UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS EN TOPOGRAFÍA, GEODESIA Y CARTOGRAFÍA

MÁSTER EN GEOMÁTICA APLICADA A LA INGENIERÍA Y A LA ARQUITECTURA

TRABAJO DE FIN DE MÁSTER

CLASIFICACIÓN DE NUBES DE PUNTOS OBTENIDAS CON SENSORES LIDAR AEROTRANSPORTADOS PARA LA IDENTIFICACIÓN DE OBJETOS MEDIANTE EL USO DE DEEP LEARNING

Madrid, septiembre, 2024

Alumno: Juan Ruiz Peralta

Tutores: José Juan Arranz Justel

Miguel Ángel Manso Callejo

UNIVERSIDAD POLITÉCNICA DE MADRID

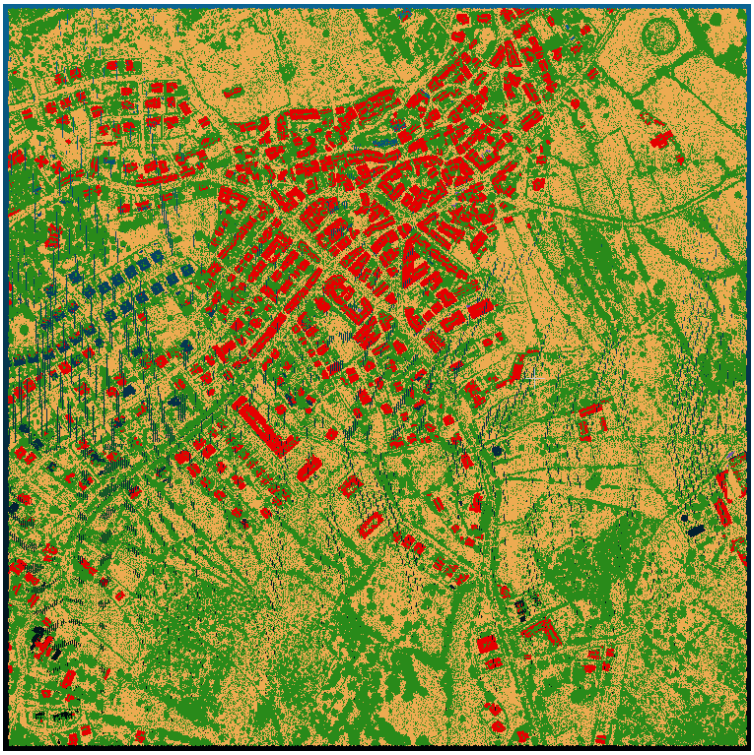
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS EN TOPOGRAFÍA,

GEODESIA Y CARTOGRAFÍA

MÁSTER EN GEOMÁTICA APLICADA A LA INGENIERÍA Y A LA ARQUITECTURA

TRABAJO DE FIN DE MÁSTER

CLASIFICACIÓN DE NUBES DE PUNTOS OBTENIDAS CON SENSORES LIDAR AEROTRANSPORTADOS PARA LA IDENTIFICACIÓN DE OBJETOS MEDIANTE EL USO DE DEEP LEARNING



Madrid, septiembre, 2024

Alumno: Juan Ruiz Peralta

Tutores: José Juan Arranz Justel

Miguel Ángel Manso Callejo

A los profesores que me han acompañado durante mi recorrido, por ser guía, motivación y consejo.

A Miguel Ángel y José Juan por las recomendaciones durante el desarrollo del proyecto.

A mi entorno, familia, compañeros y amigos, por el apoyo y lo vivido.

Resumen

El objetivo del presente proyecto es el estudio de las capacidades de clasificación de nubes de puntos LiDAR por parte de la red neuronal Myria3D. Dicho desarrollo se aplica a la segunda cobertura del PNOA LiDAR, con las nubes obtenidas y clasificadas por el IGN. Para garantizar su funcionamiento se necesita tratar las 1093 nubes de puntos utilizadas, adaptando su formato y uniendo la componente RGB con la información de infrarrojos en un único archivo.

Con el tratamiento del conjunto de datos se estudia y parametriza la red neuronal, obteniendo hasta una precisión del 86% en la clasificación inferida. Además, se muestran los resultados obtenidos con otras configuraciones, a pesar de incluir las pruebas más destacadas en el repositorio creado en github.

Los resultados muestran la importancia de una buena parametrización, además de la relevancia del volumen de datos a tratar durante los entrenamientos. A mayor cantidad más se evitan las generalizaciones y se aumenta la cantidad de puntos clasificados con precisión.

La utilización de redes neuronales como metodología de clasificación de nubes de puntos requiere unos conocimientos avanzados, además de un gran trabajo previo de clasificación manual con su pertinente adecuación de formatos.

Palabras clave: LiDAR, Nubes de Puntos, Clasificación, Deep Learning

Abstract

The aim of this project is to study the performance of Myria3D, deep learning library for semantic segmentation of LiDAR point clouds. This neural network is applied to the second PNOA LiDAR coverage, with the clouds obtained and classified by the IGN. To guarantee its operation, the complete dataset, composed of 1093 point clouds, must be processed. This involves modifying its format and merging the RGB component with the infrared information in a single file.

With the treatment of the dataset, the neural network should be studied and parameterized, reaching an accuracy of 86% in the inferred classification. Furthermore, it shows the results obtained with different configurations. Although, the most remarkable tests are included in the repository created in github.

The results show the importance of a good configuration, as well as the relevance of the volume of data to be processed during training. The greater the number, the more generalizations are avoided and the more points are accurately classified.

The use of neural networks as a point cloud classification methodology requires advanced knowledge, in addition to a significant amount of manual classification work with its corresponding format adaptation.

Keywords: LiDAR, Point Clouds, Classification, Deep Learning

Contenido

Bloque I INTRODUCCIÓN

Capítulo 1 Motivación

En las últimas décadas se ha dado una gran evolución de las tecnologías de captura de nubes de puntos, concretamente la realizada mediante sistemas LiDAR (Light Detection and Ranging), ofreciendo grandes beneficios en el manejo de datos geoespaciales. Con un gran abanico de aplicación, la creciente importancia de estas tecnologías ha creado la necesidad de buscar nuevas metodologías de tratamiento. Para poder llevar a cabo una buena manipulación de los datos, se debe aplicar una clasificación de las nubes de puntos LiDAR. Este proceso consiste en la correcta caracterización de cada punto de la nube a una de las clases de interés, pudiendo extraer la información geoespacial de relevancia.

El presente Trabajo de Fin de Máster se centra en el manejo de LiDAR aerotransportado, donde las principales capas a detectar son el terreno, la vegetación y los edificios. Generalmente, las nubes de puntos son tratadas con procesados manuales o semiautomáticos supervisados; sin embargo, requiere de una gran labor y estudio, cuando es una tarea que puede ser automatizada.

Con el reciente auge del aprendizaje automático, las redes neuronales y el deep learning, se puede trabajar en esta tarea de automatización, optimizando tiempos y simplificando el flujo de trabajo de las nubes de puntos.

La temática del proyecto nace con el objetivo de dotar de una continuación avanzada a mi Trabajo de Fin de Grado titulado “Optimización del Proceso de Ajuste, Clasificación y Obtención de Entidades a Partir de Datos LiDAR”. Dicho proyecto, referente a la titulación en Ingeniería Geomática y defendido en el año 2022, se centraba en el tratamiento de datos LiDAR incidiendo en la clasificación parametrizada y supervisada de las nubes de puntos obtenidas con esa tecnología.

Puntualizando la creciente relevancia de la inteligencia artificial, se decidió dar aplicación a los conocimientos adquiridos en el Máster en Geomática Aplicada a la Ingeniería y Arquitectura sobre la programación orientada al manejo de datos geoespaciales. Con ello, uniendo ambos puntos, la motivación principal de este Trabajo de Fin de Máster consiste en la exploración, utilización y alcance de redes neuronales aplicadas a la clasificación de nubes de puntos.

El desarrollo de este proyecto exige de conocimientos de programación en Python, permitiendo profundizar en ellos. Su necesidad de uso toma valor para realizar procesos específicos del tratamiento de los datos, para llevar a cabo verificaciones o para entender el funcionamiento de la propia red neuronal utilizada.

Capítulo 2 Objetivo

El objetivo prominente de este proyecto consiste en el estudio de la clasificación de nubes de puntos LiDAR aéreas mediante el uso de redes neuronales profundas. En específico, se va a utilizar una librería de deep learning de código abierto (Myria 3D), lo cual implica los siguientes objetivos secundarios:

1. Familiarizarse con el manejo de redes neuronales, además de conocer sus necesidades operativas, limitaciones y oportunidades.
2. Profundizar en el manejo de datos geoespaciales mediante el uso de la programación aplicada a tratamientos complejos.
3. Proponer un flujo de trabajo para clasificar nubes de puntos mediante el uso de redes neuronales.

Capítulo 3 Antecedentes

3.1 LiDAR

La tecnología LiDAR (Light Detection and Ranging) aplicada a la ingeniería geoespacial consiste en la composición de un sensor láser generalmente acompañado de sistemas GNSS e INS para dotar a los datos obtenidos de un correcto posicionamiento.

El sensor láser basa su funcionamiento en la emisión de un pulso de luz con una frecuencia determinada. Dicho pulso recorre una distancia hasta colisionar con un objeto, volviendo de esta forma al sensor, el cual registra el tiempo transcurrido. Con dicha cuantificación, se calcula la distancia que ha recorrido la luz, pudiendo ubicar el obstáculo con el que ha topado.

$$D = \frac{vt}{2}$$

D = Distancia sensor-objeto

v = Velocidad del pulso de luz emitido

t = Tiempo registrado entre la emisión y recepción del pulso

Una de las principales ventajas del LiDAR es la capacidad de registro de grandes cantidades de datos geoespaciales con una toma de datos poco diluida en el tiempo. Dicha información puede ser utilizada en diversos campos, desde su aplicación en la gestión de recursos forestales como su uso en la topografía. En el sector de la geomática, la utilización de las nubes de puntos LiDAR requiere de su clasificación en las entidades a estudiar, siendo generalmente terreno, vegetación y edificaciones, existiendo entidades más concretas según el objetivo del proyecto para el cual se ha obtenido la nube.

El proceso de clasificación se realizaba predominantemente de forma manual o parametrizada con supervisión, necesitando de aplicaciones concretas, es decir, de una licencia que limita el acceso a la gestión de estos datos.

3.1.1. Clasificación Manual

En los comienzos de la utilización de tecnologías LiDAR, la clasificación de nubes de puntos se realizaba manualmente, requiriendo de un experimentado en la materia para poder discernir correctamente las entidades representadas. Esto se llevaba a cabo en aplicaciones que permiten visualizar las nubes en 3D facilitando su manejo.

La clasificación, realizada de forma empírica, se basaba en la identificación visual, por lo cual los resultados estaban altamente condicionados por el operador, su experiencia y la dificultad del área capturada. Además, es una operación laboriosa, requiriendo de mucho tiempo para tratarlo correctamente, limitando su empleo a proyectos de pequeña envergadura, muy específicos y contando con operadores cualificados.

