industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Análisis e implementación de sistemas de detección y clasificación en nube de puntos basados en redes neuronales para su aplicación en conducción autónoma

TRABAJO FIN DE MÁSTER

MÁSTER EN INGENIERÍA INDUSTRIAL

Autor: **Eduardo Salas Fernández**
Director: Pedro Javier Navarro Lorente

Cartagena, julio de 2020

Universidad Politécnica de Cartagena

Resumen

Este trabajo de fin de máster aborda el análisis de las redes neuronales de detección y clasificación de objetos en nubes de puntos en el ámbito de la conducción autónoma. Se plantea el entrenamiento de una red neuronal de aprendizaje profundo para la detección de automóviles en un entorno de conducción autónoma. Para evaluar el modelo de detección se realizarán ensayos donde se simularán escenarios de conducción reales y con condiciones ambientales particulares.

Abstract

This final master project addresses the analysis of object detection and classification neural networks in point cloud data in the field of autonomous driving. The training of a deep learning neural network for car detection in an autonomous driving environment is proposed. To evaluate the detection model, test will be carried out where real driving scenarios and particular environmental conditions will be simulated.

Índice de contenidos

CAPÍTULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

En los últimos años las redes neuronales convolucionales nos han permitido disponer de los medios para la identificación y clasificación de elementos mediante las imágenes obtenidas por cámaras, análogamente a los procesos cognitivos que realizamos los seres humanos. Mientras que las redes neuronales aplicadas a imágenes han tenido un enorme desarrollo, lo que permite su aplicación en casi cualquier entorno, las redes aplicadas a nubes de puntos presentan por lo general mayores complicaciones y en su mayoría rinden con velocidades menores, sin embargo, recientemente han comenzado a aparecer arquitecturas que permiten el análisis de los datos a una mayor velocidad.

A lo largo de los años, el ser humano se ha valido del ingenio para facilitar y con el tiempo automatizar buena parte de las tareas de su día a día, así como sus métodos de trabajo y producción. Esta automatización no solo libera la carga de trabajo de los seres humanos y ahorra tiempo, sino que además permite mejorar la calidad de estas tareas, así como un importante aumento de seguridad en las mismas. Uno de los principales retos tecnológicos a los que se ha enfrentado la ingeniería es la automatización de la conducción, lo que conocemos como conducción autónoma.

Aunque tanto las carreteras como los vehículos que circulan por ellas han supuesto una revolución en los desplazamientos, también se han convertido en uno de los principales riesgos a los que se expone el ser humano de manera frecuente. Los vehículos autónomos no solo suponen una automatización del proceso de conducción, sino un gran aumento de seguridad en el propio vehículo. Tanto la mejora de los tiempos de reacción como la alta sensorización de estos vehículos les permite tener un mayor control de la información en el entorno de circulación, y por consiguiente una mayor capacidad de reacción ante situaciones peligrosas.

Una de las piezas clave de los vehículos inteligentes es la gestión de la información aportada por sus sensores. Mientras que la aplicación de determinados algoritmos como los filtros de Kalman permiten obtener un posicionamiento más preciso del vehículo, las redes neuronales de aprendizaje profundo han supuesto un gran paso adelante para la gestión de información tan compleja como pueden ser las imágenes de las cámaras del vehículo. Las redes neuronales convolucionales tienen la capacidad de extraer características a diferentes niveles y aportar una identificación y clasificación de elementos de interés en los datos de entrada, como la presencia y localización de otros vehículos o peatones. Así mismo esta tecnología es aplicable a otro tipo de datos como son las nubes de puntos aportadas por sensores como los LIDAR, cuya información no es sustituyente de la información aportada por las cámaras, pero si complementaria al poder captar cierta información que mediante imágenes es difícil obtener.

1.2 Objetivos

El objetivo principal del proyecto es el estudio de arquitecturas de aprendizaje profundo y de su aplicación en datos procedentes de vehículos autónomos, particularizando dicha aplicación en redes neuronales de detección de objetos adaptadas para sensores LIDAR. Para lograr dicho fin se han definido una serie de objetivos:

1) Estudio del estado del arte sobre las técnicas de detección de objetos en nubes de puntos particularizando en el caso de conducción autónoma.

2) Selección de la arquitectura de red neuronal de aprendizaje profundo para datos procedentes de LIDAR más adecuada al escenario de conducción autónoma.

3) Desarrollo de software que permita el entrenamiento y la simulación de la implementación de la red de aprendizaje profundo en datos procedentes de otros vehículos autónomos.

4) Realización de ensayos que permitan analizar y verificar el funcionamiento del modelo de aprendizaje profundo desarrollado.

1.3 Descripción de los capítulos del proyecto

La estructuración de este proyecto de fin de estudios se divide en los capítulos descritos a continuación:

Capítulo 1: Introducción y objetivos

En este capítulo se describe el proyecto, así como sus objetivos y la estructuración para llevarlos a cabo.

Capítulo 2: Estado del arte

Se realizará un estudio del estado del arte acerca de las tecnologías que afectan a las redes neuronales, profundizando en los casos particulares de redes neuronales de aprendizaje profundo de detección en nubes de puntos. Se hará un breve repaso por el estado de la conducción autónoma y se presentarán algunos conceptos básicos de interés.

Capítulo 3: Selección y análisis de la arquitectura

El objetivo final de este capítulo es la selección de la arquitectura de aprendizaje profundo más adecuada al escenario de conducción autónoma. Se presentarán unos criterios de selección adecuados y se analizará en profundidad la arquitectura seleccionada.

Capítulo 4: Diseño y programación

En este capítulo se aborda el diseño y la programación del software que permita la ejecución de la red neuronal de aprendizaje profundo seleccionada. También se realizará el planteamiento de unos criterios de diseño básicos y un análisis de riesgos potenciales para el desarrollo.

Capítulo 5: Entrenamiento y ajuste del modelo

Se expondrá en este capítulo el proceso de entrenamiento de la red neuronal seleccionada con diferentes configuraciones. Se seleccionará la configuración óptima y se realizará un ajuste de parámetros del modelo.

Capítulo 6: Evaluación y discusión de los resultados

El modelo entrenado y ajustado en el capítulo anterior será empleado para el procesado de datos procedentes de otros vehículos autónomos. Se realizará una evaluación del modelo mediante los parámetros de evaluación más adecuados y se analizarán las principales causas de error. Finalmente se realizará una discusión de los resultados obtenidos en este capítulo.

Capítulo 7: Conclusiones y trabajos futuros

Se analizará el trabajo realizado y el cumplimiento de los objetivos del proyecto. Se discutirán los trabajos que podrían realizarse a partir de los resultados obtenidos.

Capítulo 8: Bibliografía

Este capítulo contiene las fuentes de todas las referencias empleadas en el presente trabajo de fin de estudios.

CAPÍTULO 2

ESTADO DEL ARTE

2.1 Introducción

En este capítulo se trata el estado actual de la tecnología que afecta a las redes neuronales. Se expondrán algunos conceptos básicos, pasando por el aprendizaje automático y el aprendizaje profundo, posteriormente se abordará una comparación entre los datos aportados por diferentes tecnologías y se tratarán diferentes arquitecturas de redes neuronales aplicadas en el aprendizaje y predicción en nubes de puntos. Finalmente se hará una revisión de las principales herramientas de software disponibles para la implementación de modelos de aprendizaje profundo.

2.2 Conceptos básicos

En esta sección se abordarán los conceptos principales en los que se basa el presente trabajo. Se tratará el aprendizaje automático como parte del campo de la inteligencia artificial, seguido del aprendizaje profundo y finalmente se abordarán las redes neuronales convolucionales.

2.2.1 Aprendizaje automático

El aprendizaje automático, generalmente conocido como “Machine Learning”, es una rama del campo de la inteligencia artificial, estando enfocado dicho campo en el diseño de algoritmos capaces de la resolución de problemas a través del aprendizaje [1]. El Machine Learning emplea algoritmos que tienen la capacidad de realizar un análisis de los datos y un aprendizaje de estos con el fin de realizar tareas como la clasificación y la predicción. En [2] se puede encontrar una extensa documentación acerca de este subcampo.

Los algoritmos de Machine Learning pueden ser divididos en 3 agrupaciones:

- Aprendizaje supervisado: Los algoritmos de esta tipología requieren un conjunto de datos previamente etiquetado, con los que es posible llevar a cabo un entrenamiento del algoritmo de aprendizaje supervisado. Una vez realizado el entrenamiento de dicho modelo, es posible llevar a cabo una clasificación de nuevos datos con las etiquetas empleadas en el aprendizaje. En la figura 2.1 puede apreciarse un esquema en el que se muestra el funcionamiento de este tipo de modelos. Un ejemplo de aprendizaje supervisado es el entrenamiento de un modelo de clasificación de libros en géneros literarios a partir del conjunto de palabras empleadas en los mismos, donde dichos géneros literarios serían las etiquetas de los datos.

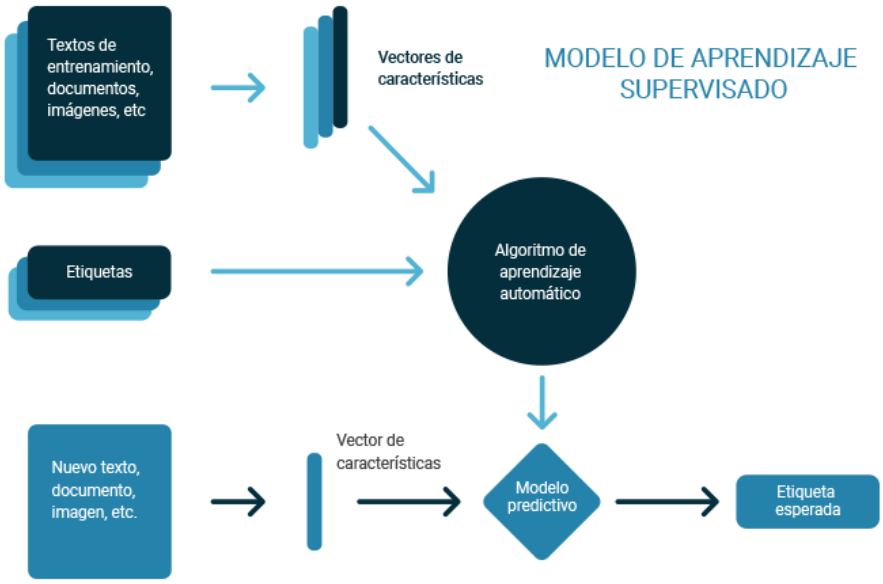

Figura 2.1 – Modelo de aprendizaje supervisado.

- Aprendizaje no supervisado: A diferencia del aprendizaje supervisado, en este caso los algoritmos no cuentan con un conjunto de datos etiquetados para realizar un aprendizaje, sino que se centran en la búsqueda de patrones en los conjuntos de datos analizados con el fin de encontrar un orden en los mismos. Un ejemplo típico es el análisis de datos extraídos de los usuarios de redes sociales con el objetivo de realizar un tipo de publicidad más adaptada.

- Aprendizaje por refuerzo: En este caso se obtiene un aprendizaje a partir de la experiencia propia del algoritmo. Tras determinadas experiencias se aporta una puntuación de recompensa o penalización al modelo en función del resultado de las decisiones tomadas por el mismo. Un ejemplo de este tipo de modelos es el entrenamiento de un robot capaz de lanzar una pelota a una diana, donde la distancia entre el punto de impacto y el centro de la diana puede traducirse a unos pesos con los que retroalimentar al modelo.

2.2.2 Aprendizaje profundo

Dentro del campo del aprendizaje automático encontramos el aprendizaje profundo, conocido como “Deep Learning”[3], mostrado en la figura 2.2. El Deep Learning surgió de la búsqueda de simular el proceso de aprendizaje existente en los seres humanos, tratando de imitar el funcionamiento del cerebro humano.

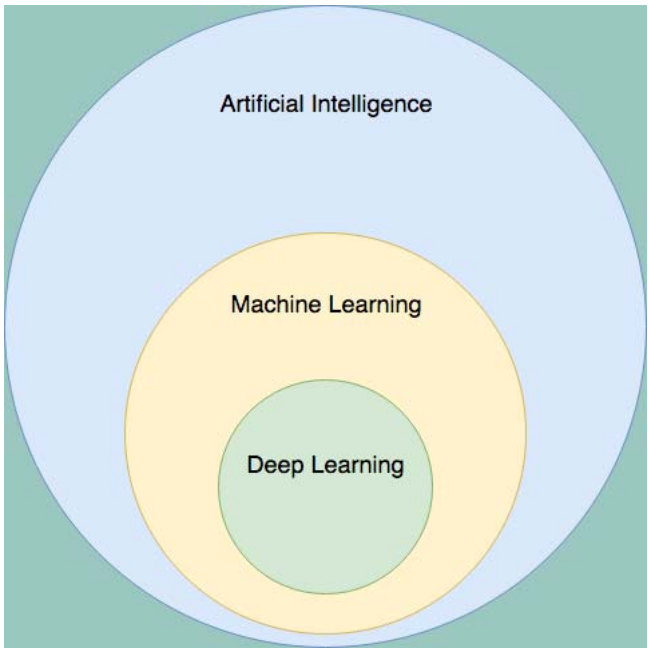

Figura 2.2 – Campo del aprendizaje profundo.

Las estructuras creadas en Deep Learning que imitan el aprendizaje humano son conocidas como "Redes Neuronales". Estas estructuras emplean neuronas artificiales ordenadas en capas con conexiones entre las mismas, como se observa en la figura 2.3. En la literatura, a las capas ocultas se la red se les denomina perceptrones. En una red neuronal, cada neurona calcula su salida a partir de los valores de entrada a la misma y con unas constantes (pesos) que han sido calculadas en el entrenamiento de la red. La red de la figura 2.3 muestra cómo se obtiene un resultado en la capa de salida a partir de 3 variables en la capa de entrada, pasando por 2 capas ocultas de 4 neuronas cada una.

