the first convolutional architecture adapted for segmentation, numerous architectures have been proposed by researchers. Notable among these are SegNet, U-Net, and DeepLab. This work presents a study on the implementation and optimization of these three convolutional neural network architectures for urban environment image segmentation. The objective of this work is to achieve a trained segmentation model that is optimal for segmenting urban road images from a pedestrian's perspective for potential future use in assistive navigation devices for visually impaired individuals. Tools such as Python, Google Colaboratory, and TensorFlow were used, and the models were evaluated based on various accuracy metrics. The results show that the U-Net architecture with optimized hyperparameters fits the data better than the other two architectures, offering segmentation with a correct pixel classification rate of 90%. This high accuracy opens the possibility of implementation in assistive devices for visually impaired individuals, facilitating more autonomous and safer navigation in urban environments.

Keywords: convolutional neural networks, image segmentation, U-Net, SegNet, DeepLab, visual impairment, urban environment

AGRADECIMIENTOS

Me gustaría dar las gracias a todas las personas que me han apoyado a lo largo de esta etapa tan importante de mi vida.

A mis padres, por su infinita paciencia y comprensión. Gracias por aguantarme durante los momentos de estrés y por vuestro constante amor y apoyo. Habéis sido un pilar fundamental.

A mis amigos, por estar siempre a mi lado. Gracias por proporcionarme apoyo incondicional y ánimos cuando más lo necesitaba.

A mis tutores, por su orientación durante el desarrollo de este trabajo.

A todos vosotros, muchas gracias.

Índice

Índice de Figuras

Índice de Tablas

1. Introducción

Actualmente, la inteligencia artificial está consolidándose como un pilar en la investigación, crecimiento y desarrollo de nuevos avances en diversos campos. Esta forma de inteligencia, que trata de emular la capacidad humana de razonamiento mediante algoritmos y sistemas computacionales, se está expandiendo y dejando una huella significativa en prácticamente todas las áreas de la sociedad (Rana, 2024).

Dentro de la amplia gama de disciplinas que componen la inteligencia artificial, el campo de visión por computadora está ganando cada vez más tracción (Sebe, 2005). Esta disciplina consiste en desarrollar sistemas que puedan interpretar la información contenida en una imagen. Mientras que para la mayoría de las personas esta tarea de percibir el entorno es algo natural y automático, para un ordenador representa un desafío de gran complejidad. Los sistemas de visión por computadora deben emular el sistema visual humano, que es capaz de reconocer formas, luces, sombras y otros detalles, todo ello mediante técnicas matemáticas y algoritmos sofisticados. Desde que surgió el concepto de visión por computadora, se han logrado grandes avances en esta disciplina. Sin embargo, aún queda un largo camino por recorrer para alcanzar el mismo nivel de detalle y precisión que conseguiría una persona.

Dentro del contexto de visión por computadora destaca la técnica de segmentación de imágenes, técnica mediante la cual se divide una imagen en distintas regiones permitiendo así la identificación y aislamiento de elementos específicos dentro de la imagen. La segmentación de imágenes presenta un desafío en el campo desde que surgió el concepto de visión artificial (Rosenfeld, 1976).

Se han desarrollado un gran número de algoritmos de segmentación de imágenes. Algunos de ellos son el corte de grafos (Boykov et al., 2001), la umbralización (Sezgin & Sankur, 2004) o la técnica de contornos activos (Kass et al., 1988). Sin embargo, entre todas las diversas técnicas para la segmentación destacan aquellas técnicas con aprendizaje profundo como son las redes neuronales convolucionales (Minaee et al., 2022). Su aplicación se ha extendido en diversas áreas como por ejemplo la medicina (Esteva et al., 2021), la construcción (Paneru & Jeelani, 2021), la agricultura (Mortensen et al., 2016) y por supuesto, la navegación (Miyamoto et al., 2019).

Existen múltiples estudios de la segmentación de imágenes de carreteras desde la perspectiva de un vehículo que contribuyen en los avances de la navegación autónoma de vehículos (Kaymak & Uçar, 2019), (Treml et al., 2016), (Feng et al., 2021). Sin embargo, es notable la falta de estudios que obtengan modelos adaptados y mejorados para la segmentación de imágenes de la vía urbana desde la perspectiva de un peatón. Se puede pensar inicialmente que la necesidad de este estudio no existe ya que los peatones al poder ver pueden navegar perfectamente por las vías urbanas. Sin embargo, las personas con discapacidad visual presentan dificultades al no poder percibir su entorno.

Este trabajo, por tanto, se centra en abordar la falta de investigación en la segmentación de imágenes de entornos urbanos desde la perspectiva de un peatón mediante la exploración de diversas arquitecturas de redes neuronales convolucionales para la segmentación estas imágenes y su posterior configuración para un entrenamiento óptimo.

1.1. Contextualización del problema

Según el informe mundial sobre la visión de la Organización Mundial de la Salud (OMS), en el mundo hay por los menos 2.200 millones de personas que padecen una discapacidad visual (Organización Mundial de la Salud, 2020). En concreto, en España, según la encuesta del Instituto Nacional de Estadística (INE) de 2020 sobre discapacidades, el número total de personas con discapacidad visual en España es de aproximadamente 1.05 millones de personas entre las cuales según otra encuesta del 2008 58.3 miles de personas tenían ceguera total. Esta población no tiene previsión de reducirse ya que analizando los datos recopilados por el INE a lo de los años se revela una tendencia clara de aumento en el número de personas con discapacidad visual en España a lo largo de las últimas décadas. En 1999, una encuesta registró 304.5 miles de personas con discapacidad visual, aumentando a 979.2 miles en 2008 y llegando a 1051.3 miles en 2020 (INE, 1999, 2008, 2020).

Esta población se enfrenta diariamente a un problema que actualmente carece de una solución aceptable: la dificultad para navegar de manera segura y autónoma por las calles y entornos urbanos. Desde las últimas décadas, continua una tendencia positiva de crecimiento y expansión de las ciudades (Ministerio de Transportes et al., 2022) lo que causa que los entornos urbanos se hagan más complejos y densos provocando a su vez que la navegación de las personas con discapacidad visual se vea más dificultada. Además, considerando la división por tamaño del municipio, el total de personas con discapacidad visual que viven en municipios de más de 10000 habitantes, municipios ya considerados ciudades, sería de 838.8 miles de personas (INE, 2020). Es decir, la gran mayoría de personas con discapacidad visual se mueven por entornos urbanos de gran complejidad debido a su extensión.

Aunque existen algunas herramientas diseñadas para ayudar a las personas con discapacidad visual en su movilidad, como son los bastones blancos y los perros guía, estas herramientas siguen sin ser totalmente efectivas. Las tecnologías existentes pueden ser limitadas en su alcance, precisión y facilidad de uso, lo que deja a muchas personas ciegas dependientes de la ayuda de otros o expuestas a riesgos al moverse por entornos desconocidos. Además, hay aspectos clave de la navegación urbana, como la detección de obstáculos en tiempo real, la interpretación de señales de tráfico y la orientación precisa, que aún no tienen soluciones totalmente satisfactorias. Por ejemplo, algo tan sencillo como cruzar la calle de manera segura es especialmente difícil para personas ciegas debido a la falta de información sobre el estado de los semáforos y las señales de tráfico. Aunque existen soluciones como dispositivos portátiles que emiten señales de audio para indicar cuándo es seguro cruzar, no están ampliamente disponibles ni son infalibles (Ayuntamiento de Cartagena, 2023). O, por ejemplo, a menudo, las personas ciegas pueden necesitar tomarse un descanso durante sus desplazamientos, pero encontrar un banco disponible puede serles complicado si no conocen ya su localización ya que no pueden verlos.

En los últimos años, han surgido múltiples aplicaciones de navegación por GPS que tratan de proporcionar una ayuda adicional al bastón blanco y al perro guía (Paratore & Leporini, 2023). Sin embargo, la mayoría de los enfoques actuales carecen de la capacidad de tomar en cuenta el estado del entorno en tiempo real, lo que limita su utilidad al tratarse la calle de un entorno dinámico y complejo.

La movilidad de las personas ciegas en entornos urbanos sigue siendo un desafío considerable, a pesar de los avances en tecnología y dispositivos de asistencia. La falta de soluciones efectivas y accesibles para la navegación segura y autónoma representa una necesidad insatisfecha que afecta no solo la comodidad física, sino a la calidad general de vida de las personas ciegas. Es por ello la necesidad de desarrollar enfoques innovadores y adaptados para mejorar la accesibilidad urbana de esta población. Al estudiar y comparar

distintas técnicas de segmentación de imágenes y entrenarlas específicamente con imágenes desde la perspectiva de un peatón, se puede mejorar la precisión de la información proporcionada y que así nuevos dispositivos accesibles utilicen esta información para proporcionar retroalimentación en tiempo real sobre el entorno permitiendo una navegación más segura y eficiente para las personas con discapacidad visual.

1.2. Objetivos

El objetivo principal de este trabajo es obtener un modelo de segmentación óptimo de imágenes de entornos urbanos desde la perspectiva de un peatón para que este modelo pueda ser usado en futuros dispositivos accesibles que proporcionen esta información obtenida como retroalimentación a personas con discapacidad visual para mejorar su movilidad en entornos urbanos. Para lograr este objetivo principal se plantean los siguientes objetivos:

- Investigar y analizar distintos modelos de segmentación de imágenes existentes actualmente basados en redes neuronales convolucionales.
- Generar un conjunto de datos de imágenes de la calle desde la perspectiva de un peatón apto para entrenar modelos de segmentación de imágenes.
- Implementar las distintas arquitecturas de redes neuronales convolucionales estudiadas, ajustando los hiperparámetros para optimizar el rendimiento.
- Comparar los resultados de los diferentes modelos de segmentación obtenidos, evaluando distintas métricas de rendimiento.

1.3. Estructura

El presente trabajo está estructurado en diversos capítulos que abordan distintos aspectos del estudio e implementación de redes neuronales convolucionales para la segmentación de imágenes. El primer capítulo corresponde con la introducción ya desarrollada por lo que se comienza con la explicación de los capítulos comenzando desde el segundo capítulo.

En el Capítulo 2, Marco teórico, se proporciona una base teórica sobre las metodologías usadas. Se comienza explicando el aprendizaje automático, pasando a desarrollar a continuación el concepto fundamental del trabajo, las redes neuronales artificiales. Se detalle de estas tanto su estructura como el concepto del aprendizaje de una red neuronal. Una vez se tiene esta base, se profundiza explicando los conceptos de aprendizaje profundo y de redes neuronales convolucionales. Finalmente, todos los conceptos explicados se aúnan para detallar la segmentación de imágenes, en concreto, la segmentación de imágenes mediante redes neuronales convolucionales.

En el Capítulo 3, Material y métodos, se detallan las herramientas y tecnologías utilizadas para el desarrollo del trabajo. También se describe el conjunto de datos de imágenes utilizado y se analizan las arquitecturas de las redes neuronales convolucionales seleccionadas para la segmentación, las arquitecturas U-Net, Segnet y DeepLabv3+. Se finaliza detallando los hiperparámetros que se analizan para la optimización de los modelos y las métricas que se utilizan para la comparación de modelos.

En el Capítulo 4, Implementación, se detallan los pasos a seguir para la replicación de los resultados obtenidos. Se describe la configuración del entorno de desarrollo utilizada y el preprocesamiento que se ha realizado al conjunto de datos previo al entrenamiento. A continuación, se detalla cómo se han implementado en código las arquitecturas seleccionadas y como se ha procedido a su entrenamiento y ajuste de hiperparámetros.

En el Capítulo 5, Resultados, se detallan los resultados obtenidos del análisis exploratorio de datos y del entrenamiento de las distintas arquitecturas con las distintas configuraciones de hiperparámetros. Se incluye también una comparación entre los resultados obtenidos de cada arquitectura.

En el Capítulo 6, conclusiones, se resumen los hallazgos principales del trabajo discutiendo la efectividad de las arquitecturas de redes neuronales convolucionales seleccionadas para la segmentación de imágenes y como afectan los hiperparámetros al entrenamiento de estas. Se reflexiona sobre la posibilidad de incorporar el modelo de segmentación obtenido en dispositivos de asistencia en la navegación de personas con discapacidad visual.

Se finaliza el trabajo con el capítulo 7, bibliografía, y el capítulo 8, anexo I, que incluyen respectivamente el listado de referencias bibliográficas consultas durante la realización del trabajo y el enlace al código utilizado para la implementación y entrenamiento de las arquitecturas de redes neuronales convolucionales seleccionadas.

2. Marco teórico

2.1. Aprendizaje automático (*Machine Learning*)

El aprendizaje es un aspecto fundamental de las máquinas. Se define aprender como el proceso de adquirir nuevos conocimientos, comportamientos o habilidades o modificar aquellos ya existentes. El concepto aprendizaje automático o *Machine Learning* originó en la década de los 50 con el avance de las investigaciones en Inteligencia Artificial. El informático Arthur Samuel fue el primero en definir el concepto en 1959 tras crear el primer programa de aprendizaje automático.

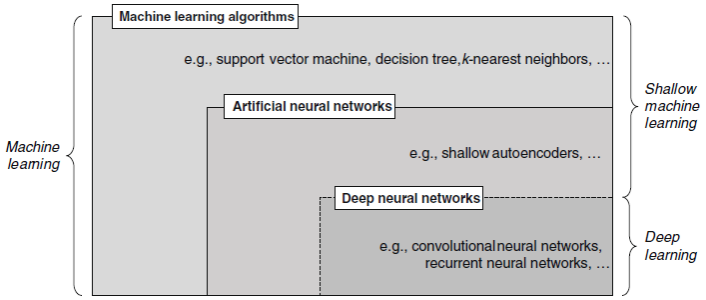
El aprendizaje automático es un campo de estudio multidisciplinario al estar relacionado con la tecnología de la información, la inteligencia artificial, la estadística, la psicología y muchas disciplinas más. Se basa en crear algoritmos que permitan a una computadora aprender, es decir, a reconocer patrones en los datos. El objetivo es tratar de recrear la forma de las personas de aprender a realizar distintas tareas a partir de la experiencia.

Los algoritmos de aprendizaje automático se pueden clasificar en su gran mayoría de la siguiente manera según como aprenden o son entrenados (Y. Zhang, 2010):

- Aprendizaje supervisado: se basa en estimar mediante una función una salida desconocida para una entrada a partir de otros pares entrada-salida conocidos, es decir, entradas cuya salida es conocida. Consiste en aprender de ejemplos ya proporcionados.
- Aprendizaje no supervisado: consiste en reconocer patrones existentes no preestablecidos o identificados inicialmente en los datos y a partir de estos patrones crear reglas. No se tiene ningún par entrada-salida conocido.
- Aprendizaje semisupervisado: combina tanto el aprendizaje supervisado como el no supervisado.
- Aprendizaje reforzado: se tienen entradas conocidas de las que se proporciona si se ha acertado o fallado en la predicción. Sin embargo, no se proporciona la salida concreta esperada. Se basa en el tanteo, es decir, ir probando y viendo si se falla o se acierta.

Entre los distintos algoritmos de aprendizaje automático existentes destacan, junto con otros muchos más, las redes neuronales artificiales. Además, este concepto de redes neuronales artificiales está intrínsecamente relacionado con el subcampo de *Machine Learning* denominado aprendizaje profundo o *Deep Learning*. Esto concepto se explicará más adelante ya que esta rama del aprendizaje automático es necesario primero entender que son las redes neuronales.

Figura 1. *Intersección entre Machine Learning y Deep Learning*



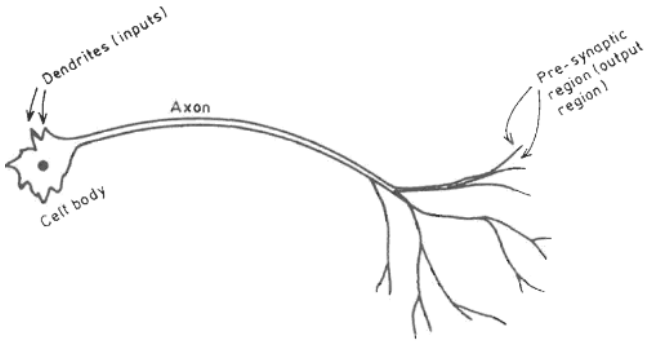
Nota. Adaptado de Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets, 31*(3), 685–695.

2.2. Redes neuronales artificiales

Una red neuronal artificial (RNA) es un modelo computacional basado en el proceso de decisión existente en la red de neuronas del sistema nervioso biológico. Está compuesta por un conjunto de neuronas artificiales interconectadas. Estas neuronas artificiales tratan de emular el comportamiento de una neurona biológica. Para entender cómo funciona una neurona artificial, hay que entender primero como funciona una neurona biológica.

Las neuronas biológicas son procesadores de información compuestos cuatro elementos principales: el soma, el axón, las dendritas y las sinapsis. Esta estructura se puede observar en la Figura 2. El cuerpo celular de la neurona, también llamado soma, es donde el ocurre el principal procesamiento constituyendo el órgano de cómputo de la neurona. La actividad neuronal pasa de una neurona a otra mediante señales eléctricas que se transportan por el axón de la neurona. El axón se considera el canal de salida. Al final del axón se llega a distintas sinapsis conectadas de manera direccional a las dendritas de las siguientes células. Las dendritas actúan como el canal de entrada de una neurona. De esta manera, al existir múltiples uniones sinápticas la salida de una neurona puede llegar a múltiples células y al existir múltiples dendritas en una célula a esa misma célula le puede llegar como entrada la salida de muchas otras células. Las interconexiones entre neuronas no tienen toda la misma prioridad. Cada interconexión tiene un peso asignado que determina el impacto de una neurona con otra. Además, estas sinapsis pueden ser de dos tipos: de excitación o de inhibición. (Graupe, 2013).

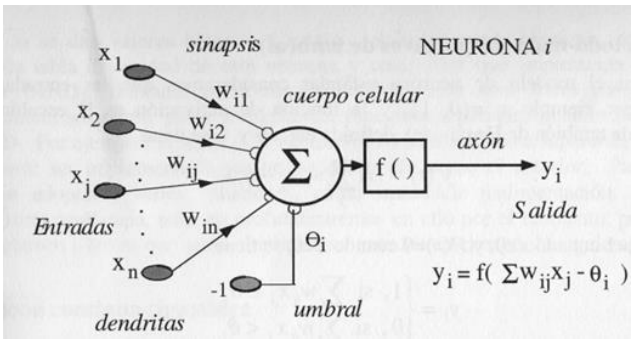
Figura 2. *Estructura de una neurona*



Nota. Adaptado de Graupe, D. (2013). *Principles Of Artificial Neural Networks (3rd Edition).* World Scientific Publishing Company.

El objetivo de las redes neuronales artificiales no es directamente la recreación literal del cerebro humano, si no reproducir las habilidades que tienen las personas de resolver problemas que todavía no han sido resueltos por los sistemas computacionales tradicionales (Anderson & McNeill, 1992). Sin embargo, al tratar de emular el comportamiento de las neuronas biológicas para obtener las mismas capacidades, las neuronas artificiales y biológicas guardan numerosas similitudes en su diseño y funcionalidad como se puede observar en la Figura 3.

Figura 3. *Estructura de una neurona artificial*



Nota. Adaptado de Larranaga, P., Inza, I., & Moujahid, A. (1997). Tema 8. redes neuronales. *Redes Neuronales, U. Del P. Vasco*, *12*, 17.

En las neuronas artificiales, la información llega mediante entradas $x_j(t)$. Tanto las variables de entrada como de salida pueden ser de cualquier tipo numérico (binario, real, discreto, bipolar, etc.). Cada entrada tiene un peso sináptico w_{ij} asociado que define la intensidad entre la neurona presináptica j y la neurona postsináptica i. A estos pesos sinápticos se suele añadir un parámetro adicional θ_i denominado umbral que se suele denotar por $w_{i0} = \theta_i$. Para combinar las entradas, sus correspondientes pesos y el valor umbral se utiliza una regla de propagación $h_i(t)$. Generalmente se utiliza la suma ponderada (Larranaga et al., 1997). Otras reglas de propagación utilizadas son la distancia euclídea, la distancia de Voronoi, Mahalanobis, etc. (Moreno Rodilla, 2023).

$$h_i(t) = \sigma\left(w_{ij}, x_j(t)\right) = \sum_{j=0}^{n} w_{ij} x_j(t)$$

A continuación, el valor obtenido de la regla de propagación se procesa con una función de activación a_i considerando también el estado de activación anterior y como resultado se obtiene el estado de activación actual de la neurona (López & Fernández, 2008).

$$a_i(t) = f_i\big(a_i(t-1), h_i(t)\big)$$

Muchos modelos de redes neuronales artificiales no suelen considerar en la función de activación el estado de activación anterior por lo que esta expresión se simplifica (López & Fernández, 2008):

$$a_i(t) = f_i\big(h_i(t)\big) = f_i\left(\sigma\left(w_{ij}, x_j(t)\right)\right) = f_i\left(\sum_{j=0}^{n} w_{ij} x_j(t)\right)$$

Existen múltiples funciones de activación posibles. La elección de la función de activación es a libertad del diseñador de la red. Algunas funciones de activación destacadas son:

- La función lineal: pondera el valor resultante de la regla de propagación por un valor constante η.

$$a_i(t) = \eta\, h_i(t)$$

- La función paso: dado un valor límite, si el valor resultante de la regla de propagación es mayor que el valor límite, si es igual tiene otro y si es menor tiene otro.

