Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

Universidad de Jaén
Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE MÁSTER

DEEP LEARNING GEOMÉTRICO APLICADO A DATOS LIDAR

Alumno

Miguel Díaz Medina

Tutores

Rafael Jesús Segura Sánchez

(Departamento de Informática)

José Manuel Fuertes García

(Departamento de Informática)

Mayo, 2021

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Rafael Jesús Segura Sánchez y Don José Manuel Fuertes García, tutores del Trabajo Fin de Máster titulado: '**Deep Learning geométrico aplicado a datos LiDAR**', que presenta Don Miguel Díaz Medina, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, mayo de 2021

El alumno: Los tutores:

Miguel Díaz Medina

Rafael Jesús Segura Sánchez

José Manuel Fuertes García

(Página intencionalmente en blanco)

Agradecimientos

A mis padres, Miguel y Rocío, por darme la oportunidad de seguir formándome y apoyarme en todas y cada una de mis decisiones.

A Sara, por su apoyo incondicional y su paciencia incluso en los días más difíciles.

A mis tutores, José Manuel y Rafa, por acceder a guiarme en este tortuoso camino de la investigación y su entrega en los momentos más necesarios.

A todas y cada una de las personas que alguna vez me dieron ánimos para seguir adelante en este proyecto.

Eternamente agradecido.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Máster en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Especialidad (solo TFM)	Sin especialidad
Mención (solo TFM)	Sin mención
Idioma	Español
Tipo	General
TFT en equipo	No
Autor/a	Miguel Díaz Medina
Fecha de asignación	07/10/2020
Descripción corta	La segmentación semántica de nubes de puntos es un problema clásico en el campo de la informática gráfica y visión por computador que desde hace décadas se ha convertido en uno de los principales objetos de estudio para investigadores de todo el mundo. Los datos LiDAR (*Light Detection And Ranging*) son generados mediante escáneres LiDAR, y se constituyen como nubes de puntos con una cantidad de información muy superior a las nubes de puntos convencionales. Además de información sobre las coordenadas ***(x, y, z)***, un escáner LiDAR es capaz de capturar otros atributos del entorno como por ejemplo el color, la reflectividad del material, el número de retornos de cada punto, además de otras propiedades sobre la superficie de los materiales. El aprendizaje profundo se ha situado en la cúspide de las soluciones para la mayoría de problemas existentes en el mundo de las ciencias de la computación, desde la **visión por computador** hasta el **procesamiento del lenguaje natural**. Se trata de resolver problemas usando un enfoque basado en la inteligencia artificial que construya su propia representación del conocimiento para resolver el problema en cuestión. En tanto, el presente trabajo tiene el objetivo de estudiar las propuestas actuales enfocadas en segmentación de nubes de puntos que parten del aprendizaje profundo, y evaluar la influencia que tiene sobre las mismas la incorporación de canales de información adicionales como metainformación a la geometría.

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES E INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
UNE 50132:1994	Traducción de la ISO 2145
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

NORMAS UTILIZADAS COMO BASE O REFERENCIA	
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
*Estas normas se han utilizado **como base o referencia** para la inclusión de algunos contenidos y definiciones sobre elaboración de proyectos, entendiendo como **proyecto** la documentación consensuada entre una empresa y un cliente, que da lugar al perfeccionamiento de un contrato para la elaboración de una obra o la prestación de un servicio. Por consiguiente, no debe esperarse la aplicación de estas normas en cuanto a la completitud de los contenidos ni a la organización de los mismos.*	

Contenido

Índice de ilustraciones

Índice de tablas

1 ESPECIFICACIÓN DEL TRABAJO

En este capítulo se presenta la especificación del trabajo, con una estructura y contenidos **inspirados** en los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en el apartado 6 (Definiciones y abreviaturas).

1.1 Introducción

1.1.1 Aprendizaje profundo

Los avances realizados en el campo de la **IA** (Inteligencia Artificial) durante las últimas décadas, concretamente en el área de **AA** (Aprendizaje Automático) o *machine learning*, han motivado a la comunidad científica a la creación de soluciones basadas en la Inteligencia Artificial, apartando del foco a las soluciones tradicionales que ya habían alcanzado una capacidad de rendimiento pico, siendo imposible en muchos casos encontrar soluciones óptimas para los problemas más complejos.

Contextualizando el presente proyecto, y puesto que se enmarca como un trabajo teórico-experimental, se definirá la solución haciendo referencia la programación en el sentido de la computación clásica. En una aplicación tradicional no basada en inteligencia artificial, un **programa** recibe como entrada **datos** y **reglas** (o alguna representación del conocimiento), y genera como resultado una **salida** para los datos proporcionados utilizando como conocimiento el esquema introducido. En cambio, cuando se trata de programación basada en AA, el **programa** recibe como entrada tanto los **datos** como la **salida deseada**, y el propósito del algoritmo es, por tanto, **encontrar una representación del conocimiento** que mejor describa la relación entre los datos de entrada y las salidas generadas. Esto permite lo que en el universo IA se conoce como **aprendizaje**, véase Figura 1.1.

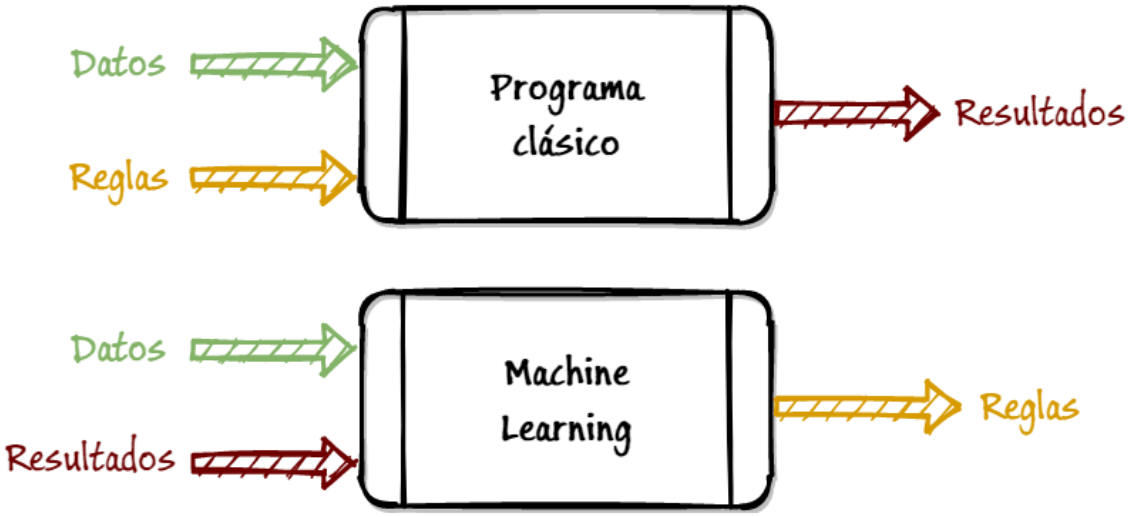

Figura 1.1: Programación tradicional frente al aprendizaje automático. *Fuente: Google.*

Por tanto, este conocimiento nos permite implementar sistemas inteligentes en los que el programa se encarga de extraer conocimiento sobre datos proporcionados en el entrenamiento, y aplicar este conocimiento en fases posteriores, lo que se conoce como **inferencia**. Se construyen, así, soluciones ***data-driven***, o guiadas por datos (ejemplos), dejando a un lado las soluciones tradicionales o ***expert-driven***. Además, se puede encontrar otra categoría de algoritmos más complejos que se engloba dentro del AA, que se conoce como **Aprendizaje Profundo** (Russell & Norvig, 2004)**,** véase Figura 1.2**.** Cuando se hace referencia al Aprendizaje Automático, habitualmente se destacan algoritmos basados en reglas, árboles, o cualquier otra representación del conocimiento que un experto en el dominio pueda entender, de forma que este último pueda verificar el modelo extraído y comprobar la eficacia del mismo de forma manual. En cambio, los algoritmos basados en Aprendizaje Profundo, soportados sobre redes neuronales (Goodfellow, Bengio, & Courville, 2016), consisten en modelos de caja negra que extraen conocimiento mediante procesos de propagación hacia adelante, ***backpropagation***, y ajuste de pesos utilizando como medida de error alguna función objetivo (Cun, 1988; Goodfellow et al., 2016; McCulloch & Pitts, 1943). Por tanto, es para un experto muy complejo interpretar el modelo una vez entrenado, y aún más difícil verificar el modelo si no es mediante un proceso de prueba y error sobre muestras reales de datos. En IA, habitualmente se pretende encontrar un equilibrio entre la **interpretabilidad** del

modelo (capacidad de un modelo para ser verificado de forma manual) y precisión del mismo. Normalmente, se cumple la norma de que precisión e **interpretabilidad** son inversamente proporcionales, y es por esto que algunas administraciones estatales tratan de evitar el uso de algoritmos cuyos modelos no puedan interpretarse, pues en caso de fallo, sería muy difícil encontrar la razón por la cual el algoritmo realizó una predicción concreta.

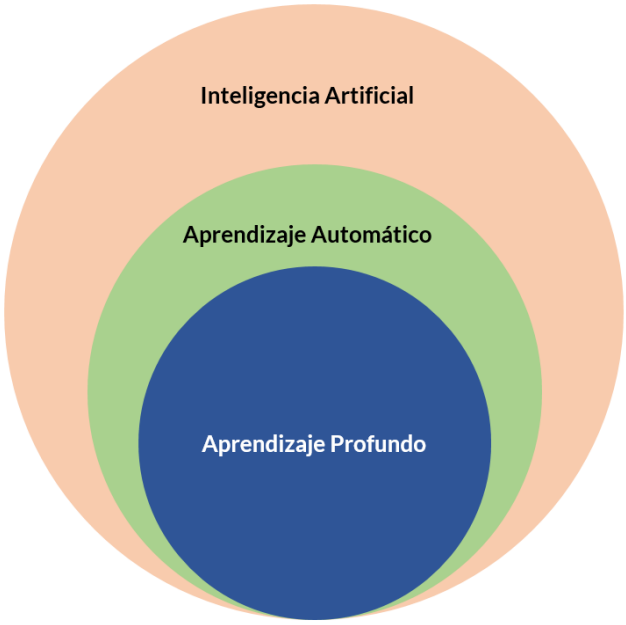

Figura 1.2: Inteligencia artificial vs. aprendizaje automático vs. aprendizaje profundo.

En multitud de problemas, es complejo encontrar una representación formal del conocimiento que describa de forma exacta y no sesgada el conocimiento del experto, de forma que puedan implementarse algoritmos que se ejecuten partiendo de esta base de conocimiento, y realicen inferencia. Es el caso, por ejemplo, de la visión por computador. Tras la aparición de numerosos modelos basados en redes neuronales, los avances en este campo han sido muy notables. Habitualmente, las tareas de **visión por computador** suelen darse entre la clasificación de entidades, esto es, reconocer qué objeto aparece en una imagen, o bien **segmentación**, tratándose en este caso de realizar clasificación de una forma diferente. Así, es complejo y prácticamente imposible conseguir una representación del conocimiento que describa formas en imágenes, por ejemplo, pues los objetos pueden estar en cualquier parte de la imagen, incluso rotados, y en diferentes escalas. El número de posibles combinaciones hace que este problema sea irresoluble mediante el enfoque ***expert-***

driven, mientras que mediante un modelo y entrenamiento adecuado se pueden conseguir porcentajes de acierto cercanos al 100%.

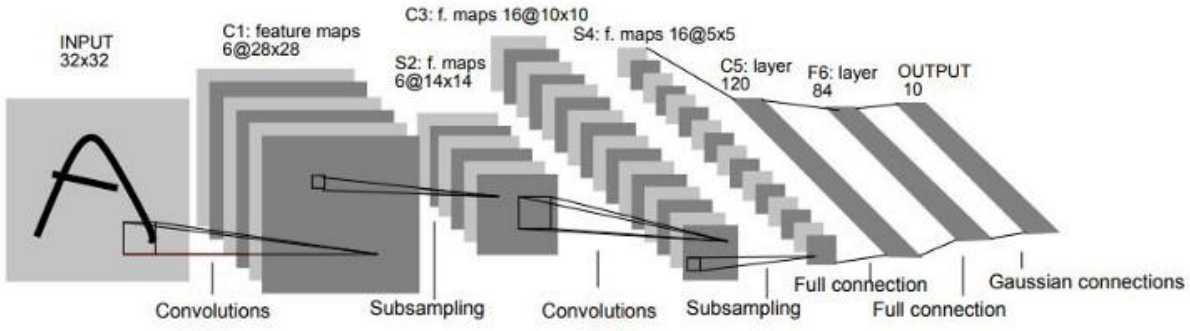

Figura 1.3: Red neuronal convolucional (LeNet-5). *Fuente: Google.*

Las redes neuronales **convolucionales** (Goodfellow et al., 2016) se ajustan perfectamente a la clasificación de imágenes, pues se trata de extractores de características que, mediante operaciones de **convolución discreta**, aplican una serie de filtros sobre las imágenes que finalmente se agrupan en un vector de características que se utiliza para predecir. En la Figura 1.3 se puede apreciar una de las primeras redes neuronales convolucionales propuestas en la literatura para reconocer dígitos (MNIST), desarrollada por **Yann LeCun**, uno de los investigadores que firman el trabajo *Geometric Deep Learning* (Bronstein, Bruna, LeCun, Szlam, & Vandergheynst, 2017), artículo sobre el que se construye el presente proyecto. Por tanto, mientras que para un experto es difícil encontrar una descripción formal del conocimiento para un problema de clasificación de imágenes, una red neuronal puede aplicar **filtros** que extraigan características de la imagen. En las primeras capas, se aplicarán filtros sencillos que normalmente buscarán bordes, esquinas, etcétera. En las siguientes capas de convolución, y dada la **naturaleza jerárquica** de las redes **convolucionales** (Goodfellow et al., 2016), se extraerán características sobre las características previamente extraídas, siendo posible detectar formas geométricas más complejas. La aplicación de operaciones de ***pooling*** permite a la red **generalizar** de una forma más óptima, esto es, centrarse en características generales y no detalles en particular, resultando finalmente en un vector de características que describe a la imagen, sobre el cual se utilizará algún modelo **perceptrón multicapa** para realizar la clasificación. El vector de características se compone de un conjunto de números

que es difícilmente interpretable para un experto humano, pues cada una de las posiciones podría ir asociada a cualquier elemento semántico de la imagen.

Sin embargo, la visión por computador no necesariamente ha de ir siempre ligada al mundo de la imagen bidimensional, sino que podrían aplicarse soluciones sobre el espacio tridimensional realizando las adaptaciones pertinentes sobre las soluciones que ya existen.

Figura 1.4: Tipos de aprendizaje automático. *Fuente: Google.*

El aprendizaje automático suele agruparse en 3 categorías clave, según la Figura 1.4:

- **Aprendizaje supervisado**. Las tareas que suelen resolverse en esta categoría consisten en clasificación y regresión. En este caso, los datos de partida están etiquetados, y el sistema recibe tanto la entrada como la salida deseada. El objetivo es encontrar un modelo de predicción que, ante nuevos datos, realice inferencia sobre los mismos utilizando el mismo esquema de conocimiento que siguen los datos con los que el sistema fue entrenado. El aprendizaje supervisado trata, por tanto, de encontrar una correspondencia $f: X \rightarrow Y$, siendo X el conjunto de datos de entrada, e Y el conjunto de datos de salida. Según sea la naturaleza

de Y, podemos distinguir entre **clasificación**, donde la salida es una etiqueta de clase y el número de posibles salidas está limitado a las clases del problema, **regresión**, donde la salida es numérica y puede pertenecer a un conjunto continuo como $\mathbb{R}$. Por ejemplo, un ejemplo del primer tipo sería identificar qué tipo de fruta aparece en una imagen, y un ejemplo claro del segundo sería tratar de identificar la edad de una persona a través de una serie de atributos de entrada. En la Figura 1.5 puede observarse un ejemplo típico de clasificación.

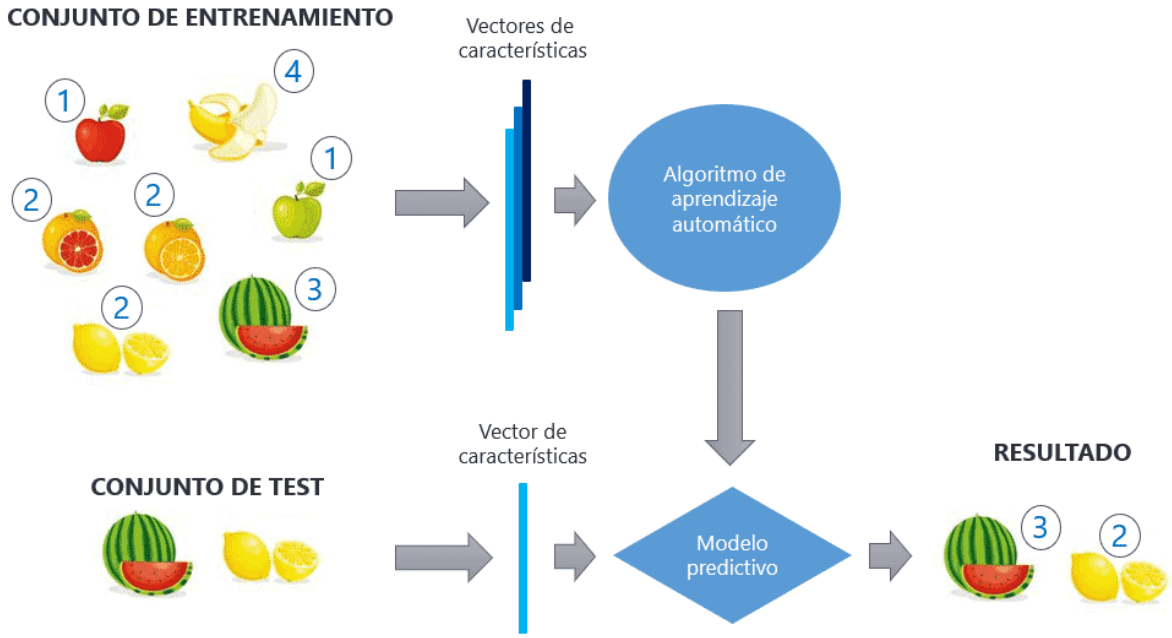

Figura 1.5: Ejemplo de aprendizaje supervisado. *Fuente: Google.*

- **Aprendizaje no supervisado.** En este tipo de problemas no se dispone de la salida asociada a los datos de entrada, por tanto, el algoritmo ha de encontrar patrones o reglas que mejor describan a los datos de entrada partiendo de las características que se tienen sobre ellos. Habitualmente, esto se conoce como ***clustering***, y suele utilizarse frecuentemente en un problema tipo como la **Segmentación del Mercado** (véase Figura 1.6, tratando este de crear campañas de *marketing* dirigido según el tipo de usuarios de una determinada compañía.

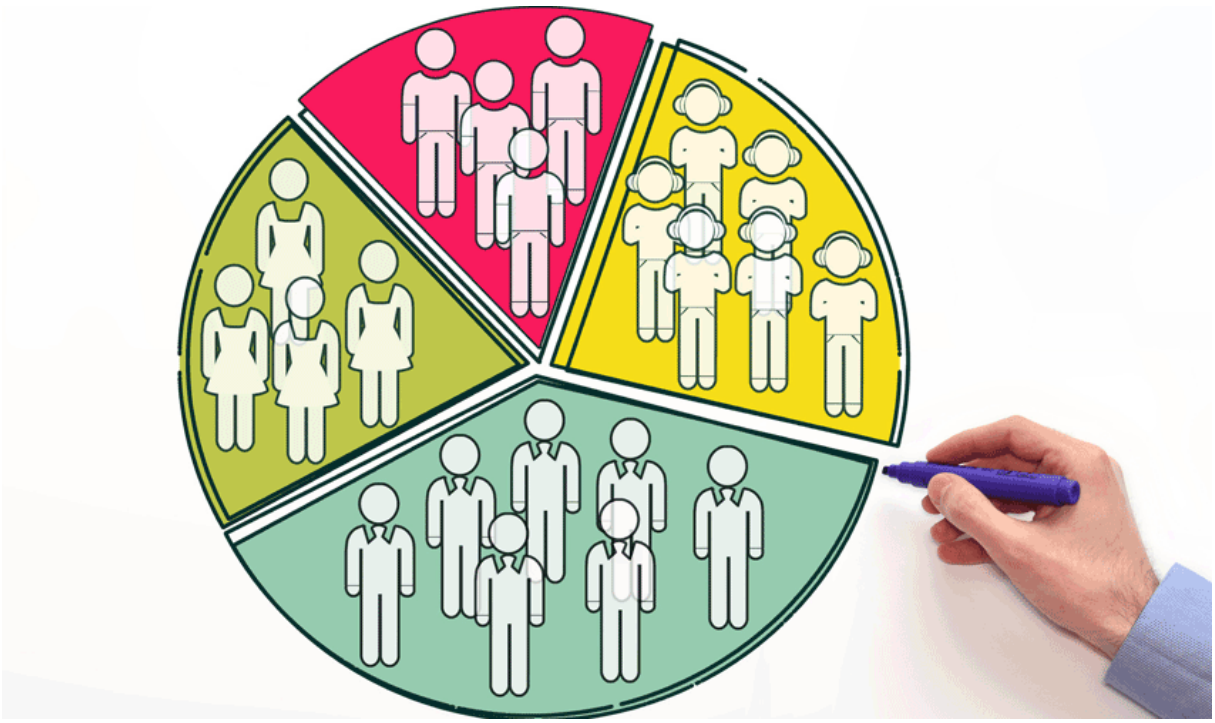

Figura 1.6: Segmentación del mercado. *Fuente: Google.*

- **Aprendizaje por refuerzo.** En esta última categoría, el sistema realiza predicciones sobre un conjunto de datos de entrada y el humano experto califica la salida como error o acierto. Según esto, el sistema tenderá a ajustarse para obtener cada vez más aciertos y realizar predicciones más acertadas, sirviéndose de *feedback* a través de recompensas tal y como se observa en la Figura 1.7.

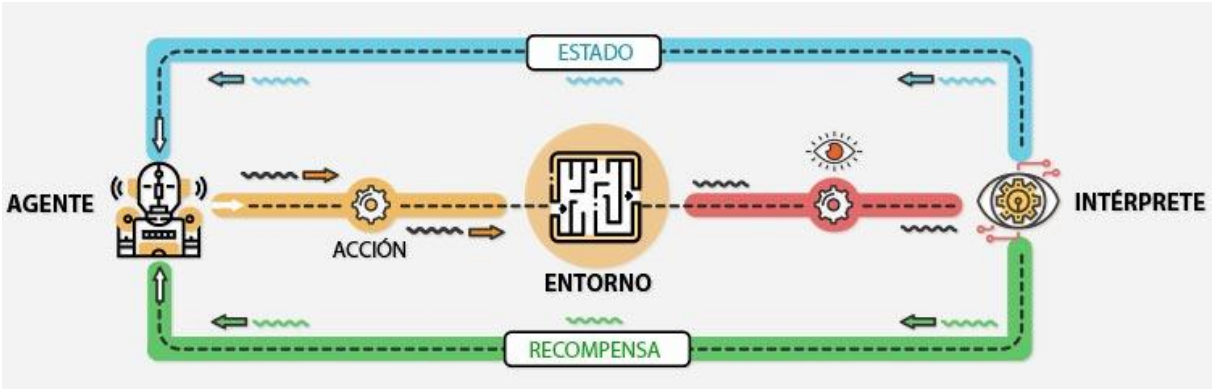

Figura 1.7: Aprendizaje por refuerzo. *Fuente: Google.*

1.1.2 Informática gráfica

No obstante, haciendo referencia a un hecho comentado justo antes, la **visión por computador** no solo describe soluciones para un formato de datos basado en imágenes (bidimensional) sino que también pueden conseguirse avances en el espacio tridimensional utilizando los mismos principios. Es aquí donde se produce una **hibridación** entre Inteligencia Artificial e Informática Gráfica, pues el principal objeto de estudio de este proyecto, las **nubes de puntos**, nacen en dicha área.

Cuando se trata de presentar información en un monitor, simulando formas de la realidad en un entorno virtual, y proyectándolas tras ello sobre un plano, entra en juego la informática gráfica. Existen diversos formatos para los modelos tridimensionales que se utilizan en informática gráfica, desde las **mallas de triángulos o *quads***, hasta las nubes de puntos. Las nubes de puntos pueden ser tanto bidimensionales como tridimensionales y, en este caso, se establecerá el foco sobre este último grupo. Se trata de conjuntos de puntos no ordenados $X = \{p \in \mathbb{R}^3\}$, donde cada elemento *p* del conjunto se compone, como mínimo, de coordenadas (x, y, z). Sin embargo, como se verá en este trabajo, cada punto puede contener más información además de las componentes espaciales.

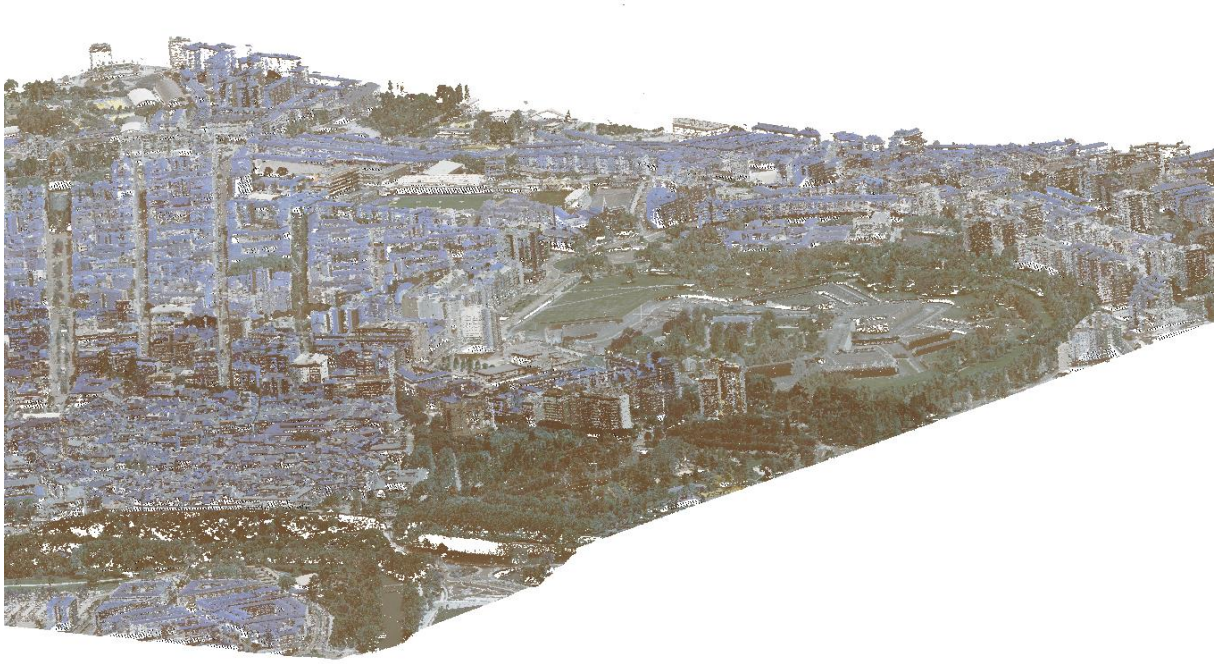

Figura 1.8: nube de puntos LiDAR de la ciudad de Pamplona (Navarra) Fuente: elaboración propia.

Los puntos contenidos en la nube no siguen ninguna estructura ni tampoco ningún orden, esto es, no puede establecerse una relación de orden tal que un punto sea mayor o menor a otro. Tampoco se dispone de ningún grafo o conexión entre los puntos, de forma que, ante cualquier intento de extracción de conocimiento, sería necesario encontrar un método de regularización de los datos de entrada que permita adecuar este tipo de datos para su aplicación sobre modelos de aprendizaje profundo.

Por otra parte, al igual que las imágenes bidimensionales son generadas mediante cámaras fotográficas, las nubes de puntos también están asociadas a una serie de métodos/dispositivos de captación para obtener modelos de puntos:

- **Fotogrametría**. La fotogrametría trata de obtener un modelo tridimensional a partir de imágenes del objeto real en sí realizadas desde diversas perspectivas, intentando cubrir en su totalidad todos los puntos de vista del objeto. Tras esto, se utilizará en postprocesamiento algún algoritmo de ***Sfm*** (Özyesil, Voroninski, Basri, & Singer, 2017)(***Structure from motion***) que estimará gradientes de color para predecir la posición de la cámara según la perspectiva del objeto, situando las imágenes en el espacio, y por último estimando puntos del objeto en sí según las posiciones de las imágenes tomadas y la concordancia entre ellas. Véase la Figura 1.9 como ejemplo de nube de puntos generada mediante fotogrametría.

Figura 1.9: Nube de puntos del Castillo de Besora generada mediante fotogrametría. *Fuente: Google.*

- **CAD/CAM.** Es posible obtener nubes de puntos mediante la interpolación de puntos contenidos en polígonos que forman objetos diseñados mediante CAD/CAM (*Computer Aided Design/Computer Aided Modelling*).
- **Escáneres.** Existen dispositivos que, mediante un proceso de emisión y reflexión de pulsos de luz, pueden generar nubes de puntos de la superficie escaneada. Un ejemplo de este dispositivo podría ser, por ejemplo, *Kinect* asociado a la consola Xbox o también (véase Figura 1.10), en un caso de uso más particular y restringido, *Face ID* como sistema de autenticación para usuarios iOS.

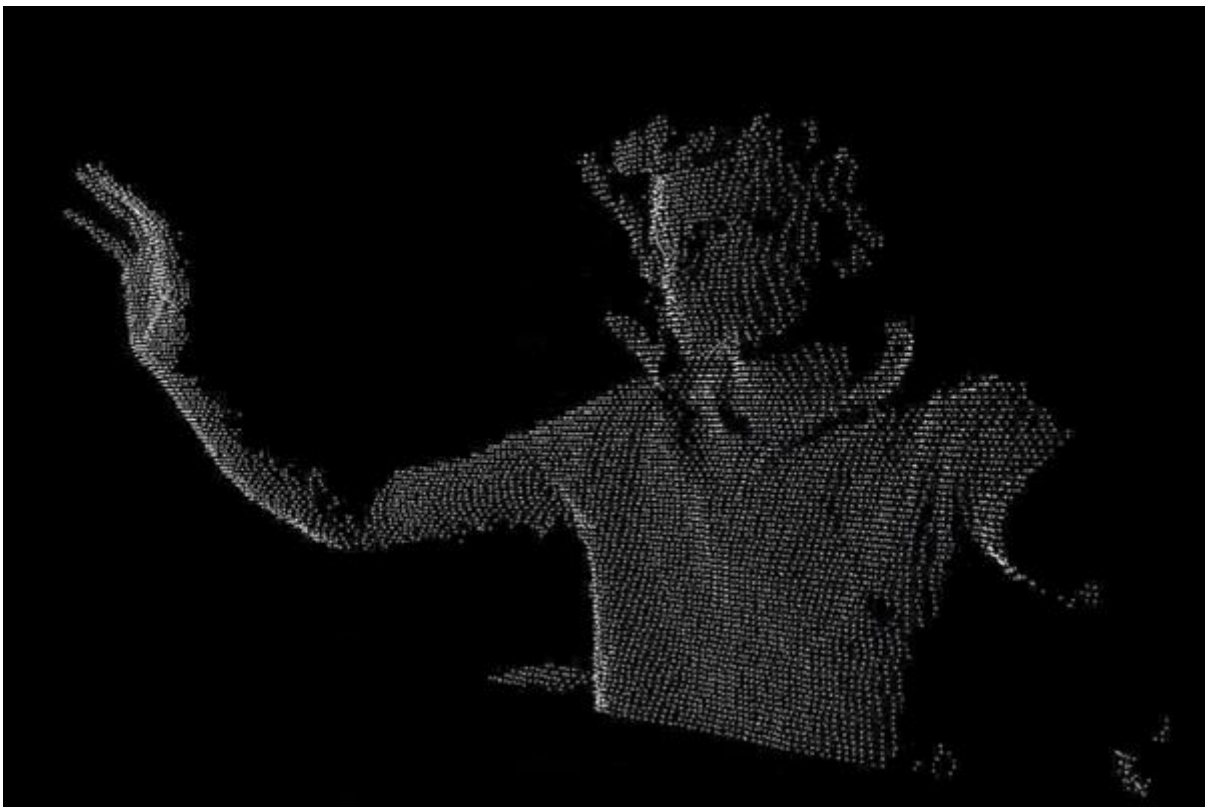

Figura 1.10: Nube de puntos generada mediante Kinect. *Fuente: Google.*

- **LiDAR (*Light Detection And Ranging*).** Aunque se trata de un tipo de escáner para obtener nubes de puntos, se trata de una tecnología muy especializada y mucho más potente que los escáneres tradicionales. Además de capturar la información geométrica con una velocidad y precisión inigualables, es capaz de obtener a la vez mucha más información sobre cada rebote como por ejemplo el color de la superficie, la intensidad (medida sobre la reflectividad de la superficie escaneada), número de retornos, información radiométrica, etcétera. Así, los datos LiDAR siguen la especificación que recibe el mismo nombre y que se ha convertido en un estándar de facto. Este tipo de escáneres generan nubes de puntos mucho más densas y con más información de los escáneres tradicionales. Como ejemplo de una nube de puntos generada mediante LiDAR, puede observarse la Figura 1.8.

Los escáneres LiDAR presentan multitud de ventajas en comparación al resto de técnicas tradicionales, aunque se trata de una tecnología excesivamente costosa desde el punto de vista de lo económico. Aun así, numerosos grupos de investigación

están invirtiendo tiempo y esfuerzo en realizar escaneos de grandes áreas geográficas, obteniendo modelos de datos cuya utilidad no va más allá de la visualización de los datos, puesto que el volumen de información y la complejidad excede a las técnicas que tradicionalmente se estaban utilizando. Además, las técnicas tradicionales para procesamiento de nubes de puntos utilizan únicamente información geométrica, siendo esto algo obsoleto, pues este tipo de escáneres recoge un mayor número de canales de información que podrían aprovecharse.

Habitualmente, cuando se trata de nubes de puntos, hay una serie de tareas clave que pretenden resolverse desde el punto de vista de la informática gráfica y visión por computador, con especial interés en geomática:

- **Clasificación.** A grandes rasgos, clasificar significa identificar qué tipo de elemento constituye una entidad. Esto es, aplicado a imágenes, clasificar una imagen tendría como resultado identificar qué objeto aparece en la imagen. En el contexto de las nubes de puntos, dada una nube de puntos cuyos elementos constituyentes componen una forma, se trata de identificar de qué forma se trata teniendo en cuenta que:
    - Las nubes de puntos son irregulares, esto es, un mismo objeto puede describirse por infinitas combinaciones de puntos y en diferentes cantidades.
    - Tan solo se dispone de la geometría que define el entorno, y no de la topología, por lo que es imposible saber qué relación existe para dos puntos cualesquiera.
    - El escaneo del objeto podría haberse realizado desde perspectivas totalmente distintas y la nube continúa siendo la misma salvo por coordenadas diferentes.
    - La nube de puntos puede que no haya sido obtenida del entorno, sino generada de forma sintética.

  En todo esto, el algoritmo de clasificación debería ser invariable a permutaciones e invariante a transformaciones geométricas. Además, el algoritmo de predicción analizará los datos de entrada y emitirá una clase de salida, que será única para todos los puntos. Esta clase

pertenecerá a un conjunto discreto definido a priori, puesto que el aprendizaje automático está lejos de alcanzar una **inteligencia artificial general** (Russell & Norvig, 2004)**,** esto es, solo se harán predicciones sobre las clases con las que el algoritmo de aprendizaje fue entrenado.

- **Segmentación**. En ocasiones como paso previo a la clasificación, la segmentación consiste en identificar regiones de puntos en una nube cuyos elementos constituyentes comparten características entre sí. Por ejemplo, dada una nube de puntos, se podría realizar una segmentación guiada por el color, de forma que regiones adyacentes de puntos cuyo color sea parecido formen segmentos de la nube. En tanto, a nivel de implementación, el resultado de la segmentación serían múltiples etiquetas para los múltiples puntos que componen la nube.

  El auge de los modelos de aprendizaje automático, y más recientemente aprendizaje profundo, ha permitido la introducción de conocimiento en los esquemas de segmentación, de forma que puede hablarse de **segmentación semántica** en lugar de **segmentación** en el sentido clásico. A lo largo del trabajo, se hará referencia en múltiples ocasiones al término **segmentación**, por lo que se deberá tener en cuenta que en realidad se está destacando la **segmentación semántica.**

  En este caso, las propiedades que compartirían los puntos como resultado de la segmentación sería la **pertenencia a la misma clase**, puesto que interviene el conocimiento extraído, y la característica que los puntos comparten sigue una semántica dotada por el experto. Según el enfoque clásico, no significa nada para un sistema el hecho de que dos puntos pertenezcan a la clase ‘coche’, puesto que las características intrínsecas de los puntos (coordenadas, color, planaridad, etc) no están relacionadas de alguna forma. Sin embargo, en el lenguaje de los humanos, es totalmente lógico el hecho de que dos puntos se parezcan porque pertenecen a la misma categoría. Así, dados los puntos y sus respectivas clases, el sistema de aprendizaje profundo aprenderá a extraer patrones geométricos para cada categoría según la semántica de los humanos.

Habitualmente, la segmentación de nubes de puntos ha venido realizándose usando técnicas clásicas cuyo procesamiento se limita al análisis de propiedades geométricas, sin tener en cuenta en ningún momento la semántica que una segmentación podría tener para los humanos. Por ello, solían utilizarse técnicas de extracción de superficies planas, ensamblado de primitivas, cuyo resultado eran agrupaciones de puntos y cuyo significado brillaba por su ausencia sin intervención de un experto durante el postproceso.

Durante los últimos años se ha abierto paso en la comunidad científica el término ***geometric deep learning*** (Bronstein et al., 2017), que no es otra cosa sino el aprendizaje profundo moderno aplicado a datos de naturaleza geométrica, como lo son los datos LiDAR y por consiguiente las nubes de puntos tridimensionales.

El incremento en potencia del hardware ha sido otra de las causas que más han contribuido a la fuerte expansión del aprendizaje profundo, pues estos algoritmos necesitan ejecutarse sobre sistemas de altas prestaciones, concretamente **GPU** (*Graphics Processing Unit*), puesto que optimizan los procesos de propagación y retropropagación (***backpropagation***) durante el entrenamiento y posterior inferencia de los mismos.

Tomando como punto de partida un trabajo previo, presentado en un congreso nacional en 2019 (Díaz-Medina, Fuertes-García, Ogayar-Anguita, & Lucena, 2019), a lo largo de este trabajo se estudiará la segmentación de nubes de puntos en formato LiDAR en el que, partiendo de un análisis del estado del arte para la técnica en cuestión, se propondrá una nueva técnica, verificando todas las posibles hipótesis de forma empírica y a través de experimentos realizados en un clúster de computación de altas prestaciones prestado por la Universidad de Jaén, introduciendo de una forma clara y concreta el ***geometric deep learning*** y su aplicabilidad sobre problemas de índole real.

1.2 Objetivos del trabajo

Los objetivos de este Trabajo Fin de Máster pasan tanto por la consolidación de conocimientos previamente adquiridos, como la consecución de hitos científicos de alto impacto en el contexto de la segmentación de nubes de puntos tridimensionales. Así, se describen los objetivos a continuación:

- Estudiar las técnicas de aprendizaje profundo para la resolución de problemas, y en concreto ***geometric deep learning*** para el tratamiento de problemas geométricos.
- Analizar y comprender la utilización de herramientas inteligentes para la resolución de problemas complejos.
- Experimentar con arquitecturas neuronales artificiales profundas para comprender su comportamiento.
- Manejar ***clusters*** de computación de altas prestaciones (**HPC, *High Performance Computing***) para ejecutar experimentos.

1.3 Antecedentes y estado del arte

La segmentación de nubes de puntos se define como el proceso de clasificar nubes de puntos en múltiples regiones homogéneas, donde los puntos en una misma región comparten las mismas propiedades (Nguyen & Le, 2013). En el caso de la **segmentación semántica**, la propiedad que comparten todos los puntos en una **subregión** es de carácter semántico, esto es, los puntos están asociados a una categoría semántica. La diferencia entre el proceso de **clasificación** y **segmentación semántica** de nubes de puntos puede observarse en la Figura 1.11.

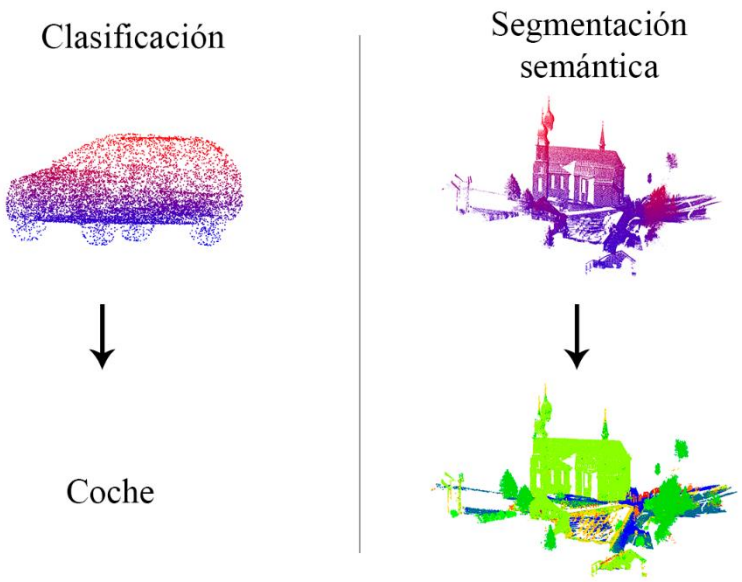

Figura 1.11: Diferencia entre el proceso de clasificación y segmentación semántica.

La segmentación supone un reto debido a la alta redundancia de puntos, densidad desigual y falta de estructura natural presentes en una nube de puntos. Este problema presenta numerosas aplicaciones en robótica, así como en vehículos inteligentes, conducción autónoma y gestión inteligente del territorio (Griffiths & Boehm, 2019; Nguyen & Le, 2013; Xie, Tian, & xiang Zhu, 2019).

Las técnicas de obtención de nubes de puntos varían desde la captación directa mediante dispositivos especializados (LiDAR o Kinect) hasta la generación de las mismas mediante técnicas de fotogrametría (algoritmo *Structure from Motion*, entre otros). Los dispositivos de captación LiDAR se caracterizan por su precisión, eficiencia, y por la cantidad de atributos adicionales que son capaces de captar para cada punto.

Debido a **la falta de estructura natural**, **alta redundancia** y la **densidad no uniforme** de puntos, los sistemas de segmentación actuales logran una precisión aceptable utilizando únicamente información geométrica que está presente de forma explícita en la nube de puntos. En cambio, con el advenimiento de la tecnología LiDAR y la unificación de la misma como dispositivo por excelencia para la captación de nubes de puntos, se dispone de información extra y adicional que podría incorporarse a los sistemas de segmentación como metainformación para la geometría y conseguir con esto una mejora en la precisión de los sistemas actuales.

El avance en la investigación ha creado una taxonomía que divide a las técnicas de segmentación en dos categorías principales: técnicas clásicas, y técnicas que hacen uso del aprendizaje profundo. En esta sección se hará un breve repaso del primer grupo, puesto que no es el objeto de estudio de este trabajo, pero aun así es importante establecer una contextualización para enmarcar este proyecto.

1.3.1 Técnicas clásicas

Las técnicas clásicas son capaces de realizar una segmentación de la nube de puntos a partir de la información geométrica y en **ausencia** de todo conocimiento, realizando meramente segmentación, y no segmentación semántica. Esto es, se basan en atributos acuñados a mano (*hand-crafted*), restricciones geométricas y reglas estadísticas (Nguyen & Le, 2013; Xie et al., 2019). El proceso de segmentación trata de agrupar puntos de la nube en regiones no solapantes, donde estas regiones

se corresponden con estructuras específicas u objetos en una escena tridimensional. Estos métodos se agrupan principalmente en 4 categorías, según (Xie et al., 2019):

- **Basadas en borde**. Estos métodos intentan extraer los bordes de los objetos presentes en la nube de puntos y distinguir con esto las diferentes regiones en la misma. Constan de dos pasos: (1) distinción de bordes y (2) agrupamiento de puntos (Nguyen & Le, 2013; Xie et al., 2019).
- **Basadas en crecimiento de regiones**. Se escogen una serie de puntos semilla al inicio del algoritmo, y se van agregando puntos hasta obtener las diferentes regiones que constituyen la nube segmentada (Besl & Jain, 1988). Existen dos enfoques: (1) *top-down*, (2) *bottom-up*.
- **Basadas en ajuste de modelos**. Tratan de hacer corresponder primitivas matemáticas geométricas como conos, esferas, etcétera, con los diferentes subconjuntos de puntos presentes en la nube. De entre todos los algoritmos de este tipo, destacaca RANSAC (***Random Sample Consensus***) descrito en (Fischler & Bolles, 1981; Xie et al., 2019).
- **Basadas en *clustering* no supervisado.** Estos métodos no requieren de la elección de los puntos semilla. Para cada enfoque de *clustering* (Filin, 2012), podrían aplicarse diferentes medidas de similitud como la distancia euclídea, densidad, vector normal, etcétera. Destacan en esta categoría *K-Means* (Shahzad, Zhu, & Bamler, 2012), *clustering* difuso (Biosca & Lerma, 2008) y *mean-shift* (Shahzad & Zhu, 2015).

Las técnicas citadas inmediatamente antes hacen uso únicamente de la información geométrica presente de forma explícita en la nube de puntos, estudiando propiedades puramente geométricas y obtenibles mediante complejos procesos de cálculos geométricos, como la estimación de curvatura o ensamblado de primitivas, los cuáles están guiados por el conocimiento de un experto que previamente ha definido el algoritmo mediante el cual se completa el proceso. Por lo tanto, el interés en cuanto a la aplicabilidad de estas técnicas se ha visto disminuido pues hoy en día se dispone de tecnología especializada que permite aplicar nuevas técnicas basadas en aprendizaje automático y, concretamente, de aprendizaje profundo. A diferencia de las técnicas anteriores, estos enfoques extraen conocimiento a partir de los datos e

intercambian el enfoque *expert-driven* por una variante *data-driven*, en el que las decisiones se toman en base a patrones contenidos en los datos y no a partir del conocimiento del experto. Desde finales del siglo pasado, los algoritmos basados en aprendizaje profundo han cobrado sentido por su habilidad de imitar el procesamiento neuronal que tiene lugar en el cerebro biológico (Goodfellow et al., 2016).

Destacan los algoritmos basados en redes neuronales como principales algoritmos propios de aprendizaje profundo (Goodfellow et al., 2016). El incremento en potencia de las GPU (***Graphics Processing Unit***) de los últimos años, así como su rápida adopción por parte de la comunidad científica, ha motivado el interés de investigadores de todo el mundo para resolver problemas de índole complejo mediante enfoques *Deep Learning*. Estos algoritmos se han convertido en un estándar de facto en tareas de reconocimiento de patrones, visión por computador, reconocimiento del lenguaje hablado, etcétera. En este sentido, sobresalen las **Redes Neuronales Convolucionales** como principales precursoras de los avances conseguidos en el campo de la visión artificial (Goodfellow et al., 2016; Krizhevsky, Sutskever, & Hinton, 2012; Simonyan & Zisserman, 2015; Szegedy et al., 2015).

En este trabajo se estudiarán algunas de las principales técnicas de segmentación semántica de nubes de puntos que están basadas en aprendizaje profundo. Pese a que cada día se publican nuevos estudios en lo relativo a enfoques de segmentación inteligente, existe poca literatura al respecto de los datos LiDAR. Así, el objeto de estudio de este trabajo será la aplicación de enfoques para segmentación semántica de nubes de puntos a los datos LiDAR en particular.

1.3.2 Técnicas del dominio de Geometric Deep Learning

A continuación, se describirán los principales avances realizados en el campo del *Deep Learning* en lo relativo a segmentación semántica de nubes de puntos.

1.3.2.1 A voxel-based deep learning approach for Point Cloud Semantic Segmentation

El trabajo (Díaz-Medina et al., 2019), presentado en el Congreso Español de Informática Gráfica (CEIG) celebrado en San Sebastián-Donostia en los días 26-28 de junio de 2019, se enmarca como conclusión al Trabajo Fin de Grado presentado previamente en el mismo mes y año y cuya línea de investigación se ha conservado

en este Trabajo Fin de Máster, abarcaba la segmentación de nubes de puntos haciendo uso de un enfoque básico basado en *Deep Learning*. El método empleado se describe a continuación.

INTRODUCCIÓN

El objetivo de esta propuesta es la consecución de un sistema de segmentación de nubes de puntos que las utiliza como entrada en su formato irregular y segmenta semánticamente los datos a partir de patrones aprendidos. Se propone una técnica de regularización de la nube de puntos, que genera *grids* de vóxeles en diferentes entornos locales de la nube, espacio donde se asignará una etiqueta de clase. En base a este conjunto de datos voxelizados, se propone una arquitectura de red neuronal convolucional que contiene capas de convolución 3D que procesan los *grids* de entrada, extrayendo patrones en forma de características. De esta forma la red infiere la etiqueta de cada punto a partir de la morfología de su entorno local.

La mayoría de propuestas que tratan nubes de puntos mediante aprendizaje automático procesaban los modelos segmentados para poder realizar una clasificación. Este trabajo propone un algoritmo de segmentación semántica basado en técnicas de etiquetado de nubes de puntos que permita realizar la tarea minimizando el conocimiento requerido a priori.

SISTEMA PROPUESTO

Este enfoque se basa en la voxelización de entornos locales de una nube de puntos para entrenar una red neuronal que aprende patrones espaciales. Para ello, deben utilizarse muestras de tamaño fijo, ya que esta es una condición de las redes neuronales convolucionales para los datos de entrada, lo que supone que siempre se consumirán *grids* de entrada de las mismas dimensiones. Hay que destacar que este enfoque no se basa en una voxelización global en un sentido estricto, esto es, la construcción de un solo modelo de vóxeles para toda la nube, sino en la generación de un número determinado de grids de vóxeles de tamaño fijo en entornos locales a lo largo de dicha nube que se corresponderán con los ejemplos de entrenamiento del *dataset* procesado. De esta forma, el sistema utiliza una red neuronal convolucional 3D que segmenta semánticamente nubes de puntos tomando como entrada *grids* de vóxeles de iguales dimensiones. Los datos de partida para el entrenamiento de la red

los constituyen nubes de puntos previamente segmentadas, donde cada punto ya tiene asignada una categoría semántica.

Este método de aprendizaje automático no necesita ni de edición manual ni de algoritmos basados en geometría, normalmente menos eficientes y precisos. Además, las redes neuronales necesitan una gran cantidad de datos de entrenamiento para ser efectivas.

PREPROCESAMIENTO DE DATOS

Las nubes de puntos utilizadas en ámbitos como la arquitectura, la planificación urbana, la gestión del territorio o la arqueología, constituyen conjuntos de datos muy densos, especialmente cuando proceden de escaneados LiDAR. Para poder tratarlos eficientemente con este enfoque inteligente, es necesario realizar un proceso de muestreo y ajuste sobre los datos espaciales de entrada, ya que debe buscarse un compromiso entre la maximización del volumen espacial a estudiar y la minimización de la cantidad de puntos a tratar. Además, todo algoritmo de aprendizaje necesita de un conjunto de datos balanceado (equilibrado) para asegurar un correcto entrenamiento, lo que supone disponer de una proporción similar en el número de ocurrencias (puntos) de cada etiqueta semántica (clase). Se proponen los siguientes tratamientos:

- **Muestreo**. Es necesario seleccionar un subconjunto representativo de los puntos de entrada para obtener un rendimiento óptimo. Se establece un nivel de precisión, ***P***, que será la distancia media entre dos puntos cualesquiera de la nube muestreada. La mínima caja envolvente de la nube de entrada se divide en un *grid* o espacio discreto de $I \times J \times K$ celdas. Cada celda de dicho espacio encierra un conjunto arbitrario de puntos, de entre los cuales se selecciona aleatoriamente un representante (para evitar un sesgo del método), que pasará a formar parte de la nueva nube muestreada. La elección del método será crucial, pues se debe mantener en todo momento la información más relevante de la nube. Si ***P*** es demasiado grande, se perderán los detalles y el método no aportará buenos resultados.
- **Balanceo del *dataset*.** En problemas de clasificación, suele ser habitual el hecho de que haya más ejemplos en una clase del problema que en

el resto de clases. Este fenómeno se conoce como *desbalanceo*, aunque realmente se trata de una mala traducción al castellano del término *unbalanced*. Para solventar esto, este trabajo selecciona exactamente N puntos de cada categoría, siendo N el mínimo número de puntos en una clase. Posteriormente, estos puntos seleccionados se convertirán en los centroides de cada una de las voxelizaciones locales con las que se alimentará a la red neuronal, si bien en dichas voxelizaciones no solo se tendrán en cuenta los puntos seleccionados como representantes de clase, sino todos los puntos originales de la nube para no perder información.

- **Voxelizaciones locales.** El núcleo del método propuesto se basa en la generación de voxelizaciones locales de tamaño fijo. Dada una nube de puntos definida sobre el espacio métrico, donde cada punto tiene coordenadas (x, y, z), se seleccionan N puntos con los criterios presentados justo antes, y se realiza una voxelización local centrada en cada uno de dichos puntos, como se aprecia en Figura 1.12. Cabe destacar que no es posible realizar una voxelización global sobre toda la nube para posteriormente seleccionar subregiones y ganar en rendimiento, dado que las regiones alrededor de cada punto seleccionado no estarán perfectamente alineadas. Para conseguir esto, la distancia entre vecinos debería ser uniforme en cada una de las tres dimensiones para toda la nube de puntos.

  Para identificar de forma correcta los patrones espaciales, es necesario que cada punto vaya acompañado de información sobre su entorno. Es necesario definir el tamaño de las voxelizaciones locales, $N \times N \times N$, así como las dimensiones de cada vóxel, esto es, la porción del espacio que representará cada vóxel. La voxelización es diferente en las fases de entrenamiento y test.

  La principal diferencia reside en que durante el entrenamiento cada *grid* de vóxeles recibe como etiqueta la mayoritaria de entre los puntos más cercanos al punto central, mientras que, durante la fase de inferencia o *test*, se generarán *grids* de manera uniforme en las tres dimensiones de la nube, se etiquetarán estos, y tras ello se etiquetarán todos los puntos

más cercanos al punto central con la misma categoría semántica predicha.

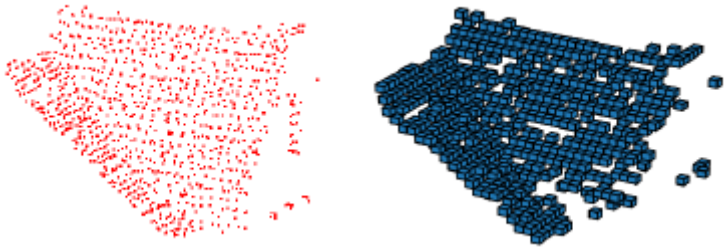

Figura 1.12: Voxelizaciones locales en (Díaz-Medina et al., 2019)

ARQUITECTURA CONVOLUCIONAL 3D

La arquitectura propuesta parte de dos capas de convolución 3D, seguida cada una de ellas de una capa de *max-pooling* 3D, concluyendo con una capa *fully-connected* tipo perceptrón multicapa (Goodfellow et al., 2016) para clasificar entre las distintas clases con las que haya sido entrenada. Este trabajo se inspira en las arquitecturas habituales para la clasificación de imágenes, extendidas en este caso a 3D. La convolución 3D surgió inicialmente para extraer patrones en datos de vídeo, aunque en nuestro caso se aplica para extraer patrones en forma de voxelización o *grid* discreto, con los que se alimentará la red. Las capas de *pooling* sirven para reducir la dimensión de los datos y resumir características, y se aplican tras cada capa de convolución. En la Figura 1.13 se observa la arquitectura de red neuronal propuesta.

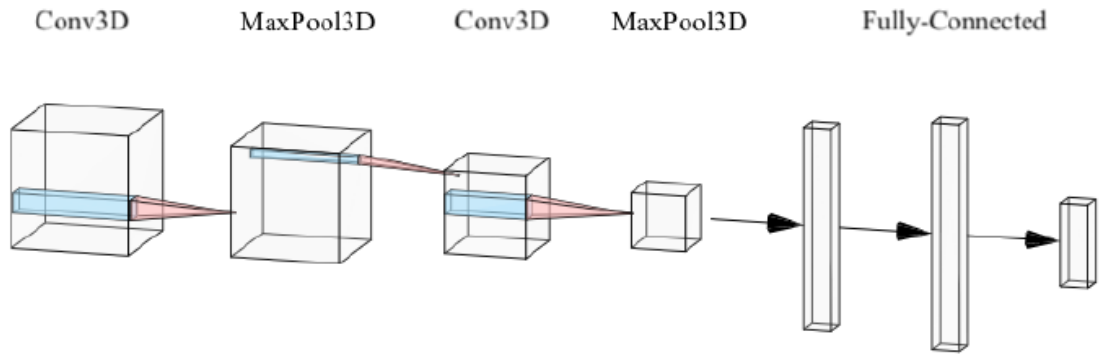

Figura 1.13: Arquitectura de red neuronal convolucional 3D propuesta en (Díaz-Medina et al., 2019)***.***

RESULTADOS OBTENIDOS

Se utiliza el *dataset* Semantic3D (Hackel et al., 2017) para evaluar la arquitectura propuesta. Utilizando un tamaño de vóxel de 0.2 metros, un *grid* de 28^3 vóxeles, tamaño de *kernel* de 5^3 en capas de convolución y de 2^3 en las capas de *pooling*, entre otros hiperparámetros, y tras 60 *epochs* de entrenamiento, se consigue una precisión (*IoU, Intersection over Union*) de 86,601% en entrenamiento y 86,376% en la fase de validación.

El hecho de que ambas cifras sean tan igualadas concluye que el entrenamiento ha sido correcto y que el modelo no se ha sobreajustado a los datos de entrenamiento. Además, puede observarse en la Figura 1.14 el resultado obtenido. Arriba, se obtiene el etiquetado semántico realizado por la red neuronal tras completar todos los *epochs* de entrenamiento. Abajo a la izquierda y en color azul se muestran los puntos mal etiquetados tras 1 *epoch* de entrenamiento frente a los puntos mal etiquetados en el último *epoch* de entrenamiento (derecha).

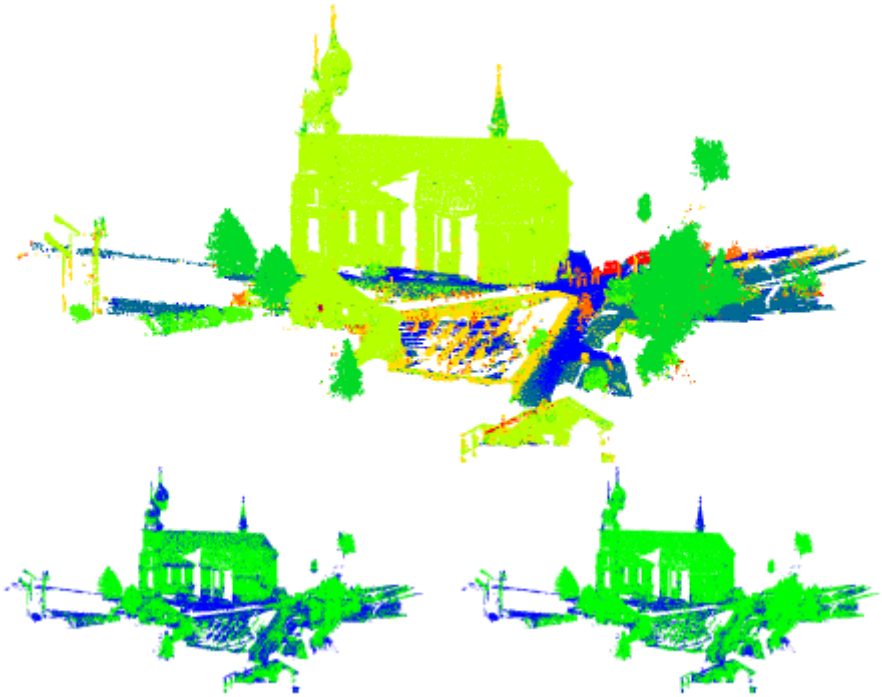

Figura 1.14: Resultado de la segmentación realizada. (Díaz-Medina et al., 2019)

CONCLUSIONES AL RESPECTO

Tras realizar numerosas experimentaciones con la arquitectura propuesta, se decidió introducir como canales adicionales a la geometría algunos datos como el

color o la intensidad. El problema es que, al utilizar voxelización, hay una pérdida importante en la precisión ya que toda variación dentro del vóxel se pierde. Dado que es bastante frecuente que dentro de un mismo vóxel se encuentren varios puntos, es necesario encontrar un estadístico lo suficientemente representativo que aglutine los valores para todos los puntos contenidos, siendo estas opciones la media, la moda o la varianza. Por tanto, ya desde la entrada se está sesgando la información que recibe la red, siendo esto un error ya que el objetivo del *deep learning* es que la red encuentre la representación de conocimiento óptima con la que alcance su mejor rendimiento.

Por otra parte, se estudió que la mayoría de vóxeles en un *grid* estaban vacíos, por el hecho de que una nube solo recoge la superficie de los objetos. Sin embargo, durante la generación de vóxeles, era necesario utilizar estructuras de datos que posteriormente fuesen aceptadas por la arquitectura, como los *arrays* tridimensionales, aunque la mayoría de celdas estuviesen vacías. Para el proceso de generación de vóxeles, realizado *offline,* se optó por utilizar un *grid* disperso. Sin embargo, posteriormente durante la carga de ejemplos para entrenar, el proceso de reconstrucción del *grid* original era muy ineficiente en tiempo, por lo que era mejor opción optar por almacenar directamente una versión compacta de los *arrays* tridimensionales conteniendo los *grids* de vóxeles.

En tanto, se concluyó que el método era ineficiente tanto en tiempo como en memoria, y no estaba adaptado para procesar grandes conjuntos de puntos como los que se utilizan en aplicaciones reales. Además, la pérdida de precisión inherente a la voxelización era un fenómeno que empeoraba gravemente el rendimiento de la técnica propuesta.

Cuando se publicó este trabajo, ya existían trabajos que proponían utilizar directamente la nube de puntos en su formato irregular como entrada a la red neuronal. *PointNet* (Charles R Qi, Su, Mo, & Guibas, 2016) es el trabajo pionero que trata de obtener una arquitectura de clasificación/segmentación usando los puntos en su formato original, sin realizar ningún tipo de preprocesamiento como la voxelización (Díaz-Medina et al., 2019; Huang & You, 2016) o la proyección en imágenes RGB-D (Eitel, Springenberg, Spinello, Riedmiller, & Burgard, 2015). Tras la publicación de *PointNet*, numerosos grupos de investigación a nivel mundial comenzaron a proponer técnicas similares, dando paso a lo que se conoce como *Geometric Deep Learning*

(Bronstein et al., 2017), siendo este último uno de los pilares del presente trabajo. A continuación, se realizará una breve descripción del trabajo en cuestión.

1.3.2.2 Geometric Deep Learning: going beyond euclidean data

Numerosos científicos alrededor del mundo estudian datos con una estructura no euclidiana subyacente. Por ejemplo, destacan las redes sociales, redes de sensores en comunicaciones, redes regulatorias en genética, y por último y más relacionado con nuestro caso de estudio, mallas de triángulos utilizadas en el campo de la informática gráfica.

Las redes neuronales profundas han demostrado ser una herramienta muy potente para resolver problemas clásicos de **visión por computador, procesamiento del lenguaje natural**, e incluso análisis de señales de audio. Sin embargo, la geometría euclidiana o bien los formatos regulares subyacen a todos estos problemas, y los modelos actuales de ajustan de forma natural a este tipo de formatos, por lo que los avances son más que posibles.

La expresión *Geometric Deep Learning* hace referencia a una generalización de los modelos *Deep Learning* tradicionales a dominios no euclidianos, como por ejemplo grafos o mallas (en el caso de informática gráfica).

INTRODUCCIÓN

El término *Deep Learning* se refiere al aprendizaje de conceptos complejos a partir de la construcción jerárquica de conceptos más simples. Uno de los factores de éxito de las redes neuronales profundas reside en su habilidad para aprovecharse de las características estadísticas intrínsecas a los datos como por ejemplo la estacionalidad y composición a través de características locales que están presentes en imágenes, vídeo y voz.

Mientras que los modelos *Deep Learning* han triunfado particularmente al tratar con señales como grabaciones de voz, imágenes o vídeo, donde subyace una estructura euclidiana, en los últimos años se ha gestado un creciente interés en lo referente al intento de aplicar estas exitosas soluciones sobre geometría no euclidiana. Este tipo de datos se encuentran en multitud de ámbitos, como las redes sociales o bien la geometría por computador. En informática gráfica y visión artificial, los objetos 3D son modelados utilizando **variedades de Riemann** acompañadas por

canales de información como el color (textura). Una **Variedad de Riemann** es una variedad diferenciable real en la que cada espacio tangente se acompaña con un producto interno de forma que varíe de forma leve en cada punto de la superficie de la variedad. Esto permite que se definan varias nociones métricas como longitud de curvas, ángulos, áreas (o volúmenes), curvatura, gradiente de funciones y divergencia de campos vectoriales.

La naturaleza no euclidiana de este tipo de datos implica que no existen propiedades familiares como la parametrización global, un sistema común de coordenadas, estructura espacio-vectorial o invariabilidad a transformaciones. De este modo, algunas operaciones básicas como la convolución, que se ajustan de forma natural a problemas con una naturaleza euclidiana subyacente, no se definen de forma automática para el caso no euclidiano.

PROBLEMAS DE APRENDIZAJE GEOMÉTRICO

Principalmente, se distingue entre dos tipos de problemas que están relacionados con el aprendizaje geométrico. En el primer tipo de problemas, el objetivo es la caracterización de la estructura de los datos. En el segundo tipo, se trata de analizar funciones definidas en un dominio no euclidiano dado.

- **Estructura del dominio**. Como un ejemplo de la primera clase de problemas, se tiene un conjunto de datos con una estructura de baja dimensión subyacente incrustada en un espacio euclídeo de altas dimensiones. En estos problemas, se trata de aplicar métodos para la reducción de la dimensionalidad que preserven la naturaleza implícita en los datos, salvando las conexiones existentes. Destaca, por ejemplo, t-SNE (van der Maaten & Hinton, 2008).

  En algunos casos, los datos son presentados como variedades o grafos. Por ejemplo, en informática gráfica y aplicaciones de visión artificial, se pueden analizar formas tridimensionales como mallas mediante la construcción de descriptores geométricos locales capturando propiedades como la curvatura.

- **Datos en un dominio.** La segunda clase de problemas trata de analizar funciones definidas en un dominio no euclidiano. Incluso, esta clase

puede subdividirse en dos subclases: problemas donde el dominio es fijo, y problemas con múltiples dominios.

En visión artificial e informática gráfica, encontrar similitudes y correspondencias entre formas es un ejemplo de la segunda subclase de problemas: cada forma está modelada como una variedad, y es necesario trabajar en múltiples dominios.

El interés en aplicar técnicas de *Deep Learning* sobre variedades o grafos se ha incrementado en los últimos años, resultando en numerosos intentos de aplicar estos métodos en un amplio espectro de problemas desde la bioquímica hasta los sistemas de recomendación.

DEEP LEARNING EN DOMINIOS EUCLIDIANOS

A priori, es necesario recordar algunos conceptos relativos a geometría básica. Considérese un dominio compacto d-dimensional euclídeo, $\Omega = [0,1]^d \subset \mathbb{R}^d$ donde se definen funciones cuadradas integrables $f \in L^2(\Omega)$. Por ejemplo, en el caso de las imágenes $d = 2$ puesto que las imágenes se construyen a partir de *grids* bidimensionales. Se considera un ajuste de aprendizaje supervisado, donde se aprende una función desconocida y: $L^2(\Omega) \to Y$.

En una configuración de aprendizaje supervisado, el espacio objetivo Y puede idearse de forma discreta tomando $|Y|$ como el número de clases del problema. Esto se adaptaría en función del tipo de problema (clasificación multiclase, regresión, etcétera).

En la amplia mayoría de tareas de visión artificial y análisis de voz, hay algunas asunciones sobre la función desconocida y que se pretende aprender, como siguen:

- **Estacionalidad**. Ha de definirse un operador de traslación que dote a la de traslación.

$$T_v f(x) = f(x - v), \;\; x, v \in \Omega$$

- **Deformaciones locales y separación de escala.** De igual forma que el caso anterior, se define un operador que aporte una capacidad de invariabilidad a transformaciones geométricas de escalado o deformaciones sobre los datos de entrada.

- **Redes Neuronales Convolucionales.** Las dos propiedades anteriores se consiguen de forma automática con este tipo de aproximaciones

GEOMETRÍA DE VARIEDADES Y GRAFOS

El principal objetivo de este trabajo es la generalización de las redes neuronales convolucionales (CNNs) tradicionales a la geometría no euclidiana. Habitualmente, se hace referencia a dos estructuras prototipo: variedades y grafos.