En el caso de utilizar este método de clasificación, se contaría con la ventaja de obtener resultados de gran precisión (de contar con personal cualificado).

3.1.2. Clasificación Parametrizada

Con el creciente uso de datos LiDAR para proyectos de ingeniería debido a su utilidad, se comenzaron a requerir tomas de datos más extensas con un mayor volumen a clasificar. Debido a esto, surgieron métodos configurables dentro de aplicaciones tales como los desarrollados por TerraSolid, los cuales podían ser adaptados al caso de estudio parametrizando las herramientas que ofrece.

Estos algoritmos de clasificación trabajan en función de las características de la nube, tales como la altura, valores de intensidad, rebotes y ángulos, juzgando cada punto respecto a los de su entorno, siempre en base a reglas preprogramadas. Con estas operaciones parametrizadas se consigue simplificar el tratamiento de las nubes de puntos dotándolo de agilidad. A su vez, se limita la atención del operador a la configuración previa y al control de la calidad resultante, pudiéndose ocupar una misma persona de varias nubes de forma simultánea.

A pesar de las virtudes de este tipo de clasificaciones, se cuenta con la desventaja de requerir una licencia para su uso, influyendo en el coste financiero de los proyectos a llevar a cabo.

3.1.3. Clasificación Automática

Actualmente, se cuenta con las ventajas del software libre y la programación, que, respaldados por algoritmos avanzados y redes neuronales, consiguen facilitar las tareas de clasificación, suprimiendo la necesidad de realizar un desembolso económico para manejar dichos datos geoespaciales.

Este tipo de clasificaciones requiere de amplios conocimientos de programación, además del funcionamiento de las redes neuronales profundas. A pesar de contar con repositorios online de libre uso, se debe adaptar, configurar e incluso modificar el código y procesos originales. En este Trabajo de Fin de Máster, centra el foco en la clasificación automática utilizando algoritmos de Deep Learning, por lo cual existen unos requerimientos propios, que se detallarán más adelante.

3.2 Deep Learning

El Deep Learning o aprendizaje profundo es una subdisciplina del Machine Learning, que a su vez está englobado en la Inteligencia Artificial. Surgió con la intención de simular el funcionamiento del cerebro humano mediante el uso de neuronas artificiales, las cuales forman redes neuronales artificiales en su conjunto. Dichas redes consisten en capas interconectadas donde cada una de ellas materializa un conjunto de neuronas. (LeCun, Bengio, & Hinton, 2015).

Cada neurona recibe una entrada, sobre la que realiza una operación matemática y emite una salida. El entrenamiento de la red neuronal consiste en ajustar los pesos de las conexiones entre capas de tal forma que el modelo aprende a manejar las entradas y con ello la información.

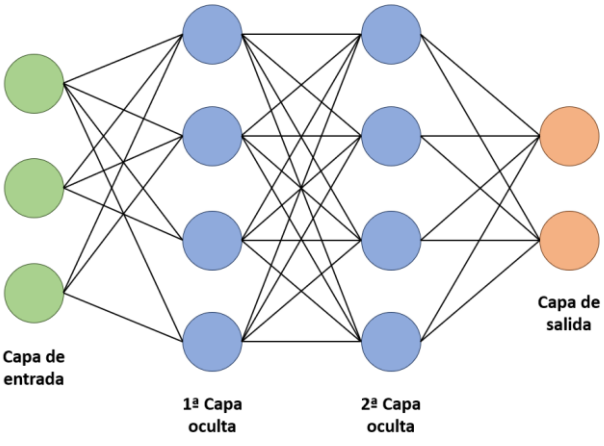


Ilustración 1: Estructura de una red neuronal profunda. (Imagen generada)

Durante los entrenamientos de la red, se obtiene un error mediante una función de pérdida, que muestra la equivocación producida en las predicciones comparándolas con las etiquetas originales. En el caso de este trabajo, se compara la clase asignada a cada punto con la capa a la que verdaderamente pertenece.

Este proceso implica una propagación de los datos hacia adelante generando una predicción. Con ello, se calcula su error, que se propaga hacia atrás para ir actualizando los pesos de las conexiones de la red. Con este algoritmo de retro-propagación, se consigue ajustar los resultados de la red (Theory of the backpropagation neural network, 1992).

3.2.1. Hiperparámetros de una Red Neuronal

Para lograr el correcto funcionamiento de una red neuronal, previamente a la fase de entrenamiento se debe configurar manualmente una serie de hiperparámetros. A continuación, se detallan algunos de gran importancia:

Función de Pérdida: Es un indicador de cómo el modelo realiza las predicciones. Su funcionamiento consiste en medir las variaciones entre la predicción y la etiqueta real, por lo tanto, este error resultante debe ser lo menor posible.

Optimización: Algoritmo encargado de ajustar los pesos de la red según los resultados de la función de pérdida.

Tasa de Aprendizaje: Controla cuanto se pueden ver afectados los pesos de la red al ser ajustados en el proceso de retro-propagación.

Épocas: Es la denominación a un ciclo completo de entrenamiento, es decir, que el modelo ha observado todo el conjunto de entrenamiento. En cada época, la red ajusta los pesos del modelo mediante la optimización.

Lotes: Los lotes o “batches”, dividen el conjunto de entrenamiento en subconjuntos, de tal forma que se ajusten los pesos del modelo tras la lectura de cada lote, evitando que sea únicamente una vez por época o después de la lectura de cada input.

Aumentación de datos: También denominado “data augmentation”, consiste en técnicas de aumentación de datos pudiendo aumentar el dataset mediante la aplicación de transformaciones geométricas. Generalmente, se aplican operaciones tales como giros, traslaciones o cambios de escalas.

3.2.2. División del Conjunto de Datos

A parte de conocer el funcionamiento de los parámetros explicados en el apartado anterior, se debe interiorizar el tratamiento de la información y su manejo para el entrenamiento del modelo. Durante este proceso se aplican verificaciones al aprendizaje, pero para ello, se debe dividir el dataset en tres conjuntos:

Conjunto de Entrenamiento: Dicha porción de los datos proporcionados a la red suele cuantificar el 70%, siendo la de mayor importancia, ya que es la utilizada para entrenar al modelo. Con estos datos se ajustan los pesos de las conexiones utilizando la función de pérdida y el optimizador, reduciendo así los errores de las predicciones.

Conjunto de Validación: Este conjunto también se utiliza durante el proceso de entrenamiento de la red, sin embargo, su función consiste en verificar el rendimiento del modelo, evaluando el conjunto tras cada época. Es necesario aislar estos datos del conjunto de entrenamiento para realizar una valoración objetiva. Suele ser el 10% de los datos y también es capaz de corregir la asignación de los hiperparámetros.

Conjunto de Testeo: Los datos reservados para este conjunto son utilizados tras el proceso de entrenamiento, para aplicar una evaluación final al modelo resultante. Al igual que el conjunto de validación, no debe haberse utilizado con anterioridad, evitando de esta forma falsas lecturas. Suele englobar el 20% del dataset y determina como se comportará el modelo conjugado.

3.2.3. Proceso de Entrenamiento

Conociendo los distintos conjuntos que dividen el dataset y el funcionamiento de los hiperparámetros, se procede a detallar el proceso de entrenamiento de la red neuronal. Como se ha explicado, se divide el dataset en los conjuntos de entrenamiento, validación y testeo, además, se determina del tamaño de los lotes. Tras esto, se lanza el entrenamiento de la red:

1) Se realiza la división en lotes del conjunto de entrenamiento.
2) Se trata el primer lote obteniendo una predicción.
3) Se compara la predicción obtenida con la etiqueta real, calculando la pérdida o el error cometido.
4) Se realiza la retro-propagación de los datos ajustando los pesos del modelo. Se tiene en cuenta la tasa de aprendizaje y la función de pérdida.
5) Se repite este tratamiento con todos los lotes hasta haber procesado todo el conjunto de entrenamiento, habiendo alcanzado una época completa.
6) Haciendo uso del conjunto de validación, se calcula la precisión del modelo resultante.
7) Se continúa iterando este proceso hasta finalizar el entrenamiento, tras lo cual, se realiza un último control de la calidad del modelo aplicándolo al conjunto de testeo.

Bloque II DATOS DE PARTIDA

Tras haber establecido los fundamentos de las tecnologías y procesos a aplicar, se procede a detallar los datos de partida, conformando la elección de la red neuronal a utilizar y las nubes de puntos a tratar para el entrenamiento.

Capítulo 4 Red Neuronal utilizada

Como se adelantaba en los objetivos de este Trabajo de Fin de Máster, se pretende aplicar una red neuronal obtenida de un repositorio online para desarrollar un flujo de trabajo enfocado a la clasificación de nubes de puntos. Tras evaluar algunas redes como PointNet, se ha escogido la librería "Myria 3D: Deep Learning for the Semantic Segmentation of Aerial Lidar Point Clouds" (Instituto Geográfico Francés, 2022).

Esta librería de Deep Learning está diseñada concretamente para realizar segmentaciones semánticas a nubes de puntos LiDAR aéreas. Se ha decidido aplicar por el foco tan concreto en la misma temática a la que se enfoca este TFM.

La librería ha sido desarrollada por el Instituto Geográfico Francés, con el objetivo de llevar a cabo la clasificación de la toma de datos LiDAR planificada para todo Francia en 2025, contando con una densidad de 10 puntos/m^2. La librería, creada en julio del año 2022, continúa bajo actualizaciones de desarrollo, aunque se ha utilizado una versión estable para el desarrollo de este Trabajo de Fin de Máster. Dicha versión cuenta con su última modificación conocida en junio de 2023.

Uno de los grandes desafíos del manejo de nubes de puntos en redes neuronales es la desestructuración causada por la componente tridimensional, donde los puntos muestran una dispersión en el espacio, dificultando el manejo de los datos. Myria3D se basa en las librerías de PyTorch e Hydra para su desarrollo, conformando su estructura a partir de una tercera librería, lightning-hydra-template (Ashleve, s.f.).

4.1 Flujo de Trabajo y Configuración

El Instituto Geográfico Francés aporta una página donde documenta la red neuronal, desarrollando procedimientos y referencias al paquete mientras que cita algunas de las modificaciones aplicables (Myria3D > Documentation, 2022). Se procede a incidir en su flujo de trabajo de forma teórica, sin explicar la instalación, cuestión que también aclara la documentación.

Recordando cómo el principal objetivo del desarrollo de Myria3D es la clasificación de datos LiDAR del Instituto Geográfico Francés, las explicaciones se centran en el manejo generalista de la información, obviando pasos detallados específicamente para sus nubes de puntos, tales como la adición de color a las mismas. A su vez, las sentencias de ejecución de cada paso se tratan en el Bloque III de este documento.