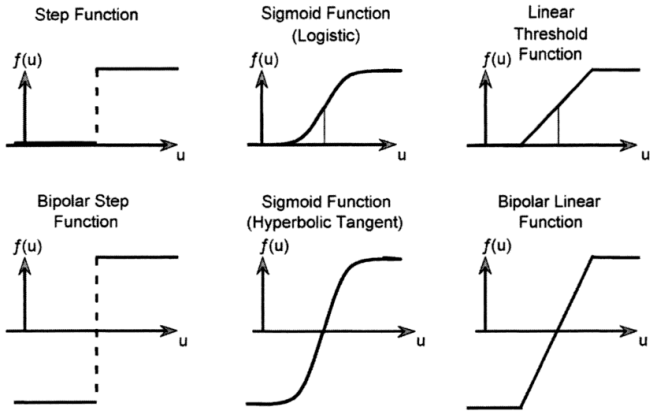
$$a_i(t) = \begin{cases} \beta & si\ h_i(t) > \theta \\ \gamma & si\ h_i(t) = \theta \\ \delta & si\ h_i(t) < \theta \end{cases}$$

- La función sigmoide: se define por la siguiente expresión:

$$a_i(t) = \frac{1}{1 + e^{-k \cdot h_i(t)}}$$

  Es la función de activación más utilizada al tener un rango de salida continuo, acotado y derivable (Vázquez Regueiro et al., 1995).

Figura 4. *Ejemplos de funciones de activación*



Nota. Adaptado de Priddy, K. L., & Keller, P. E. (2005). *Artificial Neural Networks: An Introduction*. SPIE Press.

Por último, al estado de activación actual obtenido se le aplica una función de salida F_i y finalmente se obtiene la señal de salida $y_i(t)$ que se transmite a las siguientes neuronas. Generalmente esta función de salida suele ser la función identidad (López & Fernández, 2008).

$$y_i(t) = F_i\big(a_i(t)\big) = a_i(t) = f_i\left(\sum_{j=0}^{n} w_{ij} x_j(t)\right)$$

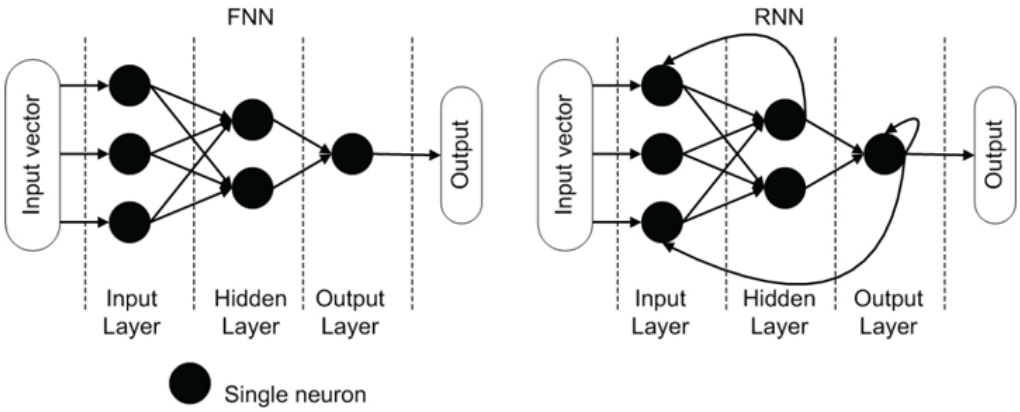
2.2.1. Arquitectura de una red neuronal artificial

Una única neurona artificial por sí sola no tiene utilidad. Es combinándolas y formando redes de neuronas artificiales cuando se consigue su máximo potencial. La forma en la que se interconectan las distintas neuronas es lo que se denomina la arquitectura de una RNA.

El conjunto de neuronas que conforman una RNA se organizan en unidades estructurales denominadas capas. Las neuronas en una misma capa se compartan de la misma manera. Existen tres tipos de capas: de entrada, de salida y oculta (Figura 5). La capa de entrada contiene las neuronas que reciben los datos del entorno. Se trata de la primera capa de una RNA. La capa de salida es la capa que contiene las neuronas que proporcionarán la respuesta resultante al entorno. El resto de las neuronas que no reciben ni proporcionan datos al entorno se encuentran en capas ocultas. Una misma RNA puede tener múltiples capas ocultas (Larranaga et al., 1997). Las redes neuronales artificiales con múltiples capas ocultas se denominan redes neuronales profundas. Estas arquitecturas suelen contener neuronas mucho más complejas que las de las típicas RNA. (Goodfellow et al., 2016)

Según la forma en la que se conecten las distintas neuronas y el flujo de datos, las arquitecturas se pueden dividir en dos grupos: arquitecturas unidireccionales (*feed-forward*) y arquitecturas recurrentes (Figura 5). En las arquitecturas unidireccionales el flujo de la información siempre es en el mismo sentido avanzando siempre desde la primera capa hasta la última. En las arquitecturas recurrentes el flujo de la información puede ir en ambas direcciones, tanto volver a capas anteriores, volver a la misma capa o ir a capas posteriores (Suzuki, 2011).

Figura 5. *Arquitectura unidireccional (FNN) y recurrente (RNN) de una RNA*



Nota. Adaptado de Suzuki, K. (2011). *Artificial neural networks: methodological advances and biomedical applications*. BoD–Books on Demand.

2.2.2. Aprendizaje de una red neuronal

Una vez que se tiene la arquitectura inicial de la red neuronal establecida, es necesario actualizar los pesos sinápticos y/o la estructura para que así sea capaz de resolver el problema intencionado. Este ajuste de la red es lo que se denomina aprendizaje. Este es el punto en el que se aplican los principios del aprendizaje automático.

La principal técnica usada para el aprendizaje es la actualización de pesos sinápticos según un determinado algoritmo de aprendizaje (Vázquez Regueiro et al., 1995). Sin este aprendizaje, si los pesos no son los adecuados, la red neuronal sería una serie de operaciones sin sentido incapaz de realizar predicciones precisas. Esta capacidad de aprender es una de las principales características de una RNA.

Existen tres paradigmas de aprendizaje tomados del aprendizaje automático: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado. Estos paradigmas de aprendizaje pueden ser utilizados por cualquier tipo de RNA y cada uno tiene múltiples algoritmos de aprendizaje. Cada algoritmo de aprendizaje está compuesto por una función de pérdida y una técnica de optimización.

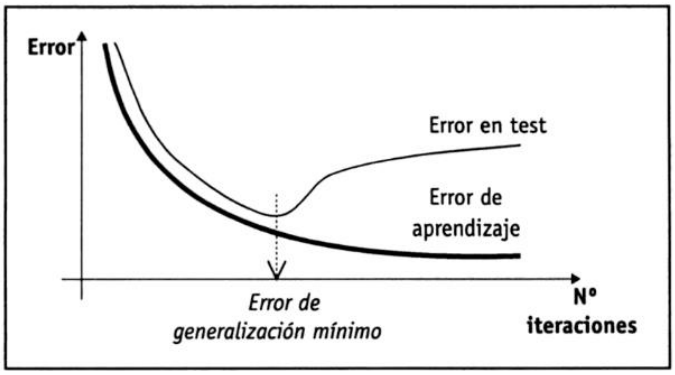
En el aprendizaje supervisado, se tiene un conjunto de datos de entrada de los cuales se sabe cuál debe ser su salida. Inicialmente, se pasa este conjunto de datos por la RNA sin entrenar y se comparan los resultados obtenidos con los esperados calculando el error obtenido según una función de pérdida establecida. Estas discrepancias son luego utilizadas para ajustar los pesos sinápticos según la técnica de optimización propagándose el error desde atrás, la capa de salida, hacia adelante, hasta la capa de entrada. Esto se repite sucesivamente hasta alcanzar un buen ajuste del problema (Anderson & McNeill, 1992). Sin embargo, este paradigma de aprendizaje no siempre es posible o práctico (Priddy & Keller, 2005).

En el aprendizaje no supervisado, no existe ningún dato de entrada del cuál se sepa su correcta salida. Se basa agrupar información desorganizada en distintas categorías no preestablecidas según características destacadas, peculiaridades o correlaciones en las entradas (Sydenham & Thorn, 2005). Este paradigma de aprendizaje se suele aplicar en problemas de estimación como son el modelado estadístico, compresión, filtrado, separación ciega de fuentes y agrupamiento en clústeres (Suzuki, 2011).

Por último, el aprendizaje reforzado es muy similar al aprendizaje supervisado. La diferencia es que, en el aprendizaje reforzado, en vez de proporcionar en cuanto difieren la salida obtenida y la salida esperada, únicamente se proporciona si se ha acertado o fallado en la predicción. Por tanto, el error que se propaga hacia atrás para corregir los pesos es binario (Sydenham & Thorn, 2005). Como en el aprendizaje no supervisado, en el aprendizaje reforzado tampoco se proporciona la salida concreta esperada.

En el proceso de entrenamiento de la RNA, se debe evitar el fenómeno denominado sobreentrenamiento. Este consiste en que, al tratar de alcanzar un error de aprendizaje muy reducido en el entrenamiento de la red, es decir, al tratar de obtener las menores discrepancias entre los valores reales y las predicciones, el error al pasar por la red datos no utilizados en el entrenamiento se degrada. Esto se debe a que la RNA se ha sobreajustado a los patrones particulares existentes en los datos de entrenamiento no funcionando ya con nuevos datos al perder la capacidad de generalización (López & Fernández, 2008). Este fenómeno se puede observar en la siguiente figura.

Figura 6. *Sobreaprendizaje de una RNA*



Nota. Adaptado de López, R. F., & Fernández, J. M. F. (2008). *Las Redes Neuronales Artificiales*. Netbiblo.

Una vez la RNA ya ha sido entrenada, al suministrar una entrada la red devuelve una salida sin modificar los pesos sinápticos que quedan ya estáticos. A partir de este momento, la RNA funciona en modo de recuerdo o ejecución (Moreno Rodilla, 2023).

2.3. Aprendizaje profundo (*Deep Learning)*

En las últimas décadas se han producido numerosos avances en el ámbito del aprendizaje automático debido a la evolución de las redes neuronales artificiales a redes neuronales profundas (Goodfellow et al., 2016). Este tipo de arquitecturas de redes poseen capacidades de aprendizaje mucho superiores recibiendo el nombre de aprendizaje profundo o *Deep Learning*.

Las redes neuronales artificiales tienen dificultades en la representación de datos complejos (Goodfellow et al., 2016). Este problema consiste en que para diseñar un algoritmo de aprendizaje para una RNA es necesario conocer los distintos factores que influyen en el

problema de estudio. Por ejemplo, para predecir si un paciente tiene una enfermedad o no se proporcionan los distintos factores que se deben analizar para determinar el diagnóstico como por ejemplo la temperatura corporal, si tiene tos o no, etc. Sin embargo, hay problemas en la vida real cuyos factores de variación no se pueden determinar simplemente.

Las redes neuronales profundas resuelven este problema al poder tener como entrada datos sin preprocesar ya aprenden mejores representaciones complejas de los mismos (Janiesch et al., 2021). Esta es la principal capacidad de una red neuronal profunda junto con la capacidad de poder manejar grandes cantidades de datos. Este aprendizaje de representación es lo que también se denomina aprendizaje profundo.

Uno de los modelos de redes neuronales profundas que más tracción tiene es la red neuronal convolucional (Li et al., 2022).

2.4. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN o ConvNet) son un modelo de aprendizaje profundo utilizado para procesar datos presentados en matrices, generalmente imágenes, inspirado en el mecanismo de percepción visual de los seres vivos. Sus fundamentos fueron presentados por primera vez en 1990 por LeCun et al. que se basó a su vez en el Neocognitron introducido en 1998 por Fukushima. Desde entonces, su implementación ha permitido múltiples avances en numerosas áreas como por ejemplo en visión por computadora y en el procesamiento del lenguaje natural (Li et al., 2022).

Surgen debido a que, en las redes neuronales profundas, al estar todas las neuronas de una capa conectadas a todas las neuronas de la capa anterior, los parámetros a estimar de la red pueden llegar a ser miles lo que provoca demasiados costes de computación y memoria (Lopez Pinaya et al., 2019). Esto sucede en el caso de que los datos sean imágenes. Las imágenes vienen representadas en matrices multidimensionales llamadas tensores. Un tensor que representa a una imagen tiene tres dimensiones: el número de filas y columnas corresponde al alto y ancho de la imagen representando cada celda un píxel. La tercera dimensión corresponde al esquema de color de la imagen. Por ejemplo, en el caso de una imagen en color, el tensor tendrá una profundidad de 3 para representar cada valor RGB y en el caso de una imagen en una escala de grises únicamente tendrá una profundidad de 1. Suponiendo una imagen en color de dimensión 200x200, solo una neurona totalmente conectada tendría 120.000 pesos sinápticos asociados. Teniendo en cuenta que las redes neuronales profundas tienen numerosas capas ocultas, ajustar tantos pesos conllevaría no solo altos recursos computacionales si no también un sobreentrenamiento.

Por ello, buscando un método más eficiente, surge la idea de en vez de tener en cuenta la imagen entera a la vez, dividir la imagen en regiones más pequeñas y analizar estas regiones paralelamente. En esta idea se basan las redes neuronales convolucionales.

2.4.1. Capas de una red convolucional

Al ser un tipo de red neuronal profunda está compuesta por una capa de entrada, una capa de salida y entre ellas múltiples capas ocultas. Lo que diferencia a las CNN, es que sus capas ocultas están compuestas por cuatros tipos de capas: capa convolucional, capa no linealidad, capa de agrupación y capa totalmente conectada. A veces, la capa no linealidad se agrupa junto con la capa convolucional. A continuación, se explicará con mayor profundidad cada capa.

2.4.1.1. Capa convolucional

Esta capa es fundamental en las redes neuronales convolucionales, recibiendo de ella su propio nombre. Su objetivo es extraer características determinantes como los bordes, colores, texturas, etc. (Sarvamangala & Kulkarni, 2022). Esto se logra al generar mapas de características mediante una operación de convolución entre un conjunto de filtros y las entradas de la capa. A continuación, se explicarán más en profundidad estos conceptos de filtros, operación de convolución y mapas de características.

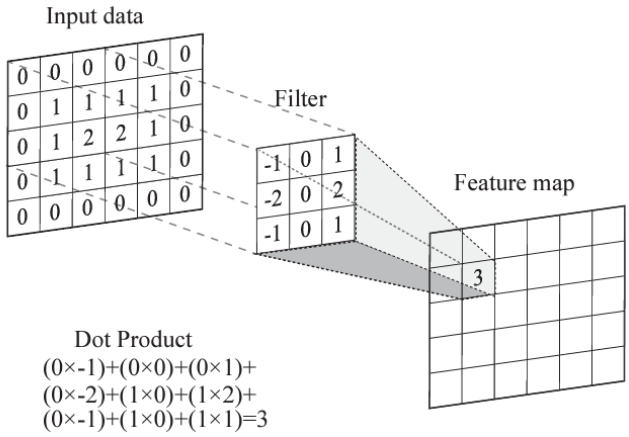
Un filtro, también llamado kernel, se define como una matriz cuadrada de dimensiones relativamente pequeñas compuesta por números discretos, donde los parámetros son los valores contenidos en dicha matriz. Estos valores representan un patrón específico que el filtro intentará identificar en la entrada de la capa.

La operación de convolución consiste en aplicar un filtro a lo largo de la entrada de la capa, representada en forma de matriz, y calcular su producto escalar en cada posición para obtener una nueva matriz, manteniendo así la relación espacial, que se denomina mapa de características (Lopez Pinaya et al., 2019). Paso a paso consiste en, posicionar el filtro sobre el cuadrante superior izquierdo de la matriz de entrada, calcular el producto escalar entre los parámetros del filtro y los correspondientes elementos de la matriz de entrada (Figura 7) y a continuación mover el filtro para repetir el proceso. Por ejemplo, si la entrada fuese una imagen de un solo canal, es decir, una matriz bidimensional, sobre cada elemento de entrada se aplica la siguiente ecuación siendo m y n el ancho y alto del filtro (Goodfellow et al., 2016):

$$Salida(i,j) = \sum_m \sum_n Imagen(i-m, j-n) \cdot Filtro(m,n)$$

El filtro se mueve de izquierda a derecha y de arriba abajo y se termina aplicando a todas las posiciones de la matriz de entrada. El valor resultante del producto escalar indica si se ha detectado el patrón que representa el filtro en esa posición de la matriz de entrada.

Figura 7. *Cálculo de un producto escalar en la operación de convolución*



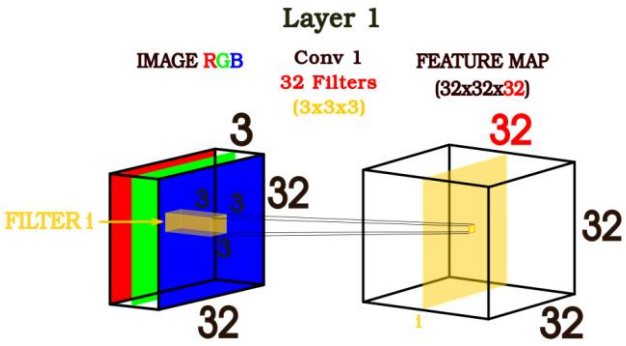
Nota. Adaptado de Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2019). Convolutional neural networks. *In Machine Learning: Methods and Applications to Brain Disorders* (pp. 173–191). Elsevier. https://doi.org/10.1016/B978-0-12-815739-8.00010-9

La correspondencia de estos conceptos con los elementos de una red neuronal es lo siguiente: cada celda de un mapa de características es una neurona, estas neuronas únicamente están conectadas a una pequeña región de la capa anterior y los pesos

sinápticos de estas interconexiones corresponden a los parámetros del filtro. Estos pesos serán los mismos para todas las neuronas de un mismo mapa de características y se van actualizando sus estimaciones a medida que se entrena la red. De esta manera, como se explicó antes, al no estar todas las neuronas de una capa interconectadas con todas las neuronas de la capa anterior y al ser los mismos pesos para distintas neuronas, se reducen los parámetros a estimar reduciendo así los costes computacionales (Hinton et al., 2012).

Para poder extraer diversas características de la imagen se deben utilizar múltiples filtros lo que resulta en varios mapas de características. Estos mapas de características se apilan formando una matriz multidimensional que es la salida de la capa de convolución.

Figura 8. *Resultado de una capa de convolución con 32 filtros sobre una imagen RGB*



Nota. Grupo de Fotónica Aplicada, UTN-FRD, Universidad Tecnológica Nacional - Facultad Regional Delta. (s. f.). *Dimensiones, filtros y operaciones de la Redes Neuronales Convolucionales*. UTN-GFA.github.io. https://utn-gfa.github.io/

Los hiperparámetros correspondientes de la capa de convolución son los siguientes:

- Tamaño del filtro (F): se suelen usar filtros de tamaño reducido siendo el más usual 3x3.
- Número de filtros (K): determina el número de características que se extraen. Usar más filtros implica una red neuronal más potente pero también incrementa el riesgo de sobreajuste debido al incremento de parámetros a estimar (Lopez Pinaya et al., 2019).
- Relleno de ceros o *padding* (P): indica el número de filas y columnas que se añaden alrededor del borde de la matriz rellenas de ceros. Esto permite mantener las dimensiones de la imagen en el mapa de características sin que se reduzca su dimensión ya que los pixeles de los bordes se convierten en pixeles centrales. En la Figura 7 se puede observar cómo se ha añadido un *padding* de (1,1).
- Paso o *stride* (S): indica el número de celdas que se desplaza el filtro al cambiar de posición. Suele valer 1. Cuanto mayor sea, menor será el alto y ancho del mapa de características.

Estos hiperparámetros deben ser establecidos antes de empezar a entrenar la red ya que no son aprendidos en el proceso de entrenamiento. Únicamente se aprenden los parámetros de los filtros (Yamashita et al., 2018). Con estos hiperparámetros establecidos, si la entrada tiene dimensión $W_1 \times H_1 \times D_1$, la dimensión de la salida de la capa de convolución es $W_2 \times H_2 \times D_2$ donde:

$$W_2 = \frac{W_1 - F + 2P}{S + 1}; \qquad H_2 = \frac{H_1 - F + 2P}{S + 1}; \qquad D_2 = K$$

2.4.1.2. Capa no linealidad

Como es común en las redes neuronales, tras la operación de convolución, que es esencialmente una operación lineal, se aplica una función de activación no lineal a los mapas de características resultantes. En algunas ocasiones, esta función puede integrarse dentro de la misma capa de convolución, mientras que en otras se implementa como una capa separada para aportar mayor flexibilidad en la arquitectura de la red.

