TRABAJOS FIN DE MÁSTER

Universidad de Jaén

Escuela Politécnica Superior de Jaén

EXPERIMENTACIÓN EN FRAMEWORK PARA REDES NEURONALES CON DIFERENTES CONJUNTOS DE DATOS LIDAR REALES Y SINTÉTICOS

Autor: Víctor Rodríguez Cano

Máster: Ingeniería Informática

Tutores: Rafael Jesús Segura Sánchez y Alfonso López Ruiz
Departamento: Departamento de informática

Fecha: 01/12/2024

Universidad de Jaén

Departamento de Informática

Don Rafael Jesús Segura Sánchez y Don Alfonso López Ruiz, tutores del Trabajo Fin de Máster titulado: '**Experimentación en framework para redes neuronales con diferentes conjuntos de datos LiDAR reales y sintéticos**', que presenta Don Víctor Rodríguez Cano, otorgan el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Diciembre de 2024

El alumno: Los tutores:

Víctor Rodríguez Cano

Rafael Jesús Segura Sánchez

Alfonso López Ruiz

Agradecimientos

Desde aquí aprovecho para envíar mis agradecimientos a todos aquellos que estuvieron a mi lado a lo largo de estos años, empezando por mis padres y a toda mi familia. Siempre habeis estado ahí apoyándome incondicionalmente y es por ello que he podido alcanzar este punto en mi vida.

A todos mis amigos, tanto a los más antiguos como a aquellos que conocí más recientemente. Siempre es un gusto estar con vosotros, pasando mi tiempo libre y en ocasiones no tan libre, porque sin vosotros siento que la vida pierde todo su color.

Muy agradecido también con mis compañeros de laboratorio a los que nunca rechacé un buen desayuno para olvidarnos de las penurias en las que nuestros proyectos se veían inmersos. Aunque solo hayan sido unos meses, me ha encantado trabajar con vosotros y sin duda ha sido una experiencia inolvidable en la que espero que lo único malo en todo ese tiempo hayan sido las impresiones 3D fallidas en caliza.

Finalmente, quiero agradecer a los profesores de la Universidad de Jaén que coincidieron conmigo de alguna manera, haciendo mención especial a mis tutores, Rafa y Alfonso, así como también a Carlos por haberme acompañado a lo largo de estos meses de investigación. Gracias por haberme ayudado a lo largo de estos meses, respondiéndome todas aquellas dudas e inquietudes que surgían.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Máster en Ingeniería Informática
Modalidad	Trabajo Teórico/Experimental
Idioma	Español
Tipo	Específico
TFT en equipo	No
Autor/a	Víctor Rodríguez Cano
Fecha de asignación	18/11/2024
Descripción corta	A día de hoy el aprendizaje automático está aplicándose dentro de muchos sectores para resolver una infinidad de problemas de ingeniería, arquitectura, medicina, arqueología, geomática, etc. Sin embargo, el entrenamiento de estos modelos de aprendizaje automático requiere de una gran cantidad de datos correctamente procesados y etiquetados. Esta investigación tendrá como objetivo reducir este problema en el contexto de entornos urbanos, combinando el uso de nubes de puntos LiDAR (Laser Detection and Ranging) reales y sintéticas. Para ello se proponen 2 hipótesis iniciales que se tratarán de validar: 1) El comportamiento de las redes obtenido a partir de un entrenamiento y testeo con nubes de puntos sintéticas generadas con el LiDAR virtual es similar al obtenido a partir de nubes de puntos reales. 2) El entrenamiento de las redes neuronales con nubes de puntos reales puede ser sustituido con un entrenamiento alimentado con datos completamente sintéticos, dando buenos resultados en el testeo con nubes de puntos reales. Entre los puntos más destacados, se abarcarán desafíos tales como la generación procedural de fragmentos de ciudades etiquetadas, la aplicación de un simulador LiDAR para la creación de un *dataset* sintético y la experimentación siguiendo el proceso *Knowledge Discovery in Databases* (KDD) con múltiples redes neuronales: PointNet++, Point Transformer y RepSurf.

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 INTRODUCCIÓN

El desembolso millonario anual por parte de las grandes empresas muestra el potencial de esta área de la informática, el aprendizaje automático. Entre sus aplicaciones se encuentran los vehículos autónomos, la prevención de tumores, la traducción en tiempo real, el contenido multimedia autogenerado, la asistencia virtual, etc. Impensable hasta hace poco, lo cierto es que hoy vivimos en medio de una revolución liderada por la inteligencia artificial (IA).

Los avances más punteros dentro de la IA están asociados con las redes neuronales y *deep learning* (DL), pudiéndose aplicar para una gran cantidad de contextos donde resolver problemas de diferente naturaleza. Sin embargo, estos algoritmos bioinspirados requieren de un proceso de aprendizaje basado en grandes cantidades de datos que no siempre son fáciles de conseguir. Esto nos lleva a uno de los principales problemas de estos algoritmos: los conjuntos de datos utilizados para la llamada fase de entrenamiento.

Este proyecto se centrará en la creación de un conjunto de datos sintético y en la experimentación con redes neuronales aplicadas a nubes de puntos, contribuyendo en paradigmas de gran relevancia actualmente como el de la conducción automática, la monitorización de proyectos de construcción o el análisis de la infraestructura y espacios verdes.

1.1 Motivación

La motivación principal no es sólo la de mostrar los conocimientos y la madurez adquirida tras finalizar el máster en ingeniería informática, sino que además con este trabajo se persigue contribuir significativamente en la reducción de los costes económicos y temporales que conlleva crear estos *datasets* de nubes de puntos etiquetadas con técnicas tradicionales.

En efecto, con ello se busca contribuir a la viabilidad del uso de las redes neuronales aplicadas al problema de segmentación de nubes de puntos dentro de entornos urbanos mediante la creación automatizada de un conjunto de datos

sintéticos y etiquetados procedentes de un sensor LiDAR (Laser Detection and Ranging) virtual.

1.2 Objetivos del proyecto

El objetivo principal es realizar un estudio experimental que permita impulsar el uso de las redes neuronales aplicadas a nubes de puntos en el ámbito de los entornos urbanos. Para ello destacaremos una serie de objetivos específicos que se tendrán en cuenta para la base del proyecto:

1. **Objetivo 1 (O1)**. Gestión y coordinación: se garantizará el establecimiento de un sistema de seguimiento y supervisión que asegure el cumplimiento de los plazos, presupuesto y recursos asignados, facilitando así la toma de decisiones, la cooperación y la evaluación continua del progreso.
2. **Objetivo 2 (O2)**. Obtención de modelos urbanos procedurales etiquetados: se generarán fragmentos de ciudades a través de técnicas procedurales con etiquetado de los objetos que los componen, siendo posible la traducción entre diferentes conjuntos de etiquetas.
3. **Objetivo 3 (O3)**. Elaboración de un *dataset* sintético: creación de un *dataset* constituido por multitud de nubes de puntos sintéticas obtenidas de un sensor LiDAR virtual en entornos urbanos, con la variabilidad suficiente como para ser de utilidad en la generación de modelos.
4. **Objetivo 4 (O4)**. preparación, adaptación y análisis del funcionamiento de los proyectos actuales de redes neuronales basados en nubes de puntos LiDAR.
5. **Objetivo 5 (O5)**. experimentación mediante el entrenamiento de múltiples redes neuronales y evaluación comparativa entre los resultados de los modelos obtenidos de diferentes conjuntos de datos reales y sintéticos.

A la vista de los objetivos presentados, es evidente que el proyecto que se busca realizar, si bien se trata de un trabajo teórico experimental, requiere de un desarrollo de software específico para la construcción procedural de entornos urbanos sintéticos y etiquetados que se puedan utilizar por el simulador LiDAR.

1.3 Estructuración de la memoria

A lo largo de esta memoria se presentará el resultado del proceso de investigación y experimentación llevado a cabo a lo largo de estos últimos meses. En primer lugar, la sección 2 (Antecedentes y estado del arte) situará al proyecto presentando los trabajos, investigaciones y avances que lo hicieron posible, siendo un apartado separado en 3 bloques temáticos: generación procedural (Generación procedural de ciudades sintéticas), conjuntos de datos (Conjuntos de datos LiDAR para entornos urbanos) y redes neuronales (Redes neuronales artificiales para nubes de puntos). Una vez conocidos los antecedentes, se detallarán las especificaciones del proyecto en el punto 3 (Especificación del trabajo), analizando puntos clave como son la metodología utilizada, decisiones tomadas o el presupuesto. La parte más práctica del proyecto se ubica en los 2 epígrafes posteriores. El primero de ellos, el apartado 4 (Generador de fragmentos de ciudades), tratará sobre el diseño y desarrollo en Unity de la aplicación de fragmentado de ciudades procedurales. Una vez en posesión de los fragmentos etiquetados se alcanza la sección 5 (Experimentación con redes neuronales), donde se harán un conjunto de pruebas experimentales siguiendo las fases del *Knowledge Discovery in Databases* (KDD) con varias redes para finalmente terminar el estudio con los apartados 6 (Resultados y discusión) y 7 (Conclusiones y trabajos futuros), en los cuales se presentará un resumen a nivel global del proyecto, unas conclusiones y un conjunto de trabajos futuros.

2 ANTECEDENTES Y ESTADO DEL ARTE

Esta investigación parte del uso de un sensor LiDAR virtual que debe ejecutarse de igual manera sobre un escenario virtual. El diseño y creación manual de entornos urbanos es una tarea que requiere de bastante tiempo, por lo que es mucho mejor definir un conjunto de reglas para generarlos proceduralmente.

En particular este proyecto se sumergirá en múltiples temas de investigación en los que predominarán la generación procedural de ciudades, los *datasets* actuales de entornos urbanos y las redes neuronales. En lo consecutivo, se hará un breve análisis de cada uno de estos ámbitos para comprender mejor el contexto del trabajo.

2.1 Generación procedural de ciudades sintéticas

La generación de ciudades sintéticas mediante técnicas procedurales es un área de estudio que lleva muchos años activa y sigue en auge dadas las aplicaciones y posibilidades que ofrece. Es por esto se han ido proponiendo múltiples estrategias para este fin dentro de la literatura.

2.1.1 Sistemas de reglas y L-System

Recordando algunos de los artículos más influyentes, Parish y Müller propusieron un artículo disruptivo en el año 2001, “Procedural Modeling of Cities” [1], en el cual presentaron los L-System como una solución para la generación procedural de ciudades. La estructura de un L-System tiene 3 partes fundamentales que deben conocerse:

- Axioma: cadena o estado inicial del sistema desde el cual se comienzan a aplicar las reglas de producción. Esta cadena inicial puede ser una secuencia simple, como "P" (plaza) o algo más complejo, dependiendo del patrón que se desea generar.
- Alfabeto: conjunto de símbolos que el sistema utiliza en las reglas de producción. Estos símbolos representan diferentes acciones; por ejemplo, en un L-System algunos símbolos comunes podrían ser F (avance de N unidades en línea recta), L (giro en ángulo recto), O (rotonda), etc.

- Reglas de producción: instrucciones que definen cómo cada símbolo del alfabeto debe transformarse en la siguiente iteración. Cada símbolo tiene reglas específicas asociadas que especifican cómo se expande con otros símbolos. Un par de reglas para F podrían ser {F → F, F → FL, F → FOF}.

Dicho esto, el artículo proponía un sistema de reglas que, dada una entrada con información del terreno, fuese capaz de dividirlo y crear la geometría apropiada tanto para el puerto carretero como para los edificios. Esto permitía tener en cuenta parámetros como la densidad de población, el tipo de ciudad o incluso el estilo de barrio. En otras palabras, las reglas de producción aplicadas dentro de estos L-System permitían aplicar métodos globales y locales para alcanzar una serie de objetivos, introduciendo coherencia general a la vez que un buen nivel de detalles.

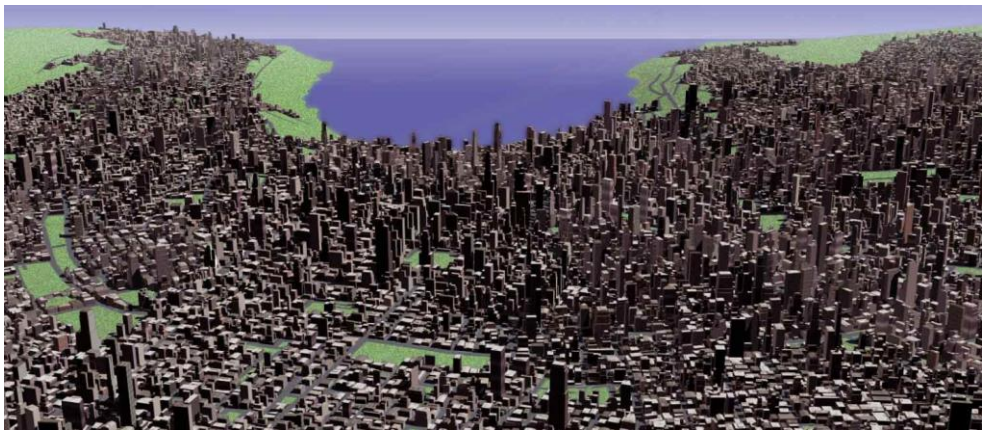

Ilustración 2.1: Ciudad creada con L-System *[1]*

Por otro lado, se encuentra “Real-time Procedural Generation of Pseudo Infinite Cities” [2], un artículo del año 2003 que proponía un enfoque para maximizar la variabilidad de la geometría de los edificios basándonos en los parámetros de ajuste y la posición. Estas variadas construcciones para edificios se conseguían con la extrusión de un conjunto de planos de planta creados aleatoriamente, dando lugar a edificios con geometría plural entre diferentes plantas. Aunque no se puede decir que fue rompedor su planteamiento en cuanto al puerto carretero, pues se trataba simplemente de una cuadrícula, sí que es cierto que supuso un avance en la generación de geometría para edificios.

Ilustración 2.2: Geometría de edificios según parámetros de ajuste y posición [2]

2.1.2 Diagramas de Voronoi

Otro enfoque que difiere en la generación de puertos carreteros vista es propuesta en 2006. Con el artículo “Duplicating road patterns in south african informal settlements using procedural techniques” [3], un trabajo que se centra en el estudio de los patrones de asentamientos más informales dentro de complejos entornos urbanos, como es el caso de algunas ciudades de Sudáfrica. Con este artículo, se propone el uso de diagramas de Voronoi para replicar estas estructuras urbanas, concepto que se combina con los L-System para obtener un patrón más cercano al de los asentamientos informales estructurados.

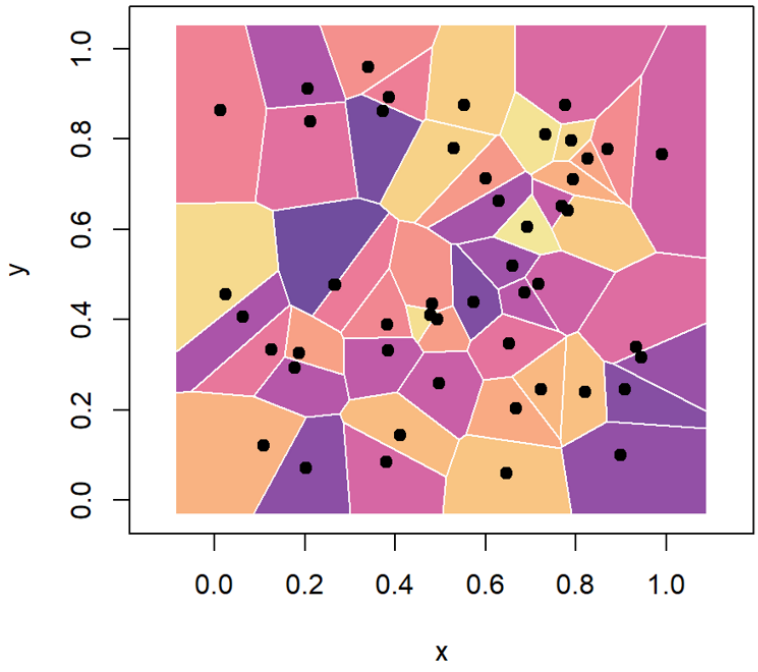

Ilustración 2.3: Diagrama de Voronoi [3]

2.1.3 Sistemas multiagente

Una tendencia también muy interesante es la que se marcaba con “Procedural City Modeling” [4] en el año 2003 por parte de Lechner et al., una visión que, en lugar de generar ciudades sobre la base del puerto carretero, lo hacía en función del suelo y la distribución de edificios. Fue un enfoque que buscó aumentar el realismo de las ciudades generadas proceduralmente y que traía como característica clave el uso de agentes inteligentes para la generación de ciudades. En este artículo propone un sistema de agentes extensible capaces de interactuar entre sí, cada uno de ellos con la capacidad de aplicar un simple conjunto de reglas y que en potencia se vuelve complejo durante la interacción con el resto de agentes.

2.1.4 Funciones de colisión de ondas

Otra técnica más reciente es la propuesta de 2016, “Wave Function Collapse Algorithm” [5], publicada por Maxim Gumin en GitHub. Aunque no está orientado directamente a la creación de ciudades, fue un algoritmo popular para la generación procedural que podría aplicarse también para este problema. Pese a su nombre, realmente no está completamente relacionado con la mecánica cuántica, aunque sí que se inspira en la idea del colapso de onda para aplicar una generación por procedimientos.

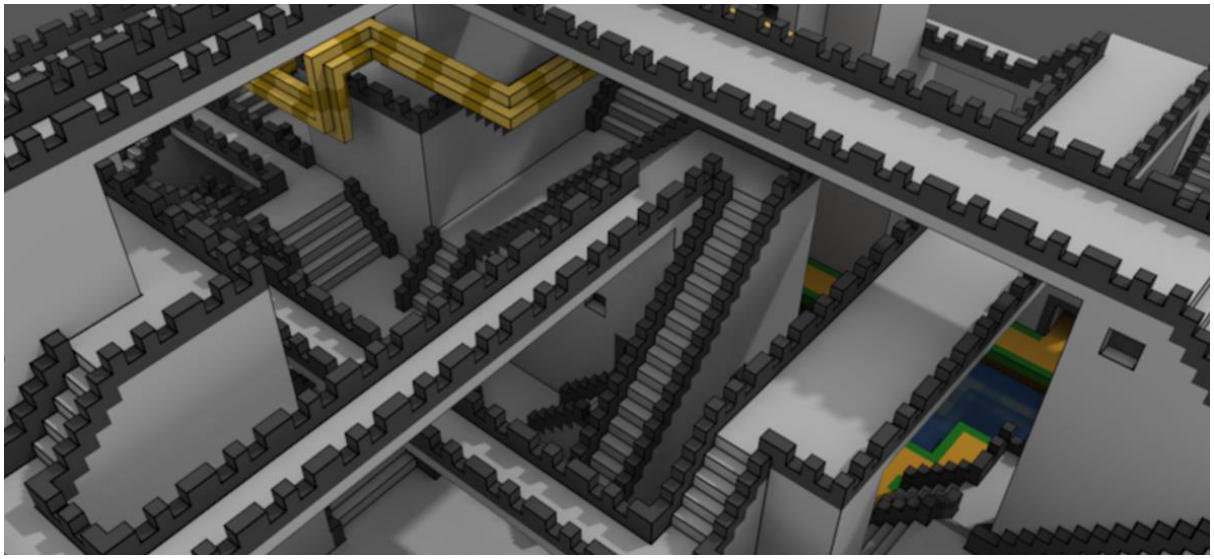

Ilustración 2.4: Generación de fortaleza infinita con funciones de colisión de onda *[4]*

2.2 Conjuntos de datos LiDAR para entornos urbanos

Los datos etiquetados son fundamentales para el éxito del aprendizaje supervisado, pues permiten a los algoritmos aprender a identificar patrones y realizar predicciones basadas en un conjunto de características específicas como pueden ser la geometría de una nube de puntos. Los datos etiquetados proporcionan a los modelos de IA la información necesaria para entrenar de manera estructurada, ayudando a establecer relaciones claras entre información espacial y etiquetas.

En el contexto de este proyecto nos centraremos en *datasets* con nubes de puntos obtenidas mediante tecnología LiDAR captadas en exteriores y en particular, en zonas urbanas. Aunque exista una gran cantidad de nubes de puntos en proyectos como lo son el PNOA (Plan Nacional de Ortofotografía Aérea [6]), la realidad es que normalmente no están etiquetados, siendo de poca utilidad para el entrenamiento de redes neuronales. Pese a ello, existen algunos *datasets* etiquetados y orientados al problema de segmentación que merecen ser descritos brevemente.

2.2.1 *Dataset* Semantic3D

Comenzamos el análisis destacando el *dataset* de Semantic3D [7], presentado en 2017 por investigadores de la Universidad de Bonn como un *benchmark* para la clasificación semántica de nubes de puntos en exterior. Contiene más de 4 mil millones de puntos etiquetados en 8 clases diferentes. Sus nubes fueron registradas con un sensor LiDAR estático y posteriormente etiquetadas por profesionales, consiguiendo nubes etiquetadas para entornos naturales y escenas urbanas con gran variedad de estructuras como iglesias, calles, vías de tren, plazas, pueblos o castillos.

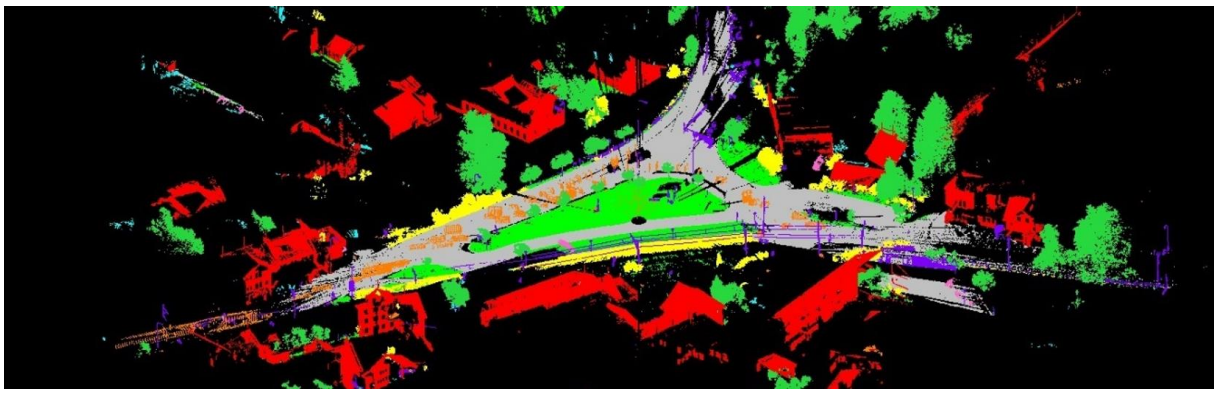

***Ilustración 2.5: Nube de Semantic3D** [7]*

2.2.2 *Dataset* Semantic KITTI

Con un nombre muy similar tenemos Semantic KITTI [8], creado en 2019 como una extensión del célebre conjunto KITTI [9]. A diferencia del conjunto anterior, Semantic KITTI fue medido con un sensor LiDAR montado en un vehículo en circulación como un sistema láser móvil (MLS), de manera que era capaz de tomar las mediciones a una velocidad de 10Hz. Esto lo hace muy interesante para problemas como el de la conducción autónoma. Cabe mencionar que es un *dataset* muy pesado dada la densidad de sus nubes, ocupando cerca de 80GB con una media de alrededor de 4,549 millones de puntos por nube. Estos datos están organizados en 28 clases, distinguiéndose objetos inmóviles (22 clases) y móviles (6 clases restantes). En general, este conjunto cubre usuarios de tráfico, áreas de estacionamiento, aceras, etc.

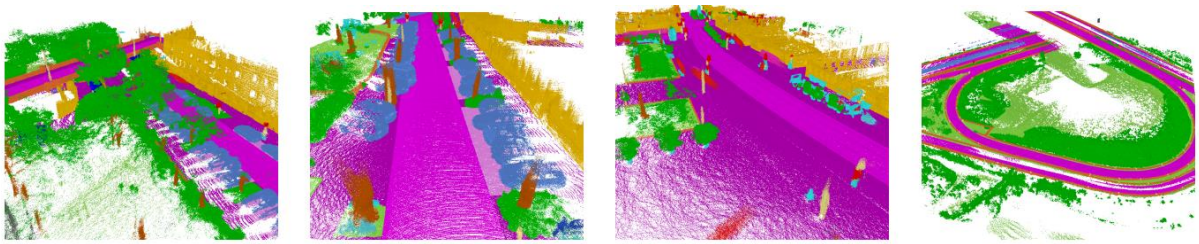

***Ilustración 2.6: Nubes de Semantic KITTI** [9]*

2.2.3 *Dataset* Paris-Lille-3D

Paris-Lille-3D [10] es un conjunto de nubes de puntos centrado en escenas urbanas de ciudades francesas. Fue creado en 2017 para contribuir con el avance de problemas para la segmentación y clasificación automatizada de nubes de puntos. Al igual que sucedía con Semantic KITTI, estos datos han sido generados por un MLS circulando por las calles de París y Lille. Contiene 143 millones de puntos etiquetados a mano y cuenta con 50 clases diferentes. No obstante, el conjunto se publicó con una transformación para conservar solo las 10 clases generales más representativas.

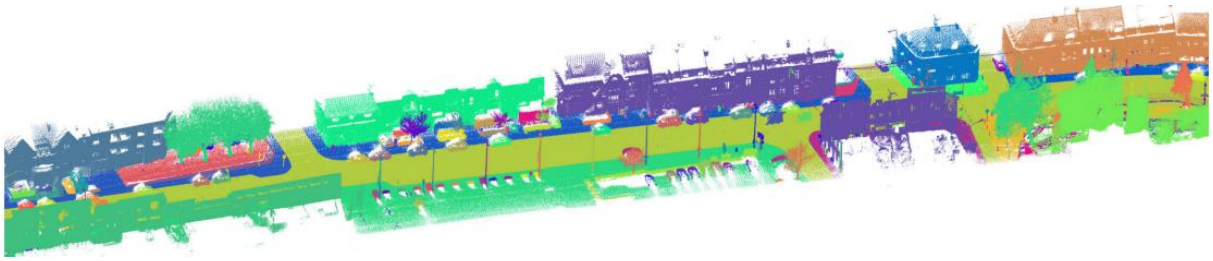

Ilustración 2.7: Nube de Paris-Lille-3D [10]

2.2.4 *Dataset* DublinCity

Publicado por investigadores de la University College Dublin en 2019, DublinCity [11] es otro conjunto de datos de nubes de puntos a gran escala capturado en la ciudad de Dublín (Irlanda) que está dirigido a la reconstrucción 3D y la creación de modelos digitales de ciudades. Supone una nube etiquetada de más de 260 millones de puntos obtenidos de las mediciones de un sensor LiDAR aéreo de alta densidad a lo largo del año 2015. Tiene 13 clases que se organizan en niveles jerárquicos de detalle, desde elementos generales a otros específicos.

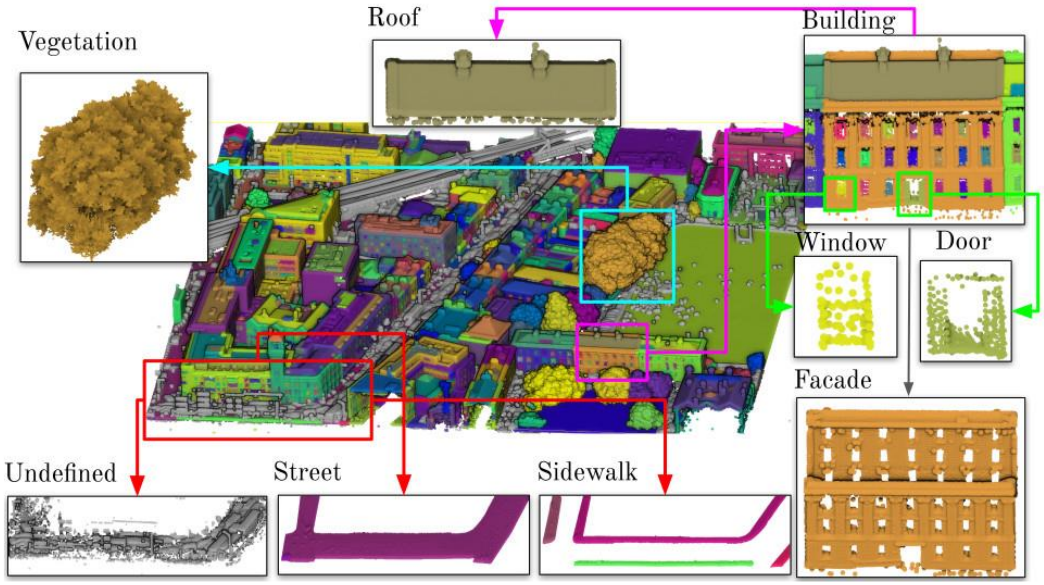

Ilustración 2.8: Estructura de las nubes de puntos en [11]

Tal y como se observa en la Ilustración 2.8, se pueden encontrar etiquetas de alto nivel como *Building* o *Vegetation*, mientras que también se pueden encontrar a su vez etiquetas de bajo nivel como *Window*. Esto convierte al *dataset* no solo en una herramienta para reconstrucción y modelado, sino también para su uso en entrenamientos de redes neuronales.

2.2.5 *Dataset* Toronto-3D

Terminamos finalmente presentando el *dataset* más influyente para este proyecto, Toronto-3D [12], un conjunto de nubes de puntos urbanas adquirido por un sistema LiDAR móvil en la ciudad de Toronto, Canadá. El conjunto fue creado por la Universidad de Toronto en el año 2020 y es uno de los más recientes. Este conjunto está completamente etiquetado y pensado para su uso en problemas de segmentación, cubriendo aproximadamente 1 km de carretera con unos 78.3 millones de puntos (3GB de datos repartidos en 4 archivos). Sin duda es un buen *dataset* para realizar entrenamientos supervisados, pues incorpora nubes de puntos con mucha información. Más concretamente, cada uno de los puntos de sus nubes contiene 10 atributos útiles para el entrenamiento como son la posición, el color, la intensidad o el GPS, así como también se asocian a las 8 clases de etiquetas.

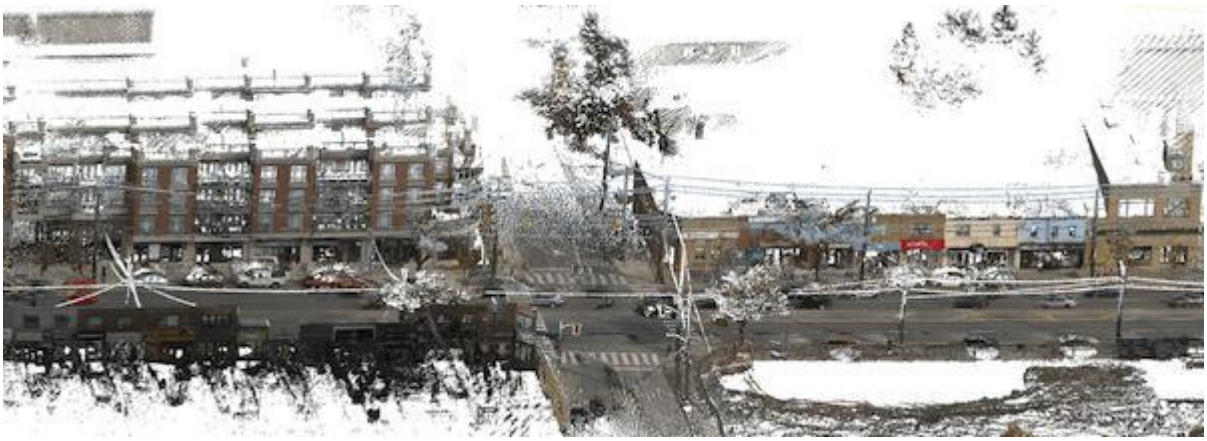

Ilustración 2.9: Nube de Toronto-3D *[12]*

2.3 Redes neuronales artificiales para nubes de puntos

Las redes neuronales son algoritmos de inteligencia artificial inspirados en el funcionamiento del cerebro humano, diseñados para procesar y aprender de grandes volúmenes de datos. Cada red está compuesta por unidades denominadas neuronas artificiales organizadas en capas, donde cada neurona recibe señales, las procesa y genera una salida que pasa a la siguiente capa. Este proceso permite a las redes aprender patrones en los datos mediante un ajuste continuo de los pesos, que representan la importancia de cada conexión entre neuronas.

Aunque surgieron de manera teórica en la década de los 70, podemos decir que en la actualidad estas proveen soluciones a una gran cantidad de campos, tales como el procesamiento del lenguaje natural, con redes como ChatGPT [13], visión artificial con Swin Transformer [14], creación de imágenes con DALL-E [15], generación de vídeos con SORA [16], detección de voz con Whisper [17], generación de audio con Suno [18], interpolación de fotogramas en películas de animación con ToonCrafter [19], etc. Cada una de ellas con una arquitectura única, aunque podemos considerar que las arquitecturas más comunes son las siguientes:

- Redes neuronales artificiales, Artificial Neural Networks (ANN): siendo la base de redes más complejas, estas están compuestas por capas de neuronas con conexiones densas para usos en tareas de clasificación y regresión.
- Redes neuronales convolucionales, Convolutional Neural Network (CNN): utilizadas principalmente para procesamiento de imágenes y visión artificial, estas redes aplican operaciones de convolución para identificar patrones en datos espaciales.
- Redes neuronales generativas adversarias, Generative Adversarial Networks (GAN): compuestas por dos redes que compiten entre sí. Una de ellas es la red generadora que tiene como misión crear datos, y la otra es la red discriminadora que intenta distinguir entre datos reales y generados. La red generadora debe ser capaz de crear datos de calidad que permitan “engañar” a la discriminadora.
- Redes neuronales recurrentes, Recurrent Neural Networks (RNN): diseñadas para procesar secuencias de datos, cuentan con conexiones recurrentes que pueden almacenar la información previa, lo cual permite comprender dependencias a lo largo del tiempo. Sin embargo, estas redes poseían el problema de localización de patrones a largo plazo, surgiendo las Long Short-Term Memory (LSTM), o también llamadas memorias a largo y corto plazo, como una extensión de las RNN.
- Redes neuronales de grafos, Graph Neural Networks (GNN): son redes neuronales que están planteadas para trabajar con datos que permiten ser

representados como un grafo, de tal manera que los nodos representan entidades individuales y las aristas representan las relaciones entre estas.

- AutoEncoders: están diseñados para reducir la dimensionalidad de los datos y reconstruir la entrada original, extrayendo características relevantes. Son útiles en compresión de datos, eliminación de ruido y detección de anomalías.
- Transformers: utilizan mecanismos de atención para procesar secuencias sin necesidad de recurrencia, lo que las hace mucho más eficientes en tareas de procesamiento de lenguaje natural. Desde su publicación, estas redes han hecho que las redes RNN hayan quedado en un segundo plano a favor de la arquitectura Transformer.

Conociendo ligeramente las arquitecturas que predominan en el campo, se van a presentar una serie de redes neuronales aplicadas a nubes de puntos que incorporan algunas de estas estructuras para la resolución de problemas típicos como la clasificación y la segmentación.

2.3.1 Redes con arquitectura convolucional: PointNet

Introducida por Qi et al. en 2017 con su artículo “PointNet: DL on Point Sets for 3D Classification and Segmentation” [20], marcó un avance significativo frente a las redes convolucionales tradicionales. Anteriormente, las redes requerían de estructuras ordenadas, sin embargo, esta red era capaz de procesar directamente nubes de puntos con una naturaleza desordenada.

Este tipo de redes se fundamentan en el uso de convoluciones, siendo esta una operación matemática que permite extraer características importantes de la vecindad de los datos de entrada, algo que se consigue con un kernel. Mediante el desplazamiento del núcleo de convolución por el espacio se consigue generar un mapa de características que permite la detección de patrones, sin importar su posición exacta. En el caso concreto de PointNet se emplea una combinación de capas de multi-perceptrones, Multi-Layer Perceptrón (MLP), para procesar cada punto de la nube de manera independiente y consigue, mediante la operación *max pooling* [21], que la red sea invariante al orden de los puntos.

Tras su éxito, la red incorporó una actualización que redefinía parte de su funcionamiento, dando lugar a la creación de PointNet++ con el artículo “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space” [22]. Esta nueva red funcionaba como la clásica, con el añadido de un funcionamiento jerarquizado para capturar características locales a múltiples escalas. Ello permitiría a la red manejar variaciones en la densidad de las nubes de puntos, mejorando el rendimiento en escenas complejas.

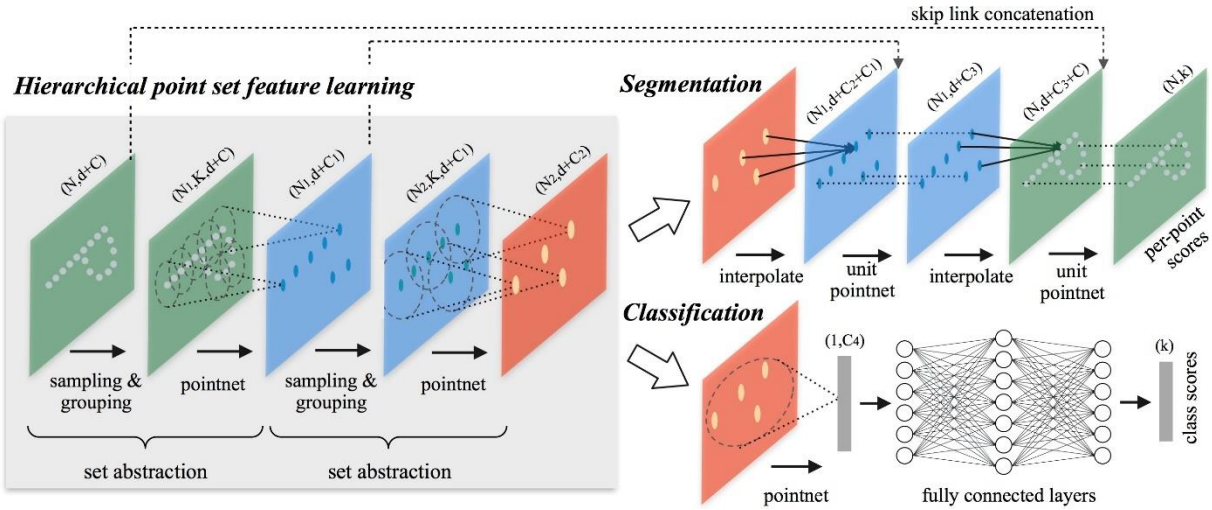

Ilustración 2.10: Estructura Jerárquica de PointNet++ [22]

Al igual que PointNet y PointNet++, se encuentran otras redes también muy conocidas con funcionamiento similar como GeomAdapt o PointConv, y otrasvoxelizan dichas nubes como FCNVoxNet que, aun teniendo un funcionamiento parecido, trabajan con vóxeles. Para ello se sigue la misma filosofía, pero en lugar de trabajar con nubes de puntos, voxelizan dichas nubes para posteriormente procesarlas.

2.3.2 Redes con arquitectura Transformer: Point Transformer

A partir del artículo escrito por investigadores de Google, “Attention is all you need” [23], no sólo hubo cambios dentro del campo del Procesamiento del Lenguaje Natural (PLN), sino que esto se extrapoló a otras tareas: análisis de imágenes, clasificación de imágenes, detección de objetos, etc. Inspirado por este éxito, Point Transformer [24] fue creado en 2020 para explotar esta nueva tecnología aplicándose a problemas de clasificación y segmentación de nubes de puntos 3D.

Recordemos que la arquitectura Transformer, mostrada gráficamente en la Ilustración 2.11, consta de varias partes:

- Codificador: esta parte se encarga de procesar la información de entrada (la nube de puntos) identificando las partes más relevantes y generando un *embedding* [25] que captura para cada punto tanto sus coordenadas 3D como características adicionales (como el color o la intensidad), consiguiendo así reflejar su importancia en relación con otros puntos de la nube.
- Decodificador: esta otra parte toma la salida del codificador y transforma los *embeddings* generados por el codificador en representaciones de salida útiles, como clasificaciones o segmentaciones de los puntos en la nube.