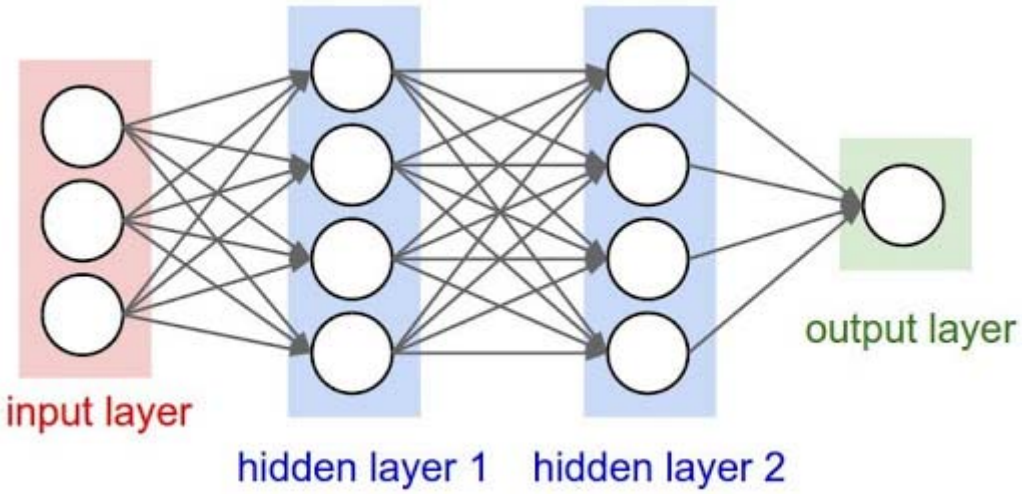

Figura 2.3 – Red neuronal con dos capas ocultas.

La fórmula habitualmente empleada que da lugar al resultado de la neurona i (h_i) a partir de los valores de entrada a la neurona (x_j) y los pesos correspondientes a cada conexión neuronal (w_{ij}) se muestra a continuación:

$$h_i = \sum_j w_{ij} x_j$$

Como se puede observar se trata de una ecuación lineal, que determinará el comportamiento de la red neuronal afectando a la velocidad de convergencia en el entrenamiento, así como al error mínimo alcanzable. Para hacer más complejo el modelo, evitando la linealidad, suele emplearse lo que se conoce como una función de activación. Una función de activación no es más que una pequeña función no lineal que afectará al comportamiento del modelo. En la práctica existe un gran número de funciones de activación empleadas [4], siendo algunos ejemplos la función sigmoide, la función escalón o la función rectificadora (ReLU), siendo esta última una de las más utilizadas.

Como se ha mencionado anteriormente, el entrenamiento de estas redes tiene el objetivo de ajustar los pesos con el fin de que la interacción de las neuronas consiga la salida deseada a partir de las entradas aplicadas. Los pesos no están asociados a las neuronas, sino a las conexiones entre las mismas. Durante el entrenamiento, se emplean unos datos de entrada a la red consiguiendo una determinada salida. El error obtenido se determina mediante una función de error, que puede ser diferente para cada red neuronal. La calibración de los pesos se realiza mediante el algoritmo de "Retropropagación" o "Backpropagation". La base del algoritmo para el ajuste de los pesos es la técnica de gradiente descendiente, que recorriendo la red en el sentido inverso a la predicción realiza un ajuste en los pesos buscando un decrecimiento rápido en la función de error. Una explicación en profundidad acerca del funcionamiento de este algoritmo puede encontrarse en [5].

Un problema muy común en este tipo de redes es el "Overfitting". Este concepto hace referencia a un entrenamiento excesivo, lo cual puede ser tan peligroso para un modelo como la falta de entrenamiento. Cuando se realiza un entrenamiento se pretende que el modelo aprenda las características generales de los datos, es decir, una generalización. Sin embargo, cuando el entrenamiento dura demasiado tiempo el modelo cae en una especialización sobre los datos, es decir, pierde la capacidad de generalizar, lo cual puede apreciarse en la tercera representación de la figura 2.4. Si se plantea el ejemplo en que hemos entrenado una red neuronal para identificar gatos en fotografías, el overfitting provocaría en la red la capacidad de tener un alto rendimiento en los datos de entrenamiento, pero sin embargo tendría serias dificultades al enfrentarse a datos nuevos, siendo incapaz de realizar la identificación en gran parte de los casos.

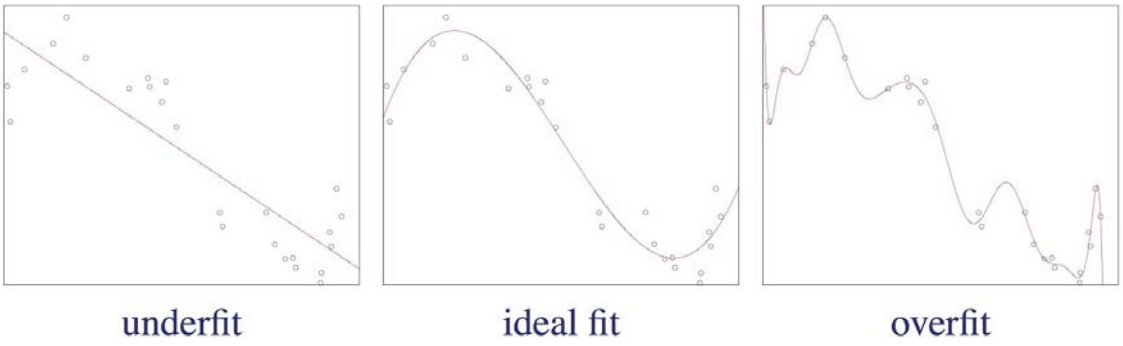

Figura 2.4 – Efectos de la duración del entrenamiento.

Para prevenir el overfitting, se emplean técnicas como el conjunto de validación. Esta técnica consiste en aislar parte de los datos de entrenamiento (conjunto de validación) y emplearlos para calcular la función de error de estos y comparar los resultados con los obtenidos de la función de error del conjunto de entrenamiento. La monitorización de estos valores (figura 2.5) permite encontrar el momento óptimo para poner fin al entrenamiento de la red, antes de que aumente el error del conjunto de validación.

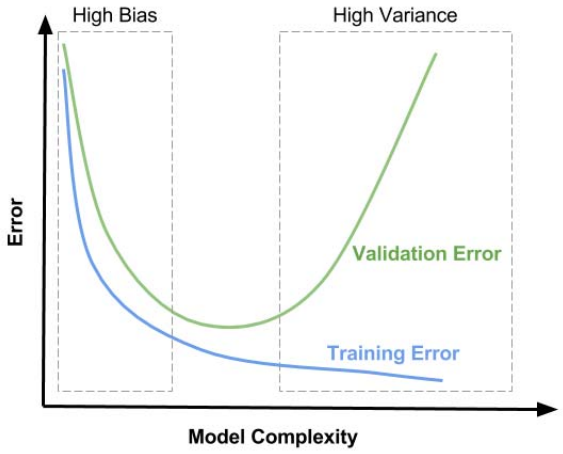

Figura 2.5 – Evolución del error en conjuntos de entrenamiento y validación.

2.2.3 Redes neuronales convolucionales

Las redes neuronales convolucionales [6] son un tipo redes neuronales adaptadas al procesamiento de datos que tienen una tipología en red, como es el caso de las imágenes. Para simplificar la explicación, en este apartado se hablará de datos en dos dimensiones, como es el caso de una imagen de un canal, es decir, una matriz bidimensional.

Para el almacenamiento de los datos en valores numéricos empleamos tensores en vez de las neuronas empleadas en las redes neuronales clásicas. Un tensor es un objeto matemático

capaz de almacenar valores numéricos y que puede tener cualquier número de dimensiones. Un tensor de una sola dimensión es un vector matemático, un tensor 2D una matriz, etc.

Esta tipología de redes neuronales de aprendizaje profundo recibe su nombre por la operación matemática de convolución, una operación aplicada a datos organizados en red mediante lo que se denomina un filtro de convolución, que en el ámbito de las redes convolucionales recibe el nombre de "Kernel". El resultado de la convolución de una imagen 2D mediante un determinado kernel, es otro tensor bidimensional, como se muestra en la figura 2.6.

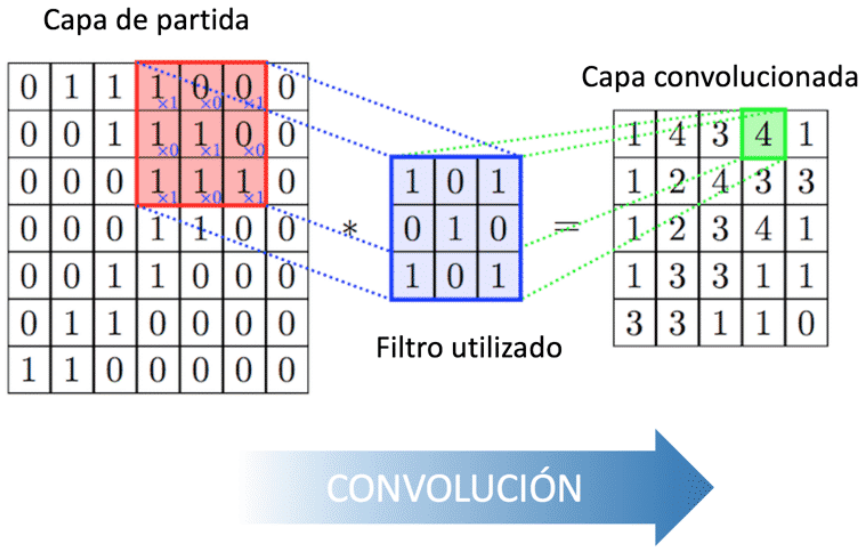

Figura 2.6 – Convolución de una matriz bidimensional.

Para reducir el tamaño de la salida de la convolución en datos de entrada de alta envergadura es común aumentar el paso de la convolución. Esto quiere decir que el filtro de convolución es aplicado a una distancia de separación que viene dada por el paso de convolución con respecto al resto de convoluciones. Como consecuencia se tiene una reducción del número de convoluciones realizadas y una reducción del tamaño a la salida, es decir, se aligera el coste computacional y el tamaño de los datos manejados por la red.

Otra estrategia empleada para la reducción del tamaño de los tensores es la utilización de otra capa conocida como "Pooling Layer" tras la realización de la convolución. Estas capas reducen el tamaño de los tensores mediante el cálculo de máximos locales o valores medios locales, como puede apreciarse en la figura 2.7.

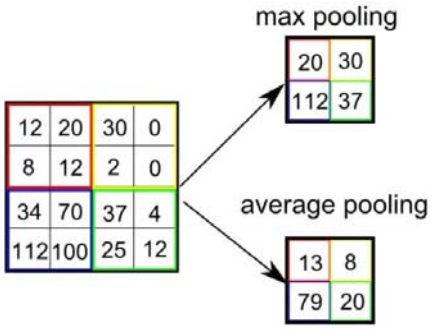

Figura 2.7 – Max pooling y average pooling.

La función de las convoluciones consiste en resaltar a la salida de estas los rasgos encontrados en la imagen de entrada que coinciden con los empleados en el filtro de convolución. De este modo una primera aplicación de unos determinados filtros de convolución permitiría obtener en una imagen la localización de unos rasgos básicos tales como esquinas o bordes. Con la aplicación consecutiva de convoluciones se extraen características cada vez de más alto nivel, tales como texturas o, a un mayor nivel, elementos más complejos como la rueda de un vehículo.

Dado que en cada capa de convolución por lo general se emplea más de un kernel, a la salida de dicha capa se tendrán tantos tensores 2D como filtros de convolución se hayan empleado, teniendo entonces un tensor 3D. Conforme avanzan los datos en la red, el tamaño bidimensional original queda enormemente reducido, mientras que crecen el número de canales (tercera dimensión del tensor) como resultado de las sucesivas convoluciones aplicadas. Finalmente se realiza un aplanamiento del tensor, esto significa que los datos son distribuidos en un vector unidimensional empleado a la entrada de una red neuronal clásica como las explicadas en el apartado anterior. En la figura 2.8 se muestra un ejemplo de la estructura mencionada.

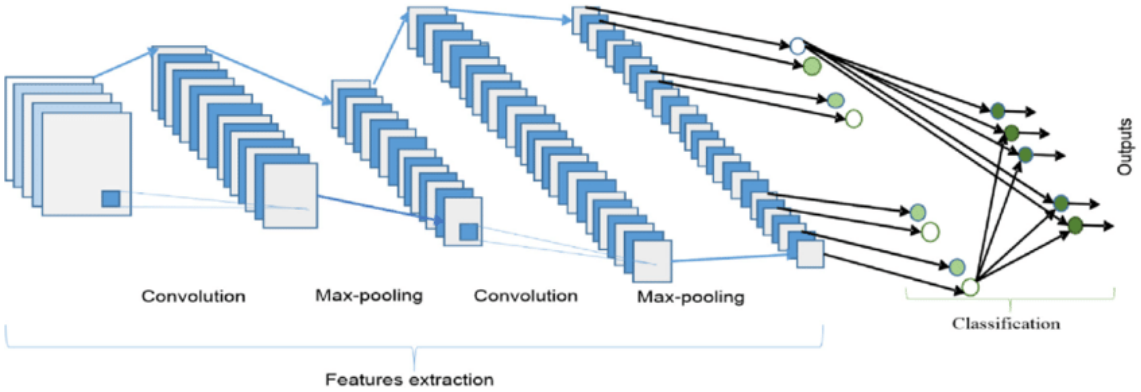

Figura 2.8 – Estructura de una red neuronal convolucional.

2.3 Sistemas de conducción autónoma

Los sistemas de conducción autónoma, o vehículos autónomos, tienen el objetivo de imitar la forma de conducción del ser humano e incluso mejorarla. Las tecnologías que forman parte de estos sistemas se han visto impulsadas en los últimos años, tanto los algoritmos de visón artificial como los algoritmos de detección y clasificación mediante Machine Learning o Deep Learning son ejemplos de los pilares de la conducción autónoma actual.