4.1.1. Parametrización

Para el correcto funcionamiento de la red se deben adaptar una serie de archivos de configuración en función del dataset utilizado y los objetivos de capas a identificar. Se van a indicar aquellos necesarios, mientras que a lo largo del capítulo 6 se destacarán las modificaciones realizadas para adaptar el código al dataset utilizado:

configs/dataset_description/20220607_151_dalles_proto.yaml

configs/experiment/RandLaNetDebug.yaml

Con esto configurado, todas las operaciones de la red neuronal se ejecutarán directamente desde el script de Python run.py, el cual se encuentra en la raíz del directorio instalado.

4.1.2. Preparación del Dataset

La primera operación consiste en dividir y cargar el conjunto de datos a utilizar en el entrenamiento, esto se realiza mediante ejecución de la tarea:

```
python run.py task.task_name=create_hdf5
```

Para ello se deben destacar los siguientes parámetros dentro del grupo “datamodule” de la configuración:

data_dir: Ruta al Directorio que almacena las nubes de puntos a utilizar en formato LAS, las cuales deben contar con código EPSG.

split_csv_path: Ruta al archivo CSV que especifica como cada nube del dataset pertenece al conjunto de entrenamiento, validación o testeo.

hdf5_file_path: Ruta donde se creará el archivo resultante a utilizar en los entrenamientos.

```
basename,split
PNOA_2016_MAD_412-4490_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4489_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4490_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4491_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4493_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4494_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4507_ORT-CLA-RGB-with-NIR-las14.las,train
PNOA_2016_MAD_413-4508_ORT-CLA-RGB-with-NIR-las14.las,val
PNOA_2016_MAD_413-4509_ORT-CLA-RGB-with-NIR-las14.las,test
PNOA_2016_MAD_413-4510_ORT-CLA-RGB-with-NIR-las14.las,test
```

Ilustración 2: Ejemplo del archivo CSV utilizado para dividir el dataset.

Con los datos divididos en subconjuntos, se procede a dividir cada nube de puntos a un tamaño determinado para facilitar su tratamiento. En la documentación se destaca el buen funcionamiento de teselar las nubes a un tamaño de 50x50 m.

Con la división, se realiza la primera operación, el preprocesado de todas las nubes de puntos teseladas a un formato HDF5. Para ahorrar futuros costes computacionales, esta primera ejecución reagrupa todo el dataset en un único archivo, simplificando las operaciones venideras, en las cuales únicamente se va a tener que señalar el directorio donde es guardado.

4.1.3. Entrenamiento del Modelo

Con el dataset reunido en un único archivo HDF5, se debe llevar a cabo otra modificación al código original. Al realizar la instalación de la librería, en el directorio raíz se facilita un archivo .env_example, el cual debe ser renombrado a .env, modificando lo siguiente:

> LOGS_DIR: Ruta al Directorio que almacena los registros, configuración y modelos resultantes de los entrenamientos.

En paralelo, se puede modificar la sección LOGGER, aunque requiere un registro en comet.ml, plataforma de machine learning que facilita estadísticas y rendimientos de la red.

Una vez configurada la dirección donde se guardarán los registros de los entrenamientos, se puede proceder al entrenamiento.

Dentro del entrenamiento cabe la posibilidad de aplicar las siguientes funciones:

Entrenamiento desde cero:

Consiste en realizar un entrenamiento de los datos almacenados en el HDF5 sin previa lectura. Se consigue obtener un modelo en formato CKPT con la información necesaria para clasificar nubes de puntos no tratadas. Se puede llevar a cabo ejecutando la tarea:

```
python run.py task.task_name=fit
```

Evaluación de un modelo:

Determinando el directorio donde se almacena el modelo, la red es capaz de evaluar su funcionamiento en base al Índice de Jaccard (IoU), comúnmente utilizado para evaluar los modelos de segmentación en redes neuronales. Tarea a ejecutar:

```
python run.py task.task_name=test
```

Ajuste de un modelo:

La red neuronal aporta la posibilidad de ajustar un modelo ya creado. Para ello, contando con un nuevo dataset en el HDF5, se puede administrar de tal forma que el modelo ajuste su funcionamiento al nuevo conjunto de datos considerando los pesos anteriormente guardados en base a entrenamientos pasados.

```
python run.py task.task_name=finetune
```

Los resultados de todos estos posibles procesos a ejecutar se facilitarán en el directorio LOGS_DIR, donde se incluye el modelo consecuente y un archivo denominado config.yaml con los hiperparámetros de la red.

4.1.4. Inferencia del Modelo

La inferencia es el proceso mediante el cual la red aplica un modelo entrenado a una nube de puntos concreta, calculando la segmentación semántica a partir de la distribución de pesos entre conexiones. Para su funcionamiento se debe contar con el modelo entrenado, la nube de puntos a clasificar y el archivo de configuración RandLaNetDebug.yaml, donde se encuentra la parametrización completa de la red.

La inferencia se puede llevar a cabo ejecutando la tarea:

```
python run.py task.task_name=predict
```

Con ello, las variables a destacar son:

src_las: Nube de puntos en formato LAS sobre la que se va a realizar la inferencia, destacando a su vez su directorio.

output_dir: Ruta donde se va a crear la nube de puntos con la clasificación inferida.

ckpt_path: Modelo en formato CKPT que se va a utilizar para la predicción de clases.

La inferencia puede ser realizada en varias nubes de puntos de forma simultánea, seleccionando el directorio donde se encuentren y "*.las" en vez del nombre concreto de una nube. A su vez, las nubes deben de contar con un sistema geográfica de referencia especificado en sus metadatos, de lo contrario, no se puede realizar la predicción.

Este procedimiento incluye los resultados en nuevas dimensiones dentro de la nube de puntos emitida en el directorio de salida. La principal, nombrada por defecto como "confidence", acumula la clase asignada para cada punto acorde a la parametrización. Por otro lado, se crea una segunda dimensión "entropy", indicador de confianza del resultado de la predicción de cada punto.

En añadido, se puede configurar la creación de una dimensión por cada capa de la clasificación. De esta forma, se calcularía el porcentaje de pertenencia de cada punto a la clase que se considere oportuno destacar. Un ejemplo de ello sería predict.interpolator.probas_to_save=[ground], donde se calcularía una dimensión añadida registrando la probabilidad de que cada punto pertenezca a la capa terreno.

Capítulo 5 Dataset utilizado

Una vez detallado el flujo de trabajo de la red neuronal utilizada, se procede a explicar la fuente del dataset utilizado para el entrenamiento de la misma. Como se detallaba anteriormente, Myria3D es una librería creada específicamente para clasificar la futura cobertura LiDAR del Instituto Geográfico Francés, además necesita que las nubes de puntos estén dotadas de información de Infrarrojos para su funcionamiento.

En una primera instancia se consideró la utilización del LiDAR de este mismo organismo, que, entre otras particularidades, almacena las nubes de puntos sin información RGB, la cual debe ser añadida en un procesamiento específico que facilita Myria3D. Sin embargo, para añadir valor y exigir un mayor manejo de la información geográfica, se escogió utilizar la segunda cobertura LiDAR del Instituto Geográfico Nacional (Centro de descargas del CNIG, s.f.).

Estos datos, a pesar de contar con el mismo propósito que los de su homólogo francés, tratan la información de forma distinta, exigiendo un procesado concreto para ser adaptado al uso con la red neuronal. Además, añade una dificultad al considerar un total de 1095 nubes de puntos, cuya información estaba dividida en dos, obligando a procesar 2190 inputs. Adicionalmente, cada nube utilizada cuenta con una cantidad aproximada de entre 2 y 3 millones de puntos, suponiendo un tamaño de entre 60 y 300 MB por archivo.

Las nubes de puntos del IGN cuentan con los requisitos mínimos de Myria3D: información RGB con Infrarrojo. Por otro lado, determinan el sistema de referencia mediante código EPSG, además de tener una clasificación cuya segmentación de clases es similar al Instituto Geográfico Francés:

Clase 1: Sin clasificar
Clase 2: Terreno
Clase 3: Vegetación baja
Clase 4: Vegetación media
Clase 5: Vegetación alta
Clase 6: Edificación

Bloque III METODOLOGÍA DE TRABAJO

Se ha elaborado un repositorio en github, donde se incluyen los scripts creados para el tratamiento del dataset. También se facilita el acceso al código de Myria3D y al centro de descargas del CNIG. Se puede acceder a partir del siguiente enlace:

https://github.com/JuanRuizPeralta/Processing-of-IGN-LiDAR-Point-Clouds-for-Training-with-Myria3D.git.

Capítulo 6 Adecuación del Dataset

El funcionamiento de la red neuronal implica una introducción de nubes de puntos con la parametrización requerida por la librería, por ello, al utilizar datos de una fuente distinta a la que se enfoca Myria3D, se deben adecuar las nubes de puntos descargadas. Este tratamiento, al realizarse en bloque a un gran número de inputs, se lleva a cabo mediante scripts de Python.

El IGN, a pesar de ofrecer información de infrarrojos en las nubes de puntos descargables, lo hace con ciertas peculiaridades que se deben conocer. Para una única nube de puntos, el IGN ofrece dos archivos separando las que aportan RGB de las que muestran el Infrarrojo, además de hacerlo en una versión LAS 1.2, cuestión también a tratar. Un ejemplo de ello es el siguiente, destacando también que las nubes están comprimidas:

PNOA_2019_CLM_NW_498-4478_ORT-CLA-RGB.laz
PNOA_2019_CLM_NW_498-4478_ORT-CLA-IRC.laz

Por ello, se muestran los 4 scripts creados para satisfacer el manejo de los datos previo a su uso en Myria3D:

1.Laz_to_Las.py
2.Version_las_14.py
3.Combine_las_files.py
4.List_files_in_directory.py

Por otro lado, se han elaborado 2 scripts para la evaluación de los resultados de las predicciones inferidas en las nubes de puntos.

5.Treatment_Inference.py
6.Statistics.py

6.1 Descompresión de las nubes de puntos

La primera operación a llevar a cabo es la descompresión, la cual se aplica a todas las nubes de puntos según se descargan a un directorio concreto. Con esto, destacando la ruta input y output se puede ejecutar el script, el cual se aporta en su totalidad en los anexos de este documento. Aun así, a continuación, se muestra el ejecutable principal de 1.Laz_to_Las.py:

```
for archivo in Datos_Partida:
    nombre_archivo, extension = os.path.splitext(archivo)

    if extension == '.laz':
        #Ruta de descompresión de la nube de puntos a formato las
        Ruta_Nube_las = os.path.join(Ruta_Output,nombre_archivo) + '.las'

        #Lectura del archivo
        Nube_bruta = laspy.read(os.path.join(Ruta_Base,nombre_archivo)+extension)

        #Descompresión
        Nube_las = laspy.convert(Nube_bruta)
        Nube_las.write(Ruta_Nube_las)
        print('Nube de puntos ',archivo,' descomprimida correctamente')
    elif extension == '.las':
        #Se va a duplicar la nube de puntos al directorio de Descomprimir, para
continuar con la ejecución del programa
        Ruta_Nube_las = shutil.copy(archivo, Ruta_Output)
        print('Nube de puntos ',archivo,' copiada correctamente')
    else:
        print('El formato de la Nube de Puntos ',archivo,' introducida no es
correcto, porfavor introducela en formato .las o .laz')
```

Ilustración 3: Extracto del código elaborado para la descompresión de las nubes de puntos (1.Laz_to_Las.py).