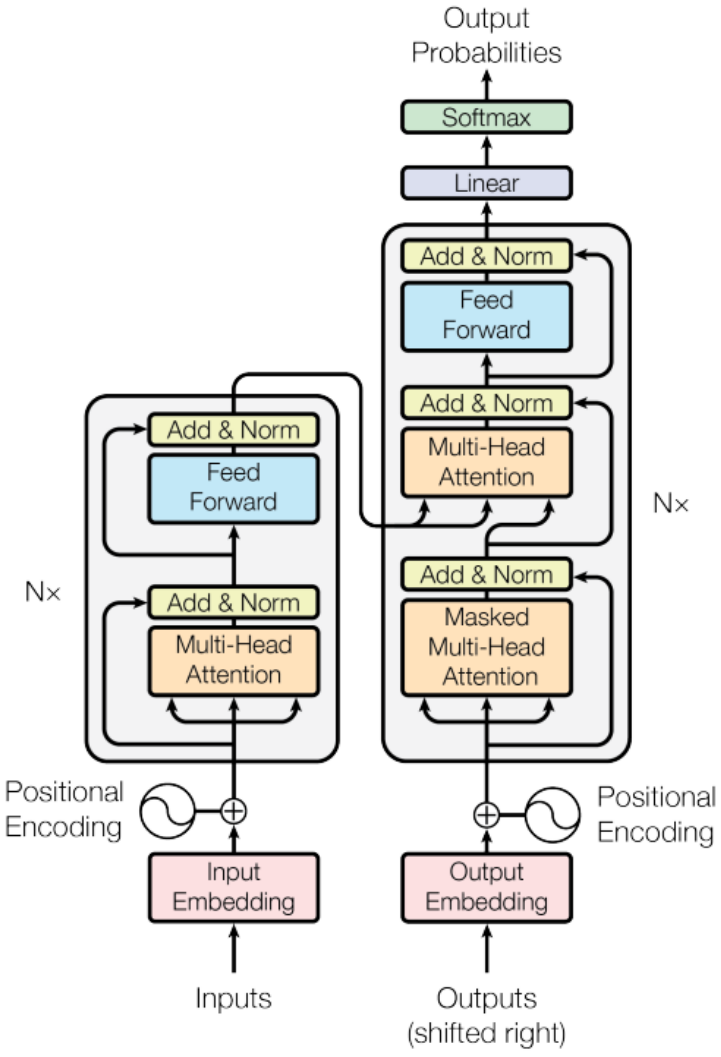

Ilustración 2.11: Arquitectura Transformer [23]

2.3.3 Redes con arquitectura de grafos: SPGraph

Esta arquitectura trabaja sobre grafos en lugar de con datos en otras formas más comunes como vectores, matrices o tensores. Aquí, el grafo se compone de nodos, siendo estos los puntos de la nube, y aristas, que los relacionan. Gracias a la propagación de información mediante este tipo de estructura, la red actualiza la representación de la nube en función de los vecinos de cada punto. Posteriormente, la red converge de manera similar a las redes convolucionales dado que se aplican capas de convolución dentro del grafo, permitiendo que las características de los nodos se fusionen de manera similar a como las CNN combinan características.

Un ejemplo de este tipo de arquitecturas se da en SPGraph, una red que trata de sobrellevar el tamaño de los escaneos LiDAR masivos y que da un paso más allá respecto del formato original de estas arquitecturas. El artículo “Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs” [26] propone una representación de Grafo de Super Puntos (SPG). Para conseguir el SPG primero ha de dividirse la nube de puntos en formas geométricas simples y significativas, lo cual se representa mediante los llamados “super puntos” tras un proceso automatizado no supervisado. Estos super puntos son los nodos de un grafo reducido y permiten el uso de métodos de *embedding* para nubes de puntos como PointNet.

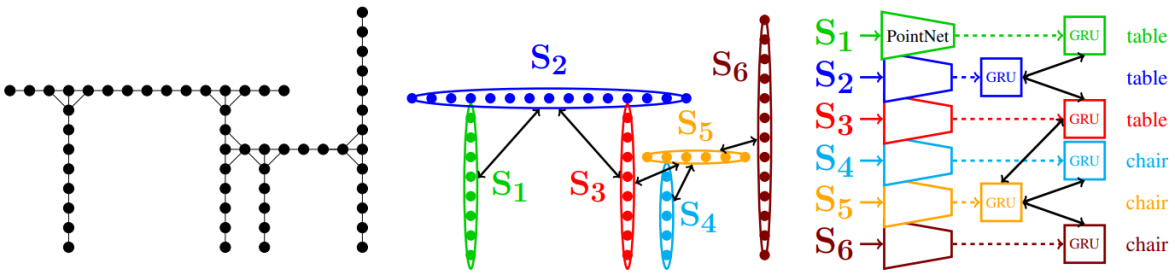

Ilustración 2.12: Grafo de superpuntos [26]

En la Ilustración 2.12 está el principal avance del artículo de SPGraph, donde se muestra cómo se pasa de un grafo que interconecta una gran cantidad de puntos a un SPG que tiene únicamente 6 nodos. Esto hace que el rendimiento mejore y los algoritmos de aprendizaje profundo basados en convoluciones de grafos puedan trabajar sobre nubes aprovechando las características de las aristas que interrelacionan los “super puntos”.

3 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”.

3.1 Requisitos iniciales

En esta sección únicamente se determinan los requisitos iniciales a alto nivel, con objetivo de fijar el aspecto del resultado buscado y utilizarlos como referencia durante la etapa de validación final. Los requisitos que se han propuesto para el proyecto se dividirán entre aquellos orientados al desarrollo y aquellos orientados a la experimentación. En cuanto a los propuestos para el desarrollo son los siguientes:

- Deben generarse ciudades de manera procedural con herramientas comerciales o algoritmos propios.
- Debe desarrollarse una aplicación para obtener fragmentos de ciudades sintéticos y etiquetados de forma procedural.
- Deben obtenerse nubes de puntos sintéticas con los fragmentos de ciudades generados a partir del sensor LiDAR virtual desarrollado en la universidad de Jaén.
- Debería hacerse un sistema de traducción para adaptar el etiquetado entre diferentes conjuntos de datos.

Relativo al estudio experimental se tienen los siguientes requisitos:

- Debe ponerse en funcionamiento en los ordenadores del laboratorio un conjunto de redes neuronales aplicadas a nubes de puntos.
- Debe llevarse a cabo un tratamiento de la información que siga el proceso KDD.

- Debe realizarse experimentación para estudiar el comportamiento de las redes para conjuntos de datos reales.
- Debe realizarse experimentación para estudiar el comportamiento de las redes para conjuntos de datos sintéticos.
- Debería realizarse un estudio del comportamiento de las redes mezclando datos reales y sintéticos.
- Debería indicarse como podrían mejorarse los resultados, así como la propuesta de alguna posibilidad para aumentar la eficacia.

3.2 Hipótesis y restricciones

3.2.1 Hipótesis

Como es común dentro de proyectos experimentales que se basan en el método científico, se presentará un listado de hipótesis iniciales:

1. **Hipótesis 1**: el comportamiento de las redes obtenido a partir de un entrenamiento y testeo con nubes de puntos sintéticas generadas con el LiDAR virtual es similar al obtenido a partir de nubes de puntos reales.
2. **Hipótesis 2**: el entrenamiento de las redes neuronales con nubes de puntos reales puede ser sustituido con un entrenamiento alimentado con datos completamente sintéticos, dando buenos resultados en el testeo con nubes de puntos reales.

3.2.2 Restricciones

Por su parte, el proyecto inicia con una serie de limitaciones que, aun estando lejos de llevar al impedimento de los objetivos, sí que condicionarán su ejecución:

1. **Restricciones temporales**: el TFM se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto sería inicialmente de 300 horas, incluyendo todas las etapas del ciclo de vida. No obstante, es importante comprender que el proyecto es un trabajo que requiere de investigación, una etapa de desarrollo, arranque de proyectos de otros

investigadores, entrenamiento de múltiples modelos de *machine learning*, experimentación, análisis de resultados, corrección de errores y en ocasiones cooperación con terceros. Esto se traducirá en que el tiempo máximo estimado subirá a 600 horas, marcando un límite temporal realista afrontable por el autor.

2. **Hardware limitado**: pese a encontrarnos en un laboratorio equipado con ordenadores especializados y tarjetas gráficas potentes, lo cierto es que para el entrenamiento de redes neuronales se requiere de mucho tiempo. Esto puede suponer varios días que, por la restricción temporal recién descrita, no siempre es factible para todos los experimentos. Añadido a esto, el coste en memoria también es muy elevado, y no es posible utilizar algunos *datasets* completos. Esto supondrá la necesidad de redimensionar el conjunto de datos para reducir el coste temporal y en memoria asociado al hardware de trabajo.
3. **Conocimientos de partida**: el proyecto se comenzó con conocimientos básicos en redes neuronales obtenidos durante las asignaturas del grado y del máster. Asimismo, tampoco se tienen grandes conocimientos en Python, aunque sí que es cierto que se tiene una base. Esto supondrá que el tiempo dedicado para la fase de investigación será superior.
4. **Calidad del fragmentador de ciudades**: debido a las limitaciones temporales y en pro del avance de la fase de experimentación, se dedicara menos tiempo del total planificado a la interfaz y usabilidad de la aplicación de fragmentación.

3.3 Riesgo del trabajo

Es muy importante destacar la gran incertidumbre de la que parte este proyecto en su conjunto, y aunque no es la única vez que se ha comentado y se comentará en esta memoria, se ha creído conveniente incluir este apartado para desarrollarlo.

No se trata de una mala gestión, ni una mala organización, sino que estamos ante un problema característico de los trabajos de investigación. El proyecto, como se irá viendo, está conformado por múltiples fases y cada una de ellas dependiente de

las demás, por lo que la calidad del conjunto del proyecto depende de la calidad individual de cada una de estas partes. Dicho esto, se debe saber que se parte de una generación de entornos procedurales cuya calidad de generación no se conoce al detalle, un LiDAR virtual que se busca validar y unas redes neuronales que, pese a haber sido probadas, no podrán utilizar la cantidad de datos necesarios para obtener resultados óptimos porque no se dispone de suficientes recursos.

Esta incertidumbre se vería minimizada si no se contara, sobre todo, con la restricción temporal destacada en los apartados anteriores. Igualmente, y completamente en relación con la restricción temporal, se podría contar con entrenamientos más extensos, así como entrenamientos hechos desde otros equipos o servidores más potentes correctamente configurados y preparados para la ejecución del proyecto, algo que como requiere de bastante tiempo no se ha podido realizar.

3.4 Estudio de alternativas y viabilidad

Durante el proyecto se han tenido que discriminar el uso de algunas herramientas a favor de otras, ya sea para la generación procedural, el etiquetado de fragmentos de ciudades, el entorno de trabajo o la elección entre los proyectos de redes neuronales disponibles. En este apartado vamos a describir las principales alternativas analizadas para su posterior elección.

3.4.1 Generación procedural

Sin duda la primera decisión del proyecto fue la elección del software para la generación procedural de ciudades. A lo largo del primer periodo de investigación se propusieron una gran cantidad de posibilidades, pero se destacarán aquellas que fueron más influyentes.

3.4.1.1 Procedural City Generator

Procedural City Generator [27] es un proyecto disponible en GitHub como software libre y gratuito que permite crear mapas que pueden descargarse como fichero PNG, mapas de alturas e incluso descargar directamente el modelo particionado en STL. El generador se basa en L-System y es capaz de generar automáticamente grandes distribuciones de calles que pueden replicar la distribución

de ciudades como las de París o Barcelona. Es un proyecto que permite algo de personalización, aunque no se obtienen resultados completamente detallados. La geometría devuelta es simple y tampoco usa texturas, siendo su punto fuerte la capacidad de generar grandes puertos carreteros introduciendo manzanas, edificios, zonas para parques, etc.

Ilustración 3.1: Ciudad creada con Procedural City Generator

Para utilizar este trabajo sería necesario hacer un *script* que unifique las salidas de este proyecto en un solo modelo, incluyendo texturas. Para dar más realismo además habría que incluir algo de ruido, personas, vehículos, mobiliario urbano, etc. Ello se podría hacer desde Blender [28], pero aun así supone la problemática de detectar la orientación de las mallas pertenecientes a aceras y carreteras.

3.4.1.2 ArcGIS CityEngine

ArcGIS CityEngine [29] es una de las herramientas de pago más avanzadas para generar ciudades procedurales en 3D. Con este programa se pueden crear grandes entornos urbanos interactivos e inmersivos y es por ello que es ampliamente usado en la creación de ciudades realistas para simulaciones, videojuegos y hasta en planificación urbana. Usa un sistema basado en reglas personalizable que permite controlar el estilo arquitectónico, la densidad de las construcciones o el trazado de las calles, entre otras muchas opciones, así como también proporciona la generación a partir de imágenes por satélite.

Para usar este software es indispensable, aunque se pueda probar de manera gratuita, comprar una licencia. Ya en posesión de una licencia se podría utilizar la aplicación durante más tiempo y crear ciudades con relativa facilidad. Es una herramienta potente con la única desventaja de que, al ser software propietario hay menos documentación publicada en Internet.

3.4.1.3 RailClone

Este *plugin* [30] para Autodesk 3ds Max permite crear ciudades con un amplio número de objetos, creando la estructura de los mismos sobre la base de los parámetros personalizables. De esta forma, se puede automatizar con reglas la creación de carreteras, puentes, rieles, edificios, cercas, muros o señales. Es muy utilizado en el ámbito de la arquitectura, la visualización de exteriores, el diseño urbano y en la creación de escenarios para películas y videojuegos.

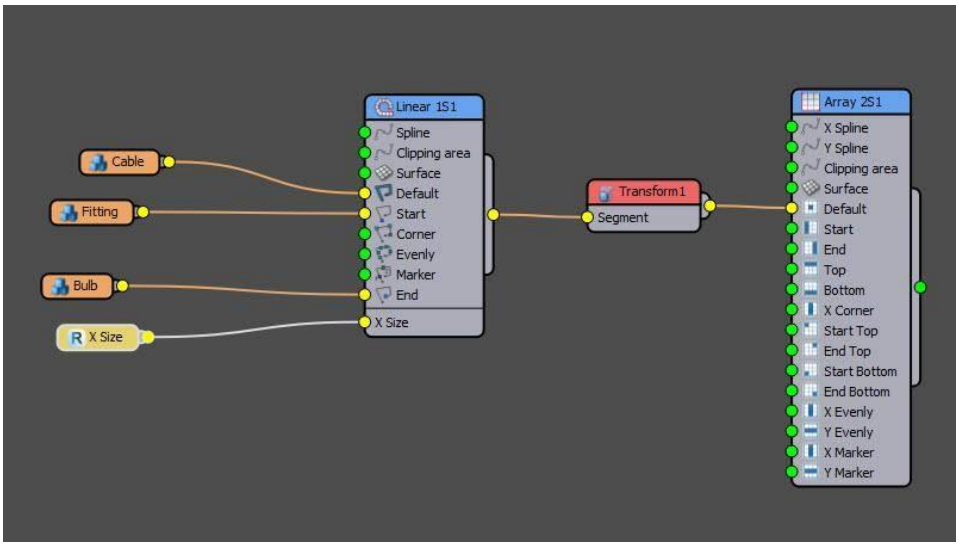

Ilustración 3.2: Sistema de nodos de RailClone

Para el uso de este *plugin* habría que comprar una licencia para usar 3ds Max, y posteriormente comprar Railclone. Este *plugin* no permite la automatización mediante *script*s, sino que usa un sistema de nodos (Ilustración 3.2) que recuerda al de otras aplicaciones como Blender.

3.4.2 Fragmentación de ciudades

Tras la obtención de ciudades existen dos opciones para la recolecta de nubes de puntos; la primera es navegar sobre las ciudades con el sensor LiDAR sintético, y

la segunda es fragmentar las ciudades. Dado que el nivel de detalle de estas ciudades es muy elevado, encontrándose decenas de millones de triángulos en los modelos, la decisión más oportuna es la de la fragmentación. Esto permitirá al LiDAR trabajar con trozos manejables a la vez que el proceso se podrá aprovechar también para etiquetar cada elemento independiente del escenario según el conjunto de etiquetas seleccionado. Por simplicidad, se tomó la decisión de utilizar un motor de gráfico ya existente para evitar trabajar a muy bajo nivel, algo que supondría mucho tiempo de desarrollo. De esta manera se tuvo que elegir entre los 2 motores más conocidos actualmente.

3.4.2.1 Unity 3D

Unity [31] ha evolucionado a lo largo de estos años siendo capaz de hacer posible el desarrollo dentro de él de videojuegos con grandes estándares de calidad, grandes campañas de marketing y publicados por grandes estudios, lo que se conoce como juegos de triple A. El trabajo flexible, el amplio conjunto de herramientas, la curva de aprendizaje accesible y la gran cantidad de documentación en Internet ha incentivado a que sea uno de los motores de videojuegos más populares actualmente.

Es una gran opción para nuestro proyecto porque permite importar y modificar de forma automática los modelos. Desde Unity se podría crear una aplicación que facilitase la fragmentación automática de las ciudades creadas anteriormente con *script*s en C#, así como también se podría manejar el proceso de etiquetado. Esta opción es gratuita usando la versión de Unity Personal, versión en la que los desarrolladores pueden crear aplicaciones sin pago alguno mientras que las ganancias no superen los 100,000 dólares anuales. Como el proyecto no plantea vender la aplicación desarrollada no debería haber problemas en lo referente a la versión gratuita, de la misma manera que tampoco debería haber problemas relacionados con el sistema operativo utilizado en el ordenador del laboratorio, pues Unity es un motor multiplataforma.

3.4.2.2 Unreal Engine

Unreal Engine [32], desarrollado por Epic Games [33], es uno de los motores de videojuegos multiplataforma más potentes de la actualidad. Desde su creación en 1998, ha evolucionado hasta su versión 5, que ofrece impresionantes resultados

gráficos y mejoras significativas para la creación de mundos abiertos en comparación con su predecesor, Unreal Engine 4. Esta última versión destaca especialmente en el desarrollo de videojuegos 3D, ya que permite trabajar en tiempo real con escenas de mallas muy detalladas y aporta innovadoras mejoras en el sistema de iluminación.

El trabajo en Unreal Engine 5 se realiza principalmente en el lenguaje de programación C++. Sin embargo, también ofrece la alternativa de Blueprints, una herramienta visual que facilita la creación de código mediante nodos y conexiones, ideal para usuarios con conocimientos básicos de programación. Al igual que Unity, Unreal Engine 5 sería adecuado para crear la aplicación para fragmentación y etiquetado de ciudades, proporcionando un rendimiento y velocidad de ejecución mejorados a cambio de un desarrollo algo menos acelerado.

Cabe mencionar que, a pesar de contar con una comunidad sólida, aprender a utilizar Unreal Engine suele tener una curva de aprendizaje más pronunciada en comparación con Unity. A ello hay que sumarle las exigencias de recursos, que son mucho más elevadas.

3.4.3 Redes neuronales

3.4.3.1 Proyecto Super Point Transformer

Este proyecto [34] presenta una red con una arquitectura basada en Transformers de superpuntos y es funcional para la segmentación de nubes de puntos en escenas 3D. La metodología utiliza un algoritmo para dividir las nubes de puntos en una estructura jerárquica de superpuntos, lo que hace que el preprocesamiento sea mucho más rápido. Todo ello viene explicado en detalle dentro de su artículo, “Efficient 3D Semantic Segmentation with SuperPoint Transformer” [35].

Es un proyecto escrito en Python y funcional desde sistemas operativos Linux y con la versión de CUDA 11.8 o 12.1. Sus creadores utilizaron equipos con 64GB de RAM (Random Access Memory), tarjetas gráficas NVIDIA GTX 1080Ti 11G, NVIDIA V100 32GB y NVIDIA A40 48Gb, y con ellos consiguieron experimentar con conjuntos tales como S3DIS [36], Semantic KITTI y DALES [37].

3.4.3.2 Proyecto PointNet2 Semantic

Este proyecto [38] es una versión modificada de PointNet++ enfocada en la segmentación semántica para comparar tres conjuntos de datos: Scannet [39], Semantic-8 [40] y un conjunto de datos aéreo LIDAR de Bertrand Le Saux. Esta investigación tenía también marcado como objetivo comparar esos conjuntos de datos con SnapNet [41] (otro proyecto destacado de segmentación semántica).

Este se ejecuta en Ubuntu 16.04 y fue probado con 3 GTX Titan Black y una GTX Titan X, así como también en otras tarjetas gráficas como la GTX 860m. Entre otras instalaciones necesarias, se destaca Python 2.7, CUDA 8.0 y TensorFlow 1.2. En GitHub se pueden encontrar enlaces para descargar los datos preprocesados y también se ofrecen instrucciones para procesar datos en bruto y para llevar a cabo el entrenamiento de los modelos.

Al igual que el anterior proyecto, es un trabajo es muy interesante, sin embargo, durante la ejecución del *script* de entrenamiento de la red surgía un problema relacionado con *daemons* que no permitía terminar el proceso.

3.4.3.3 Proyecto RepSurf

Este otro proyecto es sin duda el más interesante y es por ello que se dedicó más tiempo para intentar arrancarlo. *Surface Representation for Point Clouds* [42], resumido como RepSurf, se trata de un trabajo de investigación que propone una nueva forma de representar nubes de puntos que permite capturar explícitamente la estructura geométrica local detallada. Explora 2 variantes, pero a lo largo de toda la investigación siempre nos referiremos a la variante Umbrella RepSurf.

Este proyecto disponible desde GitHub incorpora la red de PointNet++, la red Point Transformer y el módulo propio de Umbrella RepSurf basado en una versión ligera de PointNet++. Presentado en su artículo, “Surface Representation for Point Clouds” [43], pero visible también desde el GitHub están algunos de sus resultados con *datasets* como S3DIS.

Los desarrolladores iniciaron el proyecto en Linux utilizando Python 3.7, PyTorch 1.6.0, CUDA 10.1 y GCC 7.2.0 entre otras versiones destacables. En el caso de este trabajo, se consiguió arrancar utilizando versiones similares dadas las diferencias en hardware.

3.4.4 Entorno de trabajo

En lo referente al sistema operativo, lo cierto es que no había demasiada posibilidad de elección dentro del laboratorio, pues todos los equipos usaban Windows. Sin embargo, la mayoría de proyectos de redes neuronales que se encontraban en GitHub estaban preparados para ejecutarse sobre diferentes entornos de Linux. Una posible solución podría haber sido instalar mediante una partición del disco la versión necesitada de Linux, sin embargo, esto era demasiado costoso teniendo en cuenta que dicha versión cambiaba entre proyectos. Es por ello que surgieron varias posibilidades: utilizar máquina virtual o usar el subsistema de Windows para Linux.

3.4.4.1 Máquina virtual con VirtualBox

La primera opción, la más conocida para la mayoría, es la de crear una máquina virtual para simular un equipo con la versión de Linux requerida. Una máquina virtual es un entorno virtualizado que actúa como un sistema informático independiente dentro de un mismo ordenador en el que se puede instalar y ejecutar de forma aislada como un sistema operativo junto con sus programas, como si fuera un segundo ordenador. Es frecuente el uso de estas para probar software, realizar experimentos y ejecutar aplicaciones en otros sistemas operativos sin comprometer al sistema anfitrión.

Todo ello se puede hacer con VirtualBox [44], un software de virtualización gratuito y de código abierto desarrollado por Oracle que se ha estudiado a lo largo de la carrera. Se puede ejecutar múltiples sistemas operativos (como Windows, Linux o macOS) en un ordenador host sin necesidad de alterar su sistema operativo principal. Sin embargo, esto supone por un lado dividir los recursos del equipo servidor y por otro incrementar el uso de estos para la superior gestión de procesos. Con esta filosofía de uso de los recursos, es seguro que habrá una bajada importante de

rendimiento a la que hay que sumarle la correcta configuración de la tarjeta gráfica con CUDA. En efecto, estos proyectos usan CUDA para poder acceder a los recursos de la tarjeta gráfica, por lo que debe antes asociarse a VirtualBox. Sin embargo, esta funcionalidad es de pago, por lo que se requeriría antes de pagar la extensión.

3.4.4.2 Subsistema de Windows para Linux

Esta opción, el subsistema de Windows para Linux (WSL), es una funcionalidad en Windows que permite ejecutar distribuciones de Linux directamente sobre el sistema operativo Windows. Para ello, WSL utiliza tecnología de virtualización ligera para ejecutar un kernel de Linux completo dentro de contenedores aislados en una máquina virtual administrada de forma automática. Ello proporciona un rendimiento de sistema de archivos mejorado y una compatibilidad completa con llamadas de sistema de Linux, lo que permite una experiencia Linux más auténtica dentro de Windows.

Con esta solución, no se requeriría de aplicaciones intermedias para la virtualización, sino que sería más directo desde la consola. Además, al estar integrado podría accederse desde la interfaz del gestor de archivos de Windows a los datos de la virtualización de Linux. Es una opción ideal y para crear una máquina virtual solo se requiere, tras la instalación de la distribución, del comando: *"wsl -d Ubuntu-20.04"*.

3.5 Descripción de la solución propuesta

En un inicio se decidió intentar utilizar el proyecto Procedural City Generator como base para un algoritmo propio de generación de ciudades. No obstante, durante un pequeño periodo de desarrollo se llegó a la conclusión de que iba a demorar demasiado tiempo construir el algoritmo y dejaría a la etapa de experimentación sin tiempo suficiente. Es por ello que, pese a ser gratuito fue la primera opción en descartarse. Tras eliminar esta opción, las siguientes alternativas más interesantes disponibles eran de pago, por lo que se revisó si la universidad poseía alguna licencia de dichos software. Una vez se solicitó esa información se conoció que en efecto la universidad poseía licencias de uno de los software, CityEngine, por lo que se determinó finalmente esa opción como la más apropiada.

Teniendo en cuenta que el objetivo de este trabajo es experimental, se decidió que la etapa de desarrollo de la aplicación auxiliar se hiciera con Unity. De darse el caso de seleccionar Unreal Engine, sería necesario programar la aplicación con C++ o Blueprints dentro de un motor sin experiencia previa por parte del autor, lo que podría convertirse potencialmente en un problema temporal que repercutiese en los resultados experimentales del trabajo. De esta manera, Unity se convirtió en la opción más conservadora, pues se tenía mucha experiencia en el motor, lenguaje y, aunque es posible que Unreal Engine sea capaz de gestionar mejor grandes cantidades de geometría, la experiencia nos dice que Unity también es capaz de gestionar los modelos que se utilizarán dentro de este trabajo.

En cuanto al proyecto para el uso de redes neuronales, con muchas dificultades se inició el proyecto RepSurf que, frente al resto de las opciones, era el más interesante. Aunque es cierto que necesitaba algunas modificaciones, es importante mencionar que es el que más redes permite ejecutar sin cambiar de proyecto, por lo que a la larga va a ahorrar mucho tiempo.

Finalmente, relativo al entorno de trabajo, se decidió usar el subsistema de Windows para Linux. Es la opción más razonable dado que es muy fácil de arrancar, es gratuito y también más eficiente, algo muy importante en procesos que pueden demorarse días si no se usan correctamente los recursos.

3.6 Alcance

El proyecto incluirá los siguientes entregables:

1. **Fragmentador de ciudades**: será un proyecto de Unity 3D creado en la versión de Unity 2023.2.12f1 que se compone de una única escena preparada para que con un mínimo de modificaciones se pueda cargar y fragmentar una ciudad. El proyecto incluye además el archivo git donde se puede comprobar las actualizaciones que se han ido implementando. Cabe destacar además que todo el código propio viene explicado al detalle con comentarios, y con archivos README.

2. **Proyecto de redes neuronales adaptado**: se trata del proyecto RepSurf modificado para poder utilizar cualquier tipo de *dataset*. Los cambios y los nuevos *script*s, así como las adaptaciones o los elementos a tener en cuenta de cualquier tipo vienen especificados igualmente con comentarios dentro del código y con archivos README.
3. ***Dataset* sintético**: es un conjunto de nubes de puntos obtenidas del sensor LiDAR virtual tras ubicarlo en múltiples ciudades virtuales generadas con reglas distintas. Estas nubes de puntos son las que se han utilizado para la fase de experimentación, encontrándose en formato PLY y conteniendo un etiquetado personalizado.
4. **Fragmentos sintéticos**: será un conjunto de modelos 3D guardados en formato FBX que contendrá los fragmentos de ciudades generados a lo largo del proyecto.
5. **Registros**: incluirá un conjunto de archivos de registro que muestran la información mostrada por la terminal a lo largo de cada uno de los experimentos. Habrá registro tanto de la actividad del entrenamiento como de la de testeo.
6. **Modelos**: guardará los archivos CKPT (*checkpoints* del modelo) generados y permitirá ejecutar las pruebas de evaluación sin la necesidad de realizar nuevamente los entrenamientos.
7. **Documentación**: se trata del documento actual y de numerosos archivos README incluidos junto a todos los *scripts* propios tal y como se ha adelantado en puntos precedentes. Como es de esperar, ello sirve para explicar en detalle los puntos más importantes del proyecto, así como para para hacerlo más usable y ayudar a los próximos investigadores que continúen este proyecto.
8. **Vídeo**: mostrará brevemente el funcionamiento del proyecto, en su conjunto, para poder ver el comportamiento del mismo sin la necesidad de ejecutar nada. Esto ahorra mucho tiempo para aquellos que solo deseen consultar los resultados finales.

3.7 Tecnologías utilizadas

Sintetizando, se procede a enumerar en los siguientes subapartados todas las tecnologías y herramientas utilizadas en este proyecto junto con una breve descripción.

3.7.1 Software para gráficos

- Unity: es el motor gráfico escogido para crear la aplicación que fragmenta y etiqueta las ciudades generadas proceduralmente.
- Blender: software especializado en modelado 3D y animación que se utiliza para visualizar archivos en formato FBX y OBJ. También se utilizó durante el periodo de investigación y pruebas para el intento de creación del algoritmo procedural de ciudades propio.
- CityEngine: herramienta que permite crear fácilmente ciudades de forma procedural partiendo de un conjunto de reglas personalizable.
- MeshLab: visor principal utilizado para mostrar las nubes de puntos.

3.7.2 Programación y lenguajes

- Visual Studio 2022: se usa como entorno de desarrollo integrado (IDE) para escribir y depurar código, principalmente en C# y Python.
- Python: lenguaje principal en el que se encuentra programado el proyecto RepSurf y el *script* de procesado de nubes de puntos.
- PyTorch [45]: biblioteca utilizada en el proyecto RepSurf para construir y entrenar los modelos de aprendizaje profundo.
- C#: lenguaje de programación utilizado en el desarrollo de la aplicación para fragmentar y etiquetado de ciudades hecha en Unity.

3.7.3 Entornos virtuales

- Anaconda: se utiliza en el proyecto para gestionar entornos de desarrollo y dependencias de Python, facilitando la instalación de librerías y la ejecución de código.

- Subsistema de Windows para Linux (WSL): se utiliza para ejecutar distribuciones Linux sin necesidad de máquinas virtuales intermedias, y hacerlo en su lugar desde la propia máquina de Windows. Es imprescindible para iniciar los proyectos de redes neuronales.

3.7.4 Documentación

- Word: editor de texto utilizado para la redacción esta memoria.
- Excel: es la aplicación con la que se organizaban los resultados de los experimentos en forma de tablas y gráficos. Sus hojas de cálculo fueron usadas también para la creación y modificación de las tablas de etiquetas de la aplicación fragmentadora.
- Visual Paradigm: esta herramienta se empleó en la creación de ciertos diagramas del proyecto y también ofrece una versión en línea que facilita su acceso.
- Umlet: una aplicación para creación y edición de diagramas. La razón principal de su uso es el diseño obtenido a partir de simples entradas y por su precio gratuito.
- Google Keep: es una herramienta para tomar notas. Fue usada para apuntar ideas a lo largo del proyecto.
- Mendeley Cite: extensión para Microsoft Word que facilita la gestión de la bibliografía.
- Google Drawings: es una herramienta Google para hacer dibujos online con posibilidad de exportarlos en formato PNG. Es utilizada para hacer algunos diagramas propios de esta memoria.

3.8 Metodología

La elección de una correcta metodología para llevar a cabo el ciclo de vida del proyecto o PMLC (Project Management Life Cycle) es uno de los pasos más importantes durante la dirección de un proyecto. Toda metodología debe de tener en cuenta la complejidad y la incertidumbre del proyecto, y es por ello que a nivel teórico se distinguen las siguientes 3 principales vertientes:

- TPM (Traditional Project Management): utilizada para proyectos poco novedosos y con objetivos claros donde no se espera un grado alto de complejidad.
- APM (Agile Project Management): utilizada para proyectos donde teniendo claro los objetivos, no es seguro la manera de alcanzarlos.
- XPM (Extreme Project Management): utilizada para proyectos altamente complejos donde no se tienen completamente claros los objetivos ni las soluciones.

Por otro lado, desde el punto de vista del ciclo de vida del desarrollo de software encontramos la necesidad de seleccionar la categoría SDLC (Software Development Life Cycle) a usar, que dependiendo del conocimiento técnico requerido en el alcance del proyecto se moverá desde un caso de categoría lineal (total certeza) hasta la categoría extrema (total incertidumbre).

Teniendo ahora en cuenta estas bases, estamos preparados para seleccionar una metodología correcta para nuestro proyecto experimental. Dado que este tipo de proyectos son de investigación y novedosos, es razonable pensar que, aunque hay algunos objetivos claros, las soluciones no lo son tanto. Esto se traduce en un enfoque principal para el proyecto ubicado en la vertiente AMP con aproximación a la vertiente XPM. De igual manera, vista la incertidumbre del proyecto de cara al desarrollo debemos plantear una categoría SDLC que nos permita cambiar de rumbo del proyecto a bajo coste, lo que nos sugiere hacer uso de metodologías ágiles frente a metodologías tradicionales.

3.8.1 Metodología general del ciclo de vida del proyecto

La metodología general del proyecto será Project Milestones, una metodología basada en hitos. Los hitos marcan puntos de referencia en el cronograma del proyecto con los que se identifica cuando una o varias actividades han sido concluidas para así iniciar otra nueva. Estas marcas de comprobación son muy útiles para supervisar los plazos, identificar fechas importantes y reconocer potenciales cuellos de botella de un proyecto.

Aplicar esta metodología dentro del proyecto actual es una buena opción para mantener un correcto ritmo y evitar retrasos con las fechas y entregables. Dado que la metodología permite incluir nuevos hitos durante el proyecto, ésta se adapta sin problemas a la situación de incertidumbre de la que se parte, siendo fácil realizar modificaciones y hacer cambios de requisitos de manera acertada. En la Ilustración 3.3, se plantea un diagrama de partida en el que se ven los 8 hitos de los que inicialmente se conforma este proyecto experimental.

Ilustración 3.3: Hitos siguiendo metodología Project Milestones

Es importante mencionar que el proceso de experimentación seguirá el esquema definido en el KDD, teniendo en cuenta: la recolección de los datos, el preprocesado, la transformación, la minería, la evaluación y la interpretación de los resultados.

3.8.2 Metodología dentro del desarrollo

Cumplir con todas las especificaciones es complicado teniendo en cuenta la gran restricción temporal de la que se parte. Esa restricción aumenta mucho la incertidumbre y provoca que las cosas que no se hagan sean principalmente por falta de tiempo. Si se dispusiese de tiempo ilimitado, se podría aplicar una técnica tradicional con la que se podrían llegar a cubrir todas las especificaciones.

Como no es el caso, ha de trabajarse con una metodología ágil para el desarrollo, eligiendo entre las posibles la conocida como Extreme Programming. Esta es muy popular y se centra en la velocidad y la simplicidad. La ideología de este modelo se basa en hacer ciclos de desarrollo muy cortos (*sprints*) donde se ejecutan pequeñas partes del proyecto alcanzando una gran calidad al haber sido debidamente probados. A diferencia de otras metodologías, la programación extrema es muy disciplinada pues, aunque la documentación ciertamente queda más reducida, se realizan con frecuencia revisiones y pruebas de código para realizar cambios rápidamente.

Este método es una buena opción porque potencia la creatividad durante todas las etapas de desarrollo y en cada uno de sus pequeños *sprints*. Añadido a esto, al tratarse de una metodología ágil cuyas bases se fundamentan en el uso de *sprints* de muy poca duración y simples, se asegura por un lado que se puede cambiar de rumbo de proyecto a bajo coste y por otro lado se asegura la simplicidad de cada *sprint*, pues el problema se ve obligado a desglosarse en subproblemas más pequeños y fáciles de resolver y depurar.

3.9 Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFM, no existen restricciones de tipo económico, sino de tipo temporal establecido inicialmente en 600 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 (autor del trabajo).

Bajo esta base, la estimación del trabajo necesario para completar el proyecto se hará mediante un desglose en paquetes de trabajo. Cada paquete de trabajo busca cumplir una serie de objetivos y dicha relación se representa en la Estructura de Desglose de Trabajo (EDT) de la Ilustración 3.4. La EDT está organizada en 4 niveles, siendo el primer nivel el más general abarcando todo el proyecto. El segundo nivel muestra los objetivos del trabajo (definidos en 1 Objetivos del proyecto) y se asocia con el nivel 3, el cual contiene los 4 paquetes de trabajo que tendrá el trabajo, mientras

que en su último nivel están las tareas de los paquetes de trabajo, que más adelante se detallarán.

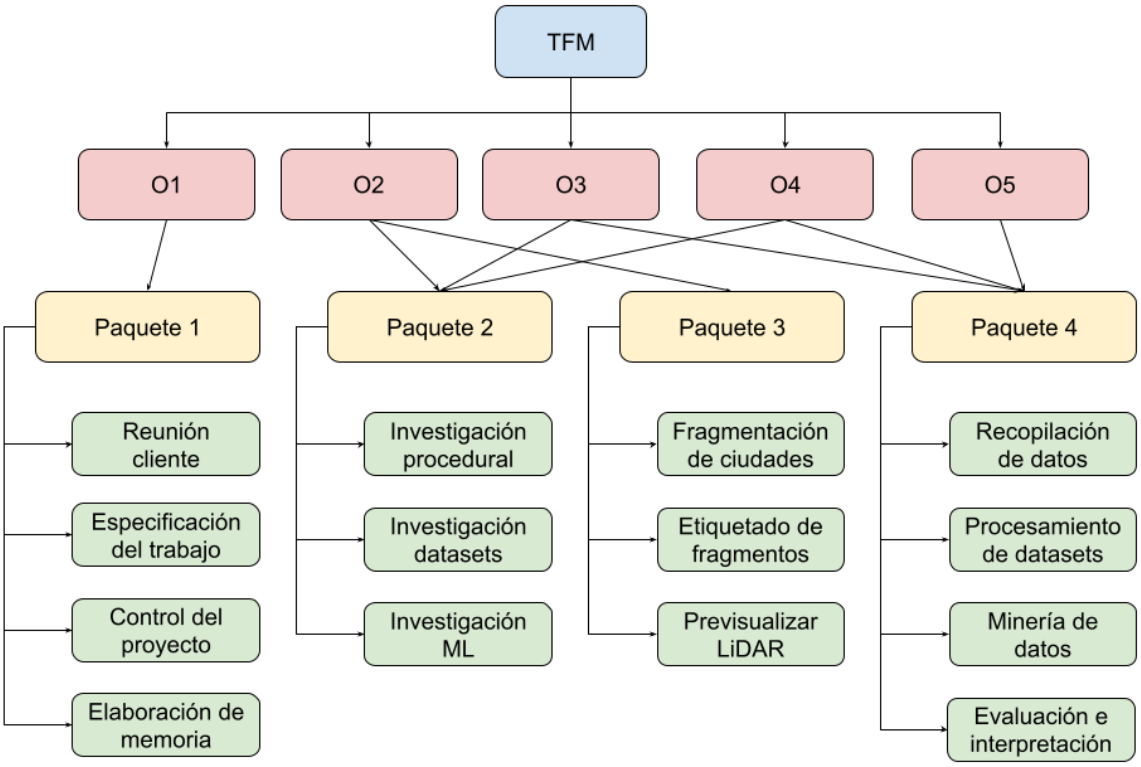

Ilustración 3.4: EDT del proyecto

A continuación, se mostrarán las tablas de tareas contenidas en los paquetes de la EDT. Dentro de cada una de ellas se incluirá una descripción e información de interés, tal como lo es el periodo de tiempo esperado para llevarse a cabo, las subtareas que la conforman, los miembros implicados, hitos que consiguen, entregables, etc.