Una de las principales motivaciones para el desarrollo de los sistemas de conducción autónoma es la seguridad. Según un estudio que analizó más de dos millones de accidentes en estados unidos [7], la causa en el 94% de los casos fue atribuible a conductores de vehículos. La elevada sensorización presente en un vehículo inteligente permite una mayor cobertura que la que pueden proporcionar los sentidos de un conductor, así como los sistemas electrónicos pueden conseguir tiempos de respuesta menores en presencia de situaciones de peligro.

2.3.1 Niveles de automatización

La sociedad de ingenieros de automoción (SAE) definió cinco niveles de automatización para los vehículos autónomos [8]. En esta clasificación un nivel cero hace referencia a la ausencia de cualquier tipo de automatización.

- Nivel 1 – Asistencia al conductor: Se trata del menor nivel de automatización. El vehículo proporciona un único sistema como asistencia, como es el caso de la velocidad de crucero, donde el vehículo toma un control supervisado del sistema de aceleración.

- Nivel 2 – Automatización parcial de la conducción: En este caso el vehículo toma el control de los sistemas de guiado y aceleración/deceleración al mismo tiempo. El piloto automático de Tesla es un claro ejemplo de nivel 2. En este caso la conducción sigue siendo supervisada por el conductor, el cual tiene la responsabilidad de monitorizar la conducción y el entorno.

- Nivel 3 – Automatización condicional de la conducción: Los vehículos en el nivel 3 tienen la capacidad de detectar el entorno y tomar decisiones como realizar un adelantamiento. Aún con el vehículo siendo capaz de observar el entorno, el conductor debe permanecer en el asiento y con la atención necesaria para tomar el control del vehículo en caso de que los sistemas no sean capaces de realizar las tareas de conducción.

- Nivel 4 – Alta automatización de la conducción: El salto del nivel 3 al nivel 4 de automatización está definido por la capacidad del vehículo de intervenir en caso de peligro o en casos de fallo del sistema. En la mayoría de las circunstancias el vehículo no requiere la asistencia del conductor, aunque este puede tomar el control en cualquier momento.

- Nivel 5 – Automatización completa de la conducción: En este nivel no se requiere un conductor. La conducción está totalmente automatizada, de tal forma que el sistema tiene la capacidad de hacer cualquier cosa que pueda hacer un conductor experimentado. En la actualidad no existe ningún vehículo que haya logrado el nivel 5 de automatización de la conducción.

2.3.2 Detección basada en imagen

La detección de objetos consiste en la identificación del lugar y tamaño de objetos de determinadas clases de interés en un conjunto de datos, como imágenes o nubes de puntos. En el caso de la detección de objetos basada en imágenes, la localización de los elementos de interés se realiza en el plano de la propia imagen, requiriendo información adicional para la localización del objeto en un entorno tridimensional. La localización de la entidad detectada se proporciona mediante el recuadro mínimo que contiene al objeto en la imagen analizada, conocido comúnmente como “bounding box”.

En el ámbito de la conducción autónoma son de especial interés las clases de objetos asociadas a peatones, vehículos de diferentes tipologías y señalización de las carreteras. Los métodos de detección de estas clases de objetos en imágenes están basados en las redes neuronales convolucionales, teniendo una división en dos categorías principales:

1. Métodos de detección en una sola etapa: Este tipo de métodos utilizan una única red, que a su salida genera las posiciones y predicciones de clase de los objetos encontrados en la imagen de manera simultánea.

2. Métodos de proposición de regiones (RPN): En este caso el proceso se divide en dos etapas, una primera etapa dedicada a la generación de regiones candidatas y una segunda etapa de clasificación de objetos para las regiones generadas. Cada etapa de estos métodos consta de una red neuronal diferente.

Los métodos de proposición de regiones son los que consiguen mejores precisiones en la actualidad, a costa de un alto coste computacional y una dificultad añadida a la hora de realizar implementación, entrenamiento y ajuste final. Por otra parte, los métodos de detección en una etapa tienen un menor coste en memoria y una mayor velocidad de detección, lo que los hace más adecuados para tareas en tiempo real, como la condición autónoma. De entre los métodos de detección en una etapa hay que mencionar YOLO (You Only Look Once) [9] y sus sucesivas modificaciones YOLOv3 [10] y YOLO9000 [11], así como SSD (Single Shoot Detector) [12]. Por parte de los métodos de proposición de regiones, algunos de los más conocidos son Incept.ResNet v2 [13], Inception v4 [13], ResNet101 [14] y DenseNet201 [15], siendo estos los cuatro métodos que menor tasa de error han conseguido en el set de datos ImageNet1K.

2.3.3 Hardware de detección

Podría decirse que, en el ámbito de la conducción autónoma, los sensores que mayor cantidad de información del entorno son capaces de captar son las cámaras de visión artificial y los LIDAR. Gracias a la aparición de las redes neuronales convolucionales, la detección de elementos en imágenes se ha mejorado enormemente. La aparición de nuevas arquitecturas de redes neuronales ha permitido la utilización de esta tecnología a altas velocidades, facilitando su uso en vehículos autónomos. Sin embargo, debido a la naturaleza tridimensional de los datos obtenidos por los sensores LIDAR, las nubes de puntos, surge una mayor dificultad a la hora de entrenar redes neuronales convolucionales para el tratamiento de estos tipos de datos. Como se explicará en el siguiente apartado, han surgido diferentes estrategias para entrenar redes neuronales adaptadas a las nubes de puntos, sin embargo, esta rama no está tan desarrollada como la rama de redes neuronales convolucionales aplicadas a imágenes.

En la actualidad se encuentran diferencias tan significativas entre ambas tecnologías que hacen que ninguna pueda ser sustitutiva de la otra. Por ejemplo, las cámaras de visión artificial aportan información sobre el color en objetos, aunque no tienen una percepción de profundidad, y por otro lado es necesario emplear un gran número de estas para abarcar los 360 grados alrededor del vehículo. Por su parte, los LIDAR, a diferencia de las cámaras, no dependen de la iluminación externa para la captación de datos, lo cual los hace mejores sensores para la detección de elementos en entornos oscuros. Por otra parte, son capaces de aportar datos de profundidad, aunque como desventaja tienen una disminución de resolución en los objetos lejanos y además no permiten la detección de color.

En la actualidad existen estructuras de redes neuronales para la identificación y clasificación de objetos en datos de cámaras y LIDAR por separado, pero también pueden encontrarse arquitecturas que realizan la fusión de los datos obtenidos por ambos sensores para apoyarse en los puntos fuertes de estas dos tecnologías.

2.4 Redes neuronales para nubes de puntos

Como se ha comentado en el apartado anterior, las redes neuronales convolucionales aplicadas a datos procedentes de cámaras han tenido una evolución importante y por tanto están enormemente desarrolladas. En este apartado se abordará el estado del arte de las redes neuronales aplicadas a nubes de puntos, desde las arquitecturas tradicionales hasta arquitecturas más novedosas que permiten su aplicación en entornos más exigentes.

2.4.1 Arquitecturas tradicionales

Las arquitecturas tradicionales de redes neuronales aplicadas a nubes de puntos recurrían a estrategias que pueden llevar a una pérdida de parte de la información aportada por los datos o a transformar dichos datos a otros formatos que aumentan el tamaño de estos en gran medida, a continuación, se muestran algunas de estas arquitecturas:

Una de las arquitecturas conocidas es la VoxNet [16], sus autores proponen un método de clasificación de objetos dado el segmento del espacio que lo contiene, es decir, su bounding box. En este caso se genera una red tridimensional en la que se introducen las posiciones de los puntos encontrados en la nube, de tal modo que se obtiene un tensor 3D al que se le aplican convoluciones como a cualquier red convolucional aplicada a imágenes. El principal inconveniente de este tipo de redes es la generación de un tensor 3D de grandes dimensiones donde la mayor parte del espacio no contiene información, dado que los puntos muestreados únicamente se encuentran en las superficies visibles.

Por otra parte en [17] puede encontrarse la arquitectura Multi-view CNN, en la cual se propone abordar el problema de la identificación de un objeto tridimensional dado por una nube de puntos a través de imágenes 2D renderizadas a partir de la representación del modelo en diferentes ángulos, como muestra la figura 2.9. El principal problema que aparece en este método es la necesidad de tener una representación completa del objeto, mientras que con un escaneo de un LIDAR solo es posible la representación desde un punto de vista, omitiendo datos de las caras ocultas del objeto. Por otra parte, la información aportada por la nube de puntos queda distorsionada y no es posible el aprovechamiento de la totalidad de los datos.

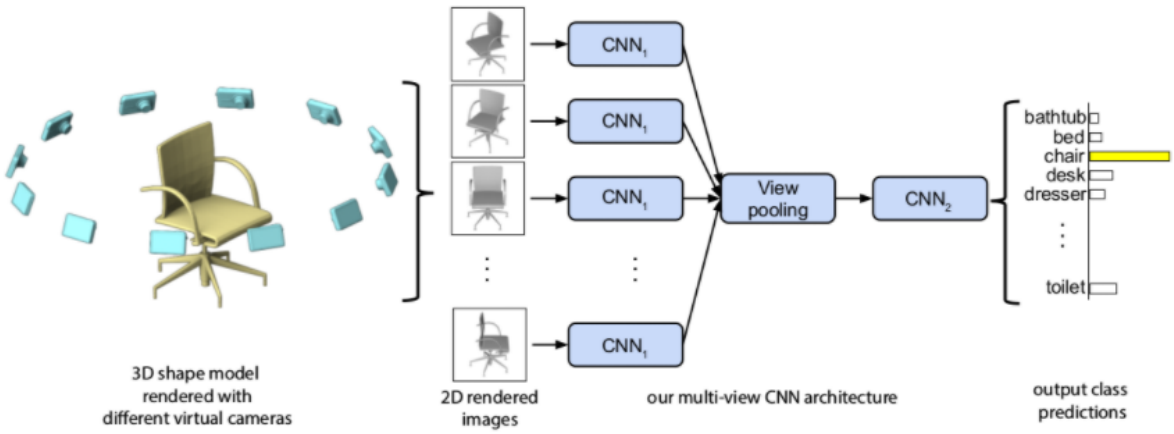

Figura 2.9 – Multi-view CNN.

2.4.2 PointNet

Como se ha visto en el apartado anterior, el uso de redes convolucionales directamente sobre nubes de puntos transferidas a tensores 3D en VoxNet resulta ineficiente, requiriendo el uso de tensores enormes para almacenar datos en unos pocos espacios. La arquitectura PointNet [18] rompe con las arquitecturas tradicionales al presentar una forma de trabajo aplicada a datos en nubes de puntos a diferencia de los casos anteriores, de tal modo que trabaja con el espacio que ocupan esos mismos datos y con toda la información inalterada de los mismos. Esta arquitectura no solo permite la clasificación, sino que también aporta una subred para la segmentación de geometrías locales. Para que una arquitectura de Deep Learning funcione correctamente sobre nubes de puntos, se deben cumplir ciertas condiciones:

- Dado que los datos en nube de puntos consisten en una lista de puntos desordenada, la red neuronal debe ser invariante con respecto al orden de los puntos de entrada a la misma.

- Dado que las características de un objeto se definen por sus formas, la red debe ser capaz de captar las interacciones entre los puntos con el fin de identificar esas características.

- La red debe ser capaz de identificar un objeto aun sufriendo una rotación, por tanto, debe de ser invariante ante transformaciones rígidas.

En la figura 2.10 se muestra la arquitectura PointNet, la cual se divide en una red de clasificación y otra de segmentación. Como puede observarse, los datos de entrada a la red de clasificación consisten en un vector de n puntos de 3 dimensiones. Los datos de entrada pasan a una T-Net [19] que se encarga de generar la matriz de transformación 3x3 por la que los datos serán rotados, de tal modo que se consigue que la red sea invariante ante las transformaciones rígidas. Estos datos pasan a continuación a un perceptrón multicapa formado por dos capas de 64 neuronas cada una, de tal modo que de tener n puntos con 3 dimensiones, se pasa a n puntos con 64 características compartidas. A la salida se repite la transformación mediante T-Net con la diferencia de que esta vez se trata de una transformación mediante una matriz 64x64. Posteriormente se vuelve a aplicar un perceptrón multicapa de 3 capas con 64, 128 y 1024 neuronas que deja n puntos de 1024 características compartidas. En este momento se realiza un max pooling que resulta en un único vector de 1024 características globales, eliminando en este punto los efectos del orden en los puntos de la nube. Mediante otro perceptrón multicapa de 3 capas de 512, 256 y k neuronas se obtiene un vector de puntuaciones de clasificación, donde k es el número de objetos candidatos, coincidente con la longitud del vector.

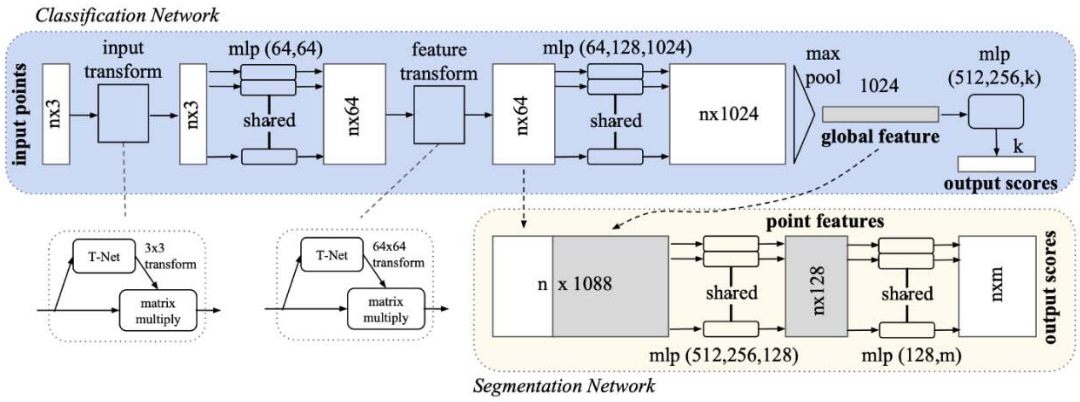

Figura 2.10 – Arquitectura PointNet.