6.2 Edición de la versión LAS

Una vez descomprimidos todos los inputs descargados, se debe actualizar el formato LAS de 1.2 a 1.4. Esta operación es necesaria para poder crear una dimensión extra a la nube vinculando la información de infrarrojos (3.Combine_las_files.py). Así, destacando la carpeta donde se encuentran los inputs y donde se generarán los outputs, se actualiza el formato LAS con 2.Version_las_14.py.

Esta conversión, aunque aparenta ser sencilla, ha sido la más problemática, ya que suprimía información de referencia al ejecutarse, por este motivo se han debido sobrescribir ciertos parámetros de la cabecera, tales como el sistema de referencia, offsets, escalas y bounding box.

Igual que con el script anterior, se facilita el código completo en anexos, mostrando ahora la operación principal:

```
for nube in input_nubes:
    #Lee la nube original
    with laspy.open(nube) as las:
        header = las.header

        #Crea una nueva nube con la versión LAS 1.4 editando su nombre
        nombre = os.path.basename(nube)
        output = os.path.join(Ruta_Output, nombre.replace('.las', '-
las14.las'))
        header.version = Version(1, 4)
        #Reescribe metadatos que de no transferir, se perderían
        with laspy.open(output, mode='w', header=header) as writer:
            writer.header.system_identifier = header.system_identifier
            writer.header.generating_software = header.generating_software
            writer.header.date = header.date
            writer.header.offsets = header.offsets
            writer.header.scales = header.scales
            writer.header.min = header.min
            writer.header.max = header.max

            #Copia todos los puntos de la nube original
            for points in las.chunk_iterator(1_000_000):
                writer.write_points(points)
    print(nombre,"convertido correctamente a versión 1.4.")
```

Ilustración 4: Extracto del código elaborado para la edición de versión LAS de las nubes de puntos (2.Version_las_14.py).

6.3 Combinar RGB e Infrarrojo

El script 3.Combine_las_files.py realiza la operación más importante del tratamiento previo, es en la cual se combinan las dos nubes de puntos, logrando una única con RGB e Infrarrojos. Dentro de las nubes de puntos con infrarrojo, denominadas "IRC" en su nomenclatura, se almacena esta información en lo que sería la capa correspondiente a la coloración "ROJO" dentro del RGB.

Su funcionamiento comienza obteniendo los archivos que se encuentran en el directorio destacado, filtrando por nombre entre aquellos con datos RGB y sus homólogos con Infrarrojo. Así, se usarán las nubes RGB como base para añadir la capa de infrarrojo. Editando el nombre de la capa y aislando el nombre base de la nube, se emparejan las nubes RGB e IRC en un diccionario de este estilo:

{RGB_1:IRC_1, RGB_2:IRC_2}

Con el diccionario vinculando todas las nubes de puntos y mediante el uso de la librería Laspy (Laspy (Version 2.5.1) [Software], 2024) se realiza la lectura de las nubes para cada registro del diccionario. Tras asegurarse de que ambos archivos cuentan con el mismo número de puntos, se añade la dimensión "Infrarred" a la nube RGB, la cual guarda la información infrarroja de la nube IRC. Guardando la nube de puntos, el script procede a continuar con el bucle repitiendo el procesado a todas las nubes restantes.

A continuación, se facilita la operación principal, adjuntando el script completo en Anexos:

```
#Se procesa cada par de nubes emparejadas
for RGB_file, IRC_file in Nubes_emparejadas.items():
    RGB_path = os.path.join(Ruta_Input, RGB_file)
    IRC_path = os.path.join(Ruta_Input, IRC_file)
    output_file = os.path.join(Ruta_Output, RGB_file.replace("RGB", "RGB-with-NIR"))

    #Lectura de nube RGB con laspy
    las_rgb = laspy.read(RGB_path)

    #Lectura de nube IRC con laspy
    las_irc = laspy.read(IRC_path)

    #Se comprueba que ambas nubes tengan el mismo número de puntos
    assert len(las_rgb.points) == len(las_irc.points), f"El número de puntos en
{RGB_file} y {IRC_file} no coincide."
```

```
    #Se añade la dimensión "Infrarred" en la nube RGB
    las_rgb.add_extra_dim(laspy.ExtraBytesParams(name="Infrared", type=np.uint16,
description="nir"))

    #Se copia la dimensión "ROJO" de la nube IRC a la RGB, ya que es la que almacena
la inforamción de infrarrojo
    las_rgb['Infrared'] = las_irc['red']

    #Escritura de la nube resultante
    with laspy.open(output_file, mode='w', header=las_rgb.header) as writer:
        writer.write_points(las_rgb.points)

    print(f"Output guardado en: {output_file}")
```

Ilustración 5: Extracto del código elaborado para la combinación RGB y NIR de las nubes de puntos (3.Combine_las_files.py).

6.4 Listar nubes en directorio

Adicionalmente, se ha confeccionado un script para ayudar a la elaboración del dataset_split.csv, archivo necesario para destacar los conjuntos de entrenamiento, validación y testeo de la red. En este caso, debido a su simpleza, si se añade el script 4.List_files_in_directory.py completo a continuación:

```
import os

def listar_archivos(directorio):
    try:
        #Se crea una Lista con los archivos en la ruta especificada
        contenido = os.listdir(directorio)
        print(f"Archivos en el directorio '{directorio}':")
        for item in contenido:
            print(item)
    except Exception as e:
        print(f"Error al listar los archivos: {e}")

#Directorio que almacena las nubes de puntos tratadas
directorio =
'D:\\2.Myria3d\\prueba_IGN\\2.transformation\\3.Union_RGB_NIR'
listar_archivos(directorio)
```

Ilustración 6: Código elaborado para elaborar un listado de las nubes de puntos resultantes (4.List_files_in_directory.py).

6.5 Tratamiento de nube de puntos a predecir

Consiste en un tratamiento a la nube de puntos a la que se va a aplicar la inferencia, para poder estudiar los resultados obtenidos. En el apartado 7.2 se detalla cómo, para realizar los entrenamientos con Myria3D, se han modificado las clases originales de las nubes de puntos. Debido a esto, el script 5.Treatment_Inference.py realiza la misma edición para poder llevar a cabo una comparación más sencilla entre clases, la cual se estudia con el ejecutable 6.Statistics.py.

Al partir de nubes de puntos descargadas desde el centro de descargas del CNIG, se cuenta con una clase dedicada al solape entre nubes (clase 12), la cual se ha aislado durante los entrenamientos de la red, evitando empeorar los resultados. Por el mismo motivo, el script elimina todos los puntos pertenecientes a esta capa.

Replicando la modificación de clases originales del IGN, el script reagrupa la vegetación en una misma clase, dejando invariantes los puntos pertenecientes a terreno y edificación. Por otro lado, todos los puntos que no pertenezcan a ninguna de las 3 clases recién mencionadas se cambiarán a la clase 0 (sin clasificar).

De esta forma, se puede proceder a evaluar los resultados con el último script.

```
#Modificacion de la nube eliminando los puntos de solape
clases = las.classification
mask = clases != 12
clases_filtradas = clases[mask]
points = las.points[mask]

#Conversión a array de NumPy
clases_filtradas = np.array(clases_filtradas)

#Union de las clases de vegetacion en una única, la 5
clases_filtradas[np.isin(clases_filtradas, [3, 4])] = 5

#Union de todos los puntos que no sean terreno (2), vegetación (5) o
edificios (6) en la clase sin clasificar (1)
clases_invariantes = [2, 5, 6]
clases_filtradas[~np.isin(clases_filtradas, clases_invariantes)] = 1
```

Ilustración 7: Extracto del código elaborado para el tratamiento de la nube con la clasificación inferida (5.Treatment_Inference.py).

6.6 Estadísticas de la inferencia

El script 6.Statistics.py realiza la evaluación de la clasificación predicha en la nube de puntos tratada en el apartado anterior. Con las nuevas clases guardadas en la dimensión creada "confidence", el script compara punto a punto la precisión e intersección sobre unión (IoU) de los resultados.

Ambos estadísticos conforman una valoración detallada del conjunto de las predicciones realizadas. La precisión muestra la proporción de puntos clasificados correctamente sobre el total de predicciones realizadas para cada clase. A su vez, la IoU evalúa la cantidad de puntos acertados frente a los que se deberían haber clasificado según el IGN y la predicción.

Esto se lleva a cabo mediante el empleo de un dataframe de pandas (Pandas, s.f.), cuyo funcionamiento simula la creación de una tabla. Dentro de la misma, cada fila es un punto de la nube y las dos columnas creadas guardan la clase a la que pertenece, por un lado, la del IGN y por otro la predicha.

De esta forma, primero se calcula una precisión global verificando la coincidencia de clases para cada punto, y contabilizando las verdaderas. Así, se calcula el promedio y su porcentaje.

Por otro lado, la precisión se calcula para cada clase comparando los aciertos con la cantidad total de puntos identificados en esa misma clase:

$$Precisión = \frac{\text{nº pts acertados}}{nº\ pts\ identificados}$$

Para ello, se utiliza un bucle, seleccionando en cada iteración una de las clases disponibles según el IGN. Así, se seleccionan todos los puntos cuya clase inferida y aportada por el IGN coinciden con la que se está calculando. Además, se suman todos los puntos que, dentro de la clasificación inferida, pertenecen a la clase seleccionada.

Finalmente, para el cálculo de la intersección sobre unión se utiliza el mismo bucle. Dentro del mismo, se dividen los puntos acertados entre la unión. Dicha unión selecciona los puntos siempre que incluya la clase en el bloque del IGN o en la predicción.

$$IoU = \frac{\text{nº pts acertados}}{nº\ pts\ en\ unión}$$

```python
#Creación de un DataFrame de panda para analizar los puntos
df = pd.DataFrame({
    'IGN': clasificacion_IGN,
    'Inferido': clasificacion_Inferida
})

#Cálculo de la precisión global
glob_prec = (df['IGN'] == df['Inferido']).mean() * 100
print(f"\n    Precisión general de la predicción: {glob_prec:.2f}%")

#Cálculo de la precisión e IoU por cada clase inferida (sin clasificar,
terreno, vegetación y edificación)
clases = sorted(df['IGN'].unique())
for clase in clases:

    Ptos_acertados = ((df['IGN'] == clase) & (df['Inferido'] == clase)).sum()
    Ptos_Inferidos = (df['Inferido'] == clase).sum()

    if Ptos_Inferidos > 0:
        precision = Ptos_acertados / Ptos_Inferidos * 100
    else:
        precision = 0

    union = ((df['IGN'] == clase) | (df['Inferido'] == clase)).sum()

    if union > 0:
        iou = Ptos_acertados / union * 100
    else:
        iou = 0

    print(f"    Clase {clase}: Precision: {precision:.2f}%, IoU: {iou:.2f}%")
```

Ilustración 8: Extracto del código elaborado para la evaluación de la clasificación inferida (6.Statistics.py).