3.9.1 Paquete de trabajo 1: Gestión y coordinación

La importancia de la correcta gestión radica en la buena coordinación para alcanzar el objetivo 1 del proyecto, donde se hablaba de control y coordinación para asegurar un correcto avance del proyecto durante todo el ciclo de vida del mismo. Este paquete se centra en garantizar la adecuada estructuración del proyecto de acuerdo a la metodología propuesta anteriormente. Está compuesto por las siguientes tareas, desarrolladas a continuación en sus tablas correspondientes:

- Tarea 1.1: Reunión con el cliente
- Tarea 1.2: Especificación del trabajo
- Tarea 1.3: Control del proyecto
- Tarea 1.4: Elaboración de la memoria

Tarea 1.1: Reunión con el cliente	
Descripción	Tarea que busca realizar una serie de tutorías iniciales para la definición del proyecto a nivel general, planteando los objetivos iniciales
Miembros	Víctor Rodríguez Cano, Alfonso López Ruiz, Rafael Jesús Segura Sánchez y Carlos Javier Ogayar Anguita
Duración	1 semana
Objetivos	Objetivo 1 (O1)
Subtareas	No dispone
Hitos y entregables	No dispone

Tabla 3.1: Tarea 1.1

Tarea 1.2: Especificación del trabajo	
Descripción	Tarea que busca llevar las ideas planteadas de manera general a un planteamiento específico para comenzar el proyecto experimental. Aquí se dejarán claros los requisitos, alcance, metodología y una estimación del esfuerzo junto con el presupuesto
Miembros	Víctor Rodríguez Cano
Duración	1 semana
Objetivos	Objetivo 1 (O1)
Subtareas	1. Definición de requisitos y alcance 2. Definición de metodología 3. Estimación del esfuerzo y presupuesto
Hitos y entregables	Hito 1 (H1): especificación del proyecto Entrega 1 (E1): documentación con la información específica del proyecto planteado

Tabla 3.2: Tarea 1.2

Tarea 1.3: Control del proyecto	
Descripción	Tarea que dura todo el periodo de vida del proyecto sin incluir la memoria del mismo. En ella se busca que la investigación se vaya llevando de forma correcta, siguiendo plazos, entregando informes cuando y coordinando las reuniones con personal implicado cuando se requiera
Miembros	Víctor Rodríguez Cano
Duración	6 meses
Objetivos	Objetivo 1 (O1)
Subtareas	1. Control y monitorización del trabajo 2. Organización de reuniones de seguimiento 3. Generación periódica de informes 4. Coordinación del proyecto con Alfonso
Hitos y entregables	No dispone

Tabla 3.3: Tarea 1.3

Tarea 1.4: Elaboración de la memoria	
Descripción	Tarea que tiene como objetivo finalizar con la entrega de una memoria que resuma todo el proceso llevado a cabo
Miembros	Víctor Rodríguez Cano
Duración	3 meses
Objetivos	Objetivo 1 (O1)
Subtareas	No dispone
Hitos y entregables	Hito 8 (H8): cierre del proyecto y conclusiones Entrega 16 (E16): memoria del proyecto finalizada

Tabla 3.4: Tarea 1.4

3.9.2 Paquete de trabajo 2: Investigación

Este paquete contiene aquellas actividades asociadas a las etapas de investigación requeridas en el proyecto, así como la prueba de herramientas y de proyectos. Trata de contribuir para el alcance de los objetivos 2, 3 y 4 del proyecto. Está compuesto por las siguientes tareas, desarrolladas a continuación en sus tablas correspondientes:

- Tarea 2.1: Investigación sobre generación procedural
- Tarea 2.2: Investigación sobre *datasets*
- Tarea 2.3: Investigación sobre Machine Learning (ML)

Tarea 2.1: Investigación sobre generación procedural	
Descripción	Tarea destinada a la investigación del estado del arte sobre técnicas de generación procedural, así como análisis de diferentes alternativas comerciales para generar entornos urbanos. Dentro de esta tarea se tiene en cuenta tanto la investigación como el periodo de pruebas y arranque de las diferentes aplicaciones
Miembros	Víctor Rodríguez Cano
Duración	2 semanas
Objetivos	Objetivo 2 (O2)
Subtareas	1. Estudio del estado del arte generación procedural 2. Prueba e instalación de herramientas comerciales
Hitos y entregables	Hito 2 (H2): generador procedural de ciudades Entregable 2 (E2): informe sobre técnicas y herramientas de generación con elección de herramienta a utilizar

Tabla 3.5: Tarea 2.1

Tarea 2.2: Investigación sobre *datasets*	
Descripción	Tarea orientada a la revisión de los conjuntos de nubes de puntos LiDAR públicos recopilados de las investigaciones más influyentes dentro del sector. En esta tarea se tiene en cuenta tanto la investigación como la descarga y la prueba del *dataset*
Miembros	Víctor Rodríguez Cano
Duración	2 semanas
Objetivos	Objetivo 3 (O3)
Subtareas	1. Estudio del estado del arte sobre *datasets* LiDAR 2. Descarga y prueba de *datasets*
Hitos y entregables	Entregable 3 (E3): informe sobre *datasets* LiDAR públicos

Tabla 3.6: Tarea 2.2

Tarea 2.3: Investigación sobre Machine Learning	
Descripción	Tarea cuya meta es obtener una idea clara de los proyectos de ML que trabajan con redes aplicadas a nubes de puntos LiDAR. Durante esta tarea no sólo se revisarán las posibles alternativas, sino que además se intentará arrancar algún proyecto
Miembros	Víctor Rodríguez Cano
Duración	1 mes
Objetivos	Objetivo 4 (O4)
Subtareas	1. Estudio del estado del arte en ML 2. Arranque de proyecto ML para LiDAR
Hitos y entregables	Hito 5 (H5): puesta en marcha de proyecto de redes neuronales Entregable 9 (E9): informe sobre proyectos de Machine Learning funcionales

Tabla 3.7: Tarea 2.3

3.9.3 Paquete de trabajo 3: Desarrollo

Este paquete contiene las actividades asociadas a la etapa de desarrollo de la aplicación fragmentadora de ciudades. Dicha aplicación debe fragmentar ciudades y etiquetar en función de diferentes conjuntos de etiquetas. Trata de contribuir en el alcance del objetivo 2 del proyecto y seguirá el capítulo 3.8.2 (Metodología dentro del desarrollo). Está compuesto por las siguientes tareas, desarrolladas a continuación en sus tablas correspondientes:

- Tarea 3.1: Fragmentación de ciudades
- Tarea 3.2: Etiquetado de fragmentos
- Tarea 3.3: Previsualizador LiDAR

Tarea 3.1: Fragmentación de ciudades	
Descripción	Esta tarea tiene como objetivo iniciar el desarrollo de una aplicación en Unity que permita fragmentar ciudades. Tras la realización de esta tarea debe obtenerse un prototipo capaz de fragmentar ciudades, sin que sea necesario su etiquetado
Miembros	Víctor Rodríguez Cano

Duración	2 semanas
Objetivos	Objetivo 3 (O3)
Subtareas	1. Sistema de instanciación pseudoaleatorizada 2. Desarrollo del objeto fragmentador
Hitos y entregables	Entregable 5 (E5): prototipo de la aplicación con capacidad de fragmentar ciudades

Tabla 3.8: Tarea 3.1

Tarea 3.2: Etiquetado de fragmentos	
Descripción	Esta tarea trata de implementar la función de etiquetado y conversión entre etiquetas para la aplicación en Unity. Además, tras esta tarea la aplicación tiene que devolver archivos en XML que puedan ser leídos por el sensor LiDAR virtual
Miembros	Víctor Rodríguez Cano
Duración	2 semanas
Objetivos	Objetivo 3 (O3)
Subtareas	1. Gestión y conversión del etiquetado 2. Generación de archivos XML para LiDAR virtual
Hitos y entregables	Entregable 6 (E6): prototipo de la aplicación con capacidad de fragmentar y etiquetar ciudades

Tabla 3.9: Tarea 3.2

Tarea 3.3: Previsualizador LiDAR	
Descripción	Esta tarea busca mejorar la aplicación de Unity para fragmentar y etiquetar ciudades implementando un previsualizador. Con este se trata de dar una idea general y aproximada de cómo será la nube de puntos que se conseguirá con el LiDAR
Miembros	Víctor Rodríguez Cano
Duración	1 semana
Objetivos	Objetivo 3 (O3)
Subtareas	No dispone
Hitos y entregables	Entregable 7 (E7): aplicación terminada con capacidad de fragmentar y etiquetar ciudades, con capacidad de devolver una previsualización de la posible nube de puntos LiDAR

Tabla 3.10: Tarea 3.3

3.9.4 Paquete de trabajo 4: Experimentación

El último paquete contiene las actividades asociadas a la etapa de experimentación de esta investigación. Siguiéndose los pasos del proceso KDD se responderán a los objetivos 3, 4 y 5. Está compuesto por las siguientes tareas, desarrolladas a continuación en sus tablas correspondientes:

- Tarea 4.1: Recopilación de datos
- Tarea 4.2: Procesamiento de *datasets*
- Tarea 4.3: Minería de datos
- Tarea 4.4: Evaluación e interpretación

Tarea 4.1: Recopilación de datos	
Descripción	Esta tarea será la primera dentro del KDD y en ella se deben recopilar todos los datos que serán utilizados dentro del sistema. Esto incluye la obtención de datos reales y la obtención de datos sintéticos, los cuales deben generarse con las herramientas disponibles
Miembros	Víctor Rodríguez Cano y Alfonso López Ruiz
Duración	2 meses
Objetivos	Objetivo 3 (O3)
Subtareas	1. Creación de ciudades en CityEngine 2. Fragmentado y etiquetado de ciudades 3. Obtención de nubes sintéticas con LiDAR virtual 4. Obtención de nubes reales de Internet
Hitos y entregables	Hito 3 (H3): fragmentos etiquetados de ciudades Hito 4 (H4): *dataset* de nubes de puntos sintéticas Entregable 4 (E4): conjunto de ciudades completas sin etiquetar Entregable 8 (E8): conjunto de fragmentos de ciudades etiquetadas Entregable 10 (E10): *dataset* de nubes de puntos sintéticas

Tabla 3.11: Tarea 4.1

Tarea 4.2: Procesamiento de *datasets*	
Descripción	Esta tarea, la segunda del KDD, sirve para preparar los datos a la arquitectura, al contexto del problema y a las limitaciones del LiDAR virtual
Miembros	Víctor Rodríguez Cano
Duración	2 semanas
Objetivos	Objetivo 3 (O3)
Subtareas	1. Adaptación a la arquitectura 2. Limpieza de datos 3. Reducción de la dimensionalidad
Hitos y entregables	Entregable 11 (E11): *script* de procesado, transformación y limpieza de datos Entregable 12 (E12): *datasets* procesados y preparados

Tabla 3.12: Tarea 4.2

Tarea 4.3: Minería de datos	
Descripción	Esta tarea, la tercera del KDD, tiene como finalidad entrenar los modelos de redes neuronales con los datos preparados, ya sean sintéticos, reales o mixtos
Miembros	Víctor Rodríguez Cano
Duración	3 semanas
Objetivos	Objetivo 4 (O4)
Subtareas	1. Adaptación del proyecto a investigación 2. Parametrización y configuración
Hitos y entregables	Hito 6 (H6): entrenamiento de las redes neuronales Entregable 13 (E13): modelos entrenados con datos sintéticos, reales y mixtos

Tabla 3.13: Tarea 4.3

Tarea 4.4: Evaluación e interpretación	
Descripción	Esta tarea está orientada a la prueba de los modelos con datos reales o sintéticos. Para ello se ha planteado un conjunto de 3 experimentos, uno para tomar una referencia y el resto para intentar validar las hipótesis del proyecto

Miembros	Víctor Rodríguez Cano
Duración	2 semanas
Objetivos	Objetivo 5 (O5)
Subtareas	1. Experimento 1 (base) 2. Experimento 2 (hipótesis 1) 3. Experimento 3 (hipótesis 2)
Hitos y entregables	Hito 7 (H7): evaluación de los modelos Entregable 14 (E14): resultados de los entrenamientos Entregable 15 (E15): informe del análisis de los resultados

Tabla 3.14: Tarea 4.4

3.10 Planificación temporal

Dado que el proyecto comenzó con su planteamiento en el mes de enero y se prevé que termine para el mes de octubre con la finalización de la documentación, la planificación vendrá dada para organizar el tiempo en el periodo de 10 meses que, teniendo en cuenta las vacaciones de verano, quedará reducido a unos 9 meses.

Se tendrá en cuenta en el diagrama de Gantt que vemos en la Ilustración 3.5 los siguientes periodos asociados a los paquetes de trabajo:

- **Periodo del paquete 1** (Gestión y coordinación): incluye el tiempo transcurrido para llevar a cabo las reuniones con los tutores del proyecto, la especificación de objetivos, hipótesis, restricciones, requisitos iniciales del proyecto y alcance. Dicho tiempo se espera que sean alrededor de 2 semanas. También se incluye en el periodo aquel tiempo destinado al control del proyecto y a la escritura de la documentación.
- **Periodo del paquete 2** (Investigación): periodo dedicado al estudio del estado del arte, lectura de artículos y alternativas para enfocar una correcta solución. Esto no solo incluye los tiempos de investigación, sino que también contiene los tiempos de prueba de software para generación procedural,

búsqueda de conjuntos de datos reales y arranque de proyectos de redes neuronales.

- **Periodo del paquete 3** (Desarrollo): incluye todo el proceso de desarrollo de la aplicación para obtención de ciudades etiquetadas. Dado que el proyecto utiliza una metodología ágil y esta planificación fue ideada al inicio del proyecto, los detalles de las iteraciones realizadas para llevar a cabo el desarrollo planteado se desarrollarán en las secciones posteriores. Es importante mencionar que esta etapa se llevará a cabo concurrentemente con la creación de ciudades procedurales mediante la herramienta seleccionada debido a que de esa manera se podrán hacer pruebas durante el desarrollo.
- **Periodo del paquete 4** (Experimentación): es un periodo sumamente importante, pues en él este proyecto cobrará sentido. Aquí se llevará a cabo todo el proceso del KDD y se obtendrán respuestas a las hipótesis formuladas. Lo más destacable dentro de este periodo es la reserva de tiempo que se hace debido a la incertidumbre para la tarea de obtención de nubes sintéticas, pues es una tarea que requiere mayor coordinación. Dicha tarea se hará de forma concurrente mientras se prueban proyectos de ML.

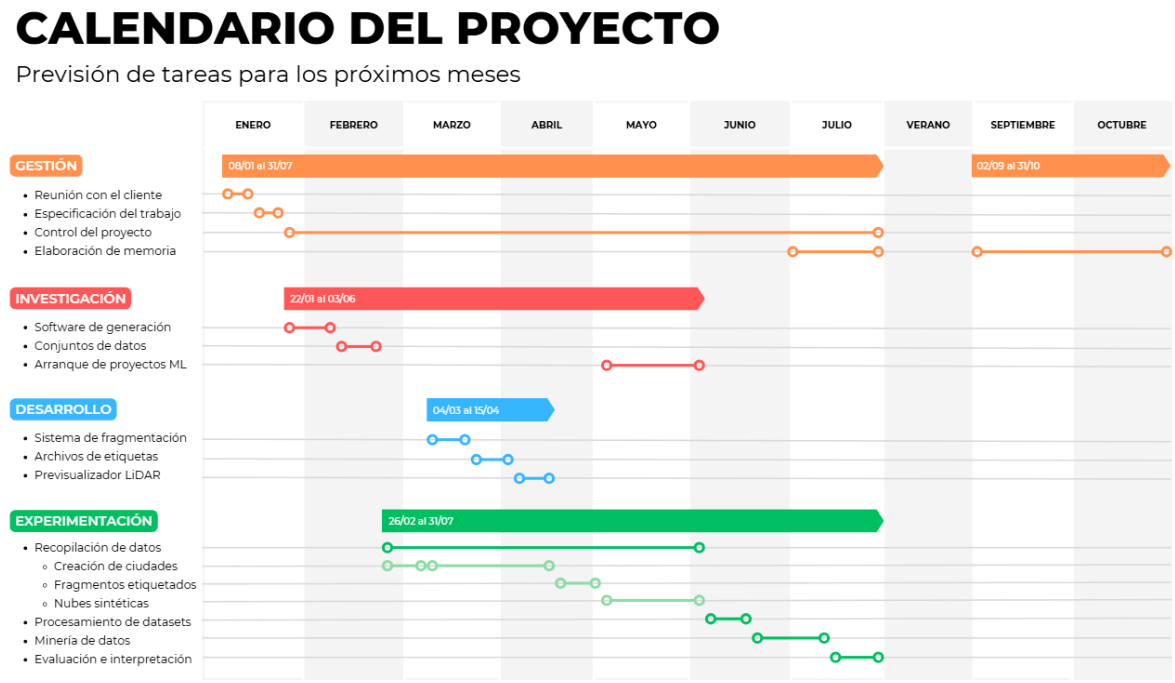

Ilustración 3.5: Calendario del proyecto

3.11 Presupuesto

Valorando las anteriores especificaciones del proyecto ahora se presentará el presupuesto, el cual se desglosará en 2 partes: coste del material utilizado y coste del personal.

Material	Precio (€)	Amortización (€/mes)	Tiempo de uso (meses)	Coste (€)
Equipamiento hardware				
Equipo con 3600TI	1200	20	4	80
Equipo con A4500	3700	61,67	2	123,34
Periféricos (MK370 Combo for Business)	49,99	0,83	6	4,98
Periféricos (monitor Keep Out)	79,90	1,33	6	7,98
Portátil ASUS ROG STRIX G15	1299,99	21,66	9	194,94
Periféricos (ratón Logitech G402)	45,75	0,76	9	6,84
Disco duro 1TB TOSHIBA	69,99	1,17	9	10,53
Software				
Windows 10/11	144,99	2,42	15 (4+2+9)	36,30
Unity 3D	0	0	2	0
CityEngine	1000	16,67	1	16,67
Visual Studio 2022	0	0	6	0
Microsoft Office	149,99	2,50	3	7,50
MeshLab	0	0	3	0
LiDAR virtual	0	0	1	0
Blender	0	0	1	0
SmartGit	0	0	6	0
Umlet	0	0	3	0
Visual Paradigm	0	0	3	0
TOTAL				**489,08**

Tabla 3.15: Costes de material

Como vemos en la tabla, se tiene en cuenta para los cálculos la amortización parcial de los componentes hardware y software, asumiendo un periodo de amortización de 5 años.

Además, se debe considerar el coste del personal, para lo cual usaremos estimaciones salariales obtenidas de la conocida plataforma InfoJobs [46]. Los cálculos se han ajustado según la cantidad de horas y el rol específico, añadiendo también un porcentaje adicional para cubrir la seguridad social, que representa aproximadamente un 33% del salario bruto mensual del empleado.

Tarea	Horas	Rol	Salario (€/h)	Seguridad social (€)	Coste (€)
Análisis y diseño	30	Analista	24,38	241,36	972,76
Investigación	120	Programador (senior)	20,32	804,67	3243,07
Programación	100	Programador (senior)	20,32	670,56	2702,56
Procesamiento de datos	80	Científico de datos	23,36	616,70	2485,50
Experimentación con ML	160	Ingeniero en IA	26,28	1387,58	5592,38
Documentación	100	Programador (junior)	20,32	670,56	2702,56
Testeo	10	Probador	17,83	58,84	237,14
TOTAL					**17935,97**

Tabla 3.16: Costes de personal

Como último aporte, se sumarán el coste del material y el coste del personal, incorporando un sobrecoste del 10% justificado con los gastos indirectos.

Concepto	Coste (€)
Gastos en material	489,08
Gastos en personal	17935,97
Gastos indirectos	1842,51
TOTAL GLOBAL	**20267,56**

Tabla 3.17: Costes totales

Por tanto, nuestro proyecto tendría un precio de **20267,56€**

3.12 Visión general del proyecto

En la Ilustración 3.6 se trata de representar el flujo de trabajo del proyecto de una manera muy clara, concisa y visual. En él se observa como el proyecto inicia con la creación de un conjunto de ciudades de forma procedural haciendo uso de la herramienta de CityEngine. Una vez conseguidas las ciudades, se fragmentarán y etiquetarán con una aplicación propia que se desarrollará en Unity 3D. Dichos fragmentos serán enviados a un LiDAR virtual y, en colaboración con Alfonso se creará un nuevo *dataset* de nubes de puntos sintéticas, Synthetic Cloud 3D. Así se tendrá un *dataset* sintético propio, y un *dataset* de datos reales (que se descargará de Internet) tomados con un sensor LiDAR real en ciudades existentes. De manera coordinada y tras el requerido procesamiento de los datos, ambos *datasets* se enviarán a un grupo de redes neuronales y se harán 4 experimentos con cada una ellas, siendo estas PointNet++, Point Transformer y Umbrella RepSurf.

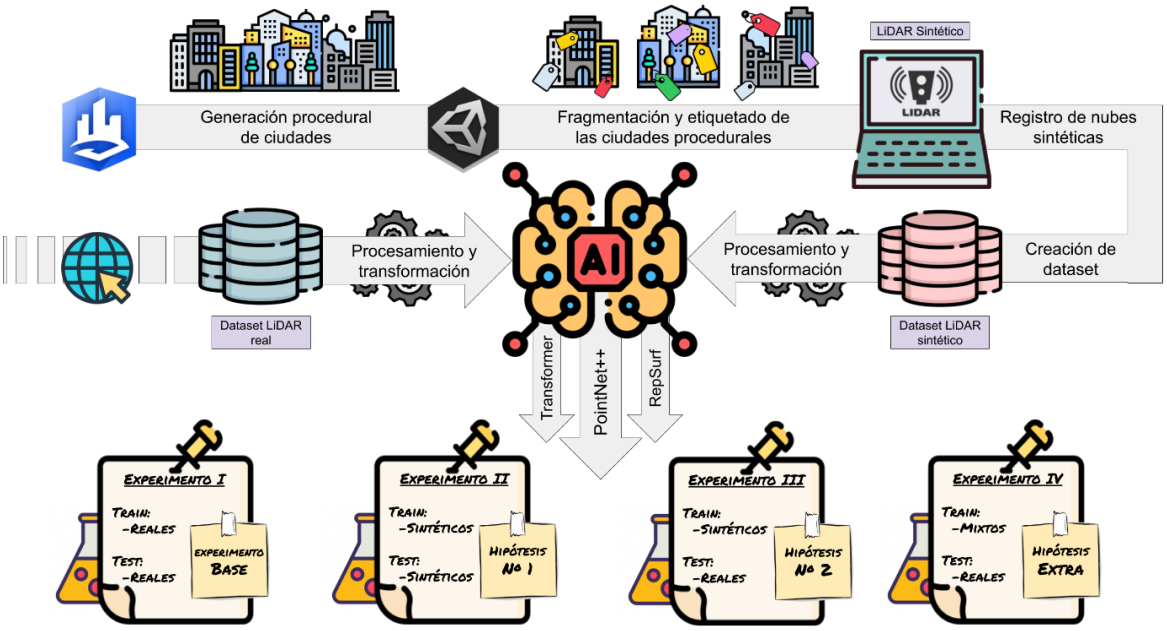

Ilustración 3.6: Flujo de trabajo del proyecto

Centrándonos en los experimentos, el experimento 1 buscará probar las redes con datos únicamente reales y sus resultados serán la referencia para validar las hipótesis con el resto de experimentos. El experimento 2 trata de comprobar la primera

hipótesis propuesta mediante el entrenamiento y prueba del modelo con datos sintéticos. El experimento 3 intenta validar la segunda hipótesis mediante el entrenamiento con datos sintéticos y la prueba del modelo con datos reales. Finalmente, el experimento 4 será un experimento extra para intentar validar una hipótesis adicional. En dicho último experimento se tomarán para el entrenamiento datos sintéticos en su mayoría, junto con unos pocos datos reales (conjunto mixto), y se evaluará el modelo con datos reales.

4 GENERADOR DE FRAGMENTOS DE CIUDADES

En esta sección se describe la etapa de desarrollo de software que se ha tenido que llevar a cabo para crear una aplicación en Unity que permita fragmentar y etiquetar ciudades. Esto son acciones necesarias que no se podían hacer desde la herramienta de CityEngine y sin ellas el sensor LiDAR virtual no funcionaría correctamente. La necesidad de fragmentar las ciudades viene condicionada por el tamaño de las mismas (en número de triángulos) y el soportado por el sensor, mientras que la necesidad de etiquetar la ciudad es imprescindible para que los puntos de la nube devuelta por el LiDAR puedan asociarse a alguna clase (sin esta relación las redes no podrán entrenar).

4.1 Diseño inicial

En este capítulo se busca definir el funcionamiento de la aplicación y establecer sus requisitos iniciales. A través de la etapa de análisis y diseño, se crearán las bases fundamentales para el desarrollo. El objetivo de este proceso es obtener un diseño preliminar que se irá refinando a lo largo de los *sprints* del desarrollo.

4.1.1 Especificaciones del sistema

A continuación, se presentará la especificación detallada de los requisitos tomados al inicio del proyecto durante las reuniones con el cliente. Para definir los requisitos funcionales elaborados utilizaremos la plantilla dada por la Tabla 4.1.

Identificador: Título	
Título	Nombre des*cript*ivo
Descripción	Detalle del requisito e información adicional
Prioridad	Alta/Media/Baja

Tabla 4.1: Plantilla para los requisitos

4.1.1.1 Extracción de requisitos funcionales

Este tipo de requisitos describe el comportamiento esperado del sistema, especificando las funciones y acciones que debe ser capaz de ejecutar. El proyecto comenzará con los siguientes requisitos funcionales:

RF-01: Botón de procesado de la ciudad	
Título	Botón de procesado de la ciudad
Descripción	La aplicación tendrá un único botón que se ocupará de iniciar todo el proceso automáticamente, incluyendo la fragmentación de ciudades, etiquetado, generación de XMLs y simulación de nubes de puntos.
Prioridad	Alta

Tabla 4.2: RF-01

RF-02: Fragmentar modelos de ciudades	
Título	Fragmentar modelos de ciudades
Descripción	El sistema fragmentará modelos de ciudades de forma pseudoaleatorizada dado un radio y una cantidad de vacío máxima permitida. No se permitirá que en una misma ejecución haya fragmentos repetidos
Prioridad	Alta

Tabla 4.3: RF-02

RF-03: Etiquetar mallas	
Título	Etiquetar mallas
Descripción	El sistema detectará la textura de cada malla del objeto inicial y le asociará una etiqueta y un material según unas tablas de conversión. La etiqueta se podrá traducir a otro conjunto soportado por la tabla de conversión (Toronto-3D, Semantic3D, Semantic KITTI o estándar LAS)
Prioridad	Alta

Tabla 4.4: RF-03

RF-04: Generar archivos para LiDAR virtual	
Título	Generar archivos para LiDAR virtual
Descripción	Deberán generarse archivos legibles por el LiDAR virtual tras el etiquetado de los fragmentos de ciudades. Para esto se seguirá el formato XML en el archivo de etiquetas y en el archivo de materiales, y el formato FBX para el fragmento de ciudad
Prioridad	Alta

Tabla 4.5: RF-04

RF-05: Previsualizar nubes LiDAR	
Título	Previsualizar nubes LiDAR
Descripción	Se podrá obtener una previsualización de la posible nube que se obtendrá de cada fragmento. Debe permitir un mínimo de configuración tal y como lo sería el número de rayos, ángulo de apertura o alcance. La simulación será aproximada
Prioridad	Baja

Tabla 4.6: RF-05

4.1.1.2 Extracción de requisitos no funcionales

Estos requisitos establecen los estándares de calidad del software, abarcando aspectos como pueden ser el rendimiento o la consistencia, entre otros. En esta aplicación, los requisitos no funcionales serán:

RNF-01: Rendimiento	
Título	Rendimiento
Descripción	La fragmentación y el etiquetado ofrecidos por la aplicación deben ser rápidos
Prioridad	Media

Tabla 4.7: RNF-01

RNF-02: Interfaz intuitiva y retroalimentada	
Título	Interfaz intuitiva y retroalimentada
Descripción	La aplicación tendrá una apariencia atractiva y será fácil de utilizar y configurar. Durante su uso debe haber un mínimo de retroalimentación para comprobar el proceso que se está llevando a cabo
Prioridad	Baja

Tabla 4.8: RNF-02

RNF-03: Consistencia	
Título	Consistencia
Descripción	La aplicación debe estar depurada para evitar errores, pues esto puede ser una fuente de problemas para fases posteriores del proyecto, así como para investigaciones posteriores
Prioridad	Alta

Tabla 4.9: RNF-03

RNF-04: Portabilidad	
Título	Portabilidad
Descripción	Se debe poder ejecutar la aplicación en cualquier ordenador que cumpla los requisitos mínimos y tenga una versión compatible de Unity
Prioridad	Baja

Tabla 4.10: RNF-04

4.1.2 Análisis y diseño del sistema

4.1.2.1 Diagrama de casos de uso

Para la definición del diagrama de casos de uso de esta aplicación se tendrán en cuenta 2 actores, dados por la Tabla 4.11 y la Tabla 4.12:

A-01: Usuario	
Tipo	Actor principal
Descripción	Persona que usa la aplicación para el procesamiento de las ciudades
Interacción	Directamente sobre la interfaz de la aplicación mediante un botón para lanzar el proceso de fragmentación y etiquetado

Tabla 4.11: Actor usuario

A-02: Sistema	
Título	Sistema
Descripción	Sistema que de forma automatizada tras la orden ejecuta sucesivamente todas las acciones requeridas para el procesamiento de las ciudades
Interacción	Directamente sobre la aplicación, los modelos y los archivos XML

Tabla 4.12: Actor sistema

En la Ilustración 4.1 se representa el diagrama de casos de uso, donde se muestran las interacciones entre los actores y la aplicación. Con esto se trata de describir qué hace el sistema sin entrar en detalles técnicos de implementación.

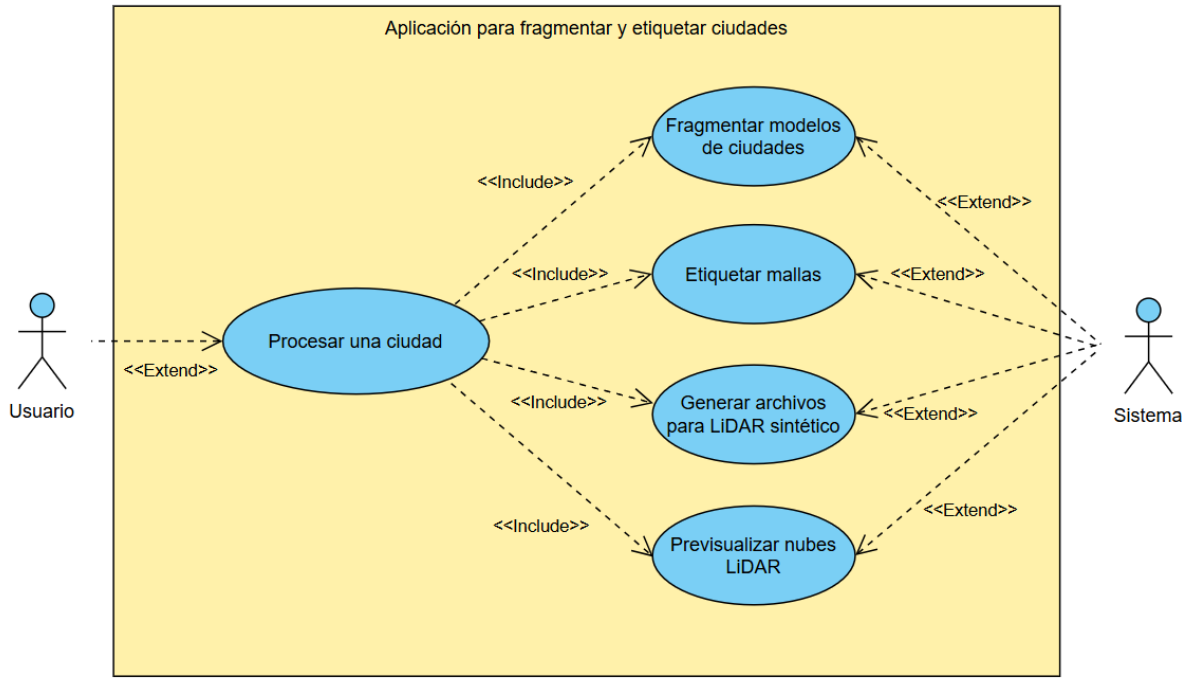

Ilustración 4.1: Diagrama de casos de uso

4.1.2.2 Casos de uso

Para desarrollar los casos de uso creados, se empleará la plantilla especificada en la Tabla 4.13:

Identificador: Título	
Título	Nombre de*scripti*vo
Actor primario	Entidad que realiza el caso
Sistema	Entorno donde se realiza la acción
Nivel	Objetivo de jugador u objetivo de sistema
Precondición	Condiciones que deben haberse dado con anterioridad
Flujo normal	Secuencia de pasos común de ejecución del caso
Flujos alternativos	Secuencia de pasos alternativa de ejecución del caso

Tabla 4.13: Plantilla de casos de uso

Considerando el diagrama de casos de uso definido y los requisitos iniciales, se han desarrollado los siguientes casos de uso:

CU-01: Procesar una ciudad	
Título	Procesar una ciudad
Actor primario	Usuario
Sistema	Aplicación para fragmentar y etiquetar ciudades
Nivel	Objetivo de usuario
Precondición	La ciudad debe estar cargada en Unity y debe haberse marcado el modelo como modificable desde el inspector *(Read/Write)*
Flujo normal	
1	El usuario inicia la aplicación con el motor de Unity
2	El usuario configura los parámetros de fragmentación
3	El usuario configura los parámetros de etiquetado
4	El usuario pulsa el botón central para iniciar el proceso
5	El sistema lleva a cabo una retroalimentación para indicar que se pulsó el botón
6	El sistema comienza el proceso de fragmentado y etiquetado con los parámetros insertados

Tabla 4.14: CU-01

CU-02: Fragmentar modelos de ciudades	
Título	Fragmentar modelos de ciudades
Actor primario	Sistema
Sistema	Aplicación para fragmentar y etiquetar ciudades
Nivel	Objetivo de sistema
Precondición	El usuario debe haber pulsado el botón de procesamiento de la ciudad
Flujo normal	
1	El sistema llama al algoritmo de fragmentado
2	El sistema genera nuevos objetos, sin etiquetar, que suponen un subconjunto de las mallas de la ciudad original
3	El sistema muestra una retroalimentación

Tabla 4.15: CU-02

CU-03: Etiquetar mallas	
Título	Etiquetar mallas
Actor primario	Sistema
Sistema	Aplicación para fragmentar y etiquetar ciudades
Nivel	Objetivo de sistema
Precondición	El usuario debe haber pulsado el botón de procesamiento de la ciudad
Flujo normal	
1	El sistema llama al algoritmo de fragmentado
2	El sistema etiqueta cada malla según la textura que tiene asociada usando la tabla de conversión
3	El sistema asocia un material según la textura que tiene relacionada usando la tabla de conversión

Tabla 4.16: CU-03

CU-04: Generar archivos para LiDAR virtual	
Título	Generar archivos para LiDAR virtual
Actor primario	Sistema
Sistema	Aplicación para fragmentar y etiquetar ciudades
Nivel	Objetivo de sistema
Precondición	El sistema debe haber fragmentado la ciudad y haber etiquetado cada una de las mallas usando el conjunto de etiquetas seleccionado por el usuario
Flujo normal	
1	El sistema toma los resultados del algoritmo de fragmentación y etiquetado
2	El sistema genera archivos en formato FBX para cada uno de los fragmentos de ciudad obtenidos del algoritmo de fragmentado
3	El sistema genera archivos en formato XML según la salida del algoritmo de etiquetado para cada uno de los fragmentos obtenidos
4	El sistema vuelca los archivos FBX y XML dentro de la carpeta de salida

Tabla 4.17: CU-04

CU-05: Previsualizar nubes LiDAR	
Título	Previsualizar nubes LiDAR
Actor primario	Sistema
Sistema	Aplicación para fragmentar y etiquetar ciudades
Nivel	Objetivo de sistema
Precondición	El sistema debe haber fragmentado la ciudad y haber etiquetado cada una de las mallas usando el conjunto de etiquetas seleccionado por el usuario
Flujo normal	
1	El sistema inicia el proceso de simulación LiDAR aproximada
2	El sistema lanza rayos para detectar los puntos de colisión
3	El sistema registra la posición y el color del punto de colisión
4	El sistema muestra una retroalimentación durante varios segundos
5	El sistema guarda en un archivo opcional en formato CSV la nube de puntos obtenida

Tabla 4.18: CU-05

4.1.2.3 Diseño arquitectónico

La Ilustración 4.2 muestra el diagrama UML seguido durante el desarrollo de la aplicación. Si bien es cierto que pueda haber variaciones respecto al producto final debido a la incertidumbre en ciertos aspectos, este diagrama presenta la estructura del proyecto de forma fiel.

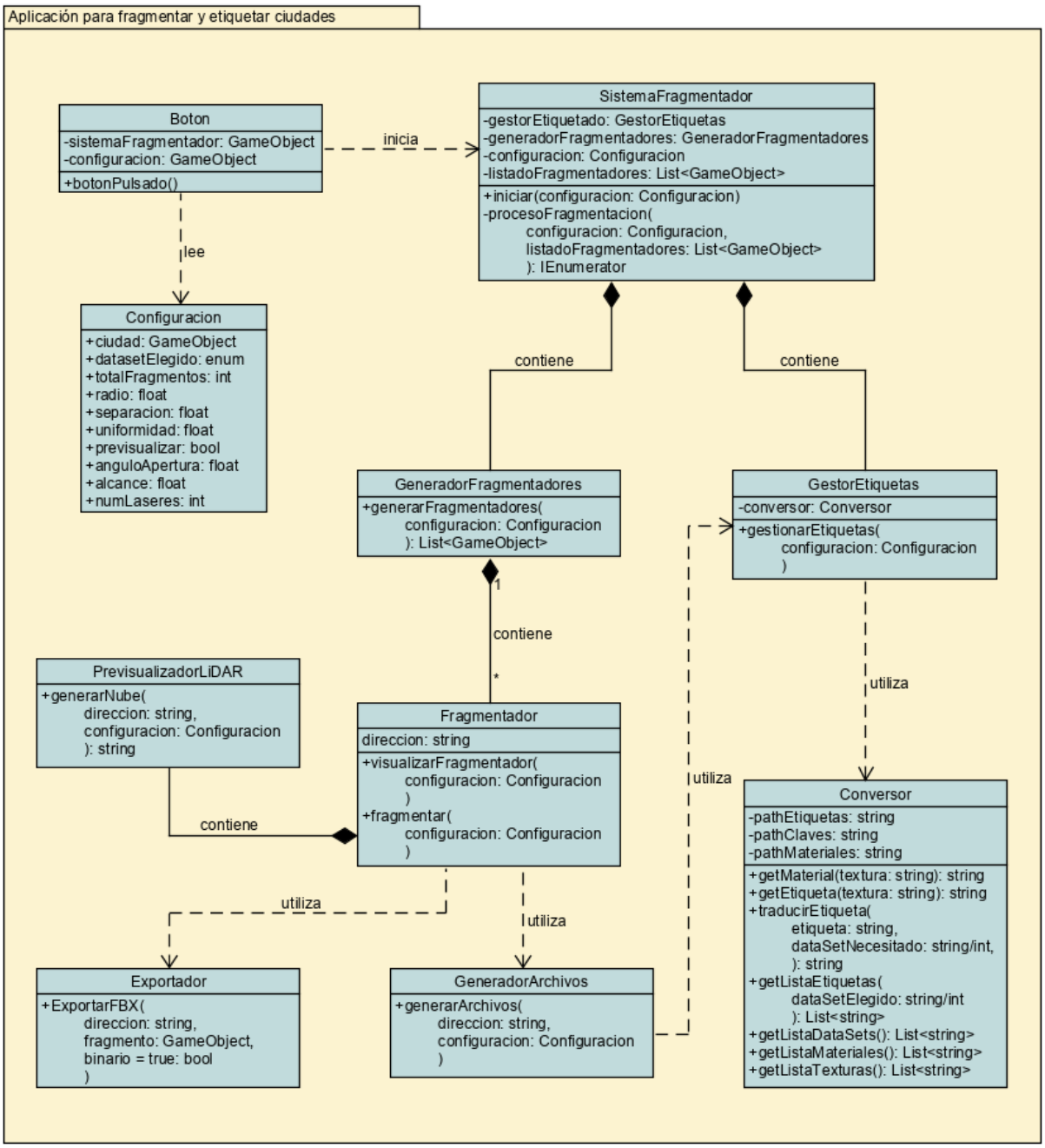

Ilustración 4.2: UML de la aplicación

4.2 Desarrollo del fragmentador de ciudades

A lo largo de esta sección se presentarán y describirán en detalle las distintas iteraciones que se han realizado para el desarrollo de la aplicación en Unity 3D usada para el fragmentado de ciudades. El desarrollo cuenta con 4 partes:

- **Sistema de fragmentación**: trata de, dada una ciudad de gran tamaño, seccionar o particionar la misma en fragmentos disjuntos definidos por un radio.
- **Etiquetado**: se ocupa de indicar para cada malla que compone de la ciudad cuál es su clasificación dado un conjunto de etiquetas. Esto se refleja en la generación de un archivo que asocia mallas y etiquetas. Análogamente al etiquetado se creará un segundo archivo para asociar a las mallas un material.
- **Previsualizador**: será una funcionalidad que será de utilidad para conocer como podrá ser, aproximadamente, el resultado del escaneo con un sensor LiDAR del fragmento enviado.
- **Interfaz**: donde se ajusta la aplicación, se creará un objeto de configuración y se facilitará dentro de lo posible su usabilidad de cara al usuario.