- **Variedades**. Se trata de un espacio que es euclídeo localmente. El ejemplo por antonomasia para este tipo de estructuras es el propio planeta tierra. En cualquier punto de la superficie, se tiene una apariencia plana, es decir,de planaridad en todos los puntos de la superficie. De manera formal, una variedad d-diferenciable χ es un espacio topológico donde cada punto x tiene un vecindario que es topológicamente equivalente (homeomorfo) a un espacio euclídeo d-dimensional, llamado espacio tangente. La colección de los espacios tangentes en todos los puntos se conoce como conjunto tangente.

Las variedades bidimensionales (superficies) embebidas en $\mathbb{R}^3$ se utilizan en informática gráfica y en visión artificial para describir la superficie de los objetos tridimensionales, coloquialmente llamadas formas 3D. El término 3D hace referencia al espacio en el que se encuentras embebidas, y no tanto a la dimensión de la variedad en cuestión.

En nuestro caso, el objeto de estudios son las nubes de puntos y, aunque recogen representaciones de formas existentes en la realidad, las cuales serían óptimamente representadas mediante variedades tridimensionales, en este caso solo disponemos de los vértices y no tenemos ninguna información sobre latopología, necesaria para obtener las aristas y finalmente, las superficies de frontera. Por tanto, y tomando siempre como base el complejo estudio realizado en *Geometric Deep Learning* (Bronstein et al., 2017), se da paso a una nueva categoría de técnicas que tienen como foco el estudio de la aplicabilidad del aprendizaje profundo sobre conjuntos de puntos no ordenados, como las nubes de puntos tridimensionales.

Trabajos como *PointNet* (Charles R Qi et al., 2016; Charles Ruizhongtai Qi, Yi, Su, & Guibas, 2017) o *Geometric Deep Learning* (Bronstein et al., 2017) motivaron a la comunidad científica a investigar nuevas formas de abordar el problema de

clasificación o segmentación semántica, y también definieron una nueva taxonomía en el campo del procesamiento inteligente de nubes de puntos.

A continuación, se introduce un trabajo a modo de *survey* que recoge la principal taxonomía establecida, así como las diferencias entre unos grupos de técnicas y otros. Habitualmente, las técnicas se agrupan según el modelo de red neuronal artificial utilizado.

1.3.2.3 Deep Learning on Point Clouds and Its application: a survey

En este trabajo (Liu, Sun, Li, Hu, & Wang, 2019), los métodos actuales para aprendizaje de características en nubes de puntos son clasificados como basados en puntos (*point-based)* o basados en árboles (*tree-based).*

INTRODUCCIÓN

Como preprocesamiento, muchos enfoques se han centrado en aumentar la densidad de la nube de puntos usando el agrupamiento de imágenes (*dense image matching*), consistiendo en aplicar algún algoritmo de tipo *Structure from motion* y aumentar la densidad de la nube de puntos original. Otra estrategia es utilizar datos complementarios obtenidos con otras técnicas como, por ejemplo, combinar datos generados mediante ***Sfm*** con datos obtenidos mediante láser.

Hay tres propiedades básicas que se cumplen en una nube de puntos:

- **Falta de orden**. No es posible definir una relación de orden en el conjunto de puntos que compone la nube de forma que los puntos sean comparables entre sí.
- **Interacción entre puntos**. Todo punto está rodeado por un vecindario, y es este vecindario lo que caracteriza a cada punto.
- **Invariabilidad a transformaciones geométricas.** Cualquier transformación geométrica (traslación, rotación o escalado) no debería modificar la naturaleza de la nube de puntos.

Las características en una nube de puntos que describen propiedades estadísticas pueden ser divididas en **intrínsecas** y **extrínsecas**, siendo invariantes a ciertas transformaciones. Algunos trabajos han optado por una transformación previa de la nube de puntos en otro tipo de formato regular y más adaptable, pero esta

discretización de la información o la extracción de descriptores en el paso previo al entrenamiento de redes neuronales supone una pérdida de precisión presente en la nube original.

APRENDIZAJE DE CARACTERÍSTICAS EN NUBES DE PUNTOS

Este trabajo define la siguiente taxonomía para las técnicas de aprendizaje profundo que procesan nubes de puntos:

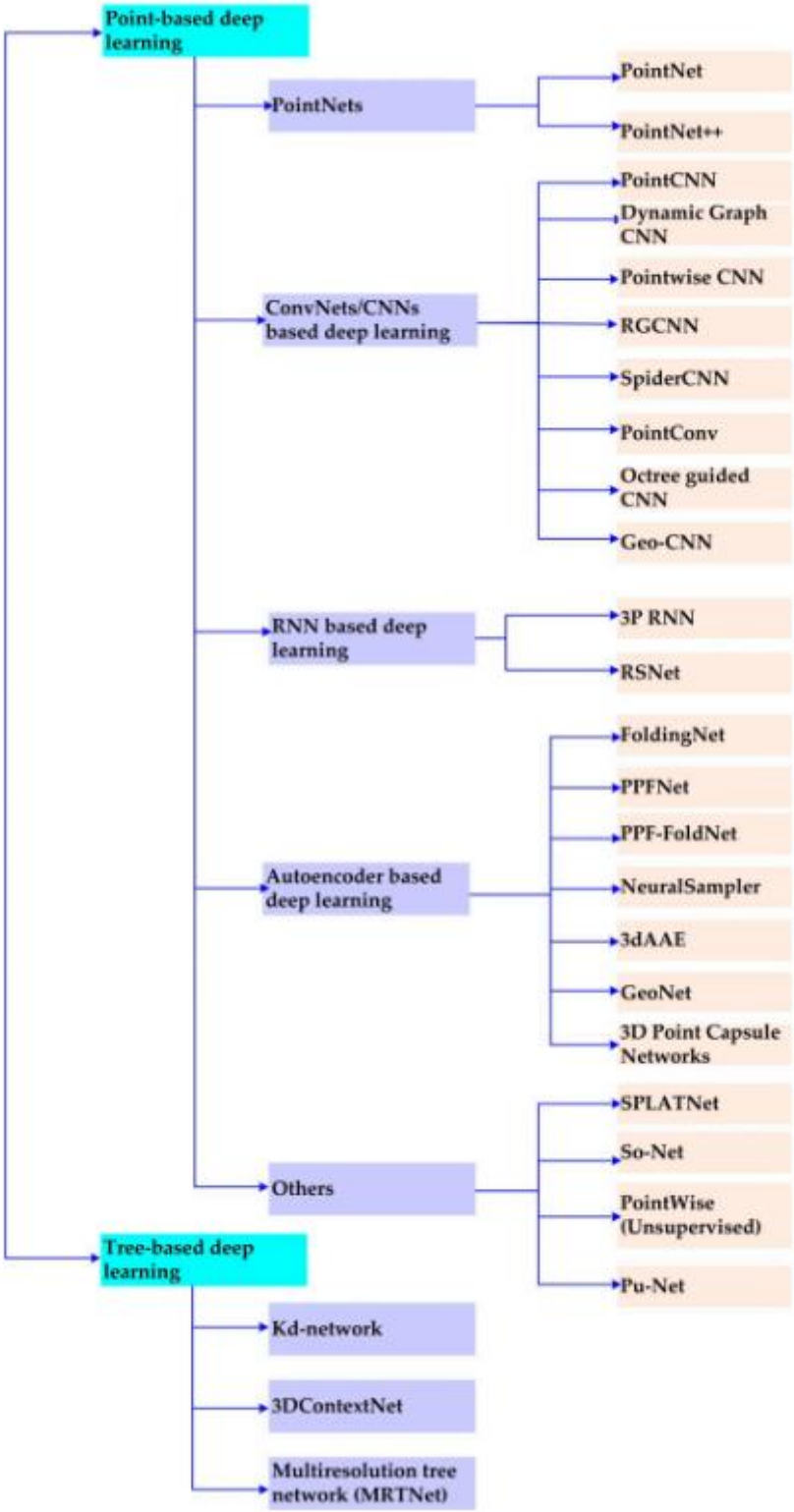

Figura 1.15: Taxonomía de los métodos Deep Learning para aprendizaje sobre nubes de puntos (Liu et al., 2019)***.***

APRENDIZAJE SOBRE CONJUNTOS DE PUNTOS

Aprendizaje al estilo de *PointNet*

Destacan, en este sentido, *PointNet* y *PointNet*++ (Charles R Qi et al., 2016; Charles Ruizhongtai Qi et al., 2017). *PointNet* presenta problemas y no puede capturar características locales en nubes de puntos, por ello que *PointNet*++ fue propuesto. Uno de los puntos fuertes de *PointNet* es el uso de una *Spatial Transformer Network* (STN) para resolver el problema de la rotación. En la comunidad de visión por computador, una STN genera una matriz de rotación para mantener la nube de puntos en una forma canónica.

PointNet tiene la capacidad de tratar el problema de la falta de orden y la invarianza a transformaciones. Es por ello que las características globales de la nube pueden ser extraídas a través de *Max-Pooling*. Como el perceptrón multicapa solo aprende las características locales de cada punto e ignora las conexiones entre puntos, *PointNet* falla al representar características locales de puntos vecinos, limitando el rendimiento.

PointNet++ fue propuesto construyendo una pirámide de agregación de características. Hay 2 aspectos clave en este trabajo: cómo dividir la nube localmente, y, cómo extraer características locales de la nube de puntos. Para lo primero, se propone un aprendizaje de características jerárquico. Consiste en 3 componentes: capa de muestreo, capa de agrupamiento, y capa *PointNet*. Para el segundo aspecto, *PointNet*++ utiliza *PointNet* para extraer características locales después de agrupar las nubes de puntos.

Aunque *PointNet*++ puede codificar las características locales de la nube de puntos, es agnóstico sobre la distribución espacial de la nube de puntos de entrada. Esto es porque el aprendizaje de características jerárquico falla al codificar la distribución espacial en la división de nubes de puntos.

Aprendizaje basado en redes convolucionales

Inspirada por procesos biológicos, la arquitectura de red convolucional (Szegedy et al., 2015) es similar a la organización del córtex visual en animales. Cada neurona del córtex solo responde a un estímulo en el campo receptivo en el cerebro.

Para cubrir el campo completo, hay un área solapante entre los campos receptivos de varias neuronas. Actualmente, hay 7 modelos que trabajan sobre nubes de puntos:

- ***Dynamic Graph CNN*** (Wang et al., 2018). Se trata de una red para clasificar y dividir datos de nubes de puntos y es una modificación inspirada por *PointNet* y *PointNet*++. Para obtener características locales, se utiliza *EdgeConv* para extraer características, que soluciona la extracción de características locales de *PointNet*. Esta será la técnica que inspirará la sección técnica de este trabajo teórico-experimental, la cual se implementará desde cero usando un lenguaje de programación, y tratará de mejorarse utilizando características innovadoras presentes en la literatura más reciente.
- ***PointCNN*** (Li et al., 2018). Utiliza convolución jerárquica y operadores x-Conv para capturar información local. El beneficio de este operador es que considera las formas de los puntos con independencia al orden de los datos. De forma similar a la red de transformación espacial (STN), K puntos se cogen de la capa previa para predecir una matriz de transformación de tamaño K * K. Las características en la capa anterior de la matriz x son transformadas, y luego las características transformadas son convolucionadas. La capacidad de aprendizaje de este modelo es muy fuerte, pero su capacidad de generalización es bastante pobre.
- ***Regularized Graph* CNN (RGCNN)** (Te, Hu, Zheng, & Guo, 2018). Consume nubes de puntos con irregularidad y ruidosas. Se ha utilizado en clasificación y segmentación semántica. Hay dos características principales de este modelo: 1) toma las características de los puntos como un nodo en el grafo basado en la teoría de grafos espectrales para tratar la irregularidad de la nube; 2) introduce la operación de convolución por aproximación polinomial de *Chebyshev* para filtrado localizado.
- ***Pointwise* CNN** (Hua, Tran, & Yeung, 2018). Se introduce una operación de convolución a nivel de punto. Esta arquitectura puede ser efectiva para aprender características locales debido a la naturaleza de la

operación de convolución, que utiliza un pequeño *kernel* (3x3) para extraer características. A diferencia de la convolución tradicional en imágenes, solo hay operación de convolución en Pointwise CNN sin submuestreo o sobremuestreo en las nubes de puntos.

- ***PointConv*** (Wu, Qi, & Fuxin, 2019). Se trata de una nueva operación de convolución y puede ser utilizada para construir redes neuronales profundas tratando la irregularidad y desorden asociado a las nubes. Presenta escalabilidad para tratar con nubes de puntos invariables a rotación y a traslación.
- ***Geo*-CNN** (Lan, Yu, Yu, & Davis, 2018). Codifica la estructura geométrica para un punto y su vecindario a través de una operación convolucional. En primer lugar, características de borde son extraídas por GeoConv para codificar la estructura geométrica con un vector y descompuesta en 3 direcciones ortorrómbicas. En segundo lugar, las características destiladas de estas direcciones son combinadas para representar la estructura geométrica de las nubes de puntos con el vector y 3 bases para adquirir características locales.
- ***SpiderCNN*** (Xu, Fan, Xu, Zeng, & Qiao, 2018). La convolución propuesta es una extensión de *grids* regulares a conjuntos de puntos irregulares. El filtro de la convolución es el producto de funciones escalonadas para codificar la información local geométrica de las nubes de puntos. El polinomio de Taylor se utiliza para asegurar la expresividad de este modelo.

Aprendizaje basado en redes recurrentes: memoria

Siendo capaces de capturar el contexto local, la redes neuronales recurrentes bidireccionales se han aplicado a *Pointwise Pyramid Pooling* RNN (3P RNN) (He, Zhang, Ren, & Sun, 2014) y *Recurrent Slice Networks* (RSNets).

Aprendizaje basado en *autoencoders*

Un *autoencoder* trabaja de forma muy similar a PCA (*Principal Component Analysis*), trata de reconstruir un ejemplo de entrada utilizando una arquitectura de red en forma de reloj de arena, minimizando la dimensionalidad de los datos de entrada,

y volviendo a aumentar la misma para obtener el ejemplo original con el mínimo error posible.

Estos métodos pueden utilizarse para aprender la representación de los datos de una forma no-supervisada. Tienen la capacidad de codificar la irregularidad de las nubes de puntos y tratar la dispersión en la fase de reconstrucción del ejemplo original (*up-sampling*).

- ***FoldingNet*** (Yang, Feng, Shen, & Tian, 2017). Para representar las nubes de puntos de 2D a 3D con pequeños errores de reconstrucción.
- ***Point Pair Feature Network***. Diseñada para aprender características 3D globales y descubrir correspondencias en nubes de puntos dispersas y sin orden. Se introduce una función de pérdida N-tuple para incrementar la diferencia entre clases y decrementar las variaciones intra-clase.
- ***PPF-FoldNet***. Propuesta para lidiar con el problema de que PPFNet era sensible a la rotación de las nubes de puntos y también fue utilizada para descriptores locales 3D no supervisados en las nubes de puntos.
- ***NeuralSampler***. Trata con nubes de puntos de varios tamaños y se ha utilizado para clasificación de objetos.
- ***GeoNet***. Se propuso para codificar información de conectividad de las nubes de puntos. Hay 2 componentes en esta arquitectura: 1) un autoencoder para extraer el vector de características para cada punto y 2) una capa de matching geodésico que actúa como una función de kernel aprendido para estimar los vecindarios geodésicos utilizando vectores latentes.

Otros tipos de aprendizaje

Excepto para los 4 tipos anteriores, muchos investigadores han utilizado estrategias especiales para tratar con el *dataset* de puntos.

- ***Self-Organizing Network***. Se trata de una red invariante a permutaciones. Utiliza la distribución espacial de la nube de puntos diseñando una red con un arreglo constante y simula la distribución espacial construyendo un mapa auto-organizativo. Destaca por el

paralelismo y la velocidad de aprendizaje. Para segmentación semántica.

- ***Sparse Lattice Network*** (SPLATNet) (Su et al., 2018). Para calcular características jerárquicas y espaciales en la nube de puntos, se propone un filtro disperso y eficiente de malla en una rejilla con un alto número de dimensiones.
- ***Pu-Net***. Utiliza la convolución multi-rama para adquirir características expandidas, que luego se divide para reconstruir la nube de puntos.
- ***Point Contextual Attention Network*** (PCAN). Primero aplica *PointNet* para extraer características locales y luego usa una capa NetVLAD para agregar características globales.

Aprendizaje basado en árboles

Debido a la irregularidad de las nubes de puntos, se propusieron enfoques basados en *Kd-tree* para explorar el contexto global y local. Los modelos basados en Kd-tree toman nubes de puntos como representaciones regulares antes de introducir información a modelos de *Deep Learning*. En este sentido, destacan Kd-network, 3D contextual network (*3DContextNet*) y *Multiresolution Tree Networks* (*MRTNet*). Los 2 primeros han sido utilizados para segmentación semántica.

1.3.2.4 KP-Conv: Flexible and Deformable Convolution for Point Clouds

Los pesos de convolución de KPConv están localizados en el espacio euclídeo por puntos de kernel, y son aplicados a los puntos de entrada que son cercanos a ellos. Su capacidad de utilizar cualquier número de cualquier número de puntos de kernel otorga a KPConv más flexibilidad que las convoluciones fijadas (en número de kernels). Las localizaciones de los puntos son continuas en el espacio y, por tanto, pueden ser aprendidas por la red. KPConv puede extenderse a convoluciones deformables que aprenden a adaptar los puntos del kernel a la geometría local. KPConv es eficiente y robusto ante densidades variables gracias a una estrategia de muestreo regular.

INTRODUCCIÓN

Una nube de puntos presenta una estructura dispersa que tiene la propiedad de carecer de orden. En cambio, comparte una propiedad común con los *grid* regulares: está espacialmente localizada. En un *grid*, las características están

localizadas por sus índices en una matriz mientras que, en una nube de puntos, están localizadas por sus respectivas coordenadas de puntos. Por ello, los puntos están considerados elementos estructurales, y las características como los datos reales.

KPConv (Thomas et al., 2019) está inspirado por la convolución aplicada sobre imágenes, pero en lugar de kernels en formato de píxeles, se utiliza un conjunto de kernels en formato de puntos para definir el área donde cada peso del kernel es aplicado. Así, los pesos de los kernels son transportados por los puntos, al igual que las características de entrada, y su área de influencia está definida por una función de correlación.

Además, se propone una versión deformable de este tipo de convolución, que consiste en aprender desplazamientos locales que son aplicados sobre los puntos del kernel. La red genera diferentes desplazamientos en cada localización de la convolución, de manera que puede adaptarse a la geometría local en diferentes regiones de la nube de puntos. Se necesita regularización para ayudar a los *kernels* deformados a ajustarse a la nube de puntos y evitar espacios vacíos. Se utiliza el Campo Receptivo Efectivo (EFF) para comparar la versión deformable con la versión rígida.

Se utiliza el vecindario de radio (todos los vecinos en un radio r) en lugar de kNN, dado que kNN no es robusto a densidades variables. La robustez de este tipo de convolución a densidades variables se asegura con la combinación de vecindarios de radio y un muestreo regular de la nube de puntos original.

Para segmentación de nubes de puntos a gran escala, se ha comprobado que la versión que mejor trabaja es la versión deformable.

TRABAJOS PREVIOS

Destaca *Pointwise CNN* (Hua et al., 2018), que localiza los pesos de los kernels con contenedores vóxeles, y pierde la flexibilidad como las redes grid. Además, su estrategia de normalización sobrecarga la red con cómputos innecesarios. KPConv mejora en eficiencia.

SpiderCNN define su *kernel* como una familia de funciones polinomiales aplicadas con un peso diferente para cada vecino. El peso aplicado a un vecino depende de la distancia al vecino, haciendo los pesos espacialmente inconsistentes.

En contraste, los pesos de *KPConv* están localizados y esto resulta en invariabilidad al orden de entrada de los puntos.

Flex-convolution utiliza funciones lineales para modelar su kernel, lo que podría limitar su fuerza representativa. También utiliza *k nearest neighbors* o cálculo de los vecinos más cercanos, que no es robusto a densidades variables.

PCNN (Li et al., 2018) es el más cercano a KPConv. Su definición también utiliza puntos para transportar los pesos de los kernel, y una función de correlación. En cambio, su diseño no es escalable porque no utiliza ninguna forma de vecindario, llevando la complejidad a $O(n^2)$ en el número de puntos. Además, utiliza una correlación Gaussiana, mientras que *KPConv* utiliza una correlación lineal más sencilla, que simplifica el proceso de aprendizaje de las deformaciones.

KERNEL POINT CONVOLUTION

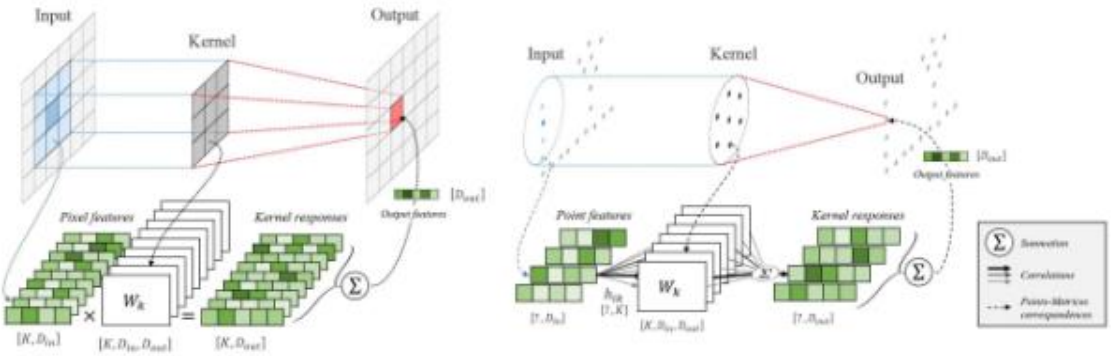

Figura 1.16: Diferencia entre la convolución clásica y la convolución definida por KPConv.

Función de Kernel definida por puntos

KPConv puede formularse con la definición general de convolución sobre puntos. Sean x_i, y_i los puntos de P^{Nx3} y sus respectivas características F^{NxD}. La convolución general de puntos de F por un *kernel g* sobre un punto $x \in \mathbb{R}^3$ es definida como:

$(F * g)(x) = \sum_{x_i \in N_x} g(x - x_i) f_i$, donde N_x representa al conjunto de vecinos de x en un radio r, tal que: $N_x = \{x_i \in P\ t.q. \|x_i - x\| \leq r\}$. La parte crucial de la ecuación anterior es la definición de la función de kernel g. g toma las posiciones de los vecinos centrados en x como entrada. Como el vecindario está definido por un radio r, el dominio de la definición de *g* es la bola $B_r^3 = N_x$ descrita anteriormente. Al

igual que ocurre en los kernel de convolución en imágenes, se pretende que g aplique diferentes pesos en diferentes áreas en su dominio. Existen diferentes formas de definir áreas en un espacio 3D, y los puntos son los más intuitivos ya que las características están también localizadas por ellos. Sea $\{\overline{x_k} \mid k < K\} \subset B_r^3$ el conjunto de puntos de *kernel* y $\{W_k \mid k < K\} \subset R^{D_{in}xD_{out}}$ los pesos asociados que mapean características de la dimensión D_{in} a la dimensión D_{out}. Se define la función de kernel *g* para cualquier punto $y_i \in B_r^3$ como:

$$g(y_i) = \sum_{k<K} h(y_i, \bar{x}_k)\, W_k \ ,$$

donde *h* es la correlación entre *x* e *y*, que debería ser más alta cuanto más cerca esté x_k de y_i. En este trabajo, se usa la correlación lineal: $h(y_i, \overline{x_k}) = max\,(0, 1 - \|y_i - x_k\| / \sigma)$, donde σ representa a la distancia de influencia de los puntos de *kernel* y se escogerá de acuerdo a la densidad de la entrada.

Kernel Rígido o Deformable

Las posiciones de los puntos del *kernel* son críticas para el operador de convolución. Los *kernels* rígidos, en particular, necesitan disponerse de forma regular para trabajar de forma eficiente. Es necesario buscar una disposición regular para cada *kernel K.* Se colocan los puntos del *kernel* resolviendo un problema de optimización donde cada punto aplica una fuerza repulsiva sobre los demás. Los puntos son restringidos a permanecer en una esfera con fuerza atractiva y uno de ellos es restringido a ocupar el centro. Finalmente, los puntos circundantes se reajustan a un radio medio de 1.5σ, asegurando un pequeño solapamiento entre cada área de influencia de los puntos del núcleo y una buena cobertura espacial.

Con los *kernel* inicializados adecuadamente, la versión rígida de KPConv es extremadamente eficiente, en particular cuando se escoge un K suficientemente grande para cubrir el dominio esférico de g. Sin embargo, puede incrementarse su capacidad aprendiendo las posiciones de los puntos. La función de kernel g es diferenciable con respecto a $\overline{x_k}$, lo que significa que tiene parámetros entrenables. Podría considerarse aprender un conjunto global de $\{\overline{x_k}\}$ para cada capa de convolución, pero esto no traería más potencia descriptiva que una disposición fija regular. En lugar de ello, la red genera un conjunto de K desplazamientos $\Delta(x)$ para cada localización de la convolución $x \in R^3$ y se define KPConv deformable como:

$$(F * g)(x) = \sum_{x_i \in N_x} g_{deform}\big(x - x_i, \Delta(x)\big) f_t$$

$$g_{deform}\big(y_i, \Delta(x)\big) = \sum_{k<K} h(y_i, \overline{x_k} + \Delta_k(x)) W_k$$

Se definen los desplazamientos $\Delta_k(x)$ como la salida de un mapeo de KPConv rígido de características de entrada D_{in} a valores 3K. Durante el entrenamiento, la red aprende el kernel rígido generando los desplazamientos y el kernel deformable generando las características de salida simultáneamente, pero el ratio de aprendizaje de la primera es 0.1 veces el ratio de aprendizaje global de la red.

Desafortunadamente, esta adaptación directa de la convolución deformable en imágenes no se ajusta a las nubes de puntos. En la práctica, los puntos del *kernel* terminan siendo alejados de los puntos de entrada. Estos puntos de *kernel* son perdidos por la red, porque los gradientes de sus desplazamientos son nulos cuando no hay vecinos en el rango de influencia. Para lidiar con este comportamiento, se propone una regularización de ajuste que penaliza la distancia entre un punto del *kernel* y su vecino más cercano entre los vecinos de entrada. Además, se añade un término de regularización repulsivo entre todos los pares de puntos de *kernel* cuando su área de influencia se solapa, de manera que no colapsen juntos. Con esta regularización, la red genera desplazamientos que se ajustan a la geometría local de la nube de puntos.

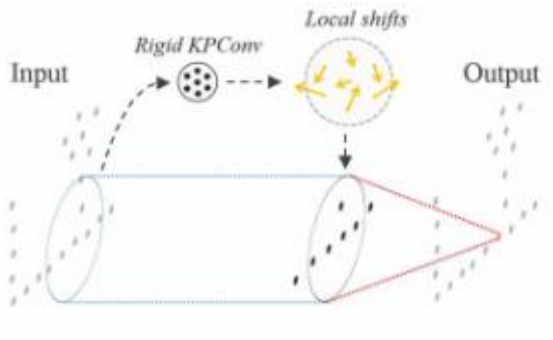

Figura 1.17: Operador deformable de KPConv.

Capas de *kernel* de puntos

- **Submuestreo para lidiar con densidades variables**. Se utiliza una estrategia de muestreo para controlar la densidad de puntos en cada

capa. Para asegurar la consistencia espacial de las localizaciones de los puntos muestreados, se apuesta por el sampling grid. Por lo tanto, los puntos soporte de cada capa, que transportan las localizaciones de las características, son escogidas como los baricentros de los puntos originales de entrada contenidos en celdas no vacías del grid.

- **Capa de *pooling***. Para crear arquitecturas con múltiples escalas de capas, se necesita reducir el número de puntos de forma progresiva. Como ya existe un submuestreo de grid, se dobla el tamaño de la celda en cada capa de pooling, al igual que otros parámetros relacionados, aumentando de forma incremental el campo receptivo de KPConv. Las características pooled en cada nueva localización pueden ser obtenidas mediante max pooling o KPConv indistintamente.
- **Capa KPConv**. Toma como entrada los puntos y sus características correspondientes, y la matriz de índices de vecindario. La matriz de vecindario se fuerza a tener el tamaño del vecindario más grande.
- **Parámetros de la red**. Cada capa *j* tiene un tamaño de celda dl_j de los que se infieren los otros parámetros.

Arquitecturas neuronales para KPConv

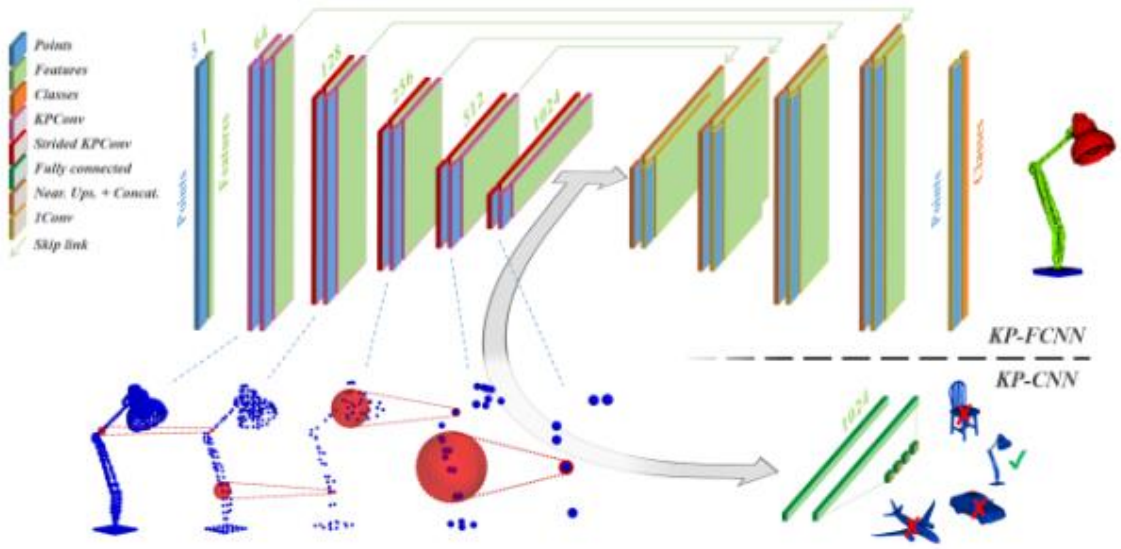

Figura 1.18: Arquitecturas neuronales propuestas para KPConv (segmentación y clasificación).

- **KP-CNN es red convolucional de clasificación de 5 capas**. Cada capa contiene 2 bloques convolucionales, el primero es strided excepto para la primera capa. Los bloques convolucionales son diseñados al igual que ResNets con KPConv reemplazando a la convolución en imágenes, normalización de batch y leaky ReLU. Tras la última capa, se agregan las características mediante average pooling global y se procesan con una red fully-connected y capas softmax al igual que ocurre en la clasificación de imágenes.
- **KP-FCNN es una red *fully convolutional* para segmentación.** La parte de codificador es la misma que el modelo KP-CNN, y la parte decodificadora utiliza upsampling del más cercano para obtener las características finales a nivel de puntos. Se utilizan skip connections para pasar información entre el encoder y decoder.

1.3.2.5 3D Recurrent Neural Networks with Context Fusion for Point Cloud Semantic Segmentation

En este trabajo se investiga el pooling piramidal a nivel de punto para capturar estructuras locales a diferentes densidades tomando vecindarios a un nivel multiescala. Luego, se aplica una red neuronal recurrente en dos direcciones y a nivel jerárquico para explorar las dependencias a largo rango. (Ye, Li, Huang, Du, & Zhang, 2018)

INTRODUCCIÓN

Muchos enfoques han optado por regularizar la nube de puntos en forma de vóxeles, por ejemplo. Debido a la dispersión de las nubes de puntos, la voxelización es ineficiente y además provoca la pérdida de detalle para evitar altos costes de computación. Para hacer uso de los *framework* 2D, también se han utilizado imágenes multivista sobre los modelos 3D. Sin embargo, la proyección inversa de las imágenes segmentadas hacia una la nube de puntos no es un problema trivial.

La arquitectura de *PointNet* presenta dos limitaciones que restringen el rendimiento en grandes nubes de puntos. En primer lugar, las características a nivel de punto se integran con la característica global, fallando al capturar las estructuras locales representadas por puntos vecinos. Por otra parte, una nube de puntos se divide en pequeños bloques volumétricos y cada bloque se predice de forma independiente sin ningún tipo de conexión.

REDES NEURONALES RECURRENTES DE FUSIÓN DE CONTEXTO

En primer lugar, se propone un *pooling* piramidal a nivel de punto para aprender el contexto vecindario multiescala. Por otra parte, el contexto a largo rango es estudiado utilizando un modelo *Recurrent Neural Network* en las dos direcciones de x e y sobre el plano horizontal, lo que permite a la red aprender el contexto espacial en nubes de puntos a gran escala.

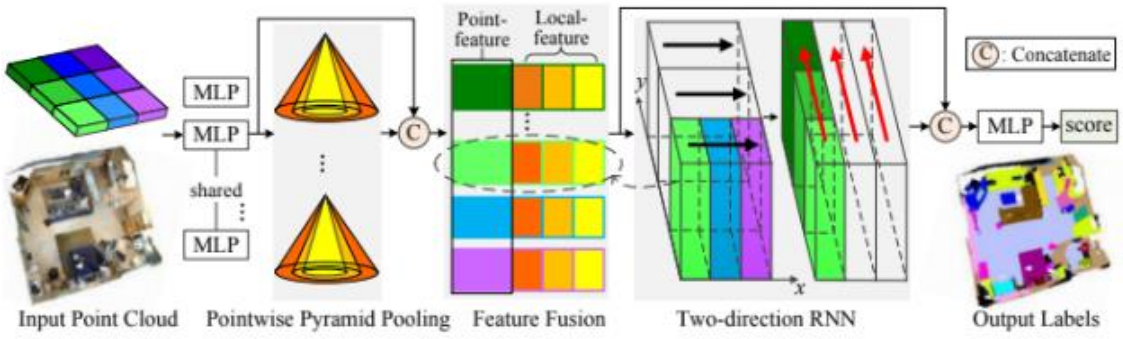

Figura 1.19: Funcionamiento de 3D recurrent neural networks with context fusion.

Pooling piramidal a nivel de puntos

Debido al *max pooling* utilizado en *PointNet* (Charles R Qi et al., 2016), se pierden muchos detalles y se causa ambigüedad. A diferencia del *pooling* en 2D que utiliza varios *strides*, se adopta un *pooling* piramidal a nivel de punto con varios tamaños de ventana de *pooling*. Esto es porque el módulo de *pooling* con un *stride* más grande que 1 podría traer una pérdida de resolución y obstaculiza la precisión en predicción densa.

Dado un conjunto de puntos no ordenados, se divide el espacio 3D completo en bloques de 1.5m x 1.5m a lo largo del plano tierra. Cada bloque es extendido para cubrir toda la altura de la sala. El *pooling* piramidal se hace en vecindarios con diferentes números de puntos. En lugar de buscar los k vecinos más cercanos para cada punto, se aplica una forma aproximada pero más eficiente de aprovechar las múltiples escalas de los cuboides. En cada escala, se aplica un módulo de *max pooling one-stride*. Por ejemplo, si el tamaño de ventana es N, se seleccionan aleatoriamente N puntos en el cuboide correspondiente para *max pooling*.

Las características *coarse-to-fine* reducidas mediante *pooling* son integradas mediante una única capa de convolución para la fase RNN siguiente.

Redes recurrentes para embebido del contexto

El espacio 3D completo se divide en bloques uniformemente espaciados en las direcciones *x* e *y* del plano tierra, L_x y L_y respectivamente. El espacio a lo largo del eje ascendente derecho se mantiene sin dividir debido a su gran dispersión y coherencia en la dirección vertical.

Las características de punto originales y las características extraídas con *pyramid pooling* se concatenan para servir de entrada al modelo RNN. El *pipeline* conlleva dos fases: 1) solo se considera la conexión espacial en la dirección del eje x, acoplando bloques Lx con el mismo índice y que el resto. La operación de cada grupo recurrente es independiente y puede realizarse en paralelo. Las características derivadas de las características de puntos concatenadas con características de *pooling* en cada bloque son desenrolladas para formar una secuencia en las celdas RNN correspondientes. En cada instante de tiempo (correspondiente a un pequeño bloque en el mismo índice y), cada RNN toma las características de bloque concatenadas como entrada y actualiza el estado a partir del estado previo. Este procedimiento se repite en el eje y.

Una vez las dos direcciones se hayan procesado, se obtienen características originadas a partir de integrar conocimiento sobre el contexto local y a largo rango. Además, podrían añadirse más capas recurrentes para procesar direcciones adicionales. Las características de salida del modelo RNN son concatenadas al final con las características de entrada, incluyendo las características a nivel de punto y características locales de *pooling* para predecir la etiqueta de cada punto.

1.3.2.6 Tree Classification in Complex Forest Point Clouds Based on Deep Learning

El artículo trata sobre la clasificación de árboles utilizando información capturada mediante LiDAR. Para ello, primero se detectan los árboles basándose en la densidad de puntos en el tronco. Después, tras un proceso de eliminación de ruido y filtrado de información, se clasifica cada árbol. (Zou, Cheng, Wang, Xia, & Li, 2017)

IDEAS CLAVE

Una vez se distingue cada instancia de árbol de forma individual (no se usa Deep Learning aquí, solo preprocesamiento clásico) se clasifica cada árbol utilizando una *Deep Belief Network* puesto que necesitan menos datos de entrenamiento que los modelos habituales para converger al óptimo.

Comparado con otras redes profundas, DBN tiene un paso especial en el entrenamiento: el pre-entrenamiento de la **máquina de Boltzmann restringida (RBM)** *greedy* por capas. La inicialización de parámetros no es aleatoria, sino que se ajustan los parámetros de una forma en que la convergencia es relativamente sencilla.

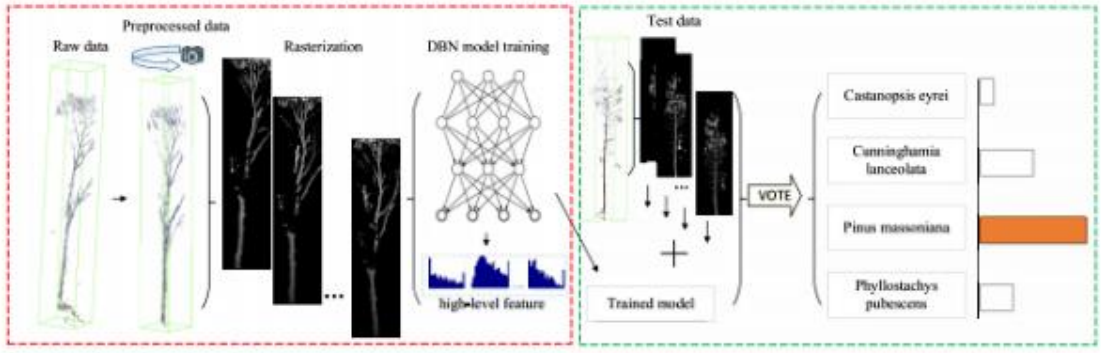

Figura 1.20: Pipeline de Tree Classification in Complex Forest Point Clouds.

DEEP BELIEF NETWORK

Un modelo DBN (*Deep Belief Network*) incluye 3 capas RBM (*Restricted Boltzmann Machine*) y 3 *fully connected layers*. Como función de activación, se utiliza la función logística. El proceso de solución para el entrenamiento del modelo se divide en 2 etapas: solución de RBM y solución de *fully connected layers*:

1. **Primera fase**. Las muestras de árboles de entrenamiento se proyectan en imágenes multivista como entrada, y se utiliza un método *greedy layer-wise pretraining* para inicializar los parámetros W_i del RBM con las imágenes de entrada. Luego, se utiliza una combinación del muestreo de Gibbs y el método de constraste de divergencia para ajustar los valores de las unidades.
2. **Segunda fase**. Se utilizan los parámetros del RBM para inicializar las capas *fully connected*. Esta fase permite la propagación hacia adelante

y el ajuste de los parámetros de la red utilizando el gradiente descendente estocástico.

Para clasificar las especies de árboles, se aplica una función de *softmax*.

FASES DE ENTRENAMIENTO Y TEST

Durante el entrenamiento, cada árbol extraído se proyecta sobre imágenes multivista (con una rotación sucesiva de 10º) y se entrena el clasificador. En fase de test, se clasifican todas las imágenes asociadas a un mismo árbol y luego se establece un sistema de votación que escoge por mayoría la clase asociada a un árbol en cuestión.

1.3.2.7 RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds

La clave de este enfoque es el uso de Random Point Sampling (muestreo aleatorio) en lugar de otros métodos de selección mucho más complejos. El muestreo aleatorio puede descartar características clave debido a la aleatoriedad. Para solventar esto, se introduce un nuevo módulo de agregación de características locales para incrementar progresivamente el campo receptivo de cada punto 3D, preservando en tanto los detalles geométricos. RandLA-Net puede procesar 1 millón de puntos en una sola propagación, resultado 200 veces más rápido que la mayoría de métodos existentes. (Hu et al., 2020)

INTRODUCCIÓN

La segmentación semántica eficiente de nubes de puntos 3D a gran escala es una capacidad fundamental y esencial para los sistemas inteligentes de tiempo real. Los métodos existentes para los datos bidimensionales no son directamente aplicables sobre la geometría tridimensional.

PointNet emergió como un enfoque prometedor para procesar directamente nubes de puntos 3D. Aprende características a nivel de punto utilizando perceptrones multicapa compartidos, y es computacionalmente eficiente, aunque no es capaz de aprender características locales en vecindarios de la nube sino globales, utilizando una operación *max-pooling* para la agregación de características. Para el aprendizaje de estructuras locales, han surgido una gran variedad de métodos categorizables en: 1) *pooling* de características de vecinos, 2) paso de mensajes en grafos, 3) convolución basada en *kernels* y 4) agregación (*pooling*) basada en *attention*.

Aunque estos trabajos son prometedores, están pensados para trabajar con nubes de puntos a pequeña escala, de algunos miles de puntos. Cuando quieren aplicarse sobre nubes de puntos a gran escala, deben aplicar técnicas de muestreo y es aquí donde surge la principal deficiencia de estos métodos, pues en su mayoría utilizan métodos de muestreo muy ineficientes.

Algunos trabajos han comenzado a procesar nubes de puntos a gran escala de forma directa, como *Superpoint Graph*, que construye una partición geométricamente homogénea procesada en una fase posterior por una RNN (*Recurrent Neural Network*), *FCPN* o *PCT*, que combinan voxelización y redes a nivel de punto para procesar nubes de puntos de forma masiva. En cambio, las operaciones que realizan son computacionalmente muy costosas (voxelización y particionamiento).

El reto consiste en diseñar una arquitectura que sea eficiente computacionalmente y en memoria. Por tanto, el sistema desarrollado ha de ejecutarse en el mínimo tiempo posible, y haciendo uso de la mínima cantidad de memoria principal disponible, lo cual requiere a su vez: 1) una estrategia de muestreo que sea eficiente computacionalmente y en memoria de manera que sea adecuada a las capacidades de las GPUs modernas, y 2) un módulo para aprender características locales. Se propone un módulo de agregación de características locales para capturar estructuras locales complejas sobre conjuntos de puntos cada vez más pequeños.

Por tanto, construido sobre los principios de *random sampling* (muestreo aleatorio) y un módulo para agregar características locales, RandLA-Net consigue ser 200 veces más rápido que la mayoría de enfoques modernos y supera el estado del arte en la mayoría de *benchmarks* para segmentación semántica de nubes de puntos.

Figura 1.21: RandLA-Net. Módulo básico para agregar características y random sampling.

RED NEURONAL RANDLA-NET

Dada una nube de puntos a gran escala con millones de puntos esparcidos en cientos de metros, procesar dichos puntos con una red neuronal profunda requiere muestrear progresiva y eficientemente en cada capa de la red.

Muestreo eficiente

Los enfoques de muestreo existentes pueden clasificarse en muestreo **heurístico** y muestreo basado en **aprendizaje**.

1. **Heurístico.**
   a. ***Furthest Point Sampling (FPS).*** Devuelve $k$puntos de un conjunto inicial de N puntos, donde cada punto de k es el punto más lejano de los $k-1$ puntos anteriores. La complejidad es $O(N^2)$, donde N es el número total de puntos de la nube.
   b. ***Inverse Density Importance Sampling (IDIS).*** Reordena N puntos de acuerdo a la densidad de cada punto, y luego selecciona los k primeros puntos. La complejidad es $O(N)$, pero es sensible a *outliers* y además no es adecuado para un sistema de tiempo real.
   c. ***Random Sampling (RS).*** Selecciona uniformemente y de forma aleatoria k puntos de un conjunto de N puntos (la nube completa). La complejidad es $O(1)$.
2. **Basado en aprendizaje**.
   a. ***Generator-based Sampling (GS).*** Aprende a generar un conjunto pequeño de puntos para representar de forma aproximada el conjunto de puntos original. En cambio, suele usarse FPS para asociar el subconjunto generado con el conjunto original en la fase de inferencia, conllevando un cómputo adicional.
   b. ***Continuous Relaxation based Sampling (CRS).*** Cada punto muestreado es aprendido basándose en una suma ponderada sobre la nube de puntos completa. Resulta en una gran matriz de pesos al muestrear todos los nuevos puntos, llevando a una multiplicación de matrices gigantescas y resultando en un problema de un coste altísimo en memoria.
   c. ***Policy Gradient based Sampling (PGS).*** Se formula la operación de muestreo como un proceso de decisión de Markov. Secuencialmente, se aprende una probabilidad de muestrear cada punto. Sin embargo, la probabilidad aprendida tiene una alta

varianza debido a la exploración del espacio extremadamente alta cuando la nube de puntos es a gran escala.

Por tanto, tras analizar los diferentes métodos de muestreo disponibles, se concluye con que el muestreo aleatorio es el enfoque más adecuado para utilizarse sobre nubes de puntos a gran escala comparado con el resto de alternativas. Sin embargo, el muestreo aleatorio podría resultar en la pérdida de algunas características relevantes de puntos. Para solventar esto, se propone el siguiente módulo.

Agregación de características locales

El módulo de agregación de características locales se aplica en paralelo a cada punto 3D y consiste en 3 unidades neuronales: 1) *Local Spatial Encoding (LocSE)*, 2) *Attentive Pooling*, y 3) *Dilated Residual Network.*

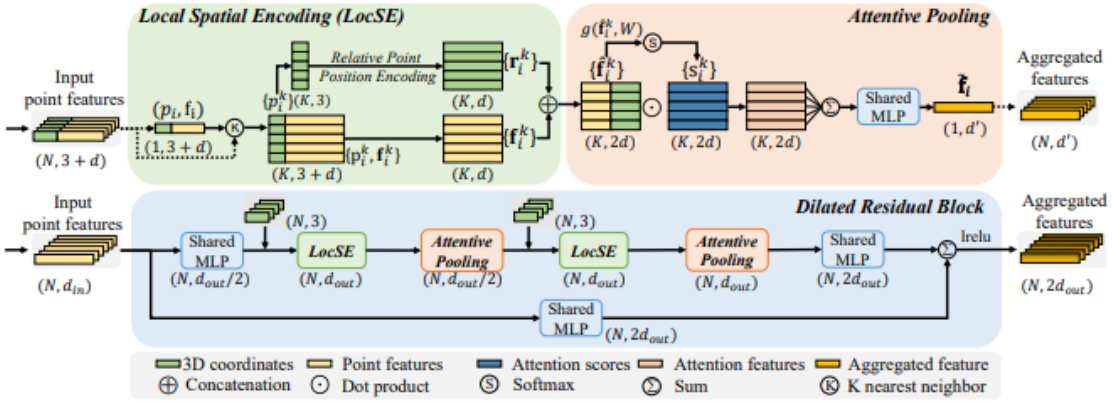

Figura 1.22: Módulo de agregación de características locales.

1. **Codificación de la localidad espacial.** Dada una nube de puntos P con sus características en cada punto, esta unidad incrusta las coordenadas (x, y, z) de todos los puntos vecinos, de manera que el punto correspondiente obtiene conocimiento sobre las posiciones relativas de los puntos. Esta unidad incluye los siguientes pasos:
    a. ***Encontrar los puntos vecinos.*** Para cada punto, se extraen sus K vecinos más cercanos utilizando *kNN*.
    b. ***Incrustación de la posición relativa de puntos.*** Para cada uno de los *K* puntos más cercanos del punto central p_i, se codifica la posición relativa del punto usando:

- $r_i^k = MLP(p_i \oplus p_i^k \oplus (p_i - p_i^k) \oplus \|p_i - p_i^k\|)$
- Donde $\oplus$ representa al operador de concatenación.

c. ***Aumento de características de puntos.*** El vector de características generado para cada punto, r_i^k, se concatena con las características de cada punto f_i^k, resultando en un nuevo vector de características para cada vecino: $\hat{f}_i^k$, para todos los puntos vecinos. Se utilizan las posiciones relativas de los vecinos para aumentar la calidad de las características del punto central.

2. ***Pooling* adaptativo (*Attentive Pooling*).** Esta unidad se utiliza para agregar las características extraídas en la fase anterior. Habitualmente se usan enfoques como *max* o *average pooling*, pero en cambio esto podría resultar en una pérdida de información. Por tanto, esta unidad surge como un potente módulo de atención que aprende a ponderar características locales importantes. Consiste en los siguientes pasos:

   a. ***Calcular pesos de ponderación (attention scores).*** Se diseña una función compartida $g()$ para aprender un único *attention score* para cada característica. La función g consiste en un perceptrón multicapa seguido por una función *softmax*:

   $$s_i^k = g(\hat{f}_i^k, W)$$

   $$donde\ W\ representa\ los\ pesos\ del\ perceptrón\ multicapa.$$

   b. ***Suma ponderada.*** Las ponderaciones emitidas en la fase anterior sirven para agregar las características de los vecinos para el punto central (del que se extraen los vecinos) de la siguiente forma:

   - $\tilde{f}_i = \sum_{k=1}^{K} (\hat{f}_i^k \cdot s_i^k)$
   - De esta forma, estas dos unidades descritas tratan de agregar características geométricas y patrones de los K vecinos más cercanos, y finalmente generar un vector de máxima información con respecto a las características $\tilde{f}_i$ que describe el entorno local de cada punto.

3. **Bloque residual dilatado (*Dilated Residual Block).*** Puesto que las nubes de puntos van a ir reduciéndose (debido al muestreo) es necesario ir incrementando progresivamente el campo receptivo para cada punto, de manera que los detalles sobre la geometría se vayan preservando.

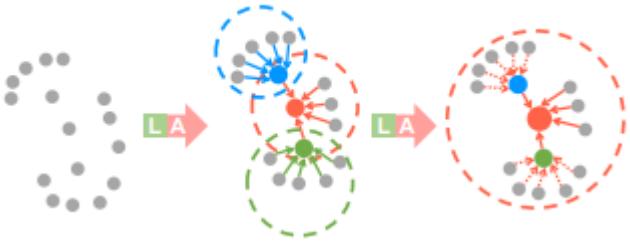

Figura 1.23: Dilated Residual Block.

- Como se aprecia, tras la primera aplicación de *LocSE (Local Spatial Encoding)* el punto central obtiene información de los 5 vecinos más cercanos, mientras que en la segunda también obtiene información de los vecinos de sus vecinos.
- El módulo de agregación local está diseñado para preservar estructuras locales complejas mediante la consideración de la geometría de vecindario e ir aumentando el campo receptivo de forma progresiva. Además, se construye usando perceptrones multicapa por lo que es computacionalmente eficiente.

ARQUITECTURA DE LA RED

A continuación, se detalla la arquitectura de la red propuesta para segmentación semántica de nubes de puntos a gran escala. La arquitectura sigue el frecuente modelo propuesto por U-Net (Ronneberger, Fischer, & Brox, 2015) para segmentación de imágenes, que consiste en una arquitectura en forma de reloj de arena con *skip-connections*.

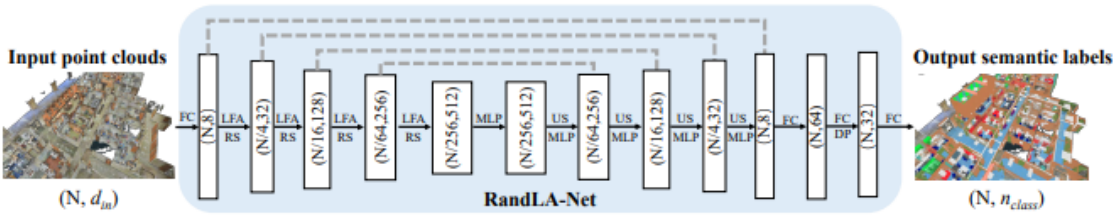

Figura 1.24: Arquitectura de RandLA-Net.

Las diferentes partes de la red son las que siguen:

- **Entrada.** Nube de puntos a gran escala con N puntos de d_{in}dimensiones, por tanto, una matriz $N \times d_{in}$.
- **Capas de codificación.** Se usan 4 capas de codificación para ir reduciendo progresivamente el tamaño de las nubes de puntos y aumentando a la vez el número de características en cada punto. Cada capa reduce $\frac{1}{4}$ el número de puntos y aumenta al doble el número de características para cada punto.
- **Capas de decodificación.** Se usan 4 capas de decodificación tras las capas de codificación. Para cada capa, se usa el algoritmo *kNN (k Nearest Neighbors)* para buscar un vecino más cercano para cada punto de **consulta**, el conjunto de características de puntos es remuestreado a través de una interpolación del vecino más cercano. Los mapas de características remuestreados se concatenan con los mapas extraídos en la fase de codificación mediante *skip connections*, y después se usa un MLP sobre los mapas de características.
- **Predicción.** Se usan 3 FC *layers* una capa *dropout* para predecir la clase de cada punto.
- **Salida.** La salida de la red es el número de puntos de entrada junto a la probabilidad asociada a cada clase: $N \times n_{class}$.

1.3.2.8 RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR Point Cloud

Propuesta de la típica arquitectura de segmentación U-Net para la segmentación de nubes de puntos LiDAR. La nube de puntos es proyectada en un mapa de profundidad 2D teniendo en cuenta el modo de funcionamiento del sensor. Luego, esta imagen se sirve como entrada a una red U-Net. El

modelo está entrenado en KITTI 3D. Se demuestra que, aunque RIU-Net es muy simple, se ofrecen resultados comparables a los métodos que constituyen el estado de la cuestión en segmentación de nubes de puntos a partir de mapas de profundidad. Se consigue una tasa de 90 FPS usando una única GPU, lo que permite su implementación sobre sistemas de tiempo real. (Biasutti, Bugeau, Aujol, & Brédif, 2019)

Las principales contribuciones de este trabajo son una arquitectura de red para la segmentación semántica de nubes de puntos generadas mediante LiDAR terrestre, concretamente de LiDAR a bordo de vehículos. Por tanto, está ideado para la industria de vehículo autónomo, en tanto que el sistema permite su integración en sistema de tiempo real. A continuación, se detalla la metodología seguida en este trabajo.

METODOLOGÍA

Entrada de la red

Puesto que los escáneres LiDAR que suelen ir a bordo de vehículos autónomos suelen generar nubes de puntos con un patrón de muestreo predefinido de líneas de escaneo y ángulos de escaneo casi uniformes. En tanto, cada punto está definido por dos ángulos y una profundidad: respecto de la vertical, respecto de la horizontal y distancia al escáner. Por tanto, esto permite proyectar la nube de puntos escaneada usando la proyección esférica y obtener un mapa de profundidad panorámico para toda la escena capturada por el sensor.

En tanto, y en perfectas condiciones de adquisición, la salida de esta fase es una imagen de rango de dimensiones 512 * 64 px con 2 canales: profundidad y elevación. Cada punto se mapea al píxel correspondiente en función de sus ángulos con respecto a la horizontal y la vertical.

Arquitectura de la red

La arquitectura de U-Net es un *encoder-decoder*, en forma de reloj de arena y haciendo uso de *skip-connections* para evitar el cuello de botella que se forma para la información que se propaga a través de la red. La primera mitad trata sobre la aplicación sucesiva de convoluciones 3×3 seguidas por una unidad ReLU, que reduce en cada aplicación el tamaño de la entrada en un factor de 2. La segunda mitad de la red aplica convoluciones inversas de forma sucesiva para obtener un mapa de segmentación con tamaño igual al inicial.

La última capa consiste en convolución 1×1 que da para cada píxel un vector K dimensional, en tanto que K es igual al número de clases del problema.

Función de coste

Se define como una entropía cruzada del *softmax* que resulta de la salida de la red. El *softmax* es definido a nivel de píxel para cada etiqueta *k*.

Entrenamiento

Se usa el algoritmo Adam como optimizador y se fija el ratio de aprendizaje a 0.001. Se usa normalización de *batch* con momento de 0.99 para asegurar una buena convergencia del modelo. El tamaño de *batch* se fija a 8, y el entrenamiento se detiene pasados 10 *epochs*.

1.3.2.9 Dynamic Graph CNN for Learning on Point Clouds

Las nubes de puntos sufren de pérdida de topología, por lo que diseñar un modelo que sea capaz de recuperar la topología subyacente podría incrementar la potencia representativa de las nubes de puntos. Se propone un nuevo operador de convolución EdgeConv que se adapta perfectamente a las tareas de segmentación y clasificación. EdgeConv se ejecuta sobre grafos calculados de forma dinámica en cada capa de la red. Tiene algunas propiedades: incorpora información del vecindario, puede apilarse para aprender características globales sobre la forma y, en sistemas multicapa, puede capturarse la afinidad entre características semánticas. En definitiva, este trabajo se asienta sobre la idea de que la distancia entre puntos es diferente cuando se pasa de un espacio euclídeo a un espacio de características.

PRINCIPALES CONTRIBUCIONES

Se propone una operación simple, *EdgeConv*, que captura las características geométricas locales mientras que mantiene la invariabilidad a permutaciones. En lugar de generar características de puntos directamente mediante incrustación, *EdgeConv* genera características de arista que describen la interacción de un punto con respecto a sus vecinos. *EdgeConv* está diseñado para ser invariante al orden de los vecinos, por ello es invariante a permutaciones. Debido a que *EdgeConv* construye de forma explícita un grafo local y aprende las incrustaciones para las aristas, el modelo es capaz de agrupar puntos tanto en el espacio de características como en el espacio semántico (Wang et al., 2018).

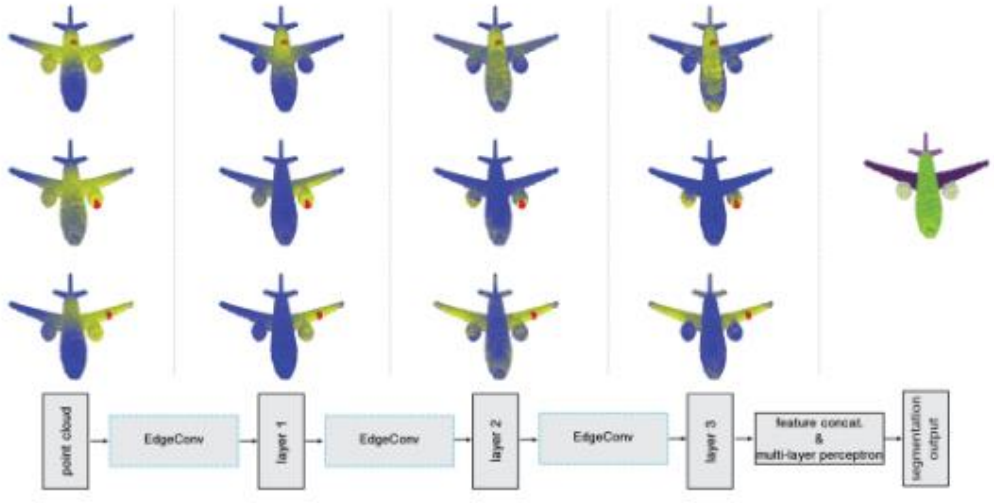

Figura 1.25: Pipeline de procesamiento de DGCNN.

En los experimentos, se integra el operador *EdgeConv* en la versión básica de *PointNet* sin utilizar alguna transformación de características. El modelo supera el estado del arte en *datasets* como *ModelNet40* y *S3DIS*. Contribuciones clave:

- Se presenta una nueva operación para aprendizaje sobre nubes de puntos, *EdgeConv*, para capturar características geométricas locales mientras se mantiene la invariabilidad a permutaciones de los miembros.
- El modelo puede agrupar puntos semánticamente mediante la actualización de un grafo de relaciones de capa a capa.

EdgeConv puede integrarse en numerosos enfoques existentes para segmentación semántica de nubes de puntos, pues su principal contribución es la construcción dinámica del grafo, que se actualiza varias veces en la propagación hacia delante de la red.

METODOLOGÍA

En lugar de trabajar sobre puntos de forma individual, tal y como lo hace *PointNet* (Charles R Qi et al., 2016), se tienen en cuenta las estructuras geométricas locales mediante la construcción de un grafo de vecindario y la aplicación de operaciones de convolución sobre los vértices que conectan pares de puntos, tal y como lo hacen las redes neuronales sobre grafos (Bronstein et al., 2017). La operación *EdgeConv* (Wang et al., 2018) muestra invariabilidad a transformaciones geométricas como la traslación.

A diferencia de las redes neuronales convolucionales para grafos (Bronstein et al., 2017), el grafo en este caso no está fijado, y se actualiza tras cada capa de la red. Esto es, el conjunto de puntos más cercanos (*k Nearest Neighbors*) de un punto cambia de una capa a otra, pues la distancia de un punto a otro en el espacio de características es diferente a la distancia entre pares de puntos en el espacio métrico de entrada. Esta última afirmación es la que da paso al resto del estudio, y sobre la que se construye este Trabajo Fin de Máster en cuestión.

Edge Convolution

Sea una nube de puntos F-dimensional con **N** puntos, $X = \{x_1, \dots, x_n\} \subseteq \mathbb{R}^F$. En el caso más sencillo, F=3, correspondientes a las coordenadas (x, y, z) de cada punto contenido en la nube. Si se dispone de tal información, podrían añadirse otros atributos de entrada como los vectores normales a cada punto, el color RGB, la intensidad, etcétera.

Se realiza el cálculo del grafo dirigido $G = (V, E)$, donde V representa el conjunto de vértices (puntos), $V = \{1, \dots, n\}$, y E, $E = V \times V$, el conjunto de aristas que se obtienen a partir del grafo de los vecinos más cercanos (*kNN*). Se definen las características de borde como $e_{ij} = h_\Theta(x_i, x_j)$, donde $h_\Theta(x_i, x_j) : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$ es una función no lineal con un conjunto de parámetros Θ entrenables.

Finalmente, se define *EdgeConv* como la aplicación de una función simétrica de agregación (sumatorio o máximo). Esta operación se conoce habitualmente como *pooling*. En este trabajo, se observará la influencia que tiene un tercer tipo de *pooling* conocido como *Attentive Pooling* que se trata de un enfoque supervisado en el que la ponderación de cada punto vecino es diferente en cada caso, y esta ponderación se infiere *on-the-fly* por el propio modelo de red neuronal.

La salida del bloque *EdgeConv* en el vértice *i* se calcula como:

$$x_i' = pool_{j:(i,j) \in E}(x_i, x_j)$$

En la Figura 1.26 se observa de forma gráfica el funcionamiento de *EdgeConv*. En definitiva, dada una nube de puntos *F*-dimensional con **N** puntos, se obtiene como salida una nube de puntos *F'*-dimensional con el mismo número de puntos.

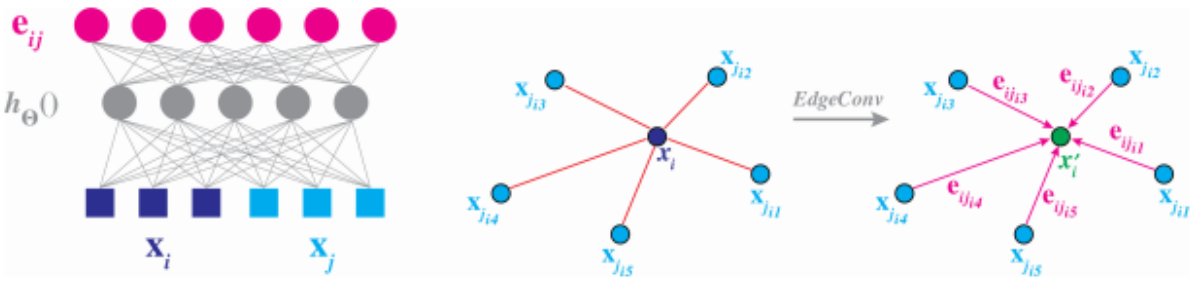

Figura 1.26: Funcionamiento de EdgeConv.

Es necesario realizar la selección de un operador no lineal $h_\Theta(x_i, x_j)$ que realice la codificación de la información de cada punto contenido en la nube. De lo que se trata es de enriquecer la información de la que dispone cada punto, de forma que no solo se tengan en cuenta las coordenadas de un punto y sus atributos a nivel individual, sino también las coordenadas y características de los vecinos más cercanos de un punto.

En el caso más simple, $h_\Theta(x_i, x_j) = \sum_{j:(i,j)\in E} \theta_m * x_j$. Este sería el enfoque de la convolución clásica, pues existen una serie de filtros entrenables que agregan la información del entorno. En un segundo caso, $h_\Theta(x_i, x_j) = h_\Theta(x_i)$. Este sería el enfoque seguido por *PointNet* (Charles R Qi et al., 2016), pues en este trabajo solo se tienen en cuenta las características de un punto a nivel individual, obteniéndose descriptores globales de objetos e ignorando características locales.

Finalmente, la opción seleccionada para este trabajo tiene en cuenta tanto la información local contenida en vecindarios de puntos como la información global que se obtiene mediante la agregación de todos los puntos en el entorno.

$$h_\Theta(x_i, x_j) = h_\Theta(x_i, x_j - x_i)$$

Esta opción combina la estructura global de forma mediante las coordenadas del punto con la estructura local obtenida a partir de $x_j - x_i$, que aporta información sobre cómo se distribuye el entorno del punto alrededor de él. Se define el operador como:

$$e'_{ijm} = ReLU(\theta_m * (x_j - x_i) + \phi_m * x_i)$$

El cuál puede implementarse utilizando un Perceptrón Multicapa (MLP, *Multilayer Perceptron*), siendo $\Theta = (\theta_1, \dots, \theta_M, \phi_1, \dots, \phi_M)$ el conjunto de parámetros entrenables de la red.

Actualización dinámica del grafo

Los experimentos sugieren que es beneficioso realizar la construcción del grafo usando los vecinos más cercanos en el espacio de características producido tras cada capa de la red. Esta es la principal distinción de este método con respecto al resto de redes neuronales convolucionales basadas en grafo.

Así, se dispone de diferentes grafos según la capa de la red estudiada. El primer grafo se construye con los vecinos más cercanos en el espacio euclídeo, esto es, los puntos más cercanos espacialmente. Sin embargo, a medida que se realiza la propagación hacia delante de un bloque de puntos en la red, se extraen diferentes características. De esta forma, y lo que se persigue con este trabajo, se intenta demostrar que dos puntos pueden estar físicamente distantes en el espacio métrico, y sin embargo permanecer muy próximos en el espacio de características.

Recordando el concepto de segmentación semántica, como tal, se trata de encontrar conjuntos de puntos que comparten características. Como se trata de segmentación semántica, la característica que comparten los puntos es la pertenencia a la misma categoría semántica. Por tanto, podremos concluir que si dos puntos están muy próximos en el espacio de características, entonces es muy probable que la geometría a nivel local de cada punto sea muy similar para ambos, y por tanto puede que pertenezcan a la misma categoría semántica.

Propiedades

Esta arquitectura es invariante a:

- **Permutaciones.** El orden en que la red se ve alimentada con los puntos contenidos en un bloque no afecta en el resultado de segmentación/clasificación.
- **Traslaciones.** La precisión de la red no se ve afectada por la localidad espacial de los puntos. Esto es, un segmento formado por puntos pertenecientes a la categoría ‘vegetación’, por ejemplo, es reconocido de la misma forma tanto si los puntos tienen coordenadas centradas en

el origen como si los puntos están desplazados con respecto al origen de coordenadas.

CONCLUSIONES A PROPÓSITO DE ESTE ESTUDIO

El trabajo (Wang et al., 2018) constituye un punto de partida para este Trabajo Fin de Máster. Se presenta una arquitectura neuronal que parte del procesamiento de conjuntos de puntos no ordenados, y que además tiene en cuenta la geometría local en cada entorno de puntos, algo que es crucial para conseguir buenos resultados de segmentación/clasificación.

En otros enfoques ya estudiados, como por ejemplo los enfoques basados en voxelización, se producía una gran pérdida de información que influía de forma notoria en los resultados de la segmentación. Además, los recursos necesarios tanto en tiempo como en memoria para la ejecución son excesivos. La pérdida de precisión inherente a cualquier proceso de discretización, como por ejemplo la voxelización, es algo inaceptable desde el momento en que se publican trabajos como *PointNet* o *Geometric Deep Learning* (Bronstein et al., 2017; Charles R Qi et al., 2016), puesto que ya se dispone de arquitecturas que procesan conjuntos de puntos de forma directa sin necesidad de realizar una discretización.

En este sentido, *Dynamic Graph CNN for Learning on Point Clouds* introduce la novedad de actualizar tras cada capa de la red el grafo que constituyen los puntos. Esto se constituye como genialidad, pues realmente la conexión entre puntos no es la misma a nivel semántico que a nivel métrico.