Capítulo 7 Parametrización de Myria3D

Una vez destacado el tratamiento previo del dataset para adecuarlo al funcionamiento de la red neuronal escogida, se debe detallar la parametrización utilizada para los entrenamientos e inferencias. En este apartado se destaca la que ha inducido los mejores resultados en las predicciones, estudio que se desarrolla en el capítulo 8.

Como se adelantaba en el apartado 4.1.1 de este documento, la parametrización de Myria3D recae sobre los siguientes archivos:

configs/dataset_description/20220607_151_dalles_proto.yaml

configs/experiment/RandLaNetDebug.yaml

Ambos archivos han sido creados por los desarrolladores de Myria3D para poder adaptarse a los datos de estudio según el uso que se le desee dar a la red. En este caso, se están adaptando a las nubes del IGN recién tratadas, centrando el foco en las siguientes clases:

Clase 1: Sin clasificar
Clase 2: Terreno
Clase 5: Vegetación
Clase 6: Edificación

Como se puede observar, se han reducido las capas originales del IGN para simplificar los entrenamientos, simplificando a su vez las capas a inferir. Esta preclasificación inicial se destaca en 20220607_151_dalles_proto.

En primera instancia, se lleva a cabo la reordenación de clases, donde se conjuga la nueva capa de “vegetación” agrupando su segmentación por alturas (vegetación baja, media y alta). A su vez, se reorganizan el resto de las capas de las que poder obtener algún punto mal clasificado para añadirlo a “sin clasificar”.

Por otro lado, se tratan dos capas con especial detalle, las clases 7 y 12, correspondientes a los puntos clasificados como ruido y solape respectivamente. Myria3D ofrece una opción para el manejo de datos a los cuales categoriza como “artefactos”. Estos puntos son aquellos que se deciden ignorar para el entrenamiento e inferencia, ya sea por haber

recibido una identificación errónea o por otros motivos. Myria3D se limitará a ignorar dichas clases y mantendrá los puntos que recibieron esa clasificación sin modificarla.

En este caso, se decide aplicarle este tratamiento al gran número de puntos clasificados como solape, que solo dificultarán el establecimiento de pesos del modelo, además de los puntos de ruido, que de añadirlos a la capa "sin clasificar" solo podrían dificultar la clasificación debido a su naturaleza aleatoria.

Para apartar dichas capas del flujo de trabajo de la red, durante el proceso de preclasificación se las debe asignar como capa destino la numerada en el puesto 65. Esto está establecido por la librería.

La preclasificación del dataset también requiere destacar las clases destino, el peso de cada una, las dimensiones de las nubes de puntos de entrada y el número de clases a clasificar. Esta sería la configuración completa de 20220607_151_dalles_proto:

```
_convert_: all
classification_preprocessing_dict: {3: 5, 4: 5, 0: 1, 7: 65, 8: 1, 9: 1,/
 10: 1, 11: 1, 12: 65, 13: 1, 14: 1, 15: 1, 16: 1, 17: 1, 19: 1, 20: 1,/
  21: 1, 22: 1, 23: 1, 32: 1}
classification_dict: {1: unclassified, 2: ground, 5: vegetation,/
 6: building}
class_weights: [0.3,0.7,1,5]
d_in: 9
num_classes: 4
```

Ilustración 9: Parametrización completa de 20220607_151_dalles_proto.yaml

Los pesos destacados se han establecido de forma heurística, de tal forma que se aporta más valor a determinadas capas para mejorar su clasificación. Si el dataset utilizado aporta clases con poca abundancia de datos, de no aplicarse dichos pesos, podrían pasarse por alto y empeorar los resultados. Para un correcto funcionamiento la suma de los pesos debe ser 7, preservando la escala de CELoss. En este caso, se ha decidido aumentar considerablemente el peso de la clase "Edificación", ya que es la que contaba con el menor número de puntos catalogados.

Habiendo configurado correctamente 20220607_151_dalles_proto.yaml se continúa con la parametrización de RandLaNetDebug.yaml. Este archivo es ejecutado durante el funcionamiento de la red neuronal sobrescribiendo todas las variables de control e hiperparámetros que se hayan decidido modificar. Por esto, solo guarda las variaciones realizadas al código, el cual compone su configuración dentro de diferentes archivos en

del directorio configs. Así, la configuración completa queda definida en el archivo generado tras lanzar el entrenamiento de la red.

Para mostrar la parametrización utilizada se van a ir detallando los hiperparámetros de la red acorde a las modificaciones destacadas en RandLaNetDebug.yaml, pero mostrando el archivo de configuración final. Esto se hace para poder detallar todas las configuraciones, además se anexa la configuración completa de RandLaNetDebug en los anexos.

Función de Pérdida:

Myria3D utiliza la función de pérdida "CrossEntropyLoss", elaborado por Pytorch para cuantificar la discrepancia entre las predicciones del modelo y las etiquetas reales. Esta función es particularmente útil cuando se entrena un dataset desbalanceado. Aporta una variable editable, "label_smoothing", que especifica el suavizado aplicado al calcular la pérdida, siendo 0 la cuantía por defecto (CrossEntropyLoss, s.f.).

```
model:
  criterion:
    _target_: torch.nn.CrossEntropyLoss
    label_smoothing: 0.0
```

Ilustración 10: Extracto de la configuración final correspondiente a la Función de Pérdida.

Optimización:

Se utiliza el optimizado "Adam" para llevar a cabo la actualización de pesos del modelo durante el proceso de entrenamiento. Utiliza la media y la varianza para adaptar la tasa de aprendizaje, variable que carga y es editable manualmente, como se muestra a continuación (Adam, s.f.).

```
model:
  optimizer:
    _target_: functools.partial
    _args_:
    - ${get_method:torch.optim.Adam}
    lr: ${model.lr}
```

Ilustración 11: Extracto de la configuración final correspondiente a la Optimización.

Tasa de Aprendizaje:

La tasa de aprendizaje, "lr" es uno de los hiperparámetros más relevantes de una red neuronal, controlando la cuantía en la que se pueden ver afectados los pesos de la red al ser ajustados. A su vez, se incluye el "lr_scheduler" que en función de cómo evoluciona el rendimiento del modelo puede reducir la tasa de aprendizaje. Utilizando "ReduceLROnPlateau", si no se está dando ninguna mejora en un número de épocas concretas (las determinadas por la variable "patience"), reduce la tasa de aprendizaje.

La variable "mode" se puede configurar con "min" reduciendo la tasa de aprendizaje cuando la cantidad monitorizada pare de descender, mientras que escogiendo "max" reducirá la tasa de aprendizaje cuando pare de aumentar. Por defecto se establece "min".

Por otro lado, "cooldown" cuantifica el número de épocas que espera tras reducir la tasa de aprendizaje para estudiar si volver a reducirla (ReduceLROnPlateau, s.f.).

```
model:
  lr_scheduler:
    _target_: functools.partial
    _args_:
    - ${get_method:torch.optim.lr_scheduler.ReduceLROnPlateau}
    mode: min
    factor: 0.5
    patience: 20
    cooldown: 5
    verbose: true
  lr: 0.004
```

Ilustración 12: Extracto de la configuración final correspondiente a la Tasa de Aprendizaje.

Aumentación de datos:

La aumentación de datos (data augmentation) consiste en una serie de transformaciones aplicadas al dataset con el objetivo de poder aumentarlo para mejorar la capacidad del modelo. Myria3D aporta tres configuraciones predefinidas en tres archivos YAML: heavy.yaml, light.yaml y none.yaml. Se puede escoger entre estas opciones o parametrizar la aumentación de datos dentro de RandLaNetDebug.

En este caso, los mejores resultados se obtuvieron al utilizar heavy.yaml, que implica la configuración light.yaml añadiendo una rotación. Es decir, se aplica un giro en "x" y otro en "y", con una probabilidad de 0.5 ser aplicado, además de una rotación de 180 º.

```yaml
datamodule:
  transforms:
    augmentations:
      x_flip:
        _target_: torch_geometric.transforms.RandomFlip
        _args_:
        - 0
        p: 0.5
      y_flip:
        _target_: torch_geometric.transforms.RandomFlip
        _args_:
        - 1
        p: 0.5
      RandomRotate:
        _target_: torch_geometric.transforms.RandomRotate
        _args_:
        - 180
        axis: 2
```

Ilustración 13: Extracto de la configuración final correspondiente a la Aumentación de datos.

Épocas:

Las épocas son el número de ciclos completos de entrenamiento, pudiéndose parametrizar el número mínimo y máximo obligando a la red a realizar un número de lecturas acotado del dataset. Con la configuración destacada a continuación, se cuenta con una duración aproximada de 6 minutos para el tratamiento de una época completa.

```yaml
trainer:
  min_epochs: 100
  max_epochs: 400
```

Ilustración 14: Extracto de la configuración final correspondiente a las Épocas.

Lotes:

Dividen el conjunto de entrenamiento en un número parametrizado de subconjuntos, ajustando los pesos del modelo tras la lectura de cada lote, sin esperar a la lectura completa del dataset.

```
datamodule:
  batch_size: 10
```

Ilustración 15: Extracto de la configuración final correspondiente a los Lotes.

A su vez, se cuenta con unos limitadores al número de lotes a utilizar durante los procesos de entrenamiento. Al contar con 1093 nubes de puntos, siendo 10 el tamaño del lote, se necesitaría parametrizar esta limitación en 110 para poder utilizar la totalidad del dataset durante los entrenamientos.

```
trainer:
  limit_train_batches: 100
```

Ilustración 16: Extracto de la configuración final correspondiente a la limitación de los Lotes.

División en Teselas:

Para facilitar la lectura de las nubes de puntos, la librería divide cada nube en teselas, cuya métrica es parametrizable, además del solape entre las mismas, pudiendo suavizar las predicciones.

En este caso, se ha configurado un tamaño máximo de 3000 m, ya que algunas de las nubes de puntos llegaban a contar con 2000 m de ancho. A su vez, la división se realiza en teselas de 50 m, existiendo un solape de 25 m entre cada una de ellas con sus colindantes, datos que aplican al entrenamiento y a la predicción.

```
datamodule:
  tile_width: 3000
  subtile_width: 50
  subtile_shape: square
  subtile_overlap_train: 25
  subtile_overlap_predict: ${predict.subtile_overlap}
predict:
  subtile_overlap: 25
```

Ilustración 17: Extracto de la configuración final correspondiente a la División en Teselas.

7.1 Configuración en CMD

No se ha detallado el proceso de instalación en la redacción de esta memoria para evitar perder el foco del estudio. Si se desea consultar, se recuerda que está disponible en la documentación de la librería: (IGNF. (s.f.). Installing Myria3D on WSL2. Myria3D tutorials. https://ignf.github.io/myria3d/tutorials/install_on_wsl2.html).

La instalación se ha hecho en Windows, cuando Myria3D está planteada para un ecosistema Linux. Por ello, se ha utilizado un Subsistema de Linux para Windows (WSL) y a su vez se ha creado un entorno (environment) propio al que se debe acceder para utilizar la red neuronal.