4.2.1 Primera iteración: sistema de fragmentación

Esta primera iteración va directamente relacionada con la tarea 3.1, anteriormente presentada y ha tenido una duración de 2 semanas tal y como se estimó. En concreto, ese tiempo se repartió para trabajar la primera semana en la primera subtarea, es decir, el desarrollo del sistema de instanciación pseudoaleatoria, mientras que la segunda semana se destinó a la creación del objeto fragmentador, lo que corresponde con la segunda subtarea. El resultado de esta tarea ha sido satisfactorio y ha devuelto, tal y como se demandaba, un prototipo funcional con capacidad de fragmentación de ciudades. A continuación, se entrará en detalle y se explicarán ambos desarrollos.

4.2.1.1 Instanciación de fragmentadores

La implementación de este sistema está basada en la ubicación pseudoaleatoria de *GameObjects*, a los que llamamos fragmentadores. Dichas ubicaciones se obtienen del lanzamiento de posiciones aleatorias sobre cada uno de los triángulos que conforman las mallas de la carretera de la ciudad (porque sobre ellas se colocará el sensor LiDAR), para posteriormente seleccionar un conjunto aleatorio de dichas posiciones. Aunque de primeras la idea es muy simple, si se quiere hacer correctamente, debe tenerse en cuenta la cantidad de puntos que se colocarán sobre cada triángulo de los que conforman las mallas de carretera.

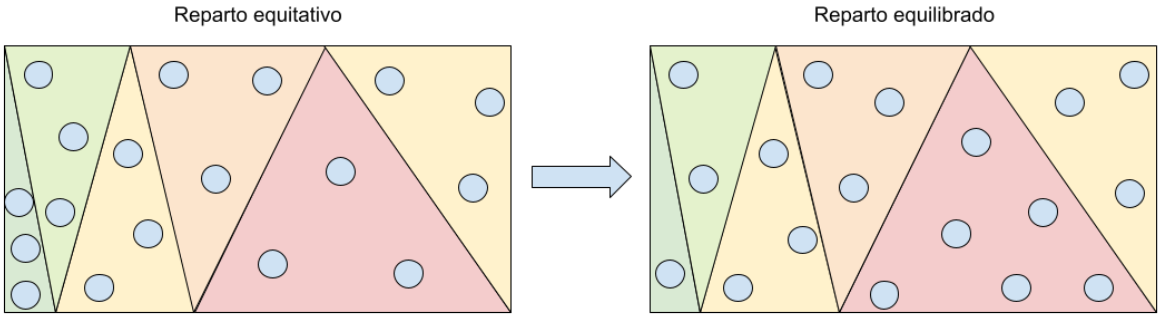

Ilustración 4.3: Reparto democrático de puntos en función del área

Si nos fijamos en el esquema superior, Ilustración 4.3, en el lado izquierdo tenemos el resultado de asignar un número de puntos fijo a cada triángulo, algo que muestra como en zonas con poca área y muchos triángulos hay más puntos generados que en áreas grandes con pocos triángulos. Si por el contrario lo que se deseamos es asignar puntos de forma democrática dentro del espacio de las carreteras, se debe pasar del reparto anterior al que se ve en la derecha del diagrama, donde existiendo el mismo número de puntos, éstos están ubicados de manera más homogénea.

Esto se consigue anotando cada uno de los triángulos que pertenecían a mallas de carretera junto con sus respectivas áreas para asignarles una cantidad de puntos acorde al espacio que ocupan. Dicha cantidad se obtiene mediante el siguiente cálculo para cada triángulo:

$$puntos_{triángulo} = Int\left(\frac{área_{triángulo} * puntos_{total}}{área_{total}}\right) \quad (4.1)$$

Podemos obtener con $área_{triángulo}$ (área del triángulo en cuestión) y $área_{total}$ (sumatoria de todas las áreas de mallas de carreteras) el peso normalizado de cada uno de los triángulos en función del espacio que ocupan. Posteriormente multiplicamos dicho peso por $puntos_{total}$ (puntos a repartir en el espacio) y se trunca a su valor entero, obteniendo el número de puntos a colocar aleatoriamente dentro del triángulo estudiado.

Una vez se obtiene un listado de puntos, seleccionados de forma aleatorizada y correctamente repartidos en el espacio de las carreteras, lo que se hace es seleccionar unos pocos (cantidad que expresa el usuario) para que pasen a ser las posiciones de los objetos fragmentadores. Para ello se estudian valores como las distancias entre los puntos y la cantidad de vacío que habría si el fragmentador generase un fragmento en la posición indicada, algo que se hace con un algoritmo basado en el estudio de las envolventes de las mallas de la ciudad que permite conocer de forma aproximada la cantidad de vacío que tendrá el fragmento. Con ello se asegurará que cada fragmento a generar sea diferente, completamente o en parte disjunto con el resto, y que no tenga zonas vacías. En la siguiente imagen, Ilustración 4.4, vemos como el sistema ubica los objetos fragmentadores, representados en rojo con transparencia.

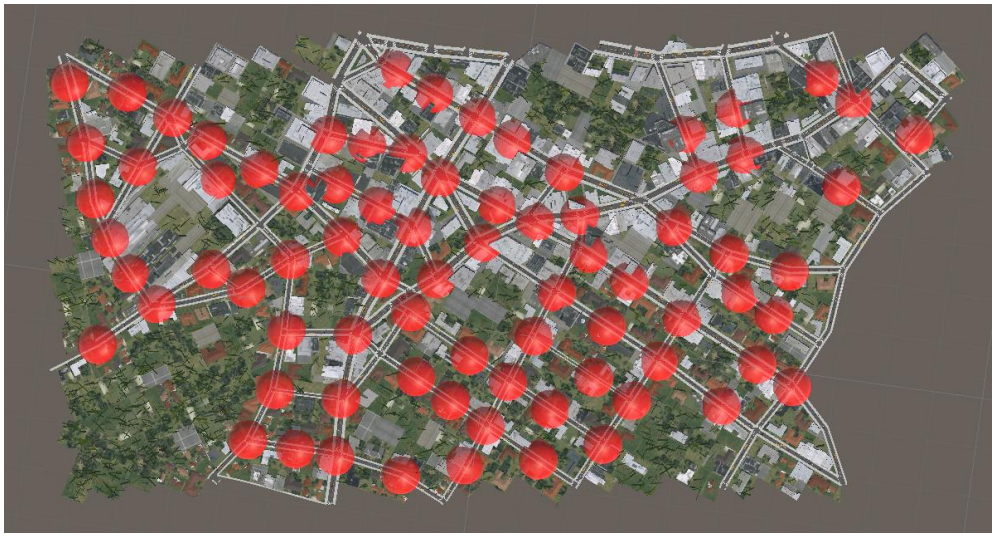

Ilustración 4.4: Instanciación pseudoaleatoria de fragmentadores

4.2.1.2 Objeto fragmentador

Durante este apartado se ha estado nombrando al objeto fragmentador, y lo cierto es que es la clave para la generación de fragmentos de ciudades. La forma de configuración de este objeto ha ido cambiando a lo largo del proyecto, mejorándose y facilitándose para el usuario, por lo que se detallará en el apartado de configuración para ver los parámetros finales a destacar. El funcionamiento del fragmentador se basa en la detección recursiva de todas las cajas envolventes existentes para comprobar si estas pertenecen completamente o parcialmente a la esfera de alcance definida por el radio especificado.

Una pregunta común acerca de este mecanismo sería que por qué no se utilizan los colisionadores de Unity. La respuesta viene de la mano del objetivo de la búsqueda de la eficiencia. Dar a todos los objetos el componente de colisión provocaría que la aplicación redujese su velocidad. Evitando tomar colisionadores, lo que se ha utilizado es la envolvente que por defecto tienen todos los objetos en Unity. Comprobando su envolvente se obtienen 8 puntos que definen el hexaedro, y con ellos podemos asegurar que un objeto está dentro del alcance si se cumple al menos una de estas condiciones, evaluadas en este orden:

1. Su caja envolvente tiene algún vértice a menor o igual distancia que la definida por el rango de detección
2. Su caja envolvente tiene un punto perteneciente a alguna arista que se encuentra en colisión con la esfera que se define por el rango de detección.

En cuanto a estos condicionales podemos decir que el primero es trivial, sin embargo, el segundo es algo más complejo. Por suerte, Unity permite hacer raycasting a bajo coste, por lo que aprovechando que la esfera de visualización roja sí que tiene colisionador, se lanzarán rayos en búsqueda de colisiones. Estos rayos viajarán entre los puntos que definen las aristas de las envolventes, devolviendo un punto en caso de colisionar con la esfera o nulo en caso contrario. De esta forma sabemos gracias a la envolvente si los objetos pertenecen o no al fragmento definido por un radio, dando resultados como el siguiente.

Ilustración 4.5: Fragmento de ciudad

4.2.1.3 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el sistema de fragmentación.

T-01 Test de instanciación de fragmentadores	
Objetivo	Las ubicaciones que devuelve el sistema para la generación de fragmentos son correctas, dando a cada área el peso correspondiente para la posterior instanciación
Resultado	El sistema respeta la ubicación de los fragmentos siguiendo la configuración y condiciones de los parámetros sin problema
Observaciones	Si la ciudad completa es pequeña, o el número de puntos demandados muy grande, el sistema no suele conseguir generar la cantidad de fragmentos que se le solicitan

Tabla 4.19: T-01

T-02 Test de fragmentación	
Objetivo	Los fragmentadores son capaces de detectar las envolventes de los objetos de la ciudad, diferenciando aquellas que están dentro y fuera del rango de detección
Resultado	El fragmentador consigue con éxito delimitar cuáles son los objetos pertenecientes al fragmento según la configuración
Observaciones	Si el interior del fragmento tiene “lagunas vacías”, es decir, huecos que no están pegados al límite del fragmento, cabe la posibilidad de que, por el funcionamiento interno que tiene no se detecten

Tabla 4.20: T-02

4.2.2 Segunda iteración: generación de archivos y etiquetado

Esta segunda iteración está directamente vinculada con la tarea 3.2, previamente mencionada, y ha tenido una duración de dos semanas, como estaba previsto. En detalle, podemos decir que el tiempo se distribuyó de la siguiente manera: durante la primera semana se trabajó en la primera subtarea, que consistía en el desarrollo de un sistema para la gestión y la conversión entre diferentes conjuntos de etiquetas, mientras que la segunda semana se dedicó a la codificación de la función para generar archivos XML legibles para el sensor LiDAR virtual. El resultado de esta tarea ha sido satisfactorio y ha devuelto un prototipo funcional con capacidad de fragmentación y etiquetado de ciudades. A continuación, se explicarán ambos desarrollos en detalle.

4.2.2.1 Gestión y conversión del etiquetado

El LiDAR virtual requiere que los objetos estén clasificados. Para esto se diseñó y se crearon varias tablas de conversión para etiquetar los objetos y traducir dicho etiquetado entre diferentes *datasets*. De esta manera, se podrá configurar el sistema para que etiquete diferentemente los objetos de las ciudades según si el conjunto de etiquetas es el utilizado en los conjuntos LAS [47], Toronto-3D, Semantic3D y Semantic KITTI (conjuntos de etiquetas disponibles desde sus páginas oficiales), incluyendo además un nuevo conjunto de etiquetas propias (el más específico de todos). Concretamente se necesitaron 3 tablas:

- TABLA_CLAVES: tabla que asocia el nombre de las texturas de los objetos con el conjunto de etiquetas propio. La clave es el nombre de la textura y el valor es la etiqueta correspondiente (*dataset* propio). En la Ilustración 4.6 se puede ver un fragmento de la tabla.
- TABLA_CONVERSION_MATERIALES: tabla que asocia el nombre de las texturas de los objetos con el nombre del material. La clave es el nombre de la textura y el valor el material. En la Ilustración 4.7 se puede ver un fragmento de la tabla.

CLAVE_MATERIAL	ETIQUETAS_PROPIAS
AcerSaccharum	VegetacionAlta
AiphanesHorrida	VegetacionAlta
asphalt	Carretera
Asphalt003_2K_Color	Carretera
Asphalt003_2K_white_Color	MarcaVial
Asphalt003_2K_yellow_Color	MarcaVial
audi	Coche
bench	Banco

Ilustración 4.6: Tabla de claves

CLAVE_MATERIAL	ETIQUETAS_PROPIAS
AcerSaccharum	WOOD
AiphanesHorrida	WOOD
asphalt	STONE
Asphalt003_2K_Color	STONE
Asphalt003_2K_white_Color	STONE
Asphalt003_2K_yellow_Color	STONE
audi	ALUMINIUM
bench	WOOD

Ilustración 4.7: Tabla de materiales

- TABLA_CONVERSION_ETIQUETAS: tabla que traduce del conjunto de etiquetas propias a otro conjunto de etiquetas. La clave es la etiqueta propia correspondiente, y el valor es, en función del *dataset* seleccionado, la traducción de la etiqueta del *dataset* propio al *dataset* elegido. Como nota adicional, es importante mencionar que cada etiqueta también tiene un conjunto numérico asociado. Esta tabla luce similar a como se ve en la Ilustración 4.8.

CLAVE_ETIQUETAS	ETIQUETAS_PROPIAS	Num_ET	Estandar_LAS	Num_Es	Semantic3D	Num_Se	SemanticKITTI	Num_Se	Toronto_3D	Num_Toronto_3D
ID	ETIQUETAS_PROPIAS	Num_ET	Estandar_LAS	Num_Es	Semantic3D	Num_Se	SemanticKITTI	Num_Se	Toronto_3D	Num_Toronto_3D
Acera	Acera	0	GROUND	2	hard scape	6	sidewalk	48	Road	1
Agua	Agua	1	WATER	9	natural terrain	2	other-ground	49	Natural	3
Autobus	Autobus	2	NOISE	7	cars	8	other-vehicle	20	Car	7
Banco	Banco	3	BUILDING	6	scanning artefacts	7	other-object	99	unclassified	0
Cable	Cable	4	WIRE_CONDUCTOR	14	scanning artefacts	7	other-object	99	Utility line	5
Carretera	Carretera	5	ROAD_SURFACE	11	man-made terrain	1	road	40	Road	1
Ciclista	Ciclista	6	NOISE	7	cars	8	bicyclist	31	Car	7

Ilustración 4.8: Tabla de conversión de etiquetas

Estas tablas se interconectan entre ellas para que, dada una textura de entrada, se relacione con la etiqueta y material correspondiente. Por ejemplo, y siguiendo el esquema de la Ilustración 4.9, si uno de los objetos de la ciudad es un tronco de árbol, en primer lugar, se comprobaría la textura que le da color desde las propiedades. Una vez conocida la textura (*text_trunk_002*), esta se toma como clave de las tablas de etiquetas y materiales, comprobando respectivamente cuál es su etiqueta por defecto y su material. Si se deseara utilizar como referencia el etiquetado de Toronto-3D, la etiqueta debería traducirse a una etiqueta de dicho *dataset*. De esta manera, habiendo tomado como entrada un trozo que correspondía a una parte de un árbol de la ciudad, se ha conseguido una salida con la etiqueta de Toronto-3D (Natural) y el material (Wood).

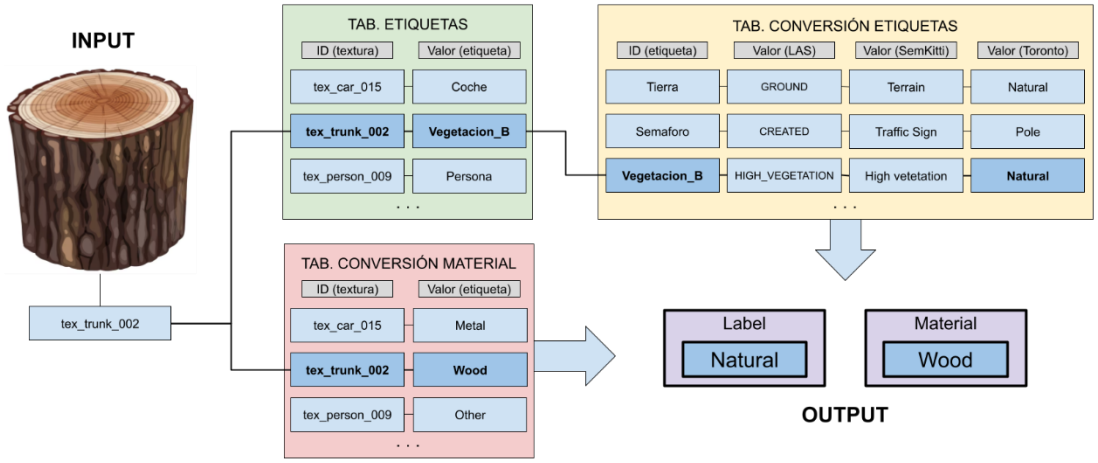

Ilustración 4.9: Proceso de etiquetado de mallas

Es importante comentar que el sistema se hizo de manera que posteriormente fuese fácil ampliar estas tablas siguiendo la documentación que acompaña a las mismas, junto con la información encontrada en este documento.

4.2.2.2 Generación de archivos XML para LiDAR virtual

La gestión de etiquetado presentada se consigue gracias a la ejecución del código del fragmentador. Tras ese proceso se prosigue con el etiquetado interno en Unity, y luego se lleva a cabo la creación de un conjunto de archivos necesarios para el LiDAR virtual que refleje dicha información. Estos archivos seguirán el formato XML, lenguaje de marcado que permite presentar la información de una manera estandarizada, relacionando texturas con materiales y etiquetas. El formato seguido es el siguiente para el archivo de etiquetado LAS en XML, siendo similar para los demás archivos.

```
<scene>
      <!-- ASPRS classes are listed below : -->
      <!-- GROUND, NOISE, BUILDING, ROAD_SURFACE, CREATED,  -->
      <!-- HIGH_VEGETATION, LOW_VEGETATION, UNCLASSIFIED -->
      <!-- ASPRS class Naming of objects affected by ASPRS classes -->

      <label key="GROUND">
          <name>
              curb
          </name>
      </label>
      <label key="ROAD_SURFACE">
          <name>
              Asphalt_
          </name>
          <name>
```

```
17.              Road_white_Color
18.          </name>
19.      </label>
20.
21.      ...
22.
23.      <label key="UNCLASSIFIED">
24.      </label>
25.
26. </scene>
```

Nótese que se está asociando patrones de texto en lugar del nombre completo de las texturas. Por ejemplo, las texturas “Asphalt_001”, “Asphalt_002” y “Asphalt_003” vienen representadas por “Asphalt_”. Esto es útil para reducir espacio y aumentar la velocidad de procesamiento del archivo. Por supuesto esta solución está adaptada al LiDAR que se utilizará dentro de esta investigación, y en caso de utilizar otro sensor habría que prepararlo para ser capaz de procesar este formato.

4.2.2.3 Pruebas

A continuación, se muestran test realizados para el sistema de etiquetado.

T-03 Test de gestión de la traducción	
Objetivo	Todo objeto se etiqueta correctamente y se permite traducir la etiqueta a cualquier otro conjunto
Resultado	El sistema etiqueta y es capaz de traducir las etiquetas entre diferentes conjuntos con éxito
Observaciones	Las traducciones solo pueden hacerse de un conjunto específico hacia otro conjunto más general. Si el conjunto de origen es menos específico que el de destino, una etiqueta podría traducirse de varias maneras, produciendo errores

Tabla 4.21: T-03

T-04 Test de gestión de materiales	
Objetivo	Se asocia a todo objeto un material haciendo uso de la tabla de materiales
Resultado	Todos los objetos son asociados correctamente a un material
Observaciones	

Tabla 4.22: T-04

4.2.3 Tercera iteración: previsualizador de nubes LiDAR e interfaz

Se trata de la última iteración, relacionada con la tarea 3.3, cuya duración fue de 1 semana, algo que equivale a la estimación inicial. En ese tiempo se desarrolló principalmente funcionalidades de ayuda para el usuario, mejorando la interfaz y su configuración, pero sobre todo introduciendo la opción de previsualización LiDAR. El resultado de esta tarea ha sido satisfactorio y ha devuelto, tal y como se demandaba, la aplicación completamente funcional. A continuación, se entrará en detalle y se explicarán ambos desarrollos.

4.2.3.1 Obtención de nubes de previsualización

La obtención de nubes de previsualización fue una funcionalidad que se introdujo con el objeto PrevisualizadorLiDAR como una herramienta adicional para saber aproximadamente cómo se vería la nube que devolvería el LiDAR virtual. De esta forma, podrían detectarse posibles errores antes de utilizar el sensor. Sabiendo esto, y como es de esperar, este mecanismo no simula un sensor LiDAR como tal, por lo que no ha de esperarse dicho comportamiento. Si se desea simular un LiDAR deberá utilizarse un simulador correctamente diseñado.

Para la generación de estas nubes de puntos, el objeto fragmentador permite activar una opción para reproducir la emisión de rayos de un LiDAR. Si esta funcionalidad está activada se llama a una función que lanza Y rayos por cada una de las X capas, de forma que:

- X indica cuántas divisiones se realizan en el plano horizontal para abarcar los 360º de visión. Por ejemplo, si tenemos el valor 40 para X, significa que se lanzarán ráfagas de rayos en vertical separadas cada 9º.
- Y indica cuantos rayos se envían en cada ráfaga vertical. Por ejemplo, si tenemos una Y de valor 20, quiere decir que por cada ráfaga se envían 20 rayos.

Así si lanzasemos una ejecución con X = 40 e Y = 20, la simulación estaría constituida por 800 rayos dando una nube etiquetada de hasta ese número de puntos

en función de la cantidad de colisiones. El objeto previsualizador también permite un mínimo de configuración, tal como lo es el ángulo de apertura o el rango de detección.

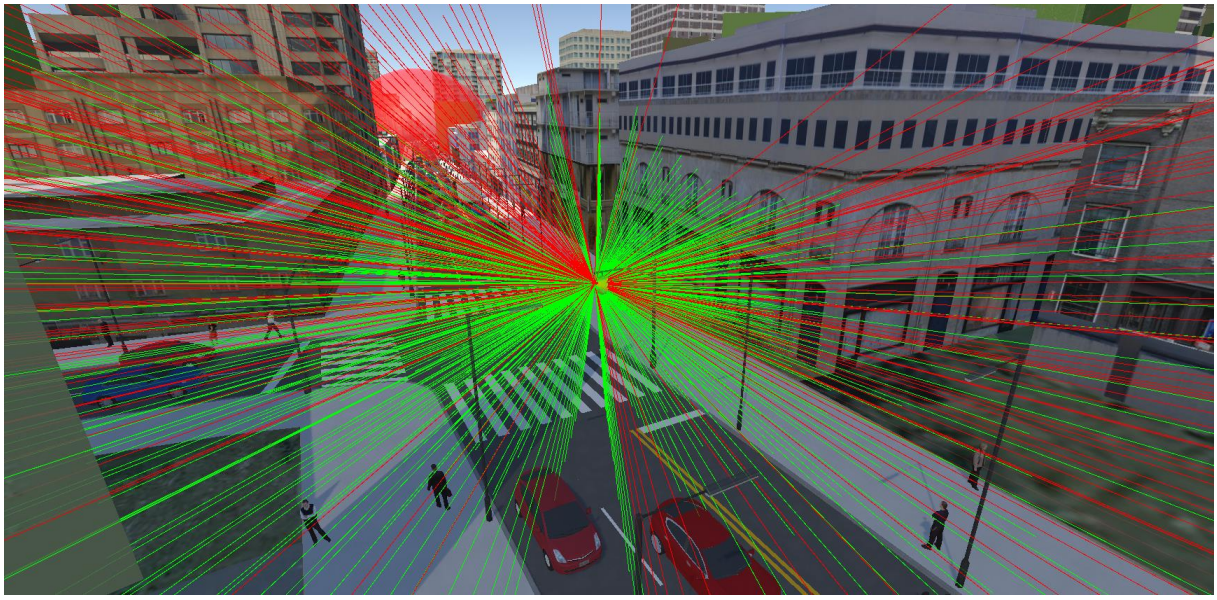

Ilustración 4.10: Funcionamiento del previsualizador LiDAR

En la Ilustración 4.10 se ve cómo funciona el sistema con la configuración comentada y teniendo un ángulo de apertura de 50 grados y 120 de rango de detección. Obsérvese que los rayos se muestran coloreados para mayor facilidad de depuración, siendo visibles desde la escena durante unos segundos: en verde si colisionan, y en rojo en caso contrario. Como era de esperar, cuanto mayor es el número de rayos, mayor es la densidad de la nube de puntos, aunque no se ha elevado demasiado para no complicar la explicación. El formato en el que se devuelve la nube de puntos es CSV, de forma que cada línea del archivo es un punto.

4.2.3.2 Interfaz de la aplicación

La interfaz de usuario ha sido un punto en el que se ha ahorrado algo de tiempo. Tiene la interfaz necesaria para hacer todo lo requerido, así como un objeto para encapsular toda la configuración, pero visualmente no tiene gran impacto. Esto es así porque el objetivo principal de este proyecto no es hacer esta aplicación, sino que esta es solo un eslabón necesario para la investigación posterior con redes neuronales.

Esta interfaz consta de un solo botón, incluido dentro del *canvas* de la escena para iniciar el proceso de fragmentación. Dicho botón inicia el proceso aplicando la

configuración del objeto configurador, un objeto modificable desde el inspector que ha sido creado para encapsular toda la información necesaria. En la Ilustración 4.11 puede verse la interfaz desarrollada.

Ilustración 4.11: Interfaz de la aplicación

En cuanto a la configuración permitida, el objeto responsable incorpora una serie de parámetros modificables desde el inspector:

- Ciudad: se le debe pasar un *GameObject* que contenga el objeto a fragmentar.
- *Dataset* Elegido: es un desplegable que permite seleccionar el conjunto de etiquetas que se utilizará para nombrar a cada elemento.
- Total Puntos: es el número de instancias de fragmentadores que se pueden generar como máximo. Cabe destacar que por temas de eficiencia, el algoritmo no asegura que se cree en todos los casos el número demandado de fragmentadores.
- Capa: será el tipo de etiqueta que debe tener asociada cualquier malla que esté en evaluación para ubicar sobre ella objetos fragmentadores. Por defecto es “Carretera”, pero puede tomarse otra etiqueta siempre que se nombre igual que en el conjunto de etiquetas propias.
- Radio Fragmento: define una esfera que será utilizada para posteriormente generar el fragmento, marcando así sus límites.

- Incremento Altura: permite dar un pequeño desfase sobre el eje Y en la ubicación de los fragmentadores, siendo este un valor entre el máximo y el mínimo solicitado.
- Distancia Separación: es la distancia mínima que debe haber entre el centro de cada fragmento generado.
- Uniformidad Fragmentos: indica la cantidad de espacio vacío que puede haber dentro de cada fragmento. Si el valor es cercano a 1 se tratará de evitar que el centro de los fragmentos se genere demasiado cerca de los bordes de la ciudad, evitando crearlos con vacio proveniente de esa causa. Si por el contrario el valor es cercano a 0, se permitirá instanciar los generadores en cualquier punto de la ciudad, independientemente de su cercanía con el borde.
- Generar Previsualización: es un *booleano* que si se verifica se activa la funcionalidad de previsualización y genera posibles nubes de puntos. Por defecto se encuentra desactivada la opción porque ralentiza el proceso.
- Ángulo Apertura: es el ángulo del abanico de emisión de rayos en vertical, partiendo desde el centro del fragmentador.
- Distancia Máxima: es el rango máximo de detección de colisiones para los rayos emitidos.
- Número Láseres X: cantidad de ráfagas emitidas en el plano horizontal para abarcar los 360º de visión.
- Número Láseres Y: cantidad de láseres lanzados en vertical por ráfaga para cubrir el espacio del abanico generado por el ángulo de apertura.

4.2.3.3 Pruebas

A continuación, se van a mostrar los resultados de los test realizados para el sistema de previsualización y la interfaz de usuario.

T-05 Test de correspondencia entre previsualización y fragmento	
Objetivo	Las nubes generadas muestran de forma fiel el fragmento en forma de nubes de puntos, respetando colores, posiciones de puntos y etiquetado
Resultado	La previsualización etiqueta correctamente, respeta colores y las posiciones se corresponden con las colisiones de los rayos simulados
Observaciones	

Tabla 4.23: T-05

T-06 Test de archivos del previsualizador	
Objetivo	Los archivos son legibles y tienen un formato que permite su visualización fácilmente con algún software
Resultado	Los archivos generados pueden leerse desde programas como MeshLab y muestran la información correctamente
Observaciones	Debe ajustarse desde el programa de visualización el rango RGB para que los colores se muestren correctamente

Tabla 4.24: T-06

T-07 Test de funcionamiento de la interfaz	
Objetivo	La interfaz funciona correctamente y permite configurar todo el sistema de manera manejable
Resultado	El botón principal funciona perfectamente y su configuración desde el inspector de Unity es muy intuitiva
Observaciones	Se puede pulsar el botón más de una vez sin que haya fallos de ningún tipo En función del tamaño de la ciudad a fragmentar, en ocasiones la interfaz tarda en mostrar retroalimentación debido a los cálculos que se hacen entre los *frames* anteriores y posteriores a la retroalimentación

Tabla 4.25: T-07

5 EXPERIMENTACIÓN CON REDES NEURONALES

En esta segunda etapa del proyecto, lejos de lo que hemos visto hasta el momento con el desarrollo de una aplicación (sección 4 Generador de fragmentos de ciudades) nos centraremos en un proceso completamente experimental. Para ello, se ha seguido la conocida metodología Knowledge Discovery in Databases (KDD), la cual consta de las siguientes fases, que se detallarán en los próximos subapartados:

- Recopilación de datos
- Preprocesamiento y transformación de datos
- Minería de datos
- Evaluación e interpretación

5.1 Recopilación de datos

La recopilación de datos se divide en la recopilación de datos reales y en la elaboración de datos sintéticos.

5.1.1 Conjunto de datos reales

Los conjuntos de nubes de puntos de entornos urbanos reales fue la tarea de recolección de datos más fácil y rápida, pues en Internet existen múltiples opciones, tal y como ya se ha visto a lo largo del apartado 2 (Antecedentes y estado del arte). Aunque en un inicio se planteaba utilizar los conjuntos de Semantic3D, Semantic KITTI y Toronto-3D, lo cierto es que las limitaciones temporales no permitieron hacerlo. Es por ello que, en búsqueda de hacer un estudio de máxima calidad dado el contexto, se decidió reducir el número de conjuntos reales para utilizar únicamente el conjunto de datos Toronto-3D, algo que se obtiene desde la página oficial.

5.1.2 Conjunto de datos sintéticos

La recopilación de datos sintéticos fue algo más trabajoso debido a que en gran parte es el interés del proyecto. Para obtener dicho conjunto se hicieron 3 tareas: la generación de ciudades, el fragmentado con etiquetado de ciudades y la obtención de nubes de puntos a partir de un LiDAR virtual.

5.1.2.1 Generación de ciudades mediante reglas

Para la creación de ciudades se ha utilizado el software de CityEngine, presentado en el apartado 3.4.1.2 (ArcGIS CityEngine). Usar esta aplicación es algo complicado y que, pese a ser una herramienta muy potente, tiene una curva de aprendizaje notable. A continuación, se va a mostrar los pasos a seguir para generar una ciudad básica, aunque como es de esperar, estas se pueden personalizar mucho incluyendo más reglas.

El primer paso será dirigirnos a File→New y crear una ciudad mediante la opción “City Wizard” (Ilustración 5.1). Desde ahí se solicitará información básica como el nombre de la ciudad, las dimensiones de la misma, altura máxima y varias direcciones donde están los mapas de altura, textura y obstáculos (Ilustración 5.2).

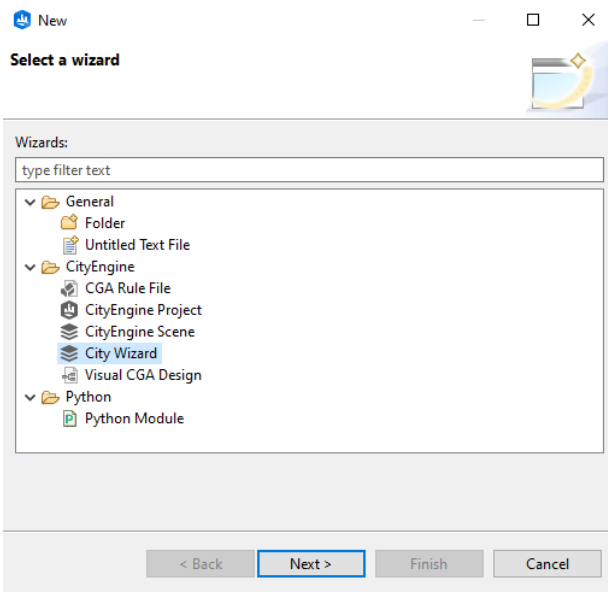

Ilustración 5.1: Crear proyecto CityEngine

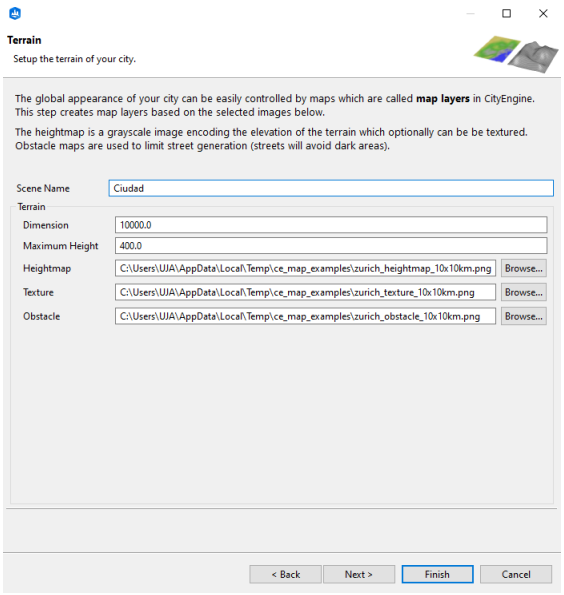

Ilustración 5.2: Configurar proyecto

Tras estas configuraciones, CityEngine pide que se seleccione un tipo de puerto carretero (Ilustración 5.3). Existen múltiples posibilidades, pero para este ejemplo se elegirá el modelo de carreteras de París, un modelo radial. Posteriormente se nos pregunta sobre el tipo de ciudad que buscamos. Esto aplicará una serie de reglas por defecto que más tarde se podrán sustituir o modificar. En este caso se dejará el conjunto “International City” (Ilustración 5.4). Hecho esto, se le da al botón para finalizar y tras varios segundos deberá mostrarse la ciudad generada.

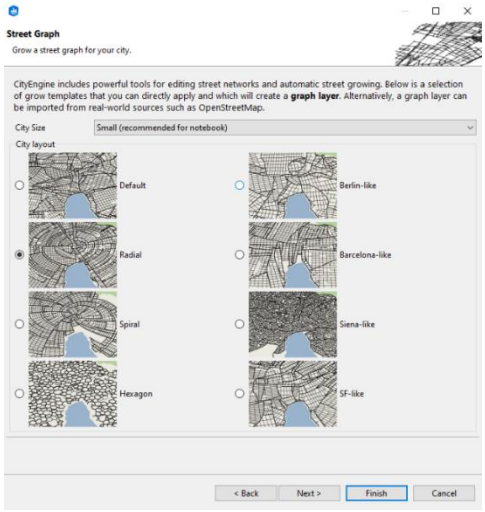

Ilustración 5.3: Puerto carretero

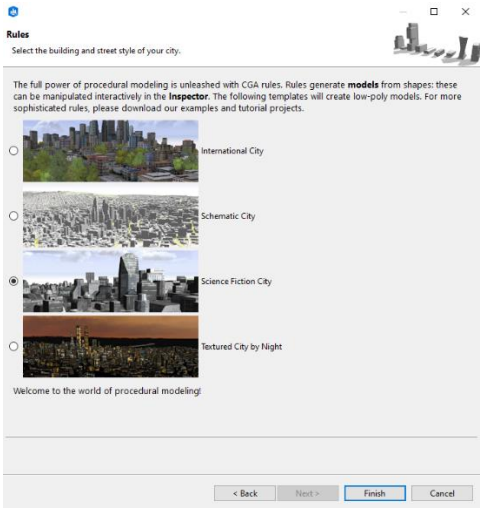

Ilustración 5.4: Reglas iniciales de generación

Una vez cargada la ciudad, debe verse una interfaz similar a la de la Ilustración 5.5, que como se observa, recuerda a la usada en otros programas como Unity o Blender. En ella tenemos una zona para ver los elementos de la escena (arriba a la izquierda), otra zona para poder ver los archivos y la jerarquía de carpetas (abajo a la izquierda), un espacio central que muestra la escena con la ciudad, y una última zona reservada para el inspector (lado derecho), que es utilizada para configurar las reglas utilizadas.

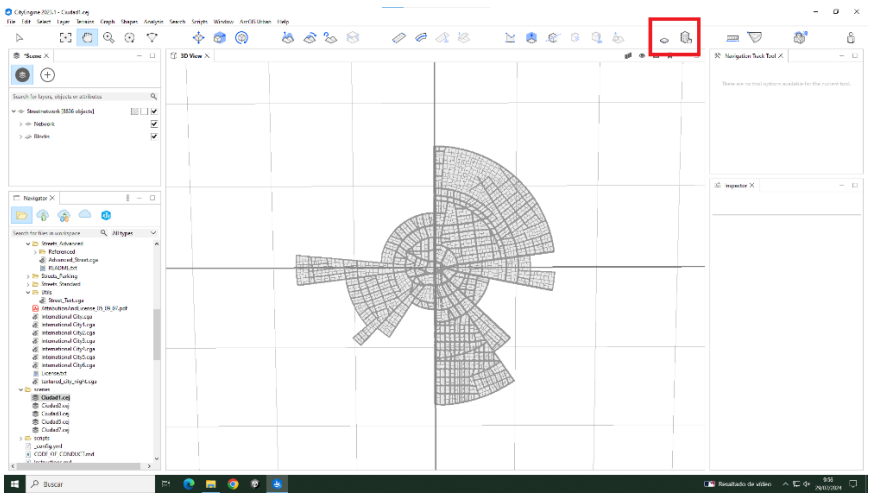

Ilustración 5.5: Interfaz de CityEngine

Sin embargo, como podemos apreciar aún no hay nada más que un puerto carretero. Esto es porque se tiene que construir la ciudad. Para ello deberán incluirse en el proyecto las reglas que se desean utilizar y posteriormente darle al botón

marcado en la Ilustración 5.5. Una vez hecho esto, la ciudad cambia su apariencia y obtiene un posible resultado final, proceso visible y comparable entre la Ilustración 5.6 y la Ilustración 5.7, donde se ha pasado automáticamente del puerto carretero esquematizado a la ciudad detallada.

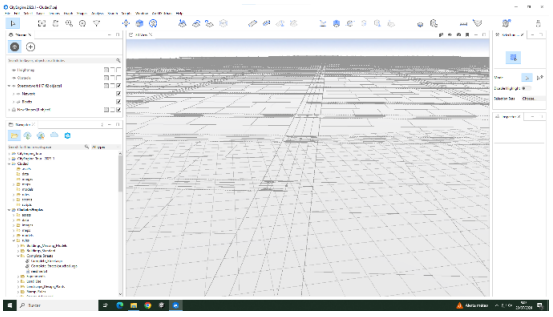

Ilustración 5.6: Ciudad sin construir

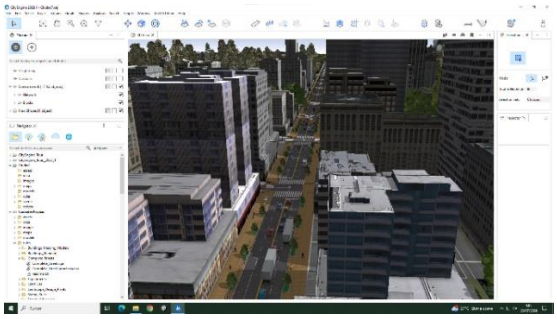

Ilustración 5.7: Ciudad construida

El conjunto de reglas que se definen es muy interesante porque cambiando estas se pueden obtener resultados muy diversos como los que se muestran a continuación en la Ilustración 5.8. Algunas de esas ciudades se crearon con conjuntos de reglas propuestos por archivos de ejemplo (aunque por desgracia no se han podido usar todos por problemas con Unity), mientras que otros han sido tomados de Internet y adaptados.