La función de activación no lineal más utilizada en las CNN es la función Unidad Rectificada Uniforme (ReLU) ya que proporciona mejores resultados (Nair & Hinton, 2010). Otras funciones de activación no lineales que también son utilizadas ocasionalmente son la función sigmoide o la tangente hiperbólica. La función ReLU corresponde con la siguiente función:

$$g(y) = \begin{cases} 0 & si\ y < 0 \\ y & si\ y \geq 0 \end{cases}$$

2.4.1.3. Capa de agrupación

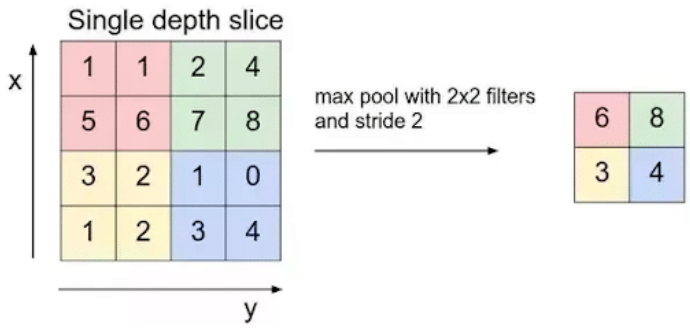
El objetivo de la capa de agrupación o capa de *pooling* es reducir la dimensión, en lo que se refiere a la altura y anchura no a la profundidad, de los mapas de características obtenidos. Esto consecuentemente reduce el número de parámetros a estimar reduciendo así la complejidad del modelo y las posibilidades de un sobreajuste. Esta capa permite introducir invariación frente a pequeños cambios locales en la entrada (Goodfellow et al., 2016).

Al igual que la capa de convolución, esta capa actúa sobre pequeñas subregiones de la matriz de entrada hasta llegar a cubrir todos los elementos de la matriz. Por ello, se debe establecer como hiperparámetros el tamaño de la ventana, análogo al tamaño del filtro de la capa de convolución, y el paso o *stride*. Sin embargo, a diferencia de la capa de convolución, esta capa de agrupación no tiene ningún parámetro que aprender mediante el entrenamiento.

Existen diversos métodos para realizar esta reducción de dimensionalidad como por ejemplo tomar la media de los valores o el valor máximo de una subregión. Las operaciones de agrupación más comunes son *max-pooling*, *average pooling*, *spectral pooling, spatial pytramid pooling, L2-norm pooling* y *multiscale orderless pooling* (Sakib et al., 2019) siendo la operación más usada la técnica de *max-pooling* al proporcionar mejores resultados en gran diversidad de situaciones (Boureau et al., 2010).

Una capa de *max-pooling* para cada subregión de la matriz de entrada toma el valor máximo y desprecia el resto de los valores. El tamaño de ventana que suele utilizar es de $2\ x\ 2$ y el *stride* también de valor 2 y en la siguiente figura se puede ver un ejemplo de implementación con estos hiperparámetros.

Figura 9. *Procesamiento de max-pooling con stride 2 y ventana 2x2*



Nota. Adaptado de Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, 1–6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

2.4.1.4. Capa totalmente conectada

Una vez que las características han sido extraídas por las capas de convolución y su representación se ha reducido comprimiendo la información con las capas de *pooling*, a continuación, y como capas finales, se encuentran capas totalmente conectadas cuya función principal es terminar de clasificar estas características obtenidas para obtener un mejor resultado (Springenberg et al., 2014).

Antes de pasar a las capas totalmente conectadas tras obtener de las capas de convolución y *pooling* los mapas de características finales, estas salidas se deben aplanar en un único vector unidimensional para poder ser utilizadas como entradas a una capa totalmente conectada.

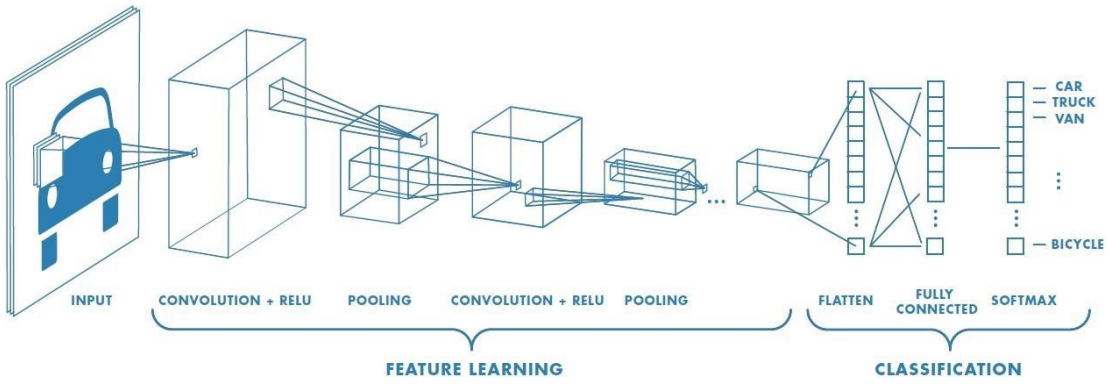
Estas capas son análogas a las capas de una red neuronal profunda tradicional. Contienen neuronas que están totalmente conectadas a todas las neuronas de la capa anterior mediante pesos sinápticos que deben ser aprendidos (Figura 10). Siguiendo el planteamiento explicado de las redes neuronales artificiales, cada capa totalmente conectada está seguida de una función de activación no linear como por ejemplo la utilizada anteriormente función ReLU (Yamashita et al., 2018).

2.4.2. Arquitecturas de redes neuronales convolucionales

Las redes neuronales convolucionales suelen seguir una arquitectura común o patrón pese a no existir una fórmula clara de cómo construir una arquitectura de una CNN. Para construir una buena CNN se suele recurrir a trabajos ya existentes y aprender de la experimentación. No se trata de combinar las capas anteriormente explicadas sin ningún sentido.

Una arquitectura común es aplicar repetidamente una capa convolucional y su función activación junto con una capa de agrupación y terminar con varias capas totalmente conectadas. Un ejemplo de esta arquitectura se puede observar en la siguiente figura.

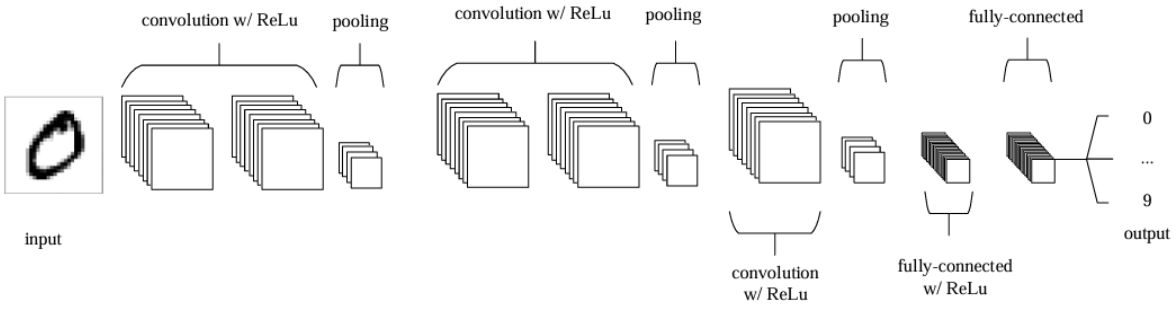
Figura 10. *Ejemplo de arquitectura de una CNN alternando capas de convolución y agrupación*



Nota. Adaptado de Classification of Palm Trees Diseases using Convolution Neural Network - Scientific Figure on ResearchGate. Disponible en: https://www.researchgate.net/figure/An-Overview-of-a-Convolutional-Neural-Network-CNN-Architecture-21_fig1_361743779

Otro patrón muy usado en la arquitectura de redes convolucionales es aplicar dos capas de convolución seguidas antes de una capa de agrupación. Esto permite extraer de las imágenes de entrada características más complejas (O'Shea & Nash, 2015). Se termina también siempre con las capas totalmente conectadas. Esta arquitectura se puede observar en la siguiente ilustración.

Figura 11. *Ejemplo de arquitectura de una CNN apilando varias capas de convolución antes de una capa de agrupación*



Nota. Adaptado de O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. http://arxiv.org/abs/1511.08458

Se han publicado numerosas arquitecturas de redes neuronales convoluciones como se puede observar en la Tabla 1 que contiene algunas de las CNN más populares en la literatura científica.

Tabla 1. *Algunas de las arquitecturas de CNN más reconocidas desde los inicios hasta más recientemente*

Nombre de la Arquitectura	Año	Contribución principal	Parámetros	Capas	Referencia
LeNet	1998	Primera arquitectura popular de una CNN	60 k	5	(LeCun et al., 1995)
AlexNet	2012	Más profunda que LeNet e introduce ReLU y pooling superpuesto	60 M	8	(Krizhevsky et al., 2012)
ZfNet	2014	Visualización de capas intermedias	60 M	18	(Zeiler & Fergus, 2014)
VGG	2014	Utiliza un tamaño de filtro pequeño	4 M	19	(Simonyan & Zisserman, 2014)
GoogLeNet	2015	Introduce conceptos de bloques, transformación dividida y combinación	23,6 M	22	(Szegedy et al., 2015)
Inception-V3	2015	Convoluciones factorizadas mejorando eficiencia computacional	23,6 M	159	(Szegedy et al., 2016)
ResNet	2016	Aprendizaje residual	25,6 M	152	(He et al., 2016)
Xception	2017	Convolución en profundidad seguida de una convolución puntual	8,6 M	452	(Chollet, 2017)
DenseNet	2017	Flujo de información entre capas	25,6 M	190	(Huang et al., 2017)

MobileNetV2	2018	Bloques de construcción optimizados	3,4 M	53	(Sandler et al., 2018)
GhostNet	2019	Introduce feature ghosting	-	-	(Han et al., 2020)

La mayoría de las arquitecturas CNN están construidas para un propósito en concreto entre los que se distinguen la clasificación, detección y segmentación de imágenes.

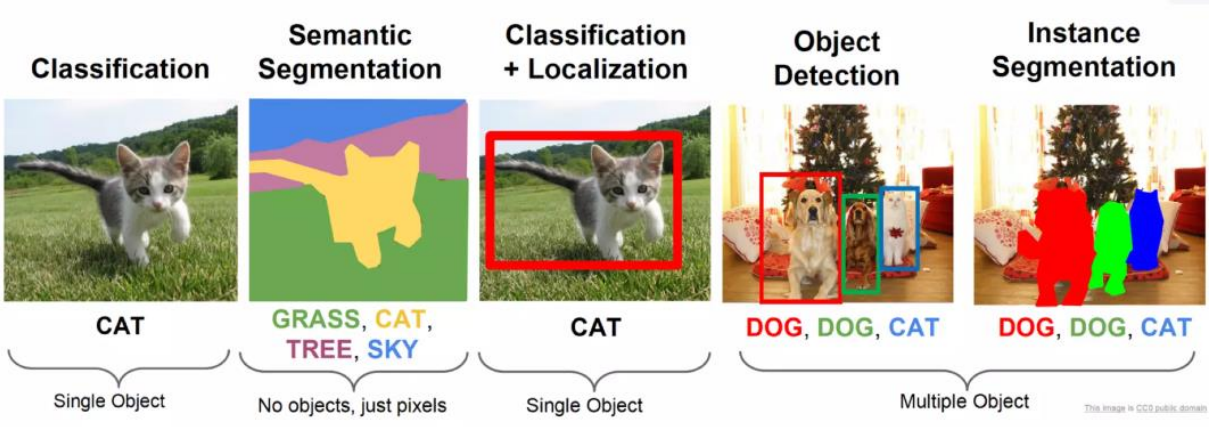
2.5. Segmentación de imágenes

En el campo del procesamiento de imágenes, el objetivo de procesar de una imagen suele diferir según el enfoque o aplicación que se desea. Este objetivo puede ser uno de estos tres: clasificación, detección o segmentación.

La clasificación de imágenes (Figura 12) consiste en determinar a qué clase entre unas clases preestablecidas pertenece cierta imagen. La detección de objetos (Figura 12) permite identificar diversos objetos en una imagen clasificándolos y determinar el área rectangular en el que se encuentran. Por último, la segmentación permite dividir las imágenes en distintas regiones según la presencia de las distintas clases. A diferencia de las dos anteriores técnicas, la segmentación actúa a bajo nivel realizando un análisis a nivel de píxel.

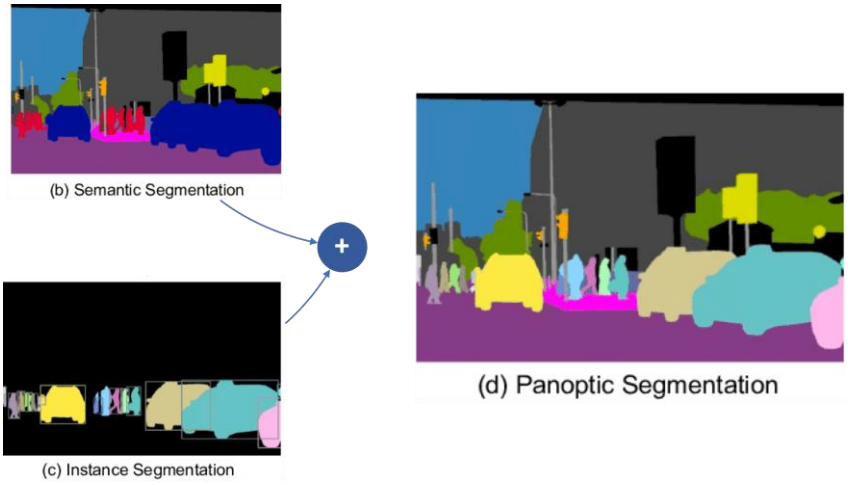
En la segmentación se deben distinguir dos principales técnicas: la segmentación semántica y la segmentación de instancias. La segmentación semántica (Figura 12, Figura 13) consiste en clasificar cada píxel de una imagen entre las distintas clases definidas, no distingue entre distintas instancias de una misma clase. La segmentación de instancias (Figura 12, Figura 13) permite identificar y localizar cada instancia de las clases definidas a nivel de píxel detectando los contornos. Existe una tercera técnica de segmentación no tan difundida en la literatura científica denominada segmentación panóptica (D. Zhang et al., 2018) (Figura 13) que combina las dos técnicas anteriores de segmentación, distingue entre las distintas instancias de los objetos a la vez que reconoce el fondo clasificándolo acordemente quedando todos los píxeles de la imagen clasificados.

Figura 12. *Ejemplos de objetivos del procesamiento de imágenes*



Nota. Adaptado de Frajberg, D. (2017). *Introduction to the Artificial Intelligence and Computer Vision revolution*. Dipartimento di elettronica, informazione e bioingegneria, Politecnico Di Milano.

Figura 13. *La segmentación panóptica como fusión de la segmentación semántica y la segmentación de instancias.*

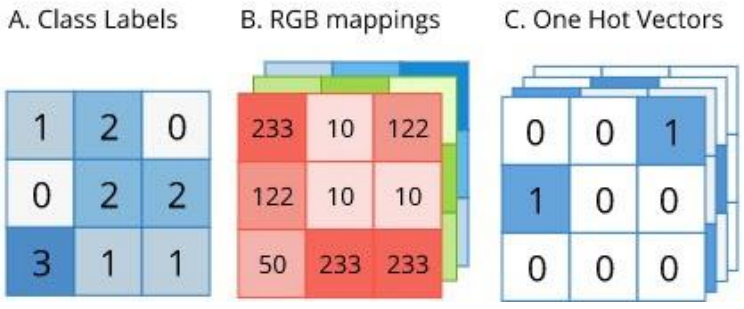


Nota. Adaptado de Cohen, J. (2023, 27 febrero). *Instance Segmentation: How adding masks improves object detection*. Think Autonomous. https://www.thinkautonomous.ai/blog/instance-segmentation/

El resultado de la segmentación es lo que se conoce como máscara de segmentación. Estas máscaras contienen una representación que asigna a cada píxel de una imagen una clase específica. En la Figura 13 se muestra la representación visual de máscaras de segmentación de distintas técnicas de una misma imagen.

Al igual que las imágenes en la segmentación se representan como matrices multidimensionales, las máscaras de segmentación también se representan como matrices bidimensionales de profundidad igual a uno o de profundidad igual al número de clases. Cuando son matrices bidimensionales de profundidad igual a uno contienen números enteros (Figura 14. A) correspondientes a las distintas clases segmentadas mientras que cuando tienen una profundidad igual al número de clases contienen ceros o unos codificando las clases en lo que se conoce como formato *one hot* (Figura 14. C). Este formato en lugar de asignar simplemente un entero a cada píxel utiliza un vector con un tamaño igual al número de clases posibles en el problema. Cada elemento del vector corresponde a una clase, y solo uno de ellos es 1, mientras que el resto son 0. El elemento que tiene el valor de 1 indica la clase a la que pertenece el píxel. Finalmente, para una representación visual, se pueden codificar las clases en formato RGB (Figura 14. B) asignando un color a cada clase. De esta manera, al representar visualmente la máscara como una imagen, se pueden distinguir las distintas clases por los colores.

Figura 14. *Tres representaciones de máscaras de segmentación: Enteros, RGB y One Hot*



Nota. Adaptado de Restrepo, R. (s. f.). *Inputs, labels, and outputs*. Ronny Restrepo. http://ronny.rest/tutorials/module/seg_01/segmentation_03_inputs_outputs/

Este trabajo se centrará principalmente en la segmentación semántica. Se pretende distinguir entre los distintos elementos que componen la vía urbana desde el punto de vista de un peatón para un futuro uso en dispositivos accesibles para la guía de personas con discapacidad visual. Por ello, no es necesario distinguir entre las instancias de los distintos elementos, únicamente es suficiente identificar y determinar la posición de las distintas categorías detectadas. Debido a esto, es más apropiada la segmentación semántica frente a la segmentación de instancias.

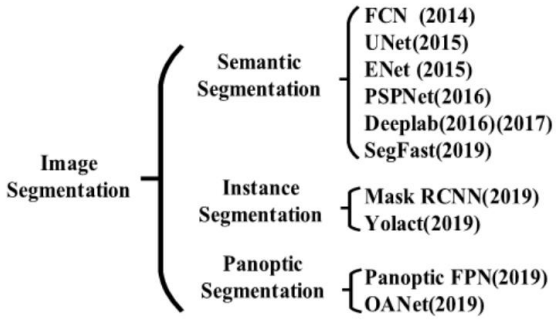
2.5.1. Segmentación de imágenes con redes neuronales convoluciones

Se han desarrollado numerosos algoritmos de segmentación de imágenes como por ejemplo corte de grafos (Boykov et al., 2001), la umbralización (Sezgin & Sankur, 2004) o la técnica de contornos activos (Kass et al., 1988). Sin embargo, estas técnicas iniciales de segmentación de imágenes quedaron prácticamente obsoletas para este propósito debido la evolución de las redes neuronales convolucionales para la segmentación con la llegada a partir de 2015 de redes convolucionales para la segmentación como las siguientes: red totalmente convolucional o *Fully Convolutional Network* (FCN) (Long et al., 2015), ResNet (He et al., 2016) y SegNet (Badrinarayanan et al., 2017).

Estas técnicas de aprendizaje profundo supervisado ya no requieren de conocimiento de técnicas de ingeniería de alto nivel al extraer la información directamente de los datos proporcionados a la vez que mantienen un rendimiento excepcional.

La principal limitación de las redes neuronales convolucionales profundas tradicionales en la segmentación de imágenes es que debido al uso de capas totalmente conectadas que aplanan los mapas de características obtenidos desaparece la información espacial degradando consecuentemente los modelos de segmentación basados en CNNs tradicionales. Para solucionar este problema, la arquitectura FCN fue pionera presentando un modelo que elimina las últimas capas totalmente conectadas remplazándolas por capas sucesivas de sobremuestreo (*upsampling*), en este caso capas de deconvoluciones o *transposed convolutions*, que aumentan la resolución de las características extraídas para recuperar el tamaño original de la imagen de entrada. Esto es lo que se conoce como arquitectura codificador-decodificador. El codificador se encarga de extraer las características de la entrada y el decodificador tiene como objetivo proporcionar una salida del mismo tamaño que la imagen original con la información extraída por el codificador.

Figura 15. *Aplicaciones de redes neuronales convolucionales en la segmentación de imágenes.*



Nota. Adaptado de Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems, 33*(12), 6999–7019. https://doi.org/10.1109/TNNLS.2021.3084827

3. Material y métodos

3.1. Herramientas

A lo largo de este Trabajo de Fin de Grado se han utilizado diversas herramientas tanto como para la creación del conjunto de datos utilizados como para el desarrollo del software que implementan las redes neuronales convolucionales utilizadas y su posterior análisis de eficiencia. A continuación, se explicarán en mayor profundidad las herramientas utilizadas para una mejor compresión del trabajo desarrollado.

3.1.1. Roboflow

Roboflow (Dwyer et al., 2024) es una herramienta online que permite a cualquier persona, sin importar sus habilidades o experiencia, construir aplicaciones de visión por computador. Abarca desde la creación del conjunto de datos para la segmentación hasta el entrenamiento de distintos modelos de visión por computadora tanto de clasificación, detección o segmentación y su posterior despliegue. Fue lanzado al público en enero de 2020 por la propia compañía Roboflow.