Las ejecuciones de la red se realizan mediante la CMD, mientras que todas las modificaciones al código se pueden llevar a cabo en un lector de código como Visual Studio Code. Previo a su uso, hay que acceder a la localización de descarga de Myria3D, además de señalar el directorio de registros (LOGS) de las ejecuciones de la red.

Para llevar a cabo este proceso, se deben ejecutar los siguientes comandos en la consola, adaptándolos a los directorios utilizados:

```
wsl
conda activate myria3d
cd /mnt/d/2.Myria3d/myria3d
export LOGS_DIR=/mnt/d/2.Myria3d/myria3d/1_Logs_Juan
```

Ilustración 18: Listado de comandos a computar en la CMD previo a las ejecuciones de la red.

7.2 Conversión a HDF5

Habiendo clarificado todas las operaciones requeridas para tratar el dataset, preparándolo para su funcionamiento en la red neuronal, se comienza con su primera ejecución: la división y carga del conjunto de datos a utilizar en el entrenamiento.

```
python run.py \
 task.task_name=create_hdf5 \
 logger=csv \
 datamodule.split_csv_path=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/dataset_split.csv \
 datamodule.data_dir=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/ \
 datamodule.hdf5_file_path=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/dataset_hdf5.hdf5
```

Ilustración 19: Ejecutable para la Conversión a HDF5

Como muestra el ejecutable, se deben especificar los directorios donde se almacenan el archivo dataset_split.csv (establecido a partir del script 4.List_files_in_directory.py) y el HDF5 a crear. Además, también se destaca el directorio que almacena todas las nubes de puntos tratadas y disponibles para el entrenamiento.

Se recuerda que el archivo dataset_split.csv registra la división de los conjuntos de entrenamiento (70%), validación (10%) y testeo (20%).

7.3 Entrenamiento del modelo

Tras la creación del archivo HDF5, con el dataset completo dividido en los conjuntos necesarios se procede al entrenamiento del modelo, donde se repite el mismo esquema de ejecución, aunque cambiando la tarea a "fit":

```
python run.py \
 task.task_name=fit \
 logger=csv \
 datamodule.split_csv_path=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/dataset_split.csv \
 datamodule.data_dir=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/ \
 datamodule.hdf5_file_path=/mnt/d/2.Myria3d/prueba_IGN/\
 2.transformation/3.Union_RGB_NIR/dataset_hdf5.hdf5
```

Ilustración 20: Ejecutable para el Entrenamiento del Modelo.

A su vez, se añade el parámetro "trainer.auto_lr_find", brindado por la librería y desarrollado por Pytorch Lightning. Consiste en un buscador automático de la tasa de aprendizaje que se computa antes del entrenamiento. Si se desea información concreta de su funcionamiento, se puede consultar la sección 3.3 del artículo de Leslie N. Smith: (Cyclical Learning Rates for Training Neural Networks, 2017).

Con el entrenamiento finalizado, se obtienen una serie de archivos guardados y ordenados por fecha en el directorio de registro. De todos ellos, se deben rescatar dos para realizar las futuras predicciones:

config.yaml

epoch_***.ckpt

El archivo de configuración config.yaml registra la parametrización de la red neuronal para el entrenamiento, mientras que epoch_***.ckpt es la época del entrenamiento que ha arrojado mejores resultados, lo que conforma el propio modelo entrenado.

7.4 Inferencia del modelo

Finalmente, con el modelo entrenado se puede proceder a la predicción de clases de una nube de puntos que no se haya utilizado anteriormente en el flujo de trabajo, para evitar falsos resultados. Por ello, se debe ejecutar el siguiente comando destacando la nube de puntos a predecir, la época (modelo) a utilizar y el número de GPUs para llevar a cabo el proceso (si es que se desea sobrescribirlo).

En paralelo, se puede calcular para cada punto el porcentaje de pertenencia a una clase concreta. Esto se realiza mediante la variable "predict.interpolator.probas_to_save", donde se destacan aquellas clases para las que se desea calcular esta estadística. Como se indica en el comando ejecutado, se ha decidido no guardarlo para ninguna clase:

```
python run.py \
 task.task_name=predict \
 logger=csv \
 predict.src_las=/mnt/d/2.Myria3d/prueba_IGN/2.transformation/\
 3.Union_RGB_NIR/PNOA_2016_MAD_416-4508_ORT-CLA-RGB-with-NIR-las14.las \
 predict.output_dir=/mnt/d/2.Myria3d/prueba_IGN/3.output/ \
 predict.ckpt_path=/mnt/d/2.Myria3d/prueba_IGN/3.output/1.Epoca_051.ckpt\
 predict.interpolator.probas_to_save=null
```

Ilustración 21: Ejecutable para la Inferencia del Modelo.

Bloque IV RESULTADOS Y DISCUSIÓN

Capítulo 8 Resultados

Con el tratamiento completo de los datos y la aplicación de la red neuronal, tal y como se detalla en el capítulo 7, se consigue obtener la nube de puntos con la clasificación inferida. De esta forma, se procede a estudiar los resultados obtenidos en la clasificación, para lo cual se utilizan los scripts 5.Treatment_Inference.py y 6.Statistics.py, explicados en el capítulo 6.

Este último ejecutable evalúa punto a punto las clases inferidas respecto a la clasificación que aporta el IGN, evaluando la precisión de la predicción. Conociendo la preclasificación a la que se han sometido las nubes del IGN, se muestra un esquema de pertenencia de las clases:

Tabla 1: Correspondencia de clases entre la clasificación inferida y la aportada por el IGN.

Clases	Clasificación IGN	Clasificación Inferida
Sin clasificar	Todas las clases menos: 2, 3, 4, 5 y 6	1
Terreno	2	2
Vegetación	3, 4 y 5	5
Edificación	6	6

De esta forma, ejecutando 5.Treatment_Inference.py y 6.Statistics.py se adecuan las clases del IGN a las recién inferidas, evaluando posteriormente la precisión y la intersección sobre unión (IoU) de cada clase. Durante el desarrollo del proyecto se han ejecutado varias computaciones del código variando los hiperparámetros. Así, se van a desarrollar las parametrizaciones con resultados más significativos, además de facilitar más configuraciones en el repositorio (https://github.com/JuanRuizPeralta/Processing-of-IGN-LiDAR-Point-Clouds-for-Training-with-Myria3D.git).

En primera instancia, se muestran los resultados de estas ejecuciones para posteriormente mostrar las variaciones en la configuración.

Tabla 2: Comparativa de resultados obtenidos considerando 3 parametrizaciones.

	1ª Config		2ª Config		3ª Config	
Clases	**Prec**	**IoU**	**Prec**	**IoU**	**Prec**	**IoU**
Sin clasificar	0.00%	0.00%	0.00%	0.00%	15.12%	0.12%
Terreno	84.67%	82.89%	83.95%	79.15%	77.71%	76.90%
Vegetación	86.10%	69.43%	78.30%	67.17%	77.84%	56.74%
Edificación	96.91%	66.75%	98.70%	35.02%	98.11%	29.69%
Prec Global	86.09%		82.17%		78.41%	

Las configuraciones han sido elegidas tras el estudio de parametrización llevado a cabo a la red, no solo por sus resultados sino por las modificaciones aplicadas. Si se desea, se puede consultar el repositorio github con el resto de las pruebas, consultando las configuraciones en "3.output/2.RandLaNetDebug" y los resultados en "3.output/1.Prediction".

Las diferentes configuraciones, detalladas en la Tabla 3, han inducido en cambios significativos a los puntos categorizados como edificación, demostrando ser la clase con mayor dificultad a inferir.

Tabla 3: Diferenciación de parámetros entre las configuraciones escogidas.

Parámetros	1ª Config	2ª Config	3ª Config
augmentations	heavy	light	light
limit_train_batches	100	110	60
auto_lr_find	true	true	true
min_epochs	100	50	100
max_epochs	400	400	400
subtile_width	50	25	100
subtile_shape	square	square	square
subtile_overlap_train	25	13	0
batch_size	10	10	10
lr	0.004	0.001	0.002
label_smoothing	0	0.5	0
subtile_overlap	25	13	0
class_weights (sin clasificar, terreno, vegetación, edificación)	0.3, 0.7, 1, 5	1, 1, 1, 4	0.3, 0.7, 1, 5

Con altas variaciones en la IoU, la 2ª configuración muestra cómo se han clasificado menos cantidad de puntos como edificación. Disminuyendo de un 66 % en 1a configuración a un 35 %, debido a la edición de los pesos escogidos para la diferenciación de clases. Al reducir el peso a la edificación pasando de 5 a 4, se ha mantenido la precisión de las identificaciones, sin embargo, se han incluido muchos menos puntos, empeorando la clasificación de los edificios. Al mantener una precisión muy pareja entre las dos configuraciones, pero con esta alta variación en el IoU de la clase edificación, se muestra la baja cantidad de puntos que incluye, dificultando su identificación.

También es destacable como con la 3ª configuración han aumentado los puntos en la clase "sin clasificar". Esto parece indicar una mayor precisión para apartar el ruido, puntos bajos y elementos que no pertenezcan al resto de clases, pero ha disminuido notablemente

la IoU para todas las clases. Esto puede justificarse con la limitación a 60 lotes, por la cual se está considerando aproximadamente el 50% del dataset disponible para los entrenamientos. Al forzar esa disminución de datos de estudio, Myria3D no puede reproducir un modelo con pesos ajustados, por lo que las predicciones son menos concluyentes. Esto también se incrementa con la disminución de la aumentación de datos, sin aplicar la rotación al dataset.

Cabe destacar cómo la limitación de épocas apenas ha afectado a la 1ª y 2ª configuración, ya que los modelos resultantes para cada parametrización fueron las iteraciones 51 y 33 respectivamente. Por otro lado, la 3ª configuración sí ha mostrado necesitar más lecturas para escoger el modelo, siendo la época 76 la seleccionada, esto puede deberse también a la limitación al dataset.

La tasa de aprendizaje (lr) escogida no debe afectar en gran medida a la formación del modelo, ya que en todas las configuraciones se ha utilizado el parámetro "auto_lr_find", buscando el valor que más se adapte al dataset.