Ilustración 5.8: Ciudades generadas con CityEngine

5.1.2.2 Fragmentado y etiquetado de ciudades

Para la construcción de los fragmentos de ciudades etiquetados ha de exportarse el modelo de la ciudad e incluirlo dentro de la escena de Unity, activando desde el inspector para todas las mallas que lo compongan la opción de lectura y escritura (tic *Read/Write* activo). Cabe recordar en este apartado que Unity tiene problemas para la identificación de mallas cuando hay demasiados objetos dentro del proyecto, por lo que, si la ciudad no se está usando en el momento, lo correcto es enviarla a la carpeta "*Fragmentador de ciudades\Assets\CiudadesCompletas\Otras ciudades~*" (invisible desde Unity), y en caso de estar usándose, pasarla a la carpeta anterior "*Fragmentador de ciudades\Assets\CiudadesCompletas*". Hecho esto, deberá revisarse si todas las texturas están registradas dentro de las tablas de etiquetas y de materiales, algo que se hace automáticamente y es avisado por consola. En caso de no estar alguna textura o patrón registrado, deberá añadirse en ambas tablas, pues de no ser así fallará. El objeto u objetos que compongan la ciudad deben incluirse de manera que sean objetos hijos del objeto "*Ciudad*", conectado con el objeto que permite la configuración del sistema, el objeto "*Configuracion*". Finalmente, y tras configurar los parámetros deseados, habría que iniciar la aplicación en Unity para pulsar el único botón de la interfaz e iniciar el proceso. Cabe mencionar que los parámetros configurables se explicaron anteriormente, por lo que se recomienda haber leído el capítulo 4 (Generador de fragmentos de ciudades). Consecuentemente, tras varios segundos, encontraremos los fragmentos etiquetados dentro de la carpeta "*Fragmentador de ciudades\Assets\FragmentosGenerados~*".

Los fragmentos deberían lucir de manera similar a como se muestra en la Ilustración 5.9, y en caso de tener activada la opción de previsualización de nube de puntos, debería adjuntar además una nube similar a la de la Ilustración 5.10.

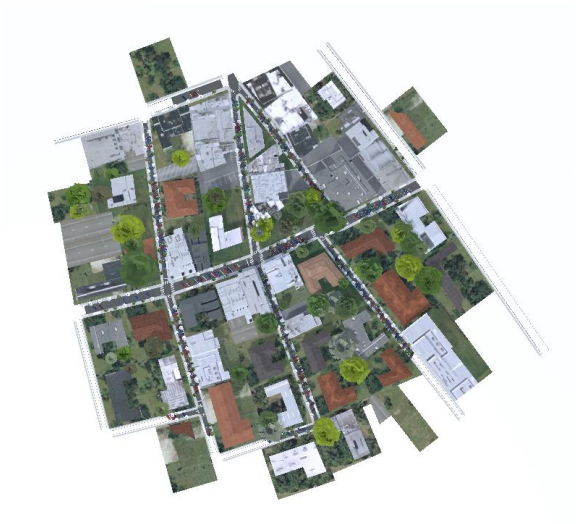

Ilustración 5.9: Fragmento de ciudad

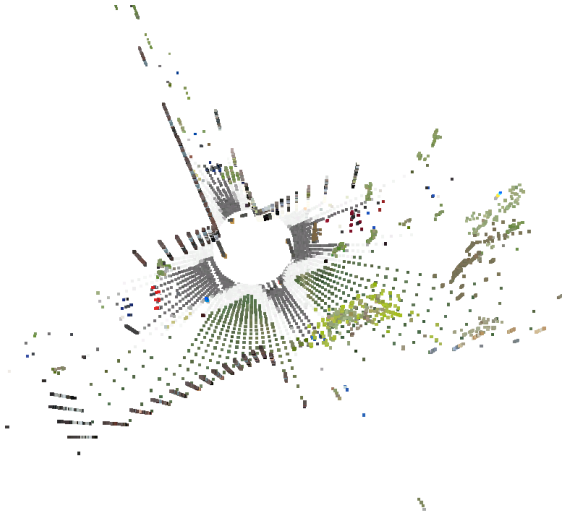

Ilustración 5.10: Previsualización de nube LiDAR

5.1.2.3 Obtención de nubes de puntos con LiDAR virtual

Hasta este punto hemos conseguido crear una ciudad, fragmentarla y etiquetarla. Ahora lo que se requiere es de la obtención de la nube de puntos con un LiDAR virtual que refleje o simule fielmente el funcionamiento de un sensor real para enviárselo a la red neuronal. Dicho proceso se puede resumir con el esquema de la Ilustración 5.11, que se trata de un fragmento del esquema mostrado en el apartado 3.12 (Visión general del proyecto).

Ilustración 5.11: Ubicación actual dentro del flujo del proyecto

Para esta tarea se ha utilizado un sensor sintético previamente citado [48]. Dicho sensor tiene un funcionamiento donde en primer lugar, se adjuntan etiquetas semánticas y materiales de una base de datos BRDF a objetos de escenas estáticas y procedurales. Luego, y haciendo referencia a la Ilustración 5.12, un sistema LiDAR virtual recrea la simulación con 1) especificaciones de escaneo, 2) desde una plataforma y 3) siguiendo una ruta o colocado en una ubicación estática. Dicha simulación se divide en dos etapas: generación del haz y solucionador de tiempo de

vuelo (ToF). En la última etapa, se adjuntan la intensidad y otros atributos relevantes a los puntos 3D. Para más información se recomienda revisar el artículo “Enhancing LiDAR point cloud generation with BRDF-based appearance modelling”.

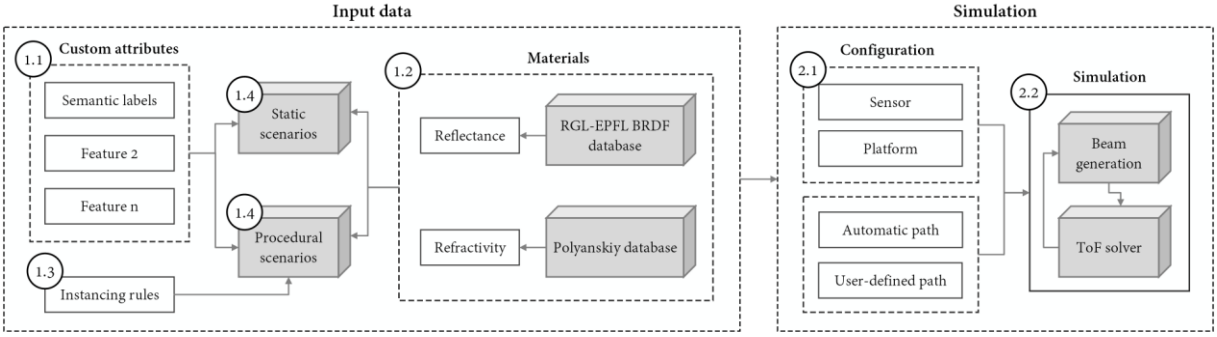

Ilustración 5.12: Funcionamiento del LiDAR virtual [48]
(Enhancing LiDAR point cloud generation with BRDF-based appearance modelling)

De esta manera se le pasaron a este sensor los fragmentos procedurales junto con XMLs que relacionaban las etiquetas con los objetos, y a partir de ello se obtuvieron las nubes de puntos como salida del sensor. Estas nubes en bruto y en formato PLY son el resultado de la tarea, estando listas para su tratamiento y posterior inserción dentro la red neuronal. Cabe destacar que el simulador LiDAR es configurable, por lo que se aprovechó para ajustarlo con la idea de que este tuviese el mismo comportamiento que el LiDAR usado durante la toma de nubes de puntos del conjunto de datos Toronto-3D.

5.2 Preprocesamiento y transformación de los datos

Se parte de la posesión de 2 conjuntos de datos, Toronto-3D y Synthetic Cloud 3D (reales y sintéticos respectivamente). El problema de estos datos es que no se pueden utilizar directamente por múltiples factores. Primeramente, algunos archivos son tan grandes como para que cualquier ordenador del laboratorio se quede sin memoria VRAM (Video Random Access Memory) durante el entrenamiento. El segundo de los inconvenientes es que los *datasets* se etiquetan con conjuntos de etiquetas diferentes, algo que obligará a adaptar el etiquetado de Synthetic Cloud 3D para que utilice el de Toronto-3D. En tercer lugar, ambos conjuntos de nubes nos dan las nubes en formatos distintos que además no son soportados, por lo que hay que traducirlos a un formato común que sí que soporte el proyecto, el formato NPY (formato binario estándar de NumPy). Sumado a esto está la diferencia entre la

cantidad de información que nos ofrecen ambos *datasets* y la información que realmente es necesaria. Posteriormente hay que tener en cuenta la posición de las nubes, es decir, si se encuentran centradas, pues de no ser así puede haber problemas con el proceso de aprendizaje y el truncado de posiciones muy alejadas. También debe considerarse si los conjuntos tienen el mismo sistema de coordenadas, algo que no sucede inicialmente porque en los datos de Toronto-3D se encuentran los ejes Y/Z invertidos frente a los datos de Synthetic Cloud 3D. A esta serie de problemas se les pueden sumar otros como por ejemplo la eliminación de puntos repetidos.

En vista de lo comentado, el preprocesamiento de los datos es algo necesario y se ha creado un programa en Python para esta preparación de los datos. A continuación, se describirán todas las acciones llevadas a cabo dentro del programa acompañadas de su correspondiente comando en caso de haberlo. Dicho programa se puede encontrar en la dirección:

"*RepSurf\segmentation\convert2npy\convert_all_to_npy.py*".

5.2.1 Integración y adaptación a la arquitectura

Para comenzar la integración de los datos, estos deben acomodarse a la arquitectura del proyecto RepSurf. Este proyecto usa formato NPY, pero nosotros tenemos datos en formatos PLY. Para hacer la lectura de los archivos de etiquetas se usa la librería pandas, mientras que para la lectura de archivos PLY se toma la librería plyfile. Haciendo uso de ellas, el código para la lectura del PLY, simplificado a lo esencial, es el siguiente para el LiDAR virtual (para el PLY de Toronto-3D habría cambios mínimos manejados con un condicional).

```
# Lee de forma personalizada el PLY pasado.
def leer_PLY(PLY_path):
    # Lectura del PLY y detección de la naturaleza del archivo
    PLY_data = PLYData.read(PLY_path)
    vertices = PLY_data['LiDAR'].data
    # Lectura de la posición tomando x, y, z del PLY
    x = vertices['x']
    y = vertices['y']
    z = vertices['z']
    # Lectura del color y de la clase
    semanticG_vertices['semanticGroup_red']
    semanticG_green = vertices['semanticGroup_green']
    semanticG_blue = vertices['semanticGroup_blue']
    # Lectura de la clase del punto del PLY
    semanticGroup = vertices['semanticGroup']
    return x, y, z, semanticG_red, semanticG_green, semanticG_blue, semanticG
```

Para la posterior conversión y guardado en formato NPY se ha utilizado la librería numpy. Para mayor facilidad de depuración, se devuelve el archivo en formato NPY y en TXT. En el código del programa las líneas responsables son las siguientes:

```
1. # Permite exportar un archivo a formato NPY dados los valores xyzrgbl
2. def exportar_NPY(out_path, x, y, z, r, g, b, labels):
3.     data =  np.column_stack((x, y, z, r, g, b, labels))
4.     np.save(out_path + 'NPY', data)
5.     np.saveTXT(out_path + 'TXT', data, fmt='%f %f %f %d %d %d %d', delimiter=' ')
```

Este proceso se hace automáticamente siempre, y dentro de él se traducen las etiquetas propias de Synthetic Cloud 3D al conjunto de Toronto-3D gracias a las tablas de conversión que se crearon para ese fin, explicadas en el capítulo 4 (Generador de fragmentos de ciudades). Cabe destacar que solo se puede convertir una etiqueta propia a etiqueta de un *dataset* concreto, pero no al revés. Esto es así porque para cada etiqueta propia existe una única correspondencia en el conjunto de etiquetas concreto (por ejemplo el de Toronto-3D), pero eso no sucede al contrario, en otras palabras, con la traducción se lleva a cabo una generalización de las etiquetas propias que son muy específicas. Esto se maneja con los siguientes argumentos:

- --labels_in: aquí se indica el conjunto de etiquetas de la nube de entrada (debe coincidir el nombre con el de la columna numérica de la tabla de conversión TABLA_CONVERSION_ETIQUETAS). Por ejemplo: Num_Estandar_LAS.
- --labels_out: aquí se indica el conjunto (numérico) de etiquetas de la nube de salida (debe coincidir el nombre con el de la columna numérica de la tabla de conversión TABLA_CONVERSION_ETIQUETAS). Por ejemplo: Num_Toronto_3D.

Aun habiendo arreglado esto, los datos tienen una forma de representación aún incorrecta. Concretamente los datos se deben centrar, algo conseguido reubicando el centroide de la ciudad, desarrollado y funcional mediante el argumento --center.

```
# Si se requiere de la nube centrada
if center:
    # Calculamos el centroide para centrar la nube
    centroide_x = int(np.mean(x))
    centroide_y = int(np.mean(y))
    centroide_z = int(np.mean(z))
    # Restamos el centroide a cada punto y asignamos el nuevo valor de los puntos
    x -= centroide_x
    y -= centroide_y
    z -= centroide_z
```

Igualmente, los datos deben pasar al mismo sistema de coordenadas, para lo cual se requiere de invertir los ejes Y/Z del conjunto de Toronto-3D con el comando --flip. Internamente, el código luce así:

```
# Invertimos ejes
if(flip_axis):
    data[:, [1, 2]] = data[:, [2, 1]]
```

5.2.2 Limpieza de los datos

En esta fase se eliminaron los archivos que se consideraron potenciales archivos problemáticos. Para ello, manualmente se repasó cada uno de los archivos para comprobar si contenían algún tipo de error, quitando archivos defectuosos que no incluían aceras o los vehículos no estaban correctamente etiquetados. Esta revisión supuso pasar de 107 nubes de puntos a 71 nubes, lo cual supone la eliminación de 36 archivos.

Debido al funcionamiento por barridos que tienen los sensores LiDAR, además es posible que haya puntos repetidos dentro de una misma nube de puntos. Esto solamente añade ruido al entrenamiento, por lo que se eliminaron los puntos repetidos de ambos conjuntos de datos, es decir, tanto dentro de las nubes de Toronto-3D, como los de Synthetic Cloud 3D. Ese proceso se hace automáticamente al ejecutar el programa "*convert_all_to_npy.py*" y básicamente tiene en cuenta la posición para la eliminación.

```
# Destruye los puntos repetidos
def borrar_repetidos(data):
    # Convertir a DataFrame
    df = pd.DataFrame(data, columns=['x', 'y', 'z', 'r', 'g', 'b', 'l'])
    # Eliminar duplicados basados en (x, y, z)
    df_unique = df.drop_duplicates(subset=['x', 'y', 'z'])
    # Convertir de nuevo a numpy array
    data_unique = df_unique.to_numpy()
    return data_unique
```

5.2.3 Reducción de la dimensionalidad

La información como el color, las normales o la intensidad finalmente se ha descartado. Más concretamente, y como se comenta en varios puntos de la memoria, la única característica que se ha tenido en cuenta a lo largo de este proyecto ha sido la posición junto con la información de la clase. En un principio se iba a respetar la organización del proyecto RepSurf, que permitía utilizar como características la posición y el color, sin embargo, por limitaciones del sensor LiDAR virtual, por causas de tiempo, para seguir alineados con otras investigaciones similares y por simplicidad, al final se decidió descartar el color de entre las características a usar. Esto significa que se tendrán en consideración los valores X, Y, Z y la clase.

No tener en cuenta el color supone tomar una de las 2 siguientes alternativas: reorganizar el proyecto RepSurf para que no lo utilice, o anularlo para que no tenga influencia en el entrenamiento. Lo más correcto sería sin duda reorganizar el proyecto RepSurf, no obstante, ello implicaría demasiado tiempo, algo que por el contexto de la investigación no se podría asumir. Es por esta razón que se tomó la segunda opción, anular el valor RGB (0,0,0). En efecto, esta alternativa puede introducir un mínimo de ruido, no obstante, se ha comprobado tras la experimentación que el impacto no es elevado. Esta eliminación del color se ha sobrellevado en el programa escrito en Python con el desarrollo de un argumento, --nullrgb, que hace dicha operación simplemente añadiéndolo a la línea de ejecución en la terminal. En código internamente se multiplica por 0 el color en caso de activarse la función.

Por otro lado, reducir el peso de los archivos individuales del *dataset* de Toronto-3D ha sido algo necesario e indispensable porque su espacio en memoria no es soportado. Abriendo el *dataset* se observa que hay algunos archivos que pesan más de 1GB, algo que se expande dentro de la fase del entrenamiento y que produce que la tarjeta gráfica del ordenador no sea capaz de entrenar debido a problemas de memoria y falta de VRAM. Existen varias alternativas para solucionar esto, pero se consideraron principalmente dos: tomar una muestra representativa de cada archivo, o subdividir el archivo en múltiples archivos más pequeños. Para la investigación se tomó la segunda opción pues, aunque ambas son viables, tomar una sola muestra que soportase el ordenador haría que se perdiese mucha información de interés para

el entrenamiento, por lo que en búsqueda de utilizar la máxima cantidad de información que el *dataset* de Toronto-3D nos brinda, se decidió dividir cada nube en 25 partes. De esta manera, aunque se pierdan conexiones entre puntos vecinos porque estos se reparten entre diferentes nubes, se reduce el peso individual de cada nube haciéndola más manejable. Esta subdivisión del *dataset* se hace con el argumento --subdivide X, donde X es el número de divisiones requeridas. Dicho argumento hace que la nube objetivo se subdivida aleatoriamente en X nubes disjuntas entre ellas. El código para la subdivisión luce como este.

```
1. # Subdivide aleatoriamente el archivo original
2. def subdividir(data, num_partes, out_path):
3.     np.random.shuffle(data)
4.     split_data = np.array_split(data, num_partes)
```

Una vez reducido el problema asociado al coste en memoria nos encontramos nuevamente frente al problema temporal que acompaña al proyecto desde que inició. Entrenar con los 25 archivos de cada nube de Toronto-3D es algo que requiere bastante tiempo y es por ello que buscando ahorrar algo de tiempo se tomaron 20 de esos archivos, algo que, aunque no sea mucha diferencia, sí que permitía ajustarse al tiempo estimado necesario.

5.2.4 División de los *datasets*

Llegados a este punto se tiene un conjunto de archivos preprocesados y no en bruto tal y como se recibieron del sensor. Ahora lo que se hará es repartir los archivos para llevar a cabo el entrenamiento, la validación dentro del mismo y las pruebas posteriores. En otras palabras, se requerirá de definir 3 subconjuntos:

- *Training set*: es el subconjunto más grande, ya que el modelo necesita muchos datos para aprender patrones. Éste se utiliza para ajustar los parámetros del modelo, permitiendo al modelo configurar sus pesos internos para minimizar el error.
- *Validation set*: se emplea para afinar el modelo y ajustar hiperparámetros, permitiendo evaluar el rendimiento del modelo mientras se entrena. Sirve para detectar si el modelo está aprendiendo correctamente.

- *Test set*: se usa después de haber entrenado y ajustado completamente el modelo. Con él se evalúa la capacidad de generalización del modelo final a datos que no ha visto antes.

Conociendo esto y basándonos en configuraciones frecuentes, se hará un *hold-out*, repartiendo las instancias como se ve en la Ilustración 5.13 de manera que, aproximadamente, se destinen el 80% de ellas al proceso de entrenamiento y validación, mientras que el 20% restante se utilicen para la etapa de pruebas. Además, de ese 80% orientado al entrenamiento, un 20% se utilizará para la validación. En otras palabras y de forma resumida, dado que se necesitarán para los entrenamientos un conjunto real, un conjunto sintético y un conjunto mixto, el reparto será el siguiente:

- **Conjunto de datos reales**: de un total de 80 instancias que tiene Toronto-3D, 40 se dedican a entrenamiento (provenientes de L001 y de L003), 20 a validación (proveniente de L004) y las otras 20 para pruebas (proveniente de L002). Cabe destacar que L001 y L003 contienen mayor cantidad de puntos que L002 y L004. Pese a no ajustarse por completo a los porcentajes propuestos, esta división es la recomendada por el artículo de Toronto-3D.
- **Conjunto de datos sintéticos**: de un total de 71 instancias que tiene Synthetic Cloud 3D, 45 se dedican a entrenamiento, 11 a validación y las otras 15 para pruebas.
- **Conjunto de datos mixtos**: de un total de 90 instancias que tiene el conjunto mixto, 56 se dedican a entrenamiento y 14 a validación (ambos con 20% de Toronto-3D y 80% de Synthetic Cloud 3D), y las otras 20 para pruebas (100% de Toronto-3D).

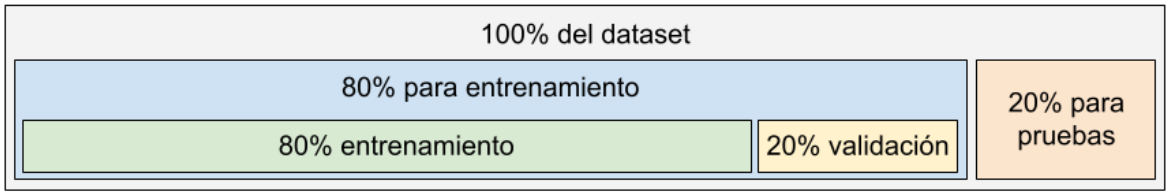

Ilustración 5.13: Distribución de los datos

Es importante comentar que se también se valoró la posibilidad de hacer *k-fold cross validation*, sin embargo, una validación cruzada conllevaría mucho más tiempo para hacerse y no daría tiempo a finalizar la fase de experimentación planteada. Para futuras actualizaciones dentro de esta investigación se recomienda que, si hay tiempo suficiente, se realice una validación cruzada tomando como K un valor típico como podría ser 5 o 10, lo cual permitirá obtener resultados moderadamente más fiables.

5.3 Minería de datos

Aquí buscamos extraer patrones útiles a partir de las nubes de puntos mediante el entrenamiento supervisado del modelo de la red con el objetivo de ser capaces de segmentar una nube de puntos para diferenciar los objetos que constituyen el fragmento de ciudad. El resultado obtenido de la minería permite clasificar cada punto asignándole una etiqueta de las nombradas anteriormente.

Ya adelantado en múltiples ocasiones, para conseguir esto se tomará el proyecto RepSurf (disponible en GitHub) y se adaptará de manera que acepte cualquier conjunto de datos. Cabe mencionar que es una fase que ha sido muy complicada de realizar, destacando la dificultad del arranque del proyecto y remarcando lo costosa que ha sido computacionalmente y temporalmente hablando, a lo que se le suman además limitaciones asociadas al hardware.

5.3.1 Adaptación del proyecto RepSurf

Encontrar un proyecto que sea compatible con una versión de CUDA aceptada por la gráfica fue una ardua tarea que requirió de investigación y muchos intentos. A esto se le suma la dificultad de las dependencias en Python, convirtiendo el simple arranque de un proyecto en una actividad alimentada por una gran incertidumbre. La búsqueda de un proyecto de redes neuronales que fuera de utilidad tuvo como punto partida los proyectos de redes que se probaron en la página de Semantic3D, sin embargo, al ser proyectos con varios años lo cierto es que no fue posible iniciar ninguno con éxito por sistemas operativos, versiones de bibliotecas, incompatibilidades con la tarjeta gráfica y CUDA, compiladores, etc. Fue algo que se repitió durante varias semanas hasta que se consiguió, con dificultades, arrancar desde el subsistema de Linux el proyecto RepSurf (Surface Representation for Point

Clouds), un proyecto más moderno que contiene 3 implementaciones hechas en PyTorch: red PointNet++, red Point Transformer y Umbrella RepSurf, un método basado en la red PointNet++ (SSG) al que se le aplica el módulo Umbrella RepSurf creado en esa investigación para mejorar la representación de superficies.

El proyecto RepSurf es un trabajo excelente, no obstante, es importante decir que la falta de documentación hizo que en ocasiones se requiriera prácticamente de ingeniería inversa. Aunque sin duda si hay algo que debía ser modificado para el actual trabajo de investigación era el tema de la parametrización relacionada con los conjuntos de datos. En efecto, el proyecto estaba preparado exclusivamente desde el código para trabajar con el conjunto de datos S3DIS, por lo que se adaptó de manera que permitiese ejecutarse entero como originalmente era, y para ejecutarse con un conjunto de datos personalizado pasando un archivo JSON con las etiquetas usadas. Esto supuso la creación de una nueva estructura de archivos paralela a la original dentro del proyecto donde se encontraban las adaptaciones de los códigos para poder ejecutar con conjuntos de datos personalizados.

Una vez adaptado el proyecto ya se pueden ejecutar los comandos de entrenamiento con conjuntos de datos personalizados. Más adelante nos centraremos en los comandos, pero en cuanto a las carpetas, las más importantes a tener en cuenta son:

- La carpeta con los archivos de comandos de entramiento y pruebas: "*RepSurf/segmentation/scripts/custom*"
- La carpeta destinada a los archivos de entrenamiento y validación: "*RepSurf/segmentation/data/CUSTOM/trainval_fullarea*".
- La carpeta destinada a los archivos de prueba: "*RepSurf/segmentation/data/CUSTOM/data/CUSTOM/testval_fullarea*".
- La carpeta con todos los logs y el modelo entrenado: "*RepSurf/segmentation/log/PointAnalysis/log*".

Como nota adicional, es importante mencionar que este proyecto está preparado para hacer validación cruzada pese a que en nuestro caso no se haya hecho por temas de tiempo. Ello es algo que se maneja con el nombre del archivo. Todos los archivos de la carpeta comienzan por “Area_X” donde X es un número que representa un subconjunto de los datos de entrenamiento, permitiendo fácilmente destinar grupos concretos de datos para la validación. Por ejemplo, si se desea hacer la validación con los archivos del área 1, es tan simple como darles de nombre Area_1+nombre_original, y dejar para el resto del entrenamiento los otros archivos, cuyo nombre podría ser, por ejemplo, Area_2+nombre_original (valdría cualquier número menos el 1).

5.3.1.1 Problemas de memoria: swapping

Pese a estar adaptado el proyecto a nuestra investigación, este seguía sin funcionar correctamente. Dando un poco de contexto, este trabajo se comenzó a elaborar en el ordenador del laboratorio con la tarjeta gráfica 3060Ti, la cual es bastante potente, pero mostraba algunas limitaciones con el proyecto RepSurf que fueron descubiertas durante unos entrenamientos de prueba con datos de ejemplo. Fue en ese instante donde, aunque no hubiera ningún fallo durante la ejecución, ésta no era capaz de terminar y además la memoria de vídeo (VRAM) estaba llena.

Para entender el problema hace falta entender su origen, y es que, con frecuencia, cuando la ejecución de algún programa llena la memoria RAM, el sistema operativo utiliza parte de la memoria del disco duro para gestionar el exceso de trabajo. Este concepto es llamado memoria virtual y sucede de manera similar entre la VRAM y la memoria RAM: si hay falta de VRAM, se transfieren datos de ésta a la RAM para soportar el trabajo.

Lo que está ocurriendo en nuestro caso es que hay VRAM insuficiente y PyTorch intenta guardar el exceso en RAM. Sin embargo, para ejecutar un proceso en la tarjeta gráfica éste debe estar cargado en VRAM. Si estamos guardando datos de VRAM en RAM, para usarlos antes se deben transferir. A esto se le llama *swapping* y ralentiza el entrenamiento debido a la latencia de las transferencias de datos, las diferencias en la velocidad de acceso a memoria y la carga añadida asociada a la gestión de memoria.

Esto se puede solventar en cierta medida con la reducción del tamaño de los lotes y la carga de datos, posible con los parámetros de PyTorch: batch_size, batch_size_val y workers. Sin embargo, eso no fue suficiente y obligó a cambiar de equipo durante el proceso en pro de otro con una gráfica especializada para el entrenamiento, una A4500. Esto fue lo que solucionó el problema de memoria que se estaba dando hasta el momento y lo que permitió continuar con la investigación.

5.3.2 Redes ofrecidas por el proyecto RepSurf

Una vez adaptado el proyecto se describirá brevemente el funcionamiento de las redes que incorpora RepSurf: PointNet++, Point Transformer y Umbrella RepSurf. Estas explicaciones, complementarias a las presentadas en el capítulo 2.3 (Redes neuronales artificiales para nubes de puntos), serán acompañadas de sus respectivos comandos para llevar a cabo tanto el entrenamiento como la evaluación.

5.3.2.1 Red PointNet++

PointNet ha sido la base de muchos de los métodos de visión 3D con nubes de puntos, siendo capaz de procesar nubes de puntos sin la necesidad de convertirlas a estructuras 3D como los vóxeles. Sin embargo, en su primera versión, pese al correcto funcionamiento para capturar las características globales de las nubes de puntos, presentaba limitaciones para detectar características locales. Esto supone que:

- Las características aprendidas en datos densos pueden no generalizarse a regiones muestreadas de forma más dispersa.
- Los modelos entrenados para nubes de puntos dispersas pueden no reconocer estructuras locales de gran detalle.

Para solventar este problema surge la red PointNet++, que procesa las nubes de puntos a diferentes escalas de manera jerárquica. Para ello la red parte de un muestreo inicial de puntos representantes y uniformemente espaciados en la nube para posteriormente agruparlos según vecindades y aplicar a estas la clásica PointNet. De esta manera, gracias al muestreo en cada nivel de la red se va reduciendo la cantidad de puntos, y a través de abstracciones del conjunto se

aumentan las características extraídas de cada grupo. Esto conlleva que en los niveles más bajos se capturan detalles locales, mientras que los niveles más altos capturan características globales.

Para usar esta red dentro del proyecto, teniendo correctamente organizado el conjunto de datos, habría que ejecutar desde la terminal los siguientes comandos para entrenamiento y evaluación respectivamente:

sh RepSurf/segmentation/scripts/custom/train_pointnet2.sh
sh RepSurf/segmentation/scripts/custom/test_pointnet2.sh

5.3.2.2 Red Point Transformer

En boca de todos desde hace un tiempo está la arquitectura Transformer, presentada en 2017. Muchas fueron las adaptaciones de esta arquitectura, inicialmente planteada para el campo de PLN, pero de entre todas ellas destacaremos la red Point Transformer, la cual aplica la idea de usar mecanismos de atención para trabajar con nubes de puntos. Así pues, esta red neuronal extiende el concepto de atención de las arquitecturas Transformers, permitiendo a cada punto de la nube "atender" a otros puntos en su vecindad para determinar su importancia relativa. Esto ayuda a que la red capture con gran eficacia las relaciones locales entre los puntos, centrándose en las aquellas interacciones más relevantes.

Para usar esta red, teniendo correctamente organizado el conjunto de datos, habría que ejecutar desde la terminal los siguientes comandos para entrenamiento y evaluación respectivamente:

sh RepSurf/segmentation/scripts/custom/train_Point Transformer.sh
sh RepSurf/segmentation/scripts/custom/test_Point Transformer.sh

5.3.2.3 Módulo Umbrella RepSurf

Dentro de este proyecto constantemente se habla de Umbrella RepSurf y lo cierto es que no se trata de una red neuronal, sino que más bien es un módulo que actúa en conjunto con la red PointNet++ en su versión SSG (Single-Scale Grouping). Aunque en este proyecto solo se aplica sobre PointNet++, la integración de RepSurf

se puede hacer con otras redes neuronales aplicadas a nubes de puntos permitiendo, a cambio de un pequeño coste computacional, captar mejor las características y con ello mejorar la eficacia en tareas como clasificación y segmentación.

En efecto, el proyecto RepSurf nace con el objetivo de proporcionar una ayuda adicional a las redes neuronales para incrementar su rendimiento gracias a la mejora de la representación de superficies que incorpora a estas como información extra. De los 2 módulos que se desarrollaron dentro del proyecto, en la actual investigación se probará Umbrella RepSurf, la segunda variante que se fundamenta en la curvatura de "sombrilla" y amplía el campo de percepción al construir superficies a partir de los K puntos vecinos más cercanos. Dicha curvatura se obtiene mediante el cálculo de la tangente de la superficie reconstruida, información que usa para obtener una representación más detallada de la geometría local. Esto supone una gran ayuda incluso cuando los puntos se encuentran organizados irregularmente sobre el espacio. Con la Ilustración 5.14, el artículo representaba la idea del modelo Triangular RepSurf y Umbrella RepSurf como representaciones de geometría local para el proceso de aprendizaje automático.

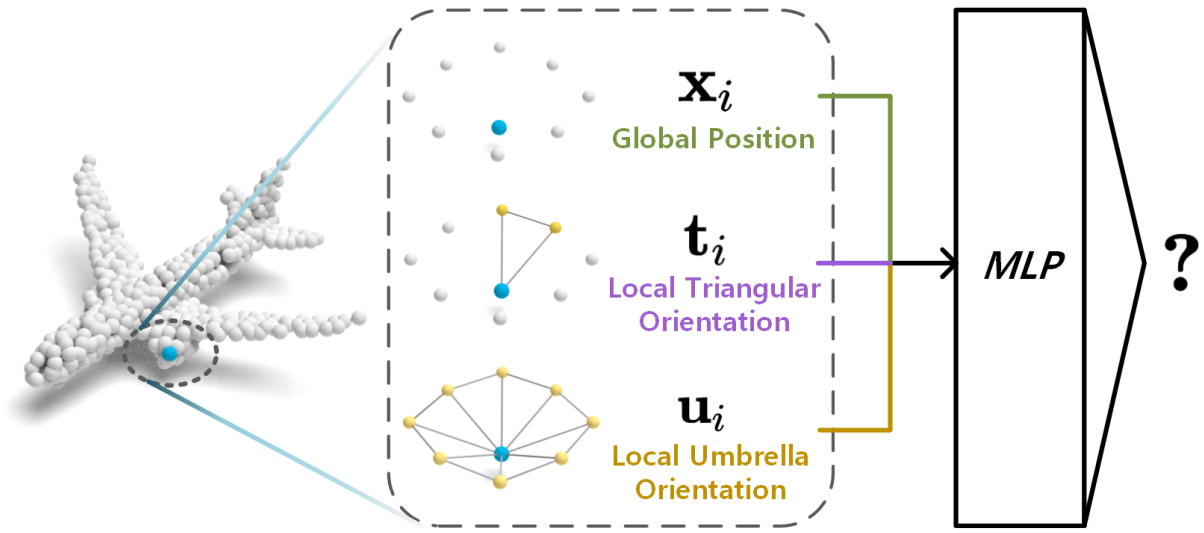

Ilustración 5.14: Concepto de RepSurf

Para usar este módulo sobre la red PointNet++ (la adaptada por defecto), teniendo correctamente organizado el conjunto de datos, habría que ejecutar desde la terminal los siguientes comandos para entrenamiento y evaluación respectivamente:

sh RepSurf/segmentation/scripts/custom/train_RepSurf_umb.sh

sh RepSurf/segmentation/scripts/custom/test_RepSurf_umb.sh

5.3.3 Métricas utilizadas para el cálculo del error

Una vez conocidas las redes y los comandos que ejecutan sus respectivos entrenamientos y pruebas, el siguiente paso es comprender qué sucede dentro de los periodos de ejecución. Esta información de interés viene incluida dentro de archivos *log* como los de la captura inferior.

```
[2024-07-24 20:24:03,757 INFO train_custom.py line 133 429506] Total Number of Parameters: 0.967 M
[2024-07-24 20:24:10,362 INFO train_custom.py line 163 429506] Totally 40 samples in train set.
[2024-07-24 20:24:10,362 INFO train_custom.py line 164 429506] Totally 20 samples in val set.
[2024-07-24 20:26:50,698 INFO train_custom.py line 327 429506] Epoch: [1/50][250/400] Batch 0.968 (0.641) Remain 03:31:06 Loss 0.7002 Accuracy 80.80
[2024-07-24 20:28:26,718 INFO train_custom.py line 346 429506] Train result at epoch [1/50]: mIoU / mAcc / OA 23.91 / 29.90 / 77.90
[2024-07-24 20:31:08,100 INFO train_custom.py line 327 429506] Epoch: [2/50][250/400] Batch 0.568 (0.646) Remain 03:28:10 Loss 0.3524 Accuracy 90.86
[2024-07-24 20:32:44,891 INFO train_custom.py line 346 429506] Train result at epoch [2/50]: mIoU / mAcc / OA 39.61 / 46.54 / 87.92
[2024-07-24 20:35:27,064 INFO train_custom.py line 327 429506] Epoch: [3/50][250/400] Batch 0.566 (0.649) Remain 03:24:52 Loss 0.3521 Accuracy 89.23
[2024-07-24 20:37:01,590 INFO train_custom.py line 346 429506] Train result at epoch [3/50]: mIoU / mAcc / OA 48.17 / 54.89 / 91.26
[2024-07-24 20:39:43,179 INFO train_custom.py line 327 429506] Epoch: [4/50][250/400] Batch 0.826 (0.646) Remain 03:19:49 Loss 0.2564 Accuracy 92.80
[2024-07-24 20:41:19,526 INFO train_custom.py line 346 429506] Train result at epoch [4/50]: mIoU / mAcc / OA 53.31 / 59.92 / 92.27

                                                [...Otras épocas...]

[2024-07-25 02:22:18,668 INFO train_custom.py line 327 429506] Epoch: [49/50][250/400] Batch 0.786 (0.666) Remain 00:06:06 Loss 0.0763 Accuracy 97.32
[2024-07-25 02:23:55,153 INFO train_custom.py line 346 429506] Train result at epoch [49/50]: mIoU / mAcc / OA 86.18 / 88.35 / 97.48
[2024-07-25 02:23:55,159 INFO train_custom.py line 352 429506] >>>>>>>>>>>>>>>> Start Evaluation >>>>>>>>>>>>>>>>
[2024-07-25 02:32:08,412 INFO train_custom.py line 402 429506] Val result: mIoU / mAcc / OA 55.79 / 66.70 / 90.79
[2024-07-25 02:32:08,413 INFO train_custom.py line 403 429506] Val loss: 0.4152
[2024-07-25 02:32:08,413 INFO train_custom.py line 406 429506] Class_0 Result: IoU / Acc 90.80/99.65
[2024-07-25 02:32:08,413 INFO train_custom.py line 406 429506] Class_1 Result: IoU / Acc 0.08/0.08
[2024-07-25 02:32:08,413 INFO train_custom.py line 406 429506] Class_2 Result: IoU / Acc 81.99/88.34
[2024-07-25 02:32:08,413 INFO train_custom.py line 406 429506] Class_3 Result: IoU / Acc 70.69/81.64
[2024-07-25 02:32:08,413 INFO train_custom.py line 406 429506] Class_4 Result: IoU / Acc 61.51/64.31
[2024-07-25 02:32:08,414 INFO train_custom.py line 406 429506] Class_5 Result: IoU / Acc 70.61/81.95
[2024-07-25 02:32:08,414 INFO train_custom.py line 406 429506] Class_6 Result: IoU / Acc 62.21/74.17
[2024-07-25 02:32:08,414 INFO train_custom.py line 406 429506] Class_7 Result: IoU / Acc 8.41/43.43
[2024-07-25 02:32:08,414 INFO train_custom.py line 407 429506] <<<<<<<<<<<<<<<< End Evaluation <<<<<<<<<<<<<<<<
[2024-07-25 02:34:51,707 INFO train_custom.py line 327 429506] Epoch: [50/50][250/400] Batch 0.563 (0.653) Remain 00:01:37 Loss 0.0783 Accuracy 96.92
[2024-07-25 02:36:31,295 INFO train_custom.py line 346 429506] Train result at epoch [50/50]: mIoU / mAcc / OA 86.20 / 88.34 / 97.57
[2024-07-25 02:36:31,300 INFO train_custom.py line 352 429506] >>>>>>>>>>>>>>>> Start Evaluation >>>>>>>>>>>>>>>>
[2024-07-25 02:44:44,507 INFO train_custom.py line 402 429506] Val result: mIoU / mAcc / OA 56.06 / 67.40 / 90.80
[2024-07-25 02:44:44,508 INFO train_custom.py line 403 429506] Val loss: 0.4028
[2024-07-25 02:44:44,508 INFO train_custom.py line 406 429506] Class_0 Result: IoU / Acc 90.90/99.62
[2024-07-25 02:44:44,508 INFO train_custom.py line 406 429506] Class_1 Result: IoU / Acc 0.05/0.05
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_2 Result: IoU / Acc 81.46/85.82
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_3 Result: IoU / Acc 71.67/88.34
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_4 Result: IoU / Acc 61.35/63.63
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_5 Result: IoU / Acc 70.72/79.91
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_6 Result: IoU / Acc 62.60/74.20
[2024-07-25 02:44:44,509 INFO train_custom.py line 406 429506] Class_7 Result: IoU / Acc 9.68/47.65
[2024-07-25 02:44:44,510 INFO train_custom.py line 407 429506] <<<<<<<<<<<<<<<< End Evaluation <<<<<<<<<<<<<<<<
[2024-07-25 02:44:44,513 INFO train_custom.py line 258 429506] ==>Training done!
Best Iou: 56.58
```

Ilustración 5.15: Output simplificado

Como se aprecia, para entender correctamente la información de este tipo de archivos es necesario conocer la teoría que respaldan las métricas. A continuación, se comentarán para qué sirven las métricas que se utilizan a lo largo de la etapa de experimentación: Loss, Acc (Accuracy), mAcc (mean Acc), IoU (Intersection over Union), mIoU (mean IoU) y OA (Overall Accuracy). Para esto, antes definiremos las siguientes variables, utilizadas con frecuencia en el ámbito del aprendizaje automático:

- TP = verdadero positivo (True Positive)
- TN = verdadero negativo (True Negative)
- FP = falso positivo (False Positive)
- FN = falso negativo (False Negative)

5.3.3.1 Valor de pérdida (Loss)

Esta medida, obtenida de la función de error, es utilizada en el entrenamiento de modelos de ML para cuantificar el rendimiento y la precisión de un modelo de machine learning teniendo en cuenta el etiquetado de los datos. Es crucial para la optimización de los modelos y actúa como guía dentro del proceso de entrenamiento de manera que a lo largo del mismo se trata de reducir para aproximarlo lo máximo posible a cero.