Esta herramienta, en este caso, se ha utilizado únicamente para la creación de la colección de datos a partir de sus características que permiten anotar imágenes y también su característica de exportación de la colección de datos permitiendo en este proceso formatear las imágenes e incluso aumentar el tamaño de la base de datos mediante distintas técnicas. Este proceso de creación de la base de datos se explicará en mayor profundidad en la siguiente sección.

Cuenta con una correspondiente biblioteca de Python. Esta biblioteca internamente realiza llamadas a la API propia de Roboflow proporcionando así métodos para interactuar con Roboflow desde el propio código Python.

3.1.2. Python

El procesamiento de las imágenes para su segmentación y el posterior análisis y comparación de los resultados obtenidos se ha realizado con el lenguaje de programación Python (Van Rossum & Drake, 1995). Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica. Una de sus principales características es su legibilidad lograda con una sintaxis simple y fácil de aprender.

Fue inventado a principio de la década de los 90 por Guido van Rossum como sucesor del lenguaje de programación ABC. Su primer lanzamiento fue en 1991 como Python 0.9.0. Desde entonces se ha ido revisando y lanzando nuevas versiones como Python 2.0 en 2000 y Python 3.0 en 2008. La versión de Python utilizada en el desarrollo de este trabajo es la versión 3.10.12. Actualmente es mantenido por Python Software Foundation, una organización sin ánimo de lucro y comunidad dedicada a desarrollar, mejorar y popularizar el lenguaje de programación Python y su entorno. Este último objetivo ha llevado a Python a ser el lenguaje de programación más popular en la actualidad según el índice de la comunidad de programación TIOBE (*TIOBE Index - TIOBE*, s. f.).

Todas las versiones de Python son *Open Source* o código abierto, es decir, su código fuente es accesible al público de manera gratis para poder verlo, modificarlo y distribuirlo como consideren conveniente.

Python cuenta con numerosas bibliotecas y frameworks que facilitan el desarrollo de una gran variedad de aplicaciones, desde el análisis de datos y aprendizaje profundo hasta el

desarrollo web. Destaca especialmente frente a otros lenguajes en el ámbito de la ciencia de datos e inteligencia artificial contando con bibliotecas especializadas ampliamente utilizadas como TensorFlow (Martín et al., 2015), Keras (Chollet & otros, 2015), PyTorch (Paszke et al., 2019) y Pandas (The pandas development team, 2024).

3.1.3. Google Colaboratory

Google Colaboratory, también conocido como Google Colab o simplemente por Colab, es un servicio alojado de notebooks de Jupyter. Estos notebooks son documentos que combinan tanto datos, código, resultados del códigos y texto explicativo. Google Colaboratory ofrece acceso a recursos informáticos como GPUs y TPUs y permite ejecutar código de Python o R desde el propio navegador al hacer uso de máquinas virtuales. No es necesario instalar ningún software en la propia máquina desde la que se utiliza para que funcione.

En ambas versiones los notebooks creados con Google Colaboratory se almacenan en Drive, la nube de Google, haciéndolos accesibles desde cualquier ordenador y pudiendo ser compartidos con otras personas para un trabajo colaborativo.

Tiene tanto versión gratuita como de pago. La versión gratuita ofrece recursos limitados y no siempre están garantizados ya sea por no disponibilidad o por llegar a un tiempo máximo de uso. Además, las versiones de pago ofrecen acceso a GPUs premium de mejores características.

Es ideal para proyectos de aprendizaje automático y de ciencia de datos como este al ofrecer acceso a GPUs y TPUs ya que al entrenar modelos de aprendizaje automático se requiere una potencia computacional significativa que un ordenador habitual no posee. Estos recursos informáticos reducen significativamente el tiempo de entrenamiento de modelos de aprendizaje automático y profundo y permiten manejar modelos más grandes y con mayor cantidad de datos en comparación con el uso de una CPU.

3.1.4. Bibliotecas

Para el desarrollo de este trabajo se han utilizado varias bibliotecas de Python como se mencionarán a continuación:

- TensorFlow (Martín et al., 2015): es una biblioteca de código abierto desarrollada por Google que facilita la creación e implementación de modelos de aprendizaje automático. Puede ser utilizada para una gran variedad de tareas. Proporciona una API de alto nivel denominada Keras (Chollet & otros, 2015) que facilita la introducción a TensorFlow al tener las siguientes ventajas: amigable al usuario, modular y configurable y fácil de extender.
- Scikit-learn (Pedregosa et al., 2011): también conocida como sklearn, es una biblioteca de Python de código abierto que contiene herramientas para el análisis predictivo de datos integrando una gran variedad de algoritmos de aprendizaje automático para problemas supervisados y no supervisados. En este trabajo se ha utilizado su módulo sklearn.metrics que proporciona métricas de evaluación de modelos de aprendizaje automático.
- ImageIO (Klein et al., 2024) y Pillow (PIL) (Lundh & Clark, 2014): son dos bibliotecas de Python que proporcionan una interfaz simple para el procesamiento de imágenes, proporcionando ImageIO también la opción para otros formatos de datos visuales como vídeos.
- Numpy (Harris et al., 2020): es el principal paquete de Python para el cálculo científico. Proporciona una representación de vectores multidimensionales y objetos derivados además de proporcionar distintas operaciones para su manejo como funciones de filtro, matemáticas, lógicas, de selección y otras muchas más.

- Matplotlib (Hunter, 2007): es una biblioteca para la creación estática, animada o interactiva de visualizaciones en Python. De esta biblioteca se utiliza el módulo pyplot para la representación de distintos gráficos e imágenes.

3.2. Descripción del conjunto de datos

El conjunto de datos utilizado en este trabajo consiste en 2481 imágenes de la vía urbana desde el punto de vista de un peatón con un ángulo ligeramente inclinado apuntando hacía el suelo con el objetivo de enfocarse exclusivamente en los diferentes elementos y obstáculos que componen el entorno peatonal y consecuentemente, sus correspondientes máscaras de segmentación al tratarse de un enfoque de aprendizaje supervisado. Estas imágenes se han obtenido de varios vídeos grabados en la ciudad de Salamanca por la autora de este trabajo. Los vídeos fueron capturados específicamente para este propósito debido a la falta de conjuntos de datos que cumpliesen con las características necesarias: el punto de vista de un peatón y una vía urbana con similares características a las encontradas en las calles de las ciudades españolas. Esta estrategia asegura que los datos sean más representativos para entornos urbanos similares a los de Salamanca y cubre una necesidad previamente no abordada en la disponibilidad de conjuntos de datos con estas características específicas.

Las imágenes obtenidas se han anotado con la herramienta explicada anteriormente Roboflow. Este proceso de anotación consiste en asignar manualmente a cada píxel una clase específica para obtener como resultado la máscara de segmentación correspondiente. Las distintas clases que se han segmentado son: acera, baldosas podotáctiles, banco, carretera, pared, paso de cebra y vegetación. A estas clases se añade automáticamente por Roboflow la clase fondo que corresponde con aquellos pixeles que no han sido asignados ninguna clase. Inicialmente también se anotó escaleras de bajada y escaleras de subida, pero debido a la falta de imágenes con estos elementos se decidió eliminar ya que resultaba un modelo muy poco eficiente para estas clases.

Roboflow facilita este proceso de anotación mediante un editor propio. En este editor se selecciona las distintas áreas de la imagen que corresponden con las clases que se quieren segmentar y se asigna la clase específica a la que pertenece cada área. Esta selección se puede hacer manualmente o con el asistente inteligente que proporciona Roboflow. La siguiente figura muestra un ejemplo de una imagen anotada. Se puede observar cómo tanto la pared, como la acera y la carrera están delimitados por un polígono cada uno de un color y cada color corresponde con su clase asignada cuya leyenda se puede ver a la izquierda.

Figura 16. *Ejemplo de imagen anotada para la segmentación en Roboflow*

Las imágenes y sus correspondientes máscaras de segmentación se han dividido en tres grupos de manera aleatoria por Roboflow:

- Entrenamiento: conjunto de datos utilizado para entrenar el modelo de aprendizaje profundo para que se ajusten los parámetros que se deben aprender a estos datos. Para este grupo se han seleccionado un 70% de las imágenes.
- Validación: conjunto de datos utilizado para evaluar el modelo ajustado a lo largo del entrenamiento. Para este grupo se han seleccionado un 20% de las imágenes.
- Prueba: conjunto de datos utilizado para evaluar el ajuste del modelo final obtenido del entrenamiento. Para este grupo se han seleccionado un 10% de las imágenes.

Para aumentar la robusticidad del conjunto de datos y reducir la posibilidad de sobreajuste, se han aplicado distintas técnicas de aumento de datos o *Data Augmentation* que ofrece Roboflow. Esta colección de técnicas permite mejorar el tamaño y calidad de un conjunto de datos para como resultado obtener modelos de aprendizaje profundo más eficientes. Para el conjunto de datos utilizado en este trabajo se han utilizado las técnicas de modificar la exposición y el brillo de las imágenes entre un -25% y un 25%. Por cada imagen original del conjunto de entrenamiento se han obtenido dos imágenes más, a parte de la original, generada con una de las técnicas mencionadas aleatoriamente.

De esta manera, el conjunto de imágenes final es el siguiente: 5217 imágenes de entrenamiento, 495 imágenes de validación y 247 imágenes de prueba.

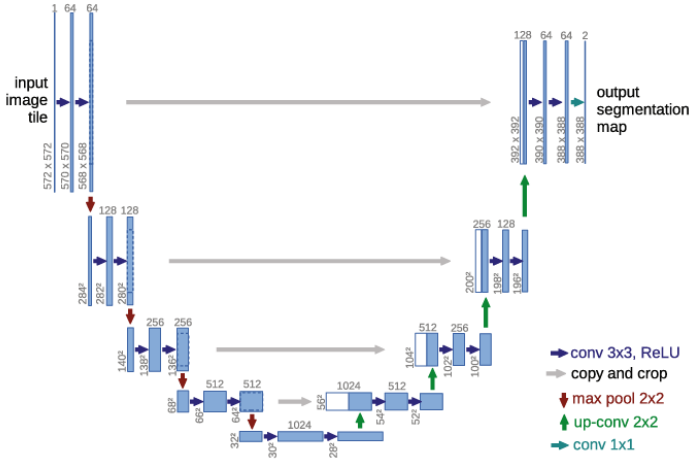
3.3. Arquitecturas CNN

A continuación, se describen las arquitecturas de redes neuronales convolucionales utilizadas.

3.3.1. Arquitectura U-Net

En 2015, Ronneberger et al. propusieron el modelo de red neuronal convolucional U-Net inspirado en la arquitectura de las redes totalmente convolucionales (FCN). Esta arquitectura modifica la arquitectura FCN para que funcione con un menor conjunto de imágenes de entrenamiento, pero manteniendo una segmentación precisa. Fue diseñada específicamente para la segmentación de imágenes médicas; sin embargo, se ha mostrado su funcionamiento en otras áreas de visión por computadora (Abderrahim et al., 2020).

Figura 17. *Ejemplo de arquitectura U-Net. Las cajas azules corresponden con los mapas de características cuyo número se específica encima. El ancho y alto se especifica al lado de la esquina inferior izquierda. Las cajas blancas son los mapas de características copiados. Las flechas corresponden con las distintas operaciones.*

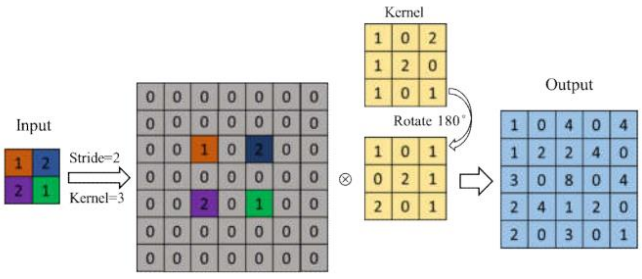


Nota. Adaptado de Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9351). https://doi.org/10.1007/978-3-319-24574-4_28

Antes de comenzar a desarrollar la arquitectura U-Net, se procede a explicar dos conceptos no explicados anteriormente que emplea esta red neuronal: la convolución transpuesta y las conexiones de salto.

El objetivo de la operación de convolución transpuesta o *up-convolution* es transformar la dimensión de la salida de una capa de convolución para recuperar la dimensión de la entrada, ya que la capa de convolución reduce las dimensiones de la entrada, a la vez que se mantiene la información de las características extraídas. Al igual que la operación de convolución tiene como parámetros el tamaño del filtro (k), el relleno de ceros o padding (p), y el paso o stride (s). Estos parámetros suelen ser iguales a los de la operación que se quiere revocar y al igual que en la convolución el contenido del filtro son los parámetros que deben ser aprendidos en el entrenamiento. Para implementar una operación de convolución transpuesta, primero se deben calcular los siguientes parámetros adicionales: $z = s - 1$ y $p' = k - p - 1$. Luego, se insertan z ceros entre los distintos elementos de la matriz de entrada para expandirla. A continuación, se añade un *padding* igual al valor del parámetro k. Después, el filtro se rota 180 grados. Finalmente, se realiza la operación de convolución tradicional con la entrada modificada, el filtro rotado y con un paso de 1 (Y. Zhang et al., 2019).

Figura 18. *Ejemplo de una operación de convolución transpuesta con paso 2, filtro 3x3 y relleno de ceros 0.*



Nota. Adaptado de Zhang, Y., Zhao, X., & Liu, P. (2019). Multi-Point Displacement Monitoring Based on Full Convolutional Neural Network and Smartphone. *IEEE Access, 7*, 139628–139634. https://doi.org/10.1109/ACCESS.2019.2943599

Este proceso asegura que la dimensionalidad de la salida se expanda. Considerando un mapa de características de tamaño $I_h \times I_w$, un tamaño de filtro de $K_h \times K_w$, un paso de s y un *padding* de p, las dimensiones de la salida serán las siguientes (Dumoulin & Visin, 2016):

$$O_h = (I_h - 1) \times s + K_h - 2p$$
$$O_w = (I_w - 1) \times s + K_w - 2p$$

El segundo concepto principal son las conexiones de salto o de atajo (*skip connections)*. Si en una arquitectura codificador-decodificador simplemente se encadenan las capas que las componen sin ninguna relación entre ambas partes, se podría perder información sobre la exacta localización de los contornos de las clases segmentadas. Para evitar esta pérdida, se introducen las conexiones de salto (Drozdzal et al., 2016). Estas conexiones combinan, mediante la suma o concatenación, las salidas intermedias del codificador con la entrada de la etapa correspondiente del decodificador (Gupta, s. f.).

Ahora que se han explicado estos conceptos se procede a explicar la arquitectura del modelo U-Net. Esta red neuronal está compuesta por dos partes: la ruta de contracción, también llamada codificador y la ruta expansiva simétrica, también llamada decodificador. La arquitectura resultante, que se puede observar en la Figura 17, es casi simétrica con forma de "U", recibiendo de ahí su nombre.

La primera parte, la ruta de contracción, emplea múltiples bloques convolucionales junto con operaciones de *downsampling* (submuestreo), como son las operaciones de *pooling*, para extraer las características semánticas y contextuales. En concreto, el codificador está compuesto por la repetida aplicación de las siguientes operaciones: dos convoluciones de filtro 3x3 y sin *padding*, seguidas por una Unidad Rectificada Uniforme (ReLU) y una operación de *max-pooling* 2x2 para el submuestreo. En cada una de estas aplicaciones se reduce la resolución de los mapas de características, pero se duplica el número de mapas.

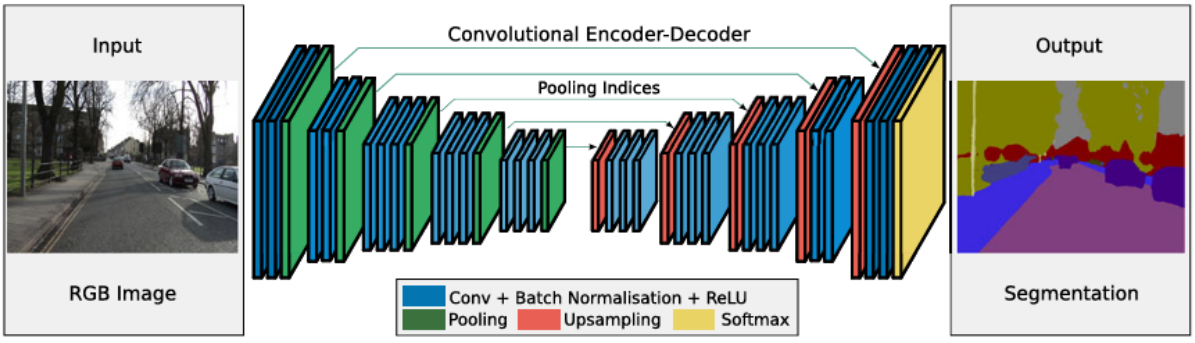
La segunda parte, la ruta expansiva simétrica, procede con el sobremuestreo con una convolución transpuesta 2x2 que reduce el número de mapas de características a la mitad a la vez que recupera poco a poco la resolución de los mapas. Continua con la concatenación de los mapas de características de la correspondiente etapa del codificador, pero, recortados para obviar la pérdida de información de los pixeles de los bordes, y de dos convoluciones 3x3 seguidas de una ReLU. Estas operaciones se repiten en varias etapas para recuperar gradualmente la resolución.

En la etapa final se termina con una capa convolucional 1x1 adicional para reducir el número de mapas de características a un valor igual al número de clases que se desean segmentar. En total U-Net cuenta con 23 capas convolucionales (teniendo en cuenta también las capas convolucionales transpuestas).

3.3.2. Arquitectura SegNet

La arquitectura SegNet fue presentada en 2015 por Badrinarayanan et al.. Esta arquitectura se basa en la arquitectura de las redes totalmente convolucionales (FCN). Su creación está motivada principalmente por aplicaciones de reconocimiento del entorno vial que necesitan tener la capacidad de modelar tanto la apariencia, forma y el contexto entre las distintas clases sin verse perjudicada por aquellas clases de pequeño tamaño ni por el tiempo de procesamiento.

Figura 19. *Arquitectura SegNet*



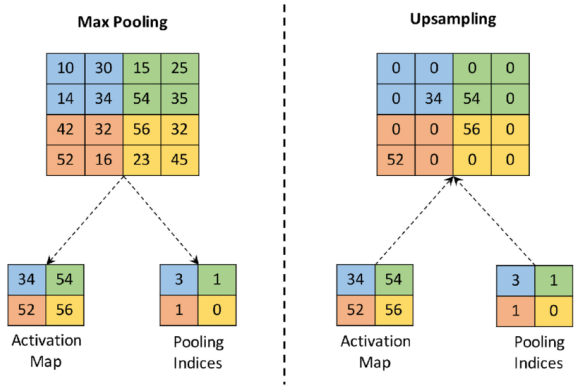
Nota. Adaptado de Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39*(12), 2481–2495. https://doi.org/10.1109/TPAMI.2016.2644615

SegNet presenta una arquitectura de codificador-decodificador ilustrada en la Figura 19. El codificador es idéntico a las capas convolucionales del modelo VGG16 (Simonyan & Zisserman, 2014). Está formado por 13 capas convolucionales. Cada codificador tiene un decodificador correspondiente, por tanto, el decodificador también está compuesto por 13 capas convolucionales. La salida del último decodificador se pasa a una función de activación softmax, también llamada función exponencial normalizada, que actúa como clasificador de clases produciendo la probabilidad de cada píxel para cada clase.

Cada decodificador de la red ejecuta una operación de convolución que produce un conjunto de mapas de características que a su salida son normalizados por lotes, seguidos de la aplicación de una Unidad Rectificada Uniforme (ReLU). A continuación, se aplica una capa de *max-pooling* con un tamaño de ventana 2x2 y con un paso de 2 de forma que la salida se reduce a la mitad. Esta operación introduce invariación frente a pequeños cambios locales en la entrada; sin embargo, en la segmentación se debe mantener los detalles de los contornos de las clases. Por ello, antes de proceder con el *downsampling* mediante la operación de *max-pooling,* para memorizar parte de esta información antes de que se pierda, pero sin utilizar demasiados recursos computacionales, se almacenan los índices de los valores de *max-pooling,* es decir, las ubicaciones con los valores máximos dentro de la ventana de agrupación (Figura 20). Esta es la innovación que introduce SegNet.

Los decodificadores reciben de su codificador correspondiente los índices de las operaciones de *max-pooling* para realizar las operaciones de sobremuestreo sobre los mapas de características obtenidos con estos índices como se muestra en la Figura 20. El uso de estos índices ha mostrado una mejora en la delineación de los contornos de las clases además de reducir el número de parámetros a aprender al no requerir esta estrategia aprendizaje. A continuación, al igual que en el codificador, se aplica una operación de convolución, seguida de una normación por lotes de los mapas de características y de una aplicación de una ReLU.