La red de segmentación se alimenta con una concatenación de los datos a la salida de la segunda T-Net y el vector de características globales, teniendo n puntos de 1088 características. Estos datos pasan a un perceptrón multicapa de 3 capas con 512, 256 y 128 neuronas y

posteriormente otro perceptrón de dos capas de 128 y m neuronas. En este caso se obtiene para cada punto un vector de m dimensiones que aporta unas puntuaciones de segmentación sobre los m candidatos de segmentación. Esta subred permite encontrar características locales tales como las alas de un avión, una vez la subred de clasificación ha identificado el candidato como avión.

Aunque PointNet emplea una red para clasificación y otra para segmentación, puede emplearse únicamente la red de clasificación si no se desean obtener los resultados de segmentación de geometrías locales. Cabe destacar que este método permite identificar un objeto dada la nube de puntos que engloba exclusivamente ese candidato, por tanto, de por sí misma no es capaz de analizar un entorno en el que pueden existir múltiples candidatos. Por esto, muchas redes de detección o segmentación emplean PointNet o subredes basadas en la misma como parte de sus arquitecturas.

2.4.3 Frustum PointNet

En el caso de Frustum PointNet [20], se presenta una arquitectura mixta basada en la combinación de nubes de puntos, procedentes de LIDAR o de cámaras de profundidad, e imágenes. Esta arquitectura permite el análisis de los datos completos detectados por ambos tipos de sensor, es decir, permite el análisis de la escena y la búsqueda de elementos en la misma. Esta arquitectura no emplea fusión de datos, sino que los datos de nubes de puntos e imágenes son usados por separado en diferentes etapas del modelo.

Frustum PointNet divide el trabajo en las siguientes 3 etapas mostradas en la figura 2.11:

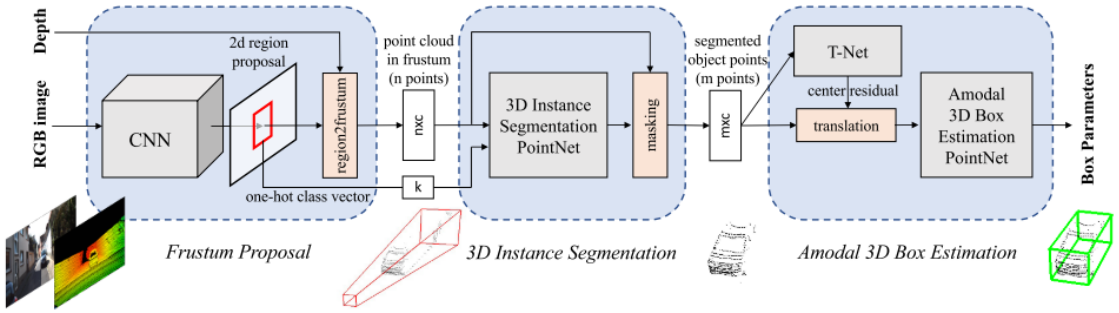

Figura 2.11 – Arquitectura Frustum PointNet.

- Frustum Proposal: En esta etapa aprovecha la resolución superior de las imágenes RGB con respecto a las nubes de puntos. La imagen RGB entra a una red neuronal convolucional que marca la ubicación de los elementos encontrados en un recuadro, aprovechando de este modo la madurez de las redes neuronales aplicadas a imágenes. Una vez se tiene el recuadro que contiene al candidato, se eliminan de la nube de puntos todos aquellos puntos que no se encuentren en la visualización. A la salida de esta etapa se obtiene la nube de puntos acotada sobre el candidato y un vector de probabilidades de tamaño igual al número de candidatos posibles.

- 3D Instance Segmentation: El tronco de pirámide que contiene la nube de puntos restante no solo incluye el objeto candidato, sino puntos correspondientes a objetos que se encuentran delante y detrás del mismo. La segmentación del objeto en esta etapa se realiza mediante una red basada en PointNet que emplea la nube de puntos contenida en el tronco. Esta subred emplea el vector de probabilidad de candidatos generado en la etapa anterior para centrar la búsqueda en estructuras de puntos similares al candidato obtenido en la etapa anterior. A la salida se tienen por tanto los puntos correspondientes al candidato referenciados a un nuevo sistema de coordenadas centrado en el mismo.

- Amodal 3D Box Estimation: Al inicio de esta etapa se emplea una T-Net de regresión que aporta una matriz de transformación para cambiar la referencia del objeto a una normalizada, igual que empleaba PointNet en el apartado anterior. Finalmente se emplea una red de regresión basada en PointNet que genera la bounding box que contiene al objeto identificado incluso si parte de este no es visible en la nube de puntos.

En resumen, Frustum PointNet realiza la detección y clasificación de elementos mediante las imágenes RGB, mientras que la nube de puntos es empleada en la localización y obtención de dimensiones. A diferencia de arquitecturas que trabajan únicamente con LIDAR, este modelo requiere la existencia de cámaras RGB en todos los ángulos del vehículo para dar la cobertura similar a las arquitecturas mencionadas.

2.4.4 PointPillars

Una de las arquitecturas de aprendizaje profundo más interesantes para nubes de puntos es PointPillars [21], la cual destaca por su alta velocidad de procesamiento, siendo capaz de procesar a 62Hz según indican sus autores. Esta arquitectura trabaja únicamente con los datos de LIDAR, por tanto, no requiere colocar cámaras en todos los ángulos de un vehículo autónomo para cubrir el completamente el espacio alrededor del mismo.

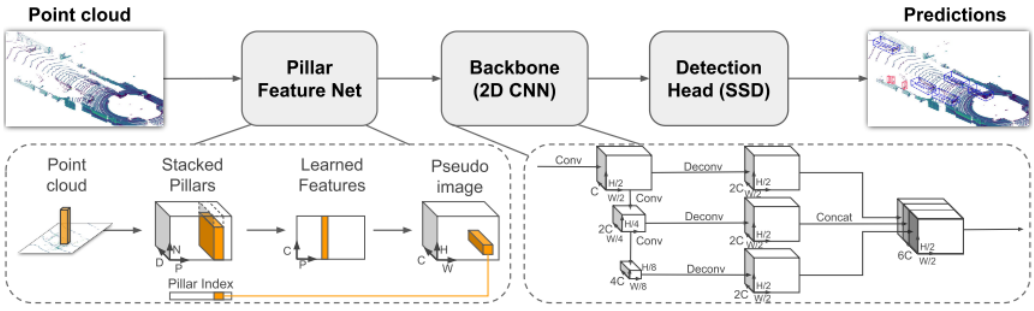

Figura 2.12 – Arquitectura PointPillars.

Como puede verse en la figura 2.12, la arquitectura interna de PointPillars se basa en 3 etapas diferenciadas. Comienza con una primera etapa de extracción de características donde se divide el espacio tridimensional en columnas, a las que se añade información adicional de los puntos de la nube para luego realizar una serie de convoluciones que permiten la extracción de características de los puntos incluidos en los pilares, lo que permite reformular el problema 3D a un problema 2D mediante la proyección de las características de las columnas en el plano horizontal. La pseudo imagen 2D formada en la primera etapa pasa a la estructura troncal de la red, que aplica sucesivas convoluciones que disminuyen el tamaño de la imagen para luego concatenar los resultados intermedios de diferentes tamaños con el fin de tener características de distintas escalas en el mismo tensor. Para finalizar, la tercera etapa trata de una estructura típica de detección en una etapa para hallar la localización sobre la pseudo imagen 2D de los objetos de interés en el plano horizontal del espacio, tras lo cual, una básica red de regresión aporta los datos de posicionamiento vertical y altura del objeto detectado.

La estructura de PointPillars dota a esta arquitectura de una gran velocidad de procesamiento, moviendo el cuello de botella en la mayor parte de casos de la red neuronal a la velocidad de obtención de datos del LIDAR. Esto junto a la precisión demostrada en el la clasificación de Kitti [22] para datos tridimensionales y la dependencia exclusiva de un único sensor como es el LIDAR, la hace una de las arquitecturas más importantes en conducción autónoma.

2.5 Software de aprendizaje profundo

Para la creación, entrenamiento y uso de redes neuronales de aprendizaje profundo existe una gran variedad de paquetes y APIs disponibles. Las más conocidas son TensorFlow [23], Pytorch [24] y Keras [25].

TensorFlow es una plataforma de código abierto de Google basada en JavaScript y dotada de los recursos necesarios para el entrenamiento y uso de modelos de redes neuronales. TensorFlow permite trabajar con bancos de datos de gran tamaño con buen rendimiento. Keras es una API de alto nivel basada en TensorFlow, aportando una mayor facilidad de uso, sin embargo, se trata de una API más lenta y aplicable a conjuntos de datos de pequeño tamaño.

Pytorch es una librería de código abierto desarrollada por Facebook. Pytorch permite conseguir una gran aceleración por GPU en el cálculo con tensores. Su arquitectura permite la creación de redes neuronales más complejas, trabajando con un alto rendimiento y aplicable a conjuntos de datos de gran tamaño. Su principal inconveniente es la complejidad de cara al usuario.

Figura 2.13 – Keras, TensorFlow y Pytorch.

Aunque se dispone de librerías para trabajar en un lenguaje de alta velocidad como C++, la mayor parte de desarrolladores recurren a Python para la implementación de redes neuronales de aprendizaje profundo. Python es un lenguaje de programación de código abierto, interpretado, multiparadigma y orientado a objetos. Por otra parte, es lenguaje de propósito general, de alto nivel y multiplataforma. Estas características han propiciado que la popularidad de este lenguaje haya aumentado con el tiempo, siendo uno de las más usados en los últimos años.

CAPÍTULO 3

SELECCIÓN Y ANÁLISIS DE LA ARQUITECTURA

3.1 Introducción

Este capítulo aborda la selección de la arquitectura de aprendizaje profundo más adecuada al caso práctico de detección en entornos de conducción autónoma. Se presentan los criterios de selección más adecuados y se realiza la selección final. Posteriormente se realizará un análisis profundo de la arquitectura elegida. Para finalizar se abordan los conjuntos de datos empleados para la realización de este trabajo de fin de estudios.

3.2 Selección de la arquitectura

Las arquitecturas presentadas en el estado del arte son una pequeña muestra de un amplio abanico de candidatas posibles. Será necesario definir unos criterios de selección con el objetivo de encontrar la arquitectura más adecuada al campo de interés, en este caso la conducción autónoma.

3.2.1 Criterios de selección

Si se considera un escenario de conducción normal, los factores de los que dependen el éxito de un conductor están basados en su capacidad de detectar los principales elementos de la conducción, ya sean elementos de señalización u otro tipo de entidades como vehículos y peatones. No obstante, la capacidad de detección no es el único factor del que depende la seguridad al volante, entra en juego el tiempo de detección como una variable importante que permite al conductor reaccionar con antelación a los elementos detectados en el entorno de conducción.

Pasando al campo de la conducción autónoma, un vehículo inteligente debe ser capaz de detectar los elementos de señalización, obstáculos en la calzada y presencia de otras entidades en un tiempo corto y con la suficiente tasa de refresco que permita un seguimiento fluido de los objetos detectados y proporcione a los algoritmos de toma de decisiones la información más precisa posible y con la mayor antelación posible. Este trabajo se enmarca en el campo de la detección, con una aplicación en nubes de puntos obtenidas por sensores LIDAR de alta densidad. Dado que los algoritmos de detección están basados en las redes neuronales de aprendizaje profundo, es necesario analizar las arquitecturas de redes neuronales disponibles en base a dos variables fundamentales, la precisión media de detección [26] y el tiempo de procesado medio por cada fotograma.

La precisión media de detección depende de dos factores principales, la arquitectura de la red neuronal y el entrenamiento de esta. Por su parte, el tiempo de procesado medio de los fotogramas depende fundamentalmente de la arquitectura de la red. En el apartado 2.3.2 se analizaron los tipos de arquitecturas dedicadas a detección en imágenes, particularizando en dos categorías principales, las redes neuronales de detección en una sola etapa y las de tipo RPN. Dado que varias arquitecturas 3D emplean estructuras basadas en redes 2D, es de interés analizar ambas categorías de redes de detección en imágenes. Las principales diferencias entre

ambos tipos son la precisión media y el tiempo de detección, teniendo los métodos de proposición de regiones unas mejores puntuaciones en cuanto a precisión media, pero sin embargo los métodos en una sola etapa son capaces de lograr velocidades de trabajo compatibles con la conducción autónoma. Por tanto, es asumible que las arquitecturas de detección en nubes de puntos basadas en métodos de detección en una etapa sean más adecuadas que las arquitecturas basadas en métodos de proposición de regiones.

3.2.2 Selección final

Tras analizar diferentes arquitecturas como las planteadas en el apartado 2.4, en base a los criterios de búsqueda de la mayor precisión posible con velocidades de ejecución que posibiliten su correcta aplicación en entornos de conducción autónoma, se ha llegado a la conclusión de que PointPillars [21] puede ser la arquitectura más adecuada. Según el artículo científico en que se presentó esta arquitectura en 2019, es capaz de realizar iteraciones en 16 ms, lo que se traduce a una velocidad de 62 predicciones por segundo o 62 Hz.