5.3.3.2 Métrica Acc (Accuracy)

La métrica Acc se usa para obtener la precisión del modelo asociada a cada clase existente. Ella valora cuántas veces el modelo acertó tanto al clasificar correctamente como al no clasificar incorrectamente los puntos. La expresión matemática que nos permite calcularla se muestra a continuación.

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

En otras palabras:

- El total de predicciones correctas de una clase se expresa como: TP + TN
- El total de todas las predicciones de una clase se expresa como: TP + TN + FP + FN

Otra métrica relacionada con Acc es mAcc, que representa la media de la Acc de todas las clases.

5.3.3.3 Métrica IoU (Intersection over Union)

La métrica IoU mide la intersección sobre la unión de las predicciones y las etiquetas reales, es decir, la precisión por clase. Es útil para medir qué tan bien el modelo ha segmentado o clasificado una clase específica en la nube de puntos y de alguna manera está directamente asociada a la correcta detección de la forma de los objetos. Matemáticamente se vería así:

$$IoU = \frac{TP}{TP + FN + FP} \tag{5.2}$$

En otras palabras:

- El total de puntos de una clase correctamente detectados como positivos (intersección) se expresa como: TP
- El total de puntos que podrían pertenecer a la clase de estudio (unión) se expresa como: TP + FP + FN

De la mano de la IoU está mIoU, métrica que representa la media de la IoU de todas las clases.

5.3.3.4 Métrica OA (Overall Accuracy)

La métrica OA calcula el porcentaje de puntos clasificados correctamente entre todos los puntos de la nube. En términos matemáticos, sea C el número de clases total, la fórmula sería la siguiente:

$$OA = \frac{\sum_{i=1}^{c} TP_i}{TP + TN + FP + FN} \tag{5.3}$$

En otras palabras:

- El total de puntos correctamente detectados independientemente de la clase se expresa como: $\sum_{i=1}^{c} TP_i$

- El total de todas las predicciones se expresa como: TP + TN + FP + FN

Con esta métrica se puede obtener la exactitud general del modelo midiendo los puntos correctamente etiquetados frente al total de puntos. La diferencia entre esta métrica y la métrica Acc es que ahora se evalúan solo los puntos correctamente etiquetados en general, mientras que la Acc distinguía entre clases. Esto implica que cuando hablamos de la mAcc, media de Acc entre las clases, se dé más importancia a aquellas instancias pertenecientes a clases minoritarias para la ponderación, mientras que en OA al tener cada instancia la misma ponderación, estas instancias de clases minoritarias quedan algo más ocluidas por las otras clases.

5.3.4 Parámetros y aumentación

Al igual que las métricas, es importante conocer qué parámetros se pueden configurar, así como conocer el término de aumentación. Comenzando por los parámetros, entre los más interesantes destacamos:

- Optimizador: algoritmo utilizado para actualizar los pesos del modelo con el fin de minimizar la función de pérdida durante el entrenamiento. En el caso de los experimentos que se harán, se tomará el optimizador Adam.
- Iteraciones totales: es el número total de iteraciones en un ciclo de entrenamiento para entrenar un modelo de aprendizaje automático. Estas iteraciones configuran 2 fases: una de entrenamiento donde se realizan actualizaciones de los pesos basadas en los datos de entrenamiento, y otra de validación, donde se mide la capacidad del modelo para generalizar los datos no vistos.
- Iteraciones mínimas: número de épocas que deben suceder antes de que se lleven a cabo validaciones dentro del entrenamiento. Hasta que no se alcanza este momento, el modelo aprende de los datos de entrenamiento pero no se estudia la capacidad que este tiene para generalizar. Aunque este proceso se podría hacer desde la primera *epoch*, lo común es que no sea así por cuestiones de tiempo.

- Iteraciones de decadencia: número de iteraciones tras las cuales se reduce la tasa de aprendizaje para permitir un ajuste más fino a medida que el modelo se acerca a la convergencia.
- Tasa de aprendizaje: afecta a la velocidad a la que el algoritmo alcanza las ponderaciones óptimas, es decir, converge.
- Tasa de decadencia: proporción en la que la tasa de aprendizaje se reduce a lo largo del tiempo para estabilizar el entrenamiento y mejorar la generalización.
- Tasa de decadencia de pesos: técnica que penaliza pesos grandes en el modelo para evitar el sobreajuste, lo que promueve modelos más simples y generalizables.

Sumado a esto, cabe destacar que el proyecto permite aumentación dinámica. La aumentación de datos se define como el proceso de generación artificial de nuevos datos, a partir de los datos ya existentes, para el entrenamiento de modelos de ML. Esto es beneficioso para el entrenamiento del modelo porque permite mejorar su capacidad de generalizar. Aunque la aumentación se puede hacer durante el procesado de los datos, la aumentación dinámica es una práctica habitual que además se da aquí porque permite optimizar el almacenamiento en disco al no requerir de guardar versiones aumentadas de los datos. Concretamente, dentro de este trabajo se permiten varias formas de aumentación, pero como en este proyecto el color no se tendrá en cuenta, se describirán los tipos de aumentación relacionados con la posición de los puntos:

- Aumentación por escalado: mediante el argumento --aug_scale se crean nuevos datos a partir de escalados a las nubes de puntos.
- Aumentación por rotación: mediante el argumento --aug_rotate se crean nuevos datos a partir de rotaciones, en los ejes especificados.
- Aumentación por translación: mediante los argumentos --aug_shift y --aug_jitter se crean nuevos datos mediante el intercambio de puntos vecinos, y mediante el desplazamiento de puntos respectivamente.

- Aumentación por volteo: mediante el argumento --aug_flip se crean nuevos datos mediante el reflejo de las nubes de forma horizontal o vertical.

5.3.5 Optimización de hiperparámetros

Para configurar estos parámetros presentados hay que recordar que este trabajo tiene limitaciones de tiempo muy importantes que han marcado su ejecución. Dada la situación y teniendo en cuenta la duración de cada entrenamiento, es clave decir que hay que ajustar con cuidado entre otros parámetros aquellos relacionados con las épocas y la aumentación, pues son los que repercuten de forma directa al tiempo de ejecución.

El proyecto en sí tenía configuraciones de ejemplo que no terminaban de ajustarse a nuestra investigación, por lo que se hicieron pruebas con las redes y los datos que serían utilizados finalmente durante los experimentos. A continuación, se van a mostrar resultados de dichas pruebas con los cuales se tratará de elegir el valor idóneo para el número de las iteraciones. Los siguientes gráficos, Ilustración 5.16 y Ilustración 5.17, muestran el comportamiento de una de las redes, PointNet++, para el entrenamiento con nubes de puntos de ciudades reales:

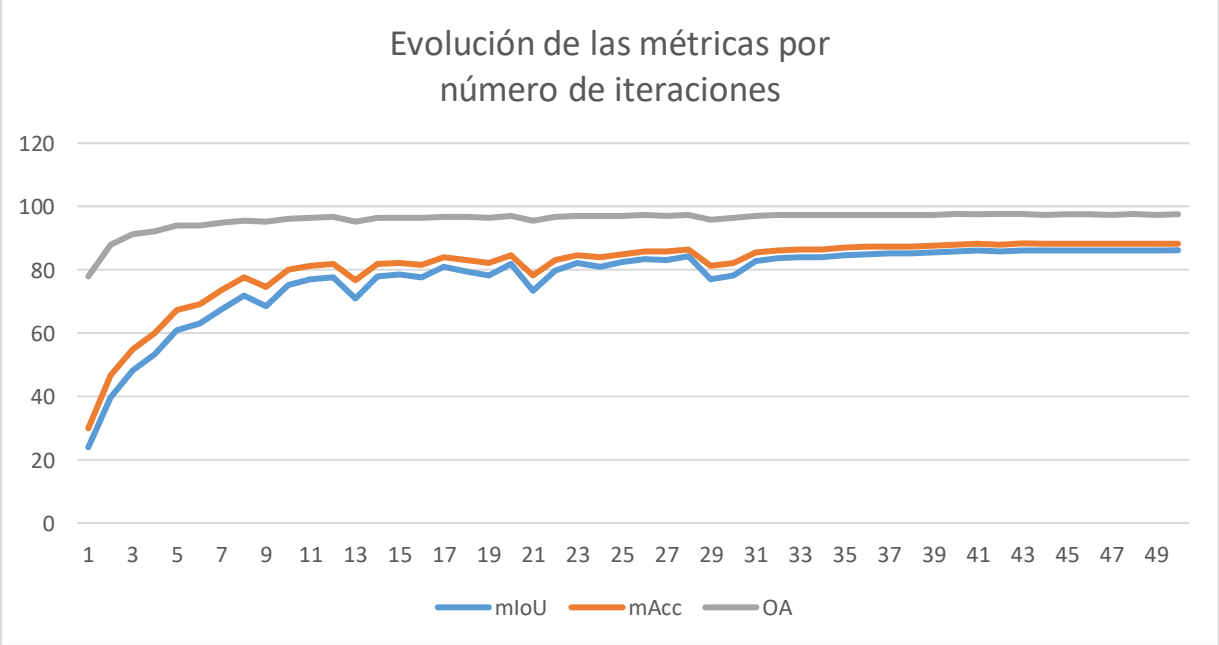

Ilustración 5.16: Evolución de métricas por épocas

Como puede observarse en la Ilustración 5.16, los resultados de las métricas (media de IoU, media de Acc y el OA) para este caso concreto dejan de mejorar aproximadamente sobre la época 40. Aunque la gráfica muestra el caso particular de la red PointNet++, el comportamiento se repite con el resto de redes de manera similar.

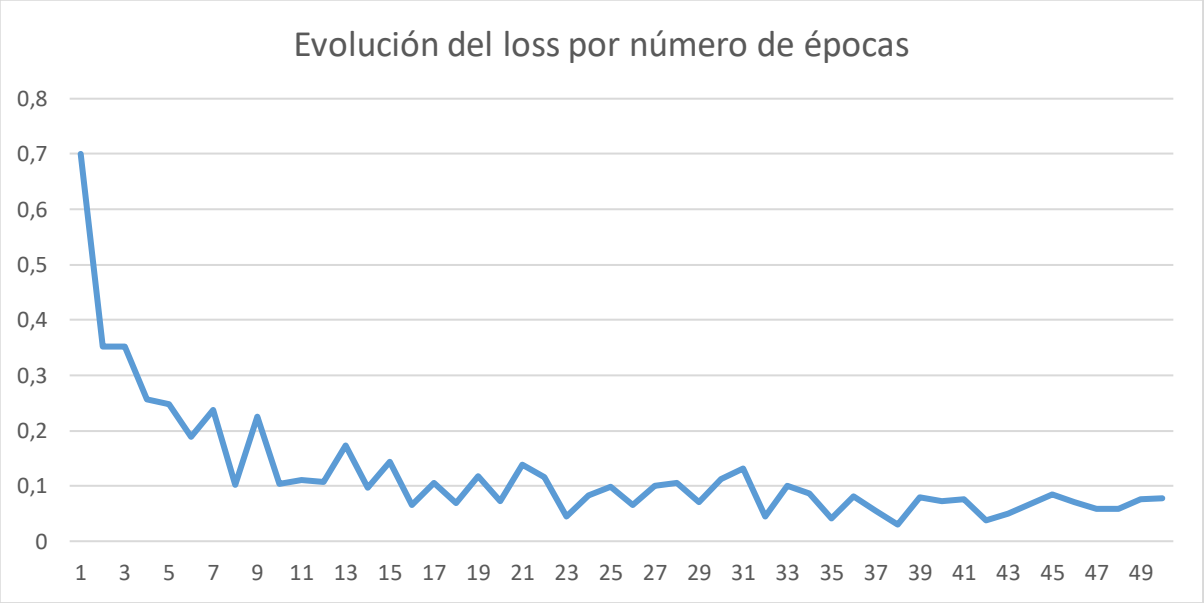

Ilustración 5.17: Evolución del valor de pérdida por épocas

Como puede verse en la Ilustración 5.17, sucede lo mismo con el valor de la función de pérdida o valor *loss*. Vemos que este valor inicialmente disminuye muy rápidamente, lo cual es algo beneficioso, pero que a partir de las 20 iteraciones apenas mejora, obteniéndose el mejor valor en la época 38.

Los parámetros de ejemplo estaban ajustados con un total de 100 épocas, sin embargo, a raíz de estas pruebas se llegó a la conclusión de que las redes convergen mucho antes. Es por esta razón que se marcó 50 como total de épocas, un valor que aseguraba en las pruebas realizadas que se encontrarían buenos resultados dando algo de margen adicional. Cabe destacar que, en este tipo de algoritmos, exceder el número de épocas necesarias puede ocasionar el sesgo del modelo y la reducción de la generalización del mismo. Posteriormente, en conocimiento del número total de épocas, se regularon también los valores para las épocas mínimas (30) y las épocas de decadencia (30→40).

En cuanto al tema de la aumentación podemos decir que es una práctica recomendada que en concreto dentro de este proyecto ha tenido cierto impacto, mejorando los resultados de pruebas preliminares hechas con los datos de ejemplo que éste traía (S3DIS). Pese la potencial mejora de los resultados, hay que tener en cuenta el aumento del coste computacional por esta aumentación dinámica. Es por ello que para evitar excesos en uso de VRAM y en tiempo, se decidió utilizar únicamente un tipo de aumentación, la aumentación por escala. Efectivamente, esta aumentación era la que mejores resultados ofrecía durante el periodo de pruebas a pequeña escala con un factor de 0,1.

Por consiguiente, se resumirá en la Tabla 5.1 toda la información asociada a la configuración de todos los entrenamientos llevados a cabo a lo largo de este proyecto, incluyendo los parámetros comentados y configuraciones recomendadas para el uso de las redes de este proyecto:

Parámetro	Valor
Optimizer (optimizer)	AdamW
Total epochs (epoch)	50
Minimum epochs for validation (min_val)	30
Learning decay epochs (lr_decay_epochs)	30→40
Learning rate (learning_rate)	0,006
Learning decay rate (lr_decay)	0,1
Weight decay (weight_decay)	0,01
Scale factor (aug_scale)	0,1

Tabla 5.1: Configuración de todos los entrenamientos

5.4 Evaluación e interpretación

Llegados a este punto, ya se entrenaron todos los modelos con los datos reales, los datos sintéticos, y los datos mixtos. Con esto nos adentramos en el corazón del proyecto, un total de 4 experimentos que tratarán de estudiar la validez de las hipótesis de inicio. Para cada uno se repasará cuál es su finalidad, su procedimiento y sus resultados, con el fin de concluir con un análisis de los patrones detectados y el resultado de la validación de la hipótesis correspondiente.

5.4.1 Experimento 1: Entrenamiento y testeo con datos reales

Este primer experimento tratará de entrenar y probar el modelo con datos reales tomados del conjunto de Toronto-3D. Para ello se ha tenido en cuenta, como se explicaba en capítulos anteriores, que los conjuntos de entrenamiento y de prueba sean disjuntos. El resultado del mismo dará los porcentajes de acierto, los resultados de la función de pérdida y los valores de las métricas que se obtienen con estas redes neuronales usando los datos reales. La importancia del experimento es mayúscula pese a que no esté orientado hacia la validación de las hipótesis planteadas porque formaliza la base del resto de experimentos.

Tras la prueba del modelo entrenado con los datos de Toronto-3D sobre las redes, se han recopilado todos los datos almacenados en los *logs* y se han volcado sobre la Tabla 5.2 (a excepción de la clase indefinida que no proporciona información). En ella queda representado por colores los porcentajes recuperados para una mejor visualización. Concretamente, los tonos más verdosos muestran mejores resultados, mientras que los más rojizos muestran los peores resultados.

Label	PointNet++		Point Transformer		Umbrella RepSurf	
	IoU	Acc	IoU	Acc	IoU	Acc
Road	94,06	99,27	93,39	99,4	94,05	99,22
Road marking	0,01	0,01	0	0	1,37	1,37
Natural	93,61	96,26	90,73	92,79	94,49	96,42
Building	88,28	97,15	84,95	97,56	89,84	96,78
Utility line	61,46	63,05	61,21	63,73	63,22	64,29
Pole	68,82	76,74	68,91	77,48	72,43	84,5
Car	84,3	87,38	74,37	79,03	81,7	88,24
Fence	12,2	29,64	14,98	22,1	18,72	43,62

Tabla 5.2: Resultados del experimento 1

Repasando uno a uno los datos, lo primero que ha de destacarse son los buenos resultados que se obtienen en todas las redes para la detección de todos los objetos, a excepción de las clases “Road marking” y “Fence”, donde todas fallan en mayor o menor medida. Sobresalen de entre los porcentajes obtenidos aquellos que reflejan las métricas de detección de los objetos Road, Natural y Building, rozando y superando el 90%. Sin embargo, para entender por qué los resultados para los objetos “Road marking” y “Fence” han sido tan bajos hay que tener en cuenta el contexto del experimento, así como el de los datos.

Comenzando con la primera puntualización, este experimento trabaja solamente con la posición de los puntos y sus etiquetas, sin tener en cuenta el color. Esto conlleva que el modelo no sea capaz de diferenciar el suelo de las marcas viales, pues estas marcas son únicamente suelo pintado donde no cambia la geometría. Determinar dónde están dichas marcas solo con la posición de los puntos sería un resultado engañoso, producto de un sobreaprendizaje. Es por ello que, aunque no se hayan podido detectar las marcas del suelo, es un resultado esperado y acorde. En cuanto a la segunda problemática, los datos contienen una gran cantidad de puntos que se han etiquetado usando las clases disponibles, pero en diferentes proporciones. La existencia de clases con mayor número de instancias registradas frente a otras conlleva que, en casos de mayor desbalanceo entre clases, el modelo no aprenda correctamente sobre aquellas con menos instancias. Esto que se acaba de explicar es lo que sucede con la clase “Fence”, donde apenas hay instancias y la repercusión se da de forma directa en los resultados. Aunque no es la única perjudicada porque, pese a obtener buenos resultados en las otras clases, este problema se refleja también para “Utility line” y “Pole”, clases en las que hay menos instancias que las que se podrían encontrar en otras como “Road”.

Vistos los datos numéricos, es interesante ver los resultados de una manera más visual para comprenderlos mejor. En la Ilustración 5.18 se muestra la nube etiquetada original, mientras que en la Ilustración 5.19 está el resultado de la segmentación con Umbrella RepSurf (la que mejor funcionó en este experimento).

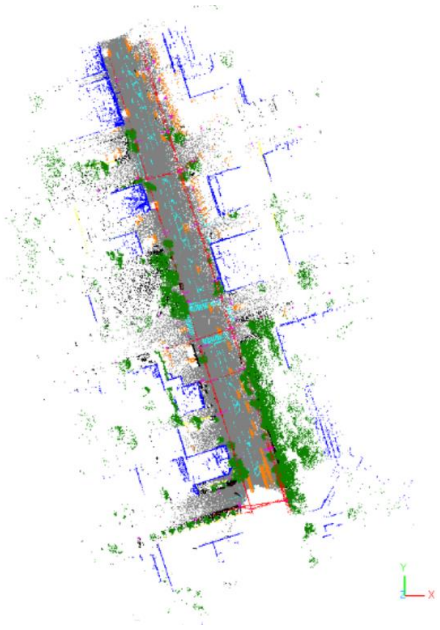

Ilustración 5.18: Nube de puntos esperada

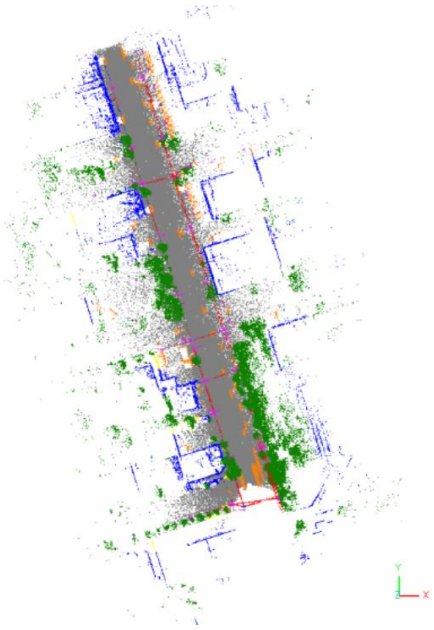

Ilustración 5.19: Nube de puntos obtenida

Los resultados obtenidos visualmente se ven casi iguales y la mayoría de los fallos se encuentran en puntos alejados de la zona con mayor concentración (además de aquellos pertenecientes a las clases minoritarias). Con esto venimos a decir que en lugares donde la densidad de puntos es muy baja es más complicado diferenciar correctamente a qué pertenece cada punto debido a la falta de vecinos que permitan catalogarlo.

Siempre es interesante ver cómo queda la nube de puntos y es por ello que se seguirá mostrando en el resto de experimentos, sin embargo, es complicado analizar en detalle con este tipo de visualización. Para aumentar el rigor del estudio, así como para facilitar las comparaciones entre los resultados de las 3 redes se agrupará todo en gráficos de barras. A continuación, se presentarán 2 gráficos para comparar las métricas IoU y Acc de las 3 redes.

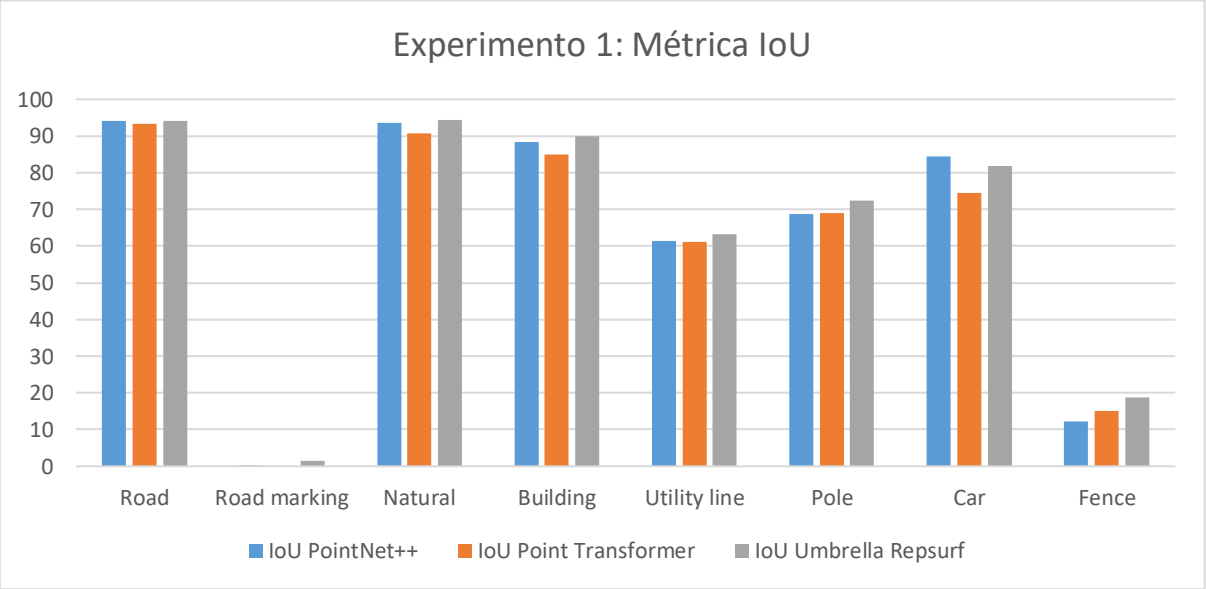

Ilustración 5.20: Métrica IoU entre redes para experimento 1

En esta gráfica, Ilustración 5.20, se muestra una comparativa de los resultados de la métrica IoU donde se corrobora el comportamiento similar que se da en todas las redes para el problema de la segmentación. De forma sintetizada recordamos que el IoU está asociado a la forma de los volúmenes segmentados por clase. Como ya se explicó, esta métrica mide la superposición entre la región predicha por el modelo y la región real. En el diagrama de barras, aunque con comportamientos muy parecidos, destaca RepSurf en casi todos los casos, seguida por PointNet++ y dejando a Point Transformer en la cola.

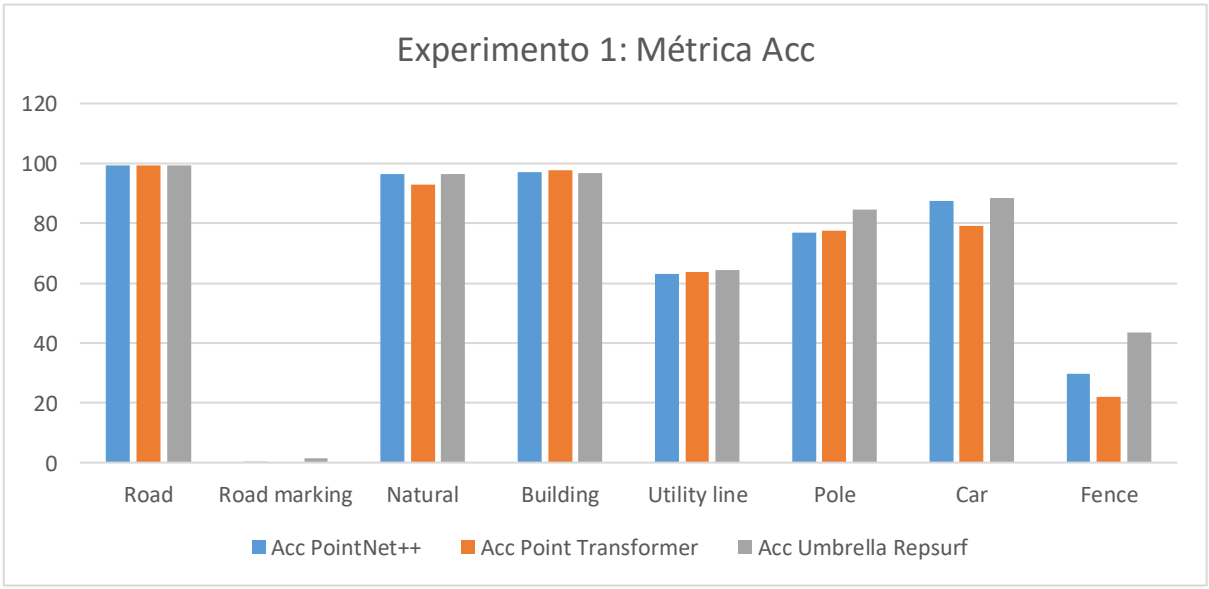

Ilustración 5.21: Métrica Acc entre redes para experimento 1

En este otro gráfico, Ilustración 5.21, vemos la misma comparativa pero ahora con la métrica Acc, que en esencia viene a indicar cuántos de los puntos se han etiquetado correctamente para cada clase en particular. Esta métrica frente a la otra puede dar algo menos de información, pero siempre es importante tenerla en cuenta para contrastar los resultados. Al igual que en la métrica IoU, RepSurf destaca como la red que es capaz de acertar mayor número de puntos a la hora de etiquetar, sobre todo con las clases minoritarias (aunque en este tipo de instancias presentan dificultades). Con el resto de clases hay que indicar que todas se comportan prácticamente similar.

Como se ha visto, todas tienen un comportamiento similar, destacando de media RepSurf, algo que se observa mejor en la Ilustración 5.22. En ella se pueden ver los valores medios de IoU y Acc, introduciendo ahora la métrica OA. La métrica OA recordemos que sirve para indicar el porcentaje de puntos etiquetados correctamente en total, diferenciándose de la mAcc porque la OA pondera todos los puntos por igual.

Neural Network	mIoU	mAcc	OA
PointNet++	66,97	72,17	94,64
Point Transformer	65,39	70,23	93,84
Umbrella RepSurf	68,42	74,94	94,82

Tabla 5.3: Resultados generales del experimento 1

Como vemos se obtienen porcentajes de acierto para OA que rondan entre el 93.84% y 94.82%, lo cual son valores muy buenos. En cuanto al acierto por clase, dado por mAcc, tenemos una media de acierto de entre el 70.23% y el 74.94%. Finalmente, en cuanto al acierto dado por la mIoU, encontramos de media valores que están entre el 66.97 y el 68.42%. Aunque la mIoU y la Acc tengan valores muy inferiores a los que muestra OA, no hay que dejarse engañar, son valores medios entre clases con un número de instancias no balanceado (cada fallo en instancias de clases minoritarias es condenado más que cada fallo en clases mayoritarias). Esto significa que las redes funcionan bien en la mayoría de las instancias, pero pecan de fallar con aquellas con las que el modelo está menos acostumbrado a trabajar.

Con estos resultados ya tenemos una base sólida y de referencia para pasar a ejecutar el resto de experimentos.

5.4.2 Experimento 2: Entrenamiento y testeo con datos sintéticos

El segundo experimento tiene como objetivo intentar validar la primera hipótesis del proyecto: *"El comportamiento de las redes obtenido a partir de un entrenamiento y testeo con nubes de puntos sintéticas generadas con el LiDAR virtual es similar al obtenido a partir de nubes de puntos reales"*. Para ello se han entrenado y probado los modelos con datos completamente sintéticos, pertenecientes al conjunto propio Synthetic Cloud 3D. La hipótesis podría ser validada si las redes pueden aprender de forma similar con valores sintéticos a como lo hacían en el experimento 1, es decir, deben obtenerse resultados parecidos en las métricas de ambos experimentos. A continuación, se muestran los resultados recopilados en la Tabla 5.4.

Label	PointNet++		Point Transformer		Umbrella RepSurf	
	IoU	Acc	IoU	Acc	IoU	Acc
Road	84,51	95,79	72,19	88,91	85,52	95,36
Road marking	5,26	5,62	8,02	8,63	9,74	10,94
Natural	84,3	91,19	54,57	68,26	88,05	93,82
Building	94,36	98,26	94,33	98,16	95,31	98,92
Utility line	0	0	0	0	0	0
Pole	69,58	72,26	65,78	67,68	70,19	72,39
Car	77	86,13	75,22	85,39	80,85	90,89
Fence	0	0	0	0	0	0

Tabla 5.4: Resultados del experimento 2

Aunque de un simple vistazo los resultados son parecidos, pero con tasas de acierto algo menos elevadas, destaca la obtención del 0% en las métricas de las clases "Utility line" y "Fence". La explicación es simple, en los datos sintéticos utilizados no había instancias de dichas clases. Es un resultado esperable y mejorable si se incluyeran este tipo de elementos en la generación procedural de ciudades.

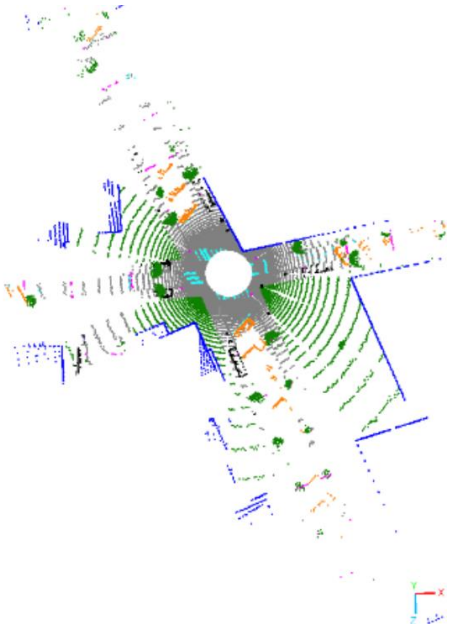

Ilustración 5.22: Nube de puntos obtenida

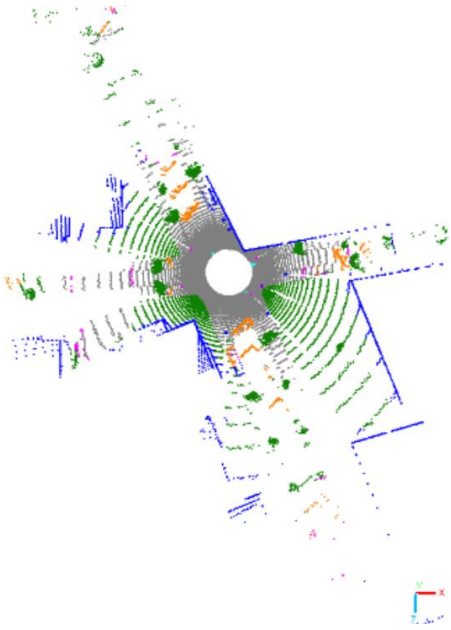

Ilustración 5.23: Nube de puntos obtenida

Visualmente, en la Ilustración 5.22 y la Ilustración 5.23 vemos que los resultados efectivamente siguen siendo buenos para un ejemplo tomado con el modelo entrenado con Umbrella RepSurf (nube original y nube segmentada respectivamente). A continuación, con la Tabla 5.5 vamos a ver la información general del experimento.

Neural Network	mIoU	mAcc	OA
PointNet++	57,22	61,03	91,43
Point Transformer	52,24	57,45	84,63
Umbrella RepSurf	58,85	62,48	92,32

Tabla 5.5: Resultados generales del experimento 2

En este caso la red que parece devolver mejores resultados es RepSurf. Ello es algo que se repite cara al anterior experimento, sin embargo, como se había adelantado, hay tasas menos elevadas de acierto. Esta reducción podría ser debida a múltiples factores, destacando:

- **La menor densidad de puntos por nube**: las nubes de Toronto-3D tienen una mayor cantidad de puntos (sobre 800 mil) que las de las nubes sintéticas (sobre 50 mil), por ende, el número de relaciones entre puntos vecinos se

reduce en este experimento y ocasiona mayores dificultades para aprender patrones.

- **La completitud de la nube LiDAR**: la nube de Toronto-3D fue tomada con un sensor en movimiento mientras que la nube sintética fue tomada con un sensor inmóvil. Pese a que el sensor sintético simulaba al sensor real en su configuración, el hecho de no haber aplicado movimiento se traduce en nubes de puntos con mayor oclusión y menos geometría. Esto hace más compleja la detección de algunos objetos al ser estudiados únicamente por los puntos de una de sus caras.

Sin embargo, aunque esto se pudiera dar, no hay que cegarse con estos valores procedentes de medias. Hay que tener en cuenta que estos valores se han perjudicado mucho debido a las clases que no tienen instancias en el conjunto sintético (clases “Utility line” y “Fence”). Para analizar correctamente el experimento se va a proceder a presentar unos gráficos de barras que comparen por redes y por métricas los resultados de los experimentos 1 y 2.

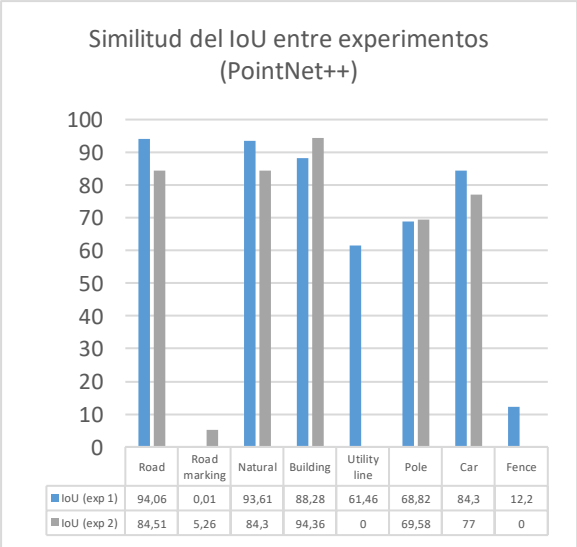

Ilustración 5.24: Comparativa entre exp 1 y exp 2 para IoU con PointNet++

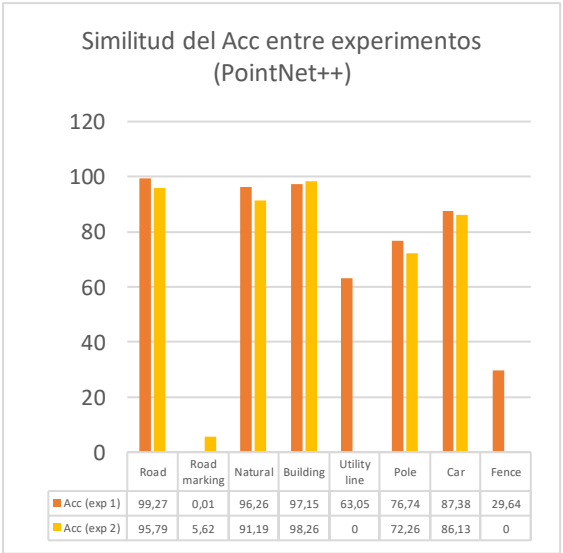

Ilustración 5.25: Comparativa entre exp 1 y exp 2 para Acc con PointNet++

En estos primeros diagramas tenemos los resultados comparativos de las métricas IoU (Ilustración 5.24) y Acc (Ilustración 5.25) para la red PointNet++. Ciertamente obtener los mismos resultados no es posible sin tener los mismos datos

y la misma semilla entre 2 ejecuciones distintas, sin embargo, se pueden alcanzar resultados aproximados. Si atendemos a los gráficos, podemos observar que para las clases “Road”, “Natural” y “Car” en el primer experimento se obtuvieron resultados ligeramente superiores. De igual manera se obtuvieron resultados mejores en el segundo experimento para las clases “Building” y “Pole” (aunque esta última levemente inferior en el estudio de la forma, la IoU).

En estos otros gráficos (Ilustración 5.26 e Ilustración 5.27) vemos que hay más diferencias. Se trata del modelo entrenado con la red Point Transformer y en este segundo experimento han mostrado resultados inferiores, sobre todo con las clases “Road” y “Natural”, y levemente superiores en las clases “Building” y “Car”. Al igual que en PointNet++, los resultados son siempre superiores para la métrica Acc que los dados por la métrica IoU. Podemos decir que Point Transformer ha funcionado por lo general peor que PointNet++ para este caso, aunque es cierto que se repiten algunos patrones que se expondrán más adelante.