Sin embargo, este trabajo se limita a un estudio muy teórico sobre la aplicabilidad del aprendizaje profundo en el procesamiento de nubes de puntos tridimensionales, y no tiene en cuenta situaciones reales en las que el número de puntos es excesivamente grande, como los entornos exteriores. Los enfoques neuronales para segmentar nubes de puntos son especialmente útiles en exteriores, pues es donde se encuentran la mayoría de las aplicaciones para los escáneres LiDAR.

Por otra parte, analizando la arquitectura desde un punto de vista más técnico, realiza una agregación de la información de puntos siguiendo un enfoque no supervisado, como la media o la selección del máximo sobre una característica. Estas técnicas, aunque exitosas, no son las únicas de las que se dispone en la actualidad,

puesto que con el avance las técnicas más pioneras en *Deep Learning* han surgido otros tipos de agregación como el *Attentive Pooling*, el cual trata de realizar una reducción teniendo en cuenta las características de los datos de entrada en cada propagación. Esto es, se realiza una suma ponderada de las características de entrada teniendo en cuenta las peculiaridades de cada entorno local procesado.

Por tanto, a lo largo de este estudio se estudiará tanto la eficiencia de esta arquitectura en el procesamiento de nubes LiDAR captadas en exteriores, como la influencia de los diferentes canales de información, así como la influencia de nuevos métodos de agregación para la geometría a nivel local.

1.4 Requisitos iniciales

Uno de los objetivos principales de este proyecto consiste en estudiar la segmentación semántica de datos LiDAR utilizando enfoques de redes neuronales convolucionales. A diferencia de los modelos clásicos de red neuronal, las redes convolucionales construyen jerarquías de conceptos complejos a partir de los conceptos más simples. El propósito científico de este trabajo es el de servir como capítulo introductorio a una posterior Tesis Doctoral. En dicha tesis, se pretende estudiar la aplicabilidad del *Deep Learning* en el ámbito geométrico, no tanto en la clasificación/segmentación sino también en la generación de geometría a partir de modelos neuronales como las redes generativas adversarias o antagónicas (GAN, *Generative Adversarial Networks*) (Goodfellow et al., 2014), cuyo esquema básico se ilustra en Figura 1.27.

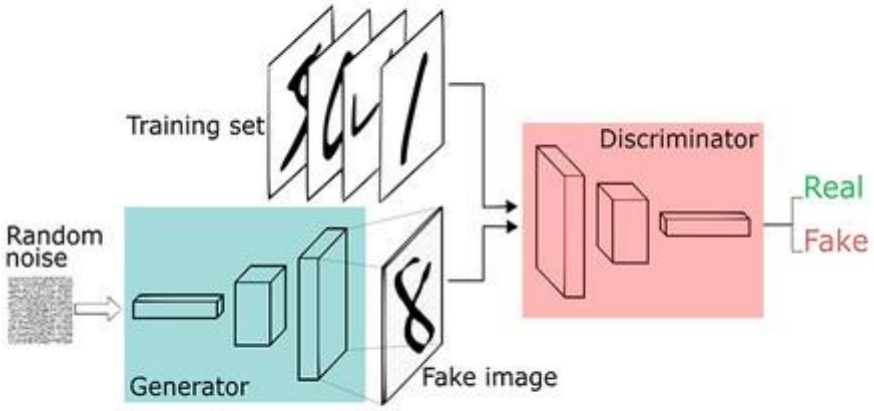

Figura 1.27: Modelo básico de red generativa adversaria. Fuente: Google.

Por lo tanto y centrándonos en este Trabajo Fin de Máster, su propósito es la elección de una técnica para segmentación/clasificación de nubes de puntos ampliamente reconocida en el ámbito científico. La técnica escogida es *Dynamic Graph CNN for Learning on Point Clouds* (Wang et al., 2018). El objetivo no es reutilizar el código publicado sino implementar la técnica desde cero, intentando extender la cobertura de la misma a los datos LiDAR, puesto que en la publicación original de esta técnica solo se consideran escaneados de interiores generados en su mayoría mediante fotogrametría.

Así, se pretenden incorporar al *pipeline* de segmentación algunas características innovadoras presentes en otras técnicas estudiadas durante la elaboración del estado del arte. De este modo, *ha* de desarrollarse un *software* que *debe* leer diferentes formatos de nubes de puntos, tanto formatos binarios *las/laz* propios de LiDAR como formatos en texto plano.

El sistema *debe* implementar la arquitectura neuronal citada previamente, (Wang et al., 2018), así como un mecanismo de generación de los datos de entrenamiento. Con vista a dejar un bagaje científico de interés para el desarrollo de la futura tesis doctoral, el sistema *debe* utilizar enfoques de validación cruzada que permitan realizar una evaluación no sesgada de la arquitectura propuesta, así como

del tratamiento de datos, mientras que a la vez *debería* ser capaz de almacenar redes entrenadas, de forma que puedan utilizarse en fases posteriores de inferencia o bien utilizarse para reentrenar utilizando otros datos como punto de partida.

El trabajo *debería* estudiar la influencia de la transferencia de conocimiento (Goodfellow et al., 2016) entre diferentes dominios de nubes de puntos, de forma que lo aprendido sobre un dominio capturado utilizando LiDAR terrestre pueda servir como punto de partida para entrenar con un conjunto de datos LiDAR aéreo mucho menos abundante en lo que a puntos se refiere.

Por otra parte, *sería deseable* realizar un proceso de inferencia sobre un conjunto de datos no utilizando en ninguna fase de entrenamiento, a modo de realizar una evaluación exhaustiva tanto de la calidad del modelo entrenado en su conjunto como de la arquitectura y técnicas de preprocesamiento de datos empleadas.

1.5 Alcance

Los datos LiDAR se posicionan como un estándar en el mundo de la ingeniería geomática y topográfica. Se trata de la principal herramienta de los topógrafos para realizar estudios sobre el terreno y el suelo, debido a la facilidad con la que se captan grandes volúmenes de estos. La facilidad de captación viene dada, como no podría ser de otra forma, de la mano de un alto coste económico y de difícil acceso.

Algo con lo que se han encontrado los científicos en los últimos años es una sensación de decepción cuando se trata de datos LiDAR. La velocidad a la que se producen los datos es significativamente alta, pero sin embargo no se dispone de tecnología especializada y renovada con la que realizar algunas tareas como por ejemplo la segmentación. Mientras que cada año salen al mercado numerosos escáneres nuevos que cada vez son capaces de captar más y más puntos, acompañados de más información, se siguen utilizando métodos arcaicos de clasificación/segmentación que tiran por tierra el esfuerzo realizado en el paso de captación pues realmente no son capaces de aprovecharse de tal cantidad de información.

Así, el presente estudio tiene el propósito de aportar una nueva perspectiva, y encontrar un método novedoso en lo que a procesamiento se refiere que sea capaz de aprovechar este superávit de información geométrica, así como que su precisión

sea directamente proporcional al volumen de información disponible, permitiendo la escalabilidad del sistema. Tanto tangibles como intangibles, los contenidos entregables de este trabajo son:

- **Introducción a la segmentación.** Cómo el problema de segmentación semántica de datos LiDAR se ha convertido en uno de los problemas más relevantes, y aún sin solución, en el campo de la visión artificial constituido durante los últimos años.
- **Perspectiva sobre las técnicas clásicas para segmentación de nubes de puntos.** De qué forma se ha resuelto este problema hasta la actualidad, teniendo en cuenta las limitaciones tanto en *hardware* como en *software*.
- **Diferentes perspectivas sobre técnicas recientes para segmentación de nubes de puntos.** Se introducen algunos de los trabajos actuales más relevantes orientados a segmentación y clasificación de nubes de puntos.
- **Propuesta de arquitectura para segmentación de datos LiDAR.** Tomando como base una técnica de éxito notorio en segmentación de nubes de puntos, se trata de mejorar dicha técnica mediante el añadido de funcionalidades capturadas en otras técnicas, así como la adaptación de la misma para ser capaz de trabajar con escaneados densos de exteriores, en los que los problemas asociados a las nubes de puntos son aún más evidentes que en interiores, a la vez que se aprovechan los diferentes atributos LiDAR.
- **Estudio sobre la influencia de diferentes métodos de agregación de información.** Cuando se trata de métodos de aprendizaje profundo, existen diferentes alternativas para resumir características presentes en diferentes elementos de información. En este trabajo se abordarán tanto el *max pooling* como el *attentive pooling*, presentes en diversas técnicas para procesamiento neuronal de información geométrica.
- **Estudio de la influencia de los diversos canales de información presentes en los datos geométricos.** Además de las coordenadas (x, y, z) de cada punto, resulta interesante aprovechar los datos LiDAR

en su totalidad y añadir los atributos que acompañan a cada punto al *pipeline* de segmentación. Estos atributos pueden ser la intensidad, el color, etcétera.

- **Código fuente del *software* desarrollado en Python.** Tanto la arquitectura de red como el resto de módulos de procesamiento se incluirán en un único proyecto Python para entregarse junto a la memoria del proyecto.
- **Batería de experimentos.** Se plantearán diferentes experimentos en diferentes configuraciones, sirviendo estos para realizar una memoria de resultados que recoja la mejor configuración y más adecuada para esta arquitectura.
- **Material utilizado.** Descripción de los materiales necesarios tanto *hardware* como *software*, así como el conjunto de datos utilizado en la fase de entrenamiento.
- ***Datasets*.** Relación de *datasets* utilizados tanto para la fase de entrenamiento como para la posterior fase de inferencia. En particular, uno de estos *dataset* se encuentra público en la web (Hackel et al., 2017), mientras que el otro conjunto de datos se ha obtenido del portal de datos LiDAR públicos del Gobierno de Navarra, correspondiente a la ciudad de Pamplona, sobre el que un grupo de investigación asociado ha realizado un proceso de verificación para analizar la idoneidad de los datos.

1.6 Hipótesis y restricciones

El TFM se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

Por otra parte, el inicio de este Trabajo Fin de Máster coincide a su vez con el comienzo de la actividad laboral del alumno a jornada completa en una empresa de consultoría. De este modo, a la principal restricción de la limitación de la duración del

trabajo se contrapone la dificultad de encontrar tiempo y motivación en el día a día para la realización del mismo.

A diferencia del Trabajo Fin de Grado desarrollado y presentado hace unos años, la participación en un entorno laboral que requiere de la dedicación exclusiva de 8 horas diarias limita en gran medida el tiempo disponible para desarrollar el presente trabajo. Sin embargo, la combinación de una actitud tenaz, así como una pasión indudable por la investigación han acompañado desde el primer momento, contribuyendo a la finalización de este estudio de una forma óptima, y sentando la base de lo que en unos años se convertirá en una tesis doctoral.

1.7 Estudio de alternativas y viabilidad

Tal y como se ha podido observar en la sección 1.3, existe una abundante cantidad de literatura científica al respecto de la clasificación y segmentación de nubes de puntos tridimensionales. La variedad de opciones a elegir como punto de partida para iniciar un proyecto relacionado con la materia en cuestión es bastante amplia desde un punto de vista científico. Sin embargo, no sería justo tratar a todas las técnicas como iguales desde un punto de vista científico, puesto que es necesario tener en cuenta aspectos como la precisión alcanzada con cada una, la dificultad de implementación, en qué medida puede optimizarse, el propósito en cuestión, etcétera.

A continuación, se analiza la forma en que se ajusta cada una de las técnicas estudiadas para el problema de segmentación de datos LiDAR en particular. Es necesario tomar como premisa que ninguna de ellas realiza el estudio de los datos LiDAR en particular, sino de nubes de puntos en general:

- **Técnicas clásicas.** Dado que este Trabajo Fin de Máster se enmarca bajo el título ‘*Geometric Deep Learning* aplicado a datos LiDAR’, no tiene sentido considerar como objeto de estudio ninguna técnica que no use un enfoque basado en aprendizaje profundo como piedra angular.
- ***A voxel-based deep learning approach for point cloud semantic segmentation.*** El estudio de la segmentación semántica de nubes de puntos usando una discretización basada en voxelización fue ampliamente analizado durante la realización del Trabajo Fin de Grado presentado por el mismo alumno que escribe esta memoria. El proceso

de voxelización en sí limita desde el principio la precisión alcanzable por el sistema. Toda variación dentro de un vóxel se pierde, puesto que, para conseguir una mayor precisión, es necesario disminuir tanto como sea posible el tamaño de un vóxel. Dado que las nubes de puntos se constituyen como modelos 2.5D, esto es, sólo se tiene la superficie de los objetos, la mayoría de las celdas del vóxel estarán vacías.

En definitiva, se trata de encontrar un compromiso entre la precisión del vóxel y el rendimiento del sistema, puesto que a mayor precisión de vóxel se conseguirían mejores resultados, pero cada vez serían necesarios más recursos tanto en tiempo de ejecución como en memoria, y esto podría volverse inestable.

- ***KPConv: Flexible and Deformable Convolutions for Point Clouds***. Esta técnica en particular constituye un éxito sin precedentes del aprendizaje profundo sobre el campo del procesamiento neuronal de nubes de puntos. La idea tras este trabajo parte de utilizar un operador de convolución espacial, que considera los puntos como elementos estructurales. Se proponen dos versiones del *kernel*: una versión fija y una versión deformable. La diferencia entre ellas reside en la localidad espacial de los puntos del *kernel*, y en el posterior ajuste sobre los datos de entrada.

  Las razones para utilizar esta idea como punto de partida son más que obvias, y además está preparado para utilizarse sobre nubes de puntos densas y masivas. Sin embargo, la reciente publicación de este trabajo así como los resultados tan buenos conseguidos complican en gran medida la realización de nuevas aportaciones desde una perspectiva científica.

- ***3D Recurrent Neural Networks with Context Fusion for Point Cloud Semantic Segmentation***. Al igual que el trabajo anterior, se trata de una técnica muy novedosa que utiliza redes neuronales recurrentes para realizar la segmentación semántica. El problema que aquí aplica es que, de una forma similar a la que ocurre en *PointNet*, utiliza descriptores locales extraídos de forma global. Esto es, se divide la nube en

subregiones y se va extrayendo el descriptor de cada subregión, que se tiene en cuenta durante la siguiente propagación. Esto puede ser óptimo, pero no utiliza ningún enfoque de red neuronal convolucional que además aproveche la información a nivel local.

- ***Tree Classification in Complex Forest Point Cloud for Tree Detection***. Este trabajo trata de clasificar árboles utilizando *Deep Learning*. No se trata de una técnica digna de considerar puesto que no detalla nada sobre el paso de segmentación.
- ***RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds***. Al igual que KPConv, se trata de una técnica óptima que aplica todos los conceptos de *Geometric Deep Learning*, y a la vez consigue resultados muy prometedores. Se trata de un trabajo publicado en 2020, de forma que realizar aportaciones científicas al respecto resulta complejo puesto que ya han sido estudiados los enfoques de interés más destacables y referenciados.

  Sin embargo, se extrae de este trabajo la idea del *Attentive Pooling* como método de agregación de características que infiere un vector de ponderación para realizar la suma ponderada en cada paso de *pooling*.
- ***RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR Point Cloud.*** Un detalle a destacar sobre esta técnica es que es la única de todas las estudiadas que hace énfasis sobre el tipo de datos con los que trabaja: LiDAR. Este trabajo está orientado a la segmentación de datos LiDAR provenientes de *mobile-mapping*, esto es, LiDAR sobre vehículos. Realiza una proyección de los puntos sobre una imagen apaisada, y tras esto emplea el conocido enfoque de *U-Net* (Ronneberger et al., 2015) inicialmente propuesto por NVIDIA para segmentación de imagen médica. Realiza tanto proyección sobre el plano como proyección inversa para fijar a cada punto una categoría semántica. Dada la naturaleza de origen de los datos con los que se va a trabajar en nuestro caso, no resulta adecuado trabajar con técnicas basadas en proyección RGB-D puesto que resulta difícil encontrar una perspectiva de proyección.

- ***Dynamic Graph CNN for Learning on Point Clouds***. Enmarcado como trabajo pionero en el campo de la segmentación y clasificación de nubes de puntos, se trata de una técnica que marcará la línea de trabajo de este proyecto pues consta de diversos aspectos sobre los que se pueden realizar aportaciones de índole científico.

1.8 Descripción de la solución propuesta

Anteriormente, se han indicado diferentes estrategias de segmentación semántica con base sobre el aprendizaje profundo que podrían aplicarse sobre datos LiDAR. Se trata de técnicas novedosas que han conseguido resultados exitosos para el problema en cuestión.

Sin embargo, la mayoría de ellas no asume una relación subyacente entre los diferentes puntos que conforman la nube, y este hecho es irreal pues realmente los puntos están conectados dada su pertenencia a los diferentes objetos que componen el escaneo. Por mencionar algunas de ellas, *RandLA-net* (Hu et al., 2020) y *KPConv* (Thomas et al., 2019) se posicionan como los métodos más exitosos para la resolución inteligente de esta tarea. En cambio, tal y como se ha comentado antes, sufren de este fenómeno de la asunción de la inexistencia de relación entre los puntos, lo cual resulta en buenos resultados desde la perspectiva neuronal, pero parte de una premisa falsa.

Por el contrario, este Trabajo Fin de Máster se construye sobre una arquitectura que además de procesar puntos en su formato original, sin necesidad de transformar y/o discretizar, tiene en cuenta las relaciones existentes entre los puntos a nivel local. Se trata de partir de la idea de *Dynamic Graph Convolutional Neural Network for Learning on Point Clouds* (Wang et al., 2018). Esta arquitectura ha demostrado éxito más que notable en tareas de segmentación semántica realizadas sobre escaneados de interiores. Se trata de un enfoque inteligente y prometedor que no ha sido explorado sobre escaneados de exteriores, habitualmente más densos y complejos, existiendo especial particularidad sobre los datos LiDAR, otro de los pilares de este proyecto.

Así, la técnica implementada en este trabajo toma inspiración del trabajo mencionado inmediatamente antes, estableciendo algunas diferencias importantes

con el objetivo de evaluar nuevos enfoques existentes en la literatura (sobre *pooling*, por ejemplo), así como utilidad sobre datos LiDAR.

A continuación, se mencionan algunos de los aspectos más destacables de esta arquitectura, dejando todos los detalles tanto de la construcción como de la implementación para su descripción en la sección 2.1.

1.8.1 Entrada de datos

La red permite introducir directamente bloques de puntos en su formato original, sin necesidad de aplicar algún tipo de transformación como la voxelización o la proyección sobre una imagen RGB-D. Esto evita el sesgo de información característico de este tipo de estrategias.

Además, cada uno de los puntos contenidos en el bloque está descrito conforme a diferentes dimensiones, introduciéndose a la vez las coordenadas (x, y, z), así como otras propiedades del estilo de los canales r, g, b, la intensidad, el número de retornos etcétera. En definitiva, se trata de información que complementa a la geometría y describe propiedades como por ejemplo la textura de la superficie escaneada por los dispositivos de adquisición, algo crucial para distinguir objetos pues define en última instancia el aspecto visual de los mismos.

Dada una nube de puntos como entrada, se divide en parcelas de área fija, tomando cada una de estas parcelas como bloque de puntos, y utilizando estos bloques de forma agregada en lotes para realizar el entrenamiento de la red neuronal.

1.8.2 Bloque EdgeConv

Este bloque se construye como la piedra angular de este proyecto, encargado de extraer características de borde mediante el cálculo de los vecinos más cercanos y la construcción de un grafo dinámico de características.

Toma como entrada un bloque de puntos, con sus diferentes dimensiones, y emite como salida otro bloque de puntos en el que se ha realizado una extracción de características mediante capas de convolución.

1.8.3 Vector de características globales

Tras la aplicación sucesiva de bloques *EdgeConv*, se obtiene un bloque de características globales mediante la concatenación de las salidas de sendos bloques, y el uso de una capa extra de convolución adicional que obtiene como salida un vector de características globales para el bloque de puntos introducido. Este vector se concatena con cada uno de los vectores de características locales asociado a cada punto de entrada, y se prepara una matriz de puntos que se utilizará en las capas de clasificación.

1.8.4 Capas de clasificación

Una vez se ha completado la extracción de características, el último hito destacable en esta arquitectura es la aplicación de capas tipo perceptrón para clasificar cada punto y resolver así la tarea de segmentación semántica

1.8.5 Conclusiones

En la sección 2.1 se realizará un análisis más detallado de las peculiaridades de esta técnica en cuestión. En dicha sección, se definen los detalles sobre la implementación, y cómo se desarrolla el cálculo del grafo de características que servirá para extraer características locales en cada punto partiendo, en principio, del grafo de los k vecinos más cercanos.

Por otra parte, se exponen las diferentes opciones existentes en cuanto al uso de capas de reducción de características (*pooling*), siendo posible elegir entre capas de tipo *max pooling* o capas de tipo *attentive pooling*.

1.9 Material y métodos

Puesto que este Trabajo Fin de Máster se desarrolla en la modalidad de trabajo teórico-experimental, se necesitan materiales de índole científico que permitan realizar los entrenamientos pertinentes de la red neuronal y concluir al respecto de los resultados obtenidos. Por otra parte, es necesario hacer referencia a los recursos *hardware* utilizados, puesto que constituyen un aspecto fundamental para el entrenamiento de la arquitectura neuronal.

1.9.1 Hardware

Los algoritmos de inteligencia artificial basados en redes neuronales se caracterizan, entre otros aspectos, por el alto coste computacional requerido durante la ejecución de los procesos de entrenamiento y validación. Una vez que un modelo basado en aprendizaje profundo ha sido entrenado, podría utilizarse una máquina con unas prestaciones más asequibles, siempre y cuando la máquina disponga de una **GPU** (*Graphics Processing Unit*) adecuada. En definitiva, una red se implementa como un conjunto de matrices (tensores) con pesos entrenables, que se van multiplicando por los datos recibidos como entrada, variando la dimensionalidad de los datos hasta obtener la salida deseada.

Es por ello que se utilizan este tipo de procesadores para entrenar, pues a diferencia de una **CPU** (*Central Processing Unit*) de propósito general, que implementa una arquitectura **SISD *(Single Instruction Single Data*)** según la taxonomía de Flynn, los procesadores gráficos o GPUs implementan la arquitectura **SIMD *(Singe Instruction Multiple Data*)** según la misma taxonomía. En tanto, el procesador aplica un único tipo de operación sobre múltiples datos a la vez, conociéndose estos procesadores como **vectoriales**. Por este motivo, estos procesadores son idóneos para la multiplicación de matrices ya que pueden calcular la matriz resultante en un solo paso de ejecución, a diferencia de una CPU que tendría que calcular cada posición de la matriz de salida de forma aislada e iterativa.

Los *frameworks* que implementan funcionalidad para desarrollar soluciones basadas en aprendizaje profundo, como PyTorch o TensorFlow, están dotados de la capacidad de poder ejecutar código en el procesador gráfico. Para ello, se integran con una tecnología propietaria de NVIDIA, CUDA *(Compute Unified Device Architecture)*, haciendo llamadas a la propia API expuesta por CUDA en la máquina en cuestión y permitiendo así realizar el entrenamiento y validación en este procesador.

Por otra parte, en cuanto a recursos de almacenamiento y procesamiento de propósito general se necesita de una máquina con unos recursos hardware de alto rendimiento puesto que van a manejarse grandes volúmenes de datos y esto ha de realizarse de forma eficiente. Así, se utilizarán principalmente dos entornos en este Trabajo Fin de Máster:

- **Equipo de desarrollo**. Se trata de mi equipo personal de casa, donde se realiza el desarrollo de todo el *software* necesario, así como todas las pruebas antes de pasar el código a un servidor de computación que realice los entrenamientos de la red. El equipo consta de un procesador Intel 8700K, 16 Gigabytes de memoria principal, 750 Gigabytes de almacenamiento SSD y 1,5 Terabytes de almacenamiento HDD, así como de una GPU NVIDIA GTX 1070 idónea para realizar el entrenamiento de la red en modo prueba.
- **Equipo de experimentación**. El equipo utilizado para realizar los diferentes experimentos forma parte de los servicios ofrecidos por la Universidad de Jaén a sus investigadores de forma gratuita. Se trata de un *cluster* de computación de altas prestaciones, compuesto por 6 nodos de computación, cada uno de ellos consta de una CPU Intel Xeon, 192 Gigabytes de memoria principal, y tarjetas gráficas. Dos nodos contienen dos GPU NVIDIA Tesla V100 cada uno, mientras que los 4 nodos restantes tienen un total de 26 GPU NVIDIA RTX 2080Ti divididas en los 4 nodos. Se trata, por tanto, de la maquinaria óptima para desarrollar proyectos de esta índole.

1.9.2 Datos

Otro recurso fundamental para desarrollar este trabajo son los datos. En nuestro caso, van a utilizarse tanto datos públicos, disponibles en la web para su descarga gratuita con fines de investigación, como datos LiDAR privados generados por un grupo de investigación de la Universidad de Jaén.

- **Semantic3D**. Se trata de un *dataset* de nubes de puntos LiDAR, que está orientado a la construcción y validación de sistemas de segmentación semántica de nubes de puntos. Se compone de un total de 15 nubes para entrenamiento y un total para validación, con muy alta densidad de puntos.

  La captura se realiza utilizando un escáner LiDAR terrestre, y cada punto se acompaña de atributos sobre la intensidad y color RGB, además de la etiqueta semántica asociada a cada punto. Los puntos se clasifican

en 8 categorías: suelo natural, suelo artificial, vegetación baja, vegetación alta, edificios, coches, etcétera.

- **Datos LiDAR Pamplona.** Se compone de un conjunto de nubes de puntos LiDAR, en formato *las*, que ha sido generado en el contexto del proyecto de investigación SPSLIDAR cuyo uno de sus Investigadores Principales es también director de este Trabajo Fin de Máster. Se trata de un escaneo LiDAR aéreo de la ciudad de Pamplona, y aunque este *dataset* se utilizará en menor medida que el anterior, será de sumo interés de cara a la realización de las tareas de transferencia de conocimiento.

  El objetivo con este *dataset* consiste en, una vez obtenido un modelo entrenado partiendo del *dataset* Semantic3D, muy abundante en cuanto a densidad de puntos, reentrenar este modelo para transferir lo aprendido en el primer *dataset* al segundo. La diferencia entre ambos *datasets* reside en el tipo de escáner utilizado, principalmente. Mientras el primero se genera a partir de LiDAR terrestre, el segundo es producto de un escaneo aéreo, por lo que las zonas con alta densidad de puntos se concentrarán verticalmente en el primer caso, y horizontalmente en el segundo, por lo que es necesario reentrenar la red para que se produzca esta adaptación.

  Además, el formato del primer *dataset* es texto plano en ficheros *txt*, mientras que los ficheros del segundo *dataset* están en formato binario *las*, más óptimo tanto para carga como para el almacenamiento de nubes de puntos.

1.10 Tecnologías utilizadas

En esta sección se describen todas las tecnologías *software* utilizadas en este proyecto.

1.10.1 Lenguaje de programación

Para este proyecto, se ha hecho uso del lenguaje de programación Python (véase Figura 1.28) puesto que dispone de todas las herramientas necesarias para

desarrollar un proyecto con base en el aprendizaje profundo. Existen multitud de librerías que facilitan el trabajo de programación, incluyendo funciones que implementan algoritmos complejos y evitando al programador la necesidad de tener que implementar cada uno de estos algoritmos.

Por ejemplo, los principales *framework* de aprendizaje profundo se desarrollan para su uso desde Python, permitiendo además la ejecución en el procesador gráfico en caso de que esté disponible. Además, aunque esto no ha de ser un motivo concluyente, Python es el lenguaje por excelencia para computación científica, pues facilita las tareas de programación tanto como sea posible de forma que el interés del desarrollo sean los resultados en sí, y no tanto los medios utilizados para conseguir esos resultados.

Figura 1.28: Logotipo de Python. *Fuente: Google.*

1.10.2 Entorno de desarrollo

Para desarrollar y depurar el *software* implementado se hace uso de PyCharm, con licencia *Community Edition*, de uso gratuito sin fines comerciales, cuya interfaz se presenta en la Figura 1.29. Este entorno es desarrollado por la compañía Jetbrains, y se ofrece junto a otros entornos conocidos como IntelliJ IDEA para el lenguaje JAVA o PHPStorm para PHP, por ejemplo.

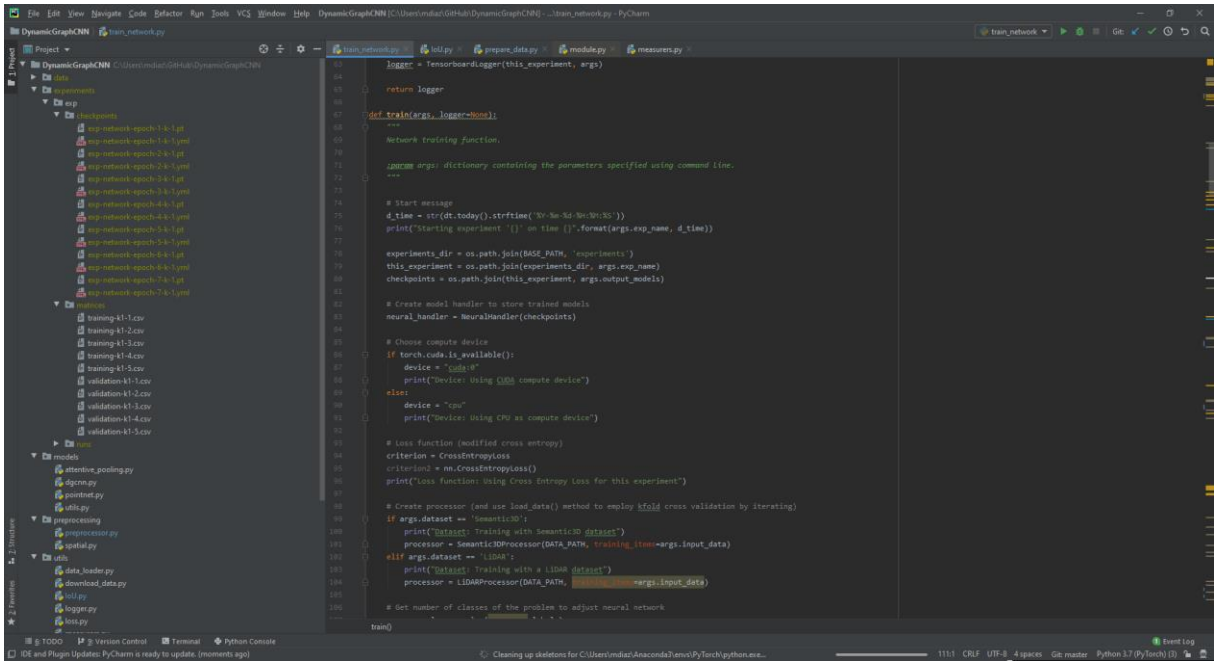

Figura 1.29: Interfaz de PyCharm Community Edition.

Por otra parte, para ejecutar código en Python y mantener las diversas librerías instaladas se necesita un gestor de entornos. En nuestro caso, se hace uso de Anaconda, que permite diferentes entornos virtuales de ejecución, y además permite exportar la configuración del entorno para su posterior importación en otra máquina, habilitando la portabilidad del código.

Figura 1.30: Logotipo de Anaconda 3. Fuente: Google.

1.10.3 *Framework* de aprendizaje profundo

Se utilizará *PyTorch* (Figura 1.31), desarrollado por el departamento de investigación de Facebook (Facebook AI, *Facebook Artificial Intelligence*). Se trata de

un *framework* cada vez más introducido en la comunidad científica, y dotado de funcionalidad para implementar arquitecturas neuronales, al igual que *TensorFlow (*Figura 1.32), de Google.

Figura 1.31: Logotipo de PyTorch. *Fuente: Facebook AI.*

Aunque ambos desarrollos son muy completos en cuanto a funcionalidad, y han demostrado un éxito notable en numerosos trabajos publicados, se toma la decisión de utilizar *PyTorch* debido en parte a la experiencia adquirida previamente en el uso de esta librería, y por otra parte a su construcción siguiendo el paradigma de la Programación Orientada a Objetos (POO). Este paradigma simplifica el desarrollo y depuración del código, puesto que la funcionalidad está modularizada en clases y resulta más sencillo realizar la codificación de *software*.

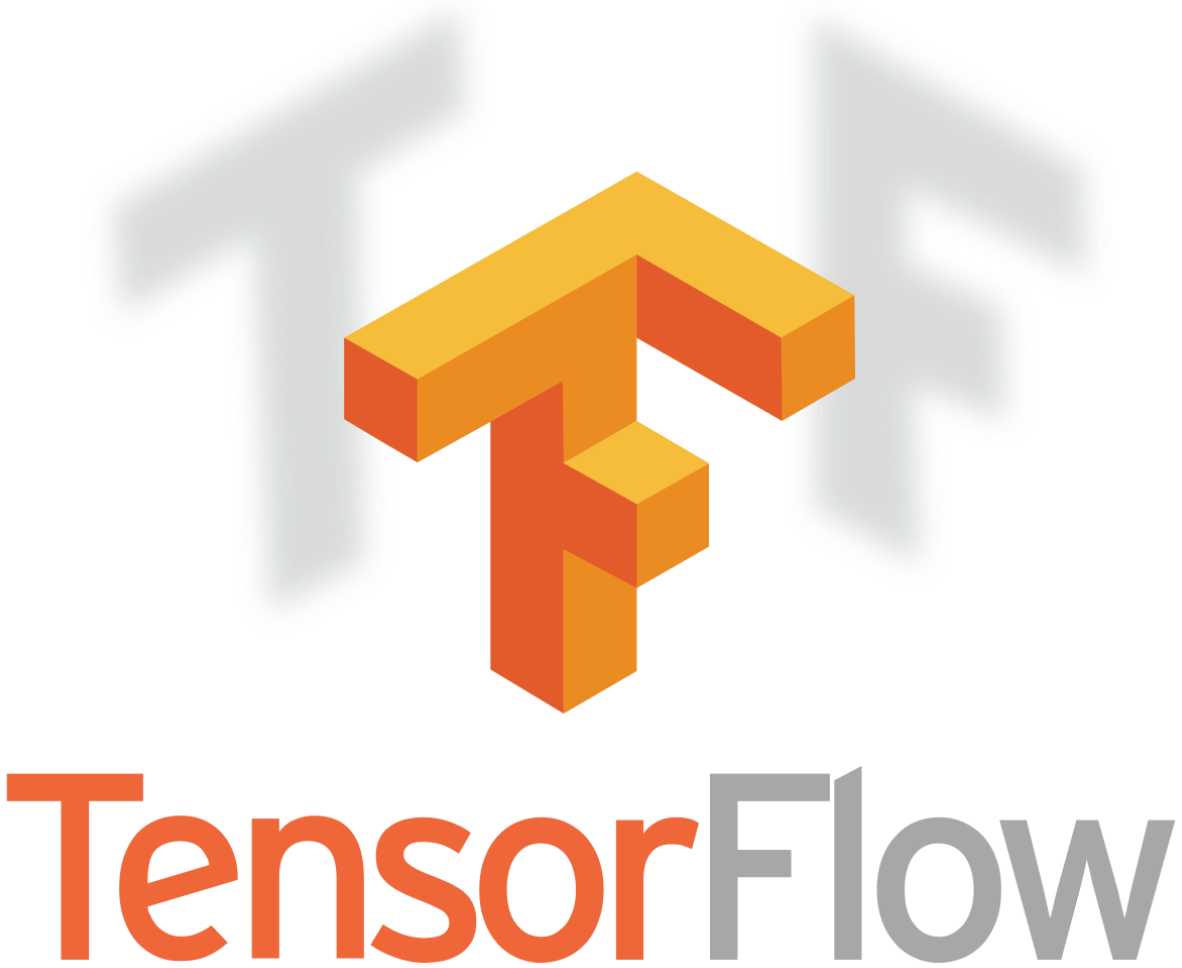

Figura 1.32: Logotipo de TensorFlow. *Fuente: Google.*

Además, ambos *framework* ofrecen funcionalidad para ejecutar código Python en la GPU haciendo uso de la tecnología CUDA, propietaria de NVIDIA, por lo que solo podrán utilizarse procesadores gráficos de esta compañía. No obstante, ambos son complementarios en algunos puntos, puesto que *PyTorch* se aprovecha de *Tensorboard*, una librería para la monitorización del entrenamiento de redes neuronales que se desarrolla en el contexto de *TensorFlow*.

CUDA (*Compute Unified Device Architecture*) se instala de forma independiente a estos *framework*, y estos realizan llamadas a una API (*Application Programming Interface*) ofrecida por dicha instalación para ejecutar funcionalidad sobre la GPU del ordenador. Para esto, se necesita además del módulo *cuDNN* (CUDA *Deep Neural Network*), que contiene las cabeceras de las funciones ofrecidas por CUDA para desarrollar soluciones *deep learning* sobre la GPU.

***Figura 1.33: Logotipo de NVIDIA CUDA**. Fuente: Nvidia.*

1.10.4 Librerías

Aunque se ha comentado la librería más importante a utilizar, *PyTorch*, se mencionan otras librerías que han resultado de utilidad en el desarrollo de este Trabajo Fin de Máster:

- ***Open3D*.** Se trata de una librería con métodos para la manipulación y visualización de geometría utilizando Python. En nuestro caso, dicha librería se ha utilizado para la visualización de nubes de puntos únicamente, aunque incluye funcionalidad adicional para muestrear nubes de puntos, registrar nubes de puntos, etcétera.
- ***NumPy*.** Librería por excelencia en computación científica, orientada a cálculos vectoriales y matriciales, aunque no existe una implementación oficial para la ejecución de operaciones en la GPU. Esta es la librería es la más utilizada en el desarrollo, pues todas las operaciones que se realizan sobre las nubes, incluida la carga de nubes de puntos, se realizan utilizando funcionalidad de esta librería. Existe una implementación de terceros para GPU, ***cuPy***, pero carece de sentido utilizarla debido al trasiego generado entre memoria principal y memoria en GPU, que resulta en una pérdida de rendimiento.
- ***Scikit-learn*.** Funcionalidad para aplicar métodos científicos. Incluye, por ejemplo, implementaciones de algoritmos de inteligencia artificial,

etcétera. En nuestro caso, se ha hecho uso de esta librería para construir las particiones en validación cruzada.

- ***LasPy*.** Librería para la carga y almacenamiento de ficheros en formato LiDAR *las/laz*, utilizada para realizar la carga del *dataset* que lleva el mismo nombre.

Aunque se han utilizado otras librerías para implementar toda la funcionalidad requerida, se considera que las anteriormente mencionadas son las que gozan de mayor interés desde la perspectiva científica.

1.10.5 Control de versiones

Como control de versiones para gestionar el código y guardar cambios, se utiliza el universalmente extendido ***Git***, utilizando como cliente ***GitHub Desktop*** (Figura 1.34) para salvar cambios y enviar los mismos a los repositorios remotos de ***GitHub***, compañía adquirida recientemente por Microsoft.

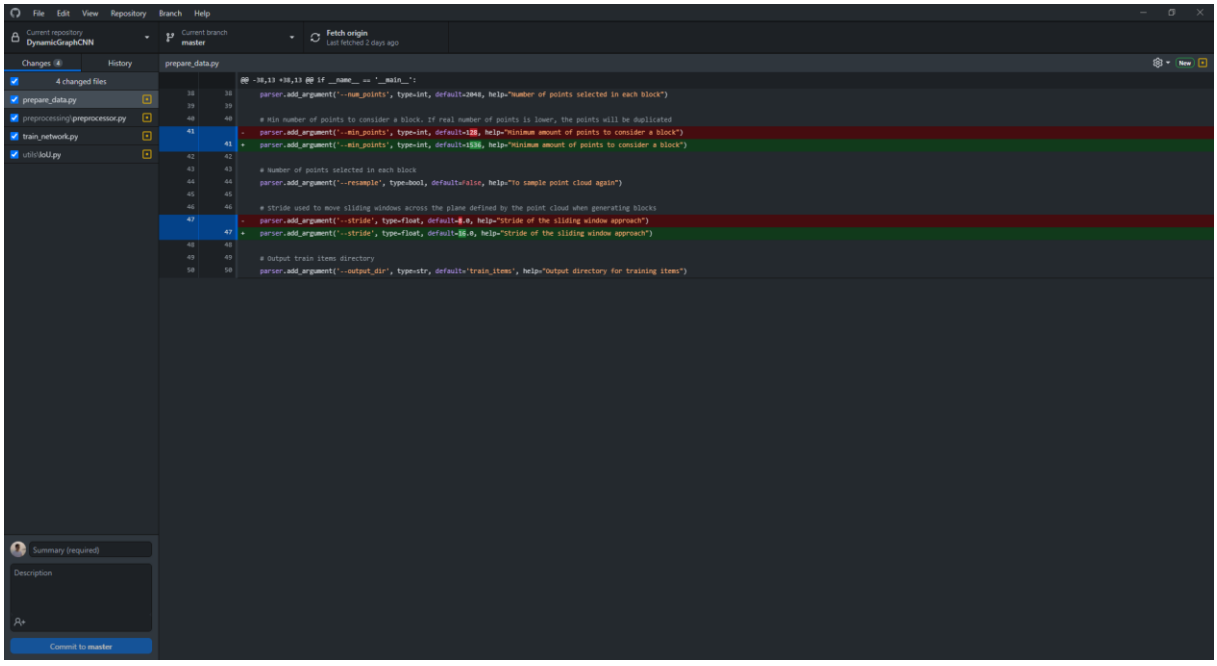

Figura 1.34: Interfaz de GitHub Desktop.

1.10.6 Visor de nubes de puntos

Aunque la visualización de nubes de puntos es, tal vez, la parte menos relevante de este Trabajo Fin de Máster, resulta conveniente añadir que se ha utilizado un visor de nubes de puntos. Permite cargar estos conjuntos de datos en cualquier

formato, tanto si están almacenadas en formato texto plano, como el caso de Semantic3D, como el formato *ply* o *las/laz*.

En nuestro caso, se ha hecho uso de la herramienta ***CloudCompare***, cuya interfaz mostrando una nube de puntos LiDAR se presenta en Figura 1.35), consistiendo en una herramienta de *software* libre ampliamente extendida para la visualización y manipulación de nubes de puntos.

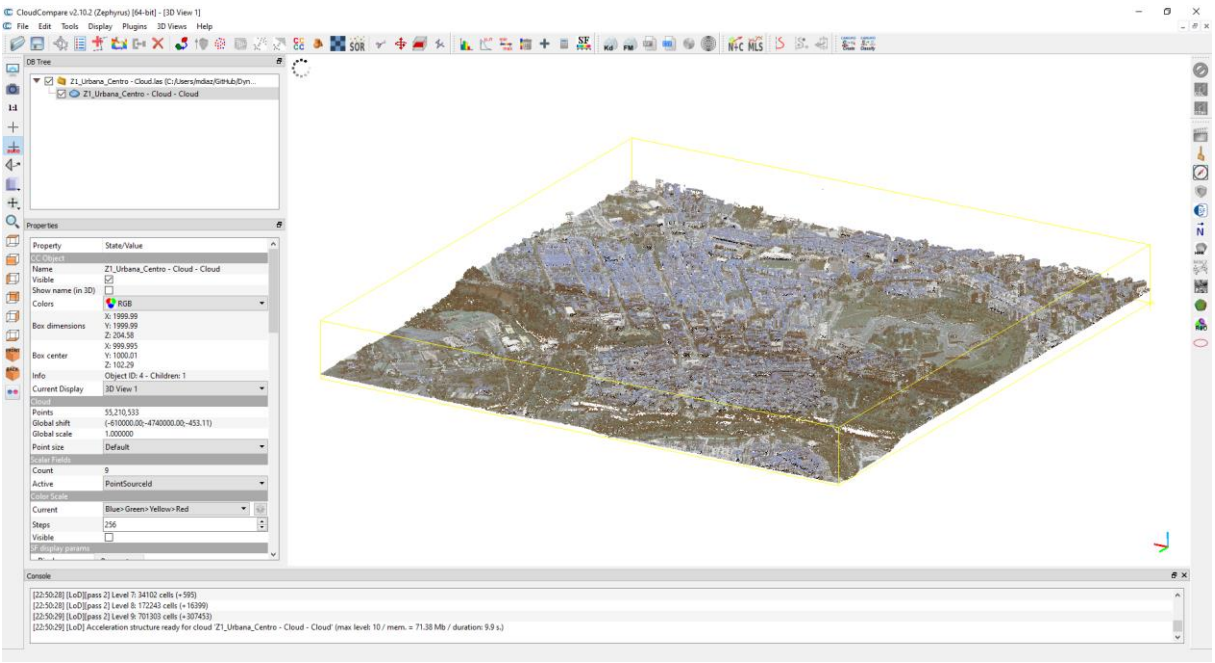

Figura 1.35: Interfaz de CloudCompare.

1.10.7 Sistemas operativos

Para el desarrollo, sobre la arquitectura mencionada en la sección 1.9.1, se ejecuta un sistema operativo Windows 10 (Figura 1.36), con CUDA 10.1 instalado. Sobre este sistema se realiza el desarrollo completo, utilizando los entornos virtuales de Python para garantizar la posterior portabilidad del código entre el ordenador de desarrollo y el servidor de experimentación.

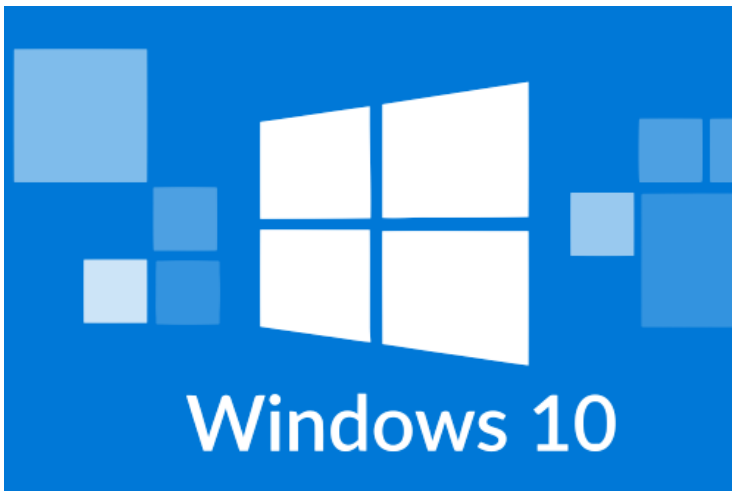

Figura 1.36: Logotipo Windows 10. Fuente: Xataka.

Para el caso de la experimentación, se utiliza un *cluster* de computación de altas prestaciones que ejecuta un sistema operativo Linux, donde se utiliza Slurm como gestor de colas (véase Figura 1.37 y Figura 1.38), y todos los paquetes (Anaconda, CUDA, etcétera) han sido previamente instalados por parte del equipo técnico que administra el *clúster*.

Figura 1.37: Gestor de colas Slurm. Fuente: Google.

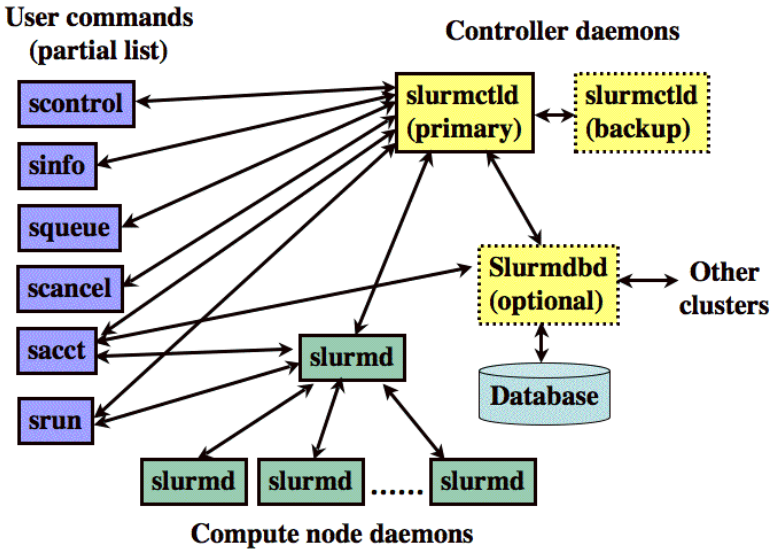

Figura 1.38: Estructura y comandos del gestor de colas Slurm. Fuente: Google.

1.11 Metodología de desarrollo de software

Como metodología de desarrollo de software, se ha optado por el desarrollo incremental. Se trata de un proceso de desarrollo de software que se creó como solución a las debilidades que presentaba el modelo de desarrollo tradicional en cascada (S. Pressman, 2006). La Figura 1.39 ilustra el modelo de proceso de este ciclo de vida.

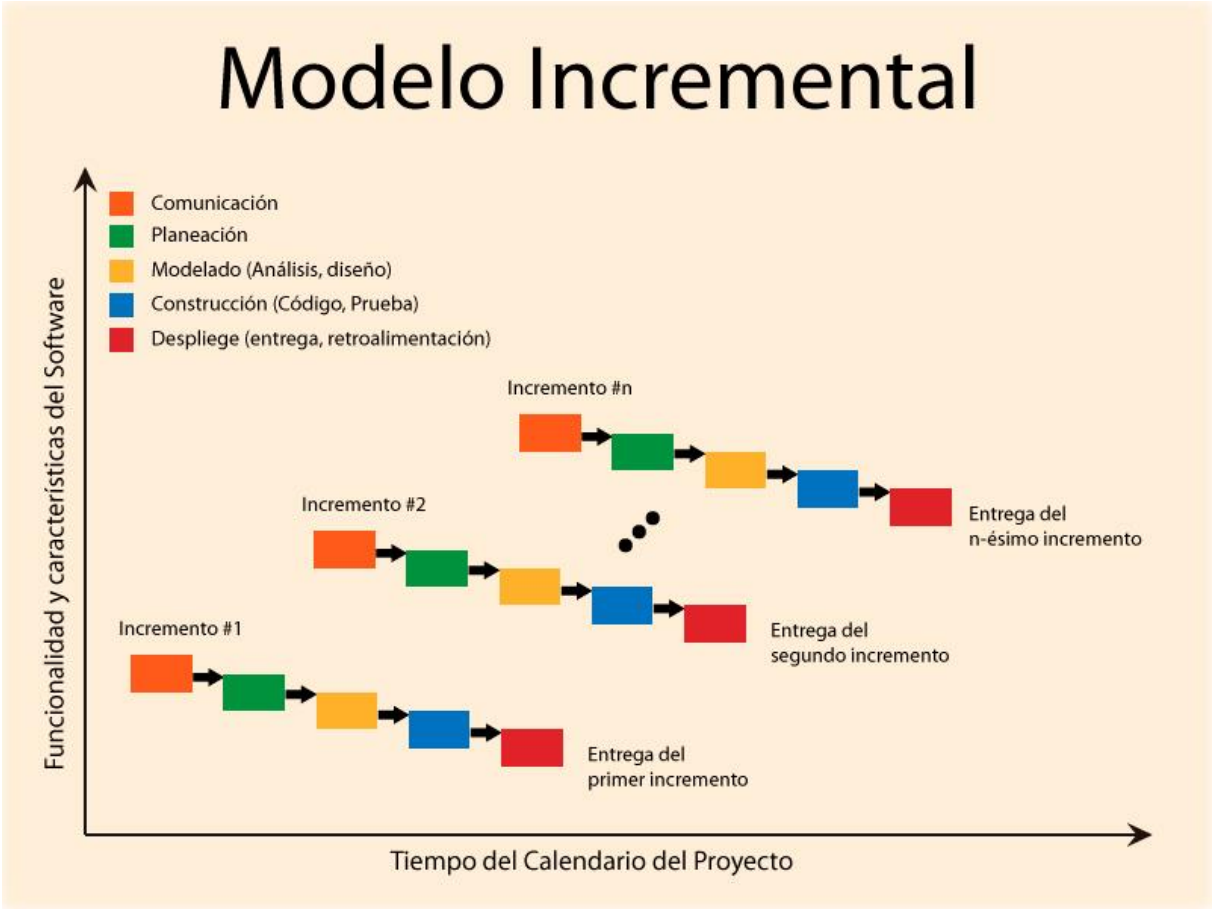

Figura 1.39: Modelo de desarrollo incremental. Fuente: Google.

El proyecto se planifica en distintos bloques temporales que se denominan iteraciones. Tras cada iteración, se repite un proceso que aporta un resultado más completo sobre el producto final, de forma que quien recibe el producto irá obteniendo progresivamente beneficios del producto de forma incremental. Para conseguir esto, cada requisito ha de tener un completo desarrollo en una única iteración que debe incluir pruebas y una documentación para que el equipo pueda cumplir con todos los objetivos que sean necesarios y esté listo para entregarse al cliente. Así, se evita tener arriesgadas actividades en el proyecto una vez finalizado.

Lo que se persigue es que en cada iteración los componentes logren evolucionar el producto dependiendo de los completados en iteraciones anteriores, agregando más opciones de requisitos y logrando así una mejora más completa.

1.11.1 Ciclo de vida

La idea principal tras la mejora iterativa es desarrollar un proyecto de forma incremental, permitiéndole al desarrollador aventajarse de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando versiones entregables del sistema.

Los pasos claves en el proceso son comenzar con una iteración simple de los requerimientos del sistema, y de forma iterativa ir mejorando la secuencia evolutiva de versiones hasta que el sistema completo esté desarrollado. En cada iteración, se realizan cambios en el diseño y se agrega nueva funcionalidad y capacidades al sistema.

El modelo parte básicamente de dos premisas:

1. **El usuario nunca sabe exactamente qué necesita para satisfacer sus necesidades.**
2. **Durante el desarrollo, los procesos son propensos a mutar.**

Así, se consideran las siguientes etapas:

- **Etapa de inicialización.** Se crea una versión básica del sistema. La meta de esta etapa es crear el producto con el que el usuario pueda interactuar, y por ende retroalimentar el proceso. Debe ofrecer una muestra de los aspectos claves del problema y proveer una solución lo suficientemente simple para ser comprendida e implementada fácilmente. Se define una lista de **control de proyecto**.
- **Etapa de iteración.** Esta etapa involucra el rediseño e implementación de una tarea de la lista de control de proyecto, y el análisis de la versión más reciente del sistema. La meta del diseño e implementación de cualquier iteración es ser simple, directa y modular, para poder soportar el rediseño de la etapa o como una tarea añadida a la lista de control de proyecto.

  El código puede, en ciertos casos, representar la mayor fuente de documentación del sistema. El análisis de una iteración se basa en la retroalimentación del usuario, y en análisis de la funcionalidad disponible.

1.11.2 Justificación al respecto de la metodología escogida

Dado que este Trabajo Fin de Máster se desarrolla en la modalidad de trabajo teórico-experimental, ligado a un potente ejercicio de investigación necesario para abordar el proyecto, lo más adecuado es optar por una estrategia incremental.

Habitualmente, cuando se desarrolla un trabajo de investigación, existe una alta incertidumbre al comienzo de todo proyecto de esta naturaleza debido, en todo o en parte, a la cuantiosa literatura científica necesaria para concienciarse del problema y al tiempo necesario a invertir hasta lograr un dominio relativo en la materia. Por tanto, en las primeras etapas del desarrollo, suele implementarse funcionalidad básica mientras se sigue avanzando en la lectura de literatura científica y la extracción de conocimiento, mientras que se posponen para fases más avanzadas los aspectos más complejos a desarrollar.

Por establecer una relación con este proyecto, en las primeras iteraciones se desarrolla funcionalidad para cargar nubes de puntos y generar los bloques, puesto que esta característica es común a la mayoría de sistemas de segmentación semántica. Mientras tanto, no es hasta las últimas iteraciones cuando se implementa y se prueba el modelo neuronal, puesto que se trata del componente más complejo de este desarrollo, y requiere de una base sólida tanto a nivel teórico como a nivel práctico.

Por otra parte, el objetivo de un trabajo teórico-experimenta no es el desarrollo en sí ni tampoco un producto software, sino el de hacer un análisis detallado del estado del arte de la problemática en cuestión, y realizar experimentos al respecto con una de las alternativas estudiadas o propuestas. Por tanto, no se considera un ciclo de vida diferente al incremental, puesto que éste se adapta de forma natural a la problemática existente en el ámbito de la investigación científica. No se dispone de unos requerimientos o requisitos mínimos a satisfacer en lo relativo a un sistema software que puedan definirse por un cliente o *stakeholder*, sino que se dispone de unos resultados conseguidos a través de unas metodologías por una comunidad científica, y se trata de mejorar dichos resultados, igualarlos introduciendo una alternativa más sencilla, o al menos emitir un juicio de valor sobre qué pasaría si se utiliza una arquitectura óptima utilizando un conjunto de datos diferente, con características diferentes.

Por tanto, y por lo comentado antes, en este Trabajo Fin de Máster solo existe la figura de desarrollador dada la naturaleza y modalidad en que se desarrolla este Trabajo Fin de Título.

1.12 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFM, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (la persona autora del trabajo).

1.13 Planificación temporal

Para conseguir todos los objetivos planteados al inicio de este Trabajo Fin de Máster, resulta necesario planificar temporalmente el proyecto de forma que se determine la tarea a realizar en cada período con respecto a la duración total. Puesto que el proyecto comienza a desarrollarse a finales de octubre de 2020 y finaliza en abril de 2021, se tiene una duración total de 25 semanas para completar el proyecto en su totalidad, a un régimen de 12 horas semanales hasta completar las 300 horas obligatorias para el desarrollo del Trabajo Fin de Máster.

Es necesario destacar que el desarrollo de este trabajo de investigación se desarrolla en paralelo a un trabajo a jornada completa desde principios de noviembre de 2020, por lo que la planificación realizada se trata de una estimación temporal y las duraciones reales podrían variar ligeramente.

1.13.1 Estimación temporal de tareas

Se considera la siguiente tabla que recoge las diferentes tareas a realizar, así como las duraciones de cada una de ellas.

	Tarea	Duración estimada
1	Coordinación y gestión del proyecto	1 semana
2	Elaboración del estado del arte	5 semanas
3	Estudio de alternativa seleccionada	2 semanas
4	Desarrollo de *software*	6 semanas
5	Pruebas de *software*	1 semana
6	Experimentación	4 semanas
7	Elaboración de la memoria	6 semanas
Total		25 semanas

Tabla 1: Estimación temporal de tareas.

1.13.2 Diagrama de Gantt

Para establecer la relación temporal y orden entre la realización de las diferentes tareas, se considera un **diagrama de Gantt**.

		Semana										
Tarea	Duración	1	2	3	4	5	6	7	8	9	10	11
1	1 s.											
2	5 s.											
3	2 s.											
4	6 s.											
5	1 s.											
6	4 s.											
7	6 s.											

Tabla 2: Diagrama de Gantt (I).

		Semana										
Tarea	Duración	12	13	14	15	16	17	18	19	20	21	22
1	1 s.											
2	5 s.											
3	2 s.											
4	6 s.											
5	1 s.											
6	4 s.											
7	6 s.											

Tabla 3: Diagrama de Gantt (II).

		Semana			
Tarea	Duración	23	24	25	
1	1 s.				
2	5 s.				
3	2 s.				
4	6 s.				
5	1 s.				
6	4 s.				
7	6 s.				

Tabla 4: Diagrama de Gantt (III).

1.14 Presupuesto

A lo largo de esta sección se realiza el análisis económico de este proyecto, tratando los costes asociados al mismo. Se distingue entre recursos software, recursos hardware, datos, recursos humanos y por último costes indirectos, todos los asociados a electricidad, etcétera.

1.14.1 Recursos software

El desglose de los recursos *software* utilizados en este proyecto, así como los datos utilizados para el entrenamiento de los modelos neuronales, se incluye a continuación:

Recurso	Coste económico
Licencia Windows 10 Pro	259,00 €
Anaconda 3 + Python 3.8	0,00 €
Semantic3D	0,00 €
Datos LiDAR Pamplona	0,00 €
PyCharm Community Edition	0,00 €
Cloud Compare	0,00 €
Slurm (gestor de colas)	0,00 €
Red Hat (cluster)	0,00 €
Nvidia CUDA	0,00 €
Total	259,00 €

Tabla 5: Desglose de presupuesto para software.

Suponiendo que el coste total de los recursos *software* se amortizan en 5 años, necesitaría pagarse una cantidad de $\frac{259}{5} = 51{,}8€/\text{año}$ para amortizar por completo el dinero desembolsado. Sin embargo, dado que la duración del proyecto es de 6,25 meses, durante el desarrollo del mismo se amortiza una cantidad de $\frac{51{,}8€/año}{12meses/año} * 6{,}25 = 26{,}97€$ en los meses que dura el proyecto.

Por tanto, con respecto al total, se amortiza un $\frac{26{,}97}{259{,}00} * 100 = 10{,}41\%$ del total gastado en recursos software.

1.14.2 Recursos hardware

1.14.2.1 Desarrollo

En primer lugar, se consideran los costes asociados al ordenador de desarrollo de software, y donde se realizará también el desarrollo de la memoria de resultados asociada al proyecto.

Componente	Coste económico
Intel Core i7-8700K 3.7Ghz BOX	408,85 €
Gigabyte Z390 Gaming X	140,00 €
G.Skill Trident Z RGB DDR4 3200 PC4-25600 16GB 2x8GB CL16	123,99 €
Nfortec Hydrus 240 Kit Refrigeración líquida Roja	59,96 €
Samsung 970 EVO Plus 500GB SSD NVMe M.2	97,40 €
Corsair Carbide SPEC-06 Cristal Templado RGB White	80,00 €
Tacens Mars Gaming Vulcano 750W 80 Plus Silver Modular Rojo	68,50 €
Gigabyte GeForce GTX 1070 G1 Gaming 8GB GDDR5	450,00 €
Total	1.428,70 €

Tabla 6: Recursos hardware del ordenador de desarrollo.

1.14.2.2 Cluster

Por otra parte, resulta de interés considerar el coste económico asociado al *cluster* de HPC (*High Performance Computing*) donde se ejecutarán los experimentos que ofrecerán los resultados del estudio.

Componente	Coste económico
26x Nvidia RTX 2080Ti	26 * 1.139,90 = 29.637,4 €
4x Nvidia Tesla V100	4 * 10.000 = 40.000 €
14x Intel(R) Xeon(R) Silver 4210 CPU	14 * 500 = 7.000 €
70.8 TB Almacenamiento	71 * 33,50 = 2.378,5 €
1,248 TB Memoria RAM	39 (módulos) * 194,99 = 7.604,61 €
Infraestructura (racks, instalación, etcétera)	45.000 €
Total	131.620,51 €

Tabla 7: Recursos hardware del cluster de HPC.

1.14.2.3 Total

La cantidad total correspondiente a recursos hardware se corresponde con:

Equipo	Coste económico
Personal/desarrollo	1.428,70 €
***Cluster* HPC**	131.620,51 €
Total	133.049,21 €

Tabla 8: Coste total de recursos hardware.

Dado que el coste total de los recursos *hardware* es de 133.049,21 €, supóngase que esta cantidad se amortiza en un período fijo de 5 años, es necesario pagar $\frac{133.049,21}{5}$ € $= 26.609,84$ € anuales para conseguir la amortización de los recursos *hardware*. Por tanto, y teniendo en cuenta que el proyecto se completa en 25 semanas, lo equivalente a 6,25 meses, se amortizaría una cantidad de $\frac{26.609,84\ €/año}{12\ meses/año} * 6,25\ meses = 13.859,29$ €, lo que supone una fracción del $\frac{13.859,29}{133.049,21} * 100 = 10,41\%$ del total amortizado en cuanto a recursos hardware.

Por tanto, el **coste de los recursos hardware imputable al proyecto** es solo la fracción amortizada, siendo esta cantidad de 13.859,29€.

1.14.3 Recursos humanos

Los recursos humanos implicados en este proyecto reciben una dotación económica que se detalla a continuación:

Rol	Tarifa	Horas	Coste económico
Investigador Principal	40€/hora	300	1.200,00 €
Subtotal	1,33 Seguridad Social		1.200,00 € * 1,33
Total			1.596,00 €

Tabla 9: Coste económico de los recursos humanos del proyecto.

1.14.4 Costes derivados del proyecto

Los costes derivados o indirectos del proyecto incluyen cantidades asociadas al consumo eléctrico, bienes fungibles de oficina, desplazamientos, etcétera. Dadas las circunstancias especiales en que se ha desarrollado este trabajo, los gastos asociados a desplazamientos para ir a reuniones o exposiciones se reducen a cero. Sin embargo, es necesario tener en cuenta las labores de mantenimiento del *cluster* de HPC, consumo eléctrico, calefacción, etcétera. Así, los costes indirectos del proyecto se calculan aproximadamente como el 21% del total del proyecto.

Recurso	Coste económico
Software (amortizado 6,25 meses)	26,97 €
Hardware (amortizado 6,25 meses)	13.859,29 €
Recursos humanos	1.596,00 €
Costes indirectos (*0.21)	3.251,27 €

Tabla 10: Costes indirectos del proyecto.

1.14.5 Total

En última instancia, se desglosa el coste total del proyecto, teniendo en cuenta únicamente la parte amortizada del mismo y no el total estimado.

Recurso	Coste económico
Software (amortizado 6,25 meses)	26,97 €
Hardware (amortizado 6,25 meses)	13.859,29 €
Recursos humanos	1.596,00 €
Costes indirectos	3.251,27 €
Total	18.736,53€

Tabla 11: Coste total del proyecto.

Por tanto, dado que el proyecto consta de una duración estimada de 25 semanas, 6,25 meses ~ 7 meses, se pagarían 2.997,84€ mensuales durante los primeros 6 meses, y 749,46€ en el último mes de desarrollo del proyecto, pues solo se ocuparía una semana del mismo.

(Página intencionalmente en blanco)

2 DESARROLLO

2.1 Técnica a implementar

En este apartado se realizará un análisis pormenorizado de la técnica a implementar en cuestión, analizando tanto el preprocesamiento de datos necesario como la posterior carga y propagación hacia delante en la red neuronal.

2.1.1 Preprocesamiento de datos

Es necesario que el software implementado sea capaz de cargar diferentes formatos de nubes de puntos. Esto es, podrán leerse de disco tanto nubes de puntos en formato *laz/las* como nubes de puntos en formato texto plano. Por tanto, será necesario distinguir entre los diferentes tipos de *dataset* a considerarse. Principalmente, en este sentido van a considerarse dos tipos: *Semantic3D*, utilizado para el entrenamiento y evaluación principal de la red neuronal, y *LiDAR*, utilizado en fases posteriores del desarrollo como la transferencia de conocimiento o la inferencia.

Con independencia al formato de datos de entrada, el tratamiento por parte del software debe ser el mismo en ambos casos, de forma que no sea necesario implementar código según el tipo de *dataset* a utilizar. Por otra parte, las nubes de puntos pertenecientes al *dataset Semantic3D* corresponden a escaneados de exteriores realmente densos. Para poder manipularlos de una forma eficiente, es necesario llevar a cabo un proceso de muestreo que seleccione una muestra lo suficientemente significativa de la nube de entrada como para que se preserve la mayor cantidad de información en el menor volumen de datos posible. En este caso se optará por el muestreo aleatorio por ser el más sencillo y óptimo en tiempo de ejecución.

A medida que se cargan los datos, se normalizan los canales RGB al intervalo [0, 1], Esto facilitará el aprendizaje de la red, así como acelerará la velocidad de convergencia del modelo. Sin embargo, en el caso del *dataset Semantic3D* no se dispone de un rango para el canal de intensidad, por lo que es imposible realizar una normalización correcta teniendo en cuenta este hecho. Así, se introducirá esta información en la red tal y como está contenida en cada punto y en el rango original.

2.1.1.1 Generación de bloques

En cada propagación hacia delante de la red, se procesará un conjunto determinado de bloques, compuesto cada uno de ellos por un número preestablecido de puntos. A diferencia de los enfoques basados en voxelización, los puntos están contenidos en su formato original y no se ha realizado ningún tipo de discretización sobre ellos. Para generar los bloques, es necesario iterar sobre todas las nubes, dividir cada una de ellas en bloques de tamaño fijo considerando el plano horizontal de la nube, y extraer puntos de cada bloque siguiendo un enfoque de ventana deslizante. Es necesario preseleccionar el número de puntos que han de extraerse cada bloque en cada momento.

Si el número de puntos contenidos en un bloque es menor al deseado, será necesario realizar un muestreo con reemplazamiento para replicar los puntos existentes tantas veces como sea necesario para conseguir el número deseado. Esto no afectará al rendimiento de la red.

Es importante destacar que el enfoque de ventana deslizante se aplica sobre los ejes X e Y, considerando el eje Z como el eje vertical o perpendicular a la superficie terrestre. Por tanto, habrá bloques con una alta variación en el eje Z, y bloques con una variación mínima, aunque el estudio de esta variación no es un objeto de interés. Los bloques han de tener un área fija, por ejemplo: $1m^2$ o $4m^2$. A medida que se generan bloques, han almacenarse en una estructura común a todas las nubes de puntos procesadas, junto a la etiqueta de clase asociada a cada punto.

Si el número de puntos contenidos en un bloque es mayor al deseado, se aplicará un muestreo sin reemplazamiento para seleccionar una muestra de tamaño igual al número deseado. En caso de que este número sea menor, se realizará lo indicado inmediatamente antes.

2.1.1.2 Generación de *batches* de entrenamiento

Una vez han terminado de procesarse todas las nubes de puntos, y se dispone de una lista no ordenada compuesta por bloques de igual número de puntos junto a sus etiquetas semánticas, es necesario almacenar estos bloques teniendo en cuenta un enfoque de validación cruzada que permita evaluar con eficacia la precisión del modelo.