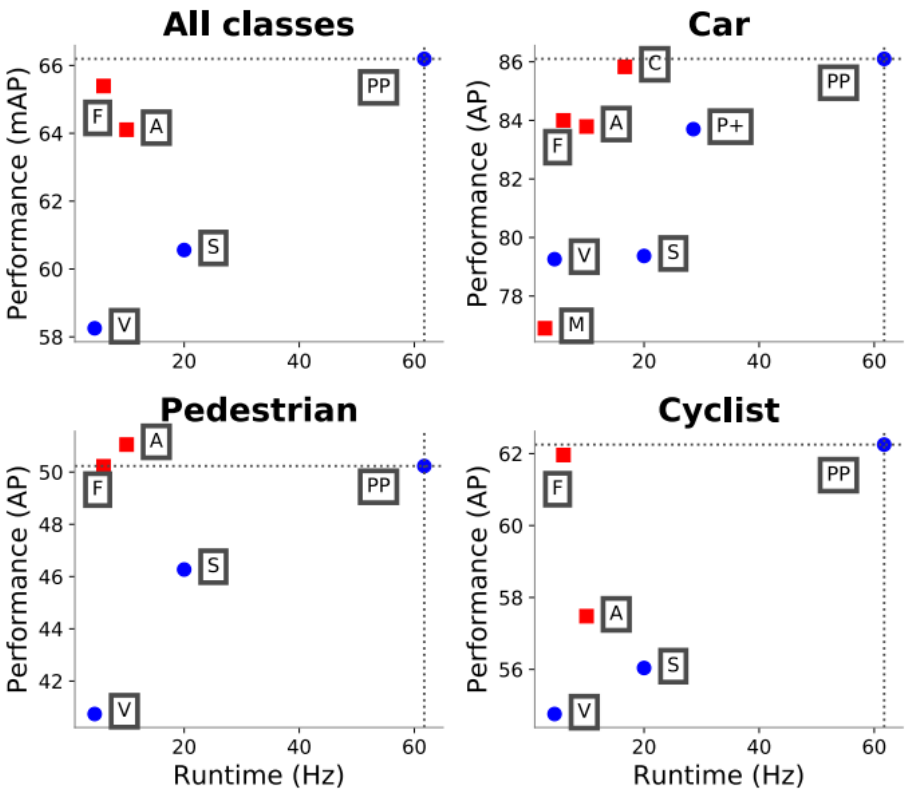

Figura 3.1 – [21] Precisión media (AP) frente a velocidad de procesamiento (Hz) de principales métodos según la tabla de clasificación del set de datos Kitti en la categoría “Bird’s eye view”.

En la figura 3.1 se realiza una comparativa entre algunas de las arquitecturas más relevantes en detección de objetos según el set de datos Kitti [27] a fecha de publicación de PintPillars, incluyendo métodos basados únicamente en LIDAR (puntos azules) y métodos

basados en LIDAR y cámaras (cuadrados rojos). En los gráficos se comparan PointPillars (PP), MV3D [28] (M), AVOD [29] (A), ContFuse [30] (C), VoxelNet [31] (V), Frustum PointNet [20] (F), SECOND [32] (S) y PIXOR++ [33] (P+) en las categorías de detección de turismos, peatones y ciclistas, obteniendo PointPillars unas puntuaciones altas con una velocidad de ejecución muy superior al resto.

La principal característica de PointPillars es la transformación de la nube de puntos tridimensional en una pseudo imagen bidimensional que consiste en un mapa de características locales sobre el plano horizontal. El modo de abordar el escenario tridimensional como un problema clásico de detección en imágenes permite emplear arquitecturas conocidas de detección de objetos en imágenes, particularmente una arquitectura del tipo de detección en una sola etapa, lo que dota a la arquitectura PointPillars de una elevada velocidad.

3.3 Análisis de la arquitectura

3.3.1 Contexto

Una de las tareas más importantes en la conducción autónoma es la detección y seguimiento en tiempo real de objetos tales como vehículos, ciclistas y peatones. De entre todos los sensores disponibles en un vehículo inteligente, el LIDAR podría ser el más importante a la hora de llevar a cabo las tareas de detección al ser capaz de medir en los 360º del vehículo y obtener la distancia a la que se encuentran los objetos mediante medida de ToF. Visto el estado de madurez del Deep Learning aplicado a visión artificial, en la situación actual queda aún margen de trabajo para desarrollar arquitecturas eficientes para la detección de objetos aplicada a nubes de puntos procedentes de sensores LIDAR. El paso de 2D a 3D eleva sustancialmente la carga computacional, con el problema añadido de generar un tensor tridimensional donde casi la totalidad del espacio estaría vacío. En [34] se aborda de manera directa el problema mediante convoluciones 3D aplicadas a nubes de puntos, tratando el problema de dispersión que caracteriza este tipo de datos. Por otra parte, [35] saca partido de las convoluciones 2D al realizar la proyección de la nube de puntos en un plano bidimensional cilíndrico centrado en el LIDAR.

Otros métodos transforman la nube de puntos a lo que se conoce como representación a vista de pájaro, lo cual no es más que una proyección sobre el plano horizontal discretizada de la nube de puntos, obteniendo una imagen 2D con canales de altura, intensidad y densidad de puntos. La representación a vista de pájaro simplifica las convoluciones volviendo una vez más al escenario bidimensional discretizado. Ejemplos de arquitecturas que emplean esta representación son [28], [29], [31] y [36]. Los métodos que empleaban directamente la vista de pájaro tenían poca capacidad de generalización debido a los métodos de extracción de características basados en la vista de pájaro.

En PointNet [18] se propone un método capaz de obtener las características directamente desde la nube de puntos, evitando la pérdida de información que ocurre al realizar transformaciones en etapas anteriores. No se puede hablar de PointNet sin mencionar su versión mejorada PointNet++ publicada en [37]. La primera arquitectura de detección en

obtener las características directamente de la nube de puntos fue VoxelNet [31], que divide el espacio en voxeles, a cada uno de los cuales aplica PointNet, seguido de convoluciones tridimensionales y, posteriormente, bidimensionales. Aunque se consigue una buena precisión con esta arquitectura, la principal limitación de VoxelNet es el tiempo de inferencia, rindiendo únicamente a 4.4 Hz, lo que la hace inutilizable en tiempo real. La arquitectura SECOND [32] consigue un aumento de velocidad con respecto a VoxelNet, sin embargo las convoluciones 3D siguen siendo el cuello de botella a la hora de conseguir más velocidad.

3.3.2 Aportación de PointPillars

PointPillars consiste en una arquitectura de detección de objetos en entornos 3D que emplea únicamente convoluciones 2D. Su principal aportación es el método que emplea para la extracción de características de los datos. Las nubes de puntos son divididas en pilares verticales de los que se extraen una serie de propiedades antes de la transformación del problema a un espacio bidimensional.

Una de las principales ventajas de la arquitectura es la capacidad de extraer toda la información que aporta la nube de puntos, en vez de realizar operaciones que transforman toda la información antes de realizar la extracción de características. Por otra parte, al seccionar el espacio en pilares en vez de en vóxeles, a diferencia de otras arquitecturas [31], se elimina el problema de seleccionar el tamaño vertical óptimo por vóxel. En último lugar, la división en pilares posibilita la proyección de las características en un plano, que a su vez permitirá la ejecución de convoluciones 2D en vez de 3D, lo que permite que el modelo sea altamente eficiente en términos computacionales, permitiéndole alcanzar velocidades de inferencia de 62Hz.

Esta arquitectura consta de 3 etapas principales:

- Pillar Feature Net: En esta etapa se discretiza la nube de puntos realizando un mallado en el plano horizontal, lo que da lugar a un conjunto de P pilares que contienen los puntos. A las coordenadas XYZ de cada punto y a la intensidad de retorno r, se añaden en XYZ las distancias hasta la media aritmética de todos los puntos del mismo pilar y las distancias XY desde el centro de dicho pilar, teniendo entonces para cada punto un vector de 9 dimensiones. Siendo N el número de puntos por pilar, D el número de dimensiones de cada punto y P el número de pilares totales, se genera el tensor de dimensiones (D,P,N) mostrado en la figura 3.2. En caso de que un pilar tenga un número mayor de puntos que N, se seleccionarán N puntos aleatoriamente que serán incluidos en el tensor. Si el pilar tiene menos de N puntos, se rellenarán con ceros los puntos restantes. En este estado se aplica a cada punto de D dimensiones una capa lineal (perceptrón), una capa de normalización y una capa de activación (ReLU), dejando el tensor con dimensiones (C,P,N), siendo C el número de características a la salida de estas capas. Se realiza en este momento un Max Pooling de todos los puntos de cada pilar, dejando el tensor con las dimensiones (C,P). Finalmente se devuelve el tensor a la representación de malla recuperando los pilares su posición original, creando una

pseudo imagen de dimensiones HxW y de C canales, dicho de otro modo, un tensor de dimensiones (C,H,W).

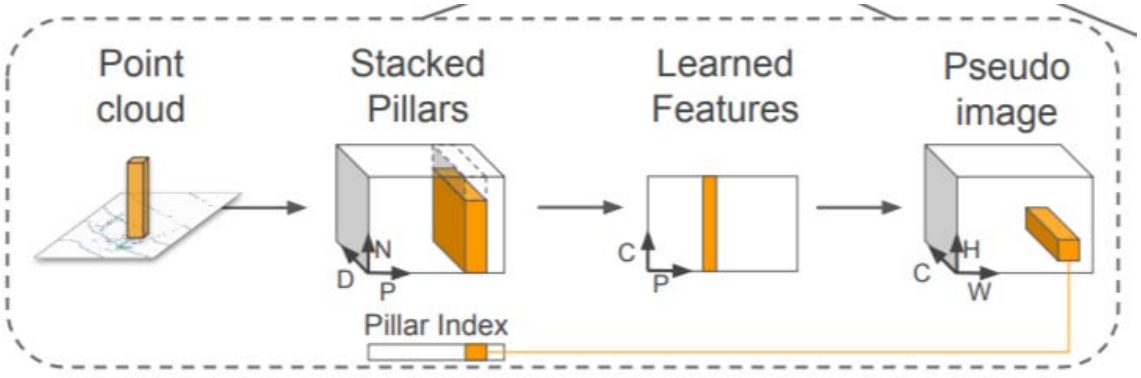

Figura 3.2 – PointPillars: Pillar Feature Net.

- Backbone (2D CNN): Dado que los datos son de características similares a los que aporta una imagen 2D, se pueden emplear redes neuronales convolucionales típicas de imágenes RGB para la extracción de características. En la figura 3.3 se muestra la estructura empleada, en la cual se realizan consecutivas convoluciones que disminuyen el tamaño del tensor en las dimensiones H y W para obtener características a diferentes niveles. Se realiza un escalado de los tensores obtenidos en las distintas etapas de convoluciones y se concatenan con el fin de obtener características de diversas complejidades en un mismo tensor.

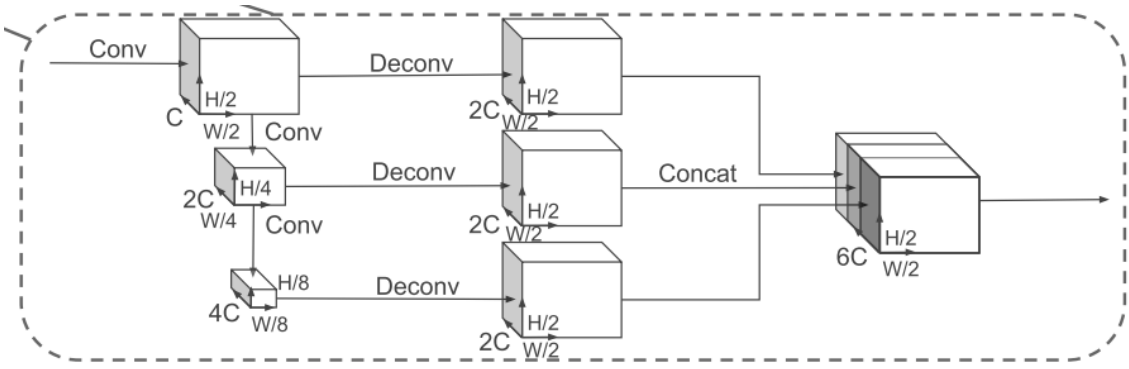

Figura 3.3 – PointPillars: Backbone (2D CNN).

- Detection Head (SSD): Finalmente, en esta etapa se emplea una subred de arquitectura SSD (Single Shot Multibox Detector) [12] que es capaz de detectar e identificar elementos en una imagen 2D con una alta precisión y velocidad. Como resultado de esta red se tienen los recuadros proyectados horizontalmente que contienen a los elementos detectados. Dada esta detección en dos dimensiones, se obtienen como resultados de regresión adicionales la altura y la elevación de la boundingbox que contiene al elemento detectado.

Un modelo entrenado de PointPillars fue evaluado mediante el set de datos de Kitti, en las clasificaciones de detección 3D y a vista de pájaro [38], obteniendo muy buenas posiciones en las tablas de clasificación a la fecha de la publicación [21]. A pesar de emplear únicamente datos procedentes de LIDAR, PointPillars ha obtenido resultados mejores que arquitecturas que emplean datos procedentes de imágenes y LIDAR de manera simultánea. En la figura 3.4 se muestran los resultados publicados de la clasificación de Kitti en la categoría de detección a vista de pájaro. En la figura 3.5 se muestran de manera análoga los resultados en la categoría de detección 3D.

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D	Lidar & Img.	2.8	N/A	86.02	76.90	68.49	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse	Lidar & Img.	16.7	N/A	88.81	85.83	77.33	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet	Lidar & Img.	10	N/A	88.20	79.41	70.02	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN	Lidar & Img.	10	64.11	88.53	83.79	77.90	**58.75**	**51.05**	**47.54**	68.09	57.48	50.77
F-PointNet	Lidar & Img.	5.9	65.39	88.70	84.00	75.33	58.09	50.22	47.20	75.38	61.96	54.68
HDNET	Lidar & Map	20	N/A	89.14	**86.57**	78.32	N/A	N/A	N/A	N/A	N/A	N/A
PIXOR++	Lidar	35	N/A	**89.38**	83.70	77.97	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet	Lidar	4.4	58.25	89.35	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND	Lidar	20	60.56	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PointPillars	Lidar	**62**	**66.19**	88.35	86.10	**79.83**	58.66	50.23	47.19	**79.14**	**62.25**	**56.00**

Figura 3.4 – Resultados en la clasificación de Kitti en la categoría de detección a vista de pájaro.

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D	Lidar & Img.	2.8	N/A	71.09	62.35	55.12	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse	Lidar & Img.	16.7	N/A	82.54	66.22	64.04	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet	Lidar & Img.	10	N/A	**83.71**	73.04	59.16	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN	Lidar & Img.	10	55.62	81.94	71.88	66.38	50.80	42.81	40.88	64.00	52.18	46.61
F-PointNet	Lidar & Img.	5.9	57.35	81.20	70.39	62.19	51.21	**44.89**	40.23	71.96	56.77	50.39
VoxelNet	Lidar	4.4	49.05	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND	Lidar	20	56.69	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
PointPillars	Lidar	**62**	**59.20**	79.05	**74.99**	**68.30**	**52.08**	43.53	**41.49**	**75.78**	**59.07**	**52.92**

Figura 3.5 – Resultados en la clasificación de Kitti en la categoría de detección 3D.