Figura 20. *Ejemplo de una operación de max-pooling y de sobremuestreo (upsampling) almacenando y recuperando correspondientemente los índices de la agrupación.*



Nota. Adaptado de Gonçalves, D. N., Weber, V. A. de M., Pistori, J. G. B., Gomes, R. da C., de Araujo, A. V., Pereira, M. F., Gonçalves, W. N., & Pistori, H. (2021). Carcass image segmentation using CNN-based methods. *Information Processing in Agriculture, 8*(4), 560–572. https://doi.org/https://doi.org/10.1016/j.inpa.2020.11.004

3.3.3. Arquitectura DeepLabv3+

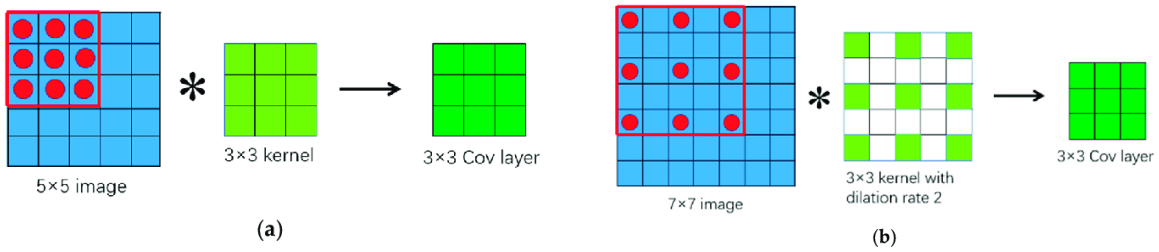
DeepLabv3+ es una arquitectura de red neuronal presentada por Chen et al. en 2018. Esta arquitectura presenta mejoras sobre sus predecesoras DeepLabv1 (Chen et al., 2014), DeepLabv2 (Chen, Papandreou, Kokkinos, et al., 2017), DeepLabv3 (Chen, Papandreou, Schroff, et al., 2017).

La primera versión introdujo las convoluciones dilatas (Figura 21), también conocidas como convoluciones *atrous* que proviene de la palabra francesa “à trous” que significa agujero, presentadas por primera vez por Mallat (1999). La salida $y[i]$ de esta capa para una entrada unidimensional $x[i]$ con un filtro $w[i]$ de tamaño K es (Papandreou et al., 2015):

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k] \cdot w[k]$$

Esta fórmula corresponde con una convolución estándar, pero introduciendo un nuevo parámetro de tasa de dilatación o *rate* (r) que indica el número de celdas que se debe desplazar el filtro al tomar las entradas por cada ventana. Una convolución dilatada con una tasa de dilatación igual a 1 es igual a operación de convolución tradicional. Esta operación permite aumentar el campo de visión de los filtros incorporando así más contexto, pero manteniendo una localización precisa. Se evita así una extensa reducción de la resolución y consecuentemente su pérdida de información.

Figura 21. *Comparativa de un ejemplo de convolución tradicional (a) y un ejemplo de convolución dilatada (b).*

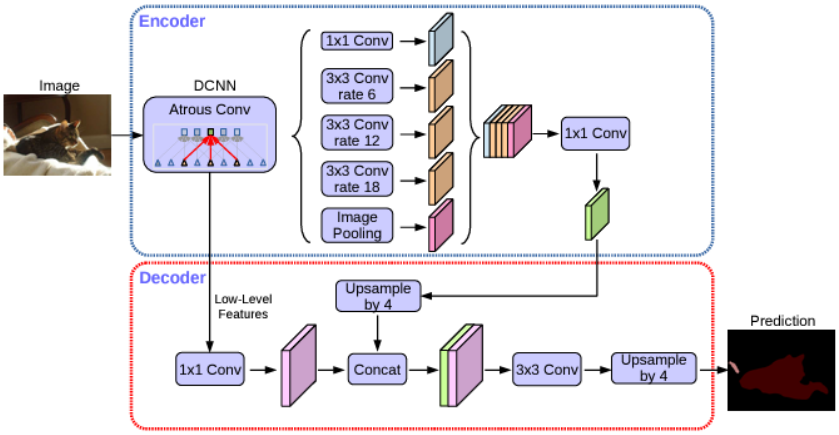


Nota. Adaptado de Si, Y., Gong, D., Guo, Y., Zhu, X., Huang, Q., Evans, J., He, S., & Sun, Y. (2021). An advanced spectral–spatial classification framework for hyperspectral imagery based on DeepLab v3+. *Applied Sciences, 11*(12), 5703.

La siguiente versión de DeepLab, DeepLabv2, introdujo el módulo *Atrous Spatial Pyramid Pooling* (ASPP). Este módulo permite el remuestreo de los mapas de características a distintas tasas de dilatación mediante el uso de múltiples capas paralelas de convolución dilatada. DeepLabv3 incorpora a esto una operación adicional de *pooling* por media global que realizar paralelamente además de una capa convolucional 1x1 tras concatenar las salidas. También incorpora la normalización por lotes.

DeepLabv3+, además de mejorar los conceptos mencionados anteriormente que incorporan las arquitecturas predecesoras, introduce una estructura codificador-decodificador. Los autores de DeepLabv3+ proponen utilizar DeepLabv3 como el codificador y añaden un módulo decodificador simple pero efectivo para obtener segmentaciones más precisas. La arquitectura de DeepLabv3+ se puede observar en la Figura 22.

Figura 22. *Arquitectura de DeepLabv3*



Nota. Adaptado de Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Proceedings of the European Conference on Computer Vision (ECCV),* 801–818.

El codificador, que corresponde con DeepLabv3, se inicia a partir de una red neuronal convolucional profunda preexistente y preentrenada generalmente diseñadas originalmente para la clasificación de imágenes. Algunos ejemplos son VGG (Simonyan & Zisserman, 2014), ResNet (He et al., 2016) y Xception (Chollet, 2017). En este trabajo se utilizará ResNet. Después de que la CNN haya extraído las características antes de la última capa de *downsampling* de la correspondiente CNN se continúa aplicando el módulo ASPP compuesto por una convolución 1x1, una convolución dilatada 3x3 con tasa de dilatación 6, una convolución dilatada 3x3 con tasa de dilatación 12, una convolución dilatada 3x3 con tasa de dilatación 18 y la operación de *average pooling*. Todas estas convoluciones utilizas 256 filtros y una normalización por lotes. La salida de estas capas paralelas se concatena y se pasa por otra convolución 1x1. Esto permite combinar las distintas salidas en una representación más uniforme de la imagen. De esta manera, es posible analizar la imagen a distintas escalas y con distintos filtros mejorando consecuentemente tanto la información contextual como la localización.

La salida del codificador son 256 mapas de características con una relación entre la resolución espacial de la imagen de entrada y la resolución de salida final de 16. Por ello, primero el decodificador realiza una operación de sobremuestreo (*upsampling)* bilineal con un factor de 4. Esta operación consiste en calcular los nuevos pixeles generados usando los píxeles cercanos mediante interpolaciones lineales. La salida de esta operación se concatena con la salida de la CNN utilizada que tiene las mismas dimensiones pasada antes por una convolución 1x1 para reducir el número de filtros ya que suelen contener un número demasiado alto que podría restar importancia a las características extraídas por el codificador. Tras la concatenación, se procede con una capa convolucional 3x3 y de seguido una capa de sobremuestreo bilineal con un factor de 4.

3.4. Hiperparámetros

Para tratar de optimizar el entrenamiento de los modelos de segmentación seleccionados, se experimenta con distintos hiperparámetros claves para mejorar su rendimiento y precisión. Estos hiperparámetros se deben especificar manualmente antes de que comience el entrenamiento a diferencia de los pesos sinápticos que son aprendidos a lo largo del proceso de entrenamiento.

Los hiperparámetros que propone este trabajo para optimizar son los siguientes:

- Número de épocas *(max epochs)* que se entrena el conjunto de entrenamiento.
- Función de pérdida del aprendizaje.
- Algoritmo de optimización del aprendizaje.

Las funciones de pérdida seleccionadas para experimentar por el presente trabajo son las siguientes:

- La entropía cruzada (Yi-de et al., 2004) se define como una medida que cuantifica las discrepancias entre dos distribuciones de probabilidad para una variable aleatoria específica o un conjunto de eventos. Es la función de pérdida más comúnmente utilizada en las aplicaciones de segmentación (Jadon, 2020). Para la segmentación de múltiples clases, se calcula de la siguiente manera siendo y_i la máscara real e $\hat{y}_i$ la máscara predicha de la clase i y n el número de clases (Koech, 2020):

$$L_{CE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

- La pérdida focal (Lin et al., 2017) se inspira en la entropía cruzada proponiendo una mejora para funcionar mejor en conjuntos de datos con clases extremadamente desequilibradas. Reduce la contribución de los ejemplos fáciles y permite que el modelo se concentre más en aprender de los ejemplos difíciles. Esta técnica se puede implementar añadiendo los siguientes dos factores a la técnica de entropía cruzada:
  - Un factor de balanceo (α) para ajustar la importancia relativa de cada clase para ajustar el desbalance. Este valor suele ser la frecuencia inversa de la clase o puede establecerse por validación cruzada.
  - Un factor de modulación (γ) que ajusta la contribución de los distintos ejemplos. Si una clase ha sido fácil de clasificar, su probabilidad estimada será cercana a 1 y consecuentemente el factor de modulación será cercano a 0. Lo contrario sucede con los ejemplos difíciles.

$$FL = -\sum_{i=1}^{n} \alpha_i (1 - p_i)^{\gamma} y_i \log(\hat{y}_i)$$

Los algoritmos de optimización seleccionados para experimentar por el presente trabajo son los siguientes:

- Descenso de gradiente estocástico (SGD) (Robbins & Monro, 1951): es el método de optimización más utilizado en modelos de aprendizaje automático (Hardt et al., 2016). Es una variante del método de descenso de gradiente. El método SGD se basa en minimizar el riesgo empírico de un modelo calculando reiteradamente el gradiente de la función de pérdida de un subconjunto de datos aleatorios del entrenamiento. Considerando cada ejemplo z como una pareja (x, y) siendo x la entrada e y la salida, la función de pérdida $l(\hat{y}, y)$ y dP una distribución de probabilidad desconocida que caracteriza el problema que aprender, se busca encontrar la función f que minimice la pérdida $Q(z, w) = l(f_w(x), y)$. El riesgo empírico viene dado por la siguiente función:

$$E(f) = \int l(f(x), y) dP(z) \qquad E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i)$$

Para minimizar este riesgo el descenso de gradiente estocástico propone actualizar los pesos w cada iteración en base al gradiente del error empírico de una única muestra z_t seleccionada aleatoriamente, en lugar de usar el conjunto completo de datos como en el descenso del gradiente. Se modifican los pesos de la siguiente forma hasta la convergencia, siendo γ la ganancia elegida de manera adecuada:

$$w_{t+1} = w_t - \gamma_t \nabla_w Q(z_t, w_t)$$

(Bottou, 2010)

- Algoritmo Adam (*Adaptive moment estimation)* (Kingma & Ba, 2015): esta técnica combina los métodos de optimización AdaGrad y RMSProp también basados en el descenso de gradiente. Su innovación se debe a que calcula tasas de aprendizaje adaptativas específicas para cada parámetro a partir de estimaciones de los momentos de primer y segundo momento de los gradientes. Sus pasos son los siguientes:
    - Adam inicializa dos vectores m y v que almacenarán la media de los gradientes y la media de los gradientes al cuadrado a lo largo del tiempo correspondientemente. También se inicializa el contador de actualizaciones t a 0.
    - Se calculan los gradientes g_t en la iteración t en función de los pesos w_t del modelo y siendo f la función que se desea optimizar:

$$g_t = \nabla_\theta f_t(w_t - 1)$$

    - Se actualiza el vector de primero momento m con la suma del valor anterior de m ponderado por la tasa de decaimiento exponencial para las estimaciones del primer momento, parámetro β_1 generalmente con valor 0,9, y el nuevo gradiente ponderado por $1 - \beta_1$:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

    - Se actualiza el vector de segundo momento v con la suma del valor anterior de v, que es el pasado gradiente al cuadrado, ponderado por la tasa de decaimiento exponencial para las estimaciones del segundo momento, parámetro β_2 generalmente con valor 0,999, y el nuevo gradiente al cuadrado ponderado por $1 - \beta_2$:

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

    - Se corrige el sesgo en los momentos con su tasa de decaimiento correspondiente de la siguiente forma:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

    - Por último, se actualizan los pesos para minimizar la función de pérdida de la siguiente manera siendo α la tasa de aprendizaje y ϵ (épsilon) un número muy pequeño como 10^{-8}:

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

La tasa de aprendizaje, que consiste en la magnitud en la que se actualizan los pesos en el proceso de optimización, se utilizará la proporcionada por defecto por Keras en sus

funciones de optimización. En el algoritmo SGD es 0,01 y en el algoritmo Adam es 0,001. Este parámetro solo se modificará en caso de un notable mal ajuste.

3.5. Métricas

Para evaluar y comparar la efectividad de distintos modelos de segmentación es necesario criterios de evaluación que cuantifiquen su precisión. Desde los inicios de la segmentación semántica mediante el aprendizaje profundo, se han propuesto numerosas métricas estando la mayoría relacionadas con la precisión a nivel de píxel sin tener en cuenta el tiempo de cálculo, el almacenamiento de memoria requerido o la calidad visual de la imagen obtenida (Minaee et al., 2022).

Se debe considerar TP *(True Positives)* como el número de píxeles de una clase predichos como positivos correctamente por el modelo, FP *(False Positives)* como el número de píxeles de una clase predichos como positivos incorrectamente por el modelo al ser en verdad píxeles negativos, TN *(True Negatives)* como el número de píxeles de una clase predichos como negativos correctamente por el modelo y FN *(False Negatives)* como el número de píxeles de una clase predichos como negativos incorrectamente por el modelo al ser en verdad píxeles positivos. Estos valores vienen representados en lo llamado matriz de confusión (Figura 23) que permite una visualización del rendimiento del modelo.

Los falsos positivos corresponden a un error de tipo I al clasificar erróneamente a un píxel negativo como positivo. Por otro lago, los falsos negativos corresponden a un error de tipo II al no reconocer un píxel positivo clasificándolo como negativo. Estos errores son cruciales ya que determinan si un modelo es demasiado permisivo, es decir, si tiene un alto error de tipo I, o si un modelo es demasiado restrictivo, es decir, si tiene un alto error de tipo II.

Figura 23. *Representación de una matriz de confusión*

Nota. Adaptado de González, L. (2019). *Matriz de Confusión - Aprende IA*. https://aprendeia.com/matriz-de-confusion-machine-learning/

A continuación, se describen las métricas más populares en la literatura científica que serán utilizadas para la evaluación de los modelos implementados en el presente trabajo.

- *Accuracy* (exactitud) por píxel: indica cuántas veces acierta el modelo. Es el porcentaje de píxeles correctamente clasificados. No es adecuada para conjuntos de datos con clases desequilibradas.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

- Precisión: es la relación entre los píxeles correctamente clasificados como positivos y todos los píxeles clasificados como positivos. Mide la calidad de las predicciones hechas por el modelo.

$$Precision = \frac{TP}{TP + FP}$$

- *Recall* (Exhaustividad): indica cuanto de los positivos reales han sido capturados por el modelo como positivos.

$$Recall\ = \frac{TP}{TP + FN}$$

- IoU (*intersection-over-union)* o índice Jaccard: es el área de superposición entre la máscara predicha y la máscara real dividida por el área de unión entre ambas máscaras. Es ampliamente utilizada.

$$IoU\ = \frac{|A \cap B|}{|A \cup B|} = \frac{TP}{TP + FP + FN}$$

- El coeficiente de Dice o *F1-score*: es el área de superposición entre la máscara predicha y la máscara real dividida por el total de píxeles predichos entre ambas máscaras. Es ampliamente utilizada.

$$Dice\ = \frac{2\,|A \cap B|}{|A| + |B|} = \frac{2TP}{2TP + FP + FN}$$

Las métricas descritas pueden ser métricas suaves (*soft metrics)* si se calculan con las probabilidades predichas a partir de la función softmax o pueden ser métricas duras (*hard metrics)* si se calculan con las predicciones categóricas. Además, a excepción de la métrica *accuracy,* el resto de las métricas se pueden calcular tanto por cada clase, globalmente *(micro)*, como la media de las clases *(macro)* o como la media de las clases ponderada *(weighted)*.

En todos los casos, el rango de valores de las métricas será [0,1]. Cuanto más se acerquen las métricas a 1 mejor será el modelo.

4. Implementación

4.1. Configuración del entorno de desarrollo

Entrenar modelos de aprendizaje profundo como son las arquitecturas de redes neuronales convolucionales explicadas anteriormente requiere de altos recursos computacionales que un ordenador estándar no posee. Por ello, se usa la herramienta Google Colaboratory que proporciona un entorno de trabajo en la nube con acceso a hardware acelerado. A continuación, se detallas las características del hardware utilizado para el entrenamiento y predicción de los modelos:

- Unidad de Procesamiento Gráfico (GPU):
  - Modelo: NVIDIA L4
  - Memoria GPU: Aproximadamente 23 GB
  - Versión de CUDA: 12.2
- Unidad Central de Procesamiento (CPU):
  - Arquitectura: x86_64
  - Modelo: Intel(R) Xeon(R) CPU @ 2.20GHz
  - Número de CPUs: 12 (6 núcleos físicos con 2 hilos por núcleo)
  - Velocidad: 2.20GHz
- Memoria RAM:
  - Total: 52 GiB
- Almacenamiento en Disco:
  - Sistema de archivos principal: 202 GB de los cuales 28 GB están en uso y 175 GB están disponibles.
  - Dispositivo de almacenamiento: 242 GB de los cuales 82 GB están en uso y 161 GB están disponibles.

4.2. Preprocesamiento de los datos

Para poder comparar los distintos modelos se han utilizado los mismos datos con el mismo procesamiento para que estén en igual de condiciones.

En Roboflow ya se divide el conjunto de datos en tres grupos de entrenamiento, validación y prueba además de aumentar el conjunto de datos con técnicas de aumento de datos como se describió anteriormente. En el cuaderno de Google Colaboratory se continua con el preprocesamiento realizando sobre el conjunto de datos las siguientes operaciones:

- Corrección del valor entero que corresponde con cada clase: Inicialmente, debido a que se consideraron dos clases adicionales (escaleras de subida y escaleras de bajada), al eliminarlas y extraer el conjunto de datos de Roboflow los valores enteros correspondientes con cada clase quedaron descuadrados siendo asignadas por Roboflow de la siguiente manera: [0, 1, 2, 3, 4, 7, 8, 9]. Este hueco en la secuencia se corrigió sustituyendo la clase 7 por la clase 5, la clase 8 por la clase 6 y la clase 9 por la clase 7.
- Redimensionamiento de las imágenes y conversión en matrices: se convierten las imágenes a formato RGB para asegurar que se mantienen los canales de color, se redimensionan todas las imágenes al mismo alto y ancho y se almacenan como matrices para poder procesarlas en el entrenamiento. En el modelo U-Net y SegNet hay un paso adicional de normalización de los píxeles mediante la división entre 255 (valor máximo de un canal de color RGB). Esta normalización no se realiza para DeepLabv3+ ya que el propio modelo preprocesa lo datos normalizándolos. Para la arquitectura DeepLabv3+ se realiza un paso de adicional para convertir las matrices en objetos Dataset de Tensorflow.

- Redimensionamiento de las máscaras y conversión en matrices: se redimensionan todas las máscaras al mismo alto y ancho y se almacenan como matrices para poder procesarlas en el entrenamiento. Las máscaras no hay que convertirlas a formato RGB ya que son imágenes de un solo canal de color, están representadas mediante una escala de grises. Para la arquitectura DeepLabv3+ se realiza un paso de adicional para convertir las matrices en objetos Dataset de Tensorflow.

4.3. Implementación de las CNNs seleccionadas

En este trabajo se han implementado las redes neuronales convolucionales explicadas, U-Net, SegNet y DeepLabv3+, respetando en la medida de lo posible como fueron presentadas en sus respectivos artículos originales. No se ha tratado de optimizar la estructura de estas arquitecturas debido a la complejidad que implica dicho proceso y la necesidad de un conocimiento mucho más profundo de la estructura del modelo.

Esta implementación se ha realizado con la API Keras que proporciona TensorFlow. Las funciones utilizadas para conformar las distintas capas que conforman las distintas arquitecturas son las siguientes:

- Conv2D: capa convolucional
- Activation: capa de función de activación
- BatchNormalization: capa de normalización de lotes
- MaxPooling2D: capa de agrupación mediante *max-pooling*
- AveragePooling2D: capa de agrupación mediante *average pooling*
- Conv2DTranspose: capa de convolución transpuesta
- Concatenate: función de concatenación de capas
- UpSampling2D: capa de sobremuestreo

4.4. Entrenamiento y ajuste de hiperparámetros

Una vez las arquitecturas están construidas con Keras se puede proceder a su compilación y entrenamiento. El método *compile* de Keras permite configurar el modelo para su posterior entrenamiento. Es en este paso donde se especifican el algoritmo de optimización, la función de pérdida y las métricas que se van a utilizar.

Una vez establecidos estos hiperparámetros, se puede proceder con el entrenamiento con el método *fit*. En este método es donde se debe especificar el número de épocas y el tamaño del lote que se utilizará.

Para tratar de encontrar los hiperparámetros óptimos, se probarán todas las combinaciones posibles de los algoritmos de optimización y las funciones de pérdida mencionadas. Además, se experimentará con diferentes números de épocas para identificar la configuración que maximice el rendimiento del modelo.