Por tanto, en una primera versión se optará por *K-fold Cross-Validation*. De esta forma, la lista de bloques ya procesados se dividirá en K partes, y se generarán $K \times 2$ ficheros diferentes, identificados de $[0, K-1]$ y en versiones de entrenamiento y test. Los ficheros de entrenamiento contendrán $K-1$ partes del total, usándose estas para entrenar, y el fichero de test correspondiente a esta parte contendrá una única parte que será la utilizada para la validación del modelo en cada *epoch*.

2.1.2 Arquitectura de la red neuronal

En esta sección se detalla el funcionamiento de la red neuronal a utilizar.

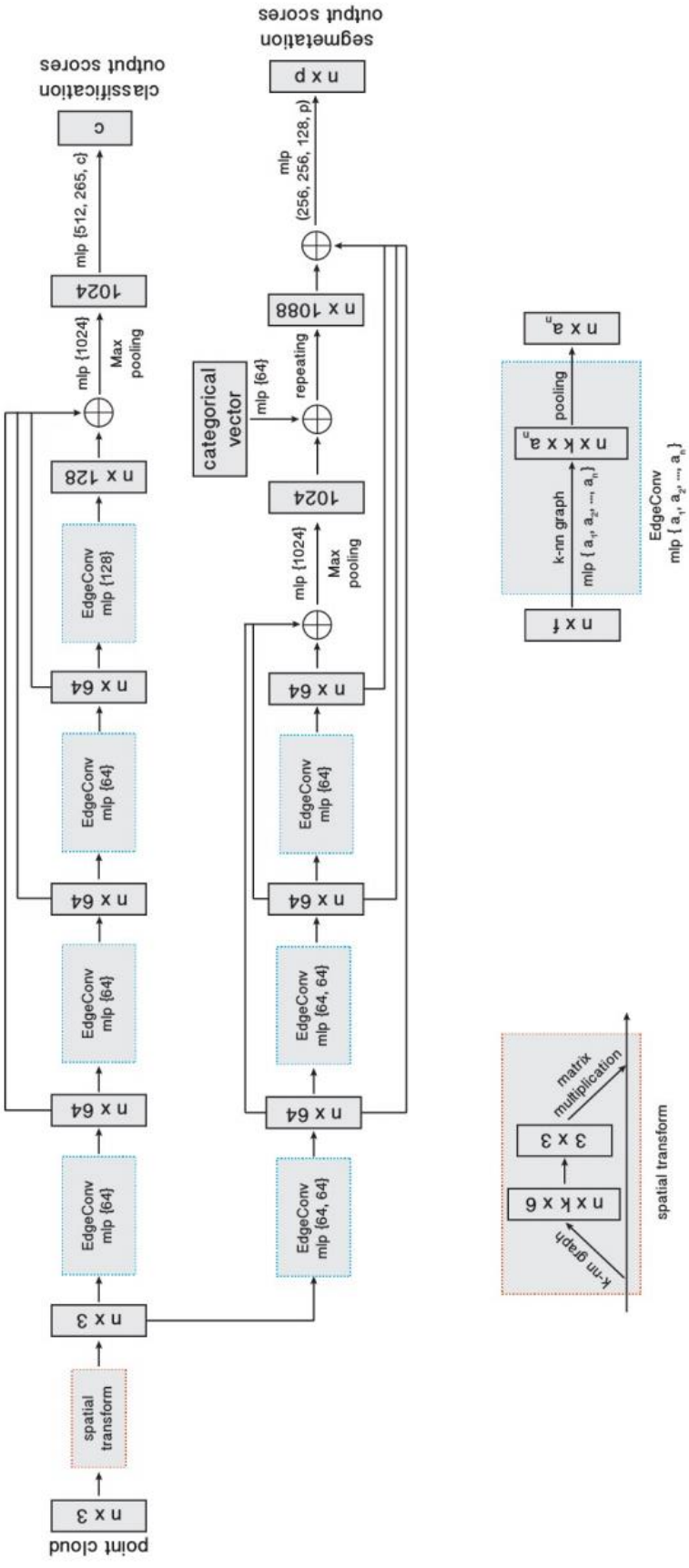

Figura 2.1: Modelo de red neuronal a utilizar.

La idea más básica y que da paso al desarrollo del trabajo completo que inspira este proyecto es la **construcción dinámica del grafo de vecinos más cercanos tras cada capa de procesamiento en la red**. Se trata de una red neuronal convolucional, pues en cada paso de convolución analiza y agrega información de los vecinos más cercanos a un punto, de forma que un punto no tenga información sobre sí mismo en el paso de segmentación, sino que también conozca peculiaridades sobre cómo se distribuye su entorno.

En tanto, en la Figura 2.1 podemos observar una arquitectura neuronal completa, donde son dos las posibles salidas de la red. Esta arquitectura está tanto orientada a tareas de segmentación semántica (brazo inferior, con N posibles salidas) como a tareas de clasificación de nubes de puntos (brazo superior, con 1 única salida para los N puntos de entrada. Recuérdese, del capítulo introductorio, que la principal diferencia que separa a un sistema de segmentación de un sistema de clasificación es el número etiquetas de salida que genera cada algoritmo, **1 para cada punto *versus* 1 para todos los puntos.**

Aunque en Figura 2.1 puede observarse un módulo STN (*Spatial Transformer Network*) situado en la entrada de la red, finalmente no se utilizará en la implementación puesto que se ha observado que en el desarrollo original, este módulo no tenía utilidad de cara a mejorar los resultados de la segmentación semántica. Así, el módulo principal sobre el que se construye esta arquitectura es *EdgeConv*.

2.1.2.1 EdgeConv

Constituye la piedra angular sobre la que se construye esta arquitectura de red, y consta de algunos pasos clave:

- **Entrada.** Toma como entrada una matriz de puntos, $n * f$, con un total de n puntos y f características cada uno de los puntos. El número de puntos se mantendrá en todas de capas de la red, mientras que lo que variará serán las características que acompañan a estos puntos. En el enfoque más básico, la entrada de la red, esta matriz estará compuesta por los n puntos de entrada de la nube, y para cada uno de ellos se tendrán las tres coordenadas (x, y, z), la intensidad, el color, etcétera. Cuando se recibe esta matriz de entrada, se calcula el grafo de características usando los k vecinos más cercanos de cada punto.

- **Grafo de características.** Para cada punto de entrada, se calculan sus vecinos más cercanos utilizando la distancia euclídea y considerando los puntos embebidos en $\mathbb{R}^F$. Cada uno de los vecinos a los puntos tiene sus propias características. Tras calcular los k vecinos más cercanos a un punto, se extraen sus características y se agregan, utilizando la última expresión citada en 1.3.2.9, de forma que cada punto tenga conciencia tanto del entorno a nivel local como a nivel global. Así, la matriz de entrada pasará de ser (n, f) a $(n, k, 2f)$, pues cada punto tendrá información sobre las características de todos los vecinos más cercanos, k, y el número de características de cada punto vecino se doblará con respecto al inicio, teniendo este por un lado las F características del punto de entrada, repetidas en los k vecinos, y por otro lado cada uno de los k vecinos tendrá otras F características, resultando de realizar la diferencia entre sus características y las características del punto original al que se calcularon sus vecinos.

- **MLP (*Multilayer Perceptron*).** Una vez calculado el grafo de vecinos más cercanos, es necesario aplicar la lógica inteligente para la identificación y búsqueda de patrones. En tanto, se trata de inferir nuevas características a partir de las características combinadas de punto origen y vecinos. Para mantener la filosofía de *PointNet* (Charles R Qi et al., 2016), y conseguir el efecto de perceptrón multicapa compartido, es necesario implementar este fenómeno utilizando capas de convolución, con independencia del *framework* de aprendizaje profundo a utilizar. La característica que distingue a las capas de convolución de los tradicionales perceptrones multicapa es el hecho de que el *kernel* pueda compartirse sobre los datos de entrada. Así, el tamaño de *kernel* será de 1 para que solo se considere 1 muestra de entrada a la vez. De esta forma, el número de puntos de salida será igual al de entrada. Ajustando el número de filtros (*kernels* utilizados), se consigue el número de características de salida.

  En tanto, para un grafo de características de entrada de dimensiones $(n, k, 2f)$, se obtiene una matriz de salida de dimensiones (n, k, a_n), siendo este último término un número arbitrario. En nuestro código,

habitualmente el número de características de salida será de 64. En algunos bloques se utilizarán 2 MLP, uno seguido de otro, mientras que el último bloque *EdgeConv* tan solo se utilizará 1 MLP.

- ***Pooling.*** El último paso dentro de un bloque *EdgeConv* consiste en la aplicación de una capa de agregación o reducción. Esta capa seleccionará las características más relevantes o frecuentes de los k vecinos de cada punto, y generará como salida un único vector de características de tamaño a_n para cada punto. Así, la matriz de entrada será de dimensiones (n, k, a_n) y la matriz de salida será (n, a_n). Para cada una de las a_n características presentes en todos los vecinos de un punto, seleccionará un valor de acuerdo a una filosofía de agregación. En nuestra implementación, se utilizará tanto la agregación por *max-pooling* como la novedosa agregación por *attentive pooling*.

En definitiva, el módulo *EdgeConv* recibe una matriz de puntos con sus diferentes características, y genera otra matriz de puntos similar de salida, variando el número y tipo de características. Las características de entrada serán de bajo nivel, mientras que las de salida serán abstracciones de más alto nivel sobre las primeras, tal y como establece la filosofía de la convolución clásica. Las características de salidas estarán formadas por una agregación abstracta de las características de cada punto de entrada con las características de sus vecinos.

Los bloques *EdgeConv* se aplican un total de 3 veces en la red, y de esta forma van dotando a todos los puntos con información local sobre su entorno, así como información global del resto de entornos locales presentes en la nube de entrada.

2.1.2.2 Agregación de información

Tras aplicar cada una de las 3 capas *EdgeConv*, la salida de sendas capas se concatena. Esto es, la salida de cada capa será una matriz $(n, 64)$, compuesta por los puntos de la nube de entrada, y 64 características. En tanto, tras la concatenación a lo largo del eje de las características, se obtiene una nueva matriz de dimensiones $(n, 192)$. En esta nueva matriz, cada punto contendrá características a 3 niveles de abstracción diferentes. A medida que el nivel de abstracción es más alto, contiene información sobre un entorno más global de la nube.

Tras la concatenación de las 3 matrices, se aplica de nuevo un Perceptrón Multicapa, codificado como una capa de convolución 1d, y se obtiene una nueva matriz de dimensiones $(n, 1024)$. Cada uno de los puntos está descrito, en este momento, por un vector de 1024 características.

El objetivo es obtener un vector de características global, tal y como realiza *PointNet* (Charles R Qi et al., 2016) en su arquitectura. Para ello, se aplica una capa de *pooling* que seleccionará las 1024 características más frecuentes en todos los puntos. Esto es, dada la matriz de características $(n, 1024)$, donde cada columna representa una característica diferente y cada fila un punto diferente, seleccionará el valor más frecuente (*max-pooling* o *attentive pooling*) de cada columna, obteniendo un nuevo vector de características global de tamaño 1024.

A continuación, a las 3 matrices concatenadas antes, se le vuelve a concatenar este vector de características para cada punto. De este modo, se obtiene una nueva matriz $(n, 192 + 1024) = (n, 1216)$, y así, cada punto tiene desde este momento información global sobre la nube de puntos.

2.1.2.3 Capas de clasificación

Una vez completada la fase de extracción de características, es necesario culminar con el proceso de segmentación semántica, consistente en asignar una etiqueta semántica a cada uno de los puntos de entrada. Para ello, se aplicarán perceptrones multicapa, codificados como capas de convolución 1d. Se aplica un total de 3 capas, reduciendo sucesivamente la dimensionalidad, migrando de una matriz de características $(n, 1216)$ a una matriz de etiquetas (n, p) donde p representa el número de clases del problema, o el número de categorías semánticas distinguibles en el problema de segmentación.

2.1.2.4 Aumento de datos

Es habitual utilizar alguna técnica de aumento de datos que permita incrementar la capacidad de generalización de la red. Este proceso consiste en modificar ligeramente los datos de entrada, de forma que los datos mantengan la distribución de probabilidad del *dataset* completo, pero varíen lo suficiente como para enseñar a la red diferentes perspectivas sobre un mismo ítem de información, incrementando la capacidad de generalización de la red.

En este caso, la contribución al aumento de datos consiste en alterar de forma aleatoria el orden de los puntos contenidos en cada una de las nubes que la red tendrá que procesar. De esta forma, la red no **memorizará** el orden de los puntos para asignar una etiqueta, sino que obtendrá invariabilidad a permutaciones, algo imprescindible para el tratamiento de cualquier nube de puntos. Esto es, el resultado de segmentación debería ser el mismo con independencia del orden en que sea alimentada la red con los diferentes puntos contenidos en la nube.

2.1.2.5 Attentive Pooling

A modo de innovación, se introduce un nuevo concepto de reducción de características conocido como *Attentive Pooling*, extraído principalmente del trabajo (Hu et al., 2020), al tratarse de una novedosa técnica para segmentación de nubes de puntos de gran tamaño.

Se trata de un concepto con una filosofía similar al *Average Pooling* ya conocido (Goodfellow et al., 2016), salvo que en lugar de emplear una media aritmética emplea una media ponderada, y los pesos de ponderación se infieren *on-the-fly* en cada propagación a través de esta capa. Así, la operación es:

$$x' = \sum_{f \in F} w^i * x^i$$

Donde w^i es el peso aplicado en cada característica, y este parámetro es inferido mediante la siguiente función:

$$W = f_\Theta(x)$$

Siendo Θ un conjunto de parámetros entrenables. En este sentido, se evaluará la calidad de este enfoque mediante la comparación del mismo con respecto al clásico y exitoso *max-pooling*.

Una vez descrita la técnica que ha de implementarse, se procede a documentar el proceso de desarrollo de cada una de las partes que componen el sistema total. Si bien se documentará el proceso completo, siguiendo un esquema de desarrollo iterativo, se comentarán los aspectos más relevantes desde el punto de vista **científico** puesto que el objetivo de este proyecto no es el desarrollo en sí sino los resultados obtenidos.

2.2 Iteraciones

2.2.1 Primera Iteración

En la primera iteración de desarrollo, es necesario plantearse la estructura completa que tendrá el sistema una vez finalizado y comenzar por las partes más fundamentales, como la creación del proyecto o la carga de datos.

2.2.1.1 Creación del proyecto

Para realizar el desarrollo de *software* de este proyecto, se utilizará el lenguaje Python en su versión 3.8, sobre un entorno gestionado por Anaconda 3. Como entorno de desarrollo integrado, IDE (*Integrated Development Environment*), se utilizará PyCharm por el hecho de poseer experiencia previa y un buen manejo de la herramienta. Es necesario configurar todas las librerías necesarias para desarrollar el proyecto, incluyendo algunas básicas como *NumPy*, así como el *framework* de aprendizaje profundo a utilizar.

Para esto, se crea un entorno virtual en el ordenador de desarrollo, sobre el que se instalarán todas las librerías que se utilizarán durante el desarrollo y posterior ejecución del proyecto. Tras la creación del entorno, se crea el proyecto en PyCharm, siguiendo la siguiente estructura, en cuanto a directorios:

- ***Data***. Contiene una jerarquía de ficheros que contienen los *dataset* a utilizar.
- ***Models***. *Scripts* Python que implementarán tanto la arquitectura de red neuronal como funciones útiles de cálculo de los vecinos más cercanos, por ejemplo.
- ***Preprocessing***. Clases y métodos para la lectura y escritura de datos, división de las nubes en bloques, etcétera.
- ***Utils***. Funciones útiles para visualización de datos, cálculo de tiempos, registro de eventos, cálculo de precisión y coste, visores de nubes de puntos, etcétera.

- ***Raíz***. Los *scripts* raíz de ejecución, siendo estos: el *script* para generar el conjunto de datos, el *script* de entrenamiento, y el *script* de inferencia sobre nubes de puntos no clasificadas.

2.2.1.2 Datasets

Principalmente, el sistema soportará dos formatos de *dataset* diferentes: Semantic3D y LiDAR. Para ello, se crearán dos clases con el mismo nombre que el conjunto de datos al que representan, y estas clases heredarán de una superclase *Processor*. Esta clase implementará toda la funcionalidad para el manejo de nubes de puntos: troceo en bloques, generación del conjunto de entrenamiento, muestreo de nubes, remuestreo, etcétera.

Sin embargo, cada *dataset* se almacenará en directorios diferentes, y este comportamiento se definirá en su respectiva clase. Mediante polimorfismo, se instanciarán a los métodos de la clase padre, y automáticamente se llamarán a los métodos de las hijas si se trata de métodos abstractos.

Un ejemplo de método abstracto es la carga de la nube. El *dataset* Semantic3D está almacenado en ficheros de texto plano *txt*, y es necesario utilizar unas funciones concretas para su carga y manipulación. Por otra parte, el rango de los datos que contiene está delimitado en un intervalo diferente que el rango de los datos que podría contener el *dataset* LiDAR.

Por otra parte, el *dataset* LiDAR está principalmente almacenado en ficheros de tipo *laz/las*, necesitándose instrucciones especiales para su carga. Así, la carga de las nubes es un ejemplo de método abstracto que necesitará implementarse obligatoriamente en cada clase que hereda de la superclase.

Sin embargo, una vez se leen las nubes de puntos, con independencia de su procedencia, se almacenan en un formato *NumPy* estándar, por lo que el resto de métodos de troceado de la nube, generación de *batches* de entrenamiento, etcétera, se implementarán en la clase padre.

2.2.1.3 Conclusiones

Tras definir el proyecto y realizar la carga de ficheros necesaria, el siguiente paso consiste en comenzar a preparar el *dataset* de entrenamiento, empleando las

normalizaciones de datos necesarias, así como realizando el troceado de la nube en cuestión.

2.2.2 Segunda Iteración

Tomando como entrada las conclusiones extraídas de la iteración anterior, es necesario comenzar a preparar el *dataset* de entrenamiento de forma que posteriormente pueda servirse como entrada a la red neuronal.

2.2.2.1 Normalización de datos

Tal y como se ha comentado antes, se van a considerar dos tipos de *dataset* en este sistema: Semantic3D y LiDAR. Como ya es conocido, de cara a optimizar la fase entrenamiento, resulta adecuado normalizar los canales de información de entrada utilizando algún método de normalización conocido (max-min, decimal, estandarización, etcétera). En nuestro caso, se utilizará la normalización max-min para llevar los canales RGB al intervalo [0, 1].

En el caso de Semantic3D, al igual que en LiDAR, estos canales se codifican en el intervalo [0, 255]. Así, para normalizarlos, basta con dividir el valor actual entre 255.0. Por otra parte, el canal de intensidad, que representa el valor de la reflectividad del material escaneado, no se encuentra tipificado en el caso del *dataset* Semantic3D, por lo que no es posible normalizarlo de una forma eficiente, así que se mantendrá en su escala original.

En una primera aproximación, cada uno de los puntos con los que se alimentará la red contendrá 10 atributos de entrada: las coordenadas (x, y, z), el valor de intensidad i, los canales normalizados (r, g, b) y por último, las coordenadas (x_n, y_n, z_n), que estarán tipificadas en el intervalo [0, 1] y corresponden a las coordenadas que tiene un punto con respecto al bloque al que pertenece. A continuación, se realizará la aclaración del concepto de bloque.

2.2.2.2 Generación de bloques/*batches* de entrenamiento

Con el objeto de regularizar la entrada de la red, de forma que todos los ítems de información contengan el mismo número de puntos, se sigue la estrategia de generación de bloques definida en la mayoría de trabajos sobre segmentación de nubes de puntos con base en aprendizaje profundo.

Esta estrategia consiste en dividir la nube de puntos en regiones de igual área, sobre los ejes X y Z utilizando el sistema de coordenadas de mano derecha (la coordenada Y es el eje vertical). En tanto, se seguirá un enfoque de ventana deslizante que irá extrayendo puntos de forma sucesiva de las diferentes subregiones de la nube.

Se considera el parámetro *stride* que define el avance de la ventana en cada iteración. Si el *stride* es igual al tamaño de la ventana, entonces se generarán regiones no solapantes, como se observa en la Figura 2.2:

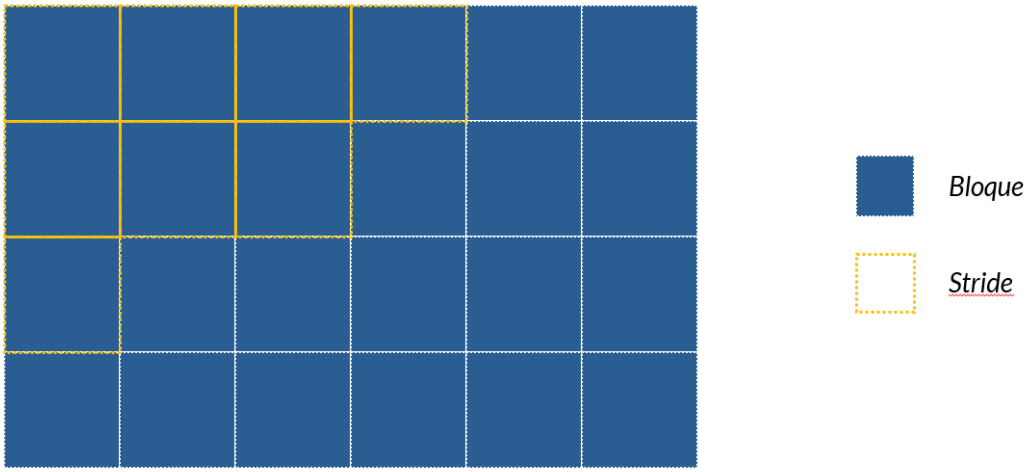

Figura 2.2: Generación de bloques con stride=tam_bloque. Fuente: elaboración propia.

En cambio, si el *stride* es menor al tamaño de la ventana deslizante, se generarán bloques cuyos puntos pueden estar en más de un bloque a la vez, algo de vital importancia para el aumento de datos y el incremento de capacidad de generalización de la red, como se aprecia en la Figura 2.3:

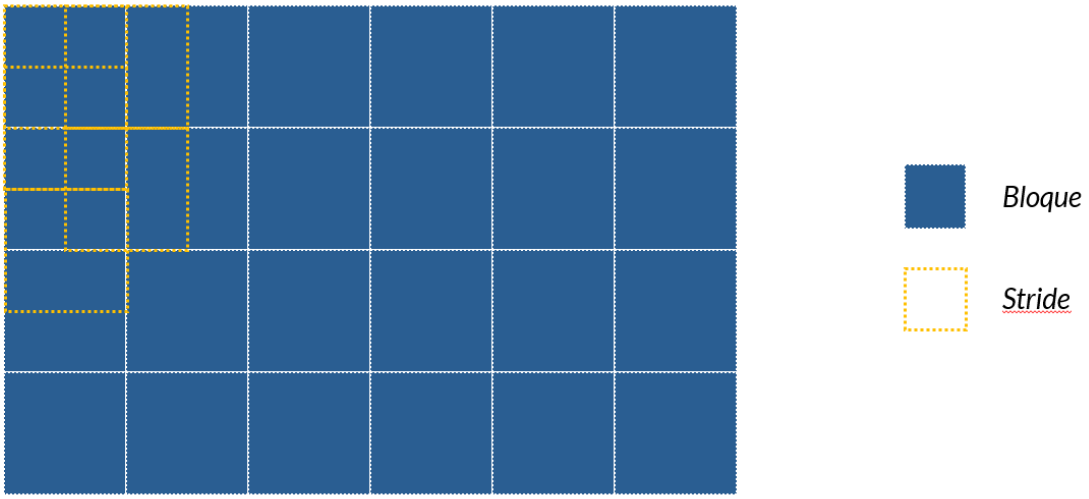

Figura 2.3: Generación de bloques con stride=$\frac{tam\_bloque}{2}$. Fuente: elaboración propia.

Por otra parte, es necesario especificar el número de puntos a extraer de cada bloque. Se generarán así muestras de entrenamiento, de forma que cada bloque contendrá el mismo número de puntos, lo que facilitará el aprendizaje. En tal proceso, pueden ocurrir varias casuísticas:

- **Hay más puntos que la cantidad solicitada.** En tal caso, la solución es sencilla puesto que se trata únicamente de realizar un muestreo sin reemplazamiento sobre los puntos de esa región, extrayendo la cantidad solicitada.
- **Hay menos puntos que la cantidad solicitada.** En esta situación, realizar el paso anterior cambiando el muestreo sin reemplazamiento por un muestreo con reemplazamiento, de forma que los puntos existentes en esa región comiencen a replicarse hasta obtener la cantidad deseada. Esto no afectará al tamaño de la red. Se define un umbral de límite por encima del cuál una región se considera apta. Si el número de puntos en una región no supera este umbral, se descarta esa región y se continúa con la siguiente.
- **No hay puntos en una región.** En este caso, se descarta este bloque/región y se continúa con los siguientes.

Una vez que todas las subregiones de una nube se han analizado, y se han generado los bloques con igual número de puntos, se añaden a una lista donde están contenidos los bloques generados en otras nubes. Cuando todas las nubes son procesadas, se tratan todos los bloques como si perteneciesen a una sola nube, y se reordena esta lista antes de preparar el conjunto de entrenamiento.

2.2.2.3 Conclusiones

El desarrollo en lo relativo a la preparación de los datos de entrenamiento casi está terminado, pero es imposible verificar de forma visual si el proceso se está realizando correctamente. Sería necesario algún tipo de herramienta que permita visualizar cada una de las nubes, y verificar que la carga y extracción de bloques se está realizando correctamente.

Por otra parte, el vector con los bloques generados se encuentra listo para su almacenamiento y posterior carga.

2.2.3 Tercera Iteración

Una vez llevado el desarrollo hasta este punto, se han implementado funciones para realizar la lectura de nubes de puntos, así como su muestreo y posterior troceado para la generación de *batches* de entrenamiento. Sin embargo, hasta este momento no se ha realizado ningún tipo de visualización tridimensional que permita verificar que el formato de las nubes es válido, al igual que el troceado de la misma.

2.2.3.1 Visualización de nubes de puntos

Para realizar la visualización de nubes de puntos desde la estructura algorítmica ya implementada, es necesario integrar algún tipo de librería en Python que disponga de funcionalidad adecuada para visualizar. En nuestro caso, se ha utilizado la librería ***Open3D***, disponible en varios lenguajes de programación.

En el caso del lenguaje Python, esta librería se integra a la perfección con la librería NumPy, y permite visualizar geometría que esté previamente en formato de *array numpy*. Así, se implementa funcionalidad tanto para visualizar nubes de puntos de forma completa como para visualizar trozos (bloques) de la nube, como los que se obtienen cuando se está generando el *dataset* de entrenamiento.

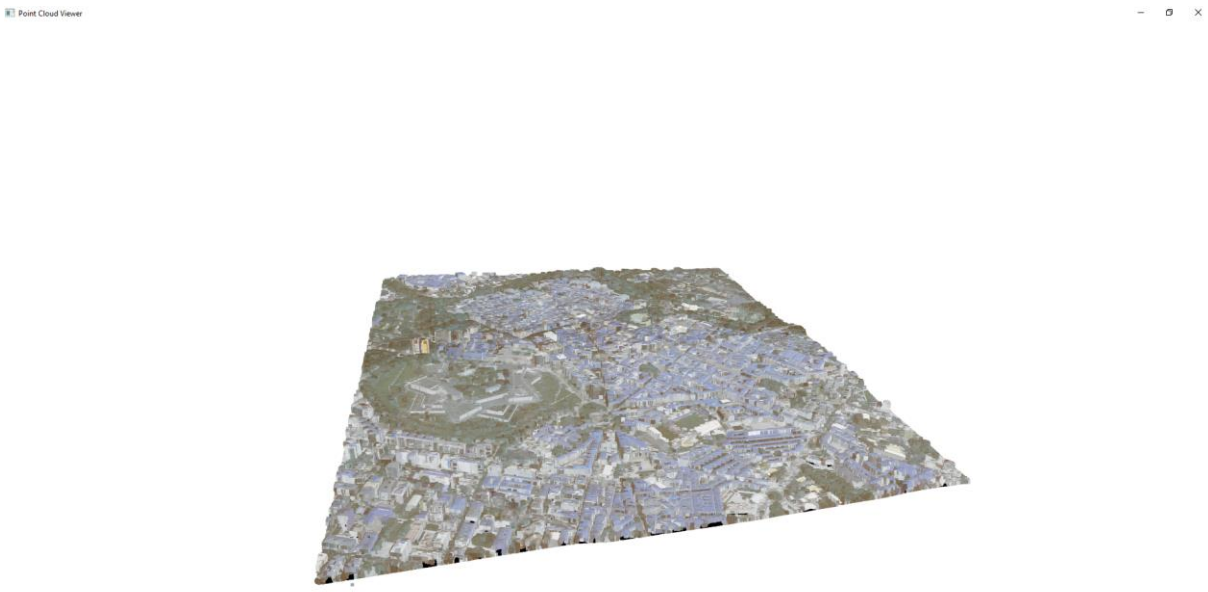

Figura 2.4: Nube de puntos visualizada con Open3D.

Realmente, el propósito de la visualización no es otro que el de la satisfacción visual que produce observar cómo se resuelve un proceso puesto que, en este proyecto orientado a investigación, resultan de interés otros parámetros como la precisión y ajuste del sistema entrenado.

2.2.3.2 Script de preparación de datos

Tras completar la visualización de las nubes de puntos, no necesaria, aunque sí útil, se procede a la implementación de un *script* principal para la preparación de datos de entrenamiento, siendo este *script* el que se invocará y ejecutará cuando el usuario decida preparar una serie de ficheros de entrenamiento.

El propósito en el desarrollo de este proyecto es que el software sea tan versátil como sea posible. Esto es, el usuario pueda configurar tantos parámetros cuantos determine, y realizar esto de una forma rápida y sencilla. Por ejemplo, pueda decidir en cada llamada al *script* de generación de datos de entrenamiento si quiere generar bloques que contengan 4096 puntos de entrenamiento, o bien que contengan 1024.

Para conseguir esto, se ha utilizado la librería ***argparse*** disponible en Python. Cuando se ejecuta un proyecto Python utilizando la consola, esta librería se encarga de recoger todos los parámetros que se pasan tras llamar al *script* y los introduce en una especie de **diccionario** Python, de forma que pueden consultarse los valores de

los argumentos desde el código. Además, la librería permite asignar valores por defecto (en el caso de que no se introduzca un valor para un parámetro en concreto desde la línea de órdenes), a la vez que permite indicar posibles valores, el tipo de datos, si un parámetro es obligatorio, etcétera.

En nuestro caso, se han utilizado los siguientes parámetros:

Argumento	Valor por defecto	Descripción
--download	*false*	Cuando se trate del *dataset* Semantic3D, descargar el conjunto de datos completo del sitio web oficial usando código.
--dataset	Semantic3D	El conjunto de datos a utilizar para preparar los ficheros de entrenamiento. Es necesario que los ficheros estén localizados en las rutas especificadas en el código entregado. De lo contrario, obtendremos un fallo.
--redirect_log	*None*	Si se especifica una ruta en este parámetro, todas las impresiones a consola se dirigirán al fichero especificado, reemplazando la salida estándar por una salida a fichero.
--subsample	False	Si este parámetro es *true*, las nubes de puntos originales se muestrearán, pasando a una porción lo suficientemente representativa pero menor en volumen de información.
--kfold	5	Valor de K utilizado para generar las particiones de la validación cruzada. Por defecto, se crearán 5 particiones que se utilizarán iterativamente en un juego disjunto de 4 a 1: 4 para entrenar, una para validar.
--split_size	4.0	Tamaño de las subdivisiones que se crearán en la nube de puntos para generar los bloques de entrenamiento. Este parámetro se especifica en metros, y se realizarán divisiones cuadradas. Por tanto, el área seleccionada en cada momento tendrá un área de $splitsize^2\ m^2$. No se realizan divisiones en el eje vertical.

--num_points	4096	Número de puntos que contendrá cada uno de los bloques generados durante el entrenamiento. Siguiendo la lógica indicada en párrafos anteriores, habrá varios comportamientos según el número de puntos real que contenga una subdivisión de la nube. Finalmente, la red se entrenará con bloques que contengan exactamente esta cantidad de puntos.
--min_points	1024	Se trata de la mínima cantidad de puntos en una subdivisión necesaria para generar un bloque. Partiendo del valor por defecto, si se tienen, por ejemplo, 512 puntos, se ignorará esta región de la nube. Si por el contrario se tienen 1536 puntos, se utilizará el muestreo con reemplazamiento hasta obtener un bloque con tantos puntos como se hayan indicado en *num_points*.
--resample	False	Si este parámetro es *true*, las nubes volverán a muestrearse aunque ya existan ficheros que contienen las nubes muestreadas, reemplazando este contenido. Por el contrario, se utilizarán los ficheros con las nubes muestreadas en ejecuciones previas para generar nuevos ficheros de entrenamiento.

--stride	1.0	Desplazamiento de la ventana deslizante a través de la nube. Este parámetro se mide en metros, al igual que el tamaño de la división. Si el valor de este parámetro coincide con el valor del tamaño de división, se tendrán regiones disjuntas no solapantes. De otro modo, las regiones se solaparán, y se generarán ficheros de entrenamiento más abundantes en cuanto a información. Este parámetro debe ser menor que *split_size*, y a la vez *split_size* debe ser divisible por *stride*. Esto es, $splitsize \ mod \ stride = 0$
--output_dir	*train_items*	Directorio donde se almacenarán los ficheros de entrenamiento generados. Por defecto, será dentro del directorio *data/<dataset>/train/train_items.*

Tabla 12: Parámetros de configuración para la generación de los ficheros de entrenamiento.

2.2.3.3 Conclusiones

Así, una vez desarrollado el *script* de generación de ficheros de entrenamiento, y añadida la funcionalidad para configurar la ejecución de forma que puedan obtenerse conclusiones científicas de una forma más eficiente, es necesario concluir la generación de los ficheros de entrenamiento, pues en la segunda iteración se completó la generación de bloques resultando en una lista sin orden de bloques con puntos listos para entrenar.

2.2.4 Cuarta Iteración

Tras implementar un *script* para generar ficheros de entrenamiento, aún es necesario desarrollar el software necesario para escribir en disco los ficheros de entrenamiento en sí, y dotar de persistencia al proceso de generación ejecutado. Se utilizará un enfoque de validación cruzada durante el entrenamiento, para validar la arquitectura y ajustar los hiperparámetros de la red convolucional.

2.2.4.1 Validación cruzada

Para conseguir un esquema de validación cruzada durante el entrenamiento, es necesario tomar esta decisión desde la base del problema y estructurar los ficheros de tal forma que después sean más sencillos de tratar durante el entrenamiento.

Así, tomando como entrada la lista de bloques o *batches*, donde cada bloque contiene el mismo número de puntos con sus diferentes atributos y la etiqueta asignada del problema, se tiene un conjunto de elementos que se dividirá exactamente en K partes.

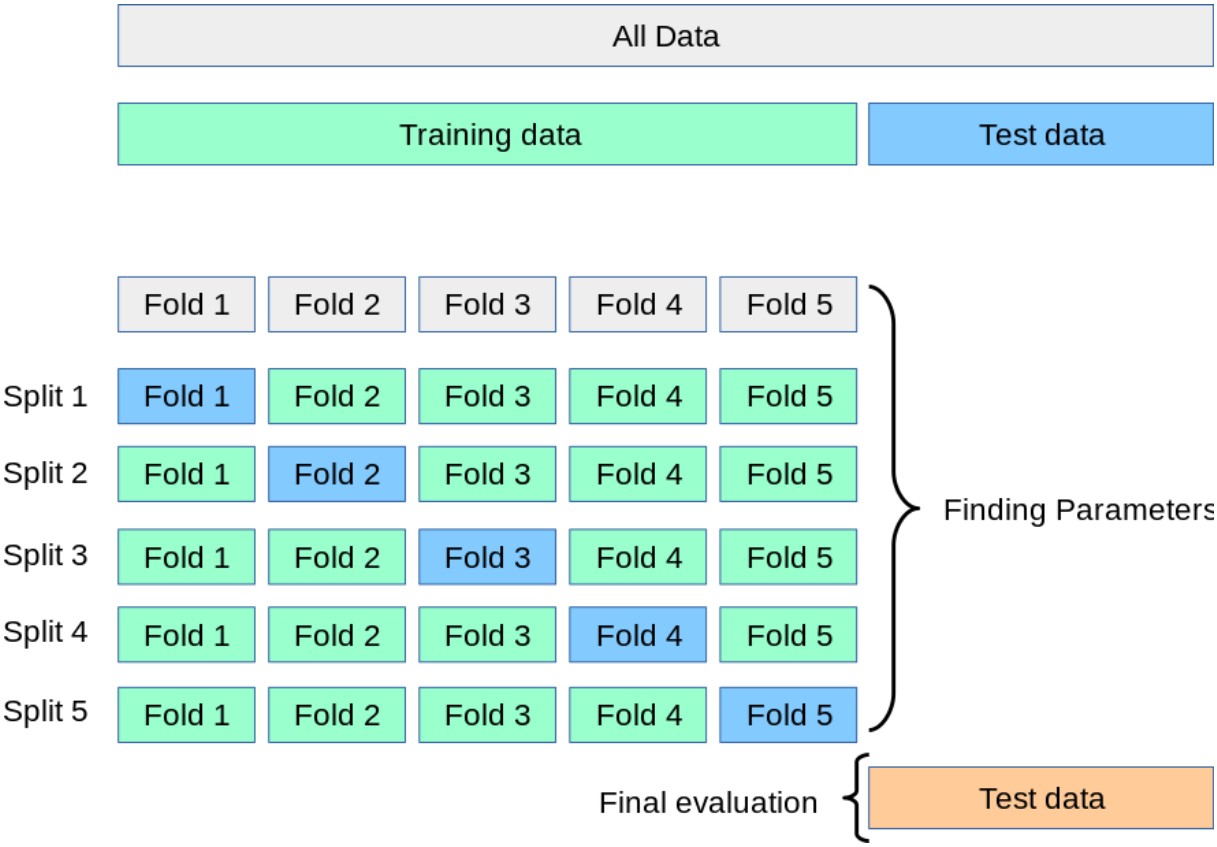

Figura 2.5: Esquema de validación cruzada K-fold. Fuente: documentación sklearn.

Dada la Figura 2.5, nuestra lista de bloques se correspondería con el elemento *All Data*, pues contiene el conjunto completo de bloques se utilizarán para entrenar/validar. Así, este conjunto se divide en K partes disjuntas, tantas como posteriores ciclos de entrenamiento/validación se completarán. Tal y como se ilustra en la imagen anterior, en cada ciclo se utilizarán K-1 partes como entrenamiento, y una parte para validar.

Para optimizar nuestro proceso de entrenamiento y validación, se realiza en proceso de división de la lista original utilizando la librería ***Sklearn*** de Python, y se almacenan exactamente $2 * K$ ficheros, K para entrenamiento y K para validación. Cada par de ficheros entrenamiento/validación contiene un fichero de entrenamiento que dispone de todos los bloques correspondientes a $K - 1$ partes, y el fichero de validación contiene la parte restante.

Hay tantos pares de ficheros como sea el valor de K, y se almacenan con la nomenclatura (para un valor de $K = 5$, por ejemplo):

K	Fichero de entrenamiento	Fichero de validación
0	*train_0.hdf5*	*validation_0.hdf5*
1	*train_1.hdf5*	*validation_1.hdf5*
2	*train_2.hdf5*	*validation_2.hdf5*
3	*train_3.hdf5*	*validation_3.hdf5*
4	*train_4.hdf5*	*validation_4.hdf5*

Tabla 13: Nomenclatura de los ficheros de entrenamiento generados.

2.2.4.2 Almacenamiento de ficheros

Tal y como podrá inferirse de la información en Tabla 13, se utiliza el formato de almacenamiento HDF5, un formato de archivos jerárquico que permite almacenar grandes volúmenes de información de manera comprimida, ahorrando una gran cantidad de espacio de almacenamiento. Este formato tiene su propia en Python para realizar labores de carga y almacenamiento de ficheros.

Permite almacenar la información utilizando una estructura de *ndarray*, esto es, el formato utilizando por la librería NumPy. Así, como cada bloque se codifica como un *ndarray* de dimensiones (n, f), siendo n el número de puntos y f el número de características de cada punto, este formato de almacenamiento resulta idóneo.

Es necesario crear un *dataset*, según la propia nomenclatura de la librería. Un fichero HDF5 puede almacenar más de un *dataset* cada uno con su propio nombre, si bien es necesario indicar el tamaño de cada *dataset* a priori, así como el formato de los datos que van a almacenarse (*int, float, str*, etcétera).

Se permite añadir un parámetro de compresión durante el almacenamiento, de forma que los ficheros resultantes ocupen un volumen de información mucho menor que el que ocuparían de no utilizar la compresión, aunque esta librería aplica por defecto un parámetro de compresión. Este parámetro es ajustable, y a mayor valor, mayor es la compresión, si bien la complejidad de la tarea será directamente proporcional a la magnitud de la compresión deseada.

2.2.4.3 Conclusiones

Habiendo implementado todo el proceso para el preprocesamiento de datos, se concluye el paso previo a la construcción del modelo neuronal y posterior entrenamiento. Así, al menos hasta ser necesario, se concluye por ahora la fase de preprocesamiento de datos.

2.2.5 Quinta Iteración

De igual forma que se realizó durante la fase de preprocesamiento de datos, modularizando diferentes funciones y clases, y finalmente generando un *script* principal que invocase a todos estos métodos y ejecutase el comportamiento deseado. Así, se crea un nuevo fichero Python llamado *train_network.py* que permitirá establecer una configuración en base a una serie de argumentos, y entrenar así la red convolucional del sistema.

2.2.5.1 Carga de ficheros de entrenamiento.

En el *script* de entrenamiento se realizará el entrenamiento propio de la red. Además, se recogerán las diferentes métricas de precisión y coste, tanto promedio como por cada clase. Dada la validación cruzada que se realizará durante el entrenamiento, deberá iterarse sobre las diferentes particiones realizadas, e ir cargándose sucesivamente. Para cada partición, se realizará un proceso de entrenamiento/validación completo, ejecutando tantas *epochs* como se hayan configurado a priori.

Así, para cargar los datos de forma secuencial, se implementa un nuevo método que lee de disco las particiones de ficheros de forma iterativa, y las **cede** al hilo principal utilizando la sentencia ***yield***. Esto permite iterar sobre un método, y cada vez que se ejecute la sentencia ***yield***, el método devolverá unos datos que se

utilizarán dentro del bucle. En este caso, los datos que se devolverán serán los pares de entrenamiento/validación. Normalmente, el objeto que contiene los datos de entrenamiento será significativamente más voluminoso en cuanto a datos que el objeto que contiene los datos de validación.

2.2.5.2 Script de entrenamiento

Una vez resuelta la carga de los ficheros de entrenamiento y validación, es necesario estructurar el *script* de entrenamiento. En primer lugar, se leerá la configuración especificada por el usuario mediante argumentos, utilizando para ello la librería Python ***argparse***.

Los parámetros que se establecen, al igual que en el caso del preprocesamiento de datos, son:

Argumento	Valor por defecto	Descripción
--exp_name	*exp*	Nombre del experimento realizado. Se considera experimento a cada invocación de este *script*, pues en cada invocación se completará un proceso completo de entrenamiento/validación utilizando validación cruzada. El nombre del experimento se utilizara posteriormente para almacenar los ficheros de log y modelos ya entrenados, de forma que pueda realizarse transferencia de conocimiento.
--dataset	*Semantic3D*	*Dataset* que se utilizará para realizar el entrenamiento. Indicando este parámetro, se extraerán automáticamente el número de clases del problema, utilizado para configurar la arquitectura de red neuronal (la última capa), así como la ruta de los ficheros de entrenamiento. No obstante, será necesario indicar el nombre de los datos de entrenamiento.

--input_data	*train_items*	Se trata del nombre que recibe el directorio que alberga los datos de entrenamiento. En relación a la Tabla 12, este argumento se relaciona con *output_dir*. Por defecto, los datos de entrenamiento se leerán del directorio *./data/Semantic3D/train/train_items/*, a menos que se especifique lo contrario. Esto nos permitirá tener varias colecciones de datos de forma disjunta, sin interferir unas sobre otras.
-learning_rate	*0.001*	Ratio de aprendizaje utilizado en el experimento. Por defecto, este argumento toma un valor de 0.001, pues se encuentra entre lo recomendado por la comunidad científica, tratándose este intervalo de $[10^{-4}, 10^{-2}]$.
--scheduler	*True*	Si el valor de este argumento es *True*, entonces se utilizará un *scheduler* para ajustar el ratio de aprendizaje tras cada *epoch*. Cuando finaliza un *epoch*, si *scheduler*=*True*, el ratio de aprendizaje se reducirá automáticamente al 95% de su valor original. En cambio, de ser *False*, no se utilizará *scheduler* y el ratio de aprendizaje se mantendrá durante todos los *epochs* de ejecución.

--epochs	*100*	*Epochs* de entrenamiento de la red. Dado que se utiliza validación cruzada, en cada una de las particiones se completarán tantas *epochs* como se haya especificado en este argumento. Así, si el total de *epochs* es 100, y $K = 5$ en validación cruzada, el número total de *epochs* que se ejecutará en el experimento será de $100 * 5 = 500\ epochs$.
--nearest_neighbors	*15*	Vecinos más cercanos que se considerarán en la construcción del grafo dinámico de características, característica propia de este modelo de red neuronal. Por defecto, se utilizan 15 vecinos de cada punto para construir el grafo de características, tal y como establece el *paper* original de (Wang et al., 2018). A mayor número de vecinos utilizado, peor será el rendimiento del sistema puesto que el cálculo de vecinos más cercanos es un paso complejo. Así, se hace necesario encontrar una cantidad de vecinos, k, que sea un compromiso entre la maximización de información conseguida, y la minimización de la pérdida de rendimiento computacional.

--momentum	*0.9*	Cuando se utiliza el algoritmo de optimización SGD (*Stochastic Gradient Descent*), el parámetro *momentum* controla la velocidad de convergencia del modelo al óptimo. *Grosso modo*, este parámetro dota de cierta inercia durante el descenso de gradiente, y evita en cierta medida que el algoritmo de optimización se estanque en óptimos locales. Por defecto, y en la mayoría de los casos, se utiliza el valor de 0.9 según las recomendaciones de la literatura científica.
--output_models	*checkpoints*	Directorio donde se almacenarán los modelos de red neuronal entrenados. Se almacenará un fichero de pesos entrenados por cada *epoch* en formato *.pt*. Esto se realizará utilizando el método *state_dicts*() de la clase *nn.Module* en PyTorch, el *framework* de aprendizaje profundo utilizado.
--optimizer	*SGD*	Algoritmo de optimización utilizado durante el entrenamiento de la red. Por defecto se utilizar el SGD, Gradiente Descendiente Estocástico, aunque también es posible utilizar Adam (Goodfellow et al., 2016).
--bias	*False*	Este argumento controla la utilización de capas de sesgo en cada una de las capas de la red (que contienen parámetros entrenables). Por defecto, no se utilizan capas de sesgo en esta arquitectura.

--emb_dims	*1024*	Tamaño del vector de características global que se obtiene tras la extracción de características geométricas. Este vector de características global se concatenará con el vector de características local de cada punto, enriqueciendo la información que acompaña a cada punto en el paso previo a la clasificación/segmentación.
--dropout	*False*	Uso de capas de *dropout* tras las capas *fully-connected* de la red neuronal. El uso de capas *dropout* evitará la aparición de sobreajuste, o disminuirá su efecto en cierto modo.
--workers	*8*	Número de hilos trabajadores que se utilizarán durante la carga de datos en el entrenamiento. Cada uno de los hilos llamará concurrentemente al proceso de carga, y cargará un *batch* de datos.
--batch_size	*2*	Número de ítems de entrenamiento que se procesarán a la vez.

--features	*x y z i r g b xn yn zn*	Características de cada punto que se utilizarán durante el entrenamiento. Cada carácter hace referencia a una característica distinta sobre los datos de entrada: • ***x.*** Coordenada X del punto. • ***y.*** Coordenada Y del punto. • ***z.*** Coordenada Z del punto. • ***i.*** Valor de la intensidad para el punto. • ***r.*** Valor del canal R (rojo) del punto. • ***g.*** Valor del canal G (verde) del punto. • ***b.*** Valor del canal B (azul) del punto. • ***xn.*** Coordenada X normalizada según las dimensiones del bloque. • ***yn.*** Coordenada Y normalizada según las dimensiones del bloque. • ***zn.*** Coordenada Z normalizada según las dimensiones del bloque. Las características a utilizar deberán entrecomillarse, y dejar un espacio en blanco entre característica y característica.
--num_features	*10*	Número de características a utilizar durante el entrenamiento. Aunque este parámetro podría inferirse sobre lo anterior, es más sencillo indicar como un nuevo parámetro.

--num_points	*2048*	Cantidad a puntos a procesar de cada bloque. Se sigue una lógica similar a la utilizada en la preparación de datos. Esto es, si el número solicitado es mayor, se vuelven a muestrear los puntos del bloque sin reemplazamiento. Si por el contrario la cantidad deseada es menor que los puntos que ya están contenidos en el bloque, se realiza un muestreo sin reemplazamiento sobre esta cantidad.
--pooling	*max*	Tipo de *pooling* utilizado en las capas de reducción que siguen a las convolucionales. Por defecto se utiliza la reducción *max (max pooling*), aunque se considera de interés implementar *Attentive Pooling* para evaluar su influencia.

Tabla 14: Parámetros de configuración para el entrenamiento de la red neuronal.

2.2.5.3 Clases para *logging*

Una vez completado el *script* de entrenamiento, en su estructura básica de iterar sobre las particiones *kFold* y en cada *epoch*, es necesario monitorizar el rendimiento de la red para poder obtener conclusiones al respecto. Para ello, se utilizará la integración del *framework PyTorch* con *Tensorboard*, un módulo de análisis de resultados embebido en *TensorFlow*, el *framework* de aprendizaje profundo desarrollado por Google.

Esta herramienta permite registrar parámetros como la precisión y coste en cada epoch y cada partición, así como etiquetas de texto de propósito general, imágenes, etcétera. También es posible insertar la arquitectura de red y analizar la evolución de los pesos, pero no se hará uso de esta característica al no considerarse de interés.

2.2.5.4 Conclusiones

Tras preparar el entorno de entrenamiento de la red, es necesario comenzar a desarrollar la arquitectura neuronal utilizando el *framework PyTorch*, para finalmente instanciar dicho modelo y comenzar a realizar propagaciones hacia adelante y hacia atrás entrenando y evaluando resultados.

También es necesario seleccionar una métrica de coste, aunque esto se realizará en la siguiente iteración. La métrica de coste nos aporta una medida del error cometido por la red al realizar una predicción. Los tipos de métricas varían según el problema, pues recuérdese que habitualmente se resuelven problemas de regresión o clasificación utilizando aprendizaje profundo, si bien en los últimos años se ha popularizado el uso de estos modelos algorítmicos sobre otros problemas de índole no supervisado.

2.2.6 Sexta Iteración

El *framework PyTorch*, desarrollado y propuesto por Facebook AI, el departamento de investigación de Facebook, se ha popularizado en los últimos años en toda la comunidad científica que trabaja con aplicaciones de aprendizaje profundo. Consta de una sencillez de configuración muy superior a otros *frameworks* de la competencia como por ejemplo *TensorFlow* o *Keras*, que necesitan de una pesada instalación para funcionar correctamente.

Es necesario utilizar CUDA, la tecnología de aceleración hardware de NVIDIA, para poder ejecutar los algoritmos contenidos en este *framework* de forma eficiente pues el peso computacional es relativamente alto.

A diferencia de las primeras versiones de *TensorFlow* (1.0+), PyTorch ha utilizado desde el principio el paradigma de la orientación a objetos para implementar toda la funcionalidad requerida en una aplicación de aprendizaje profundo, tanto la carga de datos como el propio modelo de red neuronal. Cualquier modelo de red neuronal que quiera implementarse ha de heredar de la clase *nn.Module*, e implementar dos métodos básicos:

- ***__init__().*** Se trata del constructor de la clase. En este método han de definirse todas las capas de la red, así como el número de unidades en

cada capa, y todos los elementos de procesamiento que intervendrán en la propagación hacia delante de la red neuronal.

- ***forward(x).*** Método que implementa la propagación hacia adelante como tal. Este método recibe un único argumento, x, que se corresponde con un dato de entrada (un ejemplo del *dataset*), y es el encargado de realizar las operaciones necesarias sobre este dato, propagándolo de principio a fin por la arquitectura de la red, hasta obtener la salida deseada, que habitualmente será el resultado de la clasificación o regresión, según el problema.

Así, se define una nueva clase llamada *DGCNN* que hereda de la superclase *nn.Module*, y que se corresponde con el modelo de red neuronal definido en este trabajo. En el constructor se definen todas las capas de la red, así como los bloques *EdgeConv.* Es necesario implementar funcionalidad fuera de este modelo de red, como por ejemplo la función para calcular el grafo de características.

2.2.6.1 Construcción del grafo dinámico de características

Esta funcionalidad se implementa de forma separada a la red neuronal. Principalmente, la misión de esta función será la de construir un grafo no dirigido a partir de los vecinos más cercanos de cada punto. Esto es, dada una matriz de tamaño (n, f) como entrada, donde n representa el número de puntos y f las características de cada punto, esta función calculará los vecinos más cercanos utilizando para ello la distancia euclídea.

Obtenidos los identificadores de los vecinos más cercanos, el siguiente paso consiste en extraer las características de los vecinos y realizar una agregación de cada punto con respecto a las características de sus vecinos.

De esta forma, cada punto estará representado por sus vecinos, pasando a tener cada uno de ellos el doble de características: diferencia entre sus características propias y las del punto original, para describir el entorno local y, por otra parte, las características originales de cada punto.

El grafo se construirá varias veces durante la propagación hacia adelante en la red, partiendo de la base sobre la que se construye este trabajo, y la idea de que

puntos lejanos en el espacio euclídeo podrían estar cerca en el espacio de características, en función de si los puntos comparten características.

2.2.6.2 Arquitectura neuronal

Haciendo referencia a la Figura 2.1, la red puede dividirse en dos módulos básicos, siendo estos el módulo de extracción de características, y el módulo de clasificación (segmentación). Se hace referencia a clasificación como concepto propio del aprendizaje automático, teniendo en cuenta lo establecido en el capítulo introductorio sobre la diferencia entre clasificación y segmentación semántica de nubes de puntos.

Así, la arquitectura de la red se divide en:

- **Obtención de características locales.** El primer módulo de la red se encarga de extraer características de índole espacial, recibiendo como entrada una matriz de dimensiones (n, f). Se aplican 3 bloques *EdgeConv* de forma sucesiva, a la vez que se concatenan en el último paso las salidas de sendos bloques para obtener un vector de características global mediante la aplicación de un perceptrón multicapa compartido entre todos los puntos.

  En los dos primeros bloques *EdgeConv* que se aplican sobre la matriz de puntos de entrada, se aplican de forma reiterada dos capas perceptrón multicapa. El objetivo de todas las capas de la red que contienen parámetros entrenables (optimizables durante el entrenamiento) es el de extraer características.

  Un modelo perceptrón, por defecto, no dispone de capacidad para procesar varios ítems de entrada a la vez, pues esta es una característica idiosincrásica de las capas de convolución. Por ello, se aplican capas de convolución 1 dimensional, utilizando un tamaño de $kernel = 1$, y de esta forma aplicará en el sentido estricto un perceptrón cuyos pesos serán compartidos para todos los puntos procesados.

  Sin embargo, existe una particularidad en este modelo. Cuando se ejecuta el bloque *EdgeConv*, se toma como entrada una matriz (n, f) y, tras realizar el cálculo del grafo de características (teniendo en cuenta a

los vecinos más cercanos) se obtiene una matriz de orden $(n, k, 2f)$. Las capas de convolución 1-dimensional están destinadas al procesamiento de matrices de orden 2, por tanto, no son idóneas para su aplicación en este caso.

Así, se considera la matriz obtenida como una imagen, correspondiéndose los canales de la imagen con las características de los puntos en este caso. Por ello, se utilizan capas de convolución 2D, idénticas a las aplicadas en el caso de las imágenes.

Para mantener la máxima de procesar únicamente un ítem de información a la vez, se configura un tamaño de $kernel = (1, 1)$. Así, en cada iteración de la convolución procesará una matriz $(1, k, 2f)$ con respecto a la original tomada como entrada, procesando a la vez un único punto y sus vecinos.

Tras aplicar los tres bloques de convolución, ya citados, se extrae un vector de características global que describirá el conjunto de puntos introducidos en la red.

- **Obtención del vector de características global.** La salida al primer módulo de extracción de características es una matriz $(n, 64)$, que se concatenará con las salidas de los dos primeros bloques *EdgeConv*, obteniéndose una matriz de características $(n, 64 * 3) = (n, 192)$. Sobre esta matriz, se aplicará de nuevo una capa de convolución (en este caso convolución 1-dimensional, pues la dimensión de la matriz es 2) seguido de una capa de reducción. Así, se obtendrá un único vector global de características. Por defecto, se utiliza un tamaño de 1024.

  Cada una de las posiciones de este vector describe en cierto modo a la nube de entrada, pero a priori es impracticable conocer el significado de estas características dada la naturaleza de caja negra inherente a los modelos neuronales.

  En tanto, la obtención de este vector completa el ciclo de extracción de características, pues se dispone tanto de información local a los puntos (sobre sus puntos más cercanos, y los puntos más cercanos a sus puntos más cercanos hasta en 3 niveles), a la vez que se tiene un vector

que describe de forma global el conjunto completo de puntos y que a priori tiene una dimensión de 1024 características.

- **Proceso de clasificación (segmentación semántica).** Tras la fase de extracción de características, es necesario aplicar capas de tipo perceptrón de nuevo para realizar predicciones y asignar una etiqueta semántica a cada uno de los puntos contenidos en el bloque de entrada.

  Así, la matriz $(n, 192)$ obtenida en el paso previo a la obtención del vector global de características, que contiene información local a los puntos, se concatena con el vector global propio. De este modo, la matriz pasa a tener una dimensión final de $(n, 1216)$.

  Una vez obtenida la matriz enriquecida, procede la aplicación de capas perceptrón multicapa compartido, implementadas mediante capas de convolución 1-dimensional. Se aplica un total de 3 capas. Tras la primera, la matriz reduce su dimensión hasta $(n, 512)$, reduciéndose tras la segunda hasta $(n, 256)$ y por último a (n, p), siendo p el número de clases del problema de segmentación.

  Recordando el capítulo introductorio de aprendizaje profundo, cada uno de los vectores p-dimensionales asociados a los n puntos contiene las probabilidades asignadas a cada clase. Cada punto recibe la etiqueta de la clase más probable.

Así, una vez descritos los dos módulos fundamentales en los que se divide la arquitectura neuronal propuesta, se concluye la implementación de la misma. La definición de las capas de convolución que aplican se realiza en el método constructor de la clase *DGCNN* definida, mientras que el proceso completo de propagación hacia adelante descrito inmediatamente antes se realiza en el método *forward()*.

De forma adicional, se configuran capas de *Dropout* y *BatchNorm* para aplicarlas en el segundo módulo de la arquitectura, esto es, las capas de clasificación. Estos módulos evitarán el sobreajuste y facilitarán el aprendizaje, respectivamente, y su utilización es configurable mediante los argumentos de configuración en el *script* de entrenamiento de la red neuronal. Sin embargo, en el trabajo original (Wang et al., 2018) no se menciona la utilización de esta tipología de capas en la arquitectura neuronal.

Notar, llegados a este punto, que el modelo neuronal implementado no se corresponde con una implementación réplica a la arquitectura original, puesto que el *dataset* utilizado en este caso es de una naturaleza diferente. Además, la configuración se ha realizado en base a criterios obtenidos a través de la experiencia con otras arquitecturas estudiadas, considerando en todo momento la configuración más óptima y conveniente para el modelo en cuestión.

2.2.6.3 Conclusiones

El modelo implementado utiliza por defecto el tipo de reducción *max-pooling*. Este modo de reducción goza de éxito más que contrastado en todas las aplicaciones de aprendizaje profundo. Sin embargo, la incesante actividad investigadora en búsqueda de mejores soluciones guiadas por aprendizaje profundo ha traído consigo la aparición de nuevos modos de reducción, como por ejemplo *Attentive Pooling*, que realiza una reducción adaptativa *per sé*.

2.2.7 Séptima Iteración

Retomando el hilo conductor en la iteración anterior, resta la implementación de las capas de *Attentive Pooling* que realicen una ponderación diferente de las características a reducir, de forma supervisada.

2.2.7.1 Attentive Pooling

A diferencia de los métodos de *pooling* tradicional, como el *max-pooling* o *average pooling*, el método *Attentive Pooling* realiza una reducción supervisada. Los métodos mencionados inmediatamente realizan siempre el mismo procedimiento: reciben la información y escogen el máximo o bien realizan la media aritmética.

Sin embargo, de esta manera podrían estar descartándose características no tan significativas y con carácter relevante, pues se trata de métodos no supervisados que siempre operan de la misma forma. El método *average* realiza una media aritmética, esto es, suma el valor numérico de todas las características y luego lo divide entre el total de características disponibles. Por otra parte, el método *max* selecciona el máximo, que equivale a asignar una ponderación de 1 al elemento del vector de características con el máximo valor numérico, y 0 al resto de elementos.

Sin embargo, el método *attentive pooling* realiza una **media ponderada**, encontrándose esto a medio camino entre los dos métodos anteriores. En lugar de asignar pesos fijos, aplica una propagación hacia adelante utilizando para ello redes perceptrón multicapa, de la misma forma en que estas capas se utilizan en la arquitectura original para extraer características. De esta forma, se obtiene un nuevo vector de pesos de ponderación que se aplicará sobre el vector de características inicial, y a partir del cual este se infiere.

Para cumplir con la máxima de que la suma de pesos ha de estar comprendida en el intervalo [0, 1], se aplica una capa de *Softmax*, cuya función es normalizar el vector de salida de tal forma que el sumatorio de todas sus componentes sea la unidad.

Este comportamiento se implementa heredando de la clase ***nn.Module*** de PyTorch, de igual forma que si se tratase de una nueva red. Por otra parte, un parámetro de configuración permite seleccionar si se utilizan capas de este tipo en el modelo ***DGCNN***, o en cambio se continúa utilizando capas de reducción de tipo *max-pooling*.

2.2.7.2 Conclusiones

El *software* desarrollado hasta el momento permite realizar tanto preprocesamiento de datos como el posterior entrenamiento a partir de los datos generados. Desde el punto de vista científico, y con interés en aprendizaje profundo, bastaría con esta funcionalidad para obtener conclusiones empíricas y concluir críticamente al respecto.

En cambio, el verdadero interés de este proyecto es, también, el de evaluar cómo podría realizarse una transferencia de conocimiento entre dominios diferentes, de tal forma que pueda dotarse de aplicación real a los *datasets* disponibles para evaluar modelos. Habitualmente, la utilidad este tipo de *datasets*, como por ejemplo Semantic3D (Hackel et al., 2017) o Paris-Lille 3D (CITAR), concluye cuando se obtiene una evaluación de los modelos a publicar y ya no se vuelven a utilizar.

Sin embargo, el conocimiento extraído durante el entrenamiento es muy valioso desde la perspectiva científica. A grandes rasgos, se trata de utilizar este conocimiento

para clasificar sobre otros dominios menos abundantes o que no se han contemplado durante el entrenamiento.

2.2.8 Octava Iteración

En los últimos años han surgido ingentes trabajos al respecto del concepto de ***transfer learning***. A *grosso modo*, se trata de utilizar lo aprendido de una forma, para aplicarlo de otra forma diferente. Para que este proceso sea óptimo, existen diferentes enfoques:

- **Bloquear todas las capas excepto la última**. Este método aboga por bloquear todas las capas de una red neuronal, excepto la última, pues es donde se produce la clasificación. Se considera que estas capas que se bloquean ya contienen pesos que han sido ajustados en entrenamientos previos.

  Por ejemplo, considérese un conjunto de datos orientado a clasificación de imágenes de frutas que distingue entre uvas y manzanas. A priori, ante nuevos ejemplos no vistos antes, como una naranja, predecirá la forma más probable distinguiendo únicamente entre uva o manzana. No existe una inteligencia artificial general, recordando el capítulo introductorio, por lo que la red solo hará predicciones según el dominio sobre el que se entrenó.

  Para conseguir que la red clasifique también naranjas, será necesario volver a entrenar añadiendo esta vez ejemplos de imágenes que contienen naranjas, y modificando el número de neuronas de salida para que sea 3 (en este caso, se harán predicciones de uvas, manzanas o naranjas).

  Dada la complejidad computacional inherente al entrenamiento de redes neuronales, se persigue a toda costa simplificar este proceso tanto como sea posible. Las primeras capas en una red neuronal convolucional suelen orientarse a extraer características de las imágenes aplicando para ello filtros que se entrenan junto a la completitud de la red. Si se bloquean estas capas, no se actualizarán los parámetros entrenables que contienen.

Por tanto, con este enfoque se consigue que la red tan solo aprenda a discernir un tipo más de fruta, evitando la necesidad de que la red aprenda también a extraer características. En la literatura se persiguen varios enfoques relativos al bloqueo de capas. En ocasiones se bloquea solo la última capa, y en otras ocasiones se bloquean todas las capas lineales que pertenecen al bloque de clasificación.

- **Reentrenar el modelo completo inicializando los pesos con el resultado de un entrenamiento anterior.** En multitud de ocasiones, la comunidad científica suele abogar por esta vía puesto que es la más natural en cuanto a la aplicabilidad del aprendizaje profundo.

  En lugar de partir de un modelo cuyos pesos han sido inicializados aleatoriamente, se utilizan como punto de partida unos pesos correspondientes a un modelo del mismo tipo que ya ha sido entrenado. En definitiva, la información que se recibe en las últimas capas de la red es el resultado del procesamiento realizadas en todas las capas anteriores.

  En ocasiones, resulta más adecuado volver a entrenar todas capas, aunque durante un número de *epochs* menor, puesto que los pesos en las primeras capas podrían estar en más consonancia con los pesos en las últimas capas, resultando en unas predicciones más óptimas y una mayor capacidad de generalización de la red neuronal.

Por tanto, han de hacerse algunas modificaciones para adaptar el *software* ya desarrollado de forma que puedan almacenarse modelos entrenados, y por otra parte modificar el *script* de entrenamiento para que almacene los modelos entrenados y también sea capaz de cargar modelos ya entrenados y realizar transferencia de conocimiento.

2.2.8.1 Almacenamiento de modelos entrenados

El almacenamiento de los modelos ya entrenados se realiza de una forma sencilla y automática utilizando métodos propios del *framework* PyTorch. Para adaptar esto a nuestra implementación, se define una nueva clase *NeuralHandler* que es

responsable de almacenar los pesos de las arquitecturas entrenadas en ficheros con extensión *.pt*.

Existen dos posibilidades para el almacenamiento de modelos:

- ***nn.Module.state_dict()***. Se trata de un método propio del *framework* de aprendizaje profundo que almacena únicamente las matrices de pesos, no teniendo en cuenta la implementación concreta en este tipo de *framework*.
- ***nn.Module* serializado**. Almacena la instancia de la clase que implementa el modelo neuronal, en este caso *DGCNN*, que se encuentra en memoria durante la ejecución utilizando el formato de objeto serializado. De esta forma, es posible restaurar el modelo posteriormente tal cual estaba en el momento del almacenamiento.

Aunque a priori ambas estrategias puedan parecer similares, distan bastante una de la otra en cuanto a efectos prácticos. La primera estrategia aboga por la universalidad, siendo independiente del *framework* utilizado en cuestión y permitiendo cargar los pesos desde *frameworks* diferentes siempre y cuando la arquitectura sea la misma. En cambio, la segunda estrategia obliga a que el *framework* utilizado para cargar el modelo sea el mismo que el utilizado para almacenarlo. Por otra parte, podrían aparecer problemas con las versiones del *framework.*

Por este motivo, en nuestra implementación se ha optado por utilizar el primer enfoque tanto para el almacenamiento como para la posterior carga de ficheros de pesos entrenados. Se implementan tres métodos en cuestión:

- ***store_model().*** Recibe la arquitectura entrenada en cuestión, el nombre asignado, así como el diccionario de argumentos pasados para el entrenamiento. Este método es responsable del almacenamiento del modelo neuronal entrenado en un fichero ***.pt***. El fichero de argumentos se almacena en formato YAML usando para ello la librería ***PyYaml***, y almacenará la configuración del modelo. Esta configuración permitirá reconstruir el modelo para tener la misma estructura que cuando fue almacenado.

- ***load_model()***. Realiza la operación inversa al método anterior, y recibe únicamente el nombre del modelo neuronal a cargar, devolviendo un objeto de tipo ***DGCNN***. Para ello, restaura el fichero YAML almacenado con la configuración, instancia a la clase ***DGCNN*** y devuelve un modelo idéntico al almacenado previamente inicializado con los pesos con los que fue almacenado. Tratándose, en tanto, de un modelo preentrenado.
- ***load_model_transfer_learning()***. Se trata de un método idéntico al anterior, salvo que además permite especificar el número de clases de salida del problema, permitiendo adaptar el conocimiento a extraído a diferentes dominios, y optando por la estrategia de reentrenar un modelo completo inicializando los pesos a un modelo preentrenado.

  Este último método está orientado a tareas de transferencia de conocimiento, de forma que el modelo resultante de entrenar con un *dataset* pueda utilizarse como punto de partida en la utilización de un *dataset* diferente.

Así, se concluye la metodología responsable del almacenamiento y carga de los modelos neuronales entrenados.