Se procede a mostrar las clasificaciones obtenidas con cada configuración:

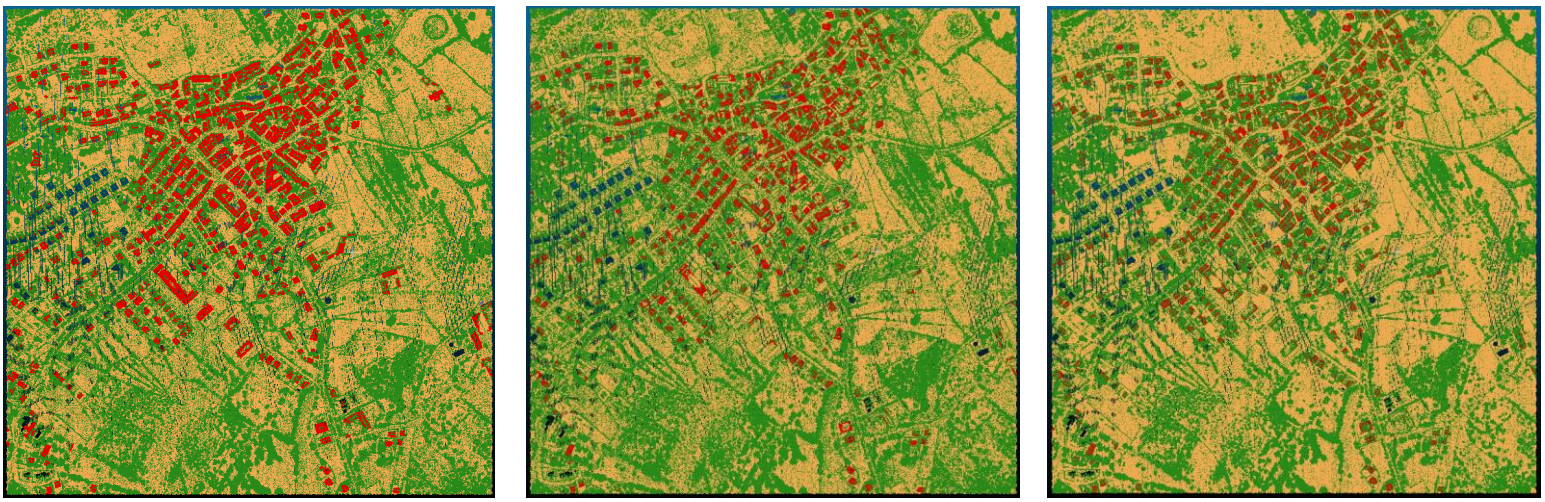


Ilustración 22: Clasificaciones inferidas tras la 1ª, 2ª y 3ª configuración respectivamente.

Adicionalmente, la tabla 4 muestra las estadísticas calculadas directamente por Myria3D tras los entrenamientos de las configuraciones escogidas.

Tabla 4: Estadísticas calculadas por Myria3D para cada parametrización.

		1ª Config	2ª Config	3ª Config
IoU	Global	0.385	0.378	0.386
	Edificación	0.086	0.056	0.049
	Terreno	0.834	0.833	0.832
	Sin Clasificar	0.0	0.0	0.0
	Vegetación	0.622	0.625	0.665
	Pérdida	0.333	1.1483	0.308

Estos resultados reiteran la dificultad en la identificación de la edificación y muestran cómo disminuye notablemente para las configuraciones 2 y 3. En paralelo el terreno y la edificación permanecen invariantes, mientras que se destaca una mayor pérdida en la 4ª configuración, a diferencia de la 3ª, que muestra un valor casi idéntico a la 1ª.

Este valor de pérdida disparado para la 2ª configuración implica que los resultados se deben a una configuración inapropiada, posiblemente por la disminución del peso indicado para la clase "edificación". A parte, el "label_smoothing" puede haber intervenido negativamente al suavizar el cálculo de la pérdida.

Se muestra una visualización comparativa entre la clasificación llevada a cabo por el IGN y la clasificación resultante tras la aplicación del modelo obtenido con la 1ª configuración.

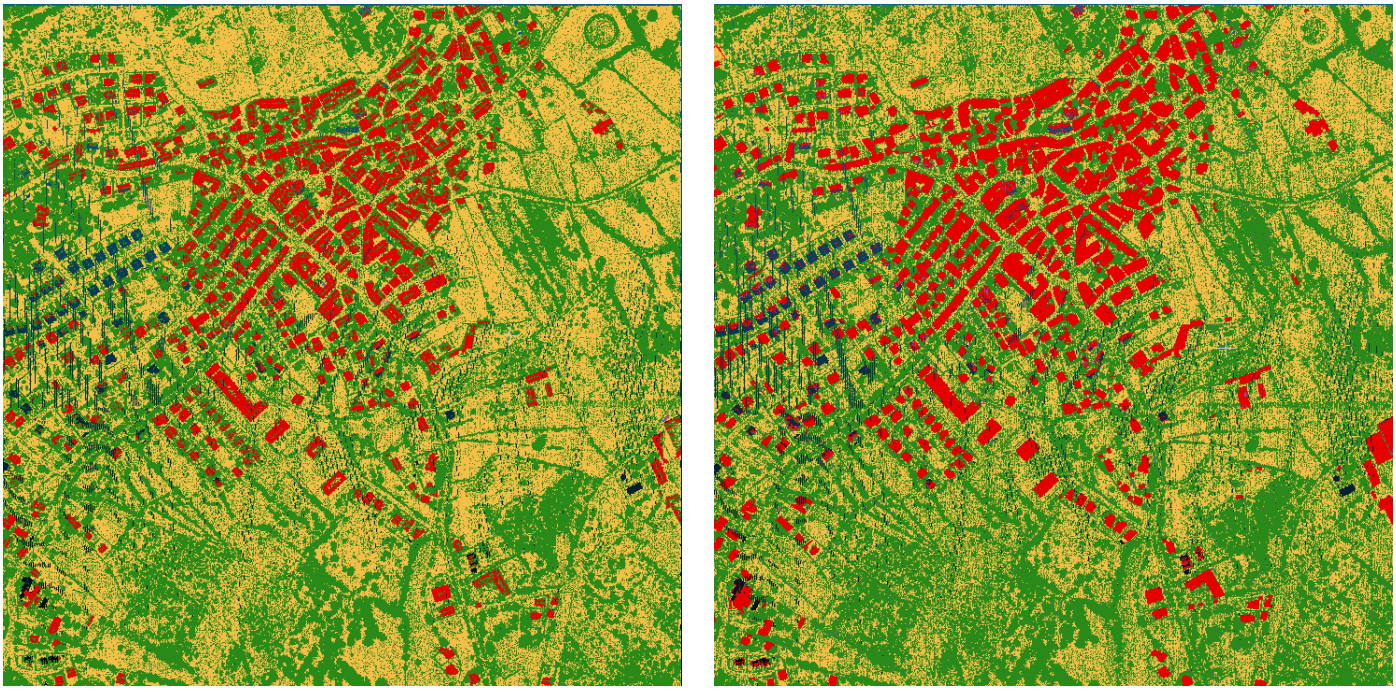


Ilustración 23: Comparativa visual entre la clasificación inferida y la aportada por el IGN, respectivamente.

Capítulo 9 Discusión

9.1 Conclusiones

Con la finalización del proyecto, se pueden despuntar una serie de conclusiones considerando los resultados obtenidos. En acuerdo a la limitación de lotes impuesta durante las pruebas, se destaca la importancia de contar un buen volumen de datos de estudio, que condicionará los resultados obtenidos.

Por otro lado, una correcta parametrización es altamente limitante y requiere conocimientos avanzados para poder lograrla. Además, también se necesitan llevar a cabo pruebas, reiteraciones y comprobaciones, que precisan de largos tiempos de entrenamiento.

Al aplicar Myria3D, la distribución de los pesos de las clases a inferir juega un papel fundamental. Se ha observado como al disminuir levemente el peso de la edificación, la red neuronal ha tenido mayores problemas para su identificación.

A su vez, la utilización de software libre exige un tratamiento detallado de los repositorios, los cuales suelen tener erratas o falta de explicaciones sobre procesos esenciales. En este caso, la librería utilizada, pese a ser realizada por un organismo importante y aportar documentación detallada, muestra los déficits comentados. También se debe añadir el detalle de ser un código que continúa en desarrollo.

El repositorio responde a una necesidad específica y ha requerido de una adaptación de los datos al variar su origen. Esto muestra la ventaja añadida de contar con la capacidad de edición de procesos y datos de entrada. Siempre que se cuente con los conocimientos pertinentes, el software libre muestra este valor añadido frente a las aplicaciones de pago.

Por otro lado, el proyecto actual ha utilizado nubes de puntos clasificadas por el IGN, lo cual exige un procesado previo que se ha omitido al no ser el foco de este estudio. Sin embargo, se debe tener en cuenta que los resultados obtenidos dependen del volumen de datos utilizado (1093 nubes de puntos). A pesar de estar planteándose una clasificación automática, requiere de una clasificación manual o parametrizada. Desde ese momento, se dispone de un punto de partida a la aplicación de la red neuronal.

9.2 Riesgos y Limitaciones

La base de datos usada para entrenar a la red neuronal representa una generalización de escenarios según los cuales se realizan las predicciones. Por este motivo, a mayor tamaño del dataset, se aplica una menor generalización, aumentando la precisión.

Este suceso se ha intentado apaliar seleccionando un dataset a detalle, incluyendo nubes de puntos con predominación urbana, rural y mixta, cubriendo las clases objetivo.

Como se detalla en capítulo 5, se han descargado 1094 nubes de puntos, aislando una de ellas para aplicar la inferencia final, lo que implica un dataset de 1093 nubes. Myria3D, originalmente formulada para llevar a cabo la clasificación de la próxima cobertura LiDAR de Francia, cuenta con todo el repositorio de datos del país para realizar los entrenamientos. El volumen de datos que se está manejando para este proyecto no es comparable con el volumen original utilizado, lo cual adelantaba un menor rendimiento de la red neuronal.

En paralelo, las clasificaciones utilizadas durante los entrenamientos han sido generadas por el IGN mediante tratamientos parametrizados, donde no se comprueban individualmente la totalidad de las nubes. Esto puede generar clasificaciones con puntos mal etiquetados, los cuales no se han corregido para el desarrollo de este proyecto, empeorando las predicciones.

A estas limitaciones, se le añade la densidad de puntos de las nubes descargadas desde el IGN. Con una densidad media mínima de 0,5 ptos/m^2 en función del lote, los datos utilizados para este proyecto también muestran otra desventaja operativa frente al planteamiento de Myria3D. Se recuerda que la red neuronal pretende satisfacer la densidad de 10 puntos/m^2 de la futura cobertura francesa.

9.3 Propuestas

A modo de líneas futuras de actuación, se podría continuar con la aplicación de Myria3D ampliando el número de nubes de puntos a considerar en los entrenamientos. De esta forma se mejorarían los recursos y se estudiaría la operatividad de la red para la clasificación de nuevas coberturas LiDAR del IGN en España.

A su vez, en respuesta a la limitación de la calidad de la clasificación del IGN, se puede llevar a cabo una corrección manual de las nubes de puntos utilizadas durante los entrenamientos, estudiando cuánto pueden mejorar las predicciones.

También, se puede componer un conjunto de datos desde su captación, comparando no solo la precisión de los resultados sino la utilidad en proyectos concretos. Dicha utilidad discretizaría, en función de las características del proyecto, si utilizar una clasificación parametrizada o automática.

En el uso de nubes de puntos aplicadas al campo de la Geomática, se pueden encontrar distintos pliegos de condiciones técnicas. Desde una solicitud de estudio de una zona reducida con muchos requisitos, hasta un área de gran tamaño donde se priorice la rapidez en las entregas. En este punto, analizar el riesgo de oportunidad de utilizar un proceso de clasificación u otro.