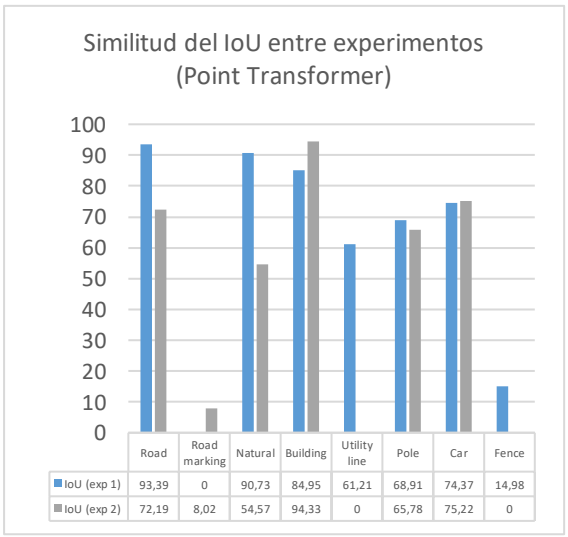

Ilustración 5.26: Comparativa entre exp 1 y exp 2 para IoU con Point Transformer

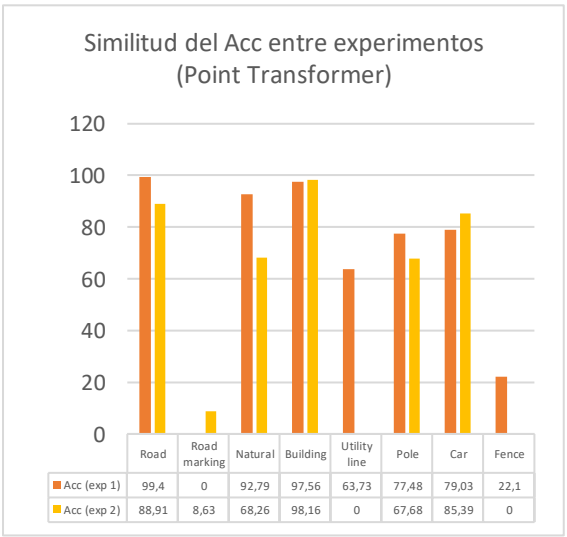

Ilustración 5.27: Comparativa entre exp 1 y exp 2 para Acc con Point Transformer

En estos últimos diagramas, Ilustración 5.28 e Ilustración 5.29, vemos los resultados con el método de Umbrella RepSurf. Ciertamente son resultados más similares a los que encontrábamos con la red PointNet++, con valores más parejos entre ambos experimentos, aunque algo por debajo a los obtenidos con el primer experimento mientras que la media de estos supera a la de las demás redes.

Encontramos mejora de rendimiento dentro este experimento con las clases “Building” y “Car” (la IoU de este último es algo inferior pero aún muy igualado con la referencia).

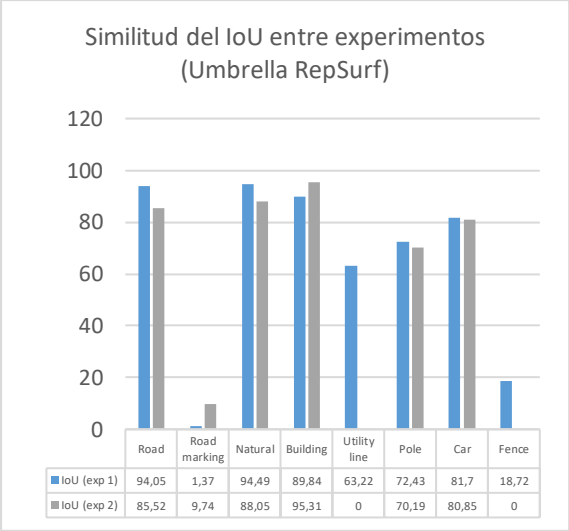

Ilustración 5.28: Comparativa exp 1 y exp 2 para IoU con RepSurf

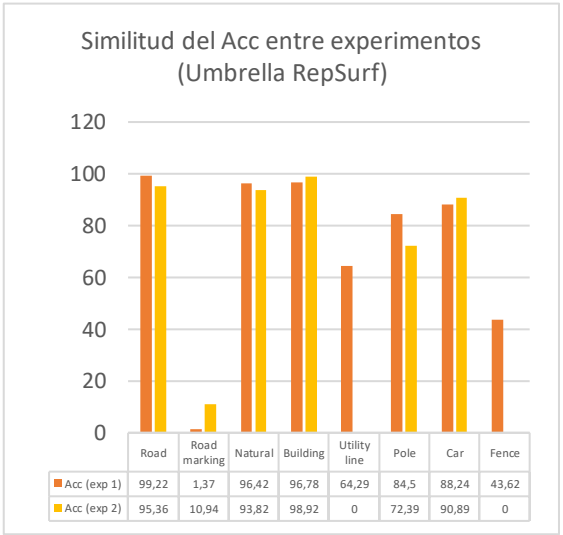

Ilustración 5.29: Comparativa exp 1 y exp 2 para Acc con RepSurf

Llegados a este punto, lo cierto es que es interesante mencionar una serie de tendencias globales que se han repetido dentro de las 3 redes. Ello nos puede llevar a la pregunta de si son fruto del azar o si por el contrario nuestros datos de Synthetic Cloud 3D no son tan parecidos a los reales (los del conjunto de Toronto-3D):

- **La clase “Buildings” con los datos sintéticos se ha visto beneficiada** en todos los casos en comparación con el experimento base. Esto podría deberse a la mayor simplicidad de la geometría que ahora tienen las paredes de estos frente a la que tenía en los datos reales, las cuales representaban pequeñas irregularidades. Si se quisiera simular dichas irregularidades en los datos sintéticos, una idea para posteriores investigaciones podría ser la de introducir aumentación en forma de ruido, asignando pequeñas variaciones en la posición de los puntos de los edificios.
- **Ha aumentado en todos los casos la detección del objeto “Road Marking”**. Como se puede observar, en este experimento los modelos están intentando predecir las marcas del suelo. Para detectar estas marcas no hay geometría ni color que ayuden a la detección, lo cual significa que está aprendiendo patrones en función de las posiciones relativas de los objetos

representados en la nube. Ello nos hace pensar que el conjunto de datos tiene poca diversidad, porque de tenerla no se podrían detectar objetos de los cuales no se tiene nada de información y que deberían de ser invisibles para el modelo. Esto también puede conllevar un posible sobreaprendizaje que sesgue nuestros resultados.

- **La clase "Road", clase mayoritaria de Toronto-3D, siempre se perjudica con los datos sintéticos**. Pese a una aparente simplificación del suelo y siguiendo el mismo razonamiento llevado con la detección de edificios, la lógica nos dice que deberían mejorarse los resultados. Pero nada más lejos de la realidad los modelos sorprenden empeorando para todas las redes. En este caso lo que puede estar sucediendo es que la simplicidad de los datos sintéticos haga que el modelo genere patrones levemente funcionales para detectar el objeto "Road Mark", entorpeciendo la decisión. Con los datos reales es seguro que hay mayor variación sobre la forma y posición de las marcas viales, haciendo que sea más complicado encontrar un patrón que se ajuste y provocando como resultado la elección del objeto "Road" cuando hay duda entre si un punto es de carretera o de marca vial.

Tras el estudio detenido de los resultados podemos decir que, aun habiendo conseguido tasas de acierto parejas con el anterior experimento, se han detectado tendencias generales que muestran posibles diferencias entre los datos reales y sintéticos. Estas diferencias son causadas principalmente por la cantidad de puntos entre las nubes, la densidad variable dentro de ellas, la falta de ruido en la geometría sintética frente a la real y la falta de variabilidad en la generación procedural.

En respuesta a lo que se plantea con la primera hipótesis, lo cierto es que no se puede llegar a validar por completo el enunciado. Efectivamente el comportamiento es parecido pese a las posibles diferencias entre datos y se podría decir que todas las redes son capaces de aprender de manera similar tanto con los datos reales como con los sintéticos. No obstante, validar por completo la hipótesis a partir de estos resultados sería irresponsable. Esto es así porque no se están teniendo en cuenta las clases "Utility line" y "Fence" (clases sin instancias sintéticas), a lo que además se añade la existencia de patrones generales de comportamiento que difieren de los

encontrados en el primer experimento, destacando en particular aquel detectado con la clase “Road mark”, cuyo origen no está claro si es fruto del azar o si en realidad las redes no se comportan igual frente a esta clase entre la primera y la segunda prueba.

5.4.3 Experimento 3: Entrenamiento con datos sintéticos y testeo con datos reales

Este tercer experimento busca validar la segunda hipótesis del proyecto, con diferencia la más complicada: *“El entrenamiento de las redes neuronales con nubes de puntos reales puede ser sustituido con un entrenamiento alimentado con datos completamente sintéticos, dando buenos resultados en el testeo con nubes de puntos reales”*. Para ello se han tomado los modelos del experimento 2, es decir, aquellos entrenados con los datos sintéticos de Synthetic Cloud 3D, pero esta vez se han probado con los datos reales de Toronto-3D. La hipótesis podría ser validada cuando las pruebas muestren resultados tan buenos, como mínimo, a los conseguidos en el experimento que marca la base (experimento 1). A continuación, se muestran los resultados recopilados en la Tabla 5.6.

Label	PointNet++		Point Transformer		Umbrella RepSurf	
	IoU	Acc	IoU2	Acc3	IoU4	Acc5
Road	85,5	89,68	93,45	98,37	90,53	95,36
Road marking	3,1	8,87	0	0	0,63	1,03
Natural	60,01	92,69	67,87	90,55	57,8	95,48
Building	31,04	37,14	50,91	64,93	18,52	19,36
Utility line	0	0	0	0	0	0
Pole	1,7	1,79	0,07	0,08	0,2	0,2
Car	12,18	14,69	16,82	25,13	18	21,88
Fence	0	0	0	0	0	0

Tabla 5.6: Resultados del experimento 2

Como se adelantaba, era un experimento complicado, con muchas variables e incertidumbre. Al igual que sucedía en el experimento 2, el modelo está entrenado con archivos que no poseen puntos de las clases “Utility line” y “Fence”, por lo que le es imposible detectarlos en el testeo. Pese a ello, a nivel general sí que parece que ha habido una bajada importante en la eficacia. Parece que la detección de la clase “Road” sigue teniendo buenos resultados, seguida de la detección de la clase “Natural”

y “Buildings”. En cuanto al resto de clases no se puede decir precisamente que haya habido buenos resultados, aunque sí que hay indicios de detección de la clase “Car”.

Visualmente hablando, la Ilustración 5.30 muestra la nube esperada y la Ilustración 5.31 es la nube obtenida. Como se aprecia, tienen bastante parecido y podemos decir que estos mejoran en la detección de los objetos de gran tamaño, aunque haya algo de imprecisión.

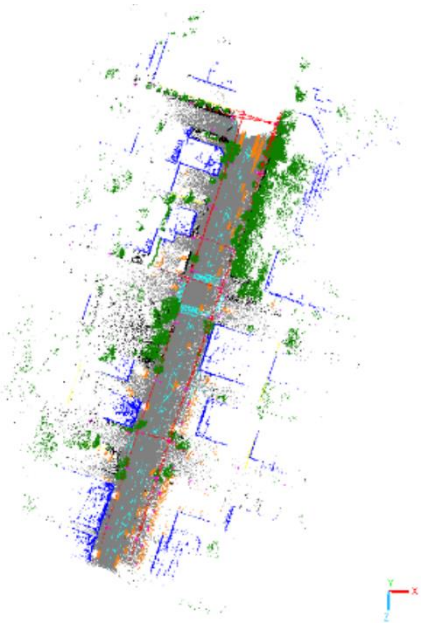

Ilustración 5.30: Nube de puntos esperada

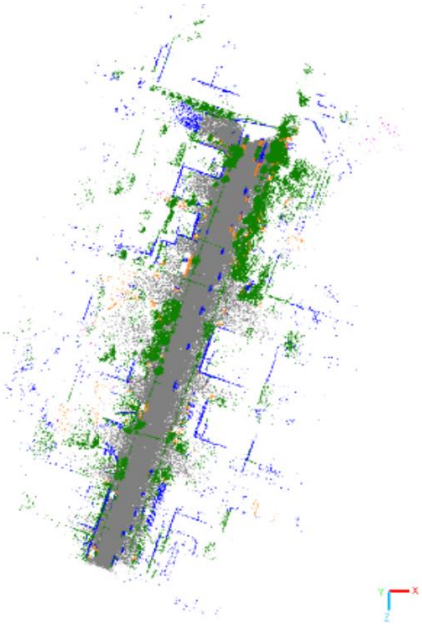

Ilustración 5.31: Nube de puntos obtenida

A nivel general entre las 3 redes podemos decir, tal y como sugiere la Tabla 5.7, que la que mejor ha funcionado es Point Transformer. Esto es curioso porque para todos los experimentos de este proyecto es la que peores resultados ha tenido siempre. Esto plantea la posibilidad de que Point Transformer sea más capaz de encontrar los atributos que caracterizan a los objetos a nivel general que el resto de redes, las cuales están más sesgadas al tipo de datos de entrenamiento.

Neural Network	mIoU	mAcc	OA
PointNet++	32,61	38,32	79,17
Point Transformer	36,57	42,12	86,55
Umbrella RepSurf	31,74	37,03	81,63

Tabla 5.7: Resultados generales del experimento 3

Es importante mencionar que, pese a los aparentes resultados deficientes obtenidos, hay un valor para el OA del 86.55, es decir, en más del 86% de los casos se han evaluado los puntos correctamente. Esto supone un error de solo un 14%, un error muy bajo pero que como vemos por el resto de métricas y que, en contraste con la tabla anterior, viene a decirnos que hay muchos puntos de las clases minoritarias que no se clasifican correctamente. Ello es algo a considerar porque en ocasiones la importancia del problema recae en encontrar ese pequeño porcentaje de puntos, algo común por ejemplo en la detección de vehículos o peatones dentro de la conducción autónoma, un problema que, con los resultados obtenidos, este modelo no sería capaz de funcionar en carreteras reales como las de Toronto-3D. Cambiando la forma de ver esta explicación, podemos decir que estos modelos, de primeras, son capaces de detectar grandes objetos representados por inmensas masas de puntos, pero no lo son para detectar los detalles representados por pocas cantidades de puntos.

En lo consecutivo se analizarán en detalle los resultados entre este experimento y el de referencia en búsqueda de más información. Primeramente, tenemos los valores del modelo entrenado con la red PointNet++. Obsérvese que la clasificación de puntos, métrica Acc (Ilustración 5.32), de las clases “Road” y “Natural” arroja resultados similares a los vistos en la primera prueba, aunque se aprecia una bajada en la precisión de la forma, dada por IoU (Ilustración 5.33), sobre todo en la clase “Natural”. Al igual que en el experimento 2, las clases “Utility line” y “Fence”, siguen sin detectarse, encontrándose también repetido el fenómeno visto con “Road marking”. Por otro lado, separamos las clases “Building” y “Car”, que se han detectado con una frecuencia destacable, aunque con mucha menos precisión tanto en IoU y Acc. Finalmente, y con gran importancia, hay que subrayar que se ha perdido casi por completo la detección de la clase “Pole” frente a anteriores experimentos, detectándose incluso menos que la clase “Road marking”, que debería ser invisible.

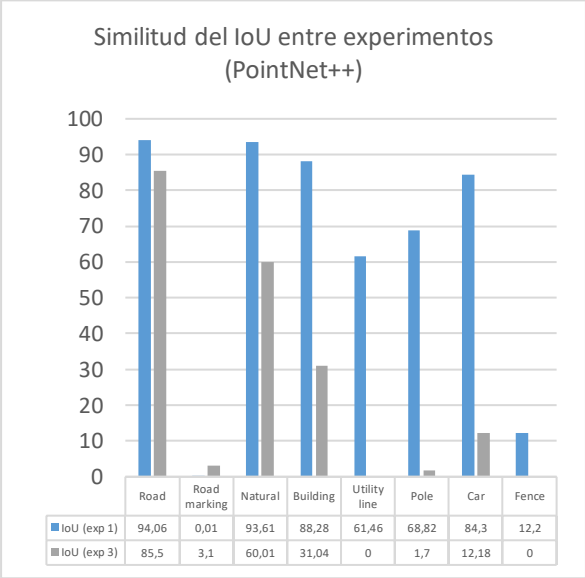

Ilustración 5.32: Comparativa entre exp 1 y exp 3 para IoU con PointNet++

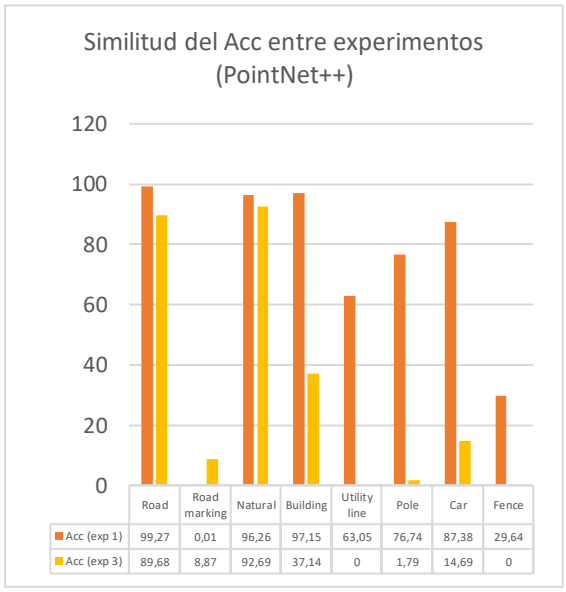

Ilustración 5.33: Comparativa entre exp 1 y exp 3 para Acc con PointNet++

Por parte de la red Point Transformer (Ilustración 5.34 e Ilustración 5.35) no hay diferencias en la detección de la carretera ni de sus marcas viales. Respecto a la red anterior, se repite la bajada de la precisión en la forma, IoU, para la clase “Natural”, de igual manera que se repite la reducción en ambas métricas para las clases “Building” y “Car”. De cara a la detección de los puntos clasificados como “Pole”, esta red tampoco parece detectar dicha clase con éxito.

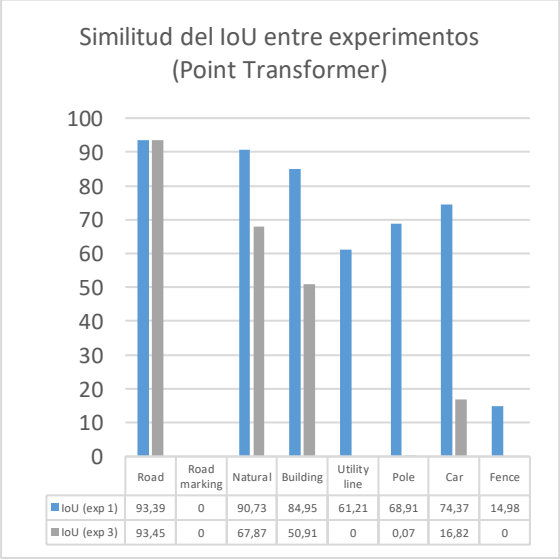

Ilustración 5.34: Comparativa entre exp 1 y exp 3 para IoU con Point Transformer

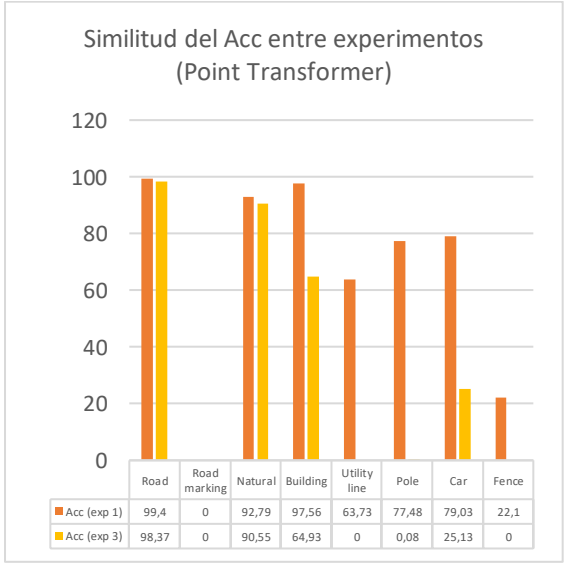

Ilustración 5.35: Comparativa entre exp 1 y exp 3 para Acc con Point Transformer

Finalmente, para Umbrella RepSurf (Ilustración 5.36 e Ilustración 5.37) vemos similitudes importantes con las anteriores redes, pero sobre todo con la red PointNet++ para todas las clases y en ambas métricas. Como vemos, la clase “Road” se detecta bien, hay una reducción en la detección de la forma para la clase “Natural”, se reduce mucho la correcta clasificación y forma de los puntos de las clases “Building” y “Car”, y se da el fenómeno de la clase “Road marking” y “Pole”. Esta red, al igual que PointNet++, obtiene peores resultados que Point Transformer.

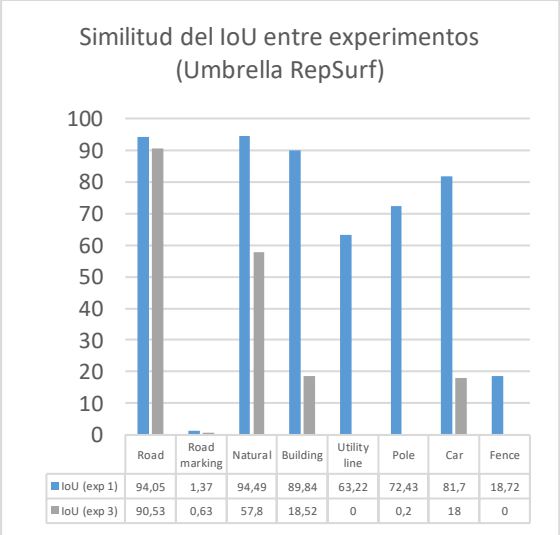

Ilustración 5.36: Comparativa entre exp 1 y exp 3 para IoU con RepSurf

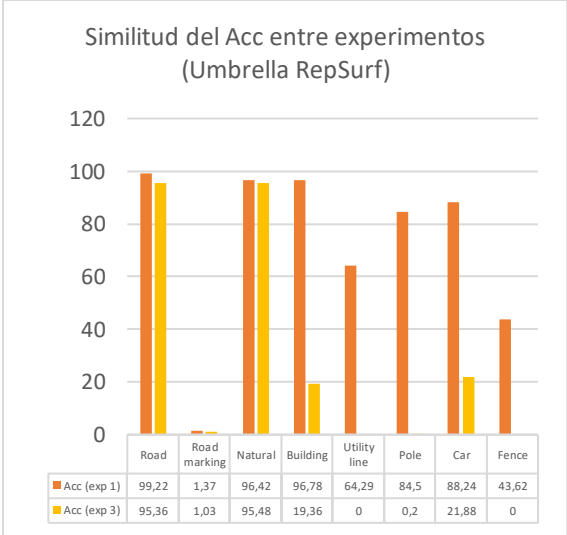

Ilustración 5.37: Comparativa entre exp 1 y exp 3 para Acc con RepSurf

Como se ve y se adelantaba en el experimento 2, los datos reales tienen diferencias apreciables, al menos cara a la detección interna de patrones, pues de no ser así se habrían segmentado las nubes con un éxito similar al del experimento 1. De entre las tendencias generales encontramos el buen funcionamiento de las clases “Road” y “Natural”, así como la reducción de la precisión en otras como “Building”, pero vamos a destacar en concreto los siguientes comportamientos:

- **La imposibilidad de detección de la clase “Pole”**, algo que posiblemente se dé porque los modelos confunden a las farolas con el tronco de los árboles, marcando todas las farolas como parte de la vegetación por ser el tipo de objeto que más ha visto en su entrenamiento.

- **La gran bajada de precisión en la detección de los vehículos**, algo que seguramente esté causado por tener fragmentos de ciudades inmóviles. Durante la medición de la ciudad de Toronto-3D había vehículos en movimiento y estos eran escaneados por el LiDAR en lugares distintos en cada pasada que hacía. Sin embargo, en los fragmentos procedurales que nosotros hemos hecho los vehículos no se mueven y en todas las pasadas todos los vehículos se sitúan en el mismo sitio. Esta diferencia en la representación de los vehículos puede estar entorpeciendo al modelo para detectar la totalidad de los puntos clasificados con la clase Car cuando se prueba con datos reales.

Tras estas pruebas, es importante comentar que no se puede validar la hipótesis de sustitución total de los datos con este experimento dado que no se ha alcanzado a igualar o superar los resultados del primero. No obstante, los resultados sí que muestran que con datos sintéticos se puede reconocer correctamente algunos elementos, lo que nos lleva a pensar que con una generación procedural más realista se podrían obtener mejores resultados que pudieran llevar a la validación de la hipótesis.

A partir de este experimento, y como paso intermedio entre esta investigación y una próxima que tenga en cuenta las mejoras que se proponen en esta memoria, surge la duda si sería posible conseguir el efecto buscado, pero con unos datos de entrenamiento parcialmente sintéticos. Con ello se propone una última hipótesis dentro de este proyecto:

- Hipótesis 3: “El entrenamiento con datos completamente sintéticos puede mejorarse mucho introduciendo un bajo porcentaje de nubes de puntos reales, mejorando la detección de detalles”.

5.4.4 Experimento extra: Entrenamiento con datos mixtos y testeo con datos reales

Este cuarto experimento adicional tratará de validar la hipótesis 3, presentada en las conclusiones del experimento 3. Ahora lo que se hará es entrenar las redes con

un 80% de datos sintéticos (Synthetic Cloud 3D) y un 20% de datos reales (Toronto-3D), para posteriormente probar los modelos con datos reales. Con ello se busca reducir la cantidad de datos reales necesarios para obtener un modelo con resultados cercanos a los que se obtendrían si se hubiera entrenado con el 100% de los datos reales. Se podrá decir que el experimento tiene éxito y valida la hipótesis si es capaz de mejorar los resultados que se tuvieron con el experimento anterior. Dicho esto, veamos la tabla de resultados, Tabla 5.8.

Label	PointNet++		Point Transformer		Umbrella RepSurf	
	IoU	Acc	IoU	Acc	IoU	Acc
Road	93,16	98,39	83,88	88,16	93,4	98,39
Road marking	0	0	3,36	11,66	0,53	0,53
Natural	84,63	96,13	76,36	87,18	83,28	98,28
Building	79,55	88,82	60,93	91,54	83,93	90,99
Utility line	28,96	29,43	2,77	2,78	28,13	28,46
Pole	48,76	55,92	8,35	8,59	50,79	59,1
Car	57,85	69,4	30,79	33,45	42,18	46,18
Fence	1,62	1,62	0	0	0,06	0,06

Tabla 5.8: Resultados del experimento 4

Los resultados de este experimento son bastante mejores a los resultados conseguidos con el experimento 3 y más cercanos aparentemente a los del experimento 1. Visualmente se observa la nube original frente a la obtenida con la Ilustración 5.38 e Ilustración 5.39 respectivamente:

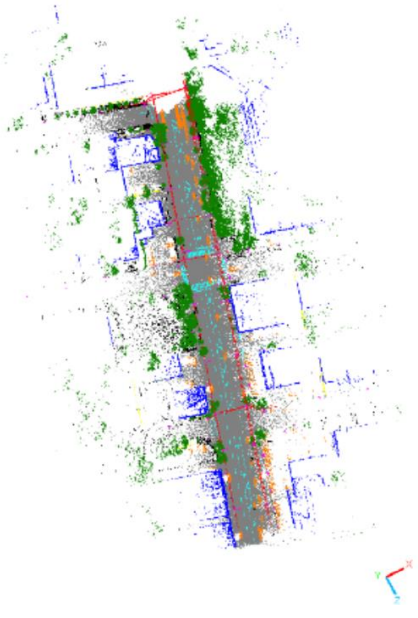

Ilustración 5.38: Nube de puntos esperada

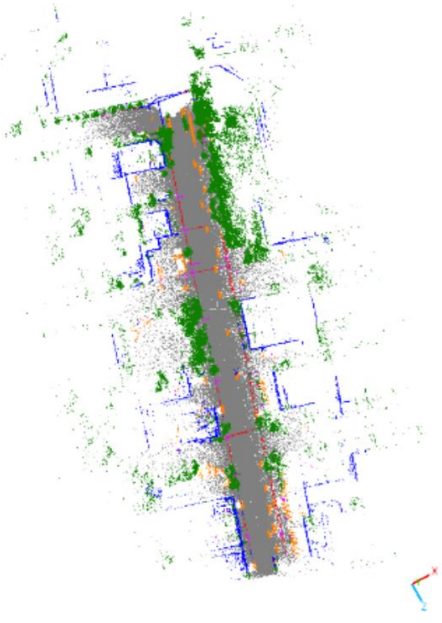

Ilustración 5.39: Nube de puntos obtenida

En cuanto a las métricas generales (Tabla 5.9), ahora la red que mejor ha funcionado de media ha sido la red PointNet++, con resultados muy cercanos a los conseguidos con el módulo Umbrella RepSurf y alcanzando un valor OA de hasta el 92.15%.

Neural Network	mIoU	mAcc	OA
PointNet++	54,95	59,97	92,15
Point Transformer	40,72	47,04	82,69
Umbrella RepSurf	53,59	58	92,05

Tabla 5.9: Resultados generales del experimento 4

Para ver en detalle los resultados se estudiarán los diagramas de barras (Ilustración 5.40 e Ilustración 5.41) que enfrentan los obtenidos en este experimento con los obtenidos del experimento 1 para ver lo próximos que se encuentran, añadiendo además los resultados del experimento 3 para comprobar si ha habido y cómo ha sido la mejora de los resultados. Comenzando con la red PointNet++, los resultados mejoran en todos los casos frente al experimento 3, obteniendo valores de IoU y Acc muy similares sobre todo con las clases “Road”, “Natural” y “Building”. En este experimento además se recupera la detección de las clases “Utility line” y “Pole”, mejorando mucho la detección de la clase “Car”.

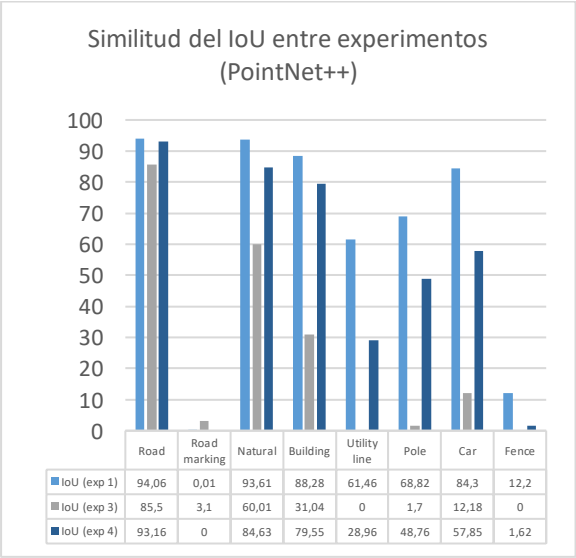

Ilustración 5.40: Comparativa entre exp 1, exp2 y exp 3 para IoU con PointNet++

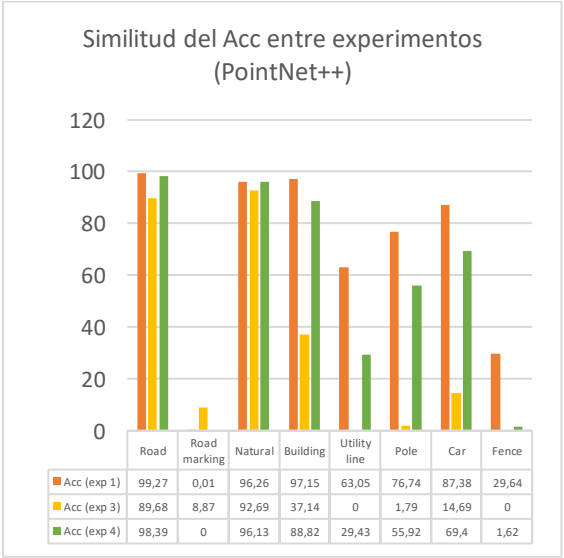

Ilustración 5.41: Comparativa entre exp 1, exp2 y exp 3 para Acc con PointNet++

En cuanto a la red Point Transformer (Ilustración 5.42 e Ilustración 5.43), lo primero que destaca es que los mejores resultados para la clase “Road” se obtuvieron en el experimento 3 y no se han podido mejorar con ningún experimento. Esto se debe a lo que anteriormente se explicó relacionado a los errores en la clasificación de la carretera procedentes del intento de detectar la clase “Road marking”. Referente al resto de clases el comportamiento del modelo en este experimento frente al anterior mejora, aunque poco en la mayoría de casos, siendo además superado en rendimiento por los resultados del experimento 1.

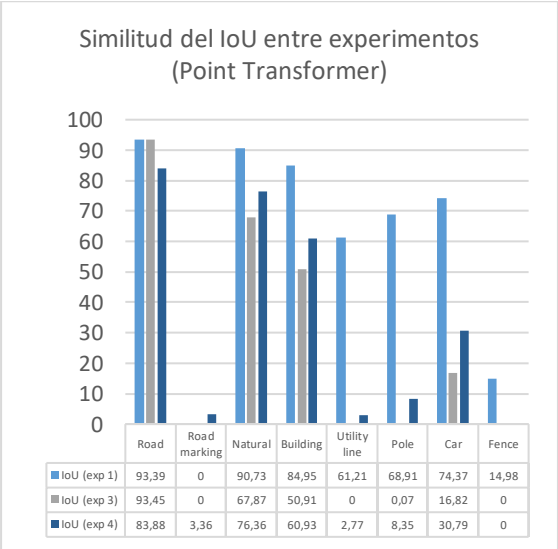

Ilustración 5.42: Comparativa entre exp 1, exp2 y exp 3 para IoU con Point Transformer

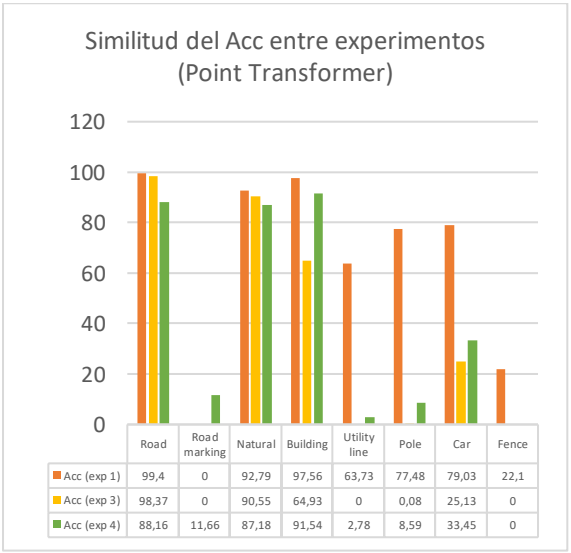

Ilustración 5.43: Comparativa entre exp 1, exp2 y exp 3 para Acc con Point Transformer

Para terminar, RepSurf (Ilustración 5.44 e Ilustración 5.45) repite los comportamientos que tenían las otras redes. Se obtienen resultados mejorados respecto al experimento 3 pero aun inferiores a los del experimento de referencia. Entre lo más destacable está la mejora de la detección de la forma, IoU, de la clase “Natural” frente al anterior experimento teniendo en cuenta que en este la cantidad de puntos correctamente clasificados era similar.

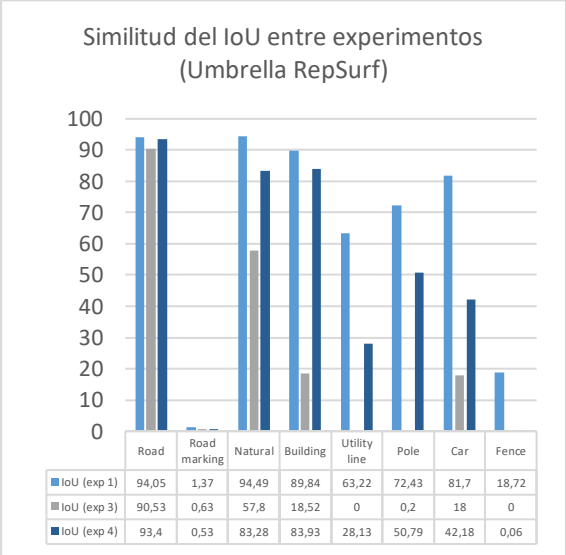

Ilustración 5.44: Comparativa entre exp 1, exp2 y exp 3 para IoU con RepSurf

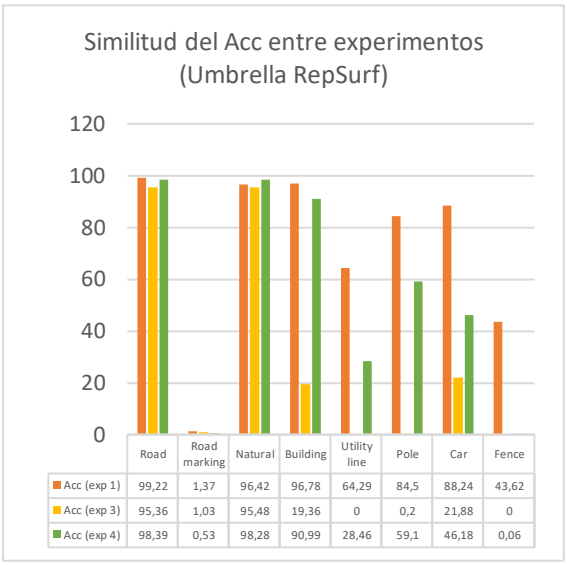

Ilustración 5.45: Comparativa entre exp 1, exp2 y exp 3 para Acc con RepSurf

En cuanto a patrones generales interesantes detectados podemos decir que:

- **Se mejoran los resultados en gran medida frente al experimento anterior** en todas las redes, destacando la red PointNet++.
- **Se recupera la detección de la clase "Pole"** en este experimento frente al anterior, lo cual nos indica que ahora es capaz de diferenciar correctamente las clases "Pole" y "Natural".
- **La clase "Fence" sigue siendo muy complicada de detectar** dado que apenas hay instancias de ella en el conjunto de entrenamiento.

En vista del análisis esta vez sí que podemos validar la última hipótesis, la hipótesis 3. Hemos conseguido demostrar que las redes son capaces de mejorar mucho su rendimiento incluyendo sólo un pequeño porcentaje (20%) de datos reales a nuestro entrenamiento.

6 RESULTADOS Y DISCUSIÓN

Para la obtención de los resultados de este proyecto fue necesario generar proceduralmente un conjunto de ciudades con el software de CityEngine. Posteriormente esas ciudades fueron fragmentadas y etiquetadas con una aplicación desarrollada en Unity exclusivamente para este proyecto. Ese etiquetado era completamente personalizado, gracias a una serie de tablas de conversión que permitían asociar etiquetas y materiales a los objetos según su textura.

Ilustración 6.1: Fragmento de ciudad sintético etiquetado

Para dichos fragmentos se ubicó un sensor LiDAR virtual y se obtuvo un conjunto de nubes de puntos que, tras aplicar una selección y procesado de datos, se utilizó para crear un nuevo *dataset*, bautizado como Synthetic Cloud 3D.

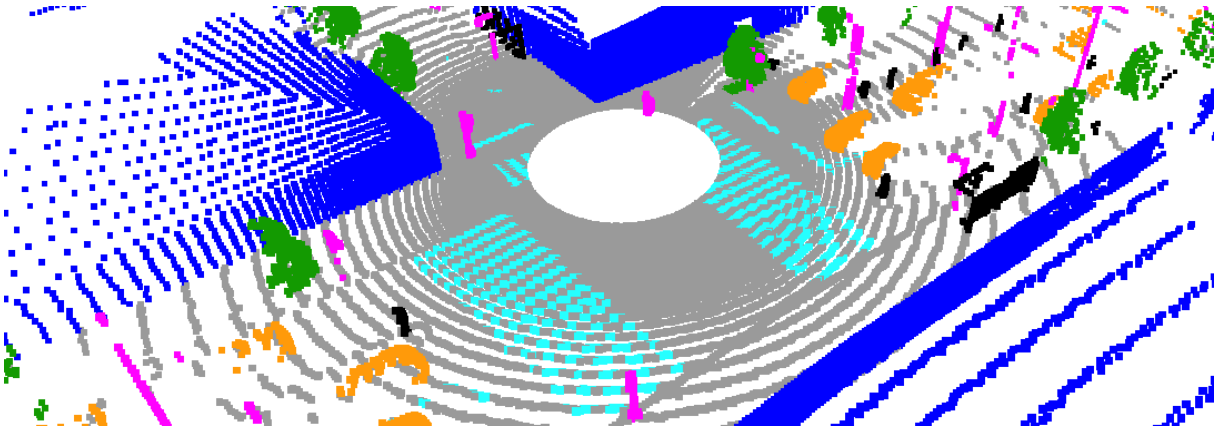

Ilustración 6.2: Nube de puntos sintética

Una vez en posesión del *dataset* sintético Synthetic Cloud 3D y del *dataset* real Toronto-3D, comenzó una etapa de experimentación que buscó validar 2 hipótesis iniciales mediante la realización de 3 experimentos. Para ello se seleccionó y se adaptó en los ordenadores del laboratorio un proyecto que incluía múltiples redes neuronales aplicadas a nubes de puntos: PointNet++, Point Transformer y Umbrella RepSurf (módulo que trabaja sobre PointNet++ SSG). Tras la realización de los experimentos no se consiguió llegar a resultados concluyentes, pero sí que hubo grandes avances para la investigación. Con un experimento se pudo comprobar que todas las redes eran capaces de aprender de manera más o menos similar, al menos para las clases que tenían instancias. Mientras tanto en otro experimento se consiguió ver que, aunque no se puedan sustituir los datos reales por datos sintéticos, sí que existían patrones que prometían que con una generación procedural más avanzada podría llegar a conseguirse. Además, a raíz del último experimento surgió una nueva hipótesis que cuestionaba si era posible que, con una baja cantidad de datos reales, se pudieran mejorar los resultados de este. Esta hipótesis extra supuso la realización de un experimento adicional que finalmente logró validarla con éxito.