2.2.8.2 Modificación del *script* de entrenamiento

Puesto que la consideración al respecto de realizar entrenamientos partiendo de modelos ya entrenados es algo que se tomó desde el principio, tan solo es necesario adaptar el *script* para realizar el almacenamiento en cada *epoch*, por cada partición de la validación cruzada.

Por otra parte, para reentrenar, tan solo es necesario añadir un nuevo argumento que reciba el nombre del modelo entrenado a utilizar, así como modificar el proceso de instanciación de la clase *DGCNN*. Si el *script* se va a utilizar en modo transferencia de conocimiento (***--retrain=true)***, cargar el modelo con el nombre indicado en lugar de crear un nuevo modelo con inicialización aleatoria.

2.2.8.3 Conclusiones

Con el *software* desarrollado hasta el momento, concluye la funcionalidad mínima necesaria para realizar experimentos con *datasets* de prueba y poder concluir al respecto.

(Página intencionalmente en blanco)

3 EXPERIMENTACIÓN, RESULTADOS Y DISCUSIÓN

En esta sección, se planteará una batería de experimentos que sirva para probar la arquitectura desarrollada, así como para obtener una serie de resultados que sirvan para contrastar las hipótesis desarrolladas en la extensión de este proyecto. Para ello, se dispone de un *dataset* de nubes de puntos escaneado usando sensores LiDAR en el que para cada punto de cada nube se tiene información sobre la geometría, intensidad y color. El objetivo de la experimentación es el de comprobar la influencia de los diferentes canales de información disponibles sobre el rendimiento de la arquitectura neuronal, estableciendo combinaciones de atributos y realizando diferentes ejecuciones.

Por tanto, en primer lugar, se expondrán los conjuntos de datos a utilizar, así como los procesamientos necesarios para obtener un conjunto de entrenamiento adecuado, y en segundo lugar se anotarán los experimentos a realizar para comprobar la influencia de los diferentes canales de información sobre el rendimiento de la arquitectura de red neuronal.

3.1 Conjunto de datos

Tal y como se indicó en secciones previas, principalmente se utilizará el *dataset* Semantic3D para realizar la prueba de la arquitectura, así como la influencia de los diferentes canales de información. Por tanto, siguiendo un esquema de procesamiento como el indicado en la sección 2.1.1.2, se genera un conjunto de datos de entrenamiento y validación con la siguiente configuración:

Parámetro	Valor
--download	*false*
--dataset	Semantic3D
--redirect_log	*None*
--subsample	False
--kfold	5
--split_size	1.0
--num_points	4096
--min_points	1024
--resample	False
--stride	1.0
--output_dir	*train_items*

Tabla 15: Parámetros de generación del conjunto de entrenamiento: Semantic3D

De esta forma, se obtiene un conjunto de datos adecuado para realizar el entrenamiento y validación de la arquitectura. Cada punto está acompañado de un total de 10 características, siendo las coordenadas (x, y, z), la intensidad i, los canales de color (r, g, b) normalizados al intervalo [0, 1], así como las coordenadas (x_n, y_n, z_n) normalizadas según el bloque al que pertenece cada uno de los puntos. Además, se almacena junto a cada punto la clase a la que pertenece, siendo la clase un número entero [0, 7], que describen las 8 clases disponibles en este conjunto de datos.

Por tanto, el objetivo en la experimentación es el de evaluar la influencia que tiene cada canal de información. Aunque las coordenadas de cada punto se tratan como características, realmente son elementos estructurales que posicionan a los puntos en el espacio, y que además van acompañados del resto de características.

3.2 Experimentaciones y pruebas

Así, dado que se dispone de 10 canales de información y un conjunto de datos abundante, se plantea un total de 6 experimentos para evaluar la influencia de los

diferentes canales de información, manteniendo en todo momento las coordenadas (r, g, b) pues se trata de la información que caracteriza a la geometría explícita en la nube de puntos. Fundamentalmente, las diferentes configuraciones para los experimentos variarán los canales de información utilizados, puesto que el resto de hiperparámetros han sido seleccionados del artículo original (Wang et al., 2018) y han sido previamente optimizados, así, no es necesario volver a realizar experimentos para validar esta configuración.

3.2.1 Experimento 1

El primer experimento utiliza únicamente las coordenadas de los puntos para realizar el entrenamiento, dejando a un lado toda información adicional. El objetivo de este experimento es el de conocer cuál es el rendimiento máximo alcanzable utilizando únicamente la información geométrica presente en la nube de puntos.

Para realizar el experimento, se parametriza el *script* de entrenamiento con la siguiente configuración:

Argumento	Valor por defecto
--exp_name	*geometric_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z*
--num_features	*3*
--num_points	*4096*
--pooling	*max*

Tabla 16: Configuración asociada al experimento 1.

3.2.2 Experimento 2

En segunda instancia, se utilizan las coordenadas de los puntos acompañados del canal de intensidad, esto es, la medida sobre la reflectividad del material escaneado para generar la nube de puntos.

Para ejecutar este experimento, se ha tenido en cuenta la siguiente configuración de parámetros:

Argumento	Valor por defecto
--exp_name	*geometric_i_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z i*
--num_features	*4*
--num_points	*4096*
--pooling	*max*

Tabla 17: Configuración asociada al experimento 2.

3.2.3 Experimento 3

Para seguir evaluando la influencia de los diferentes canales, se utiliza en este caso el color para realizar el entrenamiento y comparar el rendimiento con respecto al caso anterior.

Se utiliza la siguiente configuración de entrenamiento usando además los canales **rgb**:

Argumento	Valor por defecto
--exp_name	*geometric_rgb_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z r g b*
--num_features	*6*
--num_points	*4096*
--pooling	*max*

Tabla 18: Configuración asociada al experimento 3.

3.2.4 Experimento 4

En este caso, se utilizan ambos canales de intensidad y color para realizar el experimento. Por tanto, se toma como entrada la configuración del experimento anterior variando en este caso los canales de información utilizados para incluir tanto el color como la intensidad:

Argumento	Valor por defecto
--exp_name	*geometric_i_rgb_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z i r g b*
--num_features	*7*
--num_points	*4096*
--pooling	*max*

Tabla 19: Configuración asociada al experimento 4.

3.2.5 Experimento 5

Para evaluar la influencia de las coordenadas normalizadas, se realiza un experimento utilizando las coordenadas originales junto a las normalizadas, para comprobar la precisión de los modelos entrenados.

A continuación, se indican los parámetros de configuración de este experimento:

Argumento	Valor por defecto
--exp_name	*geometric_norm_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z xn yn zn*
--num_features	*6*
--num_points	*4096*
--pooling	*max*

Tabla 20: Configuración asociada al experimento 5.

3.2.6 Experimento 6

Finalmente, se utilizan todos los canales disponibles para comprobar la mejora existente en el uso de los 10 canales de información elegibles en comparación al uso de un conjunto reducido de información.

Así, los parámetros de configuración utilizados son:

Argumento	Valor por defecto
--exp_name	*geometric_base*
--dataset	*Semantic3D*
--input_data	*train_items*
-learning_rate	*0.001*
--scheduler	*True*
--epochs	*100*
--nearest_neighbors	*20*
--momentum	*0.9*
--output_models	*checkpoints*
--optimizer	*SGD*
--bias	*False*
--emb_dims	*1024*
--dropout	*False*
--workers	*8*
--batch_size	*2*
--features	*x y z i r g b xn yn zn*
--num_features	*10*
--num_points	*4096*
--pooling	*max*

Tabla 21: Configuración asociada al experimento 6.

3.3 Resultados y discusión

En esta sección se muestran los resultados de los experimentos planteados anteriormente. El objetivo final es el de comparar el resultado obtenido en los diferentes experimentos, de forma que pueda verificarse si realmente existe una ganancia en la precisión al incorporar canales de información adicional con respecto a la utilización de la geometría única y exclusivamente.

Por tanto, se detallarán los resultados experimento a experimento, indicando tanto las gráficas de precisión y coste, tanto la métrica *IoU* (*Intersection over Union*) ampliamente utilizada en problemas de segmentación, las matrices de confusión y finalmente precisión y coste promedios en las *k* ejecuciones realizadas según el esquema de validación cruzada empleado en cada caso.

Para monitorizar el rendimiento de la red neuronal en todos los experimentos se ha hecho uso de *Tensorboard*, un módulo del *framework TensorFlow* que está adaptado para su uso desde PyTorch que permite registrar progresivamente los resultados obtenidos y construye automáticamente las gráficas a mostrar. Además, consta de un parámetro que permite configurar el suavizado a aplicar en cada gráfica. Este suavizado facilita el análisis posterior dado que elimina la alta variabilidad de los resultados, dejando ver la tendencia del experimento, siendo esto objeto de más interés en las diferentes ejecuciones realizadas.

3.3.1 Experimento 1

A continuación, se muestran los resultados correspondientes al primer experimento. En esta ejecución se han considerado únicamente los canales de información geométrica, (x, y, z), y el objetivo es evaluar el rendimiento máximo alcanzable usando únicamente esta información.

Puesto que se ha considerado un esquema de validación cruzada, *cross validation K-fold* con $K = 5$, se mostrarán las gráficas de precisión y coste en cada una de las particiones de entrenamiento y validación. Para realizar el cálculo de la precisión se ha hecho uso de la **intersección sobre la unión**, que mide la cantidad de puntos que han sido etiquetados correctamente con respecto al total. Se trata de la medida estándar utilizada en problemas de segmentación, tanto en modelos de datos tridimensionales como en modelos bidimensionales (imágenes).

3.3.1.1 Precisión y coste general

Se adjuntan las gráficas de **precisión y coste** en las 5 particiones realizadas para entrenamiento y validación.

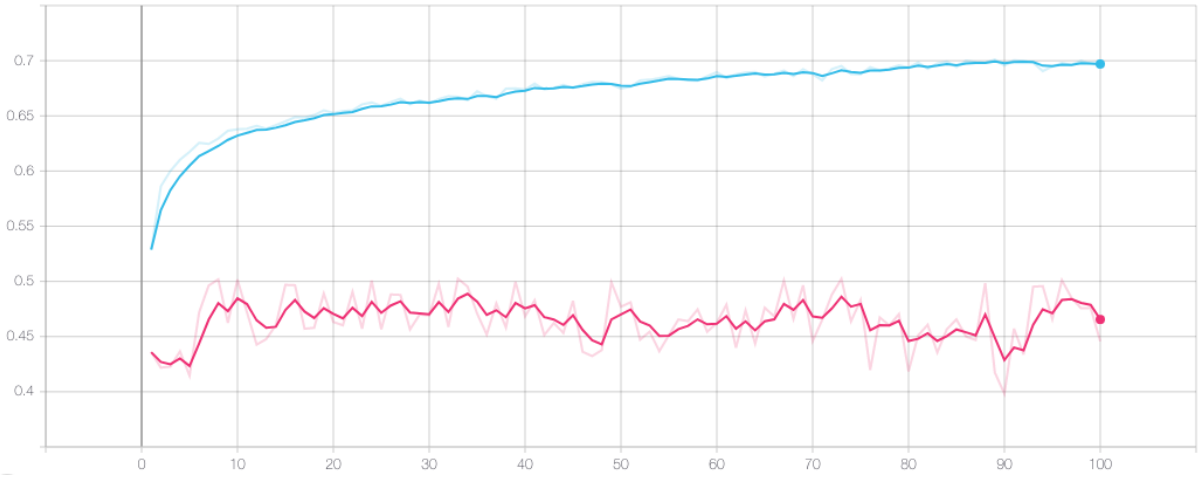

Figura 3.1: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 – experimento 1.

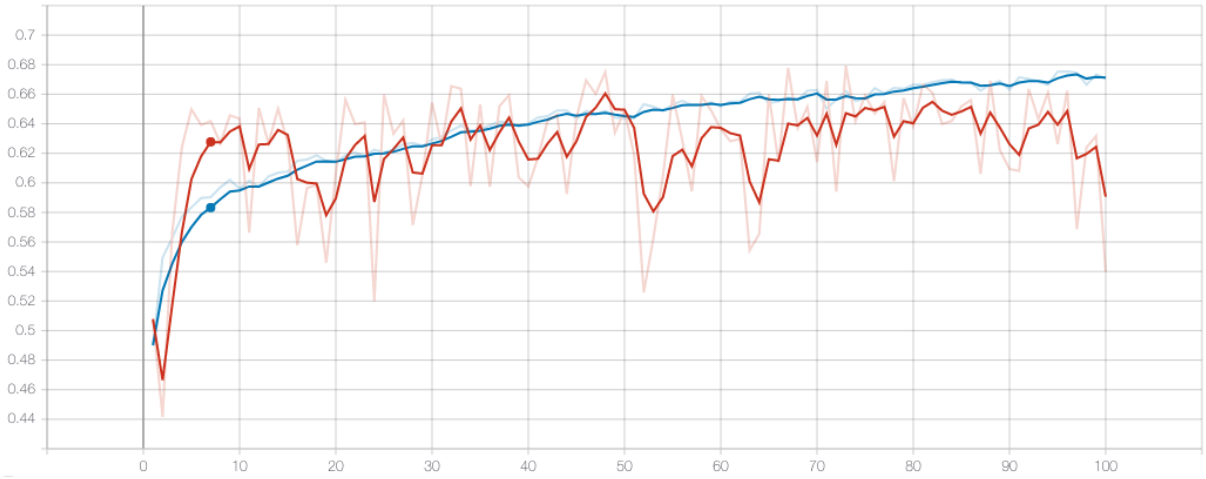

Figura 3.2: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 1.

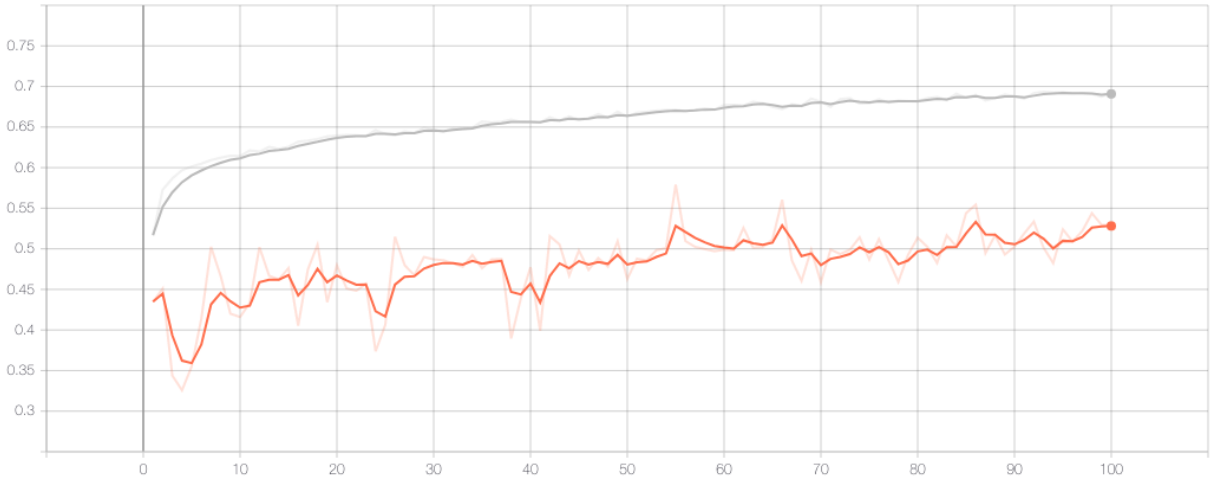

Figura 3.3: Precisión entrenamiento (gris) vs. validación (naranja) en partición 3 - experimento 1.

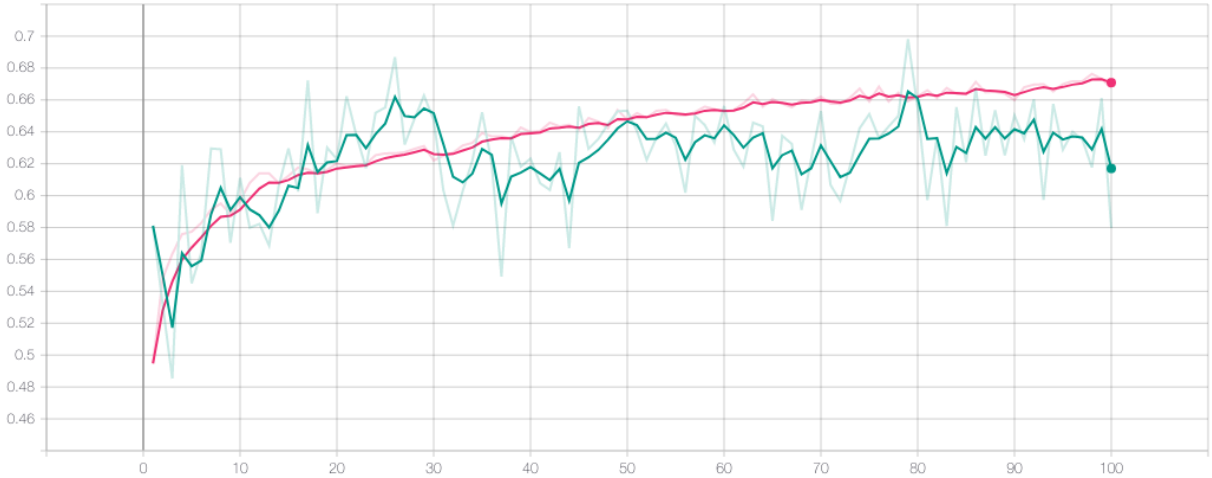

Figura 3.4: Precisión entrenamiento (rosa) vs. validación (verde) en partición 4 - experimento 1.

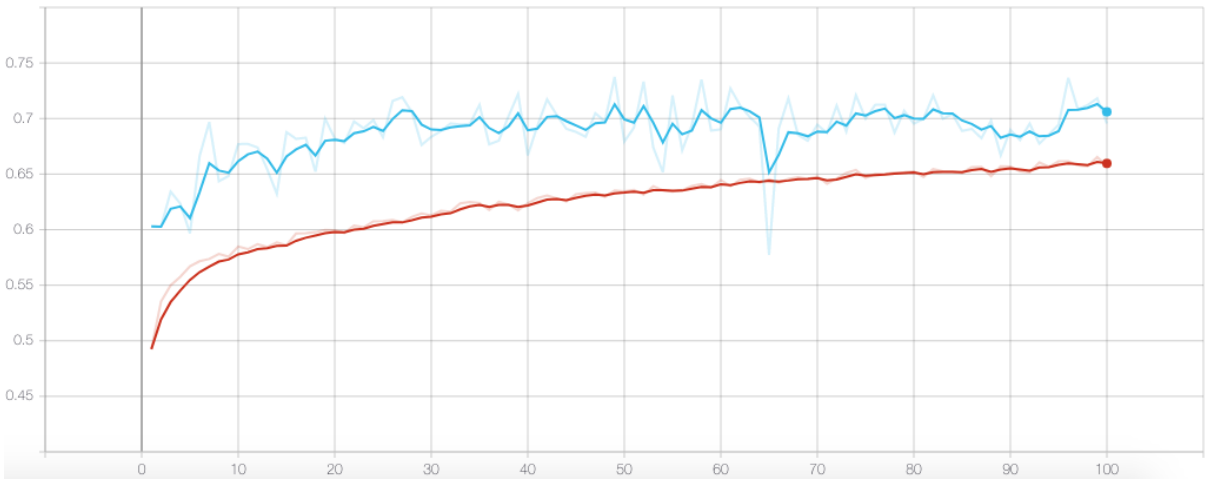

Figura 3.5: Precisión entrenamiento (rojo) vs. validación (azul) en partición 5 - experimento 1.

Como se puede observar, el rendimiento máximo alcanzable para el entrenamiento en todas las particiones está en torno al 70%, lo cual es una cifra algo baja teniendo en cuenta los resultados alcanzables por la mayoría de las arquitecturas actuales. La precisión promedio en las 5 particiones oscila entre el 40% y 60% teniendo en cuenta el esquema de validación cruzada utilizado en este caso. Aunque los resultados muestran un rendimiento bajo a priori, es necesario observar la precisión a nivel de clase, puesto que es probable que en algunas clases se alcance un rendimiento más óptimo que en el resto debido a que las formas que se agrupan en determinadas clases como por ejemplo **edificio** o **terreno** son geométricamente más simples que las que podrían agruparse en la categoría **vehículo**, por ejemplo.

A continuación, se observan los resultados obtenidos en cuanto a coste para identificar el posible sobreajuste en las diferentes ejecuciones del experimento.

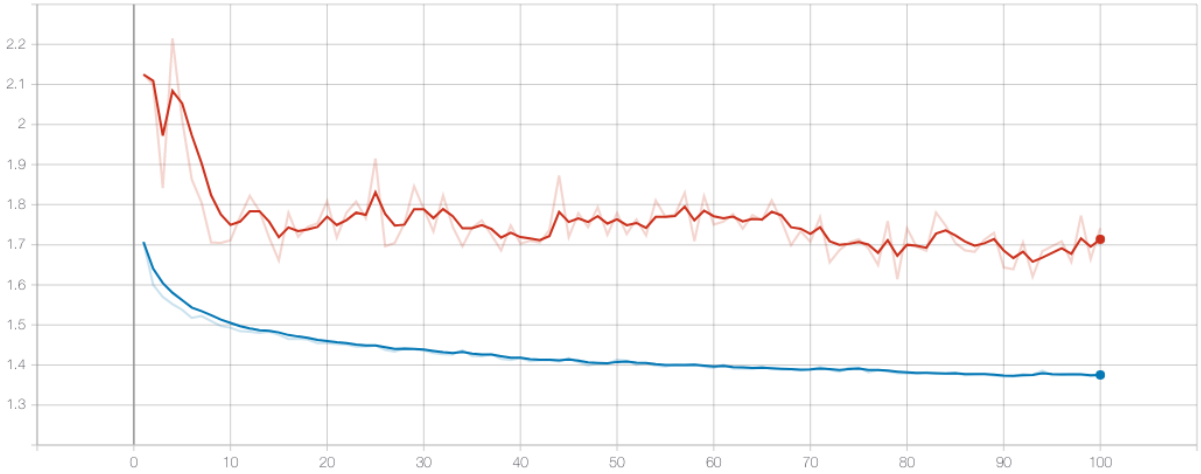

Figura 3.6: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 1.

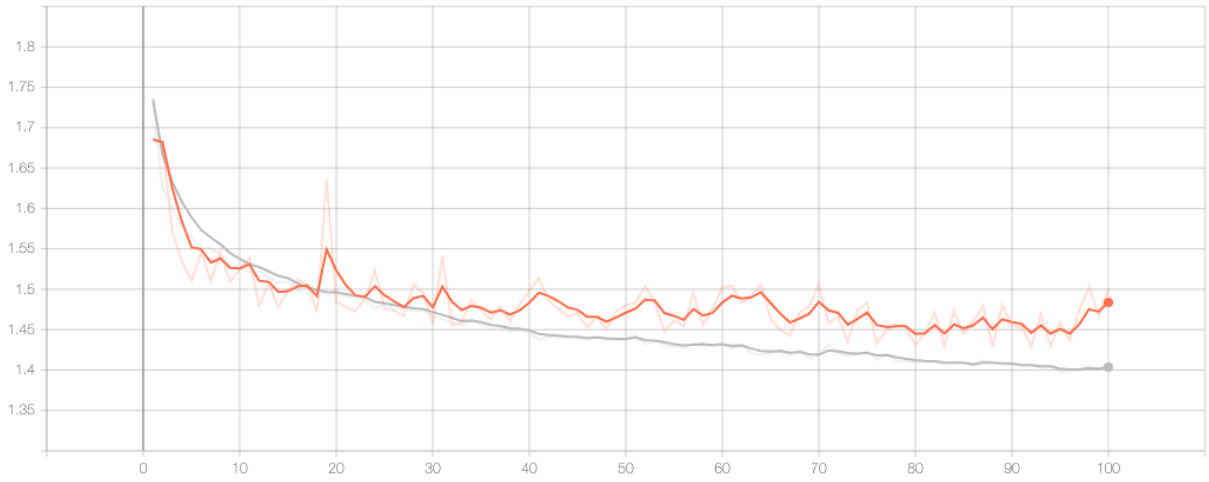

Figura 3.7: Coste de entrenamiento (gris) vs. validación (naranja) en partición 2 - experimento 1.

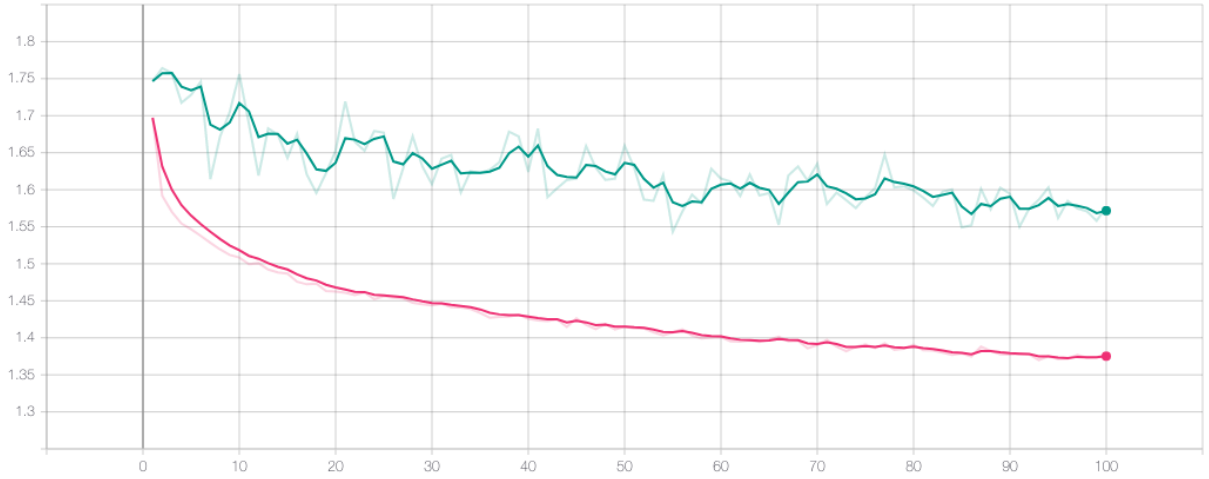

Figura 3.8: Coste de entrenamiento (rosa) vs. validación (verde) en partición 3 - experimento 1.

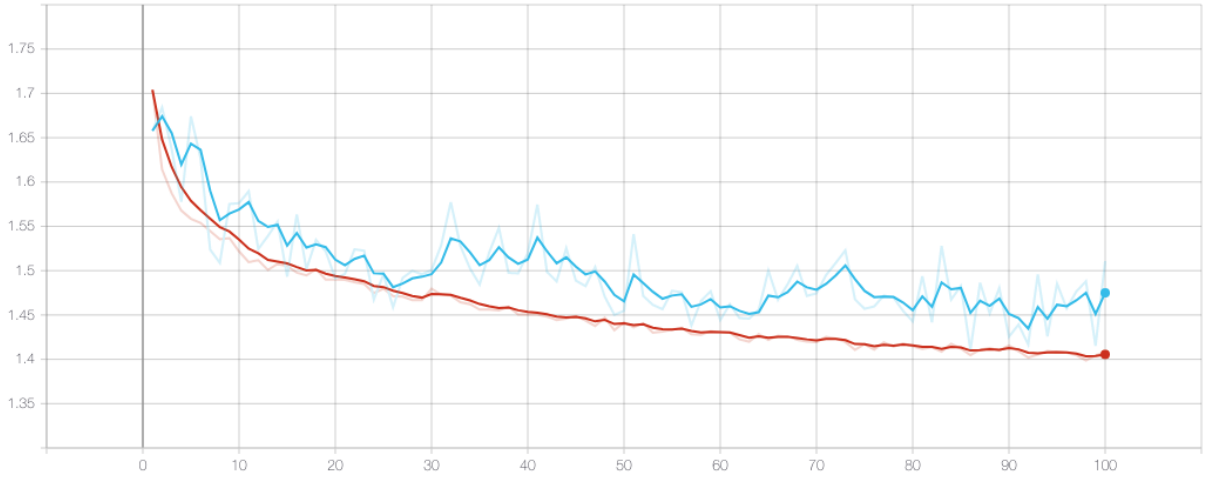

Figura 3.9: Coste de entrenamiento (rojo) vs. validación (azul) en partición 4 - experimento 1.

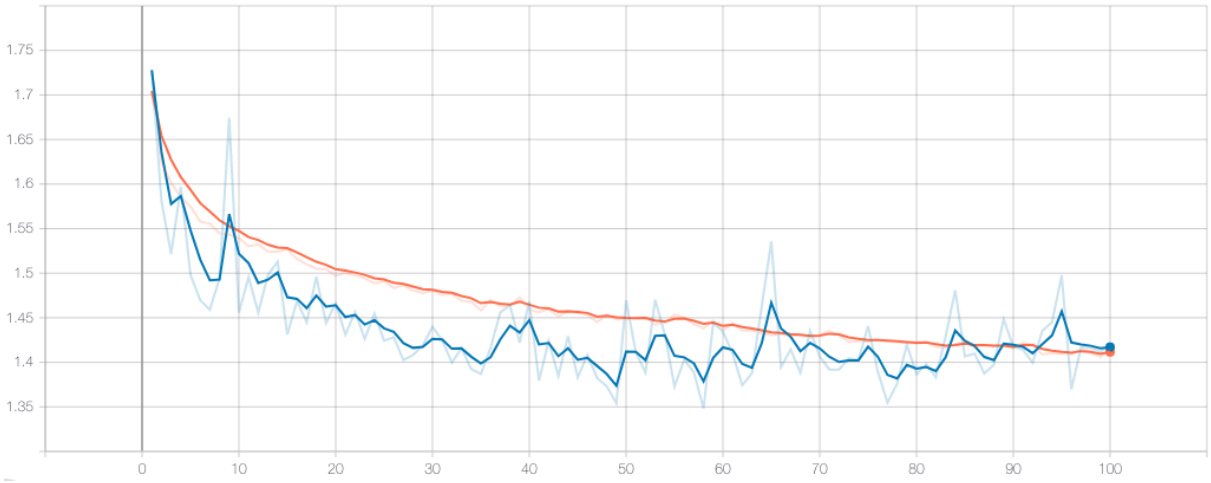

Figura 3.10: Coste de entrenamiento (naranja) vs. validación (azul) en partición 5 - experimento 1.

Como puede apreciarse, el proceso de aprendizaje es evidente puesto que el coste asociado a las predicciones realizadas por la red se disminuye conforme se ejecutan las diferentes iteraciones en cada partición. Por otra parte, en ninguno de los 5 ciclos de entrenamiento/validación puede apreciarse una divergencia significativa entre las gráficas de coste de entrenamiento y validación, esto es, no existe **sobreajuste** del modelo.

El fenómeno del sobreajuste hace referencia al hecho de que la red realiza predicciones óptimas para el conjunto de entrenamiento, mientras que éste empeora significativamente cuando ha de predecir sobre datos que no se han contemplado en el entrenamiento, imitando el hecho de que la red ha **memorizado** los datos de entrenamiento, y no ha conseguido extraer conocimiento de valor, mermándose así su capacidad de **generalización.** En términos de valores, esto se traduce en una divergencia entre las gráficas de coste asociadas a la fase de entrenamiento y posterior validación en cada *epoch*. Esto es, a partir de un determinado *epoch* la gráfica de coste asociada al entrenamiento sigue disminuyendo mientras que la gráfica de coste asociada a validación comienza a aumentar de forma significativa.

3.3.1.2 IoU por clase

Dado que no es adecuado mostrar únicamente la cifra de precisión obtenida a nivel promedio para todas las clases, en esta sección se muestra cómo evoluciona la **intersección sobre la unión** (IoU, *Intersection over Union*) o precisión en cada una de las clases que conforman el problema.

Para facilitar el análisis, se adjunta una tabla que hace referencia a la codificación de las clases del problema para el *dataset* Semantic3D.

Identificador de clase	Clase asociada
0	Terreno artificial
1	Terreno natural
2	Vegetación alta
3	Vegetación baja
4	Edificios
5	Paisaje accidentado
6	Artefactos de escaneo
7	Coches

Tabla 22: Clases del dataset Semantic3D.

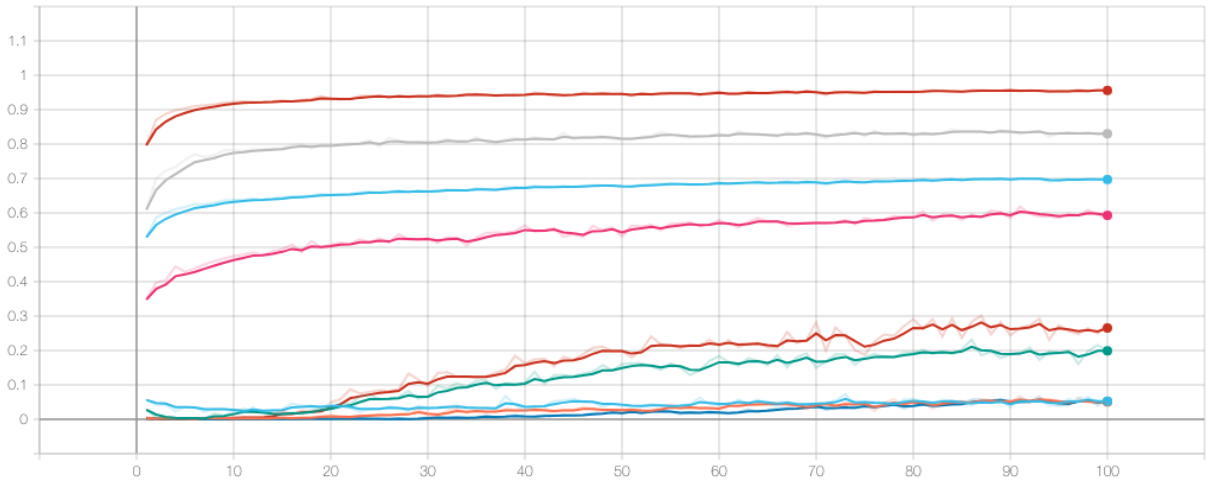

Figura 3.11: IoU asociada al entrenamiento en partición 1 - experimento 1.

Name	Smoothed	Value
1-fold_iou_train_class_0	0.956	0.9557
1-fold_iou_train_class_1	0.05248	0.05232
1-fold_iou_train_class_2	0.5928	0.5856
1-fold_iou_train_class_3	0.199	0.1986
1-fold_iou_train_class_4	0.8301	0.8298
1-fold_iou_train_class_5	0.05046	0.05086
1-fold_iou_train_class_6	0.05176	0.05631
1-fold_iou_train_class_7	0.2653	0.2801
1-fold_iou_train_mean_IoU	0.697	0.6963

Figura 3.12: Leyenda asociada al entrenamiento en partición 1 - experimento 1.

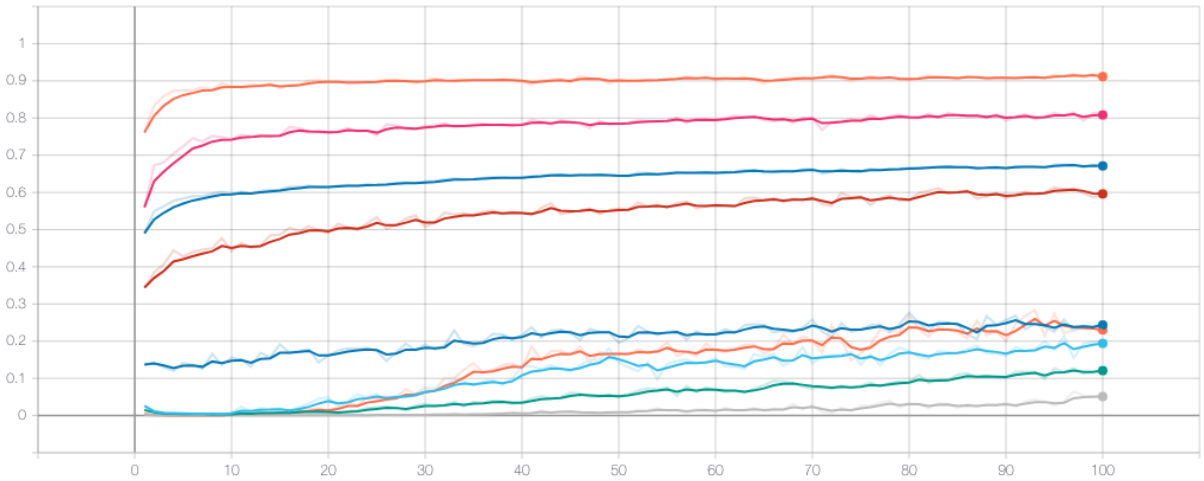

Figura 3.13: IoU asociada al entrenamiento en partición 2 - experimento 1.

Name	Smoothed	Value
2-fold_iou_train_class_0	0.9115	0.9059
2-fold_iou_train_class_1	0.2436	0.2516
2-fold_iou_train_class_2	0.596	0.5941
2-fold_iou_train_class_3	0.194	0.1996
2-fold_iou_train_class_4	0.8083	0.8096
2-fold_iou_train_class_5	0.1208	0.1266
2-fold_iou_train_class_6	0.05111	0.05161
2-fold_iou_train_class_7	0.2302	0.2236
2-fold_iou_train_mean_IoU	0.6713	0.6707

Figura 3.14: Leyenda asociada al entrenamiento en partición 2 - experimento 1.

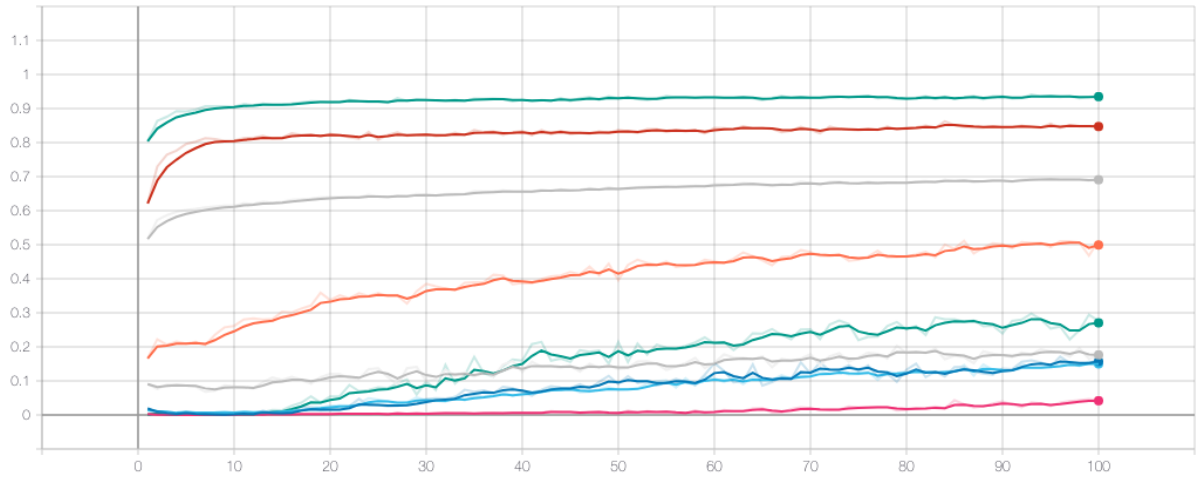

Figura 3.15: IoU asociada al entrenamiento en partición 3 - experimento 1.

Name	Smoothed	Value
3-fold_iou_train_class_0	0.9349	0.9362
3-fold_iou_train_class_1	0.1762	0.174
3-fold_iou_train_class_2	0.4992	0.5114
3-fold_iou_train_class_3	0.1573	0.1641
3-fold_iou_train_class_4	0.8471	0.8455
3-fold_iou_train_class_5	0.1501	0.1499
3-fold_iou_train_class_6	0.04174	0.04186
3-fold_iou_train_class_7	0.2705	0.2761
3-fold_iou_train_mean_IoU	0.6908	0.6922

Figura 3.16: Leyenda asociada al entrenamiento en partición 3 - experimento 1.

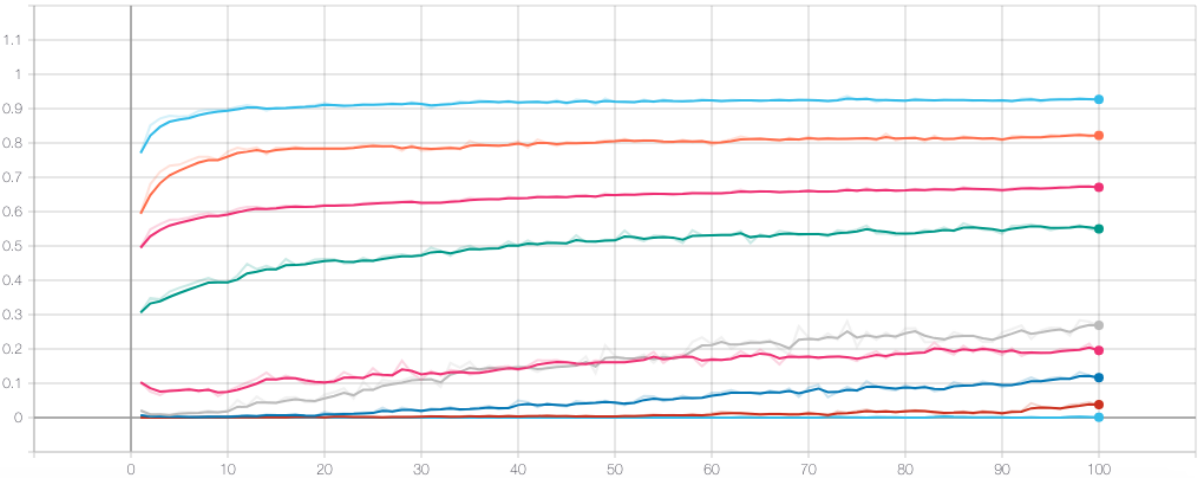

Figura 3.17: IoU asociada al entrenamiento en partición 4 - experimento 1.

Name	Smoothed	Value
4-fold_iou_train_class_0	0.9269	0.9259
4-fold_iou_train_class_1	0.1956	0.1824
4-fold_iou_train_class_2	0.5497	0.5434
4-fold_iou_train_class_3	0.2688	0.2676
4-fold_iou_train_class_4	0.8219	0.8228
4-fold_iou_train_class_5	0.1165	0.1087
4-fold_iou_train_class_6	0.03784	0.03656
4-fold_iou_train_class_7	1.0396e-3	0
4-fold_iou_train_mean_IoU	0.6709	0.6677

Figura 3.18: Leyenda asociada al entrenamiento en partición 4 - experimento 1.

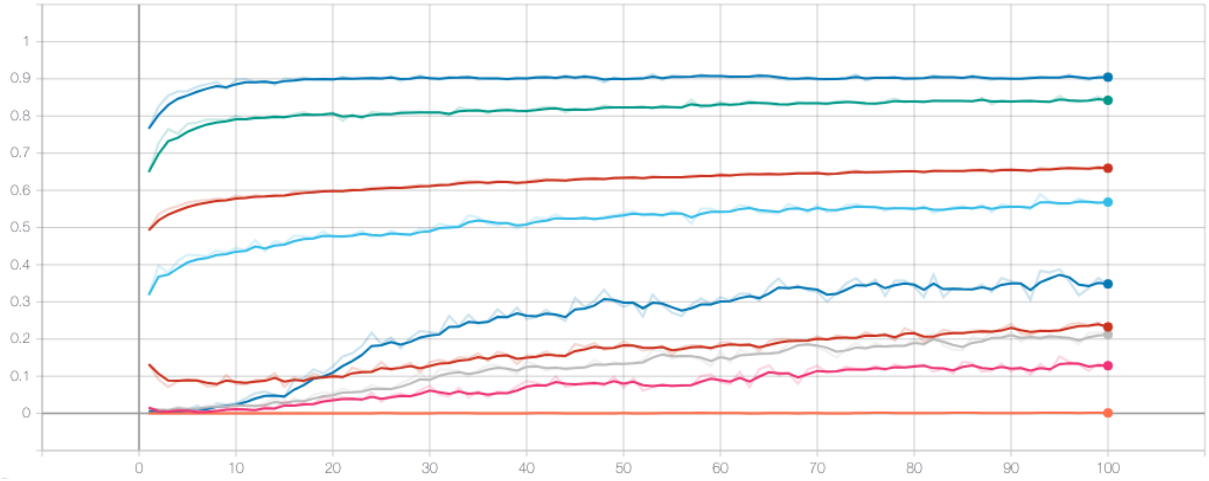

Figura 3.19: IoU asociada al entrenamiento en partición 5 - experimento 1.

Name	Smoothed	Value
5-fold_iou_train_class_0	0.9045	0.9058
5-fold_iou_train_class_1	0.2327	0.2227
5-fold_iou_train_class_2	0.5684	0.5703
5-fold_iou_train_class_3	0.1283	0.127
5-fold_iou_train_class_4	0.8422	0.8369
5-fold_iou_train_class_5	0.2121	0.2156
5-fold_iou_train_class_6	1.314e-3	5.6467e-4
5-fold_iou_train_class_7	0.3483	0.3435
5-fold_iou_train_mean_IoU	0.6596	0.6574

Figura 3.20: Leyenda asociada al entrenamiento en partición 5 - experimento 1.

Como se puede apreciar en las diferentes gráficas adjuntas, existe diferencia en la precisión obtenida para las diferentes clases del problema. Las categorías donde se alcanza la precisión máxima en todos los casos es en la clase 0 y clase 4, correspondientes a terreno artificial (*man made terrain*) y edificios, pues contienen las formas geométricas más simples y sencillas de identificar.

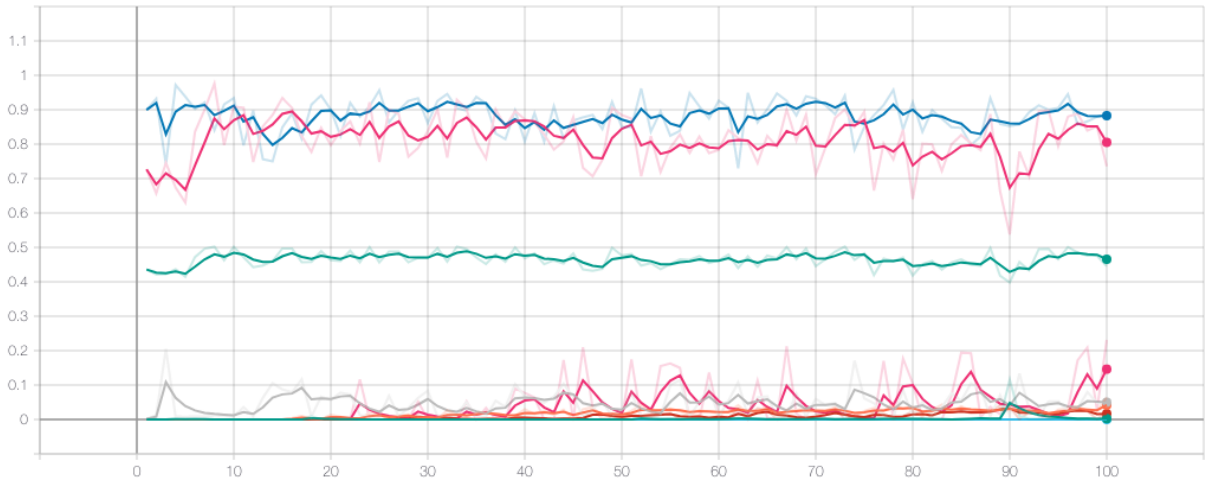

Figura 3.21: IoU asociada a validación en partición 1 - experimento 1.

Name	Smoothed	Value
1-fold_iou_test_class_0	0.8053	0.7357
1-fold_iou_test_class_1	2.7009e-3	3.627e-3
1-fold_iou_test_class_2	0.05013	0.04817
1-fold_iou_test_class_3	0.04162	0.06343
1-fold_iou_test_class_4	0.8828	0.8852
1-fold_iou_test_class_5	0.01669	0.01877
1-fold_iou_test_class_6	1.0333e-4	0
1-fold_iou_test_class_7	0.1463	0.231
1-fold_iou_test_mean_IoU	0.4655	0.4456

Figura 3.22: Leyenda asociada a validación en partición 1 - experimento 1.

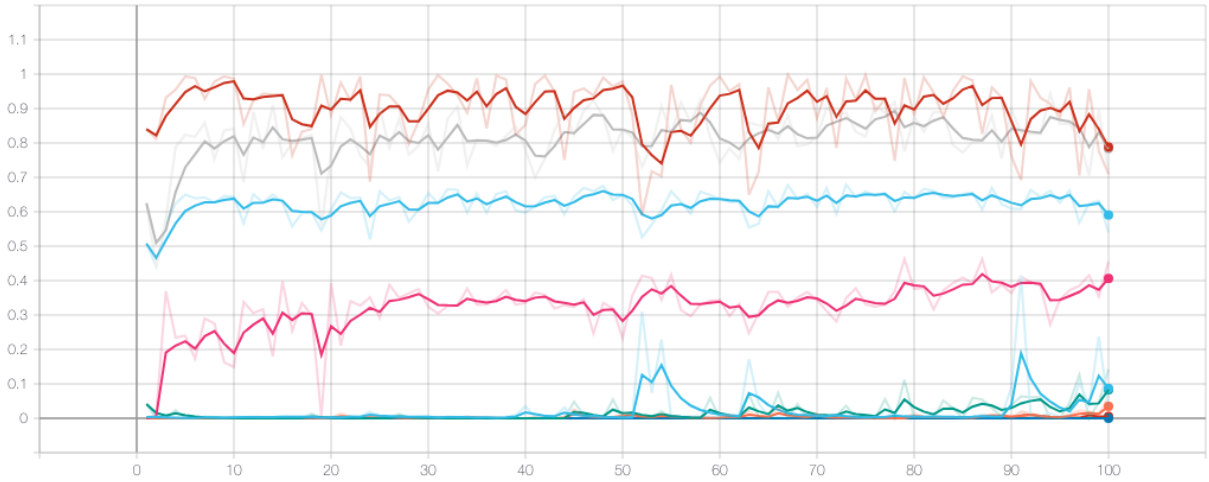

Figura 3.23: IoU asociada a validación en partición 2.

Name	Smoothed	Value
2-fold_iou_test_class_0	0.788	0.7091
2-fold_iou_test_class_1	0.0866	0.03153
2-fold_iou_test_class_2	0.4063	0.4554
2-fold_iou_test_class_3	0.08292	0.1421
2-fold_iou_test_class_4	0.7837	0.7129
2-fold_iou_test_class_5	0.03465	0.06677
2-fold_iou_test_class_6	1.737e-23	0
2-fold_iou_test_class_7	5.3456e-3	5.2453e-3
2-fold_iou_test_mean_IoU	0.5906	0.54

Figura 3.24: Leyenda asociada a validación a partición 2 - experimento 1.

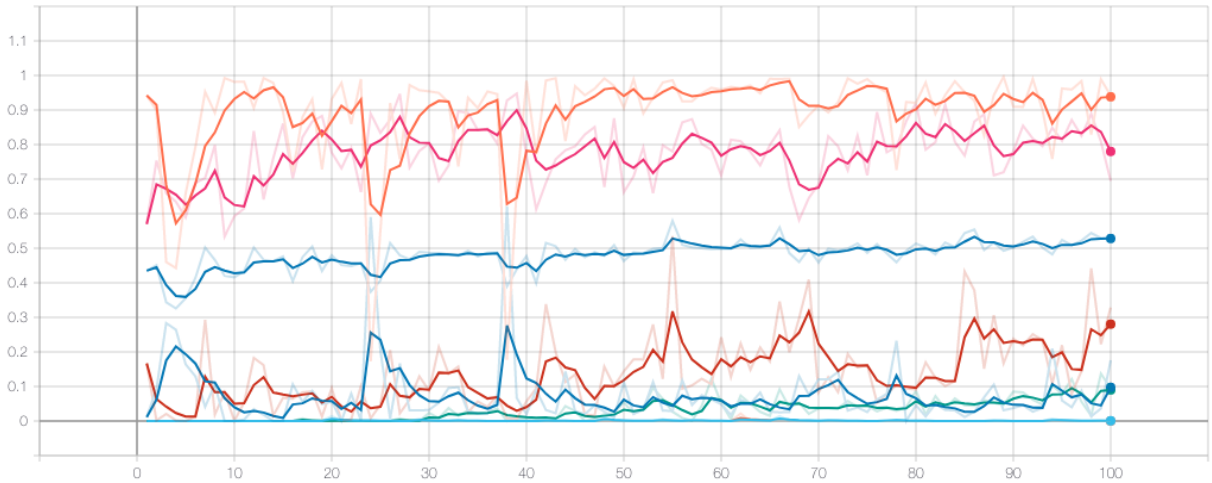

Figura 3.25: IoU asociada a validación en partición 3 - experimento 1.

Name	Smoothed	Value
3-fold_iou_test_class_0	0.9382	0.9417
3-fold_iou_test_class_1	0.09753	0.1759
3-fold_iou_test_class_2	0.2803	0.3284
3-fold_iou_test_class_3	7.3204e-4	6.6981e-6
3-fold_iou_test_class_4	0.7801	0.6951
3-fold_iou_test_class_5	0.0899	0.09392
3-fold_iou_test_class_6	1.2168e-5	3.0421e-5
3-fold_iou_test_class_7	3.1973e-7	0
3-fold_iou_test_mean_IoU	0.5281	0.5291

Figura 3.26: Leyenda asociada a validación en partición 3 - experimento 1.

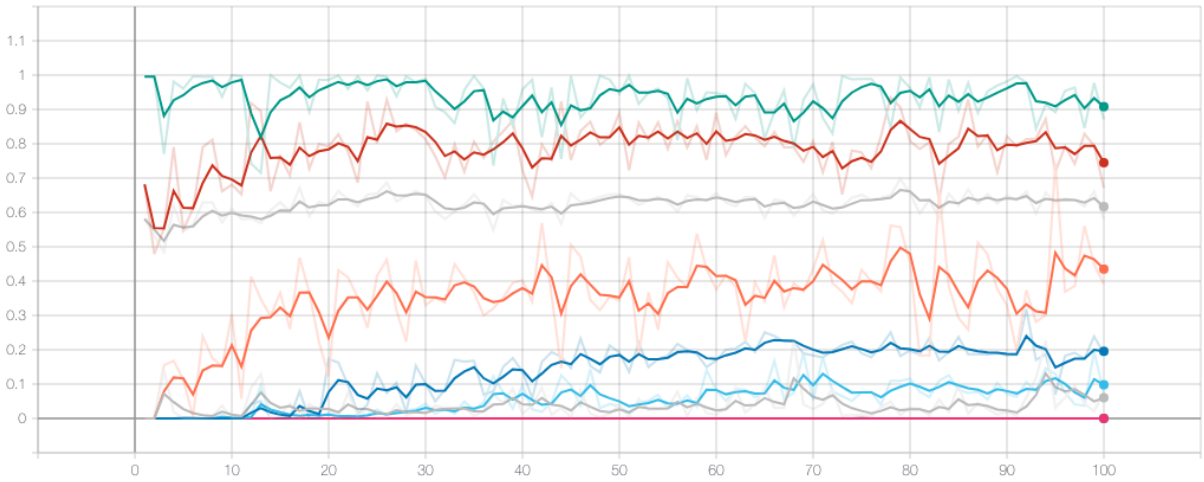

Figura 3.27: IoU asociada a validación en partición 4 - experimento 1.

Name	Smoothed	Value
4-fold_iou_test_class_0	0.9081	0.871
4-fold_iou_test_class_1	0.06035	0.0765
4-fold_iou_test_class_2	0.4349	0.3915
4-fold_iou_test_class_3	0.1955	0.1888
4-fold_iou_test_class_4	0.745	0.6716
4-fold_iou_test_class_5	0.09803	0.07364
4-fold_iou_test_class_6	1.5497e-5	2.757e-5
4-fold_iou_test_class_7	1.7255e-9	0
4-fold_iou_test_mean_IoU	0.617	0.5798

Figura 3.28: Leyenda asociada a validación en partición 4 - experimento 1.

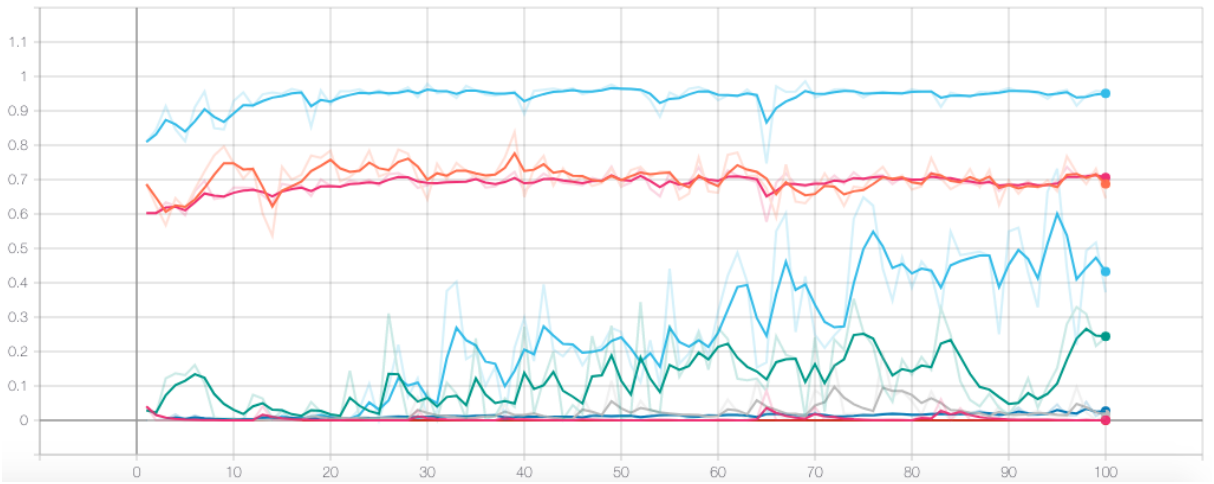

Figura 3.29: IoU asociada a validación en partición 5 - experimento 1.

	Name	Smoothed	Value
●	5-fold_iou_test_class_0	0.9506	0.9549
●	5-fold_iou_test_class_1	6.6136e-4	1.1798e-3
●	5-fold_iou_test_class_2	0.244	0.2407
●	5-fold_iou_test_class_3	0.01749	8.8002e-3
●	5-fold_iou_test_class_4	0.6876	0.6458
●	5-fold_iou_test_class_5	0.02687	0.02992
●	5-fold_iou_test_class_6	4.1376e-7	0
●	5-fold_iou_test_class_7	0.4326	0.3723
●	5-fold_iou_test_mean_IoU	0.706	0.6955

Figura 3.30: Leyenda asociada a validación en partición 5 - experimento 1.

En el caso de la validación vuelve a repetirse el fenómeno del entrenamiento, pues las clases 0 y 4 son las categorías donde se ha conseguido el rendimiento óptimo, con un 95,4% y 64,5% de precisión respectivamente. Se trata, al igual que en el caso de entrenamiento, de las categorías **terreno artificial** y **edificios**, pues se trata de las categorías que albergan las formas geométricas más sencillas.

Ha de destacarse que en este experimento se está utilizando únicamente la información geométrica para desarrollar los diferentes ciclos de entrenamiento y validación. El resto de clases albergan formas geométricas muy complejas, y dada la falta de orden inherente a las nubes de puntos, así como la alta variabilidad de los puntos, dificultan la resolución inteligente de este problema utilizando únicamente esta información.

3.3.1.3 Precisión y coste promedio k-Fold

Se obtienen los siguientes resultados promedio considerando el rendimiento obtenido en el **último *epoch*** de entrenamiento/validación:

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,39
Coste promedio validación	1,55
Precisión promedio entrenamiento	67,9%
Precisión promedio validación	55,8%

Tabla 23: Resultados promedio del experimento 1.

3.3.2 Experimento 2

Además de la información geométrica, en este experimento se tiene en cuenta el canal de intensidad. Esto es, la medida sobre la reflectividad de las superficies escaneadas haciendo uso del sensor LiDAR. El objetivo de este experimento es el de evaluar la ganancia que aporta este canal de información en comparación al uso exclusivo de la información geométrica.

3.3.2.1 Precisión y coste general

A continuación, se añaden las gráficas de precisión en las 5 particiones de entrenamiento y validación:

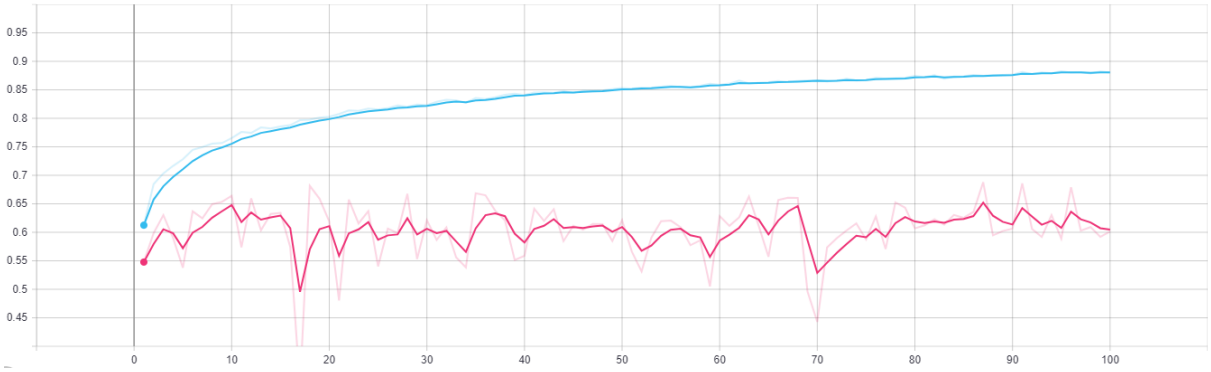

Figura 3.31: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 - experimento 2.

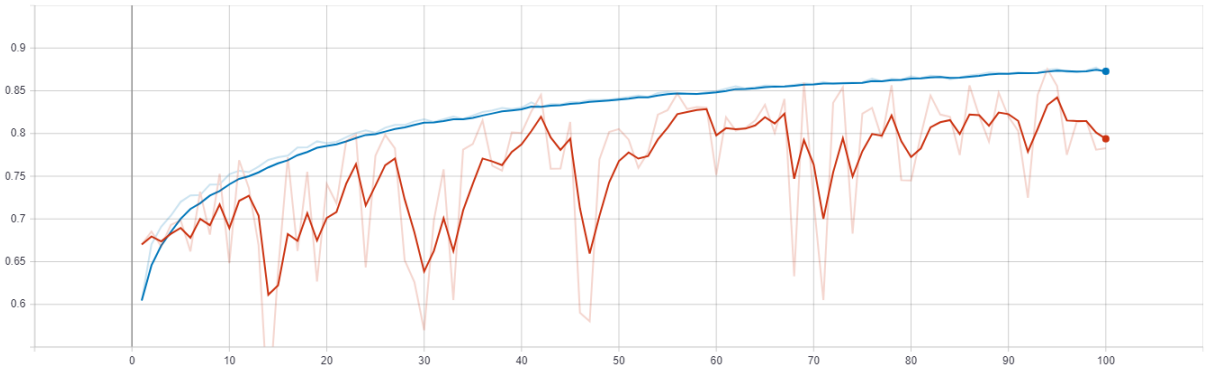

Figura 3.32: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 2.

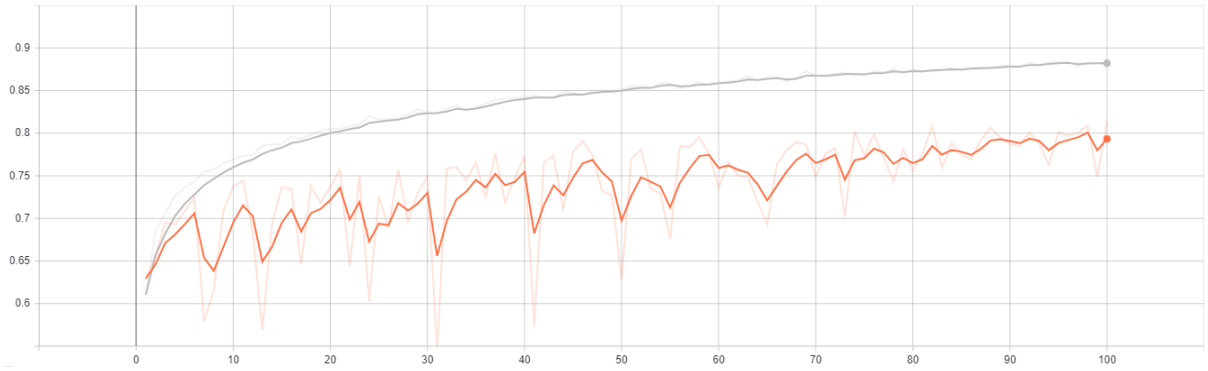

Figura 3.33: Precisión entrenamiento (gris) vs. validación (naranja) en partición 3 - experimento 2.

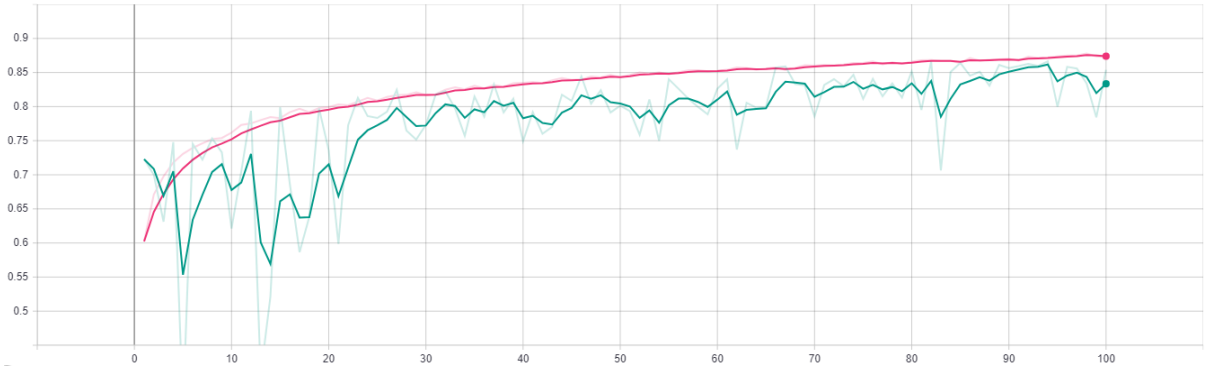

Figura 3.34: Precisión entrenamiento (rosa) vs. validación (verde) en partición 4 - experimento 2.

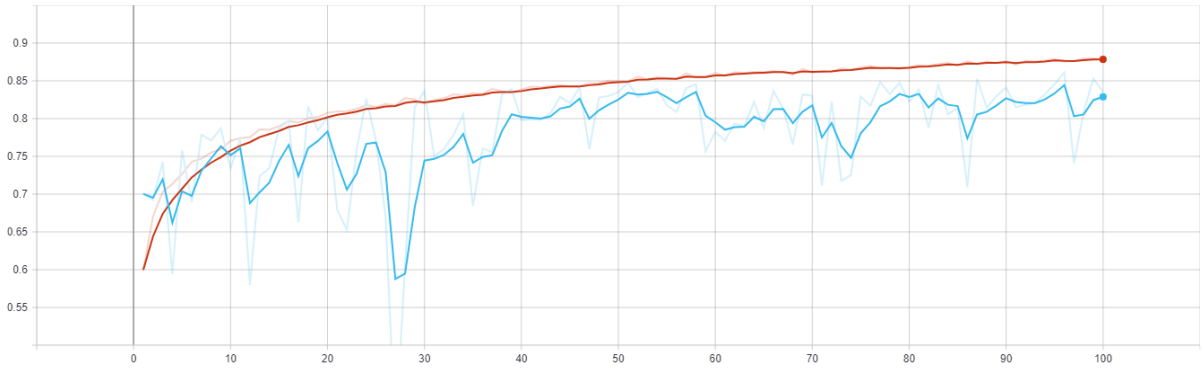

Figura 3.35: Precisión entrenamiento (rojo) vs. validación (azul) en partición 5 - experimento 2.

Como puede observarse en las figuras anteriores, se consiguen resultados significativamente mejores que los obtenidos usando únicamente el canal geométrico en las mismas condiciones. En el caso del entrenamiento, se consigue una precisión que supera el 85% en la totalidad de las particiones, mientras que los resultados en validación giran en torno al 80%.

Una vez analizada la precisión obtenida añadiendo el canal de intensidad, se evalúan las gráficas de coste para identificar si ha aparecido sobreajuste en el entrenamiento de la red.

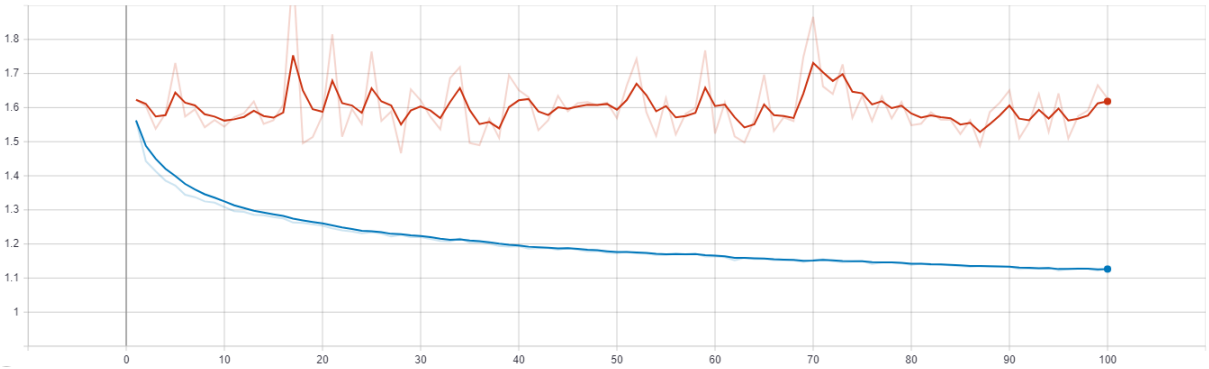

Figura 3.36: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 2.