Para ello, también se debe estudiar el volumen de datos mínimo del que se debe disponer para entrenar correctamente a la red y cumplir con las condiciones de contrato. A su vez, el tiempo que se necesita para componer ese conjunto de datos, si no se dispone de nubes de puntos clasificadas previamente.

Por otro lado, actualmente se han utilizado nubes de puntos con infrarrojos, siendo de interés prescindir de este indicador. De esta forma, se podría aplicar el funcionamiento de la red neuronal a nubes de puntos aéreas obtenidas con el único uso de sensores LiDAR.

Bloque V BIBLIOGRAFÍA

Capítulo 10 Bibliografía

Ashleve, L. (s.f.). *lightning-hydra-template*. Obtenido de lightning-hydra-template: https://github.com/ashleve/lightning-hydra-template

Instituto Geográfico Francés. (2022). *Myria3D > Documentation*. Obtenido de https://ignf.github.io/myria3d/index.html#

Instituto Geográfico Francés. (2022). *Myria3D: Deep Learning for the Semantic Segmentation of Aerial Lidar Point Clouds*. Obtenido de https://github.com/IGNF/myria3d

Instituto Geográfico Nacional. (s.f.). *Centro de descargas del CNIG*. Obtenido de https://centrodedescargas.cnig.es/CentroDescargas/index.jsp

Laspy Development Team. (2024). *Laspy (Version 2.5.1) [Software]*. Obtenido de https://laspy.readthedocs.io/en/latest/index.html

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 436-444.

Pandas. (s.f.). *pandas.DataFrame*. Obtenido de https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

PyTorch. (s.f.). *Adam*. Obtenido de https://pytorch.org/docs/stable/generated/torch.optim.Adam.html#adam

PyTorch. (s.f.). *CrossEntropyLoss*. Obtenido de https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Pytorch Lightning. (s.f.). *Trainer auto-lr-find*. Obtenido de https://lightning.ai/docs/pytorch/stable/common/trainer.html#auto-lr-find

PyTorch. (s.f.). *ReduceLROnPlateau*. Obtenido de https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html#reducelronplateau

Smith, L. N. (2017). Cyclical Learning Rates for Training Neural Networks. *U.S. Naval Research Laboratory*, 10. Obtenido de https://arxiv.org/pdf/1506.01186

Theory of the backpropagation neural network. (1992). En R. Hecht-Nielsen, *Neural Networks for Perception.* Elsevier.

Bloque VI ANEXOS

Anexo A Referencias

A.1. Relación de figuras

A.2. Relación de tablas

Anexo B Scripts realizados para el manejo del dataset

A continuación, se exponen los scripts realizados para el manejo de las nubes de puntos descargadas del IGN para el manejo de la red neuronal Myria3D.

B.1. 1.Laz_to_Las.py

```
import laspy
import os
import shutil



#Para la correcta ejecución del programa se debe introducir la dirección donde se va
a trabajar
#En este directorio se debe introducir la nube de puntos a tratar
Ruta_Input = 'D:\\2.Myria3d\\prueba_IGN\\1.input'

#En este directorio se obtendrán las nube de puntos en formato .las
Ruta_Output = 'D:\\2.Myria3d\\prueba_IGN\\2.transformation\\1.Descompresion'


##############################################################################################
#Pensado para listar en jupyter y verificar formatos
##############################################################################################

#Listado de los archivos que hay en la carpeta de trabajo
Datos_Partida = os.listdir(Ruta_Input)
print("Archivos en la carpeta:")
print(Datos_Partida)

#Obtener las extensiones de los archivos (solo debería estar la nube en formato .las
o .laz)
Nombres = [os.path.splitext(archivo)[0] for archivo in Datos_Partida]

#Verificar los nombres de los archivos
print("\nNombre de los archivos en la carpeta:")
print(Nombres)

#Obtener las extensiones de los archivos (solo debería estar la nube en formato .las
o .laz)
Extensiones = [os.path.splitext(archivo)[1] for archivo in Datos_Partida]

#Verificar la extensión del archivo
print("\nExtensiones de los archivos en la carpeta:")
print(Extensiones)

##############################################################################################
for archivo in Datos_Partida:
```

```
    nombre_archivo, extension = os.path.splitext(archivo)

    if extension == '.laz':
        #Ruta de descompresión de la nube de puntos a formato las
        Ruta_Nube_las = os.path.join(Ruta_Output,nombre_archivo) + '.las'

        #Lectura del archivo
        Nube_bruta = laspy.read(os.path.join(Ruta_Input,nombre_archivo)+extension)

        #Descompresión
        Nube_las = laspy.convert(Nube_bruta)
        Nube_las.write(Ruta_Nube_las)
        print('Nube de puntos ',archivo,' descomprimida correctamente')
    elif extension == '.las':
        #Se va a duplicar la nube de puntos al directorio de Descomprimir, para
continuar con la ejecución del programa
        Ruta_Nube_las = shutil.copy(archivo, Ruta_Output)
        print('Nube de puntos ',archivo,' copiada correctamente')
    else:
        print('El formato de la Nube de Puntos ',archivo,' introducida no es
correcto, porfavor introducela en formato .las o .laz')
```

B.2. 2.Version_las_14.py

```
import os
import laspy
import glob
from laspy.header import Version

#Directorio con las nubes de puntos descomprimidas
Ruta_Input =
'D:\\2.Myria3d\\prueba_IGN\\2.transformation\\1.Descompresion'
#Directorio con las nubes de puntos resultantes
Ruta_Output =
'D:\\2.Myria3d\\prueba_IGN\\2.transformation\\2.Version_las'

#Comprueba que exista la ruta de salida, sino la crea
if not os.path.exists(Ruta_Output):
    os.makedirs(Ruta_Output)

#Busca todas las nubes .las en la carpeta de entrada
input_nubes = glob.glob(os.path.join(Ruta_Input, '*.las'))

for nube in input_nubes:
    #Lee la nube original
    with laspy.open(nube) as las:
        header = las.header

        #Crea una nueva nube con la versión LAS 1.4 editando su nombre
        nombre = os.path.basename(nube)
        output = os.path.join(Ruta_Output, nombre.replace('.las', '-
las14.las'))
        header.version = Version(1, 4)
        #Reescribe metadatos que de no transferir, se perderían
        with laspy.open(output, mode='w', header=header) as writer:
            writer.header.system_identifier = header.system_identifier
            writer.header.generating_software =
header.generating_software
            writer.header.date = header.date
            writer.header.offsets = header.offsets
            writer.header.scales = header.scales
            writer.header.min = header.min
            writer.header.max = header.max

            #Copia todos los puntos de la nube original
            for points in las.chunk_iterator(1_000_000):
                writer.write_points(points)
    print(nombre,"convertido correctamente a versión 1.4.")
```

B.3. 3.Combine_las_files.py

```
    las_rgb.add_extra_dim(laspy.ExtraBytesParams(name="Infrared",
type=np.uint16, description="nir"))

    #Se copia la dimensión "ROJO" de la nube IRC a la RGB, ya que es la
que almacena la inforamción de infrarrojo
    las_rgb['Infrared'] = las_irc['red']

    #Escritura de la nube resultante
    with laspy.open(output_file, mode='w', header=las_rgb.header) as
writer:
        writer.write_points(las_rgb.points)

    print(f"Output guardado en: {output_file}")
```

B.4. 4.List_files_in_directory.py

```
import os

def listar_archivos(directorio):
    try:
        #Se crea una Lista con los archivos en la ruta especificada
        contenido = os.listdir(directorio)
        print(f"Archivos en el directorio '{directorio}':")
        for item in contenido:
            print(item)
    except Exception as e:
        print(f"Error al listar los archivos: {e}")

#Directorio que almacena las nubes de puntos tratadas
directorio =
'D:\\2.Myria3d\\prueba_IGN\\2.transformation\\3.Union_RGB_NIR'
listar_archivos(directorio)
```

B.5. 5.Treatment_Inference.py

```
import laspy
import numpy as np

#Directorios de entrada y salida
Ruta_Input = "D:/2.Myria3d/prueba_IGN/3.output/PNOA_2016_MAD_416-
4508_ORT-CLA-RGB-with-NIR-las14.las"
Ruta_Output = "D:/2.Myria3d/prueba_IGN/3.output/PNOA_2016_MAD_416-
4508_ORT-CLA-RGB-with-NIR-las14_Inferencia.las"

#Lectura de nube con laspy
las = laspy.read(Ruta_Input)

#Modificacion de la nube eliminando los puntos de solape
clases = las.classification
mask = clases != 12
clases_filtradas = clases[mask]
points = las.points[mask]

#Conversión a array de NumPy
clases_filtradas = np.array(clases_filtradas)

#Union de las clases de vegetacion en una única, la 5
clases_filtradas[np.isin(clases_filtradas, [3, 4])] = 5

#Union de todos los puntos que no sean terreno (2), vegetación (5) o
edificios (6) en la clase sin clasificar (1)
clases_invariantes = [2, 5, 6]
clases_filtradas[~np.isin(clases_filtradas, clases_invariantes)] = 1


#Asignación de la nueva clasificación
points['classification'] = clases_filtradas

#Creación de una nueva nube
header = las.header
output = laspy.create(point_format=las.point_format,
file_version=las.header.version)
output.points = points
output.header = header

#Guardado de la nube
output.write(Ruta_Output)
print(f"Nube de puntos procesada en {Ruta_Output}")
```

B.6. 6.Statistics.py

```
import laspy
import pandas as pd

Ruta = "/mnt/d/2.Myria3d/prueba_IGN/3.output/1.Prediction/"
Nombre = "14.Inferencia.las"
print("\nEstadísticas de la nube:", Nombre)
Ruta_Nube = Ruta + Nombre

#Lectura de nube con laspy
Nube = laspy.read(Ruta_Nube)

#Aislar la clasificación manual del IGN de la inferida
clasificacion_IGN = Nube.classification
clasificacion_Inferida = Nube.confidence

#Creación de un DataFrame de panda para analizar los puntos
#Más información en https://pandas.pydata.org/pandas-
docs/stable/reference/api/pandas.DataFrame.html
df = pd.DataFrame({
    'IGN': clasificacion_IGN,
    'Inferido': clasificacion_Inferida
})

#Cálculo de la precisión global
glob_prec = (df['IGN'] == df['Inferido']).mean() * 100
print(f"\n    Precisión general de la predicción: {glob_prec:.2f}%")

#Cálculo de la precisión e IoU por cada clase inferida (sin clasificar,
terreno, vegetación y edificación)
clases = sorted(df['IGN'].unique())
for clase in clases:
    Ptos_acertados = ((df['IGN'] == clase) & (df['Inferido'] ==
clase)).sum()
    Ptos_Inferidos = (df['Inferido'] == clase).sum()

    if Ptos_Inferidos > 0:
        precision = Ptos_acertados / Ptos_Inferidos * 100
    else:
        precision = 0

    union = ((df['IGN'] == clase) | (df['Inferido'] == clase)).sum()
    if union > 0:
        iou = Ptos_acertados / union * 100
    else:
        iou = 0

    print(f"    Clase {clase}: Precision: {precision:.2f}%, IoU:
{iou:.2f}%")
```