3.3.3 Funciones de error

Las redes neuronales son entrenadas mediante el método de descenso de gradientes estocástico, por tanto, requieren la definición de una función de error que permita la correcta ejecución del proceso de optimización de la red durante el entrenamiento. La arquitectura PointPillars emplea las mismas funciones de error que SECOND [32]. La bounding box asociada a un objeto se define con el vector (x, y, z, w, l, h, θ), siendo (x, y, z) las coordenadas del centro, (w, l, h) las dimensiones y θ el ángulo de orientación de dicha bounding box. Los residuos de estas variables se definen según las expresiones mostradas a continuación, donde el superíndice *gt* ("ground truth") hace referencia a los valores reales, el superíndice *a* hace referencia a los valores generados por el modelo y d es la diagonal de la proyección horizontal de la bounding box.

$$\Delta x = \frac{x^{gt} - x^a}{d^a} \qquad \Delta y = \frac{y^{gt} - y^a}{d^a} \qquad \Delta z = \frac{z^{gt} - z^a}{h^a}$$

$$\Delta w = \log\frac{w^{gt}}{w^a} \qquad \Delta l = \log\frac{l^{gt}}{l^a} \qquad \Delta h = \log\frac{h^{gt}}{h^a}$$

$$\Delta\theta = \sin(\theta^{gt} - \theta^a)$$

El error de localización se define como el sumatorio de la función de error L1 suavizada de cada uno de los residuos anteriores. La función de error L1 suavizada tiene mayor robustez que la función de error L2 frente a valores extremos.

$$\mathcal{L}_{loc} = \sum_{b\in(x,y,z,w,l,h,\theta)} Smooth\ L1(\Delta b)$$

$$Smooth\ L1(x) = \begin{cases} 0.5x^2, & |x| < 1 \\ |x| - 0.5, & x < -1 \text{ ó } x > 1 \end{cases}$$

Dado que el error de localización del ángulo de orientación puede aportar valores muy bajos con direcciones opuestas de orientación real y orientación predicha, es necesario añadir otra función de error. En este caso se calcula un término adicional de error $\mathcal{L}_{dir}$ mediante una función softmax. Como función de error en clasificación de entidades se emplea el error focal $\mathcal{L}_{cls}$ [39].

$$\mathcal{L}_{cls} = -\alpha(1 - p^a)^\gamma \log p^a$$

Siendo p^a la probabilidad de cada bounding box generada por la red y empleando los parámetros $\alpha = 0.25$ y $\gamma = 2$ como en la publicación original. El error total es calculado del siguiente modo:

$$\mathcal{L} = \frac{1}{N_{pos}}(\beta_{loc}\mathcal{L}_{loc} + \beta_{cls}\mathcal{L}_{cls} + \beta_{dir}\mathcal{L}_{dir})$$

Donde N_{pos} es el número de bounding boxes positivas generadas por el modelo y los factores de ponderación toman los valores $\beta_{loc} = 2$, $\beta_{cls} = 1$ y $\beta_{dir} = 0.2$. Para la optimización de la función de error, en el artículo original de PointPillars se emplea el optimizador Adam con un factor de aprendizaje inicial de 0.0002 y un factor de disminución de este de 0.8 cada 15 épocas, con un total de 160 épocas.

3.4 Conjuntos de datos

Para que las redes neuronales de aprendizaje profundo sean capaces de generalizar de manera correcta y por tanto puedan trabajar en diferentes condiciones de funcionamiento es

necesario realizar el entrenamiento del modelo con un set de datos muy variado. Un conjunto de datos con la suficiente riqueza debe proporcionar una gran variedad en los elementos que sean de interés para el modelo que se esté entrenando. En el caso de una red de detección de coches es ideal que el conjunto de datos tenga ejemplos de todo tipo de vehículos de diferentes marcas, años y estilos, de tal manera que el modelo sea capaz de generalizar mejor y aprender las características básicas que definen un coche en vez de características específicas que están presentes en unos y no lo están en otros. Por otra parte, en un set de datos se debe aportar variedad en el entorno, prestando atención a la existencia de datos tomados en diferentes horas del día con diferentes condiciones de iluminación, así como diferentes condiciones climáticas, tanto favorables como adversas.

En el presente proyecto se realizará el entrenamiento de un modelo de red neuronal basado en la arquitectura PointPillars. El conjunto de datos elegido para la realización del entrenamiento es el set de datos de Kitti de 2017 para detección de objetos en 3D [22]. Este conjunto de datos contiene 7481 fotogramas de entrenamiento y 7518 fotogramas para pruebas, donde cada fotograma contiene imágenes de diferentes cámaras del vehículo, la nube de puntos correspondiente, datos de calibración de cámaras, etiquetas de los principales objetos de detección (vehículos, ciclistas y peatones), etc. Dado que Kitti emplea una tabla de clasificación de las arquitecturas de detección, divide todos los objetos del set de datos en 3 categorías de dificultad en función de la visibilidad del objeto, siendo fácil cuando el área máxima no visible del objeto es un 15%, dificultad moderada para un 30% y un 50% de área oculta para dificultad difícil.

Con el modelo entrenado previamente con los datos de Kitti, se realizará una adaptación del software para evaluar su funcionamiento en un conjunto de datos diferente, simulando la implantación de un modelo pre entrenado en un vehículo inteligente. Para la evaluación del modelo se ha seleccionado el conjunto de datos de percepción de “Waymo Open Dataset” [40]. Este conjunto de datos consta de unos 950 escenarios de conducción con todos sus datos captados durante 200 fotogramas por escenario. Los datos de este conjunto están tomados con una tasa de 10 fotogramas por segundo. Cada fotograma contiene datos de etiquetas, nube de puntos, imagen de cada una de las 5 cámaras del vehículo de toma de datos, ect. Las etiquetas de los datos corresponden a vehículos, peatones, ciclistas y señales de tráfico. La utilización de escenarios completos grabados durante varios fotogramas permite realizar una simulación realista del comportamiento de la red neuronal en un entorno de conducción real.

CAPÍTULO 4

DISEÑO Y PROGRAMACIÓN

4.1 Introducción

En el presente capítulo se abordará el diseño y la programación de los algoritmos que permitan el entrenamiento y la simulación de un modelo de aprendizaje profundo con la arquitectura PointPillars para la detección de automóviles en nubes de puntos. Se abordarán los criterios de diseño relativos a los parámetros básicos de la red, se analizarán los problemas potenciales para el buen funcionamiento de esta, se tratarán algunos módulos de interés de la librería Pytorch, se definirán los sistemas de referencia empleados en los distintos conjuntos de datos y se diseñará la estructura de los principales algoritmos.

4.2 Consideraciones previas

Con el objetivo de facilitar el diseño de los algoritmos y evitar obstáculos durante el proceso, se debe realizar una reflexión acerca de los criterios de diseño que permitan desarrollo óptimo del proyecto de acuerdo con la aplicación a la que está orientado. Por otra parte, se realizará un planteamiento de las causas de fallo potenciales en el transcurso de las siguientes etapas.

4.2.1 Criterios de diseño

Con el fin de adoptar unos criterios de diseño adecuados, es necesario tener en cuenta la finalidad de los algoritmos que se van a diseñar. El objetivo es el diseño y programación de algoritmos para el entrenamiento y simulación de un modelo PointPillars de aprendizaje profundo para la detección de elementos en un entorno de conducción autónoma. Para garantizar que se lleve a cabo esta tarea se debe priorizar la velocidad de funcionamiento, tratando de obtener el mejor tiempo de procesado por cada lote de datos. Una mayor frecuencia de actualización de las predicciones del modelo resultará en un aumento de la seguridad al conseguir un seguimiento más preciso de los elementos detectados por el vehículo.

Para el diseño de algoritmos que implementan redes neuronales de aprendizaje profundo se emplea principalmente el lenguaje de programación Python. Este lenguaje multiplataforma de propósito general es uno de los más empleados en la actualidad. Se trata de un lenguaje interpretado, por lo que no se requiere la compilación de los programas basados en Python. También se incluye entre los lenguajes de alto nivel y orientados a objetos, facilitando las labores de programación. Finalmente, al tratarse de un lenguaje de tipado dinámico no requiere de la asignación de tipos de las variables en su declaración. Se ha considerado que para el desarrollo de los algoritmos del presente proyecto Python resulta ser el lenguaje de programación más adecuado. Dentro de Python se utilizará la librería Pytorch para las tareas relacionadas con redes neuronales, tales como la definición de estas, la gestión de los datos, el entrenamiento y las pruebas.

Resulta fundamental para el diseño de la red neuronal la selección de unos determinados parámetros que den forma a los aspectos fundamentales del modelo que se va a

entrenar. Algunos de los parámetros básicos como las dimensiones del área rectangular del plano horizontal en que se generarán los pilares para la separación de la nube de puntos se tomarán de igual valor que los indicados en la publicación principal de PointPillars. En el caso concreto del área de trabajo, se ha decidido mantener la misma área de trabajo por considerarse la zona de mayor interés, suponiendo un área rectangular ubicada en la parte delantera del vehículo con una longitud en la dirección frontal de 70m y 40m en cada una de las direcciones laterales al mismo, tal y como se muestra en la figura 4.1. Las distancias mencionadas se han decidido mantener en esos valores debido a que la mayor parte de los puntos obtenidos por el LIDAR se encuentran en esos márgenes, teniendo una reducción significativa de la densidad de puntos en distancias mayores. Dado que las pruebas se realizarán con datos procedentes del conjunto de datos Waymo Open Dataset, y que el mismo presenta imágenes RGB en las direcciones que permiten una total visibilidad en la zona delantera del vehículo, se ha optado por mantener fuera del área de trabajo la zona trasera del vehículo. Por parte de la altura de trabajo se va a mantener la original, estando en el rango de los 3 metros por debajo del LIDAR hasta 1 metro por encima del mismo.

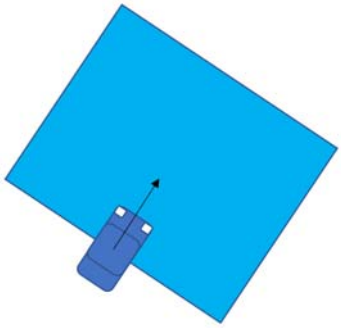

Figura 4.1 – Área de trabajo horizontal relativa al vehículo (representada en color azul claro).

Otros parámetros de interés resultan ser las dimensiones de la sección horizontal de los pilares y el número de puntos máximo contenido en los mismos, eliminando puntos aleatorios en caso de que un pilar exceda dicho límite. Se espera que un tamaño menor de la sección cuadrada de los pilares y un mayor número máximo de puntos contenidos en los mismos permita obtener resultados más precisos, sin embargo, como desventaja aparece un mayor tiempo de procesamiento. Para la selección de los valores óptimos de estos parámetros se ha decidido realizar ensayos con diferentes valores.

4.2.2 Problemas previstos

Durante el desarrollo natural de cualquier proyecto se espera la aparición de problemas que pongan en peligro el trabajo realizado, así como puedan suponer un aumento en tiempo de trabajo. Para reducir en gran medida estos problemas se recurre a un análisis previo de los riesgos potenciales. Aunque no todos los problemas previstos pudieran ser solventados durante el desarrollo, algunos podrían servir para explicar resultados anómalos en futuras pruebas que se mostrarán en sucesivos capítulos.

El primer problema se plantea como consecuencia del trabajo con datos procedentes de fuentes diferentes. Para el entrenamiento de la red neuronal de aprendizaje profundo se ha decidido la utilización de datos procedentes de Kitti, mientras que se tiene como objetivo la implementación de la red en otro vehículo, concretamente en este proyecto dicha implementación se realiza mediante simulaciones con datos procedentes del conjunto de datos de Waymo. Las nubes de puntos procedentes de sensores LIDAR no solo aportan una lista de puntos en su posición espacial en un sistema de referencia (x, y, z), sino que presentan un cuarto valor para cada punto que representa la intensidad del pulso láser de retorno (x, y, z, r), dicho valor dependerá del sensor emisor, del medio y del objeto en que rebota el pulso, como se muestra en la figura 4.2. Al realizar la simulación con datos de procedencia diferente a los datos de entrenamiento de la red de aprendizaje profundo es posible la disminución del desempeño de dicha red como consecuencia de variaciones en las intensidades aportadas por las diferentes fuentes. Este problema podría ser solventado si desde el principio se plantea un modelo que emplee en su entrada únicamente coordenadas espaciales, excluyendo datos de intensidad, sin embargo, en el momento previo al desarrollo, se considera despreciable este posible efecto, por lo que se emplearán los 4 valores en la descripción de cada punto de la nube, con el objetivo de que la red pueda obtener más información y por tanto realizar predicciones más precisas. Otra opción posible es la realización de un normalizado de los valores de intensidad, sin embargo, dado que uno de los objetivos del trabajo es la evaluación de la arquitectura seleccionada, se mantendrá inalterada la intensidad de retorno.

Figura 4.2 – Representación de una nube de puntos con valores de intensidad.

En segundo lugar, los resultados aportados por la publicación original de PointPillars [21] proceden de alimentar el modelo diseñado con nubes de puntos preprocesadas, por lo que la utilización directa de datos en aplicación real o simulación se verá afectada en cuanto a tiempos de procesado, disminuyendo significativamente la frecuencia de inferencia. Este procesado previo de la nube de puntos del set de datos de Kitti consiste en una reducción notable del tamaño de la nube de puntos mediante la eliminación de puntos que no se encuentran dentro de la zona de trabajo de la red neuronal.