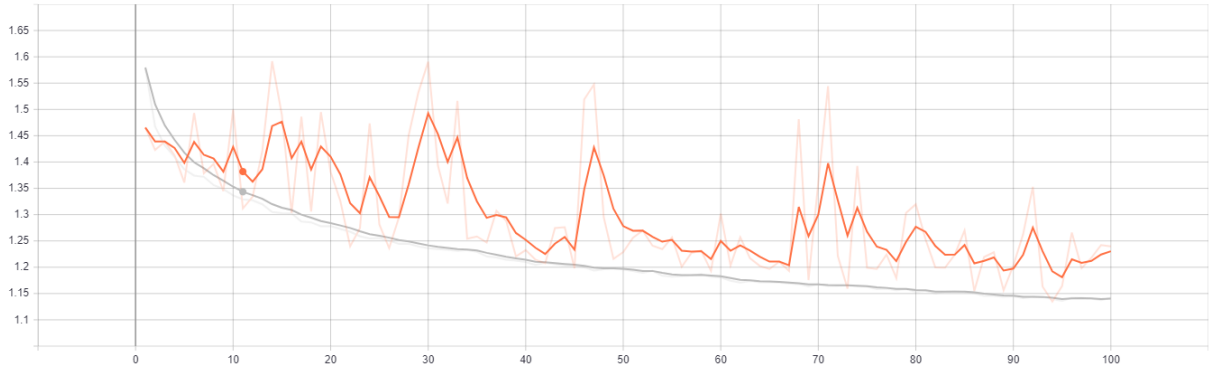

Figura 3.37: Coste de entrenamiento (gris) vs. validación (naranja) en partición 2 - experimento 2.

Como se puede observar, en la segunda partición disminuye el coste asociado a validación de una forma muy similar a como lo hace en el caso de la partición 1. Esto se debe a que la calidad de los datos de entrenamiento es más óptima en esta partición que en la primera.

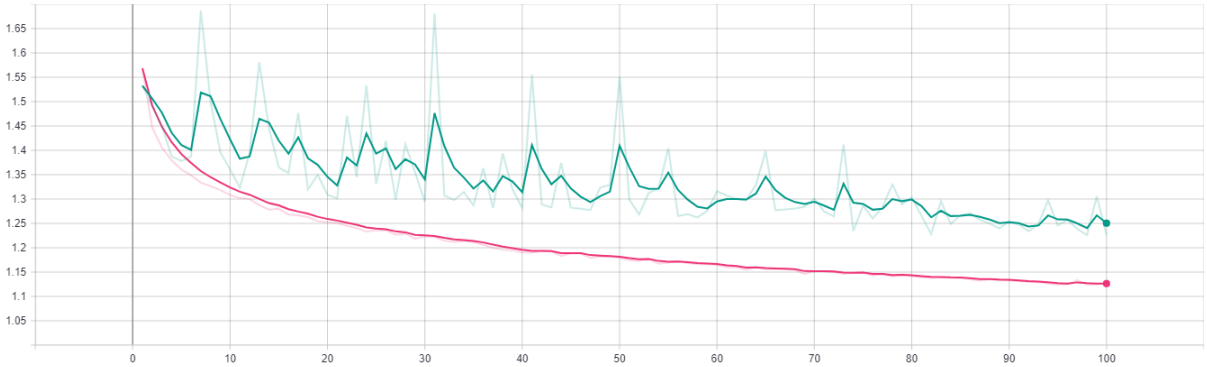

Figura 3.38: Coste de entrenamiento (rosa) vs. validación (verde) en partición 3 - experimento 2.

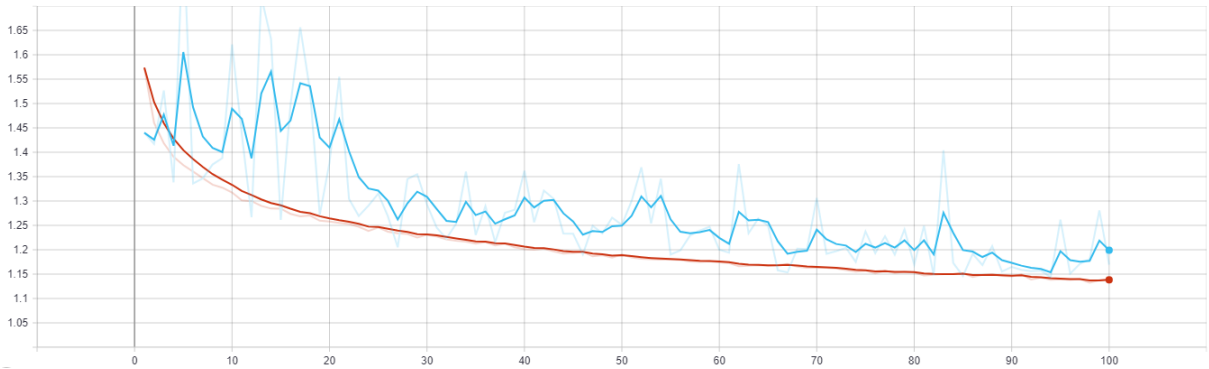

Figura 3.39: Coste de entrenamiento (rojo) vs. validación (verde) - experimento 2.

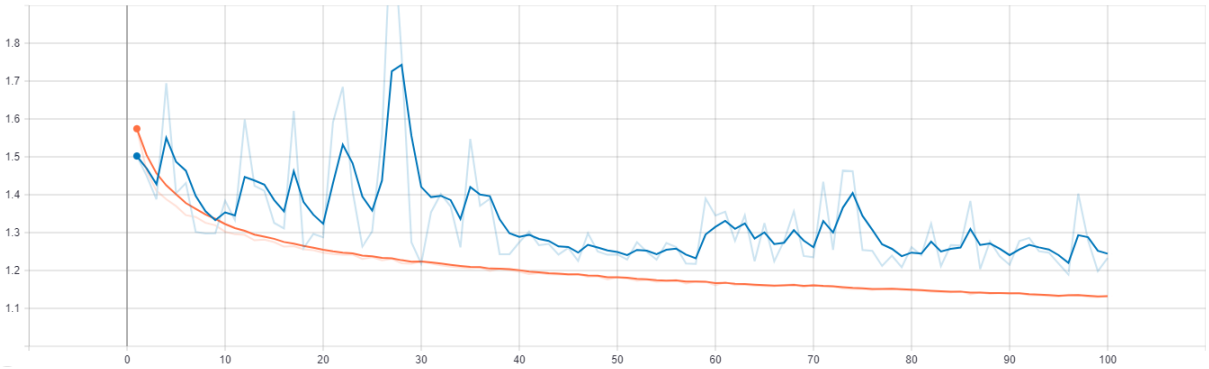

Figura 3.40: Coste de entrenamiento (naranja) vs. validación (azul) en partición 5 - experimento 2.

Se puede apreciar que en ninguna de las 5 particiones asociadas a *k-fold cross validation* se distingue una divergencia entre las gráficas de coste de entrenamiento y validación. En tanto, puede concluirse que el modelo no se ha sobreajustado, y por tanto los resultados obtenidos son adecuados.

3.3.2.2 IoU por clase

En esta sección, se muestran las precisiones obtenidas en cada clase tanto para el entrenamiento como para validación en cada una de las particiones asociadas. En primer lugar, se adjuntan las gráficas de precisión en cada clase para el entrenamiento.

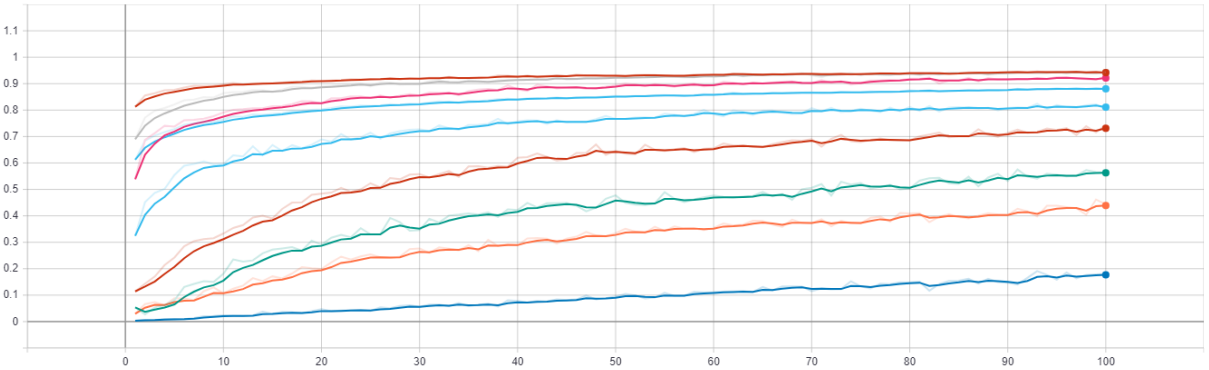

Figura 3.41: IoU asociada al entrenamiento en partición 1 - experimento 2.

Name	Smoothed	Value
1-fold_iou_train_class_0	0.9418	0.9385
1-fold_iou_train_class_1	0.811	0.8018
1-fold_iou_train_class_2	0.9214	0.9275
1-fold_iou_train_class_3	0.5627	0.5632
1-fold_iou_train_class_4	0.942	0.9442
1-fold_iou_train_class_5	0.439	0.4434
1-fold_iou_train_class_6	0.1769	0.1801
1-fold_iou_train_class_7	0.7312	0.7447
1-fold_iou_train_mean_IoU	0.8805	0.8802

Figura 3.42: Leyenda asociada al entrenamiento en partición 1 - experimento 2.

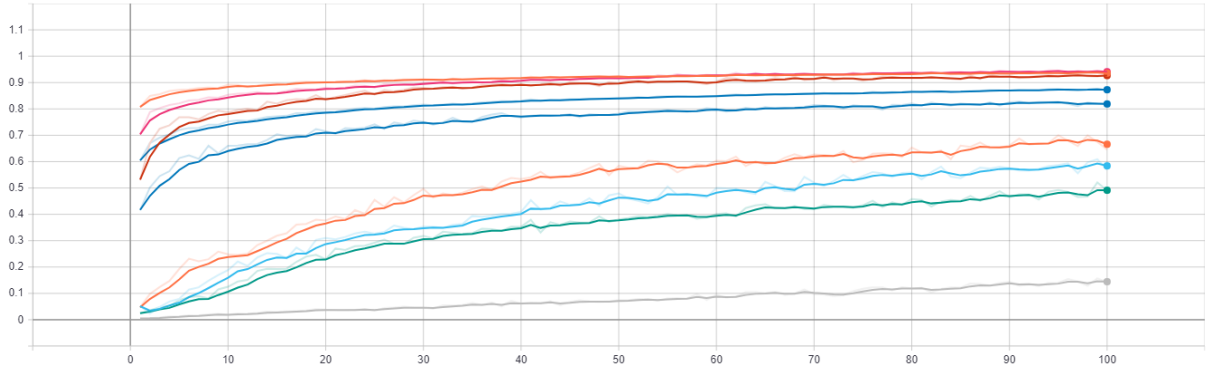

Figura 3.43: IoU asociada al entrenamiento en partición 2 - experimento 2.

Name	Smoothed	Value
2-fold_iou_train_class_0	0.9357	0.9325
2-fold_iou_train_class_1	0.8188	0.8171
2-fold_iou_train_class_2	0.9265	0.9291
2-fold_iou_train_class_3	0.5839	0.5698
2-fold_iou_train_class_4	0.9412	0.9391
2-fold_iou_train_class_5	0.4913	0.4904
2-fold_iou_train_class_6	0.1443	0.1425
2-fold_iou_train_class_7	0.666	0.6454
2-fold_iou_train_mean_IoU	0.8728	0.8701

Figura 3.44: Leyenda asociada al entrenamiento en partición 2 - experimento 2.

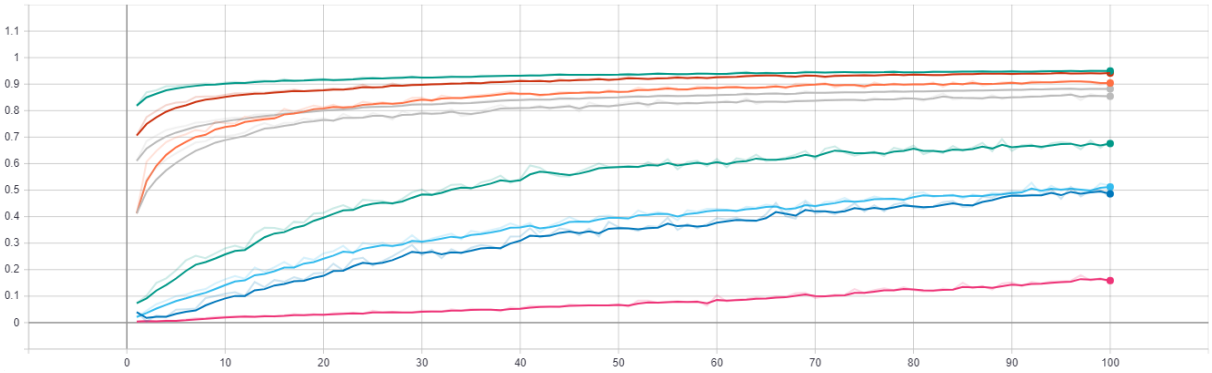

Figura 3.45: IoU asociada al entrenamiento en partición 3 - experimento 2.

Name	Smoothed	Value
3-fold_iou_train_class_0	0.9498	0.9491
3-fold_iou_train_class_1	0.8542	0.8527
3-fold_iou_train_class_2	0.9043	0.9044
3-fold_iou_train_class_3	0.4861	0.4726
3-fold_iou_train_class_4	0.9417	0.9431
3-fold_iou_train_class_5	0.512	0.5165
3-fold_iou_train_class_6	0.1587	0.1498
3-fold_iou_train_class_7	0.6758	0.6867
3-fold_iou_train_mean_IoU	0.8821	0.8821

Figura 3.46: Leyenda asociada al entrenamiento en partición 3 - experimento 2.

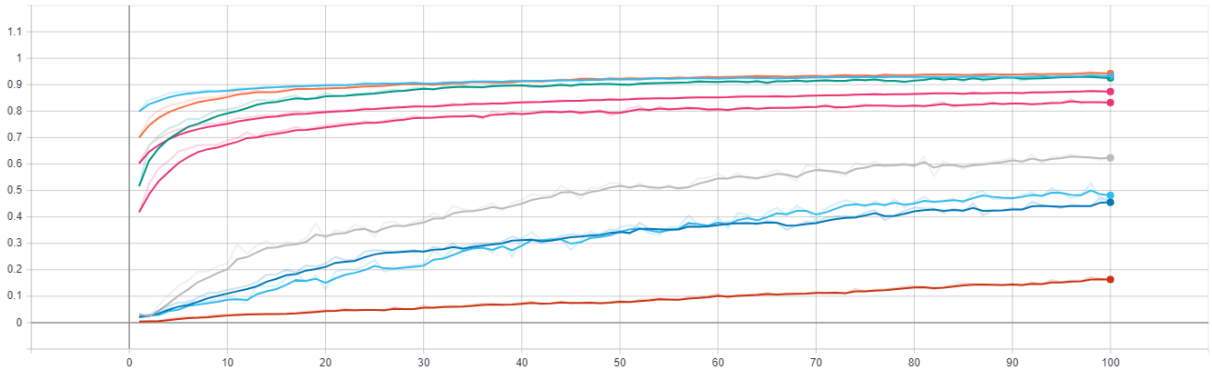

Figura 3.47: IoU asociada al entrenamiento en partición 4 - experimento 2.

Name	Smoothed	Value
4-fold_iou_train_class_0	0.933	0.9322
4-fold_iou_train_class_1	0.8323	0.8304
4-fold_iou_train_class_2	0.9253	0.9215
4-fold_iou_train_class_3	0.6233	0.6263
4-fold_iou_train_class_4	0.9423	0.9407
4-fold_iou_train_class_5	0.4552	0.4569
4-fold_iou_train_class_6	0.1633	0.1626
4-fold_iou_train_class_7	0.4816	0.4762
4-fold_iou_train_mean_IoU	0.8739	0.8726

Figura 3.48: Leyenda asociada al entrenamiento en partición 4 - experimento 2.

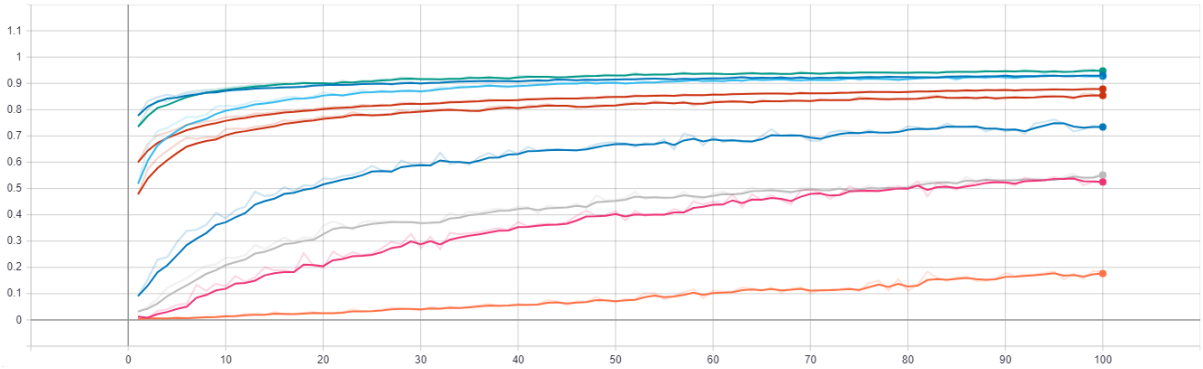

Figura 3.49: IoU asociada al entrenamiento en partición 5 - experimento 2.

Name	Smoothed	Value
5-fold_iou_train_class_0	0.9295	0.9297
5-fold_iou_train_class_1	0.8539	0.8527
5-fold_iou_train_class_2	0.9267	0.9268
5-fold_iou_train_class_3	0.5248	0.522
5-fold_iou_train_class_4	0.9474	0.9444
5-fold_iou_train_class_5	0.551	0.5657
5-fold_iou_train_class_6	0.1765	0.1816
5-fold_iou_train_class_7	0.734	0.7338
5-fold_iou_train_mean_IoU	0.8784	0.8784

Figura 3.50: Leyenda asociada al entrenamiento en partición 5 - experimento 2.

De los resultados anteriores puede observarse como la precisión se incrementa de forma significativa al incluir el canal de intensidad. Mientras que en el caso base, usando únicamente la geometría, se entrenaba el modelo consiguiendo los mejores resultados en las clases 0 y 4, terreno y edificios, en este caso (añadiendo el canal de

intensidad) se consiguen precisiones superiores al 50% en todas las clases excepto en la clase 6 (artefactos de escaneo) en la partición 5, por ejemplo. Esto indica de forma aproximada la ganancia en rendimiento cuando se incorporan canales adicionales a la geometría.

Tras completar el análisis sobre el entrenamiento, se procede a incluir los resultados de la validación.

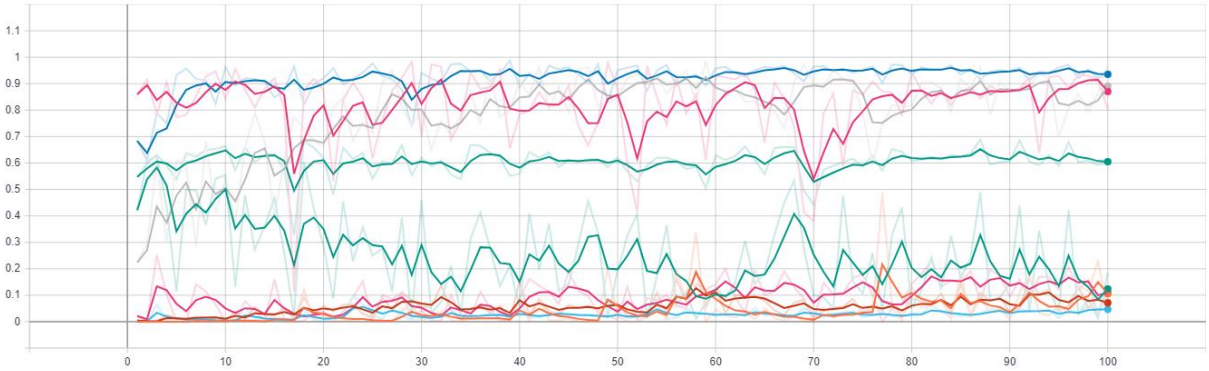

Figura 3.51: IoU asociada a validación en partición 1 - experimento 2.

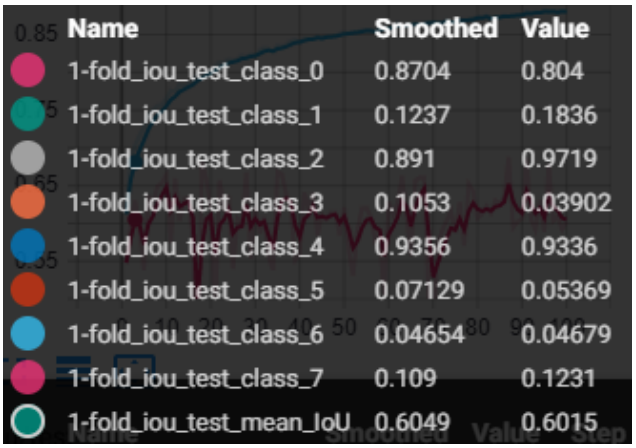

Figura 3.52: Leyenda asociada a validación en partición 1 - experimento 2.

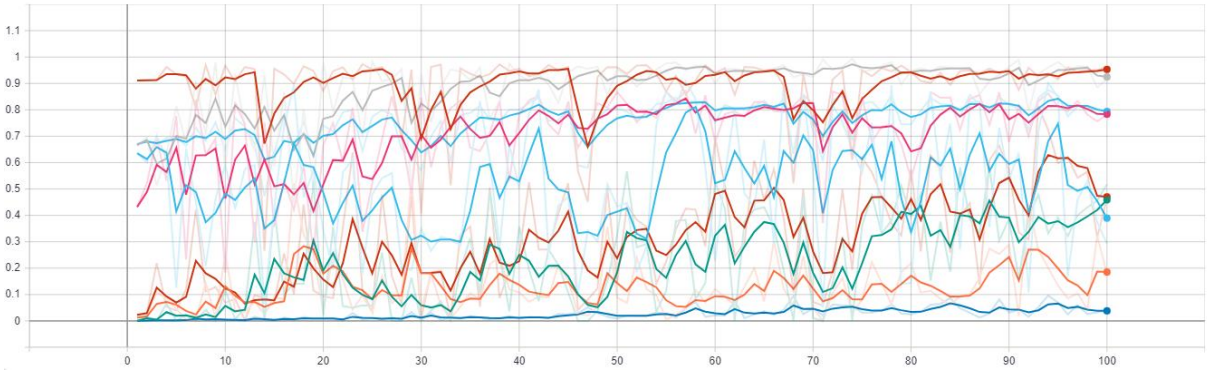

Figura 3.53: Leyenda asociada a validación en partición 2 - experimento 2.

Name	Smoothed	Value
2-fold_iou_test_class_0	0.9532	0.9618
2-fold_iou_test_class_1	0.3895	0.2918
2-fold_iou_test_class_2	0.7833	0.7816
2-fold_iou_test_class_3	0.4588	0.5171
2-fold_iou_test_class_4	0.9247	0.9171
2-fold_iou_test_class_5	0.1851	0.1822
2-fold_iou_test_class_6	0.03822	0.0378
2-fold_iou_test_class_7	0.4701	0.4643
2-fold_iou_test_mean_IoU	0.7938	0.7828

Figura 3.54: Leyenda asociada a validación en partición 2 - experimento 2.

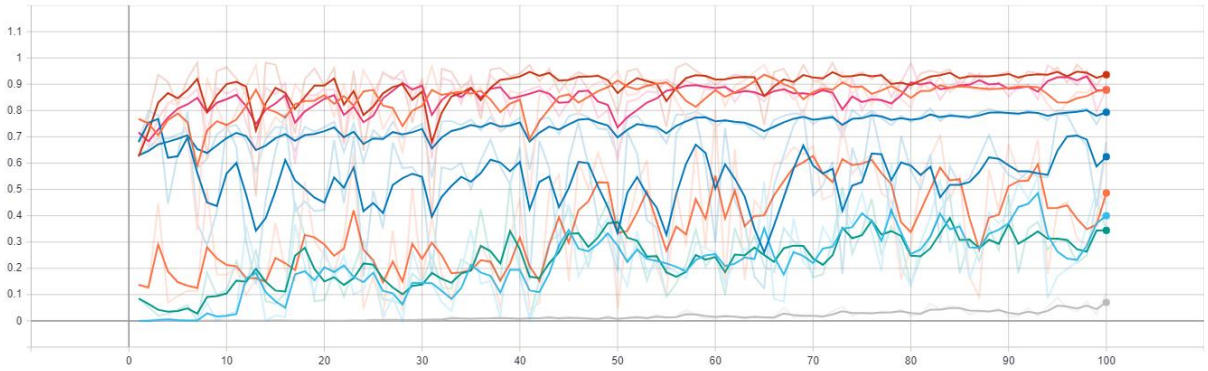

Figura 3.55: Leyenda asociada a validación en partición 3 - experimento 2.

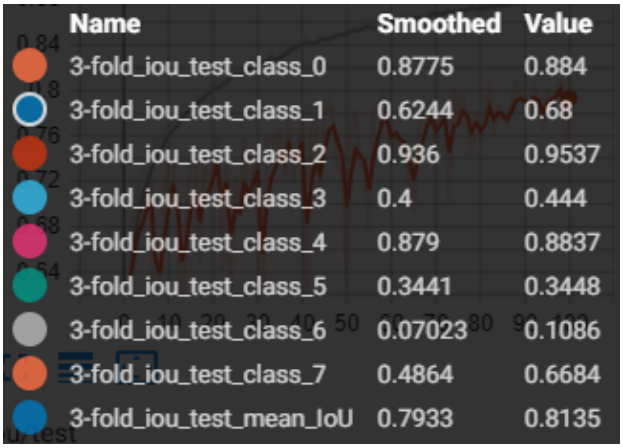

Figura 3.56: Leyenda asociada a validación en partición 3 - experimento 2.

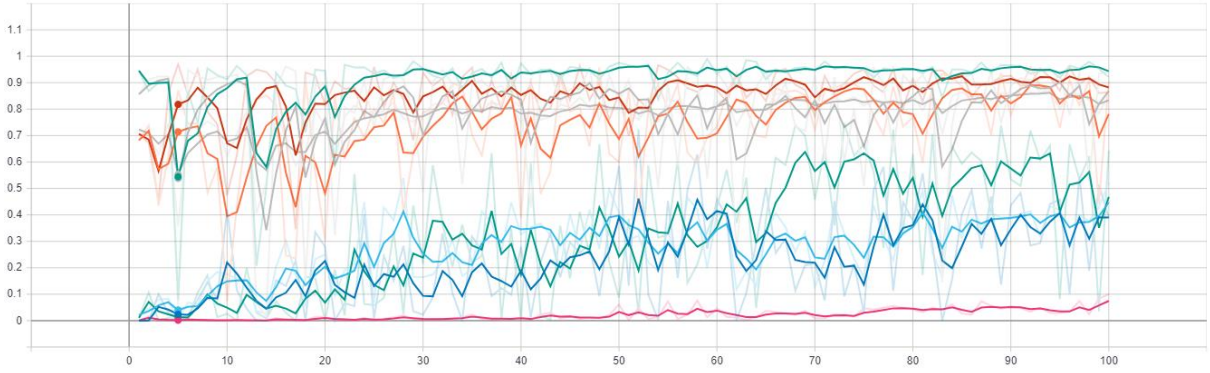

Figura 3.57: Leyenda asociada a validación en partición 4 - experimento 2.

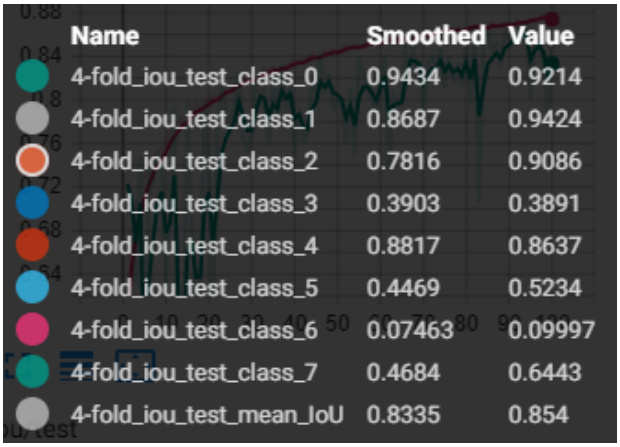

Figura 3.58: Leyenda asociada a validación en partición 4 - experimento 2.

Figura 3.59: Leyenda asociada a validación en partición 5 - experimento 2.

Name	Smoothed	Value
5-fold_iou_test_class_0	0.9382	0.9372
5-fold_iou_test_class_1	0.4444	0.5023
5-fold_iou_test_class_2	0.7842	0.8072
5-fold_iou_test_class_3	0.3059	0.2853
5-fold_iou_test_class_4	0.8528	0.8627
5-fold_iou_test_class_5	0.1592	0.1962
5-fold_iou_test_class_6	0.05351	0.05322
5-fold_iou_test_class_7	0.5278	0.6628
5-fold_iou_test_mean_IoU	0.8288	0.8354

Figura 3.60: Leyenda asociada a validación en partición 5 - experimento 2.

En el caso de validación, verdadero objeto de interés pues muestra de forma directa la capacidad de generalización de la red, las clases en las que se obtienen los mejores resultados son en 0, 2 y 4, siendo estas: terreno artificial, vegetación y edificios respectivamente. Los objetos que se agrupan en estas categorías constan de características geométricas muy específicas, así como valores de intensidad muy característicos.

En tanto, el canal de intensidad complementa a la geometría, y permite que la red realice predicciones de una forma más óptima que utilizando únicamente los canales de información geométrica.

3.3.2.3 Precisión y coste promedio k-Fold

En última instancia, se consideran los resultados promedio de las 5 particiones de entrenamiento/validación:

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,13
Coste promedio validación	1,29
Precisión promedio entrenamiento	87,67%
Precisión promedio validación	77,4%

Tabla 24: Resultados promedio del experimento 2.

3.3.3 Experimento 3

En este experimento se consideran únicamente los canales geométricos y el color asociado a cada punto en el espacio de color RGB. Cada canal (r, g, b) está normalizado al rango $[0, 1]$, y el objetivo es evaluar la influencia del color en comparación al canal geométrico.

3.3.3.1 Precisión y coste general

En primer lugar, se analiza la precisión obtenida en las 5 particiones de entrenamiento/validación.

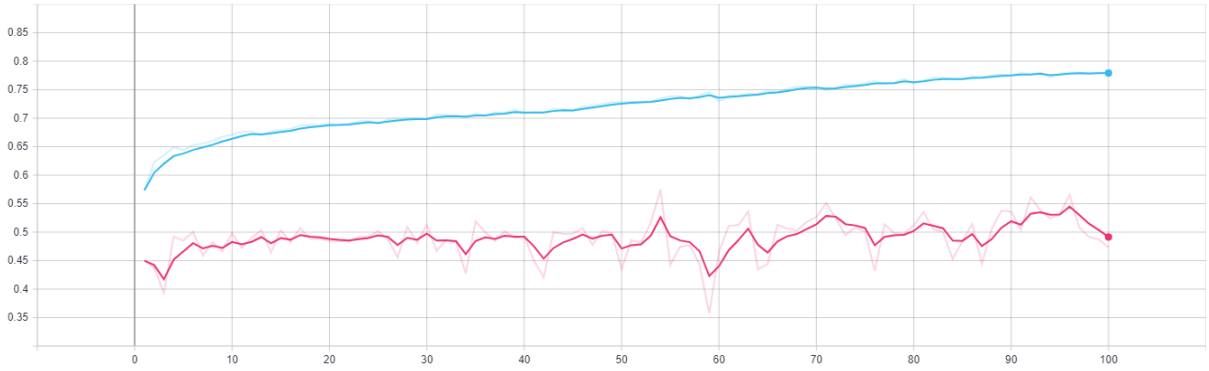

Figura 3.61: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 - experimento 3.

En la primera partición, tan solo se consigue un 50% de precisión en validación, lo cual es un resultado bastante bajo.

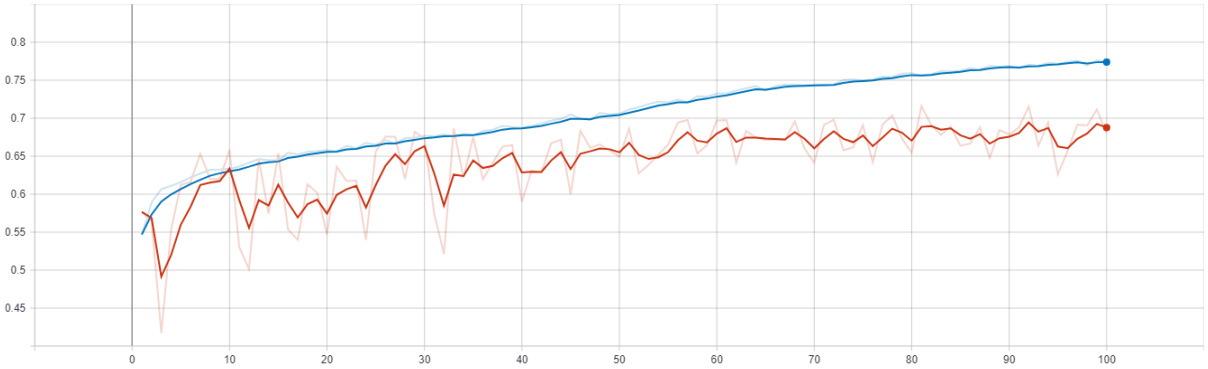

Figura 3.62: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 3.

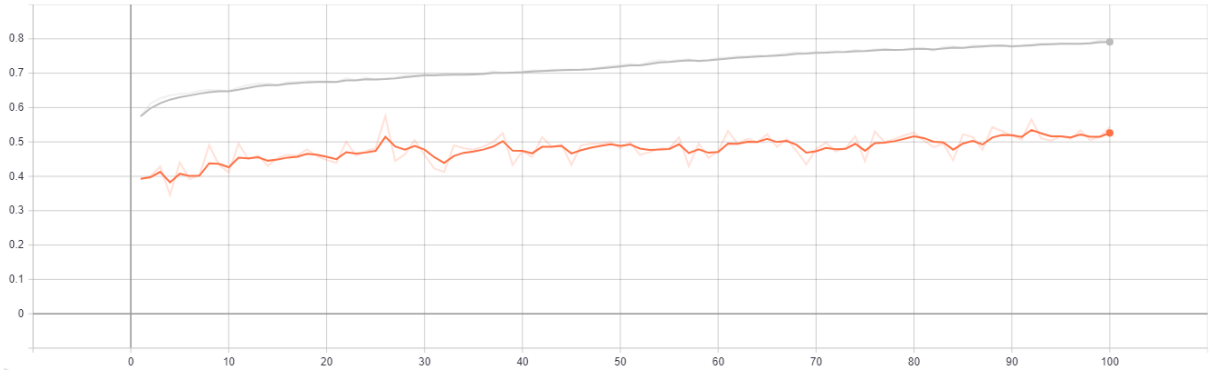

Figura 3.63: Precisión entrenamiento (gris) vs. validación (naranja) en partición 3 - experimento 3.

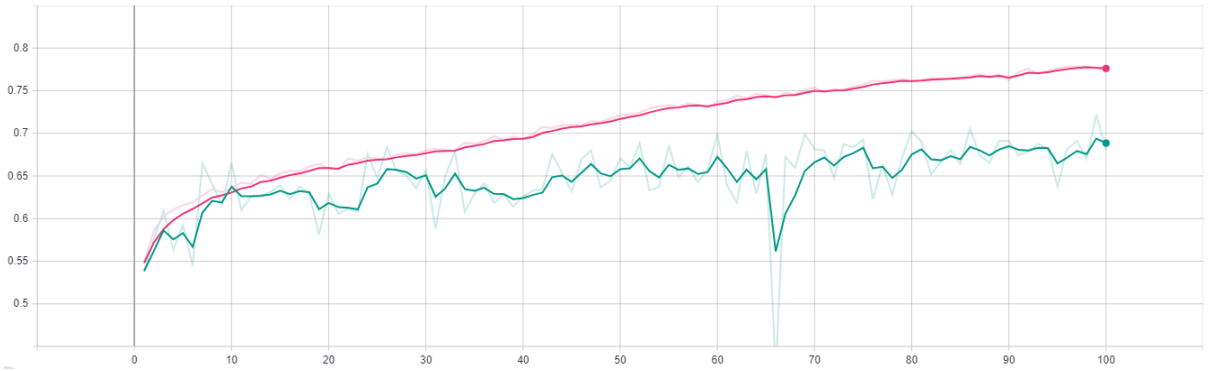

Figura 3.64: Precisión entrenamiento (rosa) vs. validación (verde) en partición 4 - experimento 3.

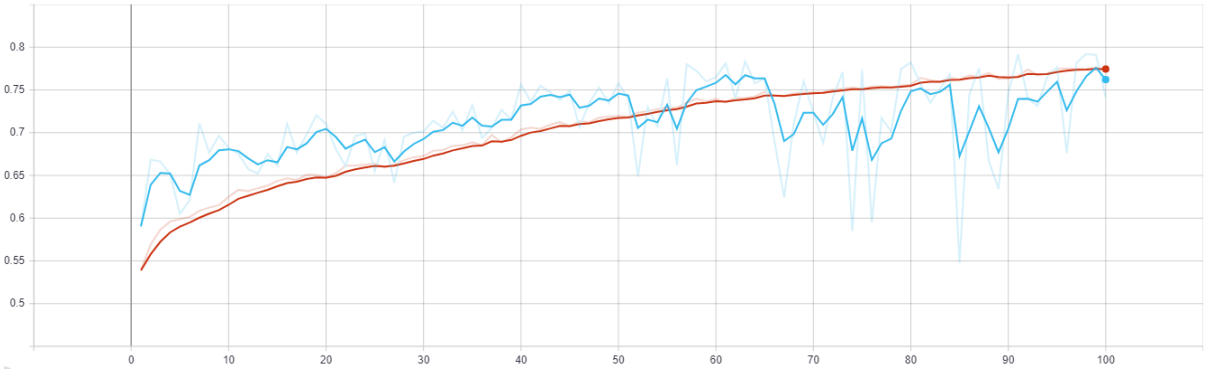

Figura 3.65: Precisión entrenamiento (rojo) vs. validación (azul) en partición 5 - experimento 3.

En el caso del entrenamiento, se consigue una precisión aproximada de entre un 75% y un 80%, mientras que en el caso de la validación se consigue un rendimiento más variable, entre 50% y 70%. Esto se debe a que la naturaleza de los datos de entrenamiento varía en las diferentes particiones, esto es, en algunas particiones se han conseguido conjuntos de datos de entrenamiento más variados con los que la red neuronal es capaz de entrenarse de forma más óptima.

Tras el análisis de la precisión obtenida, se procede a incluir la evolución de los valores de coste tanto en entrenamiento como en validación, lo que nos dará información sobre el posible sobreajuste.

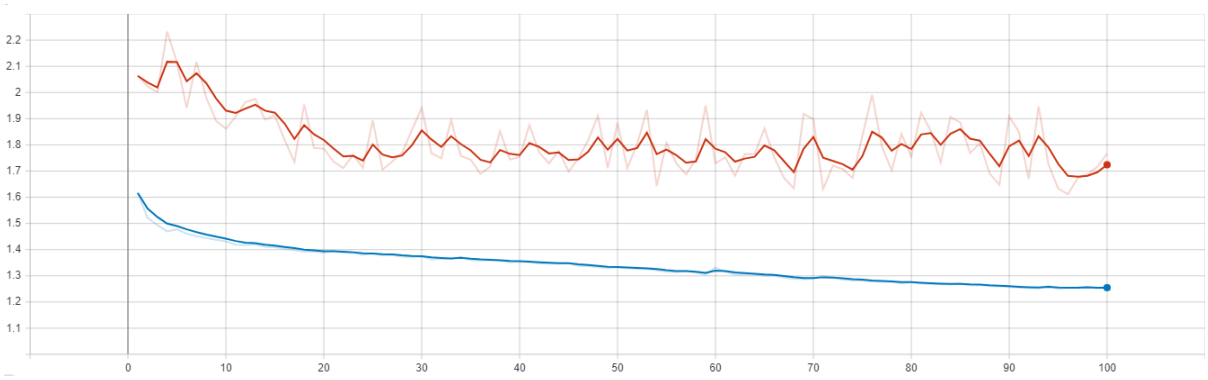

Figura 3.66: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 3.

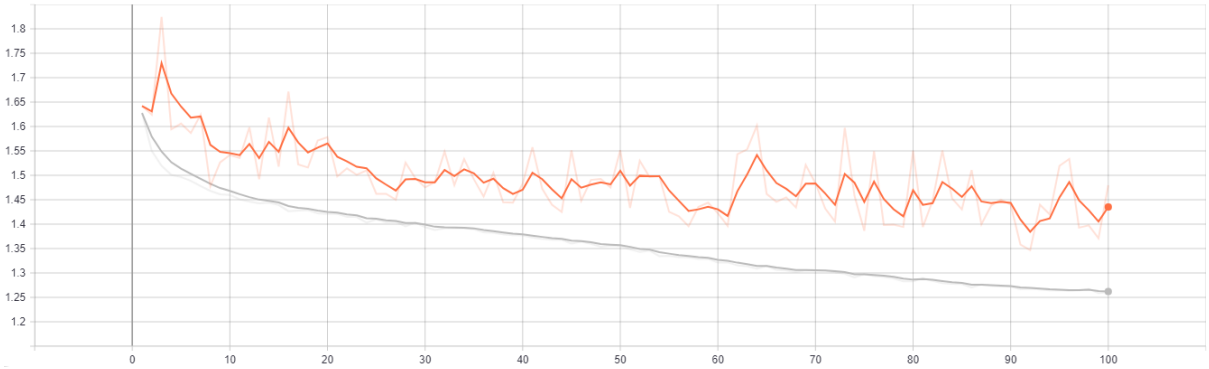

Figura 3.67: Coste de entrenamiento (gris) vs. validación (naranja) en partición 2 - experimento 3.

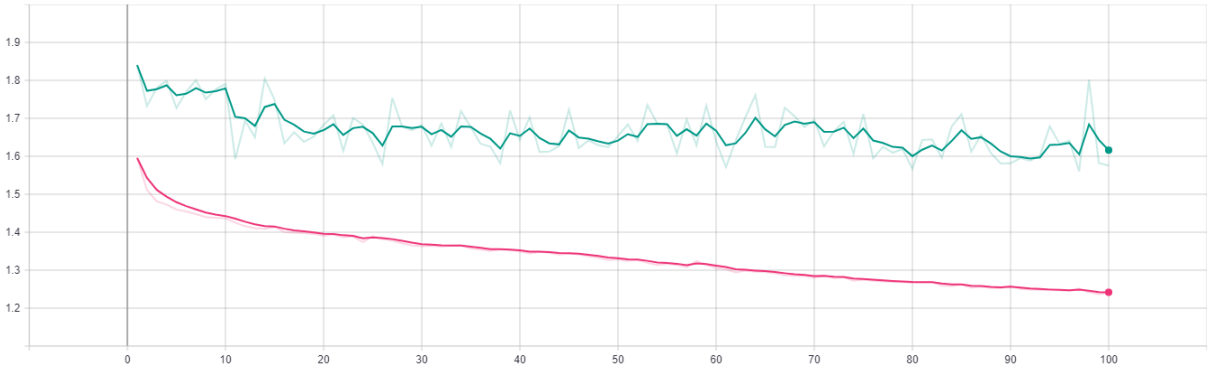

Figura 3.68: Coste de entrenamiento (rosa) vs. validación (verde) en partición 3 - experimento 3.

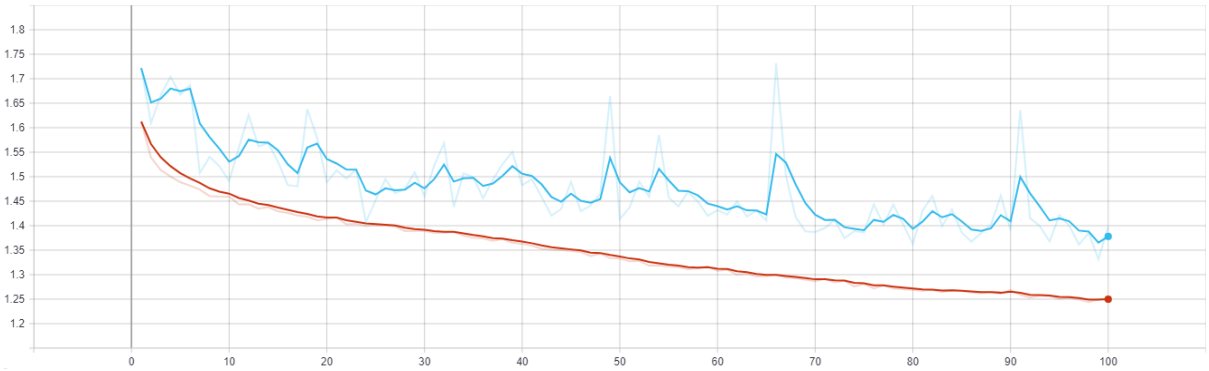

Figura 3.69: Coste de entrenamiento (rojo) vs. validación (azul) en partición 4 - experimento 3.

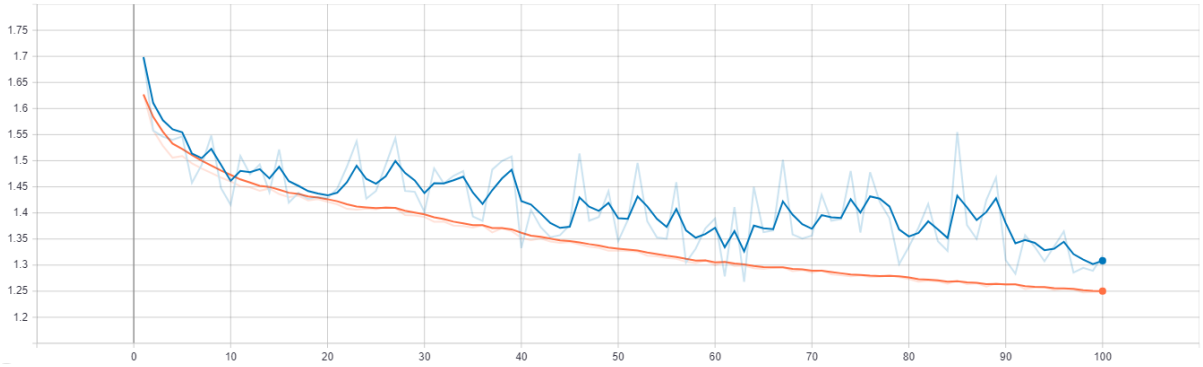

Figura 3.70: Coste de entrenamiento (naranja) vs. validación (azul) en partición 5 - experimento 3.

Observando las gráficas anteriores de entrenamiento *versus* validación en términos de coste, puede apreciarse que no ha aparecido en ningún momento sobreajuste (no existe divergencia entre ambas gráficas en ninguna partición), con lo que el modelo ha sido entrenado de forma correcta.

3.3.3.2 IoU por clase

A continuación, se analiza la precisión obtenida en cada una de las clases del problema tanto en entrenamiento como en validación.

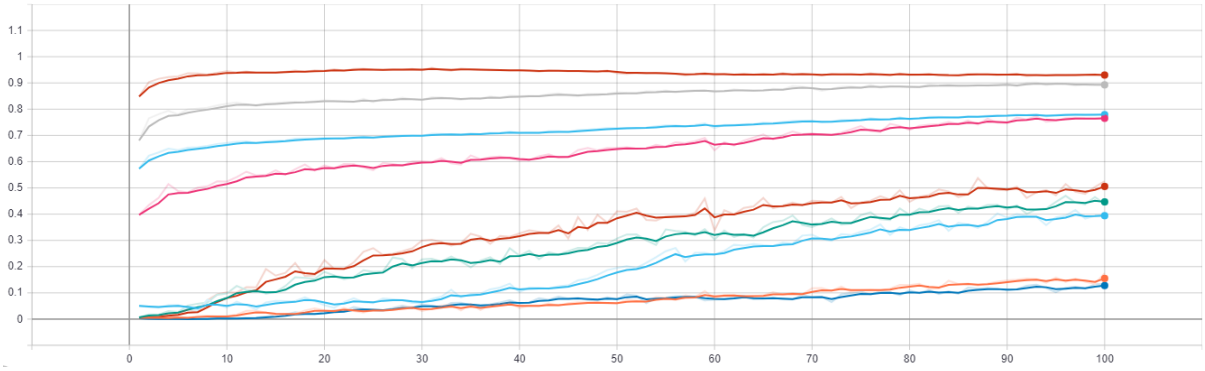

Figura 3.71: IoU asociada al entrenamiento en partición 1 - experimento 3.

Name	Smoothed	Value
1-fold_iou_train_class_0	0.9301	0.9279
1-fold_iou_train_class_1	0.3941	0.3963
1-fold_iou_train_class_2	0.7653	0.7673
1-fold_iou_train_class_3	0.4467	0.4398
1-fold_iou_train_class_4	0.8927	0.8918
1-fold_iou_train_class_5	0.1551	0.175
1-fold_iou_train_class_6	0.1279	0.1341
1-fold_iou_train_class_7	0.5056	0.5252
1-fold_iou_train_mean_IoU	0.7792	0.7797

Figura 3.72: Leyenda asociada al entrenamiento en partición 1 - experimento 3.

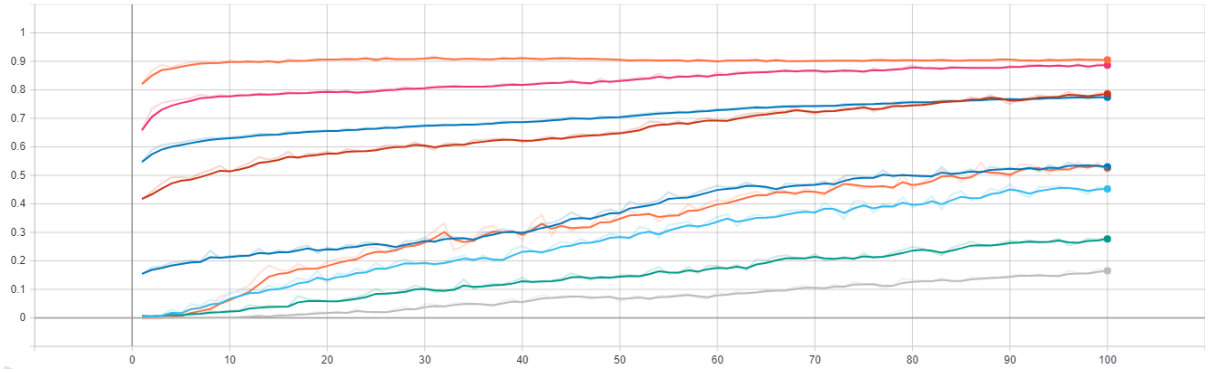

Figura 3.73: IoU asociada al entrenamiento en partición 2 - experimento 3.

Name	Smoothed	Value
2-fold_iou_train_class_0	0.9051	0.9044
2-fold_iou_train_class_1	0.5298	0.5241
2-fold_iou_train_class_2	0.7856	0.7913
2-fold_iou_train_class_3	0.4524	0.4552
2-fold_iou_train_class_4	0.8864	0.8875
2-fold_iou_train_class_5	0.277	0.2853
2-fold_iou_train_class_6	0.1653	0.1715
2-fold_iou_train_class_7	0.5251	0.5106
2-fold_iou_train_mean_IoU	0.7739	0.7741

Figura 3.74: Leyenda asociada al entrenamiento en partición 2 - experimento 3.

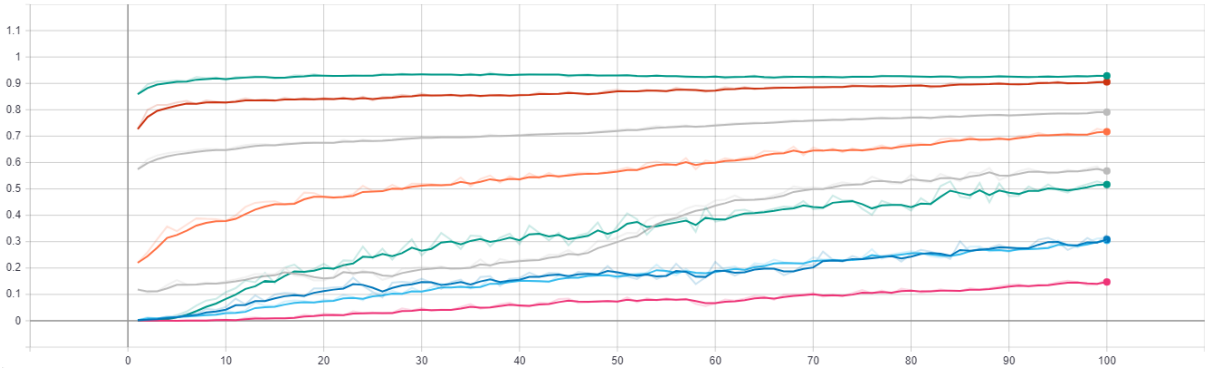

Figura 3.75: IoU asociada al entrenamiento en partición 3 - experimento 3.

Name	Smoothed	Value
3-fold_iou_train_class_0	0.9287	0.9294
3-fold_iou_train_class_1	0.568	0.556
3-fold_iou_train_class_2	0.7168	0.7204
3-fold_iou_train_class_3	0.3086	0.3268
3-fold_iou_train_class_4	0.9055	0.908
3-fold_iou_train_class_5	0.3047	0.3145
3-fold_iou_train_class_6	0.1471	0.1582
3-fold_iou_train_class_7	0.5165	0.5186
3-fold_iou_train_mean_IoU	0.791	0.7921

Figura 3.76: Leyenda asociada al entrenamiento en partición 3 - experimento 3.

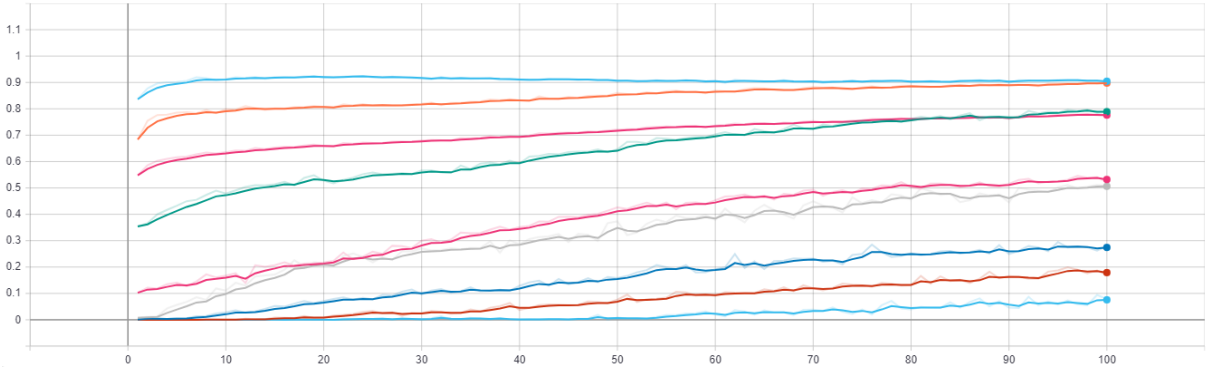

Figura 3.77: IoU asociada al entrenamiento en partición 4 - experimento 3.

Name	Smoothed	Value
4-fold_iou_train_class_0	0.9046	0.9018
4-fold_iou_train_class_1	0.532	0.5232
4-fold_iou_train_class_2	0.7892	0.789
4-fold_iou_train_class_3	0.5063	0.508
4-fold_iou_train_class_4	0.8977	0.8992
4-fold_iou_train_class_5	0.2743	0.2799
4-fold_iou_train_class_6	0.179	0.1708
4-fold_iou_train_class_7	0.07567	0.0783
4-fold_iou_train_mean_IoU	0.7761	0.7746

Figura 3.78: Leyenda asociada al entrenamiento en partición 4 - experimento 3.

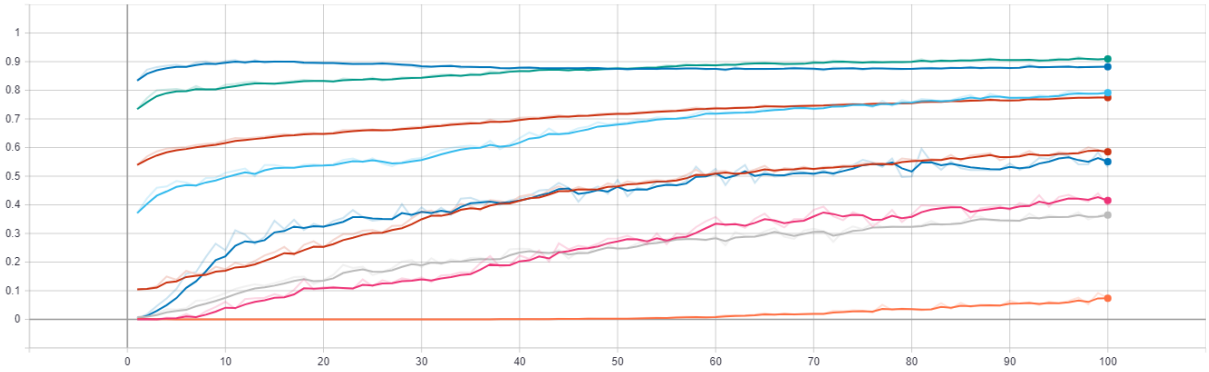

Figura 3.79: IoU asociada al entrenamiento en partición 5 - experimento 3.

Name	Smoothed	Value
5-fold_iou_train_class_0	0.8821	0.8829
5-fold_iou_train_class_1	0.5851	0.5779
5-fold_iou_train_class_2	0.7914	0.7959
5-fold_iou_train_class_3	0.4152	0.3981
5-fold_iou_train_class_4	0.9094	0.9111
5-fold_iou_train_class_5	0.3643	0.372
5-fold_iou_train_class_6	0.07366	0.0741
5-fold_iou_train_class_7	0.5506	0.5315
5-fold_iou_train_mean_IoU	0.7744	0.774

Figura 3.80: Leyenda asociada al entrenamiento en partición 5 - experimento 3.

Al igual que ocurría en el experimento anterior, es en las clases 0, 2 y 4 donde se alcanza la precisión máxima, puesto que el color es un canal de información muy similar al canal de intensidad, y los objetos que se incluyen en estas categorías

(terreno artificial, vegetación baja y edificio) comparten una gama de colores muy concreta.

Resultados del proceso de validación:

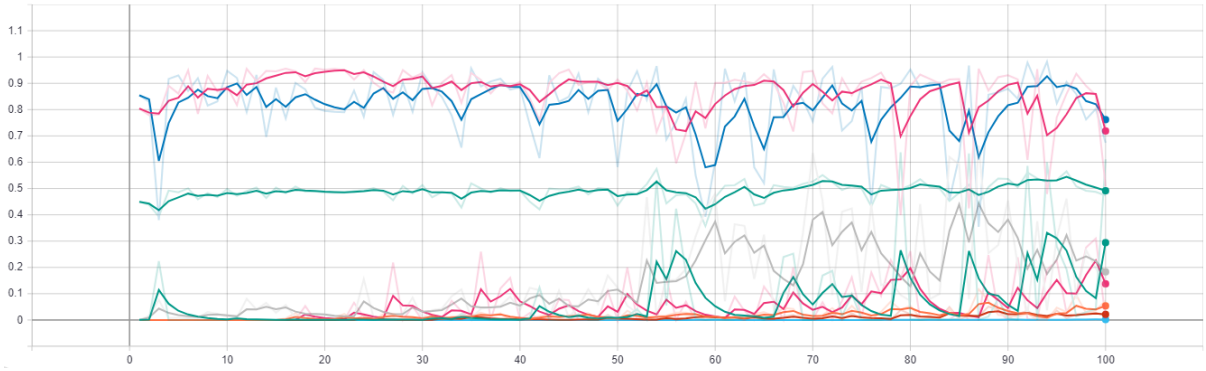

Figura 3.81: IoU asociada a validación en partición 1 - experimento 3.

Name	Smoothed	Value
1-fold_iou_test_class_0	0.7188	0.508
1-fold_iou_test_class_1	0.2942	0.6109
1-fold_iou_test_class_2	0.183	0.1214
1-fold_iou_test_class_3	0.05376	0.07361
1-fold_iou_test_class_4	0.7619	0.6736
1-fold_iou_test_class_5	0.0213	0.01624
1-fold_iou_test_class_6	1.627e-3	1.1004e-3
1-fold_iou_test_class_7	0.1376	4.5119e-3
1-fold_iou_test_mean_IoU	0.4915	0.4731

Figura 3.82: Leyenda asociada a validación en partición 1 - experimento 3.

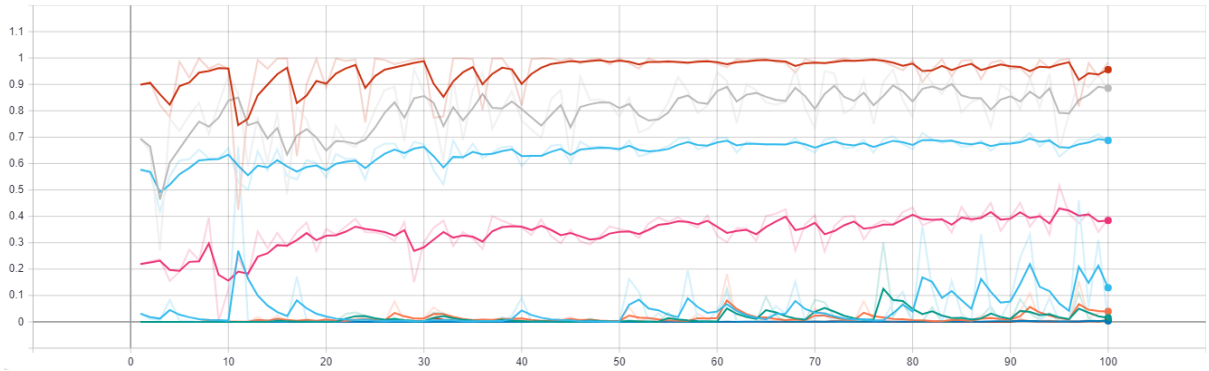

Figura 3.83: IoU asociada a validación en partición 2 - experimento 3.

Name	Smoothed	Value
2-fold_iou_test_class_0	0.9559	0.9835
2-fold_iou_test_class_1	0.1292	4.0373e-3
2-fold_iou_test_class_2	0.3843	0.3898
2-fold_iou_test_class_3	0.01534	6.2399e-3
2-fold_iou_test_class_4	0.8854	0.8763
2-fold_iou_test_class_5	0.03942	0.03697
2-fold_iou_test_class_6	3.7942e-3	4.606e-3
2-fold_iou_test_class_7	5.5368e-3	0.01116
2-fold_iou_test_mean_IoU	0.6876	0.6803

Figura 3.84: Leyenda asociada a validación en partición 2 – experimento 3.

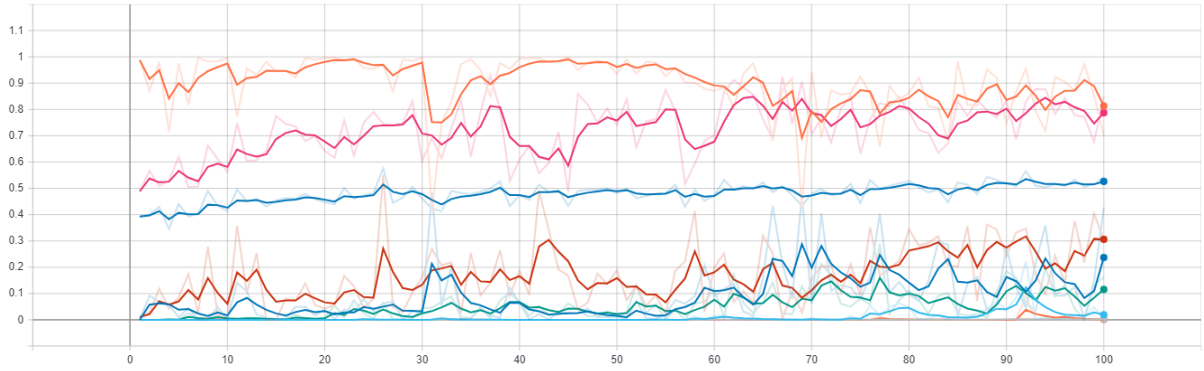

Figura 3.85: IoU asociada a validación en partición 3 - experimento 3.

Name	Smoothed	Value
3-fold_iou_test_class_0	0.8131	0.7006
3-fold_iou_test_class_1	0.2368	0.4277
3-fold_iou_test_class_2	0.306	0.3032
3-fold_iou_test_class_3	0.01921	5.7898e-3
3-fold_iou_test_class_4	0.7873	0.8483
3-fold_iou_test_class_5	0.1156	0.1628
3-fold_iou_test_class_6	2.4509e-3	3.5593e-3
3-fold_iou_test_class_7	1.3978e-3	0
3-fold_iou_test_mean_IoU	0.5262	0.5427

Figura 3.86: Leyenda asociada a validación en partición 3 - experimento 3.

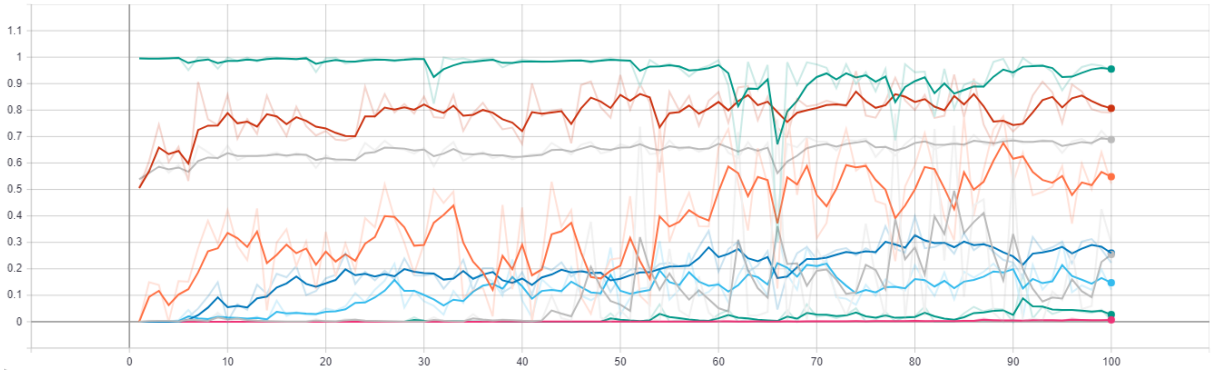

Figura 3.87: IoU asociada a validación en partición 4 - experimento 3.

	Name	Smoothed	Value
	4-fold_iou_test_class_0	0.9552	0.9487
	4-fold_iou_test_class_1	0.2543	0.2973
	4-fold_iou_test_class_2	0.5482	0.5215
	4-fold_iou_test_class_3	0.2595	0.2229
	4-fold_iou_test_class_4	0.8067	0.7906
	4-fold_iou_test_class_5	0.1474	0.1208
	4-fold_iou_test_class_6	7.039e-3	8.8087e-3
	4-fold_iou_test_class_7	0.0269	5.4784e-3
	4-fold_iou_test_mean_IoU	0.6886	0.6806

Figura 3.88: Leyenda asociada a validación en partición 4 - experimento 3.

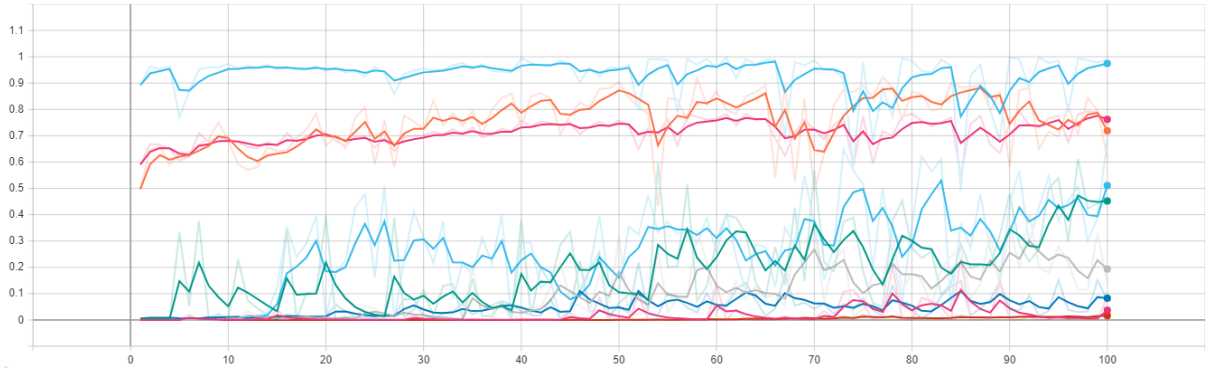

Figura 3.89: IoU asociada a validación en partición 5 - experimento 3.