Resumiendo de forma comparativa y a gran escala los resultados entre los experimentos tenemos los siguientes gráficos, los cuales nos muestran por red cuáles han sido los valores registrados por las métricas.

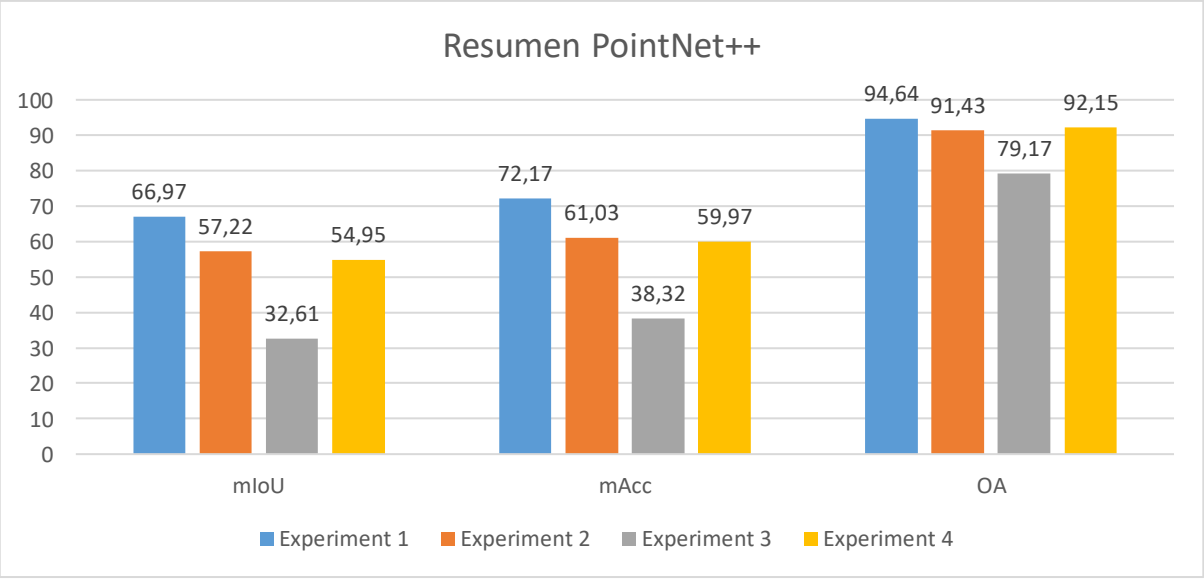

Ilustración 6.3: Comparación de resultados generales en PointNet++

En PointNet++ (Ilustración 6.3) se han obtenido en todos los casos los mejores resultados con el experimento 1 (entrenamiento y prueba con datos reales) mientras que los peores datos se han conseguido con el experimento 3 (entrenamiento con sintéticos y prueba con reales). Los resultados del cuarto experimento (entrenamiento con datos mixtos y prueba con reales) a nivel general fueron muy similares a los resultados con el experimento 2 (entrenamiento y prueba con datos sintéticos). En todos los casos y para todas las métricas, el experimento 4 mejora al experimento 3. Centrándonos en las métricas, se muestra mayor facilidad en la medición de la Acc, cantidad de puntos acertados por clase de media, que la del IoU, asociado a la forma. De cara al OA podemos ver que la amplia mayoría de los puntos se clasifican correctamente, superando el 90% en todos los experimentos a excepción del tercer experimento.

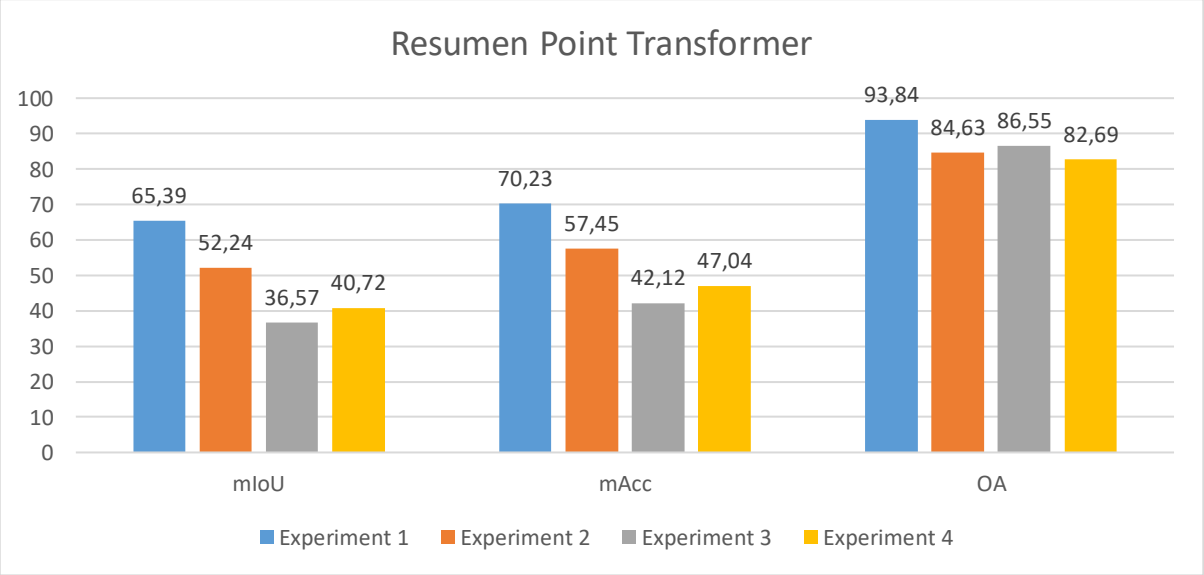

Ilustración 6.4: Comparación de resultados generales en Point Transformer

En Point Transformer (Ilustración 6.4) se vieron los peores resultados a nivel general entre los experimentos. Curiosamente esta red consiguió destacar sobre las otras en el experimento 3, donde se obtuvieron resultados que llegaron a crecer frente al experimento 2 y 4 en la métrica OA. Pese a esos resultados que incluso aumentaron el OA, lo cierto es que relacionado al IoU y al Acc los resultados eran más bajos. Igualmente, los mejores resultados se daban con el primer experimento, pues el segundo experimento tuvo peores resultados y el cuarto experimento tampoco llegó a igualar ni al primero ni al segundo.

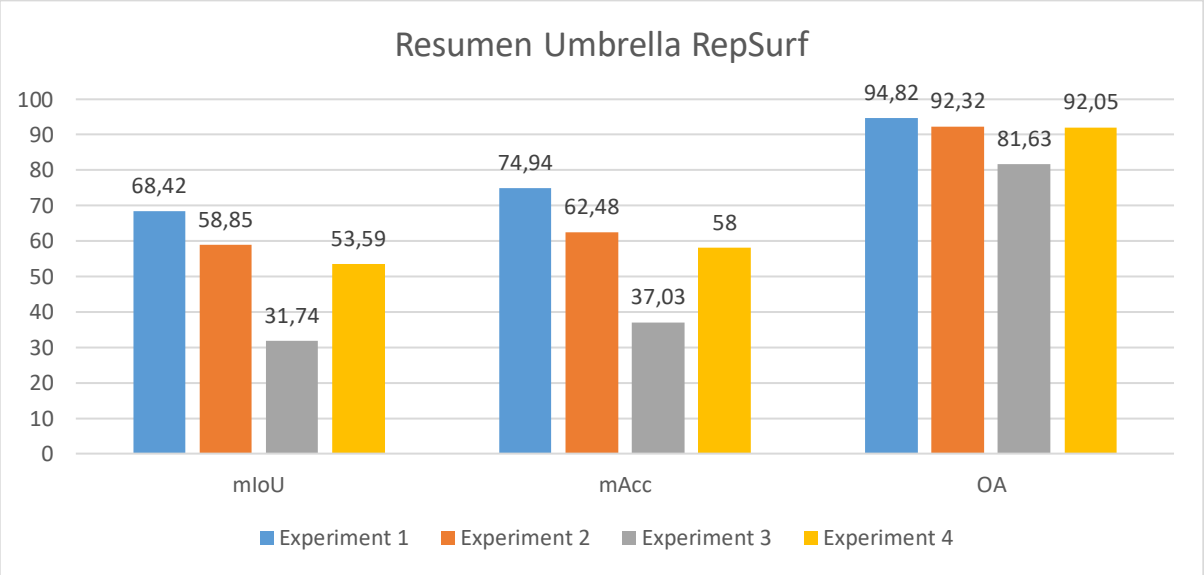

Ilustración 6.5: Comparación de resultados generales en RepSurf

Umbrella RepSurf (Ilustración 6.5) dio resultados que en más de una ocasión eran similares a la de la red PointNet++. Los mejores resultados siguen siendo aquellos del experimento 1 y los peores los del resultado 3, encontrándose los resultados de los experimentos 2 y 4 prácticamente parejos. Al igual que antes, la OA muestra como la mayoría de los puntos son correctamente etiquetados, siendo el 81% en el peor de los casos y superando el 92% para el resto.

En lo que a tiempos de ejecución se refiere (entrenamiento + prueba) como vemos en la Tabla 6.1, el experimento 1 fue el más lento, mientras que el experimento 2 fue el más rápido. Esto se debe a que el experimento 1 trabajaba únicamente con datos reales, los cuales de media tienen 800000 puntos por nube, y el experimento 2 únicamente con datos sintéticos, los cuales tienen 50000 puntos por nube. Por tanto, a mayor número de puntos mayor es el tiempo de ejecución y viceversa. Relacionado a la velocidad de ejecución en función de la red, se puede apreciar que la red más rápida es la red PointNet++, mientras que la más lenta es Point Transformer.

	PointNet++	Point Transformer	Umbrella RepSurf
Experimento 1	393	2388	948
Experimento 2	152	507	164
Experimento 3	164	549	192
Experimento 4	247	997	411

Tabla 6.1: Comparativa de tiempos totales entre redes (en minutos)

Tras los análisis realizados, cabe mencionar también que se considera que la experimentación tiene un margen de mejora que podría hacer posible la validación de la hipótesis 1 y 2 del proyecto. Para ello habría que tener en cuenta tal y como se ha comentado, la densidad y el número de puntos por nube, la característica del color, la falta de ruido en las nubes sintéticas, la movilidad del sensor LiDAR y la naturaleza inmóvil de los fragmentos sintéticos.

7 CONCLUSIONES Y TRABAJOS FUTUROS

El presente no sería hoy sin el futuro que un día alguien imaginó. La creatividad propone los proyectos de ingeniería más punteros y quien los hace posibles son la investigación, el desarrollo, la innovación y equipos de personas como yo con muchas ganas de cambiar el mundo.

Sin duda este ambicioso proyecto I+D+i ha impulsado una nueva línea de investigación dentro del laboratorio al que pertenezco. La incertidumbre y la necesidad de nuevos conocimientos para dar con soluciones novedosas provocaron que me adentrase en el mundo de la investigación. Esta se me presentó en un inicio como un árido terreno que iba mostrando su claridad conforme más me adentraba en campos como el *Machine Learning*, la generación procedural, tecnologías LiDAR, *datasets* para aprendizaje con nubes de puntos, etc. Áreas que, si bien no eran del todo nuevas para mí, sí que no las había trabajado tan en profundidad como hasta el momento.

Gracias al TFM podemos asegurar que he adquirido una gran cantidad de conocimientos en el ámbito de la inteligencia artificial, planificación y gestión de proyectos de investigación, así como también dentro del desarrollo de software gráfico. Sumado a esto, no solo he afianzado una gran cantidad de saberes de la carrera y profundizado en muchos otros más novedosos, sino que he sido capaz de orientarlos de manera práctica y novedosa, dando un fruto merecido a las horas de investigación dedicadas al trabajo. En resumidas cuentas, he madurado, al menos desde el punto de vista personal, en el ámbito de la ingeniería.

Personalmente, y dando mi opinión, pienso que este tipo de trabajos son apuestas para el mañana donde, dada la dificultad que presentan, aportan un valor añadido pese a que no siempre se consigan los resultados esperados. Siendo muy complicado obtener soluciones concluyentes, me alegro de pensar que esto será la base para continuar el trabajo en el laboratorio. Los avances conseguidos permitirán a otros investigadores continuar la investigación por donde la dejé, mejorando aquellos puntos que se han ido detallando dentro de esta memoria, aportando ideas renovadas y más novedosas de un futuro que a día de hoy aún está por descubrir.

Para terminar, he de mencionar que este proyecto ha sido emocionante desde que inició, sentimiento que impulsó el esfuerzo y dedicación hacia el mismo. Con ello, esta investigación finaliza con unos resultados que, aun no siendo concluyentes para todas las hipótesis, considero muy buenos y sobrepasan las estimaciones calculadas al inicio del trabajo. Relativo a los trabajos futuros, se enumerarán algunos de los aspectos a pulir que quedarán pendientes para una próxima ocasión:

- **Creación de un prototipo que aplique los modelos** dentro de una actividad real que pueda ayudar de forma directa a la población proporcionando un servicio.
- **Experimentación con otros conjuntos de datos**, tales como los ofrecidos por Semantic3D, Semantic KITTI, Paris-Lille-3D o DublinCity.
- **Ampliación del hardware** y uso de servidor para ejecutar el proyecto de redes neuronales con mayor cantidad de datos, lo que permitiría obtener un modelo más consistente.
- **Incremento de la variabilidad de las ciudades**, introduciendo para su generación un conjunto mayor de reglas de generación procedural, una mejora del realismo de los modelos y mayor ruido (por ejemplo, personas y vehículos con animación).
- **Diseño de un algoritmo para la generación procedural de ciudades** que permita una mayor personalización y suponga una alternativa de código abierto útil para la comunidad científica. Esta propuesta puede incluir el uso de las redes generativas para generar modelos de edificios, mobiliario urbano, vegetación y personas para utilizar las últimas tecnologías en pro de unos resultados más variados y de calidad.
- **Mejora de las nubes sintéticas** usadas para los experimentos propuestos en este proyecto, incorporando al menos: un LiDAR virtual en movimiento para captura más detallada del entorno, captura del color junto con la posición de los puntos y mayor aumentación.
- **Mejora de la interfaz de la aplicación** para fragmentar y etiquetar ciudades, algo en lo que por las restricciones temporales no se ha entrado en

profundidad, pero sí que es importante porque simplificaría su uso para futuras investigaciones.

Como vemos aún hay trabajo por hacer para desarrollar esta línea de investigación, por lo que espero fantásticas ideas y mejoras de aquella persona que continúe, de igual manera que también esperaré ansioso ver algún día resultados que confirmen nuestras hipótesis.

8 APÉNDICES

En esta sección se añadirá la documentación complementaria requerida para garantizar una comprensión completa del proyecto.

8.1 Guía original del Trabajo Fin de Título

El entrenamiento de redes convolucionales para la segmentación y clasificación de puntos en nubes de puntos está muy limitado por la escasez de *datasets* completamente etiquetados que permitan realizar un entrenamiento completo, más allá de los atributos relativos a la posición de los puntos. En este sentido, la necesidad de generar *datasets* sintéticos, completamente clasificados, que permitan el entrenamiento de las redes se vuelve imprescindible, si bien es crucial que la calidad de los *datasets* sintéticos sea similar a la de los *datasets* reales. En este TFM se propone la creación de un generador de entornos urbanos 3D, con alto grado de realismos, de manera que las escenas generadas con el mismo puedan ser utilizadas como datos de entrada de simuladores de sensores LiDAR. De esta manera se lograrían *datasets* sintéticos que podrían ser utilizados para realizar el entrenamiento de CNNS, permitiendo controlar los tipos de escenas usadas.

El trabajo incluirá no solamente la generación de estos *datasets* sino también el entrenamiento de algunas de las arquitecturas de CNNs más utilizadas en el ámbito de las nubes de puntos, con el fin de realizar una comparativa entre los resultados obtenidos por entrenamientos de las mismas con datos reales y sintéticos.

8.1.1 Conocimientos previos

Se requieren conocimientos de informática gráfica y manejo de CNNs.

8.1.2 Objetivos del TFM

- Integrar elementos de Inteligencia Artificial e Informática Gráfica para obtener productos de gran calidad.
- Obtener un generador de entornos urbanos, parametrizable, que permita la definición de dichos entornos de forma sencilla.

- Integrar los resultados con un simulador LiDAR externo para la obtención de *datasets* etiquetados.
- Iniciarse en los mecanismos de investigación en el ámbito de la informática, en concreto en realización de experimentación de calidad.

8.1.3 Metodología a desarrollar

Se utilizará una metodología incremental, dado que no se conocen con detalle los resultados a obtener. Además, se utilizará una metodología propia de la experimentación científica, con formulación de hipótesis, diseño de experimento, ejecución, obtención de resultados y análisis de los mismos.

8.1.4 Documentación y formatos de entrega

Se entregará la documentación en formato PDF y el código generado en un repositorio GIT o similar.

8.2 Manuales de usuario

8.2.1 Aplicación para generar fragmentos de ciudades

Esta aplicación no requiere instalación, pues es un proyecto de Unity 23.2.7.f1, simplemente deberá iniciarse con esa versión. Este proceso puede tardar un poco si es la primera vez que se abre. Tras iniciarlo, deberá ejecutarse la única escena que hay y pulsar el botón “GENERAR FRAGMENTOS”, tal y como se aprecia en la Ilustración 8.1.

Tras darle una vez habrá que esperar unos segundos hasta que la interfaz cambie y se empiecen a devolver fragmentos. Una vez termine de generar los fragmentos se puede volver a generar fragmentos (incluso cambiando los valores de configuración) tocando de nuevo el botón.

Ilustración 8.1: Aplicación fragmentadora abierto en Unity

A continuación se mostrará la información a tener en cuenta en formato de pregunta y respuesta.

8.2.1.1 ¿Cómo configurar el fragmentador?

Si se observa la jerarquía de objetos a la izquierda de la interfaz, se encontrará un objeto llamado “Configuracion”. Seleccionándolo aparecerá en el inspector un conjunto de atributos personalizables de manera que se permite:

- Indicar cuál es la ciudad que se fragmentará (objeto Ciudad de la escena).
- Indicar el *dataset* customizado. Esto servirá para obtener un archivo útil para el sensor LiDAR virtual que permita etiquetar objetos según el conjunto de un *dataset* concreto o personalizado.
- Indicar el número de puntos donde se generarán fragmentos. Atención: este número es posible que no equivalga al número de fragmentos que realmente se generen posteriormente. Esto se debe al funcionamiento interno y no es un error.
- Indicar la capa, que es “Carretera”. Si se desea cambiar se tiene que utilizar el nombre de la etiqueta del *dataset* ETIQUETAS_PROPIAS.
- Indicar el radio del fragmento.
- Indicar el incremento de altura máximo y mínimo, siendo estos la altura que varía respecto al suelo para que no colisione la malla con el centro del fragmento.
- Indicar la distancia de separación es la distancia entre centros de los fragmentos. Para que dos fragmentos cualesquiera no compartan nada, esta distancia debe ser mayor o igual al doble del radio del fragmento.
- Indicar la uniformidad de fragmentos marcando la cantidad de vacío que se permite en cada uno de ellos. Por ejemplo, un valor de 0.8 tiene poco vacío, mientras que un valor de 0.2 puede llegar a tener mucho vacío.
- Indicar si se desea previsualización LiDAR.

→ El resto de parámetros son atributos típicos utilizados para configurar un LiDAR.

8.2.1.2 ¿Cómo cambiar de ciudad para procesarla en la aplicación?

Para cambiar las ciudades que se segmentarán hay que ir a la carpeta:

...\Fragmentador de ciudades\Assets\CiudadesCompletas

Una vez ahí, deberán meterse todos los archivos que haya dentro de:

...\Fragmentador de ciudades\Assets\CiudadesCompletas\Otras ciudades~

Posteriormente se moverá la carpeta con las mallas y texturas a la siguiente dirección, junto con su archivo de metadatos correspondiente:

...\Fragmentador de ciudades\Assets\CiudadesCompletas

Esto se hace así porque Unity internamente identifica con un ID todas las mallas y objetos del proyecto, por lo que si hay demasiados, se queda sin IDs que asociar y falla. Para evitarlo lo que hacemos es guardar en la carpeta *Otras ciudades~* aquellas ciudades que no se estén utilizando. Atención: Unity dará problemas si se usa más de una ciudad a la vez, por lo que se recomienda que únicamente haya una en uso dentro de la carpeta *CiudadesCompletas*.

Una vez la ciudad esté ubicada en la carpeta de carga, carpeta *CiudadesCompletas*, se abre Unity y esperamos a que cargue. Este proceso puede conllevar varios minutos. Una vez esté Unity abierto, Tendremos que irnos a la única escena, ir al objeto Ciudad, y eliminar todos los objetos hijos que cuelguen de él en la jerarquía. Si en algún momento se borra el objeto de la ciudad, existe un prefab igual en la carpeta de prefabs.

Una vez limpio el objeto ciudad, se debe ir a la carpeta donde está la ciudad (*CiudadesCompletas*) y abrirla, algo que puede demorar varios minutos o más si la ciudad es grande. Una vez abierta, debe ponerse en modo de lectura y escritura tanto los modelos como las texturas (las texturas, si se desactiva el previsualizador, no son

necesarias de cambiar, aunque recomiendo mucho que se activen). La activación se hace desde el inspector con un tic con el nombre Read/Write.

Tras estar preparados, se seleccionan todos los modelos y se arrastran dentro del objeto Ciudad que hay instanciado en la escena. Finalmente nos aseguramos de que la configuración es correcta y tiene todos los objetos requeridos asociados.

8.2.1.3 ¿Cómo añadir nuevas texturas a las tablas?

Si una textura queda etiquetada como indefinida se mostrará por la terminal. En este caso, deberá incluirse en las tablas de la carpeta:

...\Fragmentador de ciudades\Tablas

Concretamente en *TABLA_CLAVES* y *TABLA_CONVERSION_MATERIALES* se debe de añadir el nombre de la textura, o un patrón representativo, por ejemplo si es "car_1", "car_2", "car_3", se debe usar "car". En la segunda columna, deberá indicarse qué etiqueta propia debe tener o que material debe tener, en función de la tabla que se esté modificando. Esto se puede hacer fácilmente desde Excel.

8.2.1.4 ¿Cómo incluir nuevos *datasets*?

Se pueden incluir nuevos *datasets* de referencia, aunque hay que tener en cuenta varias cosas:

- Esto se hace en el archivo siguiente fácilmente modificable desde Excel:

...\Fragmentador de ciudades\Tablas\TABLA_CONVERSION_ETIQUETASCSV

- Debe añadirse una nueva columna con los nombres de las etiquetas, y otra columna más con el mismo encabezado poniendo delante "Num_". Esta segunda columna incluirá el valor numérico de la etiqueta.
- Debe incluirse en el código de configuración, concretamente en el enumerado del *dataset* elegido, el nombre exacto de la cabecera de la columna de nombres del nuevo *dataset* (la primera columna que hemos añadido).

8.2.1.5 Recomendaciones adicionales para la visualización

Para visualizar los fragmentos se recomienda utilizar Blender. Puede tardar un poco en cargar el modelo.

Para nubes de puntos se recomienda utilizar CloudCompare o MeshLab.

8.2.2 Proyecto RepSurf

Si es la primera vez que se ve el proyecto, es importante mencionar que este ha sido modificado de forma que sea lo menos invasiva posible, adaptándolo para que pueda leer otros conjuntos de datos con otras etiquetas diferentes a las de S3DIS. A continuación se describirán los puntos clave a tener en cuenta.

En primer lugar deberá tenerse iniciado el entorno de Anaconda en el WSL, comando: *conda activate entorno-repsurf*. Tras esto, nos desplazamos hasta la carpeta de *segmentation*. Desde ella, los comandos de entrenamiento se ejecutan desde la terminal con:

sh ./scripts/custom/train_pointnet2.sh
sh ./scripts/custom/train_pointtransformer.sh
sh ./scripts/custom/train_repsurf_umb.sh

Los comandos de testeo son los siguientes:

sh ./*script*s/custom/test_pointnet2.sh
sh ./*script*s/custom/test_pointtransformer.sh
sh ./*script*s/custom/test_repsurf_umb.sh

Los parámetros de los comandos mostrados se cambian en los archivos de la carpeta siguiente, nunca desde el código:

RepSurf/segmentation/scripts/custom/

Los *datasets* se pondrán en estas carpetas, ya sea para entrenamiento o testeo en su carpeta correspondiente:

data/CUSTOM/trainval_fullarea
data/CUSTOM/testval_fullarea

→Atención: el proyecto está preparado para validación cruzada, por ello es que todo empieza por “Area_X”. Se debe indicar en la configuración qué archivos se usarán para la validación, por ejemplo, los que inician por “Area_1”.

El modelo resultante del entrenamiento con cada red está en las siguientes direcciones:

log/PointAnalysis/log/CUSTOM/pointnet2/checkpoints/model.ckpt
log/PointAnalysis/log/CUSTOM/pointtransformer/checkpoints/model.ckpt
log/PointAnalysis/log/CUSTOM/repsurf_umb/checkpoints/model.ckpt

Los logs y las visualizaciones aparecen en las siguientes direcciones, y estas últimas se pueden ver con MeshLab o con CloudCompare:

log/PointAnalysis/log/CUSTOM/pointnet2/logs
log/PointAnalysis/log/CUSTOM/pointnet2/visual
log/PointAnalysis/log/CUSTOM/pointtransformer/logs
log/PointAnalysis/log/CUSTOM/pointtransformer/visual
log/PointAnalysis/log/CUSTOM/repsurf_umb/logs
log/PointAnalysis/log/CUSTOM/repsurf_umb/visual

8.2.2.1 ¿Cómo se cambia el conjunto de datos?

Para ello deberá hacerse un archivo de labels.json semejante a los que hay ya creados y ubicarlo en la carpeta *custom_labels*. El archivo con el nombre de *labels.json* es el que está activo cuando se hace uso de los *script*s CUSTOM. Después deberá adaptarse el *script* de procesamiento, *convert2npy/convert_all_to_npy.py* para las nubes de puntos al formato requerido.

→Atención: es posible que el conversor no pueda convertir los datos con el formato que se requiera. Para más información mirar README ubicado en la *convert2npy*.

Luego se eliminarán los archivos anteriores de la carpeta de entrenamiento y testeo y se añadirán los nuevos datos, teniendo en cuenta lo explicado acerca del "Area_X" para determinar el conjunto de validación. Las carpetas son:

data/CUSTOM/trainval_fullarea
data/CUSTOM/testval_fullarea

8.2.2.2 ¿Cómo solucionar los problemas de VRAM?

Este tipo de problemas se pueden evitar en la mayoría de las ocasiones de varias formas:

- Reduciendo la densidad de las nubes de puntos.
- Reduciendo desde los *scripts* de entrenamiento el tamaño del: --batch_size, --batch_size_val, --workers. En cuanto a los parámetros de testeo se reducirá el tamaño de: --batch_size_test.

→Otra opción es cambiar a un servidor, aunque tocará reinstalar.

8.2.2.3 ¿Cómo instalar desde cero el proyecto?

Si hay que arrancar desde cero el proyecto habrá que seguir el proceso explicado en el repositorio original del proyecto RepSurf, pero con el proyecto modificado de esta investigación. En el README del proyecto modificado se encontrarán consejos adicionales para la instalación que podrían ser de utilidad.

8.2.3 *Script* para preparación de los datos

Este *script* se ubica en la carpeta *RepSurf\segmentation\convert2npy* y permite:

- La transformación a NPY con archivos PLY obtenidos del sensor LiDAR virtual.

- La transformación a NPY con archivos PLY obtenidos del *dataset* de Toronto-3D.
- Eliminar automáticamente los puntos repetidos (mismo xyz).
- La traducción de las etiquetas de un estándar a otro, aunque requiere de actualizar la función del código *leer_PLY* y *procesar_archivos*.
- La traducción de etiquetas de su formato de nombre a su formato numérico.
- La subdivisión del archivo procesado con --subdivide.
- La anulación del color, sustituyendo el RGB a valor 0,0,0 con el argumento: --nullrgb.
- Centrar las nubes automáticamente en el origen con el argumento: --center.

Este programa convierte todo lo que reconozca dentro de la carpeta *ArchivosSinConvertir* y lo vuelca en *ArchivosConvertidos*, borrando todo lo que esta última tuviera dentro.

→Para ver los ejemplos revisar el README.

9 DEFINICIONES Y ABREVIATURAS

Acc: Accuracy. Métrica que mide la precisión de un modelo, calculando la proporción de predicciones correctas sobre el total

ANN: redes neuronales artificiales (Artificial Neural Networks)

APM: gestión ágil de proyectos (Agile Project Management)

CNN: redes neuronales convolucionales (Convolutional Neural Network)

DL: Deep Learning. Conocido aprendizaje automático en español, se trata de una rama de la inteligencia artificial que utiliza redes neuronales profundas para aprender y procesar datos complejos.

EDT: Estructura De Trabajo. Esquema jerárquico que descompone un proyecto en tareas o entregables.

GAN: redes neuronales generativas adversarias (Generative Adversarial Networks)

GameObjects: son contenedores que guardan las diferentes piezas que son requeridas para hacer un personaje, una luz, un árbol, un sonido, etc. A estos además se les pueden asociar componentes para dotarlos con algún comportamiento

GNN: redes neuronales de grafos (Graph Neural Networks)

IoU: Intersection over Union. Métrica para evaluar la precisión de la superposición entre predicciones y resultados verdaderos en visión computacional

KDD: descubrimiento de conocimiento en bases de datos (Knowledge Discovery in Databases). Proceso de descubrir patrones y conocimiento útil en grandes bases de datos mediante técnicas de minería de datos

LiDAR: Light Detection And Ranging. Es una tecnología que utiliza láser para medir distancias y crear mapas tridimensionales precisos, útil para vehículos autónomos, topografía y arqueología

LSTM: memorias a largo y corto plazo (Long Short-Term Memory)

ML: Machine Learning

MLP: capas de multi-perceptrones (Multilayer Layer Perceptron)

MLS: sistema láser móvil (Mobile Laser System)

OA: Overall Accuracy. Métrica que evalúa el rendimiento general de un modelo, calculando la precisión total de todas las clases

PLN: Procesamiento del Lenguaje Natural. Rama de la IA que permite a las máquinas entender, interpretar y generar lenguaje humano

PMLC: ciclo de vida del proyecto (Project Management Life Cycle)

PNOA: Plan Nacional de Ortofotografía Aérea. Proyecto que genera ortofotos aéreas precisas de todo el territorio nacional, utilizadas en cartografía y geodesia

RAM: Random Access Memory

RNN: redes neuronales recurrentes (Recurrent Neural Networks)

SDLC: ciclo de vida del desarrollo de software (Software Development Life Cycle)

SPG: grafo de super puntos (Super Point Graph). Representación gráfica en visión por computadora donde puntos de interés se agrupan en "super puntos" para procesamiento

SSG: Single-Scale Grouping. Técnica de agrupamiento de puntos en visión computacional, enfocada en un solo nivel de escala

TPM: gestión tradicional de proyectos (Traditional Project Management)

VRAM: Video Random Access Memory

WSL: subsistema de Windows para Linux (Windows Subsystem for Linux)

XPM: gestión extrema de proyectos (Extreme Project Management)

10 BIBLIOGRAFÍA

[1] Y. I. H. Parish and P. Müller, “Procedural modeling of cities,” *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001*, pp. 301–308, 2001, doi: 10.1145/383259.383292.

[2] S. Greuter, J. Parker, N. Stewart, and G. Leach, “Real-time procedural generation of ‘pseudo infinite’ cities,” *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE ’03*, 2003, doi: 10.1145/604471.604490.

[3] K. R. Glass, C. Morkel, and S. D. Bangay, “Duplicating road patterns in South African informal settlements using procedural techniques,” *ACM International Conference on Computer Graphics, Virtual Reality and Visualisation in Africa*, vol. 2006, pp. 161–169, 2006, doi: 10.1145/1108590.1108616.

[4] “Procedural City Modeling.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.researchgate.net/publication/242374560_Procedural_City_Modeling

[5] “GitHub - mxgmn/WaveFunctionCollapse: Bitmap & tilemap generation from a single example with the help of ideas from quantum mechanics.” Accessed: Nov. 05, 2024. [Online]. Available: https://github.com/mxgmn/WaveFunctionCollapse

[6] “Inicio - Plan Nacional de Ortofotografía Aérea.” Accessed: Nov. 05, 2024. [Online]. Available: https://pnoa.ign.es/

[7] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, “Semantic3D.net: A new Large-scale Point Cloud Classification Benchmark,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, no. 1W1, pp. 91–98, Apr. 2017, doi: 10.5194/isprs-annals-IV-1-W1-91-2017.

[8] J. Behley *et al.*, “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences,” *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, pp. 9296–9306, Apr. 2019, doi: 10.1109/ICCV.2019.00939.

[9] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” *Proceedings of the IEEE Computer Society*

Conference on Computer Vision and Pattern Recognition, pp. 3354–3361, 2012, doi: 10.1109/CVPR.2012.6248074.

[10] X. Roynard, J. E. Deschaud, and F. Goulette, “Paris-Lille-3D: a large and high-quality ground truth urban point cloud dataset for automatic segmentation and classification,” *International Journal of Robotics Research*, vol. 37, no. 6, pp. 545–557, Nov. 2017, doi: 10.1177/0278364918767506.

[11] S. M. Iman Zolanvari *et al.*, “DublinCity: Annotated LiDAR Point Cloud and its Applications,” *30th British Machine Vision Conference 2019, BMVC 2019*, Sep. 2019, Accessed: Nov. 05, 2024. [Online]. Available: https://arxiv.org/abs/1909.03613v1

[12] W. Tan *et al.*, “Toronto-3D: A Large-scale Mobile LiDAR Dataset for Semantic Segmentation of Urban Roadways,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2020-June, pp. 797–806, Mar. 2020, doi: 10.1109/CVPRW50498.2020.00109.

[13] “ChatGPT.” Accessed: Nov. 22, 2024. [Online]. Available: https://chatgpt.com/

[14] Z. Liu *et al.*, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9992–10002, Mar. 2021, doi: 10.1109/ICCV48922.2021.00986.

[15] “DALL·E: Creating images from text | OpenAI.” Accessed: Nov. 22, 2024. [Online]. Available: https://openai.com/index/dall-e/

[16] “Sora | OpenAI.” Accessed: Nov. 22, 2024. [Online]. Available: https://openai.com/index/sora/

[17] “Introducing Whisper | OpenAI.” Accessed: Nov. 22, 2024. [Online]. Available: https://openai.com/index/whisper/

[18] “Suno.” Accessed: Nov. 22, 2024. [Online]. Available: https://suno.com/

[19] “ToonCrafter - Generador de Animaciones de Dibujos Animados por IA | ToonCrafter | ToonCrafter.” Accessed: Nov. 22, 2024. [Online]. Available: https://toon-crafter.com/es

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 77–85, Dec. 2016, doi: 10.1109/CVPR.2017.16.

[21] “Max Pooling Explained | Papers With Code.” Accessed: Nov. 05, 2024. [Online]. Available: https://paperswithcode.com/method/max-pooling

[22] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5100–5109, Jun. 2017, Accessed: Nov. 05, 2024. [Online]. Available: https://arxiv.org/abs/1706.02413v1

[23] A. Vaswani *et al.*, “Attention Is All You Need,” *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999–6009, Jun. 2017, Accessed: Nov. 05, 2024. [Online]. Available: https://arxiv.org/abs/1706.03762v7

[24] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, “Point Transformer,” *Proceedings of the IEEE International Conference on Computer Vision*, pp. 16239–16248, Dec. 2020, doi: 10.1109/ICCV48922.2021.01595.

[25] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” Oct. 2013, Accessed: Nov. 05, 2024. [Online]. Available: http://arxiv.org/abs/1310.4546

[26] L. Landrieu and M. Simonovsky, “Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 4558–4567, Nov. 2017, doi: 10.1109/CVPR.2018.00479.

[27] “City Generator by ProbableTrain.” Accessed: Nov. 05, 2024. [Online]. Available: https://probabletrain.itch.io/city-generator

[28] “blender.org - Home of the Blender project - Free and Open 3D Creation Software.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.blender.org/

[29] “Generador de ciudad 3D por procedimientos | Diseño de ciudades 3D para entornos urbanos.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.esri.com/es-es/arcgis/products/arcgis-cityengine/overview

[30] “RailClone 4.3.1 | Reference & Documentation.” Accessed: Nov. 05, 2024. [Online]. Available: https://docs.itoosoft.com/es/changelog/2021/05/10/railclone-4_3_1

[31] “Plataforma de desarrollo en tiempo real de Unity | Motor de 3D, 2D, VR y AR.” Accessed: Nov. 05, 2024. [Online]. Available: https://unity.com/es

[32] “The most powerful real-time 3D creation tool - Unreal Engine.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.unrealengine.com/en-US

[33] “Inicio - Epic Games.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.epicgames.com/site/es-ES/home

[34] "GitHub - drprojects/superpoint_transformer: Official PyTorch implementation of Superpoint Transformer introduced in [ICCV'23] 'Efficient 3D Semantic Segmentation with Superpoint Transformer' and SuperCluster introduced in [3DV'24 Oral] 'Scalable 3D Panoptic Segmentation As Superpoint Graph Clustering.'" Accessed: Nov. 05, 2024. [Online]. Available: https://github.com/drprojects/superpoint_transformer

[35] D. Robert, H. Raguet, and L. Landrieu, "Efficient 3D Semantic Segmentation with Superpoint Transformer," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 17149–17158, Jun. 2023, doi: 10.1109/ICCV51070.2023.01577.

[36] I. Armeni *et al.*, "3D Semantic Parsing of Large-Scale Indoor Spaces," *Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 1534–1543, Dec. 2016, doi: 10.1109/CVPR.2016.170.

[37] N. Varney, V. K. Asari, and Q. Graehling, "DALES: A Large-scale Aerial LiDAR Data Set for Semantic Segmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2020-June, pp. 717–726, Apr. 2020, doi: 10.1109/CVPRW50498.2020.00101.

[38] "GitHub - mathieuorhan/pointnet2_semantic: A pointnet++ fork, with focus on semantic segmentation of differents datasets." Accessed: Nov. 05, 2024. [Online]. Available: https://github.com/mathieuorhan/pointnet2_semantic

[39] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-January, pp. 2432–2443, Feb. 2017, doi: 10.1109/CVPR.2017.261.

[40] "Semantic3D - Data." Accessed: Nov. 05, 2024. [Online]. Available: https://www.semantic3d.net/view_dbase.php?chl=1

[41] A. Boulch, J. Guerry, B. Le Saux, and N. Audebert, "SnapNet: 3D point cloud semantic labeling with 2D deep segmentation networks," *Computers and Graphics (Pergamon)*, vol. 71, pp. 189–198, Apr. 2018, doi: 10.1016/j.cag.2017.11.010.

[42] "GitHub - hancyran/RepSurf: [CVPR 2022 Oral] Official implementation for 'Surface Representation for Point Clouds.'" Accessed: Nov. 05, 2024. [Online]. Available: https://github.com/hancyran/RepSurf

[43] H. Ran, J. Liu, and C. Wang, “Surface Representation for Point Clouds,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2022-June, pp. 18920–18930, May 2022, doi: 10.1109/CVPR52688.2022.01837.

[44] “Oracle VirtualBox.” Accessed: Nov. 05, 2024. [Online]. Available: https://www.virtualbox.org/

[45] “PyTorch.” Accessed: Nov. 05, 2024. [Online]. Available: https://pytorch.org/

[46] “InfoJobs - Bolsa de trabajo, ofertas de empleo.” Accessed: Nov. 06, 2024. [Online]. Available: https://www.infojobs.net/

[47] “LAS file format - Wikipedia.” Accessed: Nov. 06, 2024. [Online]. Available: https://en.wikipedia.org/wiki/LAS_file_format

[48] López-Ruiz, A., Ogáyar-Anguita, C.J., Segura-Sánchez, R.J., Enhancing LiDAR point cloud generation with BRDF-based appearance modelling, ISPRS Journal of Photogrammetry and Remote Sensing, 2024 (submitted, under revision)