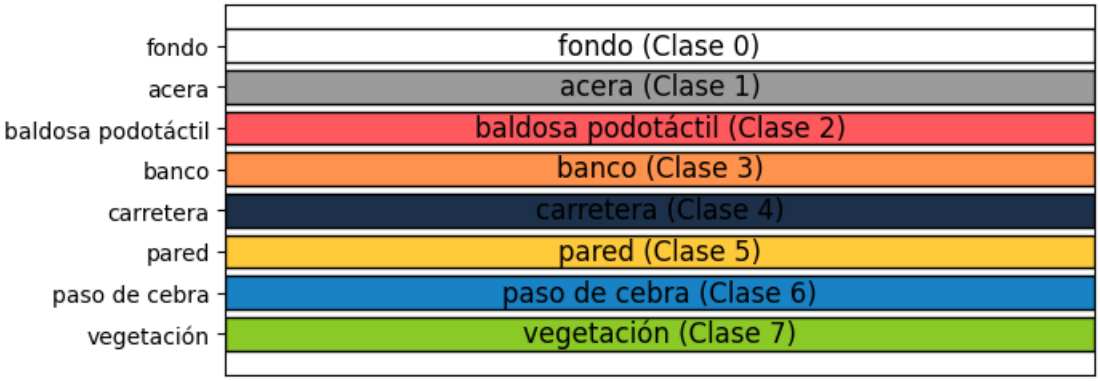
5. Resultados

5.1. Análisis Exploratorio de Datos

Se procede primero con un análisis exploratorio de datos del conjunto de datos para observar su distribución y particularidades.

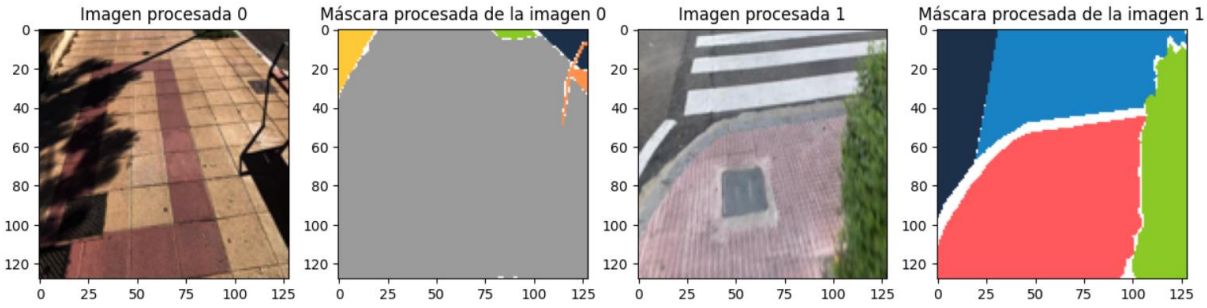
Antes de proceder con una visualización inicial de las imágenes y sus máscaras se incluye a continuación la correspondencia entre los colores, los números de clase y los nombres de estas.

Figura 24. *Correspondencia entre clases, números de clase y colores*



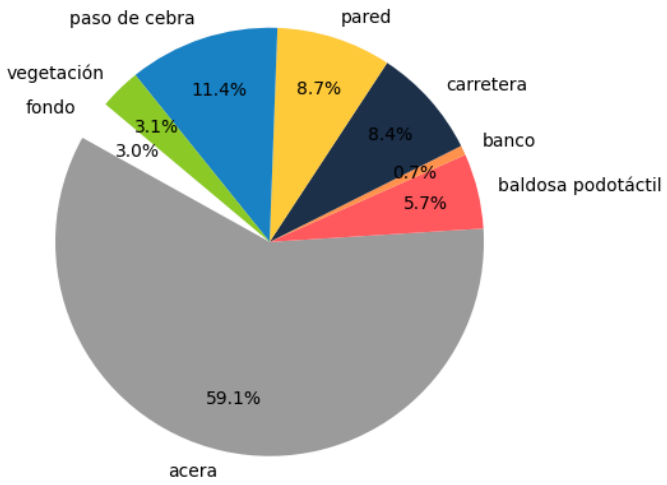
A continuación, se procede con una visualización inicial de algunas imágenes y su máscara correspondiente ya preprocesadas. Esto permite verificar visualmente la correcta correspondencia entre imágenes y sus máscaras.

Figura 25. *Ejemplos de imágenes y su máscara correspondiente preprocesadas.*



Se procede a continuación a analizar el conjunto de máscaras mediante distintos métodos gráficos. En la Figura 26 se puede observar como la clase Acera posee casi el 60% de los píxeles del conjunto de imágenes, es decir, en la gran parte de las imágenes la acera es el elemento principal y mayoritario. Esto es de esperar ya que las imágenes se han tomado apuntando hacia el suelo y la gran parte de la vía urbana es acera. Sin embargo, esto muestra un gran desequilibrio entre las clases que puede afectar negativamente a los modelos. Recalcar también como hay extremadamente pocas muestras de píxeles de la clase banco. La disparidad entre la clase mayoritaria y la clase minoritaria es extremadamente significativa siendo la proporción entre la clase acera y la clase banco de 80,69.

Figura 26. *Distribución de clases entre todas las máscaras*



En cuanto a las clases que contienen las imágenes (Figura 27), tanto la clase acera como las clases carretera, pared y fondo están presentes en más de la mitad de las imágenes. El resto de las clases únicamente no llegan a estar presentes en más de un 40% de las imágenes totales. Pese a los pocos píxeles que tiene la clase banco aparece en un número mayor de imágenes de las que se podía esperar significando esto que ocupan un espacio muy reducido en las imágenes en los que aparecen. Esto mismo se evidencia en la Figura 28. En cuanto a la clase fondo, pese a estar presenta en casi la totalidad del conjunto solo llegan a ocupar de media un 3% del espacio de la imagen.

Figura 27. *Porcentaje de máscaras que contienen cada clase*

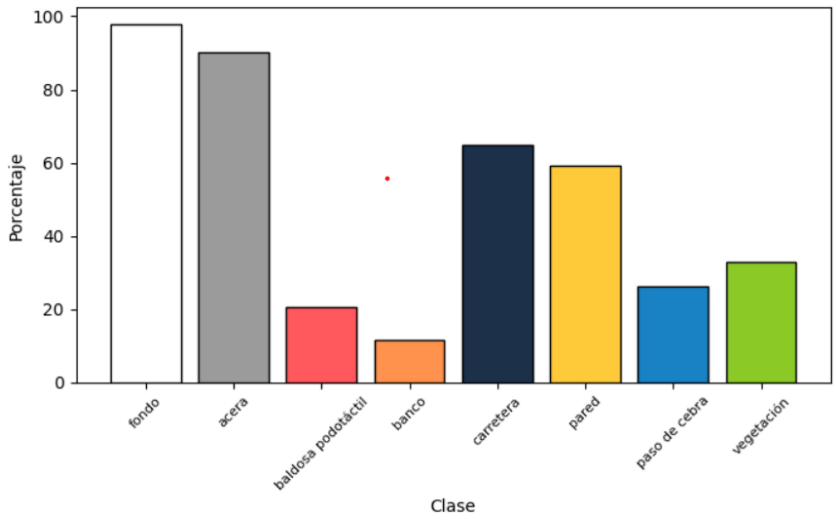
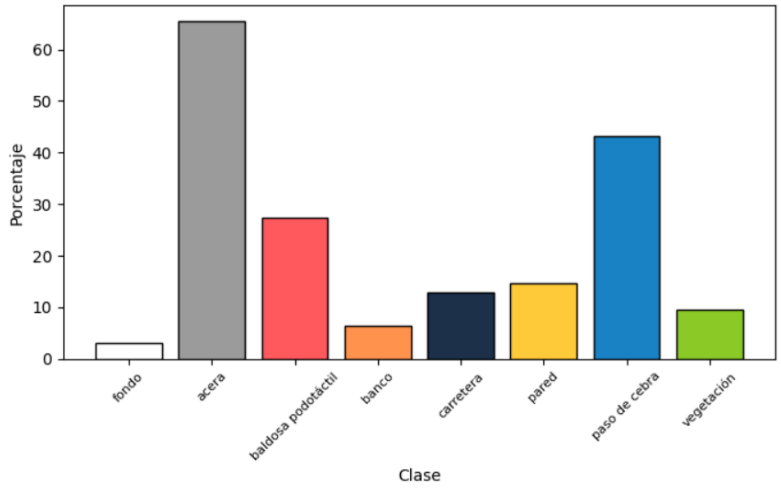
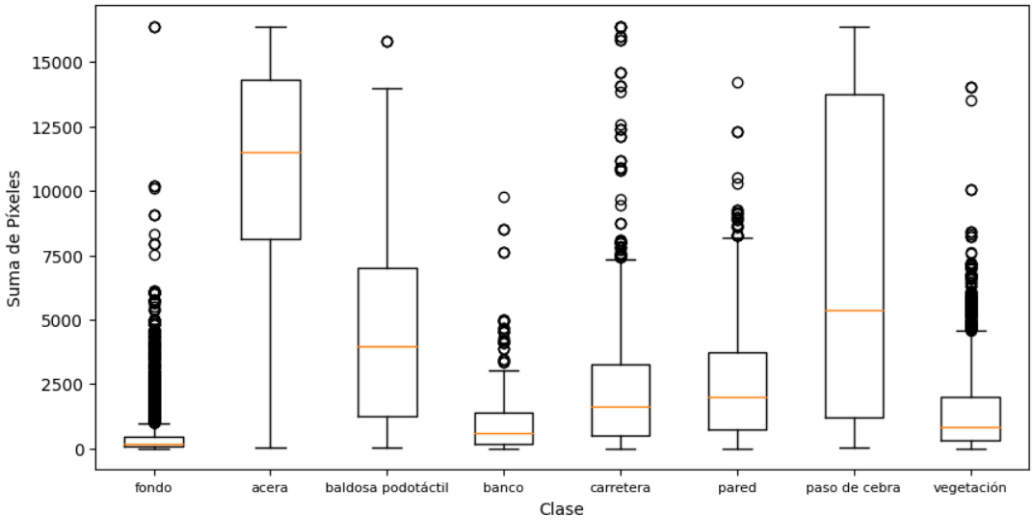


Figura 28. *Porcentaje medio de píxeles por clase en las máscaras*



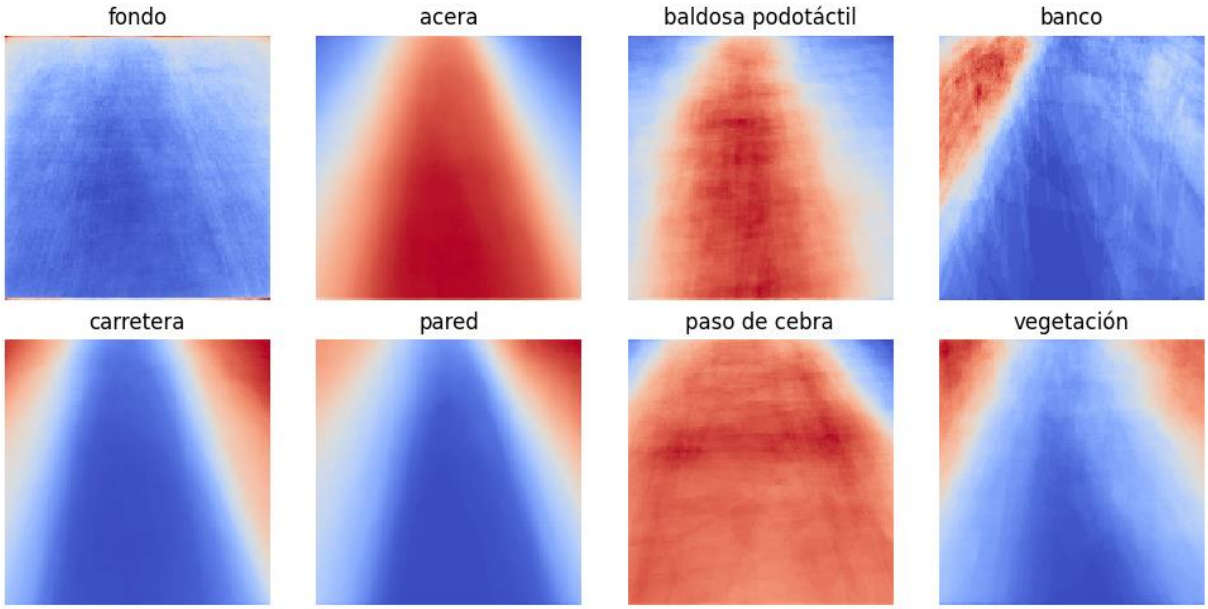
La distribución de píxeles por el conjunto de máscaras se puede observar en el siguiente gráfico boxplot. En la clase paso de cebra se observa que hay una gran variedad de tamaños pudiendo llegar a ocupar desde muy poco hasta un espacio notable de la imagen. Esto se puede deber a que la cruzar un paso de cebra la acera desaparece pasando a ser la vía andante el paso de cebra ocupando esta clase ahora la totalidad de la imagen en estos casos. Hay que destacar que las clases fondo, banco, carretera, pared y vegetación presentan numerosos *outliers.*

Figura 29. *Boxplot de la suma de píxeles por clase*



Por último, para observar si se encuentra algún patrón en la posición de los píxeles de cada clase se han realizado mapas de calor para observar las posiciones más frecuentes de estas clases. La Figura 30 muestra estos mapas de calor. Las posiciones en rojo son las posiciones más frecuentes de esas clases mientras que los píxeles azules reflejan su ausencia.

Figura 30. *Mapas de calor promedio por clase*



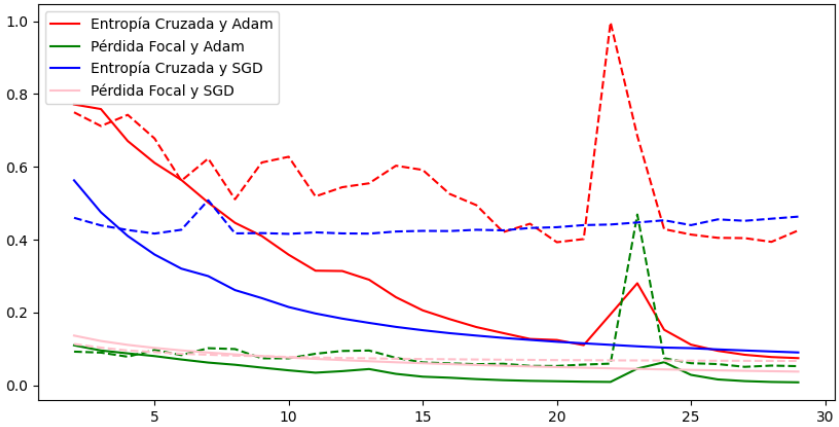
5.2. Evaluación de cada arquitectura

Una vez realizado un análisis exploratorio del conjunto de datos, se ha procedido a entrenar los distintos modelos con las distintas combinaciones de hiperparámetros. Primero se ha buscado de cada modelo la combinación óptima de función de pérdida y de algoritmo de optimización.

A lo largo del entrenamiento, la salida del código actualiza las métricas y se observó que todas las métricas a excepción de la métrica IoU tenían el mismo resultado. Esto se debe a que al calcularse globalmente en vez de por clase son todas las métricas equivalentes a excepción de la métrica IoU. Por ello, mientras se hable del entrenamiento solo se mencionará a la métrica IoU y como ejemplo del resto de métricas a la métrica exactitud (*accuracy).*

Comenzando con DeepLabv3+, se probó inicialmente con un tamaño de lote de 32 y con 30 épocas las cuatro posibles combinaciones de las funciones de pérdida y algoritmos de optimización estudiados. Para una rápida visualización del resultado del entrenamiento se han graficado conjuntamente los cuatros modelos representado tanto sus resultados con el conjunto de entrenamiento (línea continua) como con el conjunto de validación (línea discontinua). En cuanto a la función de pérdida (Figura 31), el modelo con pérdida de entropía cruzada y optimizador Adam presentó un gran valor de pérdida en sus dos primeras épocas que afectaba la visualización gráfica de los resultados por ello, en el gráfico se han procedido a obviar las dos primeras épocas para poder analizar y comparar el conjunto de modelos. Los dos modelos que utilizan la función de pérdida focal presentan muy poca pérdida desde el inicio del entrenamiento además de no divergir prácticamente nada los resultados del entrenamiento y la validación. En contraste, los modelos que utilizan la función de pérdida de entropía cruzada muestran inicialmente mayores pérdidas que pasan a disminuir asemejándose a las pérdidas de los otros modelos, pero únicamente con el conjunto de entrenamiento. En el conjunto de validación entorno a la época seis, la pérdida se estanca. Esto es un posible signo de sobreajuste junto con el pico en pérdida que se observa en tres de los modelos.

Figura 31. *Comparación de la pérdida de los modelos DeepLabv3+ a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Respecto a las métricas, la evolución de todas las métricas es igual como se puede observar en la Figura 32 y Figura 33 únicamente diferenciándose los valores exactos de IoU. La métrica IoU se estabiliza en valores entorno al 0.8 y el resto de las métricas se estabilizan en valores ligeramente superiores, entorno al 0.85. Al igual que en la pérdida, en la época 23 se observa una decadencia puntual de las métricas que a la siguiente época ya se recupera.

Figura 32. *Comparación de la exactitud de los modelos DeepLabv3+ a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*

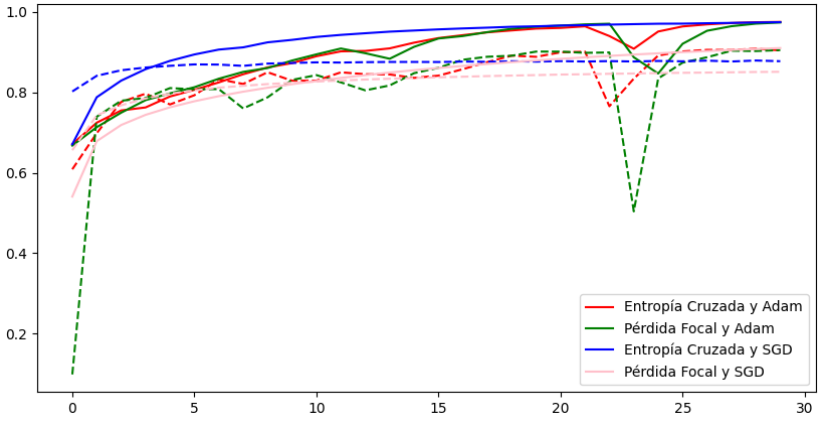
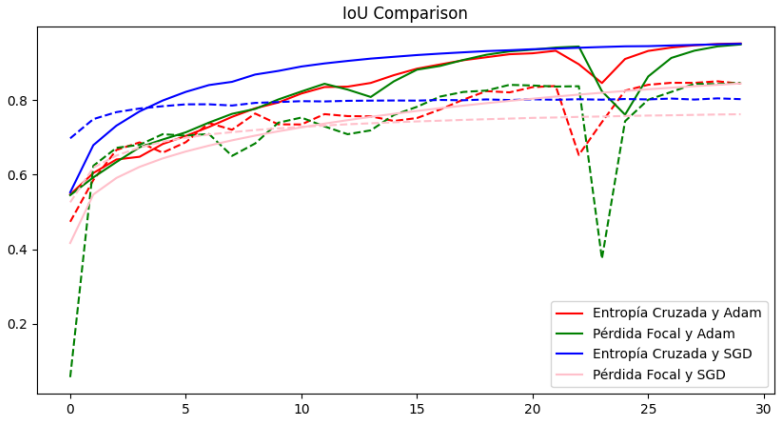


Figura 33. *Comparación de la métrica IoU de los modelos DeepLabv3+ a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Los tres gráficos presentados muestran evidencia de un sobreajuste en todos los modelos debido a que los resultados de todos los modelos se estacan sobre la época 12 y pese a que los resultados en el conjunto de entrenamiento pueden mejorar ligeramente los resultados en el conjunto de validación no presentan mejoras e incluso a largo que aumenten las épocas podrían llegar a hasta a presentar caídas.

El resultado de este entrenamiento (Figura 31) mostró un posible sobreajuste del modelo debido a la divergencia entre el entrenamiento y la validación. Pese a que en el entrenamiento siguen mejorando las estadísticas y disminuyendo la función de pérdida, la validación se ve estancada a partir de entorno a la época 10 e incluso comienza a aumentar sus pérdidas.

Para evaluar cual es el mejor modelo de los cuatro presentados se han tenido en cuenta junto con los gráficos presentado las métricas presentas en la Tabla 2 obtenidas del conjunto de datos de prueba. Se han tenido en cuenta las métricas ponderadas en la medida de lo posible para responder al desquilibrio entre las clases en el conjunto de datos. Se ha concluido que el mejor modelo de entre los cuatro es como función de pérdida la pérdida focal y como algoritmo de optimización Adam. Este será el modelo que se utilizará para comparar con las otras arquitecturas pero disminuyendo el número de épocas de entrenameinto para evitar el sobreajuste mostrado.