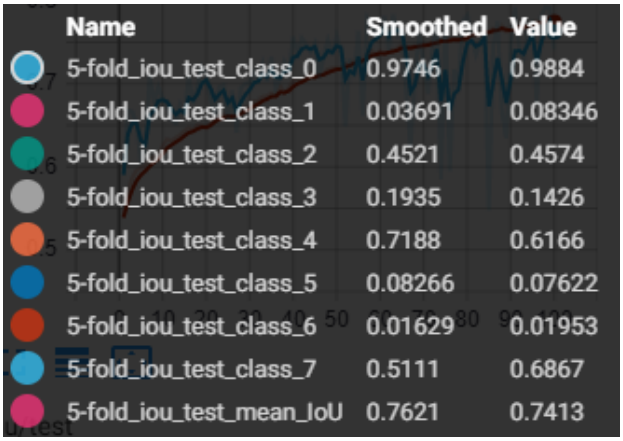

Figura 3.90: Leyenda asociada a validación en partición 5 - experimento 3.

En este experimento, la precisión máxima se alcanza en las clases 0, 1, 2 y 4 (terreno artificial, terreno natural, vegetación alta y edificios). A diferencia del experimento anterior, donde la precisión máxima no se dividía entre los dos tipos de terreno, sino que siempre se conseguía ésta en terreno artificial, en este caso se utiliza el color a diferencia de la intensidad. El color es un canal mucho más variable que el canal de intensidad.

En tanto, dos superficies pueden tener un color completamente distinto, y sin embargo tener un valor de intensidad igual o muy parecido. Además, dos tonos de color que visualmente son similares pueden ser radicalmente diferentes en términos de valores (r, g, b). Esto afecta directamente sobre la capacidad de aprendizaje y generalización del modelo.

3.3.3.3 Precisión y coste promedio k-Fold

En última instancia para este experimento, se añade una tabla de resultados promedio en las 5 particiones de entrenamiento/validación:

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,251
Coste promedio validación	1,507
Precisión promedio entrenamiento	77,89%
Precisión promedio validación	62,36%

Tabla 25: Resultados promedio experimento 3.

3.3.4 Experimento 4

Al momento de ejecutar este experimento se tienen en cuenta los dos canales de información incorporados en los experimentos anteriores, pero de forma conjunta en este caso. Así, se ejecuta el entrenamiento teniendo en cuenta los canales geométricos, intensidad y color (r, g, b).

3.3.4.1 Precisión y coste general

En primer lugar, se analiza la precisión máxima obtenida:

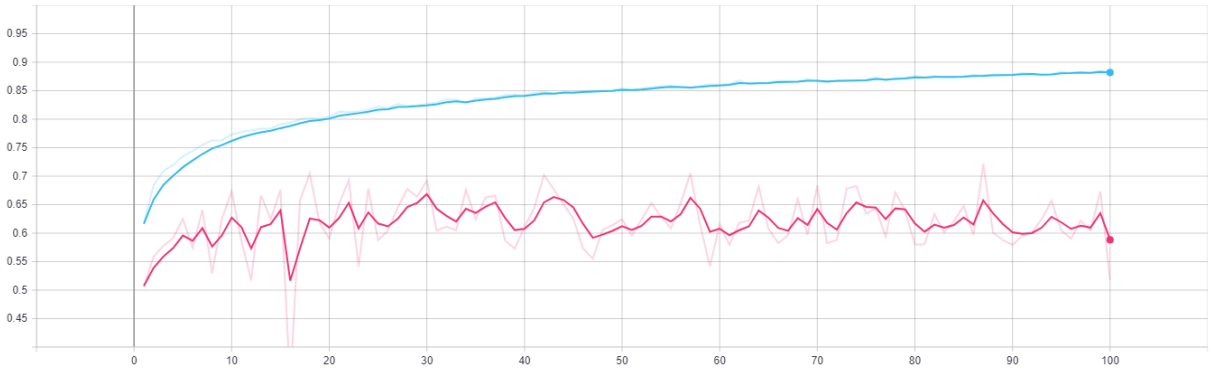

Figura 3.91: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 - experimento 4.

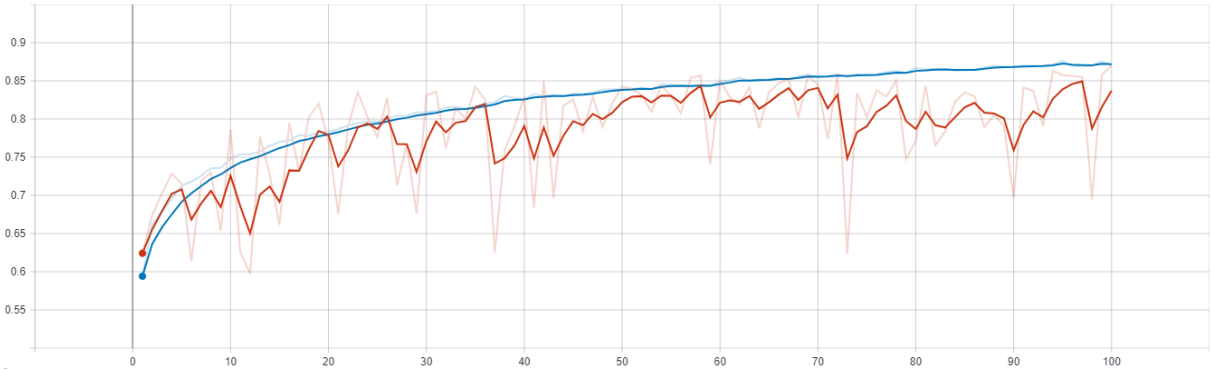

Figura 3.92: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 4.

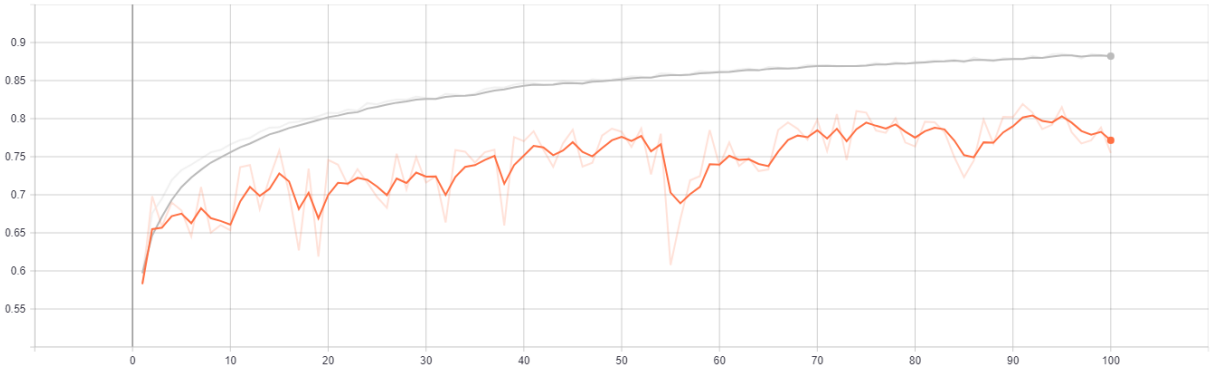

Figura 3.93: Precisión entrenamiento (gris) vs. validación (naranja) en partición 3 - experimento 4.

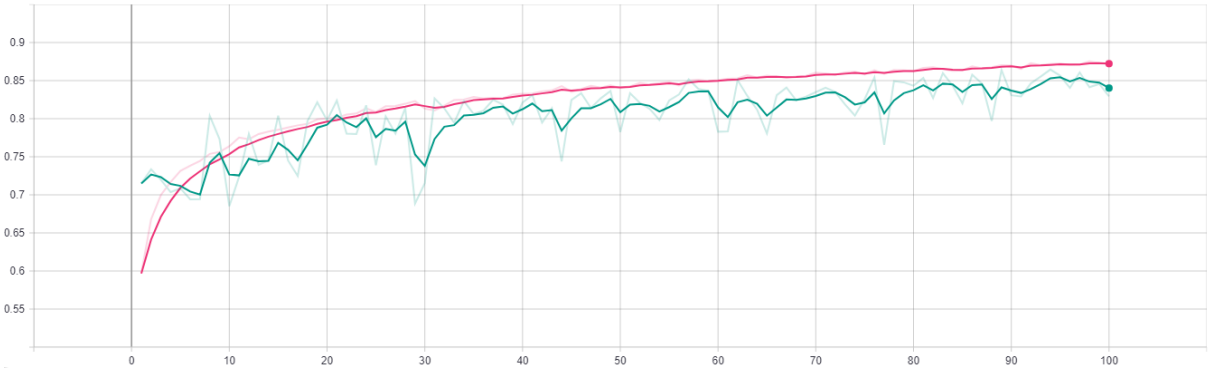

Figura 3.94: Precisión entrenamiento (rosa) vs. validación (verde) en partición 4 - experimento 4.

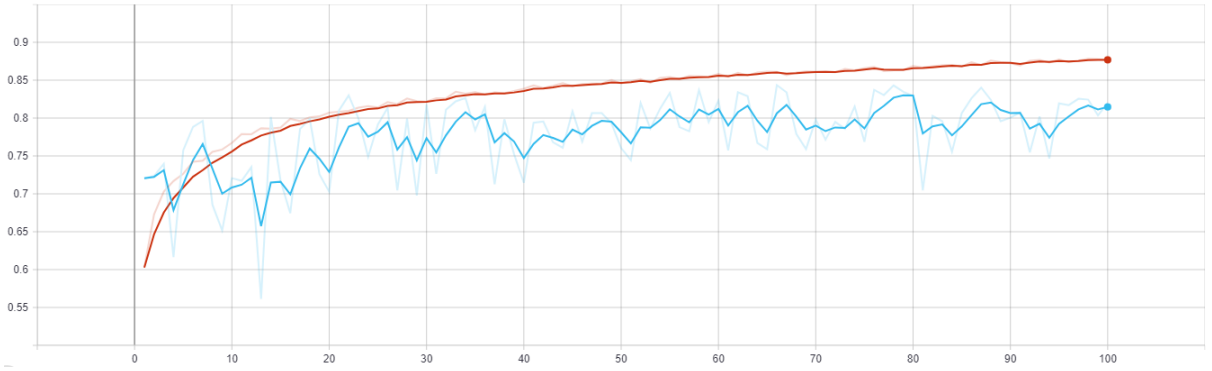

Figura 3.95: Precisión entrenamiento (rojo) vs. validación (azul) en partición 5 - experimento 4.

Como puede observarse, la tendencia de los resultados es muy similar a la obtenida en 3.3.2. Esto sucede por el hecho de que los resultados de utilizar el canal de intensidad dominan a los resultados de utilizar únicamente el canal de color, esto es, el canal de intensidad aporta más información de interés que el canal de color. Por

tanto, la combinación de color e intensidad no supone una diferencia significativa con respecto a usar únicamente intensidad.

En segundo lugar, se analiza la tendencia de las gráficas de coste para distinguir la aparición de sobreajuste.

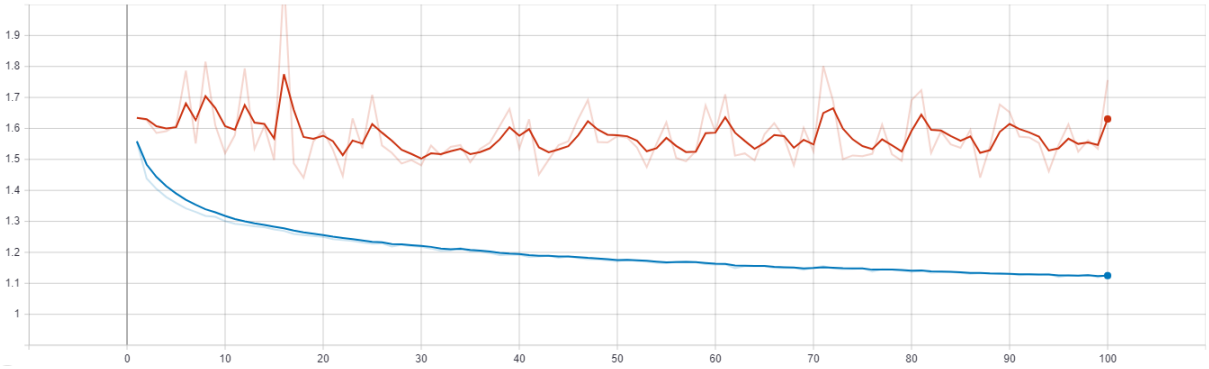

Figura 3.96: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 4.

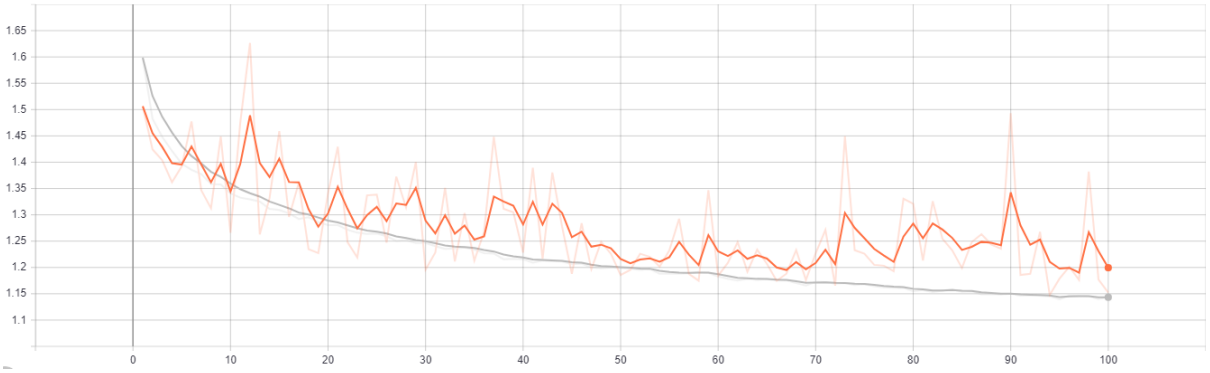

Figura 3.97: Coste de entrenamiento (gris) vs. validación (naranja) en partición 2 - experimento 4.

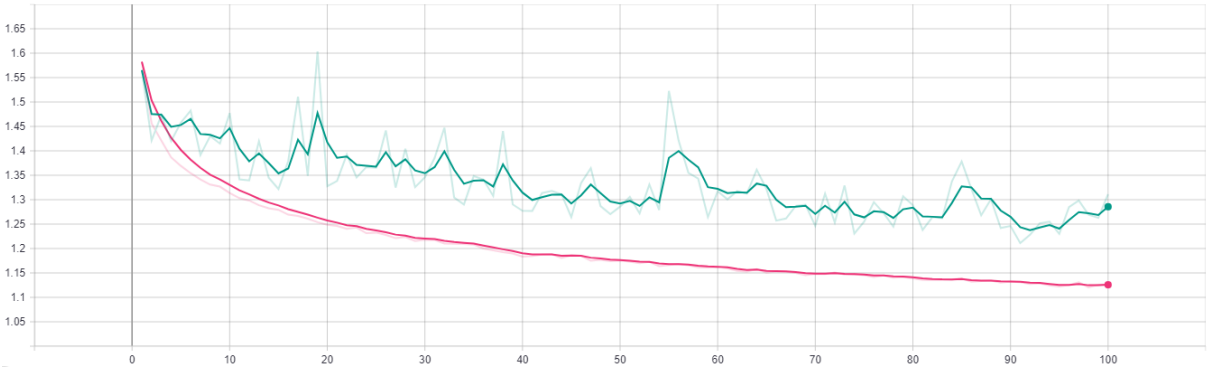

Figura 3.98: Coste de entrenamiento (rosa) vs. validación (verde) en partición 3 - experimento 4.

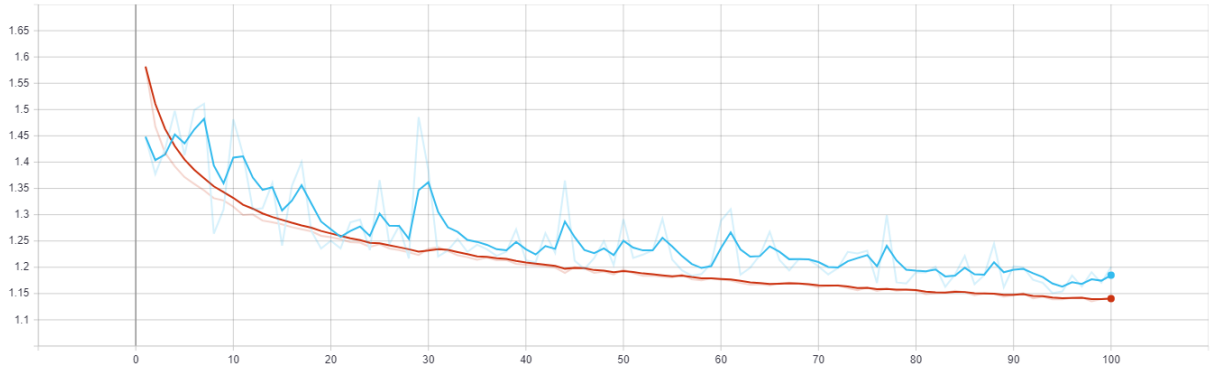

Figura 3.99: Coste de entrenamiento (rojo) vs. validación (azul) en partición 4 - experimento 4.

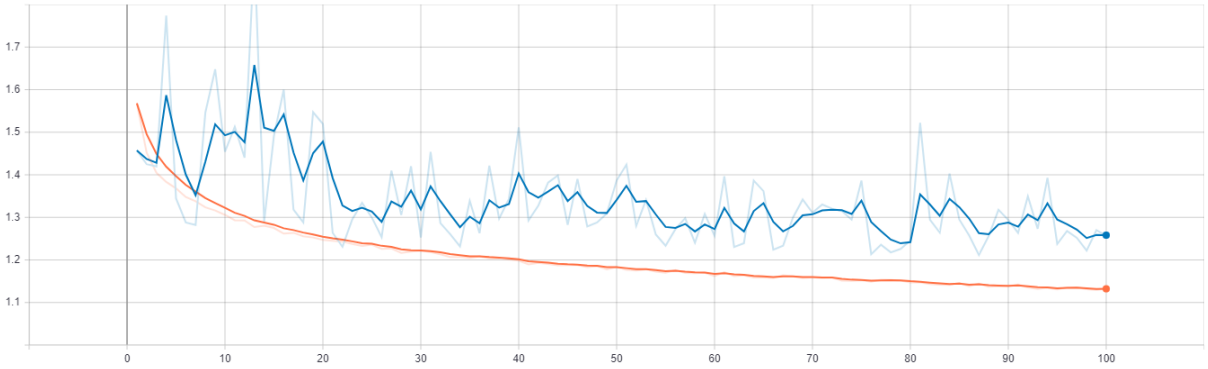

Figura 3.100: Coste de entrenamiento (azul) vs. validación (rojo) en partición 5 - experimento 4.

Puede observarse que, aunque la gráfica de coste asociada a la validación disminuye de una forma más irregular a la gráfica de coste asociada al entrenamiento, la tendencia siempre es a decrecer y no hay ningún punto de inflexión en el que una gráfica comience a divergir de la otra.

3.3.4.2 IoU por clase

En esta sección se adjuntan las gráficas de precisión en cada una de las categorías del *dataset* utilizado:

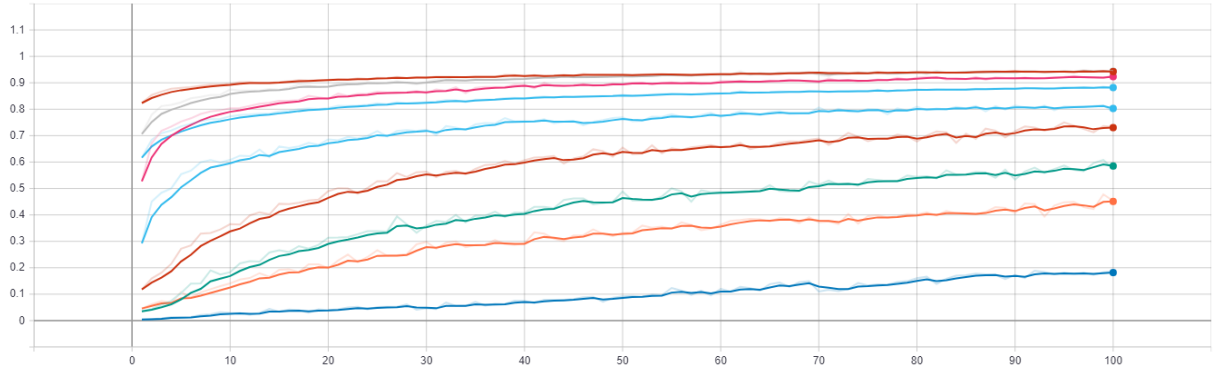

Figura 3.101: IoU asociada al entrenamiento en partición 1 - experimento 4.

Name	Smoothed	Value
1-fold_iou_train_class_0	0.9423	0.9407
1-fold_iou_train_class_1	0.8029	0.7892
1-fold_iou_train_class_2	0.9232	0.9286
1-fold_iou_train_class_3	0.5849	0.5753
1-fold_iou_train_class_4	0.9438	0.9439
1-fold_iou_train_class_5	0.451	0.4534
1-fold_iou_train_class_6	0.1817	0.1857
1-fold_iou_train_class_7	0.7303	0.7322
1-fold_iou_train_mean_IoU	0.882	0.8805

Figura 3.102: Leyenda asociada al entrenamiento en partición 1 - experimento 4.

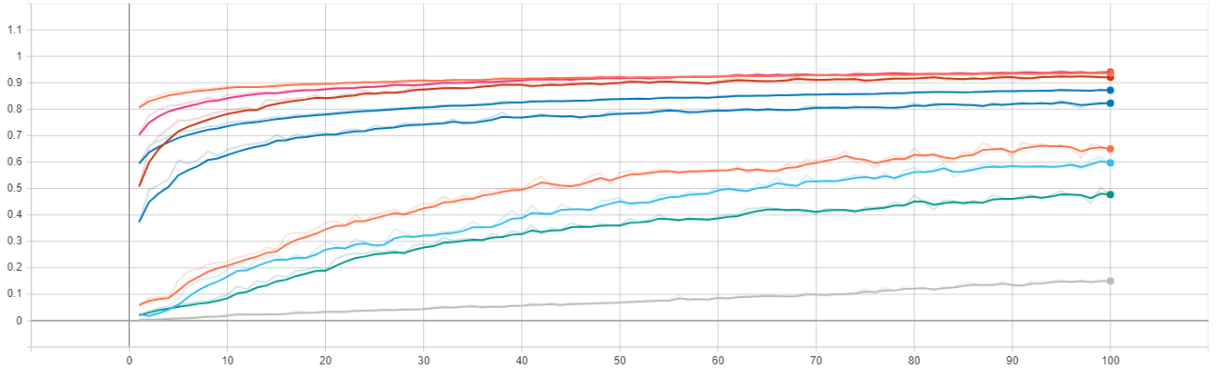

Figura 3.103: IoU asociada al entrenamiento en partición 2 - experimento 4.

Name	Smoothed	Value
2-fold_iou_train_class_0	0.9353	0.9331
2-fold_iou_train_class_1	0.8227	0.8249
2-fold_iou_train_class_2	0.9209	0.9206
2-fold_iou_train_class_3	0.597	0.5882
2-fold_iou_train_class_4	0.9404	0.9417
2-fold_iou_train_class_5	0.477	0.4724
2-fold_iou_train_class_6	0.15	0.1515
2-fold_iou_train_class_7	0.6501	0.6422
2-fold_iou_train_mean_IoU	0.8717	0.8706

Figura 3.104: Leyenda asociada al entrenamiento en partición 2 - experimento 4.

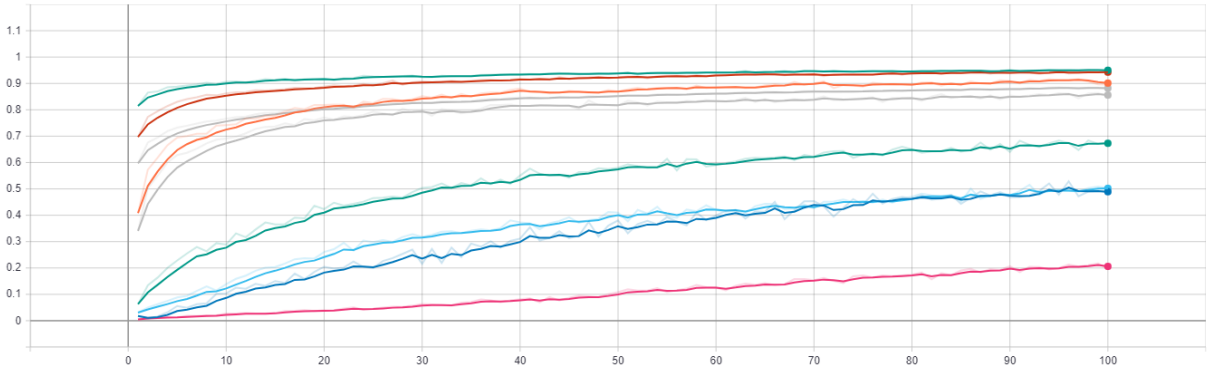

Figura 3.105: IoU asociada al entrenamiento en partición 3 - experimento 4.

Name	Smoothed	Value
3-fold_iou_train_class_0	0.9499	0.9487
3-fold_iou_train_class_1	0.8559	0.8507
3-fold_iou_train_class_2	0.9013	0.8954
3-fold_iou_train_class_3	0.4892	0.4859
3-fold_iou_train_class_4	0.9428	0.9438
3-fold_iou_train_class_5	0.5022	0.5037
3-fold_iou_train_class_6	0.206	0.1981
3-fold_iou_train_class_7	0.6729	0.6759
3-fold_iou_train_mean_IoU	0.8821	0.8807

Figura 3.106: Leyenda asociada al entrenamiento en partición 3 - experimento 4.

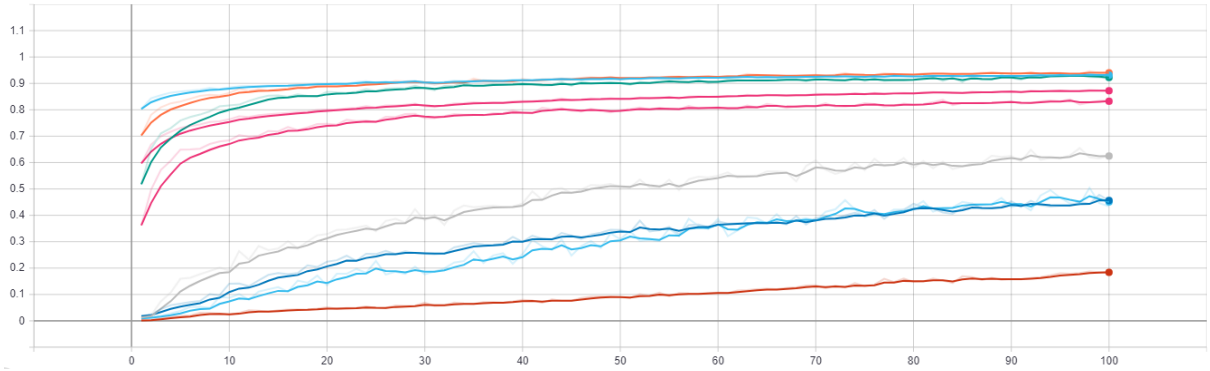

Figura 3.107: IoU asociada al entrenamiento en partición 4 - experimento 4.

Name	Smoothed	Value
4-fold_iou_train_class_0	0.9313	0.9302
4-fold_iou_train_class_1	0.8324	0.8353
4-fold_iou_train_class_2	0.9225	0.9179
4-fold_iou_train_class_3	0.6247	0.6262
4-fold_iou_train_class_4	0.9401	0.94
4-fold_iou_train_class_5	0.4563	0.4553
4-fold_iou_train_class_6	0.1838	0.1855
4-fold_iou_train_class_7	0.4509	0.434
4-fold_iou_train_mean_IoU	0.8722	0.8714

Figura 3.108: Leyenda asociada al entrenamiento en partición 4 - experimento 4.

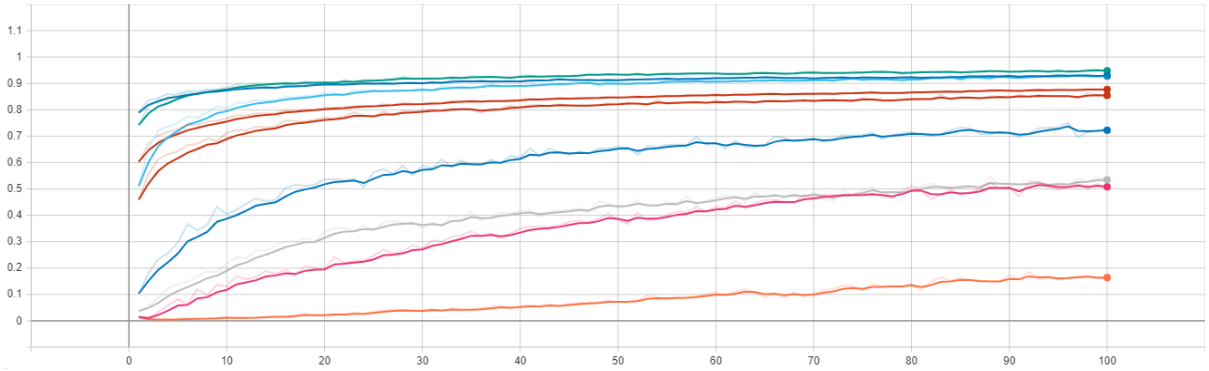

Figura 3.109: IoU asociada al entrenamiento en partición 5 - experimento 4.

Figura 3.110: Leyenda asociada al entrenamiento en partición 5 - experimento 4.

La red consigue una precisión superior al 70% en las clases que albergan las figuras geométricas más sencillas, como son 0 (terreno artificial), 1 (terreno natural), 2 (vegetación alta), 4 (edificios) y 7 (coches). El problema aparece principalmente con las categorías 3, 5 y 6 pues se trata de las categorías más geométricamente más complejas.

A continuación, se adjuntan los resultados obtenidos en la validación:

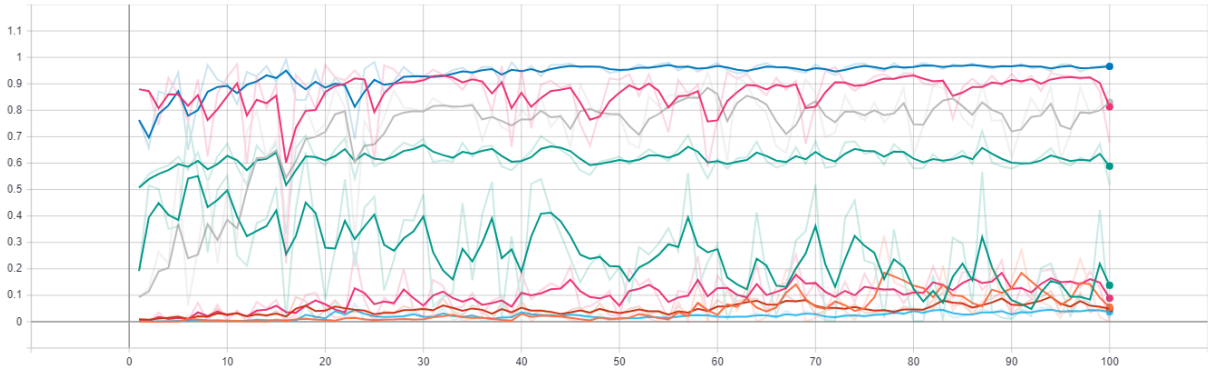

Figura 3.111: IoU asociada a validación en partición 1 - experimento 4.

Name	Smoothed	Value
1-fold_iou_test_class_0	0.8132	0.6776
1-fold_iou_test_class_1	0.1376	0.01493
1-fold_iou_test_class_2	0.83	0.8757
1-fold_iou_test_class_3	0.05483	2.2066e-5
1-fold_iou_test_class_4	0.9659	0.9706
1-fold_iou_test_class_5	0.04855	0.0371
1-fold_iou_test_class_6	0.03713	0.02877
1-fold_iou_test_class_7	0.08877	0
1-fold_iou_test_mean_IoU	0.5883	0.518

Figura 3.112: Leyenda asociada a validación en partición 1 - experimento 4.

Figura 3.113: IoU asociada a validación en partición 2 - experimento 4.

Name	Smoothed	Value
2-fold_iou_test_class_0	0.9198	0.9538
2-fold_iou_test_class_1	0.7779	0.8903
2-fold_iou_test_class_2	0.8559	0.8621
2-fold_iou_test_class_3	0.1565	0.2002
2-fold_iou_test_class_4	0.9627	0.9717
2-fold_iou_test_class_5	0.08584	0.08871
2-fold_iou_test_class_6	0.05199	0.06829
2-fold_iou_test_class_7	0.3599	0.4617
2-fold_iou_test_mean_IoU	0.8373	0.8701

Figura 3.114: Leyenda asociada a validación en partición 2 - experimento 4.

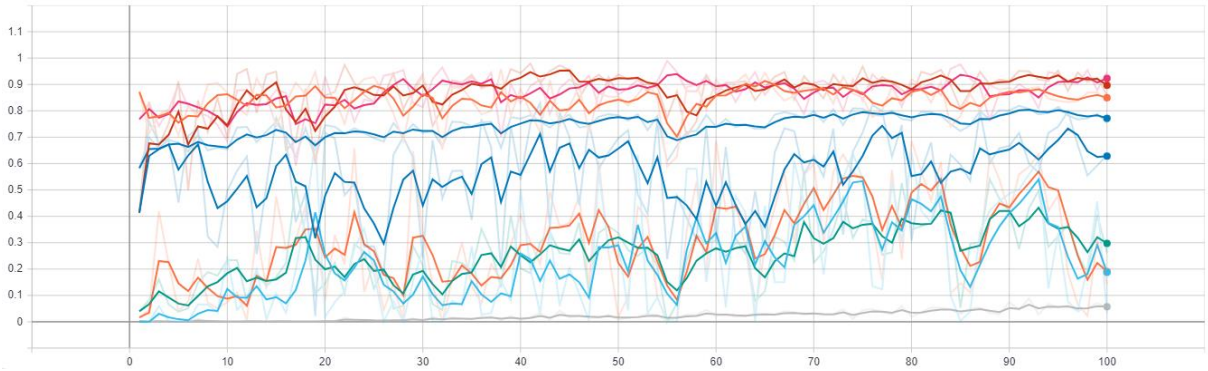

Figura 3.115: IoU asociada a validación en partición 3 - experimento 4.

Name	Smoothed	Value
3-fold_iou_test_class_0	0.8497	0.8339
3-fold_iou_test_class_1	0.6282	0.6331
3-fold_iou_test_class_2	0.8967	0.8598
3-fold_iou_test_class_3	0.1876	0.03187
3-fold_iou_test_class_4	0.9231	0.9467
3-fold_iou_test_class_5	0.2973	0.2617
3-fold_iou_test_class_6	0.05715	0.05436
3-fold_iou_test_class_7	0.1902	0.1411
3-fold_iou_test_mean_IoU	0.7716	0.7549

Figura 3.116: Leyenda asociada a validación en partición 3 - experimento 4.

Figura 3.117: IoU asociada a validación en partición 4 - experimento 4.

Name	Smoothed	Value
4-fold_iou_test_class_0	0.9319	0.8912
4-fold_iou_test_class_1	0.8856	0.9506
4-fold_iou_test_class_2	0.8801	0.8803
4-fold_iou_test_class_3	0.3908	0.4564
4-fold_iou_test_class_4	0.8882	0.8655
4-fold_iou_test_class_5	0.435	0.4814
4-fold_iou_test_class_6	0.06701	0.05761
4-fold_iou_test_class_7	0.4594	0.4737
4-fold_iou_test_mean_IoU	0.8402	0.8293

Figura 3.118: Leyenda asociada a validación en partición 4 - experimento 4.

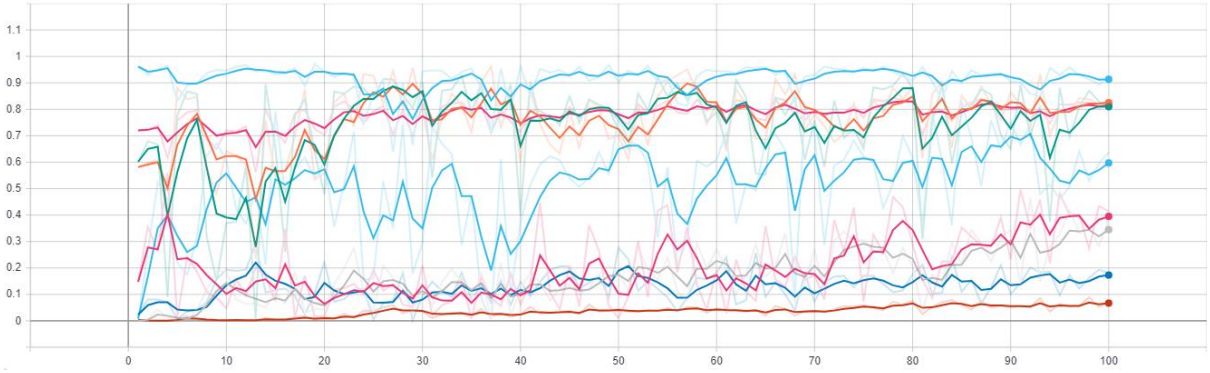

Figura 3.119: IoU asociada a validación en partición 5 - experimento 4.

Name	Smoothed	Value
5-fold_iou_test_class_0	0.9137	0.916
5-fold_iou_test_class_1	0.3949	0.4136
5-fold_iou_test_class_2	0.8097	0.808
5-fold_iou_test_class_3	0.3449	0.3832
5-fold_iou_test_class_4	0.825	0.8305
5-fold_iou_test_class_5	0.1731	0.1816
5-fold_iou_test_class_6	0.067	0.07382
5-fold_iou_test_class_7	0.5972	0.638
5-fold_iou_test_mean_IoU	0.8147	0.8197

Figura 3.120: Leyenda asociada a validación en partición 5 - experimento 4.

3.3.4.3 Precisión y coste promedio k-Fold

Para concluir este experimento, se adjunta una tabla que muestra los resultados promedio en las 5 particiones de entrenamiento y validación.

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,13
Coste promedio validación	1,33
Precisión promedio entrenamiento	87,6%
Precisión promedio validación	75,84%

Tabla 26: Resultados promedio experimento 4.

Como se puede comprobar, es verificable que los resultados son muy similares a los obtenidos en el experimento 2, correspondiente al uso exclusivo del canal de intensidad.

3.3.5 Experimento 5

Para evaluar la influencia de la geometría normalizada al intervalo de caja $[0, 1]$ en cada una de las direcciones del espacio, se realiza un nuevo experimento que verifique únicamente si es posible conseguir un rendimiento mayor usando los canales normalizados como complemento a los canales sin normalizar.

En tanto, no se trata de atributos propios de LiDAR sino de un postproceso, por lo que no tiene ningún interés desde el punto de vista de la segmentación de datos LiDAR.

3.3.5.1 Precisión y coste general

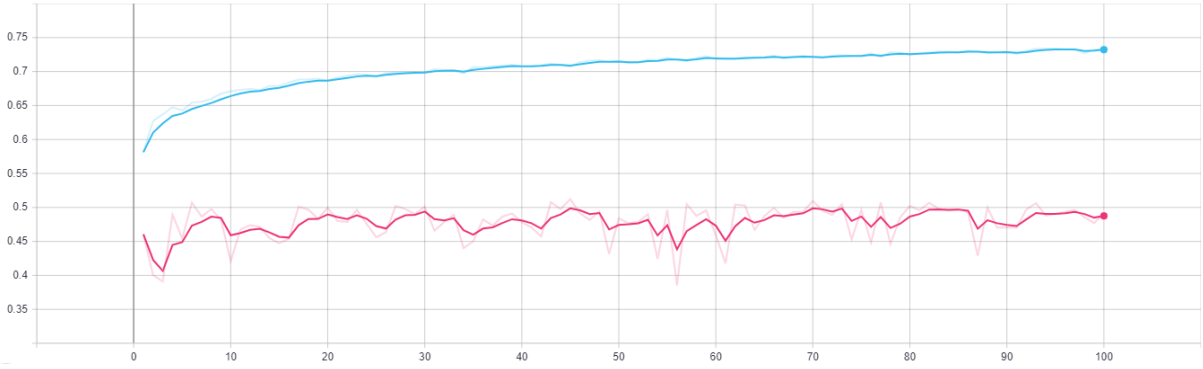

Figura 3.121: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 - experimento 5.

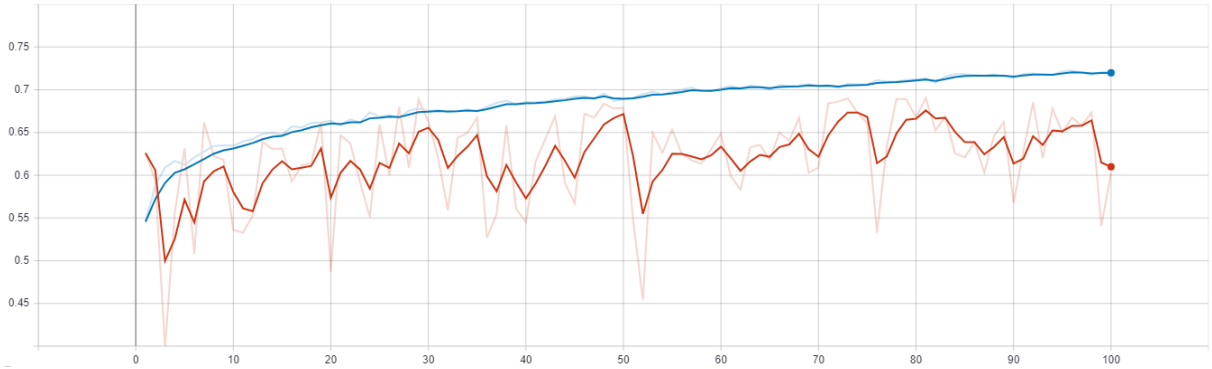

Figura 3.122: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 5.

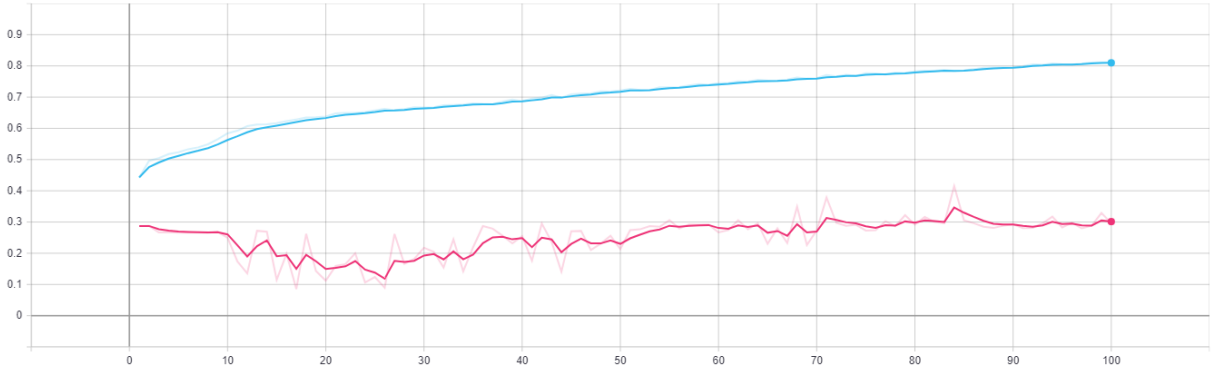

Figura 3.123: Precisión entrenamiento (azul) vs. validación (rosa) en partición 3 - experimento 5.

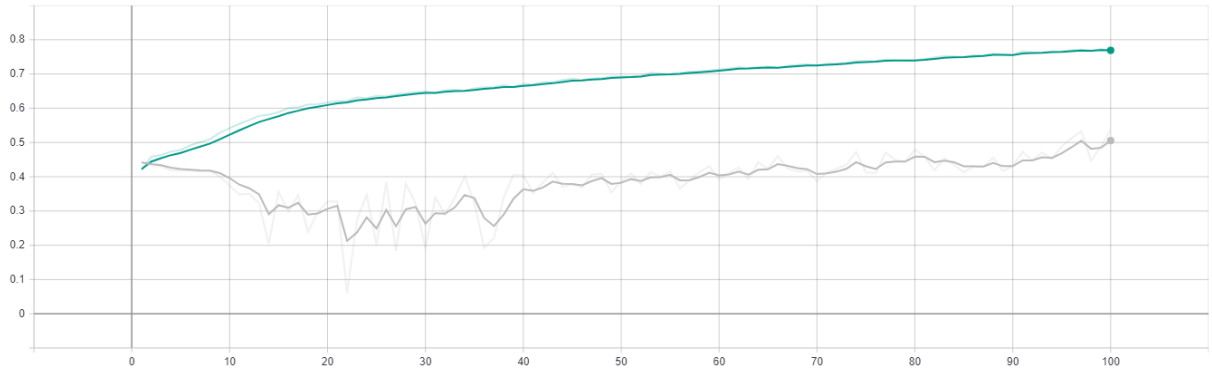

Figura 3.124: Precisión entrenamiento (verde) vs. validación (gris) en partición 4 - experimento 5.

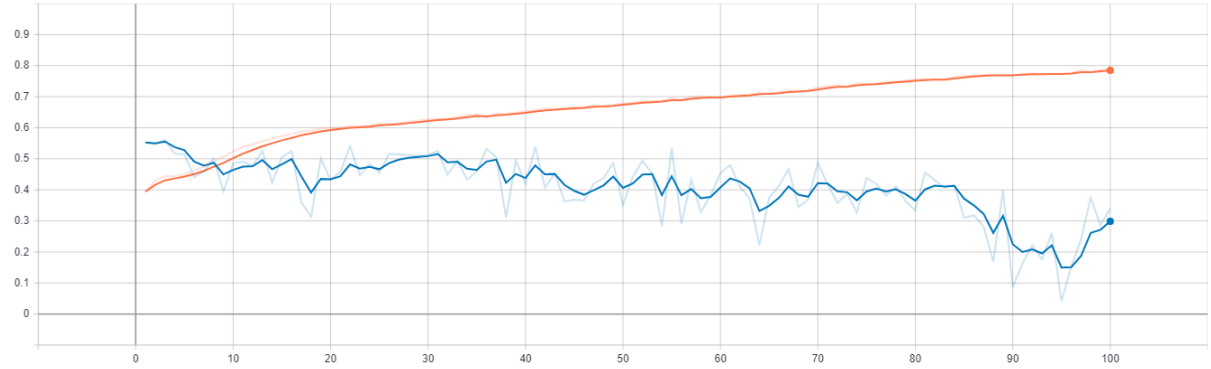

Figura 3.125: Precisión entrenamiento (naranja) vs. validación (azul) en partición 5 - experimento 5.

Se puede apreciar la mala calidad de los resultados, pues no superan el 30% de precisión para esta última partición, además del empeoramiento que se produce en la precisión de validación conforme mejora la precisión de entrenamiento. Esto es un síntoma clave de sobreajuste, aunque este hecho ha de contrastarse visualizando las gráficas de coste.

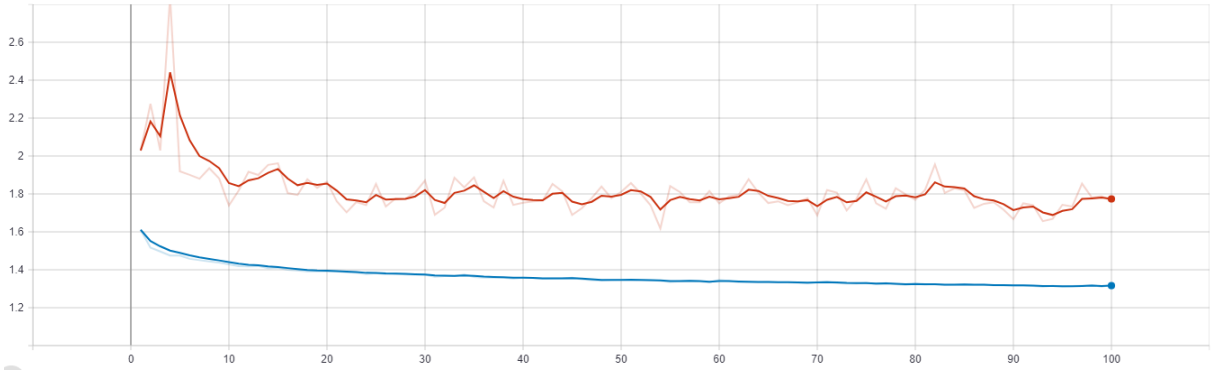

Figura 3.126: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 5.

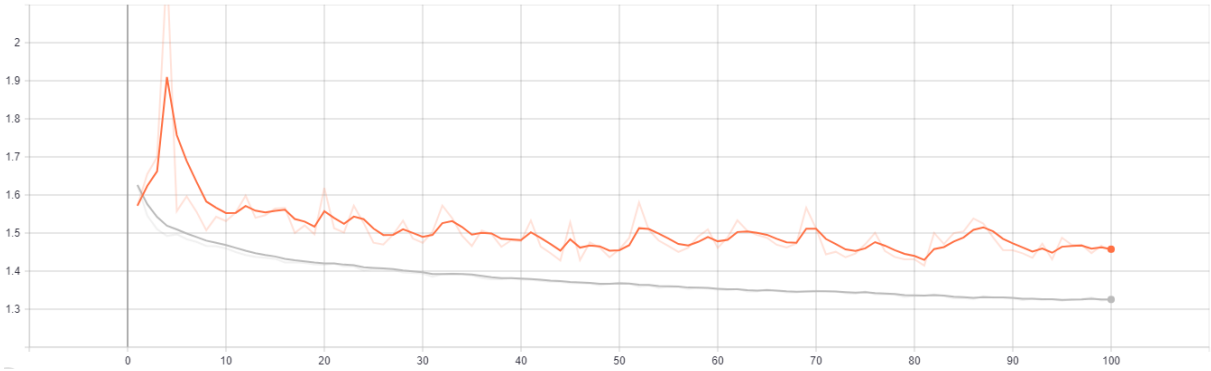

Figura 3.127: Coste de entrenamiento (gris) vs. validación (naranja) en partición 2 - experimento 5.

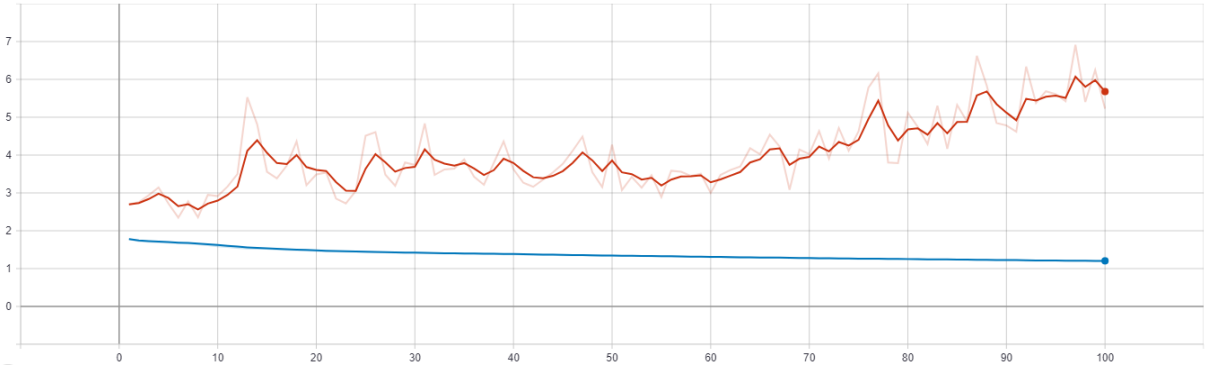

Figura 3.128: Coste de entrenamiento (azul) vs. validación (rojo) en partición 3 - experimento 5.

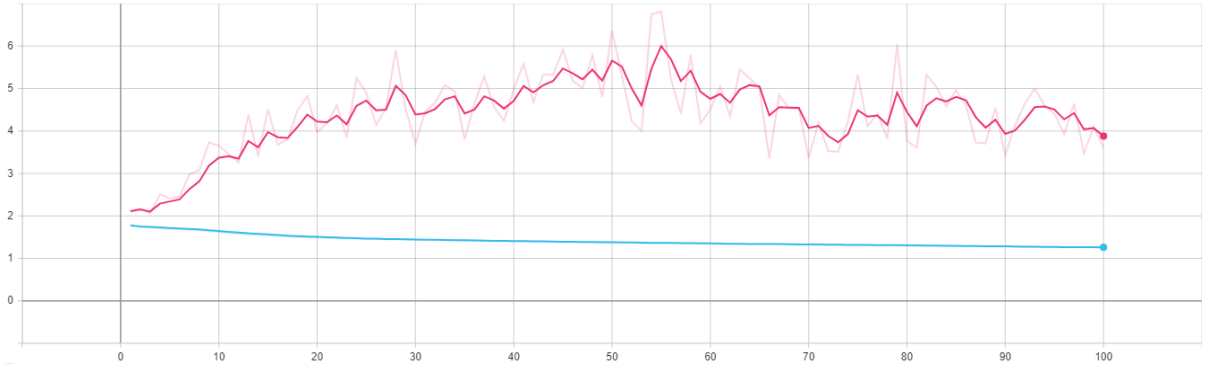

Figura 3.129: Coste de entrenamiento (azul) vs. validación (rosa) en partición 4 - experimento 5.

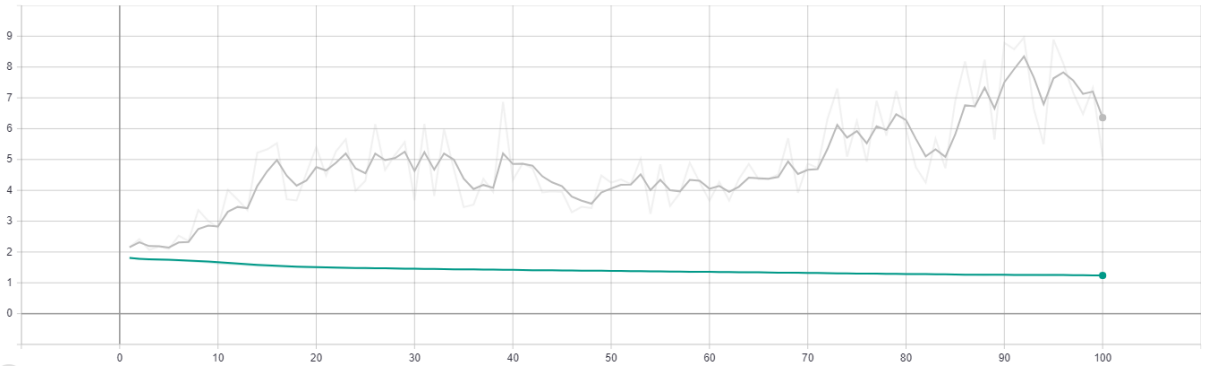

Figura 3.130: Coste de entrenamiento (verde) vs. validación (gris) en partición 5 - experimento 5.

En efecto, a diferencia de los casos anteriores, existen puntos clave en los que la gráfica de coste en validación comienza a crecer mientras que la gráfica asociada al entrenamiento sigue disminuyendo. Este es el momento en el que se empieza a producir el sobreajuste, esto es, la red comienza a **memorizar** el conjunto de entrenamiento en lugar de extraer conocimiento de valor del mismo.

Por tanto, las predicciones realizadas sobre el conjunto de entrenamiento son correctas, mientras que las predicciones realizadas sobre el conjunto de validación comienzan a empeorar en tanto el modelo se ajusta al conjunto de entrenamiento en cuestión.

3.3.5.2 Precisión y coste promedio k-Fold

Por último, se muestran los resultados promedio de coste y precisión para este experimento.

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,269
Coste promedio validación	3,425
Precisión promedio entrenamiento	76,44%
Precisión promedio validación	45,32%

Tabla 27: Resultados promedio experimento 5.

Aunque el valor de la precisión y coste en entrenamiento son competitivos conforme al resto de experimentos, es evidente la descompensación que existe en el caso de validación, siendo este valor muy distante a lo obtenido en otros experimentos realizados anteriormente.

3.3.6 Experimento 6

Por último, y con el máximo interés, se realiza una nueva ejecución para comprobar cuál es el rendimiento máximo alcanzable usando todos los canales de información disponibles y que han sido utilizados de forma aislada en los experimentos anteriores.

La única diferencia existente entre este experimento y el experimento 2 es que en este caso de utilizan además las coordenadas (x, y, z) normalizadas al intervalo $[0, 1]$ según las dimensiones de cada bloque.

3.3.6.1 Precisión y coste general

En primer lugar, se estudian la precisión y coste, al igual que ha ocurrido en los experimentos anteriores.

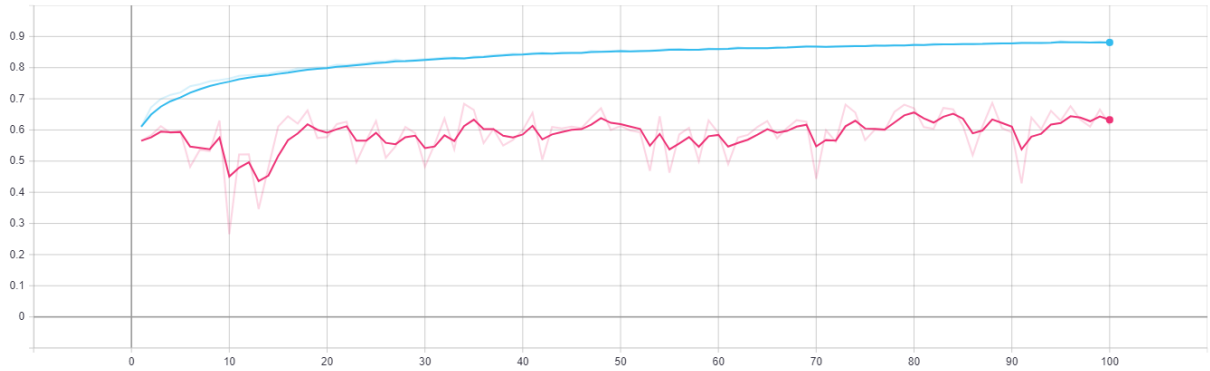

Figura 3.131: Precisión entrenamiento (azul) vs. validación (rosa) en partición 1 - experimento 6.

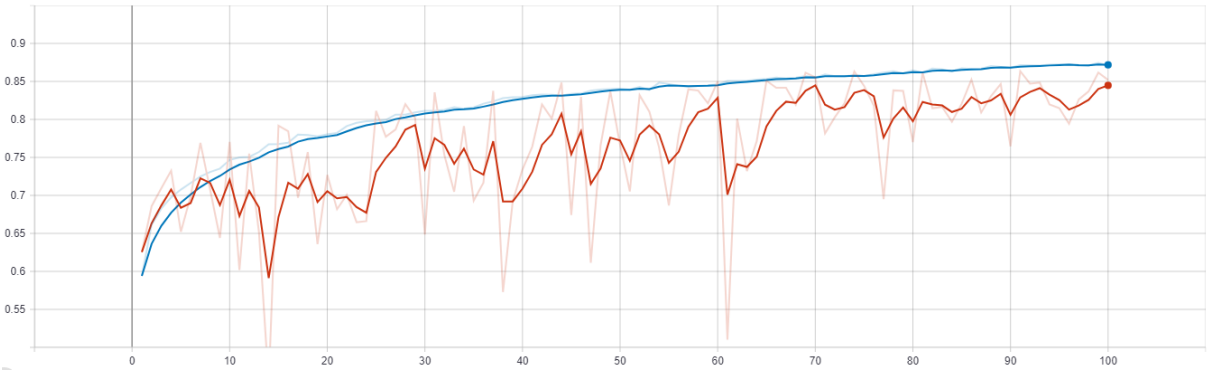

Figura 3.132: Precisión entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 6.

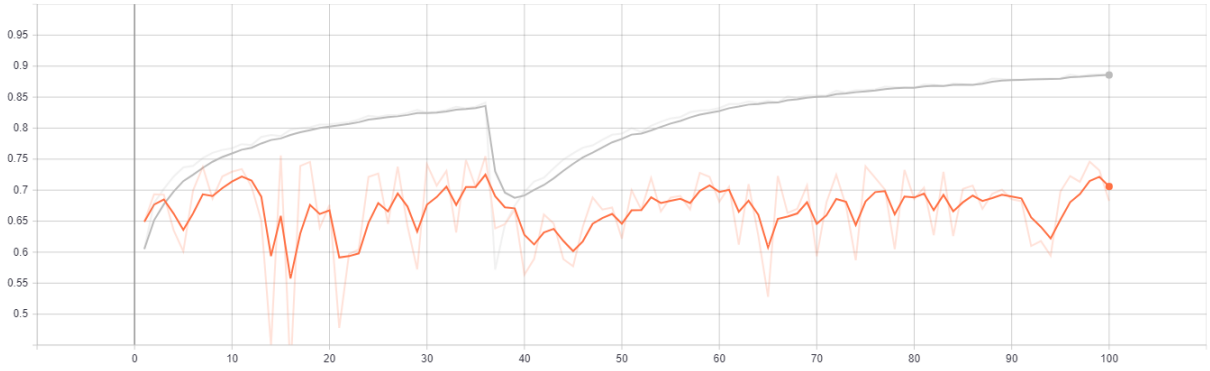

Figura 3.133: Precisión entrenamiento (gris) vs. validación (naranja) en partición 3 - experimento 6.

En la figura anterior (Figura 3.133) puede apreciarse un extraño fenómeno en el ajuste del modelo en el *epoch* 36. Esto se debe a que el experimento fue cancelado del *cluster* principal, y el entrenamiento se retomó en una máquina diferente. Por tanto, aunque se comenzó a entrenar desde el último modelo almacenado (*epoch* 35), alguna variación en algún parámetro provoca que la red tenga que ajustarse de nuevo, y es por ello que se produce un extraño valle que unos *epochs* más tarde se estabiliza de nuevo.

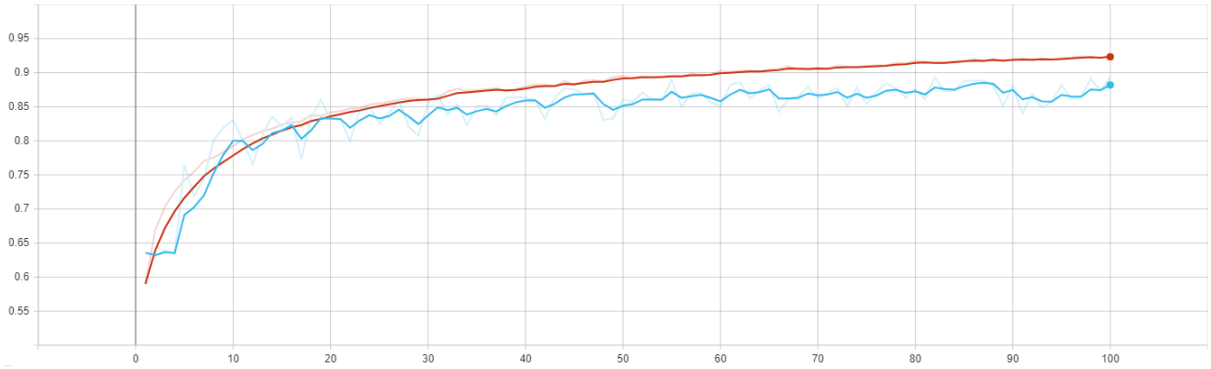

Figura 3.134: Precisión entrenamiento (rojo) vs. validación (azul) en partición 4 - experimento 6.

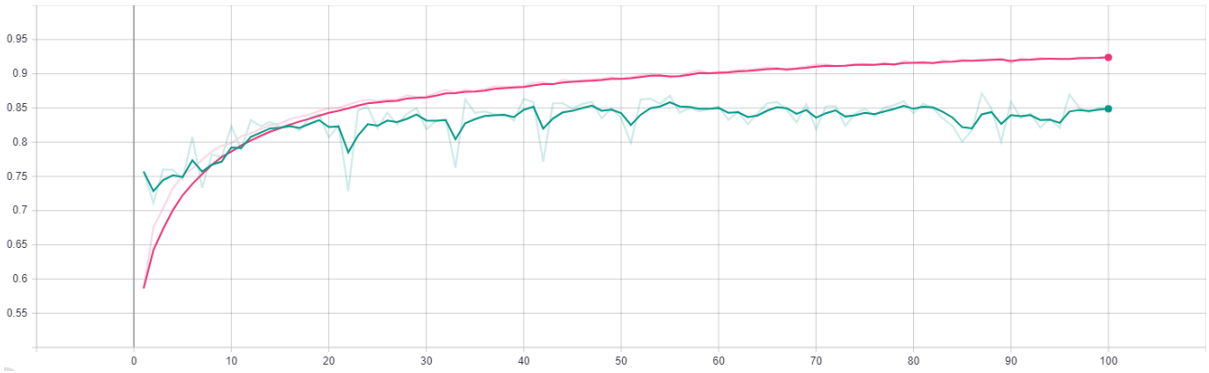

Figura 3.135: Precisión entrenamiento (rosa) vs. validación (verde) en partición 5 - experimento 6.

La precisión obtenida está en torno al 85% en validación en al menos 3 de las 5 particiones existentes para entrenamiento y validación. Esto se debe que es posible que en estas particiones se contengan datos de mayor calidad y variabilidad, lo que permite a la red extraer conocimiento de más alto valor, y posteriormente permite generalizar mejor sobre datos no contemplados en el entrenamiento.

Coste asociado a las ejecuciones:

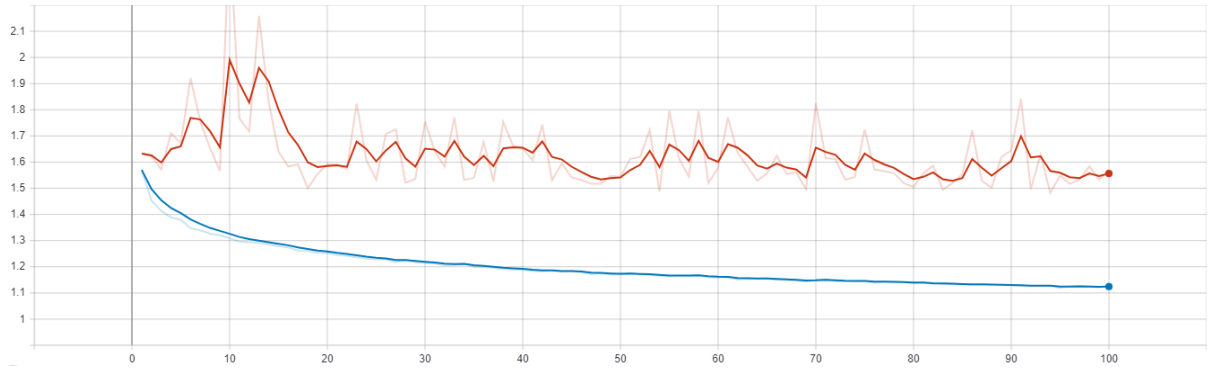

Figura 3.136: Coste de entrenamiento (azul) vs. validación (rojo) en partición 1 - experimento 6.

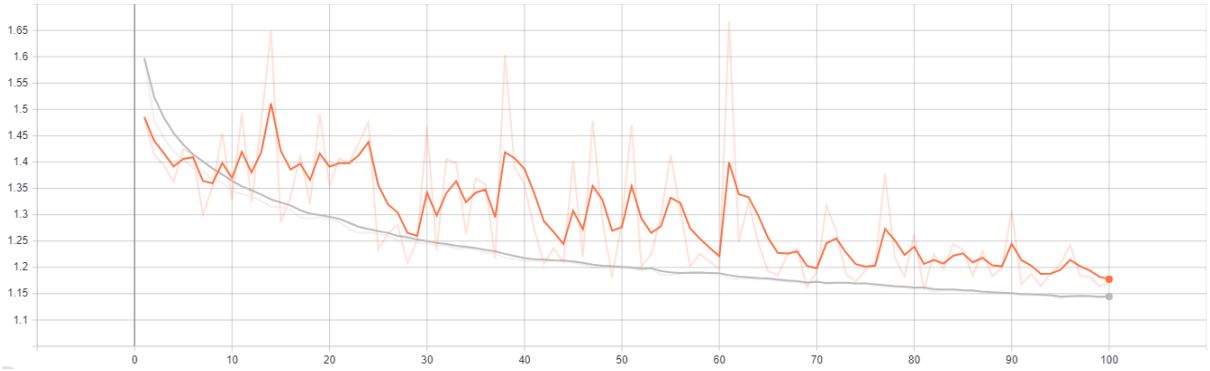

Figura 3.137: Coste de entrenamiento (azul) vs. validación (rojo) en partición 2 - experimento 6.