Finalmente, de las cuestiones con la relevancia suficiente para ser mencionadas, existe una diferencia con respecto a la altura en los sistemas de referencia empleados en los dos conjuntos de datos. El conjunto de datos de Kitti coloca el centro del sistema de referencia a la altura del LIDAR, mientras que Waymo tiene el sistema de referencia centrado en el LIDAR para los ejes horizontales, pero a la altura del suelo para el eje vertical. Dado que los datos de altura del sensor LIDAR empleados por Waymo no están disponibles, se debe realizar una estimación de esta para realizar un preprocesado en los datos de simulación de manera que estén referenciados del mismo modo. El valor de altura estimada se obtendrá mediante análisis estadístico de nubes de puntos en escenarios que presenten alta similitud de los dos conjuntos de datos.

4.3 Utilidades de Pytorch

Una de las principales razones por las que Pytorch es una librería adecuada para el trabajo con redes neuronales de aprendizaje profundo, es la ayuda a la programación que suponen los módulos que contiene. A la hora de trabajar con estructuras de redes neuronales complejas resulta fundamental que las librerías empleadas se ajusten a las necesidades de programación.

Pytorch ofrece la posibilidad de crear redes neuronales personalizadas mediante “nn.module”. Este módulo permite la definición de una red neuronal y de sus respectivas capas mediante la creación de una clase personalizada. En la clase asociada a la red neuronal es necesario la definición de dos métodos principales. En primer lugar, el método __init()__, como su nombre indica, es el encargado de la inicialización de la red, se definen determinados parámetros así como las capas que emplea. Por otro lado le método forward() define las iteraciones a la que se somete cada lote de datos, generalmente preprocesado, paso de los datos por las capas de la red y postprocesado. Dado que los métodos de esta clase son totalmente definidos por el usuario, existe la posibilidad de crear con el mismo módulo subredes que puedan ejecutarse dentro del método fordward(). Esta posibilidad facilita la separación de grandes redes de varias etapas en subredes más pequeñas que se ejecutan dentro de la red principal.

A la hora de trabajar con grandes conjuntos de datos se hace imposible tener cargado el conjunto de datos completo en la memoria durante el entrenamiento de la red o durante las fases de pruebas. Para resolver este problema Pytorch permite emplear lo que se conocen como “datasets” y “dataloaders”. Para gestionar el proceso de carga de datos se permite la creación de una clase “Dataset” personalizada en la que programar todas las operaciones relacionadas con los datos. El empleo de estas clases personalizadas permite evitar mantener en memoria todo el conjunto de datos, ya que se ofrece la posibilidad de programar de manera personalizada la carga de datos, pudiendo cargar lotes de datos de pequeño tamaño, evitando que desborde la memoria. Igual que en el caso anterior, al ser una clase totalmente programada por el usuario, es posible realizar un preprocesado a cada lote de datos antes de enviarlo al resto del algoritmo. El Dataloader consiste en un objeto de tipo envoltorio que contiene la clase Dataset y la comunica con el resto de los módulos.

Al trabajar con dos conjuntos de datos diferentes como Kitty y Waymo, se van a utilizar dos clases Dataset personalizadas para cada uno de los conjuntos de datos. Esto permitirá que una misma red neuronal de aprendizaje profundo creada en Pytorch pueda trabajar con conjuntos de datos con referencias diferentes, así como emplearlos en los diferentes programas de entrenamiento y simulación.

4.4 Sistemas de referencia

Para cualquier programa que trabaje con posicionamiento de objetos es necesario definir desde el principio un sistema de referencia. En el caso actual se trabaja con datos que han sido adquiridos por dos fuentes diferentes, Kitti y Waymo, por lo que es necesario comenzar exponiendo dichos sistemas de referencia. Dado que el entrenamiento de la red neuronal se va a realizar con los datos procedentes de Kitti, se adoptará su sistema de referencia como el sistema de trabajo, por tanto, los datos que no estén en este sistema deberán ser transformados. Como se ha mencionado en el apartado anterior, Pytorch ofrece la posibilidad de crear clases de tipo "Dataset" que gestionan la carga de datos. Cada conjunto de datos empleará una clase diferente, de tal modo que las funciones de transformación de sistema de referencia serán incluidas en la clase de tipo "Dataset" de los datos de Waymo, en la función que carga dichos datos y los envía a la red neuronal.

A continuación, en la figura 4.3, se muestra el sistema de referencia empleado por los datos procedentes del conjunto de Kitti. Como se puede observar, el sistema de referencia toma su centro en el propio vehículo, concretamente en el sensor LIDAR. El eje X es coincidente con la dirección frontal del vehículo de toma los datos, siendo el eje Y perpendicular al mismo y teniendo su sentido positivo en el lateral izquierdo del vehículo. Por parte del eje Z, el punto cero de este corresponde con la altura del LIDAR, quedando cualquier punto tomado bajo el mismo con una altura negativa. Este sistema de referencia mide la orientación de los elementos detectados mediante el ángulo θ que toma su cero en el eje Y, siendo positivo en sentido horario.

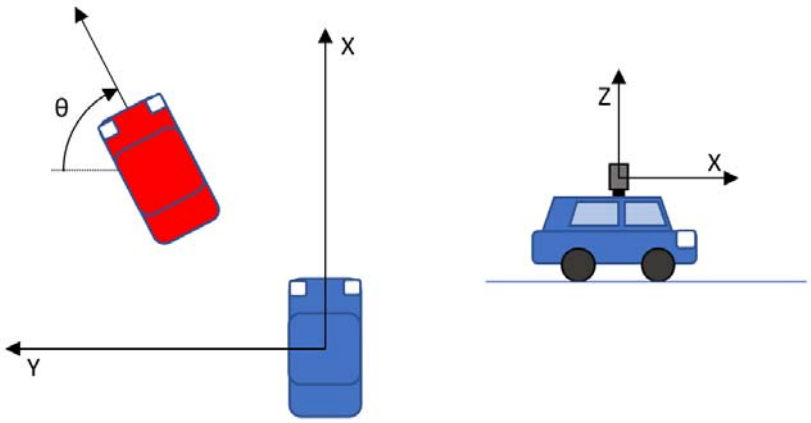

Figura 4.3 – Sistema de referencia de Kitti Dataset.

Por otra parte, Waymo Open Dataset emplea el sistema de referencia alternativo mostrado en la figura 4.4. Al igual que en el caso anterior el sistema de referencia está centrado en el LIDAR del propio vehículo en el plano horizontal, pero no en el plano vertical. En este caso el cero del eje Z está situado a la altura del suelo, lo que implica la necesidad de obtener la altura del LIDAR para realizar la conversión al sistema empleado por Kitti. Por parte de los ejes del plano horizontal, el eje X vuelve a estar orientado hacia la parte frontal del vehículo, mientras que el eje Y se encuentra orientado al lateral izquierdo del vehículo, perpendicularmente al eje X. Finalmente, en este caso, el ángulo de orientación de los elementos detectados θ se referencia al eje X, teniendo valor positivo en el sentido antihorario, a diferencia del caso anterior.

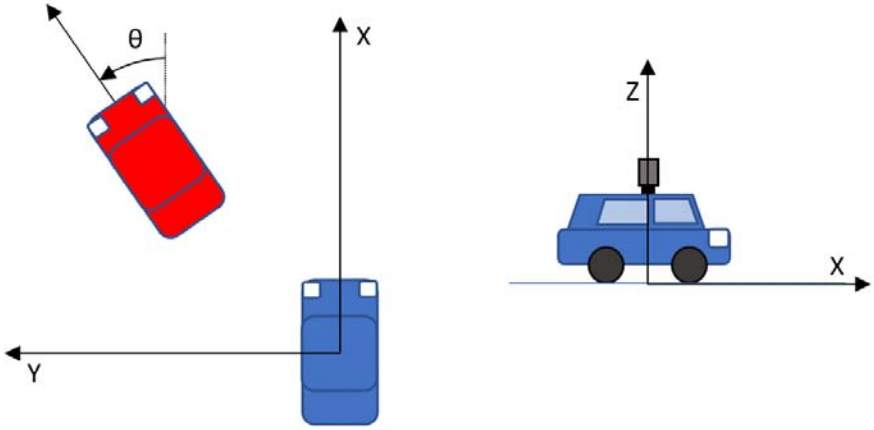

Figura 4.4 – Sistema de referencia de Waymo Open Dataset.

Las ecuaciones de cambio de sistema de referencia de Waymo Open Dataset a Kitti Dataset son las siguientes:

$$z_{Kitti} = z_{Waymo} - h_{LIDAR} \qquad \theta_{Kitti} = \frac{\pi}{2} - \theta_{Waymo}$$

Además de los sistemas de referencia empleados, es necesario abordar el formato empleado para los vectores de posición de cada una de las entidades detectadas por el modelo. El vector de posición para cada elemento detectado se define como $(x_c, y_c, z_c, w, l, h, \theta)$, siendo x_c, y_c y z_c las coordenadas del centro de la bounding box que contiene al elemento detectado, mientras que w, l y h representan el ancho, longitud y altura de esta. Como se ha mencionado anteriormente, θ es el ángulo de orientación de la entidad detectada. Las variables x_c, y_c, w, l y θ son proporcionadas por la tercera etapa de PointPillars, que consta de una red bidimensional de detección en una etapa. Por otra parte, la posición vertical z_c y la altura h son proporcionadas por una red de regresión, como se ha mencionado en el capítulo 3.

4.5 Estructura de los algoritmos

En el presente trabajo de fin de máster se utilizan dos programas principales, los cuales son: el script de entrenamiento de los modelos de PointPillars y el script de simulación de los modelos entrenados en datos de Waymo Open Dataset. Por otra parte, se emplean otros scripts secundarios para la descompresión de los datos de Waymo y creación de archivos de visualización y un script de tipo dataset de Pytorch.

4.5.1 Algoritmo de entrenamiento

Para el entrenamiento del modelo de aprendizaje profundo de detección en nube de puntos se emplea como base código fuente de los autores de PointPillars con algunas modificaciones menores que permiten su ejecución con las versiones de Ubuntu y de las librerías de Python utilizadas. El código fuente con el que se realizaron los ensayos de la publicación se encuentra disponible en [41]. Los algoritmos utilizados en esta referencia son una variación de los que emplea SECOND [32], disponibles en [42].

En la figura 4.5 se muestra un flujograma sobre el funcionamiento básico del programa de entrenamiento. Para empezar, se realiza la carga de un archivo de configuración del modelo que incluye parámetros básicos de la red neuronal como el tamaño de pilares. Con los parámetros procedentes del archivo anterior se realiza la definición de la red neuronal y esta se carga en la GPU para poder beneficiarse de la aceleración por hardware. A continuación, se crean los objetos dataset y dataloader que gestionan la carga de datos del conjunto de datos de Kitti. Mediante el dataloader se realiza la carga de datos y la iteración de entrenamiento del modelo. Cada cierto número de iteraciones se realiza una evaluación del modelo y, finalmente, cuando se han superado las iteraciones programadas se realiza el guardado del modelo.

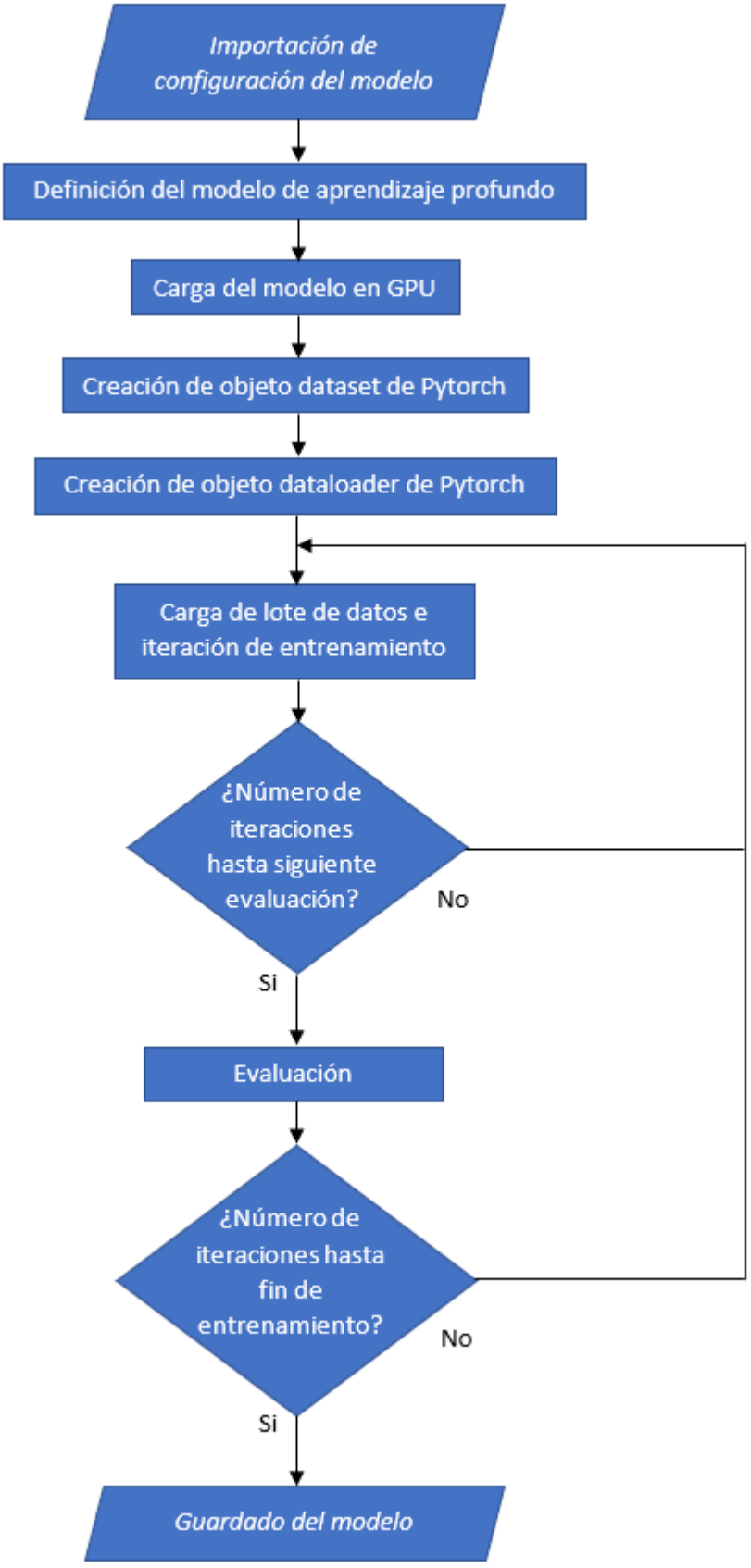

Figura 4.5 – Flujograma del algoritmo de entrenamiento.