Tabla 2. *Métricas obtenidas de los distintos modelos de DeepLabv3+ con el conjunto de datos de prueba con tamaño de lote 32 y 30 épocas.*

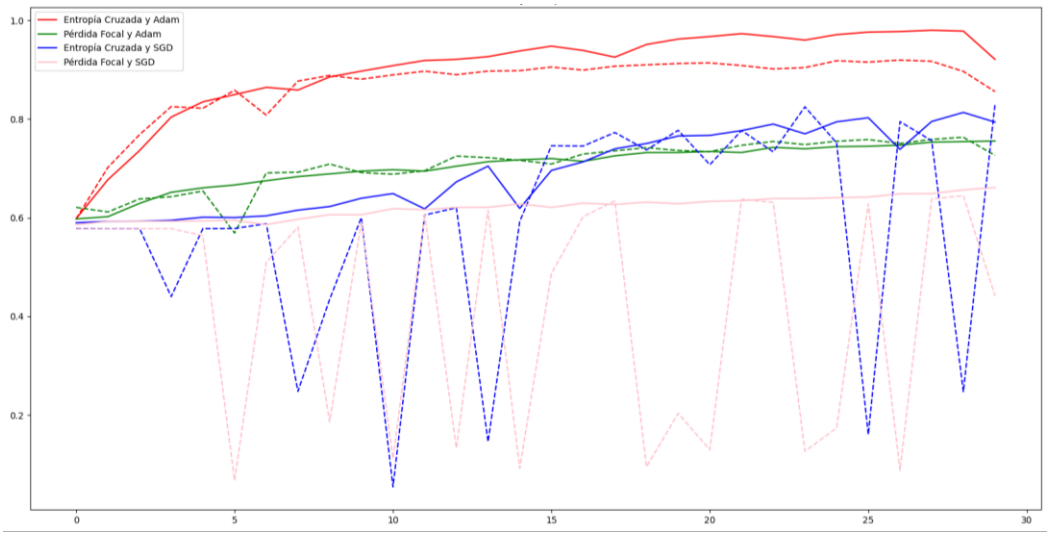
Modelo	Coeficiente Dice ponderado	Precisión ponderada	Exhaustividad (*recall*) ponderada	IoU ponderado	Exactitud (*accuracy*)
DeepLabv3+ Pérdida focal Adam	0,8975	0,8978	0,9036	0,8307	0,9036
DeepLabv3+ Pérdida focal SGD	0,8345	0,8333	0,8388	0,7364	0,8388

DeepLabv3+ Entropía cruzada Adam	0,8907	0,8904	0,8963	0,8222	0,8963
DeepLabv3+ Entropía cruzada SGD	0,8658	0,8661	0,8689	0,7804	0,8689

Respecto a la arquitectura U-Net, la prueba inicial con un tamaño de lote de 32 y con 30 épocas de las cuatro posibles combinaciones de las funciones de pérdida y algoritmos de optimización estudiados mostraron notables diferencias en la evolución de las métricas (Figura 34). Los modelos que utilizan el algoritmo de optimización SGD muestran un claro desajuste en los datos de validación. Este desajuste se ve presente desde las primeras épocas fluctuando con grandes variaciones entre valores extremadamente bajos y valores en torno a 0,6. Este puede ser una posible evidencia de un sobre ajuste de estos modelos con optimizador SGD. También se puede deber a que el algoritmo SGD al no tratar con todo el conjunto de datos por aleatoriedad al optimizar los pesos haya tenido en cuenta una muestra no representativa de los datos y por ello únicamente en esas épocas de mala representación de la muestra los pesos han recaído.

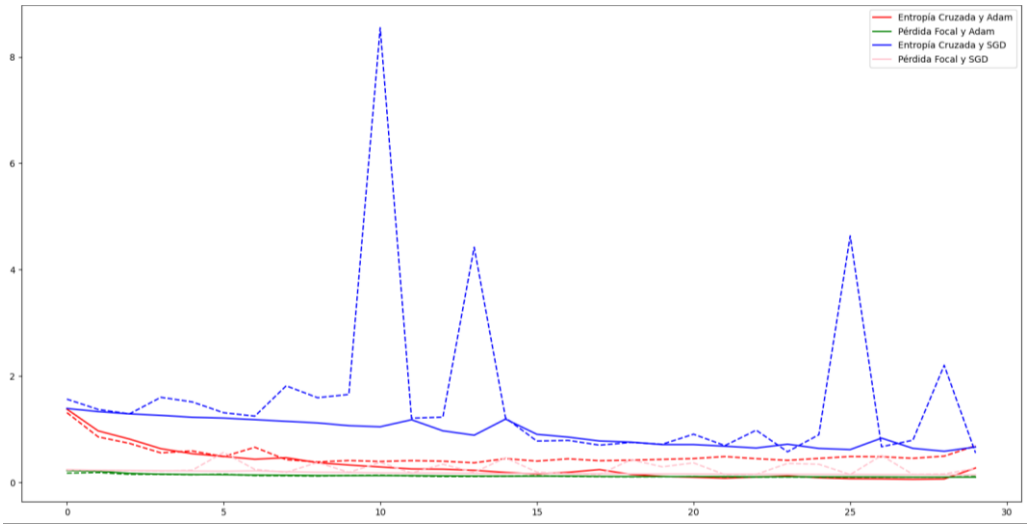
Respecto a los modelos que utilizan el algoritmo de optimización Adam, reflejan una estabilidad que los modelos con SGD no. El modelo con Adam y pérdida de entropía cruzada es consistente mejor que el que otro modelo con Adam, pero función de pérdida focal proporcionando valores superiores en 0,2 puntos. Sin embargo, ambos modelos se estancan y dejan de aumentar tanto el conjunto de validación como en el conjunto de entrenamiento por la época 12 por lo que para la comparación con las otras arquitecturas se reduce el número de épocas de entrenamiento.

Figura 34. *Comparación de la exactitud de los modelos U-Net a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



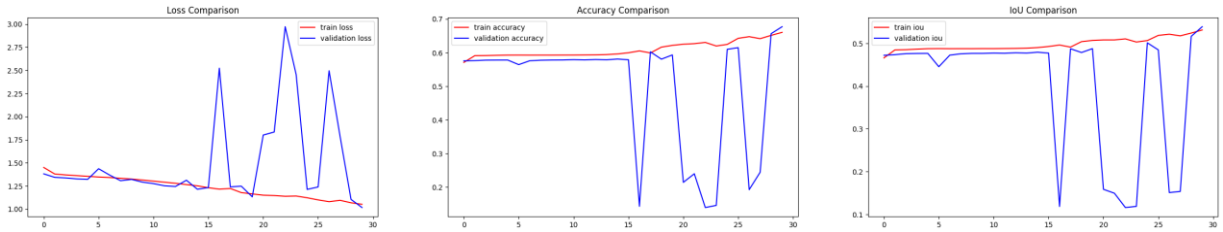
Sin embargo, en las pérdidas (Figura 35), únicamente el modelo U-Net tanto con optimizador SGD y función de pérdida de entropía cruzada muestra refleja estas fluctuaciones extremas en el conjunto de validación. El otro modelo con optimizador SGD refleja pequeñas fluctuaciones, pero un tamaño significativamente menor. Se comporta de manera más similar a los modelos con optimizador Adam. Esto se debe a que la función de pérdida focal consistentemente ha mostrado en todos los modelos presentados valores de pérdida muy reducidos debido a que trata el problema del desequilibrio de las clases en el conjunto de datos al dar mayor peso a los ejemplos más difíciles de clasificar.

Figura 35. *Comparación de la pérdida de los modelos U-Net a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Los modelos U-Net con algoritmo de optimización Adam se trataron de optimizar para reducir las fluctuaciones extremas encontradas en el conjunto de validación y para comprobar si lograban optimizados mejores estadísticas que los modelos con optimizador Adam. Para tratar de evitar el posible sobreajuste que podría ser la causa de estas fluctuaciones se probaron distintas reducciones de la tasa de aprendizaje. La tasa de aprendizaje que se encontró óptima fue la tasa de aprendizaje 0,005. Se pasó de inicialmente ser 0,01 a 0,005. Sin embargo, este cambio no eliminó por completo las fluctuaciones. En las primeras etapas los modelos eran estables, pero pasada la época 15 el modelo comenzaba a presentar el mismo desajuste (Figura 36). Por tanto, estos modelos no se han conseguido optimizar para reflejar una buena representación de los datos.

Figura 36. *Comparación de la pérdida de los modelos U-Net a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



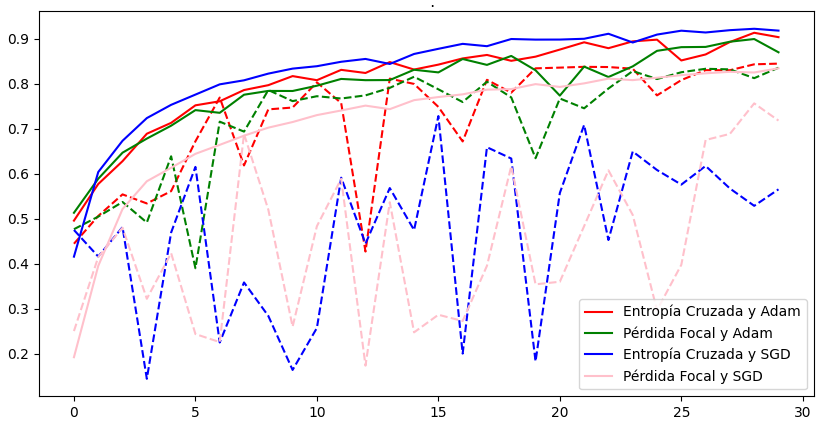
La siguiente tabla (Tabla 3) muestra las métricas obtenidas de los cuatros modelos en el conjunto de datos de prueba para poder comparar su ajuste. Para los modelos con SGD se mantuvo la tasa de aprendizaje inicial debido a que la variación de tasa de aprendizaje no logró la optimización. Se obtuvo que el mejor modelo es como función de pérdida la entropía cruzada y como algoritmo de optimización Adam al presentar los mejores valores en todas las métricas a excepción de en la métrica exactitud que es la segunda mejor. Se le da más peso a las otras muestras ya que la métrica accuracy es calculada globalmente no teniendo en cuenta el desbalance de clases. Este será el modelo que se utilizará para comparar con las otras arquitecturas pero disminuyendo el número de épocas de entrenamiento para evitar el estancamiento final mostrado.

Tabla 3. *Métricas obtenidas de los distintos modelos de U-Net con el conjunto de datos de prueba con tamaño de lote 32 y 30 épocas.*

Modelo	Coeficiente Dice ponderado	Precisión ponderada	Exhaustividad (*recall*) ponderada	IoU ponderado	Exactitud (*accuracy*)
U-Net Pérdida focal Adam	0,6963	0,7095	0,7204	0,5621	0,7204
U-Net Pérdida focal SGD	0,4083	0,5895	0,4291	0,2846	0,4291
U-Net Entropía cruzada Adam	0,8532	0,8510	0,8655	0,7682	0,7629
U-Net Entropía cruzada SGD	0,8015	0,8122	0,8217	0,7061	0,8217

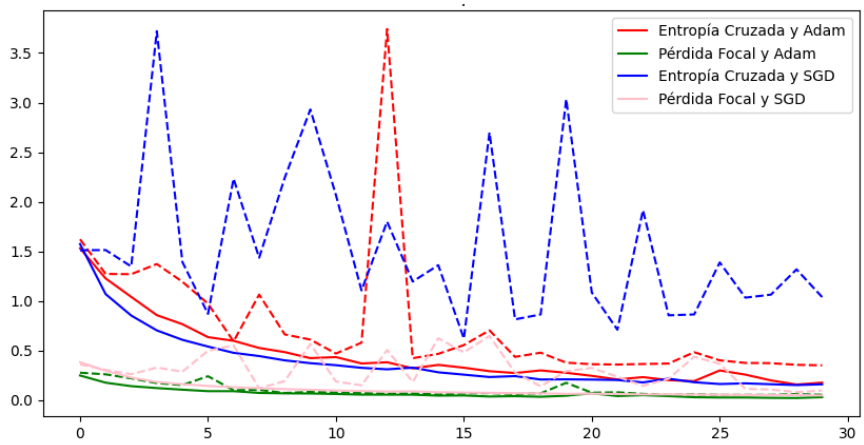
Por último, las distintas combinaciones de algoritmo de optimización y función de pérdida en la arquitectura SegNet reflejan comportamientos similares a los encontrados en la arquitectura U-Net. Los modelos con optimizador SGD vuelven a presentar fluctuaciones extremas en las métricas para el conjunto de validación mostrando un claro desajuste (Figura 37). Los modelos con algoritmo de optimización Adam presentan métricas más estables en ambos conjuntos de datos, aunque también con ligeras fluctuaciones.

Figura 37. *Comparación de la métrica IoU de los modelos SegNet a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Respecto a las pérdidas de los modelos SegNet, aquellos con función de pérdida de entropía cruzada vuelven a mostrar mayores pérdidas que aquellos modelos con pérdida focal. Los modelos con optimizador SGD vuelven también a presentar fluctuaciones en el conjunto de validación siendo más extremas cuando se utiliza entropía cruzada como función de pérdida. El único modelo que no presenta ninguna fluctuación notable en el conjunto de validación ni tampoco presenta divergencia entre el conjunto de validación y de entrenamiento es el modelo con pérdida focal y optimizador Adam ya que el modelo con pérdida de entropía cruzada y con optimizador Adam presenta una fluctuación extrema en la época 12.

Figura 38. *Comparación de la pérdida de los modelos SegNet a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Las métricas obtenidas de estos modelos en el conjunto de datos de prueba se pueden ver en la Tabla 4. Se concluye que el mejor modelo de SegNet de entre los comparados es el que utiliza de función de pérdida la entropía cruzada y de algoritmo de optimización el algoritmo Adam. Este será el modelo que se utilizará para comparar con las otras arquitecturas pero disminuyendo el número de épocas de entrenamiento para evitar el estancamiento final mostrado.

Tabla 4. *Métricas obtenidas de los distintos modelos de SegNet con el conjunto de datos de prueba con tamaño de lote 32 y 30 épocas.*

Modelo	Coeficiente Dice ponderado	Precisión ponderada	Exhaustividad (*recall*) ponderada	IoU ponderado	Exactitud (*accuracy*)
SegNet Pérdida focal Adam	0,8875	0,8861	0,8932	0,8164	0,8932
SegNet Pérdida focal SGD	0,7885	0,7897	0,8077	0,6803	0,8077
SegNet Entropía cruzada Adam	0,8942	0,8923	0,9014	0,8264	0,9014
SegNet Entropía cruzada SGD	0,7215	0,7510	0,7040	0,5820	0,7040

5.3. Comparación entre modelos

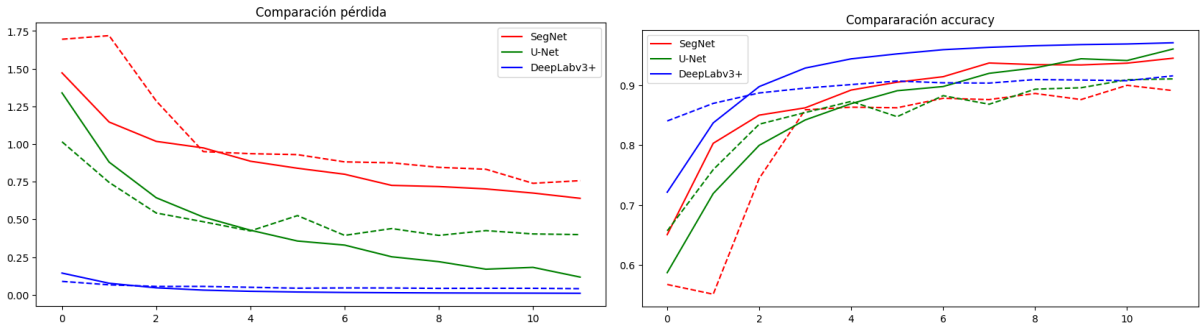
Los hiperparámetros que se han considerado óptimos para cada una de las arquitecturas han sido tras experimentar y evaluar han sido los siguientes:

- Arquitectura SegNet: 12 épocas, 32 de tamaño de lote, algoritmo de optimización Adam con tasa de aprendizaje 0,0001 y función de pérdida entropía cruzada.
- Arquitectura U-Net: 12 épocas, 32 de tamaño de lote, algoritmo de optimización Adam con tasa de aprendizaje 0,0001 y función de pérdida entropía cruzada.

- Arquitectura DeepLabv3+: 12 épocas, 32 de tamaño de lote, algoritmo de optimización Adam con tasa de aprendizaje 0,0001 y función de pérdida focal.

Los resultados con estos hiperparámetros de los tres modelos a lo largo del entrenamiento han sido los siguientes:

Figura 39. *Comparación de la pérdida y exactitud de las arquitecturas SegNet, U-Net y DeepLabv3+ a lo largo del entrenamiento en el conjunto de entrenamiento (línea continua) y en el conjunto de validación (línea discontinua)*



Todos los modelos presentan resultados estableces sin fluctuaciones notables a diferencia de los experimentos anteriores. Además, la divergencia entre las predicciones en el conjunto de entrenamiento y en el conjunto de validación tampoco es notable hasta las últimas épocas que si hay evidencias del comienzo de un sobreajuste si se aumentan el número de épocas como se observó en los experimentos anteriores. Visualmente DeepLabv3+ aparenta ser el modelo que mejor se ajusta al conjunto de datos al tener pérdidas significativamente menores que el resto de los modelos y al tener la mejor exactitud tanto en el conjunto de entrenamiento como en el de validación de entre los tres modelos por un pequeño margen.

Sin embargo, para decidir qué modelo es el mejor se debe utilizar el conjunto de datos de prueba. Las métricas obtenidas de las predicciones en este conjunto de datos sin distinguir entre las clases son las siguientes:

Tabla 5. *Métricas obtenidas de los modelos SegNet, U-Net y DeepLabv3+*

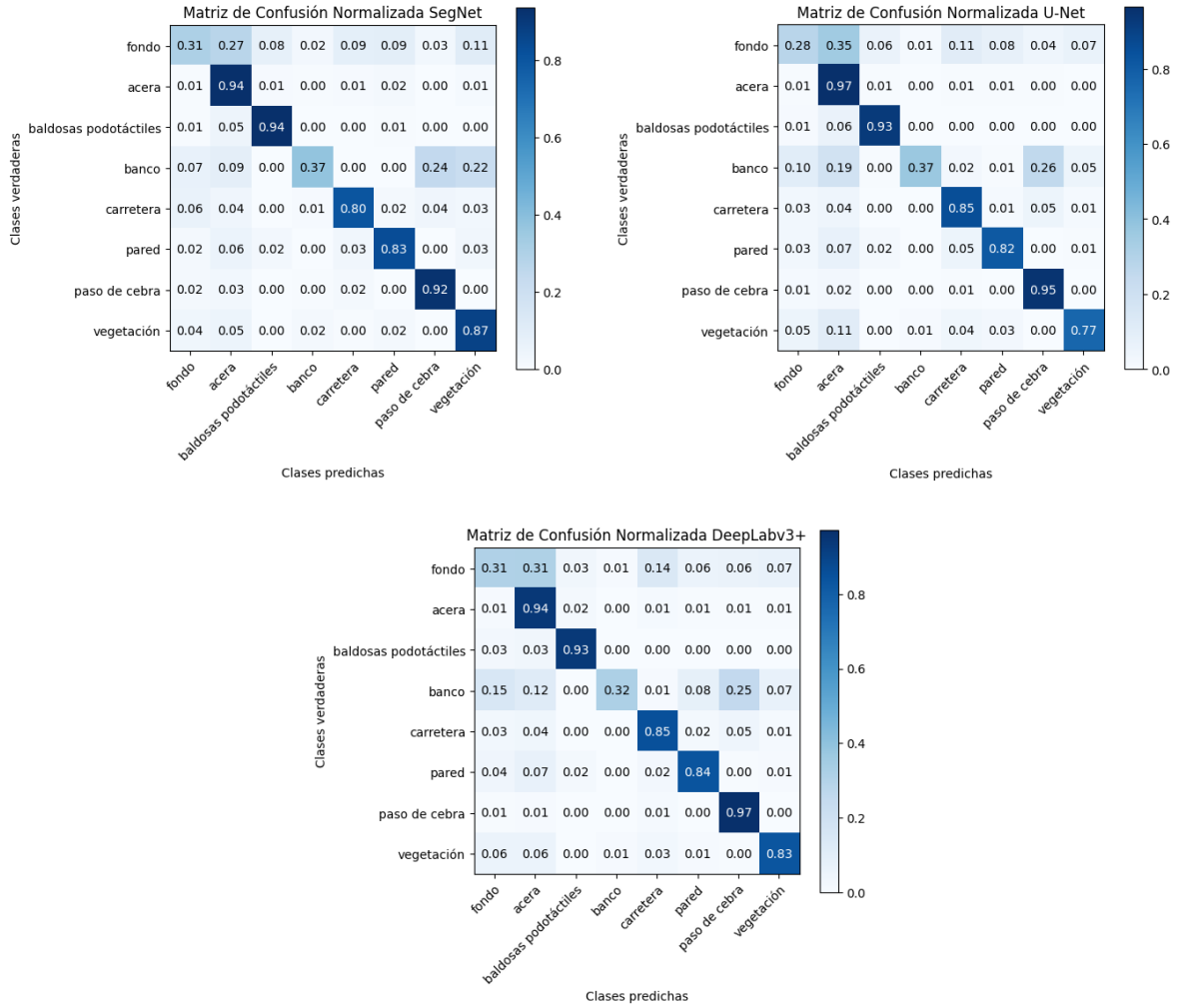
Modelo	Coeficiente Dice ponderado	Precisión ponderada	Exhaustividad (*recall*) ponderada	IoU ponderado	Exactitud (*accuracy*)
SegNet	0,8893	0,8915	0,8895	0,8100	0,8895
U-Net	0,9034	0,9013	0,9079	0,8400	0,9079
DeepLabv3+	0,8984	0,8990	0,9005	0,8316	0,9005

El modelo que obtiene mejores valores en todas las métricas es la arquitectura U-Net.