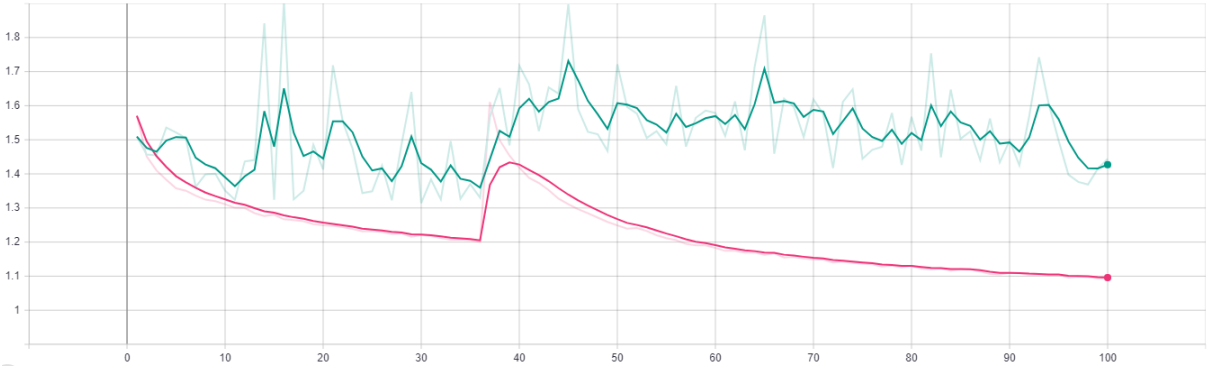

Figura 3.138: Coste de entrenamiento (azul) vs. validación (rojo) en partición 3 - experimento 6.

En la tercera partición ocurre el fenómeno comentado antes, relacionado con el reinicio necesario del experimento a partir de este *epoch* de entrenamiento.

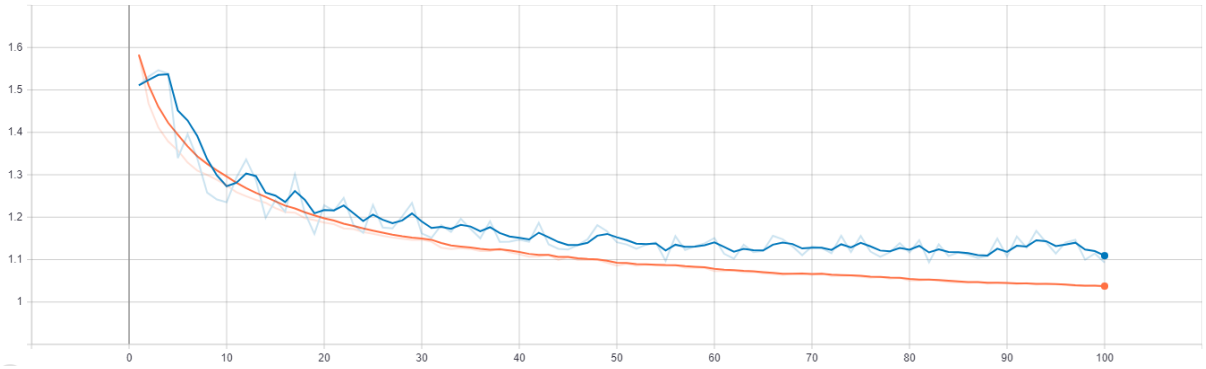

Figura 3.139: Coste de entrenamiento (azul) vs. validación (rojo) en partición 4 - experimento 6.

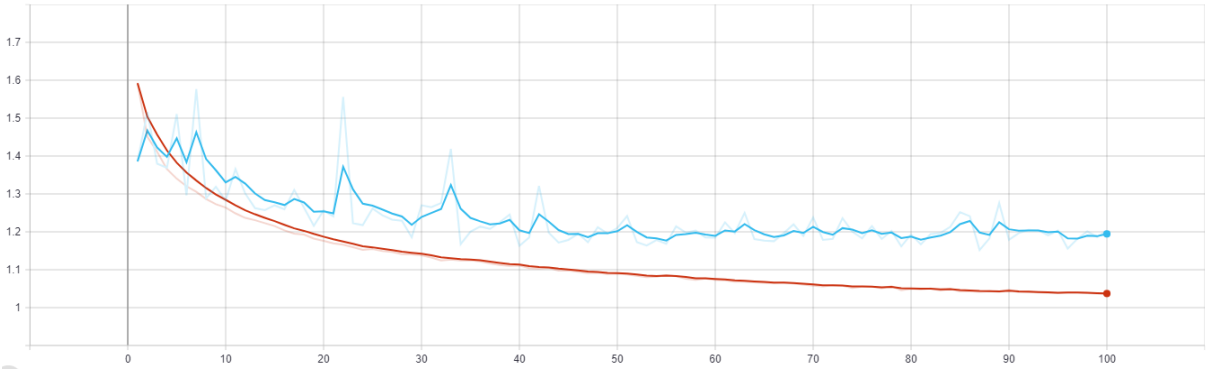

Figura 3.140: Coste de entrenamiento (azul) vs. validación (rojo) en partición 5 - experimento 6.

No se aprecia sobreajuste en ninguno de los 5 ciclos de entrenamiento/validación ejecutados.

3.3.6.2 IoU por clase

A continuación, se analiza la precisión asociada a cada una de las clases del problema.

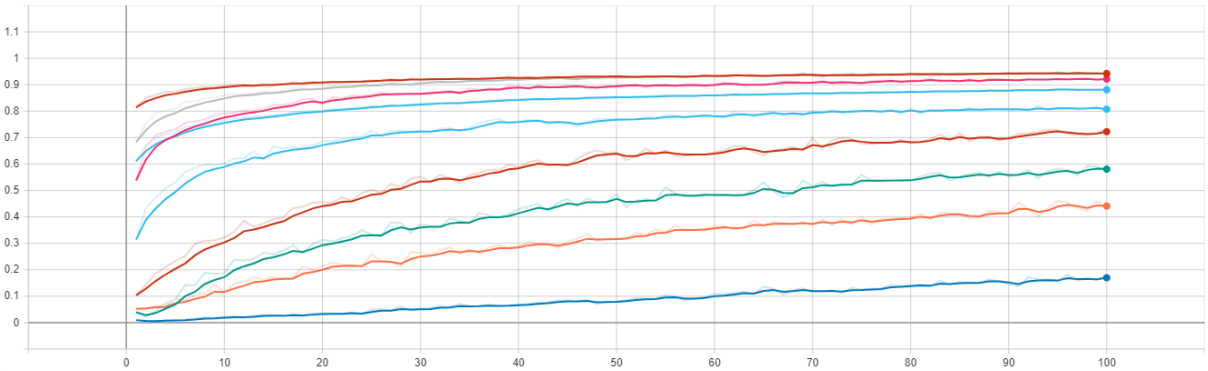

Figura 3.141: IoU asociada al entrenamiento en partición 1 - experimento 6.

Name	Smoothed	Value
1-fold_iou_train_class_0	0.9419	0.9398
1-fold_iou_train_class_1	0.8074	0.8002
1-fold_iou_train_class_2	0.9213	0.9231
1-fold_iou_train_class_3	0.5807	0.5778
1-fold_iou_train_class_4	0.9429	0.9447
1-fold_iou_train_class_5	0.4412	0.4376
1-fold_iou_train_class_6	0.1697	0.1786
1-fold_iou_train_class_7	0.7225	0.7334
1-fold_iou_train_mean_IoU	0.8809	0.8801

Figura 3.142: Leyenda asociada al entrenamiento en partición 1 - experimento 6.

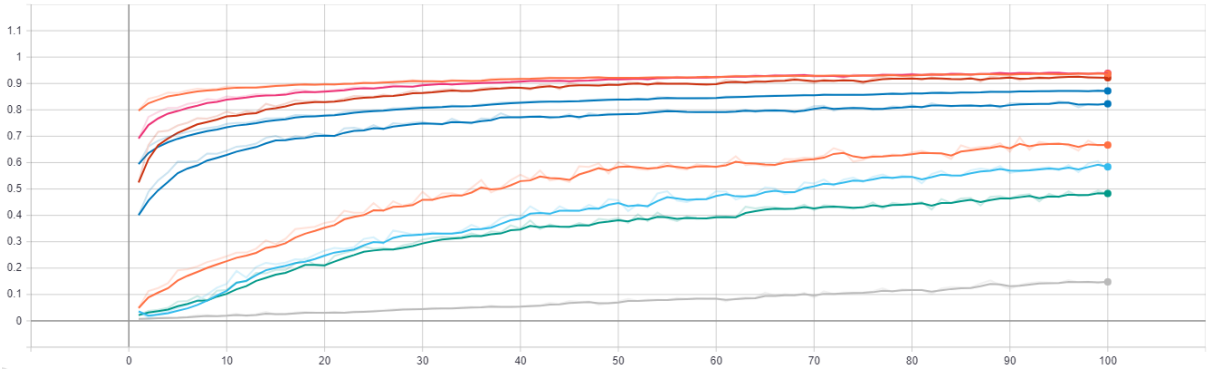

Figura 3.143: IoU asociada al entrenamiento en partición 2 - experimento 6.

Name	Smoothed	Value
2-fold_iou_train_class_0	0.9363	0.9344
2-fold_iou_train_class_1	0.8229	0.8269
2-fold_iou_train_class_2	0.9217	0.9206
2-fold_iou_train_class_3	0.5838	0.5714
2-fold_iou_train_class_4	0.9386	0.9381
2-fold_iou_train_class_5	0.4829	0.4818
2-fold_iou_train_class_6	0.1477	0.1507
2-fold_iou_train_class_7	0.6666	0.6656
2-fold_iou_train_mean_IoU	0.8718	0.8708

Figura 3.144: Leyenda asociada al entrenamiento en partición 2 - experimento 6.

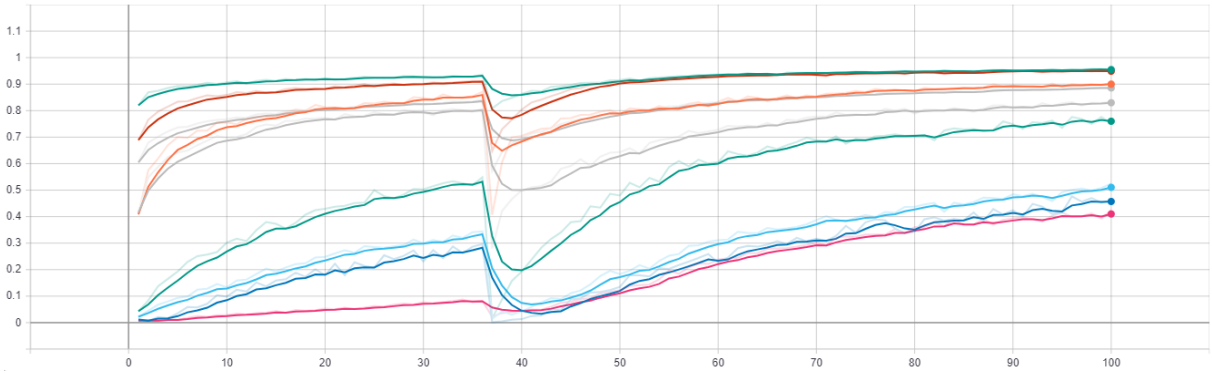

Figura 3.145: IoU asociada al entrenamiento en partición 3 - experimento 6.

Name	Smoothed	Value
3-fold_iou_train_class_0	0.9521	0.9534
3-fold_iou_train_class_1	0.8139	0.8201
3-fold_iou_train_class_2	0.8911	0.8905
3-fold_iou_train_class_3	0.407	0.4081
3-fold_iou_train_class_4	0.947	0.9494
3-fold_iou_train_class_5	0.4641	0.467
3-fold_iou_train_class_6	0.3802	0.3874
3-fold_iou_train_class_7	0.7392	0.7592
3-fold_iou_train_mean_IoU	0.8766	0.8792

Figura 3.146: Leyenda asociada al entrenamiento en partición 3 - experimento 6.

Ocurre lo mismo que lo comentado anteriormente con el reinicio del experimento.

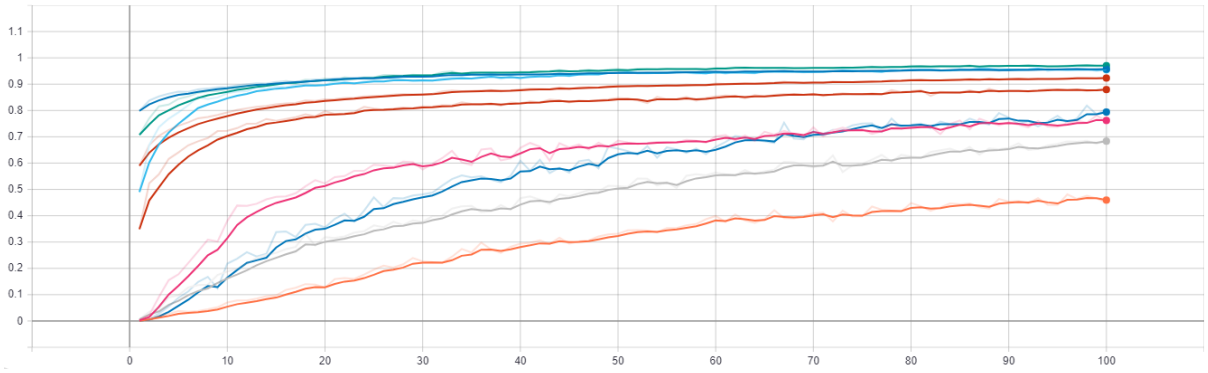

Figura 3.147: IoU asociada al entrenamiento en partición 4 - experimento 6.

Name	Smoothed	Value
4-fold_iou_train_class_0	0.9571	0.9584
4-fold_iou_train_class_1	0.8795	0.8834
4-fold_iou_train_class_2	0.9551	0.957
4-fold_iou_train_class_3	0.7621	0.7595
4-fold_iou_train_class_4	0.9704	0.9706
4-fold_iou_train_class_5	0.6834	0.693
4-fold_iou_train_class_6	0.4595	0.4485
4-fold_iou_train_class_7	0.794	0.8064
4-fold_iou_train_mean_IoU	0.9233	0.9251

Figura 3.148: Leyenda asociada al entrenamiento en partición 4 - experimento 6.

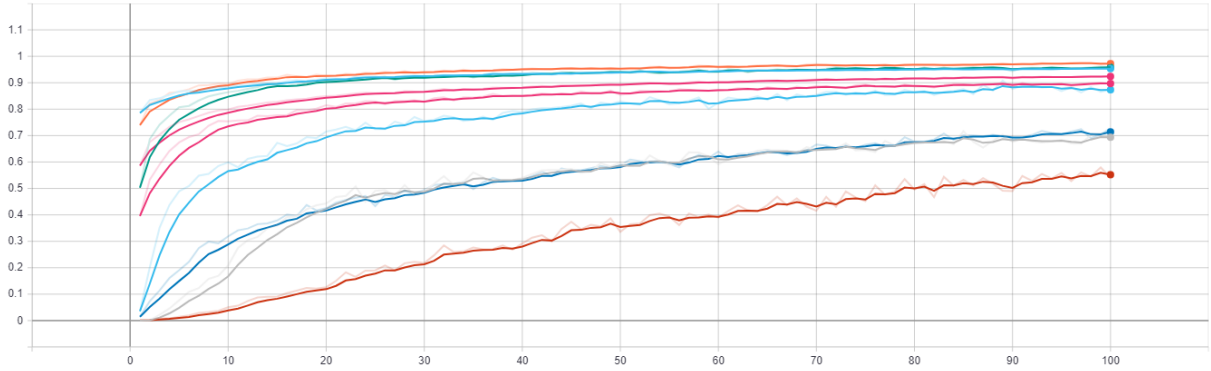

Figura 3.149: IoU asociada al entrenamiento en partición 5 - experimento 6.

Name	Smoothed	Value
5-fold_iou_train_class_0	0.9535	0.9543
5-fold_iou_train_class_1	0.8971	0.8966
5-fold_iou_train_class_2	0.9581	0.9585
5-fold_iou_train_class_3	0.6939	0.6926
5-fold_iou_train_class_4	0.9723	0.9725
5-fold_iou_train_class_5	0.7139	0.7273
5-fold_iou_train_class_6	0.552	0.5406
5-fold_iou_train_class_7	0.8733	0.8743
5-fold_iou_train_mean_IoU	0.9239	0.9249

Figura 3.150: Leyenda asociada al entrenamiento en partición 5 - experimento 6.

Utilizando todos los canales de información disponibles, el modelo es capaz de entrenarse de tal forma que se consigan resultados muy aceptables en todas las categorías del problema.

Intersección sobre Unión en cada clase para el caso de validación:

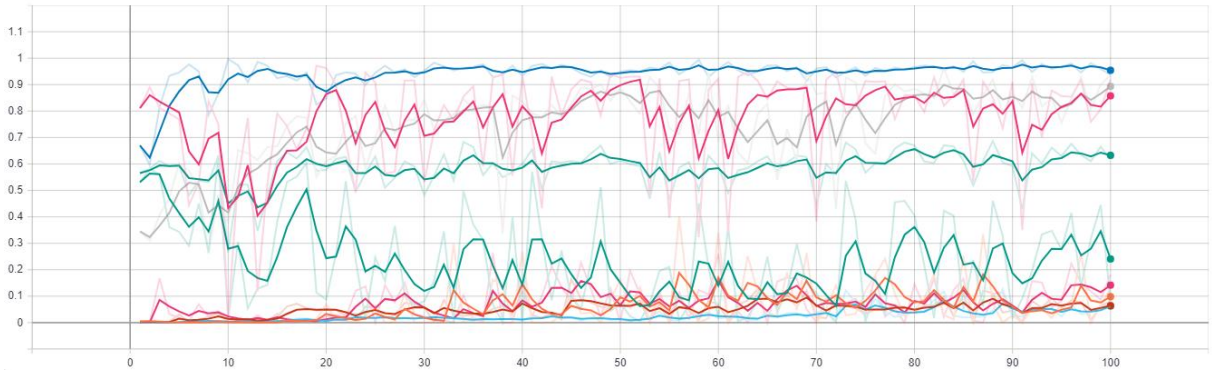

Figura 3.151: IoU asociada a validación en partición 1 - experimento 6.

Name	Smoothed	Value
1-fold_iou_test_class_0	0.8576	0.9192
1-fold_iou_test_class_1	0.2407	0.08295
1-fold_iou_test_class_2	0.8934	0.9349
1-fold_iou_test_class_3	0.09831	0.1314
1-fold_iou_test_class_4	0.9542	0.9376
1-fold_iou_test_class_5	0.06419	0.07743
1-fold_iou_test_class_6	0.06244	0.08567
1-fold_iou_test_class_7	0.1417	0.1814
1-fold_iou_test_mean_IoU	0.6326	0.617

Figura 3.152: Leyenda asociada a validación en partición 1 - experimento 6.

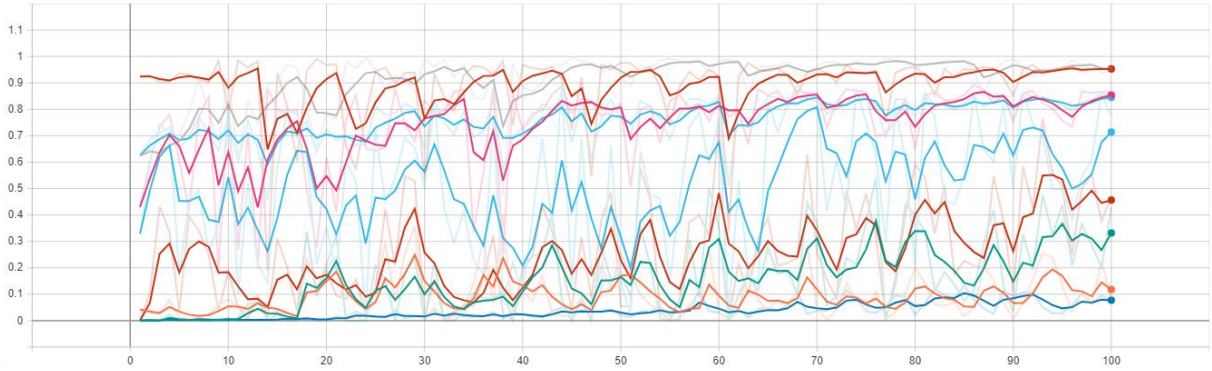

Figura 3.153: IoU asociada a validación en partición 2 - experimento 6.

Name	Smoothed	Value
2-fold_iou_test_class_0	0.9527	0.9532
2-fold_iou_test_class_1	0.7142	0.772
2-fold_iou_test_class_2	0.8539	0.8671
2-fold_iou_test_class_3	0.332	0.4294
2-fold_iou_test_class_4	0.9544	0.9531
2-fold_iou_test_class_5	0.118	0.07748
2-fold_iou_test_class_6	0.07747	0.07428
2-fold_iou_test_class_7	0.4565	0.4725
2-fold_iou_test_mean_IoU	0.8448	0.8519

Figura 3.154: Leyenda asociada a validación en partición 2 - experimento 6.

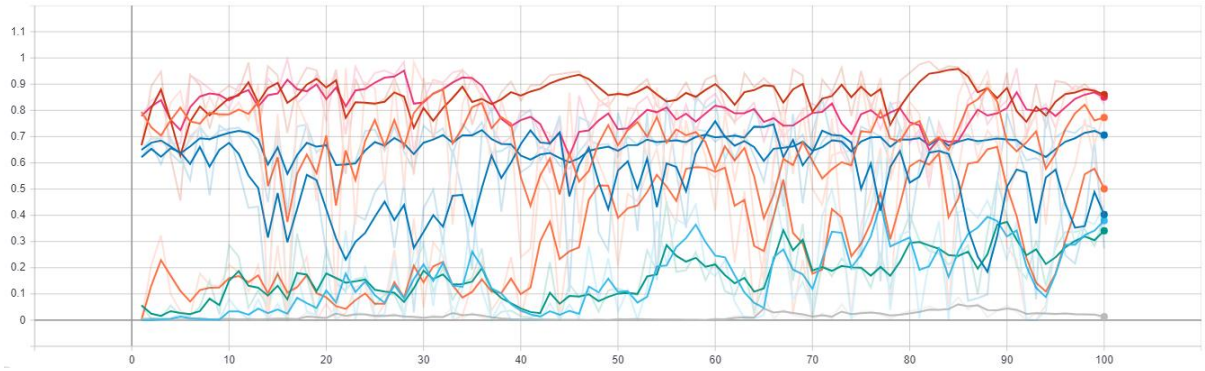

Figura 3.155: IoU asociada a validación en partición 3 - experimento 6.

Name	Smoothed	Value
3-fold_iou_test_class_0	0.7726	0.789
3-fold_iou_test_class_1	0.4029	0.2739
3-fold_iou_test_class_2	0.8601	0.8376
3-fold_iou_test_class_3	0.3794	0.4367
3-fold_iou_test_class_4	0.8506	0.8235
3-fold_iou_test_class_5	0.3408	0.3947
3-fold_iou_test_class_6	0.01326	7.6053e-4
3-fold_iou_test_class_7	0.5008	0.3855
3-fold_iou_test_mean_IoU	0.7059	0.6826

Figura 3.156: Leyenda asociada a validación en partición 3 - experimento 6.

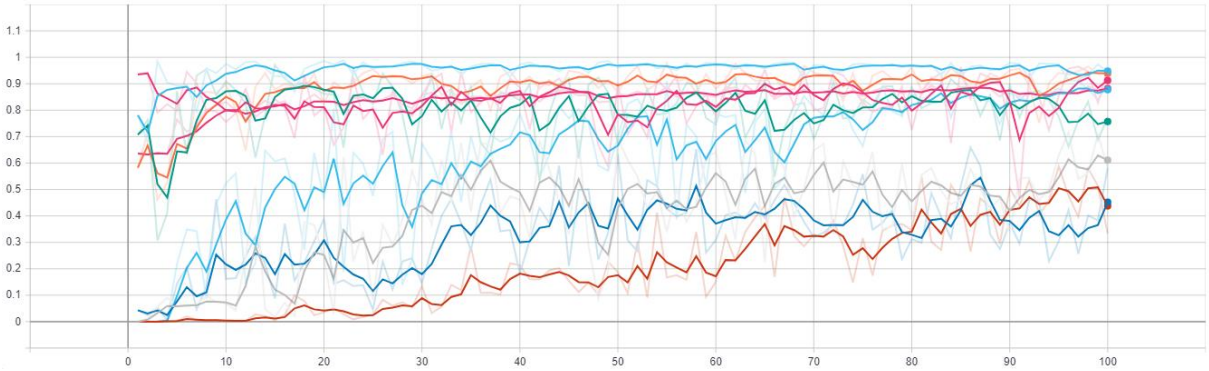

Figura 3.157: IoU asociada a validación en partición 4 - experimento 6.

	Name	Smoothed	Value
	4-fold_iou_test_class_0	0.948	0.945
	4-fold_iou_test_class_1	0.9126	0.9548
	4-fold_iou_test_class_2	0.7571	0.7718
	4-fold_iou_test_class_3	0.611	0.5838
	4-fold_iou_test_class_4	0.9394	0.9376
	4-fold_iou_test_class_5	0.452	0.5817
	4-fold_iou_test_class_6	0.4379	0.3323
	4-fold_iou_test_class_7	0.8775	0.8969
	4-fold_iou_test_mean_IoU	0.8819	0.8931

Figura 3.158: Leyenda asociada a validación en partición 4 - experimento 6.

Figura 3.159: IoU asociada a validación en partición 5 - experimento 6.

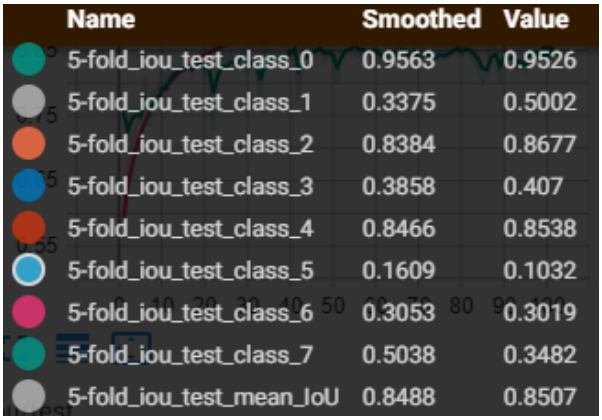

Figura 3.160: Leyenda asociada a validación en partición 5 - experimento 6.

Aunque es necesario estudiar las particiones como un conjunto para establecer conclusiones, y no a nivel individual, puede observarse como en la partición 4 se consiguen resultados muy prometedores para todas las clases del problema. En concreto, las clases 0, 1 ,2, 4 y 7 muestran una precisión por encima del 75%, lo cual indica que el modelo se ha ajustado correctamente y es capaz de reconocer las diferentes formas que componen cada clase utilizando todos los canales de información utilizados.

3.3.6.3 Precisión y coste promedio k-Fold

Para concluir la experimentación, se adjunta una tabla que recoge los resultados de precisión y coste promedios en las 5 particiones de entrenamiento y validación utilizadas.

Parámetro	Resultado obtenido
Coste promedio entrenamiento	1,08
Coste promedio validación	1,29
Precisión promedio entrenamiento	89,37%
Precisión promedio validación	77,9%

Tabla 28: Resultados promedio experimento 6.

3.4 Resumen de experimentos

Una vez los diferentes experimentos han sido completados, es necesario construir una gráfica que permita evaluar el rendimiento de la red neuronal según el

conjunto de entrada utilizado en cada caso. Puesto que la tesis de partida es evaluar si utilizar canales de información adicionales a la geometría supone algún aporte sobre usar únicamente la geometría, se construye una tabla que compare los diferentes resultados obtenidos a nivel de clase.

Por otra parte, una vez obtenido el resultado óptimo, se mostrarán las matrices de confusión correspondientes a ese experimento en cuestión.

3.4.1 Comparativa

Se construye una tabla para comparar los resultados obtenidos en todos los experimentos, según los canales de información utilizados en cada caso. Para simplificar el análisis, se tienen en cuenta únicamente los resultados obtenidos en la 5ª partición de entrenamiento/validación pues es en la partición en la que la totalidad de experimentos ha logrado los mejores resultados, lo que permite comparar justamente los diferentes enfoques seguidos.

Además, se tienen en cuenta únicamente los resultados correspondientes a la validación, pues se trata de un indicador clave sobre la capacidad de predicción del modelo una vez ha sido entrenado, midiendo a la vez la capacidad de aprendizaje a partir de un conjunto de datos, así como la capacidad de utilizar ese conocimiento o bien el valor del conocimiento extraído.

Experimento	*Avg.*	*ioU_0*	*IoU_1*	*IoU_2*	*IoU_3*	*IoU_4*	*IoU_5*	*IoU_6*	*IoU_7*
geometría	0.69	0.95	0.00	0.24	0.00	0.64	0.02	0.00	0.37
geometría + intensidad	0.83	0.93	0.50	0.80	0.28	0.86	0.19	0.05	0.66
geometría + rgb	0.74	0.98	0.08	0.45	0.14	0.61	0.07	0.01	0.68
geometría + int. + rgb	0.81	0.91	0.41	0.80	0.38	0.83	0.18	0.07	0.63
geometría + norm	0.33	0.59	0.00	0.00	0.00	0.08	0.47	0.00	0.00
geom. + int. + rgb + norm	0.85	0.95	0.50	0.86	0.40	0.85	0.10	0.30	0.34

Tabla 29: Comparativa relativa a la métrica IoU en las diferentes clases del dataset en todos los experimentos.

Como puede observarse en la tabla anterior, el experimento que utiliza todos los canales de información disponibles domina al resto de experimentos, consiguiendo un 2% más de precisión que el segundo mejor resultado (usando únicamente la intensidad). Por tanto, esto nos muestra que el canal de intensidad es fundamental para realizar una buena segmentación semántica de la nube de puntos.

Mientras que propiedades como el color (r, g, b) se pueden obtener usando escáneres convencionales, o bien técnicas fotogramétricas como ***Sfm***, el canal de intensidad es **exclusivo** de los escáneres LiDAR.

Por otra parte, la pequeña diferencia existente en la precisión asociada al experimento 2 y experimento 6 podría deberse a la inicialización aleatoria de los parámetros entrenables de la red neuronal, y es por ello que se consigue un resultado un 2% más alto en el segundo caso. En cualquier caso, todas las configuraciones son capaces de identificar **terreno artificial** con un porcentaje de acierto superior al 90%.

Para concluir sobre el análisis comparativo de los diferentes experimentos, se construye otra tabla que refleja la precisión promedio obtenida en cada experimento, resultado de realizar la media de los IoUs promedios en cada partición, para cada experimento. En verde, el mejor valor para la columna. En naranja, el peor valor.

	Entrenamiento		***Validación***	
Experimento	***Coste***	***Precisión***	***Coste***	***Precisión***
geometría	1,39	67,90%	1,55	55,8%
geometría + intensidad	1,13	87,67%	1,29	77,40%
geometría + rgb	1,251	77,89%	1,507	62,36%
geometría + int. + rgb	1,13	87,66%	1,33	75,84%
geometría + norm	1,269	76,44%	3,425	45,32%
geom. + int. + rgb + norm	1,08	89,37%	1,29	77,90%

Tabla 30: Comparativa general entre los resultados obtenidos en los experimentos.

Es apreciable que los resultados asociados al experimento 6, que utiliza todos los canales de información disponibles, dominan en los 4 parámetros de valoración al

resto. Además, la exclusiva utilización de canales geométricos empeora significativamente el rendimiento de los modelos entrenados, tanto si se trata de canales geométricos sin procesar como de canales geométricos normalizados según las coordenadas de cada bloque.

Sin embargo, resulta adecuado tener en cuenta que la diferencia entre el experimento que únicamente utiliza el canal geométrico con respecto al uso de la totalidad de canales es mínima. Es necesario encontrar un compromiso entre la maximización de la precisión obtenida y la minimización de los recursos necesarios para entrenar y probar los modelos. La cantidad de recursos (memoria, procesamiento) necesaria para entrenar un modelo usando únicamente el canal de intensidad es significativamente menor a los recursos necesarios para entrenar un modelo utilizando toda la información, y ha de encontrarse un punto intermedio entre ambos límites de decisión.

3.4.2 Matrices de confusión

En última instancia, partiendo de la base de que el experimento que aporta los mejores resultados es el que utiliza todos los canales de información, se adjuntan las matrices de confusión relativas al último *epoch* de entrenamiento en cada una de las 5 particiones de **validación**, ignorando los resultados de entrenamiento puesto que no miden la capacidad de generalización del modelo neuronal.

		Predicción							
		0	1	2	3	4	5	6	7
Ground truth	0	91,92%	0,63%	0,10%	0,73%	6,17%	0,24%	0,09%	0,12%
	1	75,40%	8,29%	1,44%	0,31%	14,26%	0,29%	0,00%	0,01%
	2	0,04%	0,16%	93,49%	0,84%	5,35%	0,11%	0,00%	0,00%
	3	10,72%	8,54%	42,83%	13,14%	21,34%	2,53%	0,49%	0,40%
	4	2,66%	0,35%	1,58%	0,18%	93,76%	1,36%	0,11%	0,00%
	5	46,09%	6,62%	5,22%	2,18%	29,40%	7,74%	0,08%	2,67%
	6	39,77%	0,28%	1,35%	7,96%	38,33%	2,58%	8,57%	1,16%
	7	36,59%	0,23%	0,00%	3,05%	40,78%	1,17%	0,05%	18,14%

Tabla 31: Matriz de confusión para la primera partición.

		Predicción							
		0	1	2	3	4	5	6	7
Ground truth	0	95,32%	3,29%	0,56%	0,10%	0,29%	0,28%	0,01%	0,16%
	1	21,28%	77,20%	0,57%	0,22%	0,54%	0,19%	0,00%	0,00%
	2	3,44%	0,73%	86,71%	3,25%	4,79%	1,08%	0,00%	0,00%
	3	32,18%	2,41%	16,73%	42,94%	5,68%	0,06%	0,00%	0,00%
	4	1,50%	0,21%	1,74%	0,62%	95,31%	0,60%	0,00%	0,01%
	5	17,13%	12,12%	9,88%	6,40%	45,79%	7,75%	0,31%	0,62%
	6	53,73%	2,28%	0,29%	0,68%	19,99%	0,90%	7,43%	14,70%
	7	40,89%	1,77%	0,10%	0,02%	1,49%	3,80%	4,68%	47,25%

Tabla 32: Matriz de confusión para la segunda partición.

		Predicción							
		0	1	2	3	4	5	6	7
Ground truth	0	67,20%	28,46%	1,52%	0,09%	2,01%	0,50%	0,00%	0,24%
	1	15,70%	68,45%	4,32%	1,06%	7,73%	2,12%	0,00%	0,61%
	2	0,87%	2,47%	86,66%	0,79%	7,54%	1,67%	0,00%	0,00%
	3	0,04%	8,73%	23,75%	36,59%	24,37%	6,52%	0,00%	0,00%
	4	1,54%	5,21%	1,83%	0,72%	88,20%	0,91%	0,87%	0,73%
	5	6,85%	7,66%	5,92%	1,47%	47,11%	28,28%	2,17%	0,52%
	6	32,72%	3,93%	1,58%	0,00%	35,64%	11,09%	2,12%	12,91%
	7	11,13%	0,37%	0,00%	0,00%	26,92%	0,09%	0,58%	60,90%

Tabla 33: Matriz de confusión para la tercera partición.

		Predicción							
		0	1	2	3	4	5	6	7
Ground truth	0	97,21%	0,65%	0,11%	0,02%	0,25%	0,45%	0,01%	1,31%
	1	25,62%	72,44%	0,64%	0,12%	0,45%	0,66%	0,00%	0,08%
	2	1,30%	1,85%	88,92%	2,45%	5,02%	0,46%	0,01%	0,00%
	3	1,03%	0,79%	29,62%	38,96%	22,87%	6,66%	0,00%	0,08%
	4	0,32%	1,22%	0,33%	1,45%	96,27%	0,30%	0,00%	0,10%
	5	8,57%	4,94%	6,34%	5,54%	33,92%	31,28%	0,02%	9,40%
	6	28,04%	1,73%	0,16%	0,00%	10,96%	6,97%	34,66%	17,48%
	7	5,78%	0,02%	0,01%	0,00%	6,81%	1,35%	0,08%	85,94%

Tabla 34: Matriz de confusión para la cuarta partición.

		Predicción							
		0	1	2	3	4	5	6	7
Ground truth	0	92,07%	3,64%	0,02%	0,11%	2,92%	0,12%	0,03%	1,09%
	1	51,00%	31,92%	4,13%	2,15%	10,20%	0,60%	0,00%	0,00%
	2	1,57%	0,45%	83,20%	10,53%	4,05%	0,17%	0,03%	0,00%
	3	5,41%	13,13%	3,12%	37,85%	33,25%	6,98%	0,04%	0,22%
	4	2,27%	0,20%	0,75%	0,53%	89,83%	2,58%	0,18%	3,66%
	5	16,86%	10,30%	5,53%	2,28%	39,20%	11,84%	2,79%	11,20%
	6	29,28%	3,65%	0,46%	6,98%	31,66%	0,78%	19,27%	7,93%
	7	9,78%	4,64%	0,56%	0,03%	11,22%	4,13%	6,53%	63,10%

Tabla 35: Matriz de confusión para la quinta iteración.

(Página intencionalmente en blanco)

4 CONCLUSIONES Y TRABAJOS FUTUROS

El principal objetivo de este trabajo es el de estudiar las principales técnicas para segmentación semántica de nubes de puntos que conforman el estado del arte de la problemática. En tanto, se pretende estudiar cómo se pueden adaptar las técnicas ya existentes para utilizar como entrada nubes de puntos LiDAR en su formato original. Hasta el momento, la práctica totalidad de técnicas que se han desarrollado o propuesto para aplicar modelos de aprendizaje profundo sobre nubes de puntos mantienen su foco sobre nubes de puntos genéricas, haciendo uso únicamente de la información geométrica, y en algunos casos se incorpora también el color, aunque esto no suele ser habitual ni tampoco un punto a destacar en los artículos científicos revisados.

En este proyecto, se ha escogido una de las técnicas más exitosas para segmentación semántica de nubes de puntos. El principal problema que surge al aplicar redes neuronales sobre nubes de puntos es que el tipo de redes que suelen utilizarse, los modelos convolucionales, no se adaptan de forma natural a este tipo de datos. En tanto, se hace necesario establecer un operador de consulta de vecinos que permita extraer características locales a cada punto de la misma forma que un filtro bidimensional lo hace sobre conjuntos de píxeles. Así, la propuesta escogida hace uso de *EdgeConv*, un operador convolucional que se sirve de los vecinos más cercanos para extraer características locales.

Una vez desarrollada la arquitectura por completo, se plantea un conjunto de experimentos para evaluar la influencia de canales adicionales de información que son captables a través de LiDAR, como por ejemplo el canal de intensidad o también el color, así como la influencia de uno sobre otro y en qué medida su actuación conjunta puede potenciar los resultados obtenidos.

De este modo, se observa que el uso exclusivo de canales geométricos se ve radicalmente empobrecido en comparación a la incorporación de canales adicionales, como la intensidad. En concreto, el canal de intensidad, que responde a una medida de la reflectividad de las superficies escaneadas haciendo uso del sensor LiDAR, supone una mejora destacable con respecto al resto de canales, como el color.

Finalmente, se observa que el resultado más destacable se obtiene utilizando todos los canales de información disponibles: intensidad, color y coordenadas normalizadas. Sin embargo, también se destaca que la mejora de utilizar toda la información disponible es inapreciable (un 2% por encima, 89,37% en **entrenamiento**) con respecto a utilizar únicamente la intensidad (87,67% en **entrenamiento**).

En tanto, el uso exclusivo de la intensidad resulta en **un 77,40% de precisión** promedio en **validación**, mientras que la incorporación de **todos los canales** incrementa este porcentaje hasta **77,90%.** Así, se concluye al respecto que el canal de intensidad resulta prometedor para iniciar un estudio de su influencia sobre otros conjuntos de datos de diferente naturaleza.

Finalmente, se recogen los resultados del resto de experimentos realizados, y se ha redactado esta memoria de iniciación a *geometric deep learning* para todo aquel lector que lo desee, así como de punto de inicio de una tesis doctoral que se desarrollará en los próximos años partiendo de la línea de investigación que se inicia en el momento en que este proyecto sea defendido ante tribunal.

4.1 Trabajos futuros

El trabajo actualmente presentado cumple con la dedicación de 300 horas exigida por la Normativa sobre los Trabajos Fin de Grado y Fin de Máster aprobada en la Escuela Politécnica Superior de Jaén para su para su defensa ante tribunal. Sin embargo, a partir de este proyecto se han abierto numerosas líneas de trabajo que finalmente no se han cerrado debido a la principal restricción temporal mencionada.

Por ejemplo, en el *software* desarrollado puede observarse la implementación de otras funcionalidades como, por ejemplo, *attentive pooling*. El objetivo principal era el de evaluar la influencia de este tipo de *pooling* en contraposición al método de reducción clásico, *max pooling*. De hecho, numerosos experimentos se estaban ejecutando para incluir en este documento, aunque finalmente se han descartado debido a que las conclusiones a extraer eran irrelevantes conforme al propósito del proyecto.

Por otra parte, también se ha desarrollado una metodología para realizar inferencia de nubes de puntos una vez un modelo ha sido entrenado. Se trata de un *script* que recibe como entrada una nube de puntos en formato original, y emite como

salida una nube de puntos segmentada utilizando el conocimiento extraído durante el entrenamiento. Sin embargo, esta funcionalidad no ha llegado a ponerse en práctica debido a que está más relacionada con la puesta en producción del modelo desarrollado, y esto es algo de carácter más comercial que no tiene tanta relevancia en el estudio en sí.

De igual manera, se ha realizado la preparación del conjunto de datos LiDAR mencionado en el apartado 1.9.2 correspondiente al escaneado de la ciudad de Pamplona, mientras que finalmente no se ha realizado una transferencia del conocimiento entre diferentes dominios dado que se excedía la principal limitación de la asignatura asociada al Trabajo Fin de Máster de no sobrepasar las 300 horas requeridas para realizar el trabajo.

En tanto, se proponen las siguientes líneas de actuación futuras a partir de lo expuesto inmediatamente antes:

- ***Attentive Pooling***. Evaluar la influencia de este método de reducción, en contraposición al método clásico de *max pooling*.
- ***Inferencia***. Realizar segmentación de nubes no etiquetadas previamente utilizando el *software* ya desarrollado en la realización de este trabajo.
- ***Datos LiDAR Pamplona***. Transferir conocimiento adquirido en Semantic3D a el dominio LiDAR de Pamplona, y evaluar cómo un *dataset* originalmente destinado a evaluación de propuestas científicas podría llevarse al ámbito comercial y utilizar el conocimiento adquirido sobre nubes de puntos de diferentes ámbitos.
- ***Parámetros***. Ejecutar experimentos utilizando otros parámetros, como por ejemplo variando el *stride* en la generación del conjunto de entrenamiento, así como aumentando el tamaño de bloque, esto es, cubrir áreas más extensas con cada bloque generado.

(Página intencionalmente en blanco)

5 APÉNDICES

5.1 Guía original del Trabajo Fin de Título

(Cód.: 19/20-448) Deep Learning geométrico aplicado a datos LiDAR

Tutor del TFM: **RAFAEL JESÚS SEGURA SÁNCHEZ**
Idioma: Castellano
Asignado al alumno con DNI:77386787K
Segundo tutor del TFM: **JOSÉ MANUEL FUERTES GARCÍA**

La segmentación semántica y clasificación de nubes de puntos es un problema clásico en el campo de la informática gráfica y visión por computador que desde hace décadas se ha convertido en uno de los principales objetos de estudio para investigadores de todo el mundo. Aunque se trata de tareas diferentes, ambas se consolidan sobre una misma base: el reconocimiento de formas a partir de información geométrica y tridimensional. Consolidándose ambas como un problema de clasificación del ámbito del aprendizaje automático, se diferencian en cuanto a la porción del espacio que ha de ser etiquetada, y se podría decir que una incluye a la otra o se sitúa en el paso previo.

Los datos LiDAR (*Light Detection And Ranging)* son generados mediante escáneres LiDAR, y se constituyen como nubes de puntos con una cantidad de información muy superior a las nubes de puntos convencionales. Además de información sobre las coordenadas (x, y, z), un escáner LiDAR es capaz de capturar otros atributos del entorno como por ejemplo el color, la reflectividad del material, el número de retornos de cada punto, además de otras propiedades sobre la superficie de los materiales.

El aprendizaje profundo se ha situado en la cúspide de las soluciones para la mayoría problemas existentes en el mundo de las ciencias de computación, desde la **visión por computador** hasta el **procesamiento del lenguaje natural**. Se trata de resolver problemas utilizando un enfoque inteligente, en el que el paradigma clásico de programación se sustituye por uno basado en la inteligencia artificial y que construya su propia representación del conocimiento para resolver el problema en cuestión.

En tanto, el presente trabajo tiene el objetivo de estudiar las propuestas actuales enfocadas en clasificación y segmentación de nubes de puntos que parten del

Figura 5.1: guía original del Trabajo Fin de Máster publicada en la web de la EPS (I).

aprendizaje profundo, y evaluar la influencia que tiene sobre las mismas la incorporación de canales de información adicionales como metainformación a la geometría. Además, el tratamiento de nubes de puntos masivas introduce la necesidad de realizar una computación *out-of-core*, puesto que el tamaño de los datos sobrepasa la capacidad de la memoria principal de un ordenador convencional.

Conocimientos Previos

No tiene

Objetivos del TFM

- Estudiar las técnicas de deep learning para la resolución de problemas, y en concreto geometric deep learning para el tratamiento de problemas geométricos.
- Analizar y comprender la utilización de herramientas inteligentes para la resolución de problemas complejos.
- Experimentar con arquitecturas neuronales artificiales profundas para comprender su comportamiento.
- Manejar *clusters* de Computación de Altas Prestaciones para ejecutar experimentos.

Metodología a Desarrollar

Elaboración de un estado del arte sobre segmentación y clasificación de nubes de puntos usando aprendizaje profundo.

- Elección de un conjunto de datos adecuados para realizar experimentaciones.
- Desarrollo de propuesta para segmentación y clasificación de datos LiDAR.
- Adaptación de arquitecturas estudiadas para aceptar datos LiDAR.
- Estudio de la influencia de los canales de información adicionales sobre los resultados de la segmentación.
- Batería de experimentos para contrastar todo lo anterior.
- Recogida de resultados y contraste de hipótesis.
- Elaboración de la memoria.

Documentos y Formatos de Entrega

- Memoria del trabajo, incluyendo manuales y anexos, en formato PDF.
- Código fuente y ejecutables del prototipo desarrollado, si procede.
- Documentos y archivos adicionales a los que se haga mención expresa en la memoria.
- Vídeo de demostración del prototipo de la aplicación.

Figura 5.2: guía original del Trabajo Fin de Máster publicada en la web de la EPS (II).

5.2 Instalación y configuración del sistema

5.2.1 Creación del entorno conda

Para poder ejecutar el *software* desarrollado, es necesario tener instalado Anaconda 3 como gestor de entornos virtuales Python. Tras esto, para que el proyecto pueda ejecutarse correctamente con todas las dependencias asociadas, el siguiente paso consiste en crear el entorno *conda* sobre el que se ejecutarán los *scripts* Python.

Así, localizar los ficheros *environment.yml* y *requirements_pip.txt* y ejecutar la orden:

conda create –f environment.yml.

La orden anterior creará un entorno *conda* e instalará todas las dependencias necesarias para la ejecución del proyecto. Sin embargo, algunas librerías han sido instaladas usando el gestor *pip*, por lo que no están disponibles en los repositorios *conda*. Por tanto, para completar la configuración del entorno, ejecutar la siguiente orden:

pip install –r requirements_pip.txt

De esta forma, se instalarán las dependencias que no se hayan podido instalar en el paso previo, y se dispondrá de un entorno preparado para la ejecución de los *scripts* desarrollados.

5.2.2 Inventario de software

En el directorio de *software* adjunto se incorporan los siguientes ficheros:

- ***data***. Contiene todos los datos a utilizar en los procesos de entrenamiento y validación de la red neuronal. Este directorio contendrá un nuevo directorio por cada *dataset* utilizado, tal y como puede observarse en el proyecto entregado.
- ***experiments***. Albergará los resultados de los experimentos ejecutados.
- ***models***. Contiene los *scripts* que definen la red neuronal utilizada, así como funcionalidad requirida para los procesos de propagación hacia adelante en la red (por ejemplo, cálculo del grafo de características, *attentive pooling*, etcétera).
- ***preprocessing***. Funcionalidad para preparar el conjunto de datos de entrenamiento utilizado durante el proceso de aprendizaje de la red neuronal utilizada.
- ***utils***. Funciones de utilidad para calcular métricas como IoU, coste, almacenamiento y carga de modelos entrenados, visualización de nubes de puntos, descarga del *dataset* Semantic3D de la web, etcétera.

- ***environment.yml*.** Para crear el entorno *conda* con todas las dependencias necesarias.
- ***requirements_pip.txt***. Dependencias instalables mediante el gestor de paquetes ***pip***.
- ***prepare_data.py***. *Script* para la preparación de los datos de entrenamiento.
- ***train_network.py***. *Script* para realizar el entrenamiento de la red neuronal, almacenamiento de modelos entrenados, reentrenamiento (transferencia de conocimiento), etcétera.
- ***resume_training.py.*** *Script* para resumir un entrenamiento a partir de un *epoch* determinado y sobre una partición determinada. Es necesario especificar el nombre del modelo preentrenado a cargar.

6 DEFINICIONES Y ABREVIATURAS

- **LiDAR**. *Light Detection And Ranging*. Se trata de un escáner de nubes de puntos basado en la emisión y posterior reflexión de puntos de luz sobre una superficie (Lv et al., 2017). Es un dispositivo que permite determinar la distancia desde un emisor laser a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.

  Existen distintos tipos de LiDAR. Estos pueden clasificarse atendiendo al tipo de laser:

  - **LiDAR de pulsos**. El proceso para la medición de la distancia entre el sensor y el terreno se lleva a cabo mediante la medición del tiempo que tarda un pulso desde que es emitido hasta que es recibido. El emisor funciona emitiendo pulsos de luz.
  - **LiDAR de medición de fase**. El emisor emite un haz laser continuo. Cuando recibe la señal reflejada, se mide la diferencia de fase entre la emitida y la reflejada. Conocida ésta, solo hay que calcular el número de longitudes de ondas enteras que ha recorrido.

  Por otra parte, también pueden clasificarse atendiendo al tipo de escaneado (véase Figura 6.1):

  - **Líneas**. Se dispone de un espejo rotatorio que va desviando el haz laser. Produce líneas paralelas en el terreno como patrón de escaneado. El principal inconveniente de este sistema es que al girar el espejo en una sola dirección no siempre se tienen mediciones.
  - ***ZigZag***. El espejo rota en dos sentidos (ida y vuelta). Produce líneas en *zigzag* como patrón de escaneado. Tiene la ventaja de que siempre está midiendo, pero al tener que cambiar de sentido de giro, la aceleración del espejo varía según su posición. Esto hace que en

las zonas cercanas al límite lateral de escaneado, la densidad de puntos sea mayor que en el centro.

- **De fibra óptica**. Desde la fibra central de un cable de fibra óptica, y con la ayuda de unos pequeños espejos, el haz laser es desviado a las fibras laterales montadas alrededor del eje. Este sistema produce una huella en forma de una especie de circunferencias solapadas. Al ser pequeños los espejos, la velocidad de captación de datos aumenta con respecto a los otros sistemas, pero el ángulo de escaneo disminuye.
- **Elíptico**. El haz laser es desviado por dos espejos que producen un patrón de escaneado elíptico.

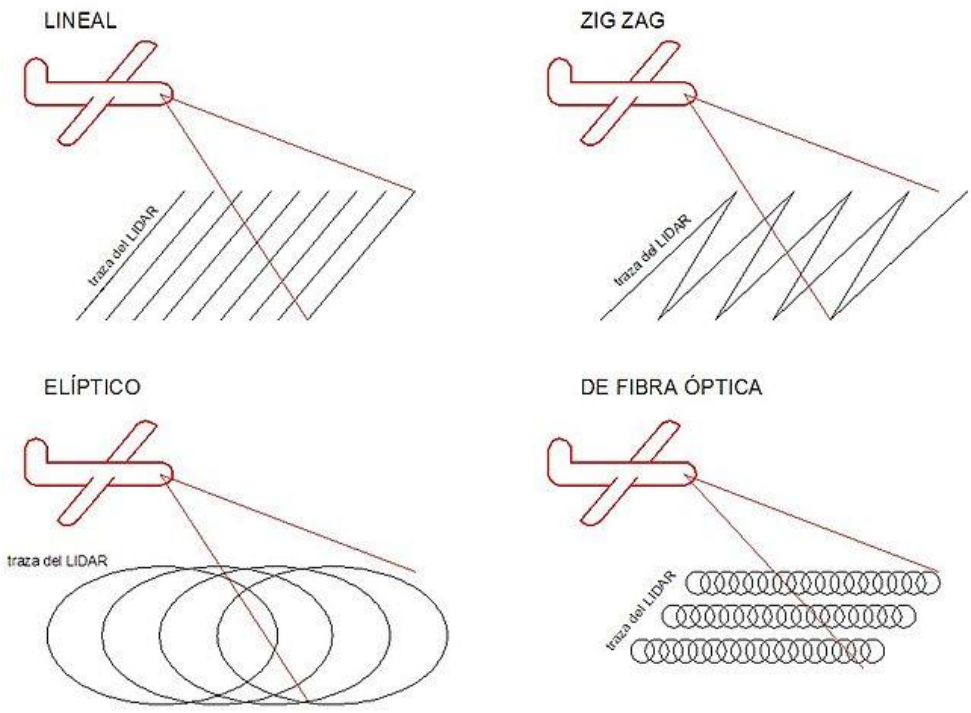

Figura 6.1: tipos de LiDAR según el tipo de escaneado. Fuente: Google.

- **Procesamiento *out-of-core***, Se refiere al procesamiento de datos voluminosos que no pueden contenerse a la vez en la memoria de un ordenador, por lo que hay que realizar una carga y proceso secuencial durante la ejecución del proceso. Este tipo de procesamiento es

especialmente útil para datos extremadamente voluminosos, usándose frecuentemente en el dominio de *Big Data.*

- **Inteligencia Artificial.** Se trata de la combinación de algoritmos, la simulación de procesos de inteligencia humana por parte de máquinas, en especial sistemas informáticos. La Inteligencia Artificial es un agente flexible capaz de percibir su entorno (a través de sensores) y llevar a cabo acciones que maximicen sus posibilidades de éxito (a través de actuadores) en algún objeto o tarea. Esos procesos pueden incluir **aprendizaje**, **razonamiento** y **autocorrección.**

  El término fue acuñado por **John McCarthy** en 1953 durante la **Conferencia de Darthmouth,** dando paso a la disciplina tal y como la conocemos en la actualidad. Actualmente, el término inteligencia artificial es más general y abarca desde la automatización de procesos hasta la robótica actual. Además, en los últimos años ha cobrado especial importancia debido al volumen masivo de datos que se está generando, así como a la variedad y velocidad con la que estos datos se producen.

  Según Russell y Norvig (Russell & Norvig, 2004), se distinguen varios tipos de inteligencia artificial:

  - **Sistemas que piensan como humanos.** Imitan el funcionamiento del sistema nervioso por medio de redes neuronales artificiales. Automatizan actividades como la toma de decisiones y la resolución de problemas.
  - **Sistemas que actúan como humanos.** Intentan realizar tareas de manera similar a como lo hacen los humanos y de forma más eficiente.
  - **Sistemas que piensan como humanos.** Tratan de imitar el pensamiento lógico humano. Se trata de investigar cómo lograr que las máquinas perciban, razonen y actúen en consecuencia.
  - **Sistemas que actúan racionalmente.** Aquellos capaces de percibir el entorno, que tratan de imitar de forma racional el comportamiento humano y actuar en consecuencia.

- **Aprendizaje Automático.** Se trata de una rama en particular de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que los ordenadores **aprendan**. Se dice que un agente aprende cuando su desempeño mejora con la **experiencia**, esto es, cuando la habilidad no estaba presente en su genotipo o rasgos de nacimiento.

  De forma más concreta, los investigadores de aprendizaje automático buscan algoritmos y heurísticas para convertir muestras de datos en programas, invirtiendo el paradigma de la programación clásica. A diferencia de un programa convencional, que recibe datos e instrucciones y produce una salida, un programa con naturaleza de aprendizaje automático recibe como entrada tanto los datos de entrada como la salida deseada, y el objetivo es obtener el conocimiento que induce a los datos de entrada a su correspondiente salida.

  En particular, destaca el aprendizaje profundo, impulsado principalmente por las **redes neuronales**.

- **Inferencia.** Haciendo referencia al aprendizaje automático, se trata de utilizar el conocimiento extraído durante el entrenamiento sobre nuevas muestras de datos no contempladas en dicho proceso, para obtener la salida correspondiente a esta muestra.

  Por ejemplo, en un problema de identificación del tipo de animal a partir de una imagen, consistiría en identificar qué animales aparecen en imágenes que no se han utilizado para enseñar en sistema.

- ***Backpropagation.*** Algoritmo utilizado para realizar el entrenamiento de los modelos de aprendizaje profundo, como las redes neuronales. Hasta su aparición en la década de los 80 del siglo pasado, y aunque las redes neuronales fueron propuestas en la década de los 40, era imposible entrenar los modelos pues era un proceso extremadamente complejo. Una vez se formula el algoritmo *backpropagation* se abre una de las puertas que dará paso a la expansión provocada en el aprendizaje profundo en los últimos años (Cun, 1988).

- **Interpretabilidad de un modelo.** La interpretabilidad de un modelo es un concepto estrechamente ligado a la inteligencia artificial, concretamente

minería de datos y aprendizaje automático. Existen diferentes representaciones del conocimiento en los procesos de aprendizaje, que a su vez dan paso a la taxonomía que clasifica a todos los algoritmos de aprendizaje o minería de datos.

Pueden utilizarse representaciones arbóreas, de reglas, en redes neuronales, etcétera. La interpretabilidad es un indicador que mide la capacidad de un modelo para ser interpretado por un experto en la materia. Por ejemplo, ante un sistema que es capaz de detectar si un paciente es positivo en COVID19 a partir de una serie de atributos de entrada como la temperatura corporal, respiración, etcétera, un médico ha de ser capaz de verificar que el conocimiento extraído durante el entrenamiento es correcto y por tanto comprender por qué el sistema emite un juicio u otro según los datos de entrada.

Las representaciones basadas en árboles y reglas, así como *kNN (k Nearest Neighbors*) permiten verificar de forma sencilla por qué una muestra de entrada es identificada como un tipo. Sin embargo, las redes neuronales son modelos de caja negra que hacen imposible verificar que el conocimiento extraído es correcto si no es a través de métricas como la precisión, cobertura, y medidas de error.

- **Perceptrón multicapa.** Es una red neuronal artificial formada por múltiples capas, de tal forma que tiene capacidad para resolver problemas que no son separables linealmente, lo cual solventa la principal limitación del perceptrón. El perceptrón multicapa puede estar totalmente o localmente conectado. En el primer caso, cada salida de una neurona en una capa es conectada como entrada a todas las neuronas de la siguiente capa, mientras que en el segundo caso cada neurona en una capa está conectadas con algunas de las neuronas en la siguiente capa (no todas).

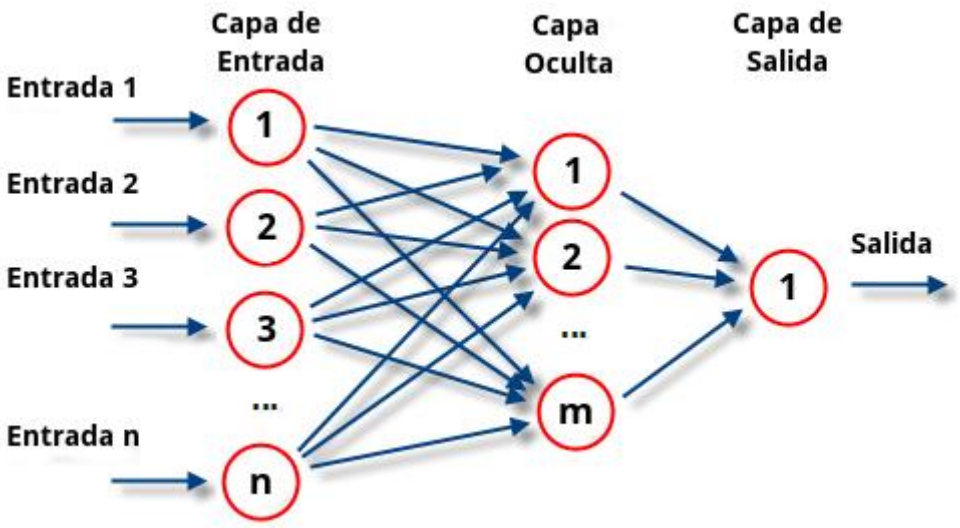

Figura 6.2: Perceptrón multicapa. Fuente: Google.

- **Malla de triángulos.** Es una colección de triángulos y vértices que aproximan una superficie tridimensional. Aunque el campo de aplicación de la generación automática de una malla regular, ha sido tradicionalmente la obtención de modelos digitales de elevaciones del terreno, su aplicación es mucho más amplia. Cualquier variable espacial relacionada con una cierta tipología es susceptible de ser modelizada como una superficie tridimensional, en la que la cota de cada punto es el valor de la variable a estudiar.

  Se utilizan habitualmente en gráficos por computador, que se componen de un conjunto de triángulos que están conectados por sus vértices.

- **RBM (*Restricted Boltzmann Machine*).** Es un tipo de red neuronal recurrente estocástica. Su nombre fue acuñado por Geoffrey Hinton y Terry Sejnowski. Las máquinas de Boltzmann pueden considerarse como la contrapartida estocástica y generativa de las redes de Hopfield. Fueron los primeros tipos de redes neuronales capaces de aprender mediante representaciones internas, son capaces de representar y resolver complicados problemas combinatorios.

  Sin embargo, las máquinas de Boltzmann sin restricciones de conectividad no han demostrado ser útiles para resolver problemas que se dan en la

práctica en el aprendizaje o inferencia de las máquinas. Van asociadas a problemas de aprendizaje no supervisado.

7 BIBLIOGRAFÍA

Besl, P. J., & Jain, R. C. (1988). Segmentation Through Variable-Order Surface Fitting. *IEEE Trans. Pattern Anal. Mach. Intell.*, *10*(2), 167–192. https://doi.org/10.1109/34.3881

Biasutti, P., Bugeau, A., Aujol, J.-F., & Brédif, M. (2019). RIU-Net: Embarrassingly simple semantic segmentation of 3D LiDAR point cloud. *CoRR*, *abs/1905.0*. Retrieved from http://arxiv.org/abs/1905.08748

Biosca, J., & Lerma, J. (2008). Unsupervised robust planar segmentation of terrestrial laser scanner point clouds based on fuzzy clustering methods. *ISPRS Journal of Photogrammetry and Remote Sensing*, *63*, 84–98. https://doi.org/10.1016/j.isprsjprs.2007.07.010

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric Deep Learning: Going beyond Euclidean data. *{IEEE} Signal Process. Mag.*, *34*(4), 18–42. https://doi.org/10.1109/MSP.2017.2693418

Cun, Y. Le. (1988). *A Theoretical Framework for Back-Propagation*.

Díaz-Medina, M., Fuertes-García, J. M., Ogayar-Anguita, C. J., & Lucena, M. (2019). A Voxel-based Deep Learning Approach for Point Cloud Semantic Segmentation. In D. Casas & A. Jarabo (Eds.), *Spanish Computer Graphics Conference (CEIG)*. https://doi.org/10.2312/ceig.20191206

Eitel, A., Springenberg, J., Spinello, L., Riedmiller, M., & Burgard, W. (2015). *Multimodal Deep Learning for Robust RGB-D Object Recognition*.

Filin, S. (2012). Surface clustering from airborne laser scanning data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *34*.

Fischler, M. A., & Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, *24*(6), 381–395. https://doi.org/10.1145/358669.358692

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 27* (pp. 2672–2680). Retrieved from http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Griffiths, D., & Boehm, J. (2019). A Review on Deep Learning Techniques for 3D Sensed Data Classification. *Remote Sensing*, *11*, 1499. https://doi.org/10.3390/rs11121499

Hackel, T., Savinov, N., Ladicky, L., Wegner, J. D., Schindler, K., & Pollefeys, M. (2017). SEMANTIC3D.NET: A new large-scale point cloud classification benchmark. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial*

Information Sciences, *IV-1-W1*, 91–98.

He, K., Zhang, X., Ren, S., & Sun, J. (2014). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *CoRR*, *abs/1406.4*. Retrieved from http://arxiv.org/abs/1406.4729

Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., … Markham, A. (2020). RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Hua, B.-S., Tran, M.-K., & Yeung, S.-K. (2018). *Pointwise Convolutional Neural Networks*. 984–993. https://doi.org/10.1109/CVPR.2018.00109

Huang, J., & You, S. (2016). *Point Cloud Labeling using 3D Convolutional Neural Network*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 25* (pp. 1097–1105). Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Lan, S., Yu, R., Yu, G., & Davis, L. S. (2018). Modeling Local Geometric Structure of 3D Point Clouds using Geo-CNN. *CoRR*, *abs/1811.0*. Retrieved from http://arxiv.org/abs/1811.07782

Li, Y., Bu, R., Sun, M., Wu, W., Di, X., & Chen, B. (2018). PointCNN: Convolution on X-Transformed Points. *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 828–838. Red Hook, NY, USA: Curran Associates Inc.

Liu, W., Sun, J., Li, W., Hu, T., & Wang, P. (2019). Deep Learning on Point Clouds and Its Application: A Survey. *Sensors*, *19*(19). https://doi.org/10.3390/s19194188

Lv, D., Ying, X., Cui, Y., Song, J., Qian, K., & Li, M. (2017). Research on the technology of LIDAR data processing. *2017 First International Conference on Electronics Instrumentation Information Systems (EIIS)*, 1–5. https://doi.org/10.1109/EIIS.2017.8298694

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. https://doi.org/10.1007/BF02478259

Nguyen, A. T. Le, & Le, H. B. (2013). 3D point cloud segmentation: A survey. *2013 6th IEEE Conference on Robotics, Automation and Mechatronics (RAM)*, 225–230.

Özyesil, O., Voroninski, V., Basri, R., & Singer, A. (2017). A Survey on Structure from Motion. *CoRR*, *abs/1701.0*. Retrieved from http://arxiv.org/abs/1701.08493

Qi, Charles R, Su, H., Mo, K., & Guibas, L. J. (2016). PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *ArXiv Preprint ArXiv:1612.00593*.

Qi, Charles Ruizhongtai, Yi, L., Su, H., & Guibas, L. J. (2017). PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems 30* (pp. 5099–5108). Curran Associates, Inc.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention -- MICCAI 2015* (pp. 234–241). Cham: Springer International Publishing.

Russell, S. J., & Norvig, P. (2004). *Inteligencia artificial: un enfoque moderno*. Retrieved from https://books.google.es/books?id=yZCVPwAACAAJ

S. Pressman, R. (2006). *Ingeniería del Software Un Enfoque Práctico*.

Shahzad, M., & Zhu, X. (2015). Robust Reconstruction of Building Facades for Large Areas Using Spaceborne TomoSAR Point Clouds. *Geoscience and Remote Sensing, IEEE Transactions On*, *53*, 752–769. https://doi.org/10.1109/TGRS.2014.2327391

Shahzad, M., Zhu, X., & Bamler, R. (2012). Façade structure reconstruction using spaceborne TomoSAR point clouds. *International Geoscience and Remote Sensing Symposium (IGARSS)*, 467–470. https://doi.org/10.1109/IGARSS.2012.6351385

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*.

Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.-H., & Kautz, J. (2018). SPLATNet: Sparse Lattice Networks for Point Cloud Processing. *CVPR*, 2530–2539. Retrieved from http://dblp.uni-trier.de/db/conf/cvpr/cvpr2018.html#SuJSMK0K18

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going Deeper with Convolutions. *CVPR*.

Te, G., Hu, W., Zheng, A., & Guo, Z. (2018). RGCNN: Regularized Graph CNN for Point Cloud Segmentation. *Proceedings of the 26th ACM International Conference on Multimedia*, 746–754. https://doi.org/10.1145/3240508.3240621

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., & Guibas, L. J. (2019). KPConv: Flexible and Deformable Convolution for Point Clouds. *The IEEE International Conference on Computer Vision (ICCV)*.

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using {t-SNE}. *Journal of Machine Learning Research*, *9*, 2579–2605. Retrieved from http://www.jmlr.org/papers/v9/vandermaaten08a.html

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2018). Dynamic Graph {CNN} for Learning on Point Clouds. *CoRR*, *abs/1801.0*. Retrieved from http://arxiv.org/abs/1801.07829

Wu, W., Qi, Z., & Fuxin, L. (2019). PointConv: Deep Convolutional Networks on 3D Point Clouds. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xie, Y., Tian, J., & xiang Zhu, X. (2019). A Review of Point Cloud Semantic Segmentation. *ArXiv*, *abs/1908.0*.

Xu, Y., Fan, T., Xu, M., Zeng, L., & Qiao, Y. (2018). SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. *CoRR*, *abs/1803.1*.

Retrieved from http://arxiv.org/abs/1803.11527

Yang, Y., Feng, C., Shen, Y., & Tian, D. (2017). *FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation*. Retrieved from http://arxiv.org/abs/1712.07262

Ye, X., Li, J., Huang, H., Du, L., & Zhang, X. (2018). 3D Recurrent Neural Networks with Context Fusion for Point Cloud Semantic Segmentation. *The European Conference on Computer Vision (ECCV)*.

Zou, X., Cheng, M., Wang, C., Xia, Y., & Li, J. (2017). Tree Classification in Complex Forest Point Clouds Based on Deep Learning. *IEEE Geoscience and Remote Sensing Letters*, *14*. https://doi.org/10.1109/LGRS.2017.2764938