Para evaluar los distintos modelos distinguiendo entre las distintas clases a segmentar se puede observar la matriz de confusión de cada modelo (Figura 40). Resulta que todos los modelos predicen las clases de manera similar. Las clases que más se predicen correctamente son la acera, las baldosas podotáctiles y los pasos de cebra. Las peores predichas son el fondo y los pasos de cebra. La interpretación de la clase fondo se ignora en este análisis ya que no predice ningún elemento en concreto, esta clase son los píxeles que quedaron sin anotar en el conjunto de datos inicial. La mala predicción de los bancos se

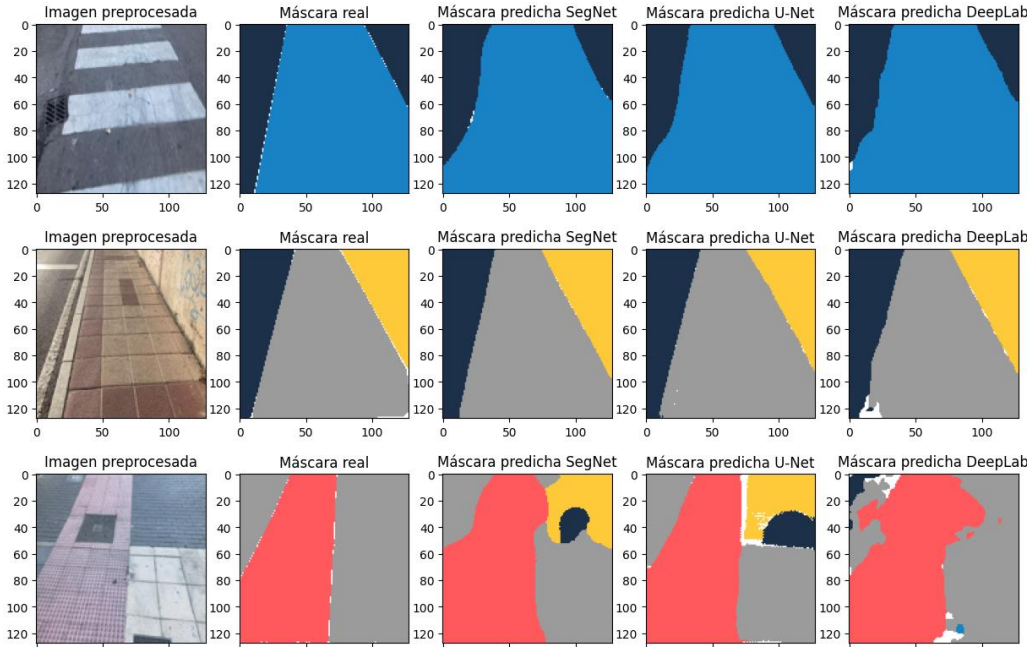
debe a que están muy poco representados en el conjunto de datos como se observó en el análisis exploratorio de datos. Los bancos los modelos los confunden como fondo, acera, pasos de cebra o vegetación. En general, si el modelo se confunde al predecir una clase, el modelo la confunde o con acera o con fondo. Confundirla con acera se puede deber a que la mayoría de los píxeles del conjunto de datos son de acera por lo que su influencia debe ser significativa.

Figura 40. *Matrices de confusión normalizadas de los modelos SegNet, U-Net y DeepLabv3+*



A continuación, se incluye una visualización de como segmentan los distintos modelos varias imágenes del conjunto de datos de prueba.

Figura 41. *Visualización de la imagen real frente a su máscara real y las máscaras predichas por los distintos modelos entrenados*



6. Conclusiones

Desde que surgieron las primeras arquitecturas de redes neuronales convolucionales para la segmentación, estas técnicas de aprendizaje profundo se han convertido en el principal método para segmentar imágenes. Inspiradas en la primera red neuronal convolucional para la segmentación de imágenes, la arquitectura FCN, han surgido numerosas arquitecturas para este propósito. Destacan las arquitecturas U-Net, SegNet y DeepLabv3+ ampliamente utilizadas y mencionadas en la literatura científica debido a su precisión y eficiencia computacional.

Existen numerosas colecciones de datos de imágenes listas para entrenar modelos de segmentación especialmente colecciones de datos de imágenes de la carretera para entrenar modelos que sirvan para la conducción autónoma y aplicaciones similares. Sin embargo, no hay prácticamente ninguna colección de datos extendida con imágenes de la vía urbana desde el punto de vista de un peatón. La generación de modelos de segmentación entrenados con imágenes de este tipo podría servir para la asistencia en la navegación de personas con discapacidad visual. Por ello, en este trabajo se ha creado un conjunto de datos con 2481 imágenes de estas características y se ha experimentado con las redes neuronales convolucionales U-Net, SegNet y DeepLabv3+ para conseguir un modelo óptimo modificando diversos hiperparámetros.

En general los modelos que han utilizado función de pérdida focal presentan menores pérdidas que los correspondientes modelos con función de pérdida de entropía cruzada. Esto se puede deber a que la función de pérdida focal trata el problema de desequilibro de las clases en el conjunto de datos al aprender más de los ejemplos difíciles consecuentemente disminuyendo así la variabilidad de las pérdidas.

Las arquitecturas SegNet y U-Net con optimizador SGD han presentado extremas fluctuaciones en las métricas con el conjunto de datos de validación a diferencia de la arquitectura DeepLab que estos modelos con optimizador SGD han sido más estables y consistentes, tanto en el conjunto de entrenamiento como en el conjunto de validación, que los modelos con optimizador Adam. Esta diferencia se puede deber a que la arquitectura DeepLab parte de una arquitectura preentrenada como es ResNet.

Entre los modelos óptimos de las tres arquitecturas, se ha hallado que el modelo que mejor segmenta el conjunto de datos utilizado es el modelo U-Net con 12 épocas, 32 de tamaño de lote, algoritmo de optimización Adam con tasa de aprendizaje 0,0001 y con función de pérdida de entropía cruzada. Este modelo tiene una exactitud en la predicción del 90,79%. Las clases mejor segmentadas son la acera, las baldosas podotáctiles y los pasos de cebra con un porcentaje de predicciones correctas superior al 90%. Las clases peor segmentadas son el fondo y los pasos de cebra. La mala predicción de ambas clases se debe a su falta de representación en el conjunto.

Esta alta precisión deja abierta la posibilidad de un futuro uso de este modelo en dispositivos de asistencia en la navegación de personas con discapacidad visual. La información que proporciona la segmentación permite ubicar los distintos elementos de la vía urbana que una persona con discapacidad visual no puede percibir. Esta información puede ser procesada por un sistema que proporcione retroalimentación a la persona ciega para una navegación autónoma y más segura.

7. Bibliografía

Abderrahim, N. Y. Q., Abderrahim, S., & Rida, A. (2020). Road segmentation using u-net architecture. *2020 IEEE international conference of moroccan geomatics (Morgeo)*, 1-4.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, 1-6. https://doi.org/10.1109/ICEngTechnol.2017.8308186

Anderson, D., & McNeill, G. (1992). Artificial neural networks technology. *Kaman Sciences Corporation*, *258*(6), 1-83.

Ayuntamiento de Cartagena. (2023, septiembre 27). *Ayuntamiento y ONCE impulsan una auditoría para ampliar la accesibilidad de los semáforos para personas ciegas en Cartagena*.

Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481-2495. https://doi.org/10.1109/TPAMI.2016.2644615

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. *Proceedings of COMPSTAT 2010 - 19th International Conference on Computational Statistics, Keynote, Invited and Contributed Papers*, 177-186. https://doi.org/10.1007/978-3-7908-2604-3_16

Boureau, Y.-L., Ponce, J., Fr, J. P., & Lecun, Y. (2010). *A Theoretical Analysis of Feature Pooling in Visual Recognition*.

Boykov, Y., Veksler, O., & Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(11), 1222-1239. https://doi.org/10.1109/34.969114

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2014). Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, *40*(4), 834-848.

Chen, L.-C., Papandreou, G., Schroff, F., & Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. arXiv. *arXiv preprint arXiv:1706.05587*, *5*.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Proceedings of the European conference on computer vision (ECCV)*, 801-818.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251-1258.

Chollet, F., & otros. (2015). *Keras*. GitHub. https://keras.io

Drozdzal, M., Vorontsov, E., Chartrand, G., Kadoury, S., & Pal, C. (2016). The Importance of Skip Connections in Biomedical Image Segmentation. En G. Carneiro, D. Mateus, L. Peter, A. Bradley, J. M. R. S. Tavares, V. Belagiannis, J. P. Papa, J. C. Nascimento, M. Loog, Z. Lu, J. S. Cardoso, & J. Cornebise (Eds.), *Deep Learning and Data Labeling for Medical Applications* (pp. 179-187). Springer International Publishing.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Dwyer, B., Nelson, J., & Hansen, T. (2024). *Roboflow* (Version 1.0). Disponible en https://roboflow.com.

Esteva, A., Chou, K., Yeung, S., Naik, N., Madani, A., Mottaghi, A., Liu, Y., Topol, E., Dean, J., & Socher, R. (2021). Deep learning-enabled medical computer vision. En *npj Digital Medicine* (Vol. 4, Número 1). Nature Research. https://doi.org/10.1038/s41746-020-00376-2

Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Gläser, C., Timm, F., Wiesbeck, W., & Dietmayer, K. (2021). Deep Multi-Modal Object Detection and Semantic Segmentation for Autonomous Driving: Datasets, Methods, and Challenges. *IEEE Transactions on Intelligent Transportation Systems*, *22*(3), 1341-1360. https://doi.org/10.1109/TITS.2020.2972974

Frajberg, D. (2017). *Introduction to the Artificial Intelligence and Computer Vision revolution*. Dipartimento di elettronica, informazione e bioingegneria, Politecnico Di Milano.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, *36*(4), 193-202.

Gonçalves, D. N., Weber, V. A. de M., Pistori, J. G. B., Gomes, R. da C., de Araujo, A. V., Pereira, M. F., Gonçalves, W. N., & Pistori, H. (2021). Carcass image segmentation using CNN-based methods. *Information Processing in Agriculture*, *8*(4), 560-572. https://doi.org/https://doi.org/10.1016/j.inpa.2020.11.004

Gonzalez, L. (2019). *Matriz de Confusión - Aprende IA*. https://aprendeia.com/matriz-de-confusion-machine-learning/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. https://books.google.es/books?id=omivDQAAQBAJ

Graupe, D. (2013). *Principles Of Artificial Neural Networks (3rd Edition)*. World Scientific Publishing Company. https://books.google.es/books?id=Zz27CgAAQBAJ

Gupta, D. (s. f.). *Image Segmentation Keras : Implementation of Segnet, FCN, UNet, PSPNet and other models in Keras*. https://divamgupta.com/image-

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020). GhostNet: More features from cheap operations. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1577-1586. https://doi.org/10.1109/CVPR42600.2020.00165

Hardt, M., Recht, B., & Singer, Y. (2016). Train faster, generalize better: Stability of stochastic gradient descent. *International conference on machine learning*, 1225-1234.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Gohlke, C., & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770-778.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. http://arxiv.org/abs/1207.0580

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700-4708.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science and Engineering*, *9*(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

INE. (1999). *Encuestas de discapacidades*.

INE. (2008). *Encuestas de discapacidades*.

INE. (2020). *Encuestas de discapacidades*.

Jadon, S. (2020). A survey of loss functions for semantic segmentation. *2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB)*, 1-7.

Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. *Electronic Markets*, *31*(3), 685-695.

Kass, M., Witkin, A., & Terzopoulos, D. (1988). Snakes: Active contour models. *International Journal of Computer Vision*, *1*(4), 321-331. https://doi.org/10.1007/BF00133570

Kaymak, Ç., & Uçar, A. (2019). A Brief Survey and an Application of Semantic Image Segmentation for Autonomous Driving. En V. E. Balas, S. S. Roy, D. Sharma, & P.

Samui (Eds.), *Handbook of Deep Learning Applications* (pp. 161-200). Springer International Publishing. https://doi.org/10.1007/978-3-030-11479-4_9

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.

Klein, A., Wallkötter, S., Silvester, S., Rynes, A., actions-user, Müller, P., Nunez-Iglesias, J., Harfouche, M., Schrangl, L., Dennis, Lee, A., McCormick, M., Pandede, OrganicIrradiation, Rai, A., Ladegaard, A., van Kemenade, H., Smith, T. D., Vaillant, G., … Inggs, G. (2024). *imageio/imageio: v2.34.1*. Zenodo. https://doi.org/10.5281/zenodo.11018412

Koech, K. E. (2020). *Cross-Entropy Loss Function*. Towards Data Science. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Larranaga, P., Inza, I., & Moujahid, A. (1997). Tema 8. redes neuronales. *Redes Neuronales, U. del P. Vasco*, *12*, 17.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, *2*.

LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., & Simard, P. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural networks: the statistical mechanics perspective*, *261*(276), 2.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(12), 6999-7019. https://doi.org/10.1109/TNNLS.2021.3084827

Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal Loss for Dense Object Detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2999-3007. https://doi.org/10.1109/ICCV.2017.324

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3431-3440.

Lopez Pinaya, W. H., Vieira, S., Garcia-Dias, R., & Mechelli, A. (2019). Convolutional neural networks. En *Machine Learning: Methods and Applications to Brain Disorders* (pp. 173-191). Elsevier. https://doi.org/10.1016/B978-0-12-815739-8.00010-9

López, R. F., & Fernández, J. M. F. (2008). *Las Redes Neuronales Artificiales*. Netbiblo.

Lundh, F., & Clark, J. A. (2014). Pillow (PIL Fork). En *GitHub repository*. GitHub.

Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

Martín, A., Ashish, A., Paul, B., Eugene, B., Zhifeng, C., Craig, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. https://doi.org/10.5281/zenodo.4724125

Minaee, S., Boykov, Y., Porikli, F., Plaza, A., Kehtarnavaz, N., & Terzopoulos, D. (2022). Image Segmentation Using Deep Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(7), 3523-3542. https://doi.org/10.1109/TPAMI.2021.3059968

Ministerio de Transportes, Movilidad y Agenda Urbana, & DG de Vivienda Suelo. (2022). *Áreas urbanas en España, 2022*.

Miyamoto, R., Nakamura, Y., Adachi, M., Nakajima, T., Ishida, H., Kojima, K., Aoki, R., Oki, T., & Kobayashi, S. (2019). Vision-Based Road-Following Using Results of Semantic Segmentation for Autonomous Navigation. *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, 174-179. https://doi.org/10.1109/ICCE-Berlin47944.2019.8966198

Moreno Rodilla, V. (2023). Redes Neuronales Artificiales. *Fundamentos de Sistemas Inteligentes*, 12.

Mortensen, A. K., Dyrmann, M., Karstoft, H., Jørgensen, R. N., & Gislum, R. (2016). *Semantic segmentation of mixed crops using deep convolutional neural network.*

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807-814.

Organización Mundial de la Salud. (2020). *Informe mundial sobre la visión*.

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. http://arxiv.org/abs/1511.08458

Paneru, S., & Jeelani, I. (2021). Computer vision applications in construction: Current state, opportunities & challenges. En *Automation in Construction* (Vol. 132). Elsevier B.V. https://doi.org/10.1016/j.autcon.2021.103940

Papandreou, G., Kokkinos, I., & Savalle, P.-A. (2015). Modeling local and global deformations in deep learning: Epitomic convolution, multiple instance learning, and sliding window detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 390-399.

Paratore, M. T., & Leporini, B. (2023). Exploiting the haptic and audio channels to improve orientation and mobility apps for the visually impaired. *Universal Access in the Information Society*, 257. https://doi.org/10.1007/s10209-023-00973-4

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Bai, J., & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*(85), 2825-2830. http://jmlr.org/papers/v12/pedregosa11a.html

Priddy, K. L., & Keller, P. E. (2005). *Artificial Neural Networks: An Introduction*. SPIE Press.

Rana, S. (2024). Exploring the Advancements and Ramifications of Artificial Intelligence. *Journal of Artificial Intelligence General science (JAIGS) ISSN:3006-4023*, *2*(1), 30-35. https://doi.org/10.60087/jaigs.v2i1.p35

Restrepo, R. (s. f.). *Inputs, labels, and outputs*. Ronny Restrepo. http://ronny.rest/tutorials/module/seg_01/segmentation_03_inputs_outputs/

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, 400-407.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 9351). https://doi.org/10.1007/978-3-319-24574-4_28

Rosenfeld, A. (1976). *Digital Picture Processing*. Elsevier Science. https://books.google.es/books?id=ANiNDAAAQBAJ

Sakib, S., 1#, A., Jawad, A., 2@, K., & Ahmed, H. (2019). *An Overview of Convolutional Neural Network: Its Architecture and Applications*. https://doi.org/10.20944/preprints201811.0546.v4

Samuel, A. L. (1959). *Some Studies in Machine Learning Using the Game of Checkers*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510-4520. https://doi.org/10.1109/CVPR.2018.00474

Sarvamangala, D. R., & Kulkarni, R. V. (2022). Convolutional neural networks in medical image understanding: a survey. En *Evolutionary Intelligence* (Vol. 15, Número 1). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/s12065-020-00540-3

Sebe, N. (2005). *Machine Learning in Computer Vision*. Springer Netherlands. https://books.google.es/books?id=lemw2Rhr_PEC

Sezgin, M., & Sankur, B. lent. (2004). Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, *13*(1), 146-168.

Si, Y., Gong, D., Guo, Y., Zhu, X., Huang, Q., Evans, J., He, S., & Sun, Y. (2021). An advanced spectral–spatial classification framework for hyperspectral imagery based on DeepLab v3+. *Applied Sciences*, *11*(12), 5703.

Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. http://arxiv.org/abs/1409.1556

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). *Striving for Simplicity: The All Convolutional Net*. http://arxiv.org/abs/1412.6806

Suzuki, K. (2011). *Artificial neural networks: methodological advances and biomedical applications*. BoD–Books on Demand.

Sydenham, P. H., & Thorn, R. (2005). *Handbook of measuring system design*. Wiley.

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

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818-2826.

The pandas development team. (2024). *pandas-dev/pandas: Pandas*. Zenodo. https://doi.org/10.5281/zenodo.10957263

TIOBE Index - TIOBE. (s. f.). Recuperado 23 de mayo de 2024, de https://www.tiobe.com/tiobe-index/

Treml, M., Arjona-Medina, J., Unterthiner, T., Durgesh, R., Friedmann, F., Schuberth, P., Mayr, A., Heusel, M., Hofmarcher, M., Widrich, M., Nessler, B., & Hochreiter, S. (2016). *Speeding up Semantic Segmentation for Autonomous Driving*.

Van Rossum, G., & Drake, F. L. (1995). *Python reference manual* (Vol. 111). Centrum voor Wiskunde en Informatica Amsterdam.

Vázquez Regueiro, C., Barro Ameneiro, S., Sánchez Vila, E., & Fernández Delgado, M. (1995). *Modelos básicos de redes neuronales artificiales*.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. En *Insights into Imaging* (Vol. 9, Número 4, pp. 611-629). Springer Verlag. https://doi.org/10.1007/s13244-018-0639-9

Yi-de, M., Qing, L., & Zhi-Bai, Q. (2004). Automated image segmentation using improved PCNN model based on cross-entropy. *Proceedings of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.*, 743-746.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, 818-833.

Zhang, D., Song, Y., Liu, D., Jia, H., Liu, S., Xia, Y., Huang, H., & Cai, W. (2018). Panoptic segmentation with an end-to-end cell r-cnn for pathology image analysis. *Medical Image Computing and Computer Assisted Intervention–MICCAI 2018: 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II 11*, 237-244.

Zhang, Y. (2010). *New Advances in Machine Learning*. IntechOpen. https://books.google.es/books?id=XAqhDwAAQBAJ

Zhang, Y., Zhao, X., & Liu, P. (2019). Multi-Point Displacement Monitoring Based on Full Convolutional Neural Network and Smartphone. *IEEE Access*, *7*, 139628-139634. https://doi.org/10.1109/ACCESS.2019.2943599

8. Anexo I: Código Fuente

En el siguiente enlace se encuentra el código fuente utilizado para el estudio e implementación de las diversas arquitecturas de redes neuronales convolucionales seleccionadas para la segmentación de imágenes de la vía urbana:

https://github.com/juliagarve/tfg-estadistica