En el flujograma anterior no se ha tenido en cuenta el funcionamiento detallado del proceso de entrenamiento y por ello no se muestran etapas como la retro propagación del error y la actualización del modelo. Por otra parte, tras cada evaluación se realiza el guardado del estado del modelo como copia de seguridad. El programa permite la carga de modelos semi entrenados, lo cual resulta imprescindible dado el elevado tiempo de entrenamiento que se requiere para un modelo funcional de PointPillars, que resulta ser de en torno a 22h. En cada evaluación del modelo se realiza el cálculo de precisión media del modelo para cada clase entrenada y para los diferentes niveles de dificultad definidos en Kitti.

4.5.2 Algoritmo de evaluación

El algoritmo de evaluación es el encargado de realizar la simulación de la implementación de la red neuronal entrenada en el apartado anterior. El flujograma de este algoritmo se encuentra en la figura 4.6, en él se puede apreciar un comienzo similar al algoritmo de entrenamiento con la importación de los parámetros de configuración del modelo y la definición del modelo de PointPillars. Dado que el modelo ha sido entrenado previamente, en este paso se realiza la importación de los pesos del modelo guardados al final del algoritmo de entrenamiento. Se generan los objetos dataset y dataloader nuevamente con la diferencia de que esta vez son usados para la detección de objetos en nuevos datos en vez de para entrenamiento. Se realiza el procesado de cada uno de los fotogramas presentes en los escenarios de Waymo Open Dataset evaluados guardando en una lista un diccionario de Python con las bounding boxes generadas, una puntuación de confianza para cada una y una etiqueta que indica la clase del objeto. Mediante una función de evaluación presente en el objeto dataset se evalúa el contenido de la lista de resultados comparándolo con los datos reales proporcionados por el objeto dataset. Tras realizar las asignaciones de verdadero positivo o falso positivo a cada detección que supere el umbral de confianza y encontrar los falsos negativos en cada fotograma, se utiliza una función de evaluación que permite obtener la precisión media (AP) como indicador de precisión del modelo y la “average orientation similariy” (AOS) traducida como similitud de orientación media que mide la precisión de la orientación generada por el modelo para las bounding boxes. Tras finalizar se realiza el guardado de todos los resultados obtenidos en la evaluación del modelo.

Para la generación de los parámetros AP y AOS, durante la evaluación se generan una serie de listas que guardan la información acumulada de precisión, recall y orientation similarity. Estas listas definen las curvas precisión – recall y orientation similarity – recall, con las que se pueden obtener AP y AOS respectivamente como el área bajo esas curvas. Para el cálculo de esa área se realiza una interpolación de 11 puntos en cada curva. Los resultados están en el rango de 0 a 1, siendo 1 un valor que indica para precisión la detección de todos los objetos presentes en el fotograma sin ningún falso negativo ni falso positivo. A la hora del cálculo del AOS se utilizan también los falsos positivos para el cálculo, teniendo por tanto una penalización adicional por cada falso positivo del modelo y dando una puntuación que será menor que el AP.

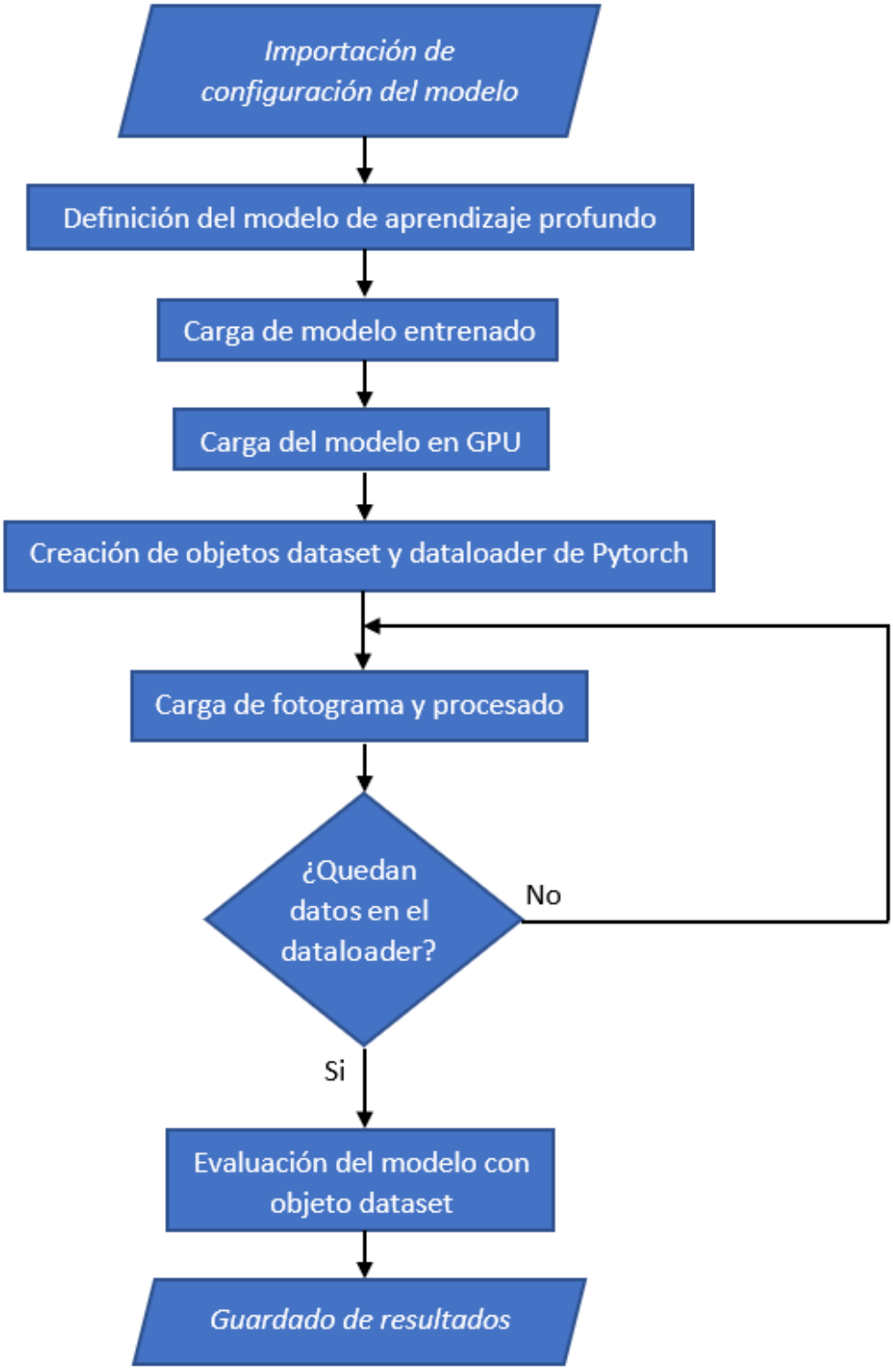

Figura 4.6 – Flujograma del algoritmo de evaluación.

4.5.3 Algoritmos secundarios

Además de los algoritmos de entrenamiento y evaluación hay otros algoritmos menores que se emplean para dar soporte a los mencionados anteriormente. El primer algoritmo de interés consta de una serie de scripts de Python encontrados en [43] que permite la extracción de datos de los archivos de Waymo Open Dataset y proporciona utilidades básicas para la visualización de los datos de cada fotograma. Un ejemplo de visualización se puede encontrar en la figura 4.7, donde se muestra una imagen combinada de las 5 cámaras del vehículo y una representación de la nube de puntos desde una perspectiva cenital con todas las bounding boxes de los objetos presentes en cada etiqueta del fotograma.

Figura 4.7 – Visualización de los datos de Waymo Open Dataset.

El segundo algoritmo de interés ha sido programado para este proyecto con el fin de tener un medio de generar archivos que sirvan de diccionarios a la hora de realizar la carga de datos en el algoritmo de evaluación. Este algoritmo toma la información extraída del anterior algoritmo y genera diccionarios que contienen toda la información de los objetos presentes y la ubicación de los archivos de nube de puntos de cada fotograma para que el objeto dataloader de Pytorch pueda cargarlas. La información de los objetos presentes se emplea en la evaluación del modelo para la identificación de verdaderos positivos y falsos negativos.

Finalmente, el objeto dataset de Pytorch se genera en base al script waymo_dataset.py que ha sido programado como medio de interacción entre el programa de evaluación y los datos de Waymo Open Dataset. Además, este archivo que define la clase de tipo dataset incorpora las funciones de evaluación del modelo. Dichas funciones se encargan del enlazado de las detecciones de la red con los objetos reales presentes en las etiquetas del set de datos, de generar la lista de verdaderos positivos, falsos positivos y falsos negativos, así como de realizar los cálculos de los valores necesarios para obtener las curvas precisión – recall y orientation similarity – recall. Todos los resultados obtenidos junto al AP y AOS son enviados por la clase dataset al programa de evaluación principal mediante un objeto de tipo diccionario que almacena los resultados de la evaluación del modelo.

CAPÍTULO 5

ENTRENAMIENTO Y AJUSTE DEL MODELO

5.1 Introducción

Este capítulo está dedicado al entrenamiento del modelo de aprendizaje profundo en diferentes configuraciones, así como a la evaluación y comparación de dichos modelos mediante sus correspondientes pruebas. Posteriormente, se llevará a cabo la simulación de la ejecución del modelo en un entorno de conducción autónoma alimentado por los datos de Waymo Open Dataset con un ajuste del modelo que permita un desempeño óptimo en la simulación.

5.2 Entorno de trabajo

En este apartado se define el entorno de trabajo, abordando tanto el hardware empleado como el software para la programación, entrenamiento y ensayos aplicados a las redes neuronales de aprendizaje profundo basadas en la arquitectura PointPillars. El equipo de trabajo de la figura 5.1 consiste en un ordenador sobremesa equipado con una GPU NVIDIA RTX 2080 Super con 8GB de VRAM, un procesador Intel Core i5-8500 y 32GB de memoria RAM. A la hora de trabajar en el entrenamiento e inferencia con los modelos de PointPillars, la potencia de una tarjeta gráfica es fundamental, una tarjeta gráfica potente permite la reducción significativa de los tiempos de entrenamiento e inferencia. La CPU no supone un factor limitante durante el trabajo con estos algoritmos, al igual que ocurre con la memoria RAM, dado que la memoria empleada en el trabajo con los modelos de aprendizaje profundo es la VRAM aportada por la tarjeta gráfica. El tamaño máximo de lote empleado en los entrenamientos depende directamente de la cantidad de VRAM disponible. Por otra parte, el sistema operativo de trabajo al igual que los datos están alojados en una unidad SSD SATA, lo que elimina el cuello de botella de las velocidades de transferencia de datos.

Por parte del software de trabajo, el sistema operativo empleado es Ubuntu 20.04. Dado que el software de entrenamiento emplea parte del código del artículo principal de PointPillars [41] y este únicamente ha sido probado en las distribuciones de Ubuntu 18.04 y 16.04, se ha decidido evitar los sistemas operativos de Windows, ampliando así las compatibilidades con scripts y librerías externas. Para aprovechar la aceleración de hardware de la tarjeta gráfica se emplea CUDA 11.2 con las librerías cuDNN 8.0.5.39. El entorno de programación es el IDE Spyder que aporta compatibilidad con los entornos de trabajo de Anaconda.

Todos los scripts se han programado y ejecutado en la versión 3.8.5 de Python. La versión empleada de Pytorch es la 1.7.0, y la de su complemento torchvision es 0.8.1. Para el acceso a los datos de Waymo Open Dataset se emplea la libraría de Python Waymo-open-dataset-tf-2-3-0 versión 1.2.0.1. Finalmente, otras librerías importantes que se deben remarcar son OpenCV 4.4.0 y Numpy 1.18.5.

Figura 5.1 – Equipo de trabajo.

5.3 Entrenamiento

A continuación, se planteará el entrenamiento de diferentes redes neuronales con diferentes parámetros con el fin de realizar una comparativa que permita la elección de las mejores configuraciones para el paso a las siguientes fases del presente proyecto. Se plantea la elección de unos parámetros generales para todas ellas y de los parámetros de ensayo que serán evaluados. Finalmente, se abordarán los resultados de los entrenamientos y se presentarán los parámetros óptimos para el caso práctico de conducción autónoma.

5.3.1 Configuración de entrenamiento

Con el fin de realizar un entrenamiento exitoso se ha decidido adoptar de manera general unos parámetros comunes para las configuraciones de todos los modelos de aprendizaje profundo que se van a generar. En primer lugar, el entrenamiento se centrará en tres tipos de entidades que ambos conjuntos de datos diferencian; automóviles, ciclistas y peatones. Debido a la diferencia de tamaño y a la disminución de la densidad de puntos de la nube con la distancia, la detección de automóviles será posible en distancias más lejanas que en el caso de ciclistas y peatones, por tanto, se definirá un plano de trabajo de mayor tamaño en automóviles que en las dos categorías restantes.

Clases	X min (m)	X max (m)	Y min (m)	Y max (m)	Z min (m)	Z max (m)
Automóvil	0	70	-40	40	-3	1
Peatón Ciclista	0	47	-20	20	-2.5	0.5

Figura 5.2 – Zona de detección por clases.

El área de trabajo, mostrada en la figura 5.2, se mantendrá similar a la empleada en [21], siendo estas dimensiones de en torno a 40m en cada dirección lateral, 70m en la dirección frontal del vehículo, 1m en dirección superior al LIDAR y 3m en la dirección inferior para la detección de automóviles, mientras que para la detección de peatones y ciclistas las dimensiones laterales se reducen a 20m en cada dirección, la distancia frontal a 47m y la superior e inferior a 0.5m y 2.5m respectivamente. Estas distancias son orientativas, ya que las dimensiones exactas dependen del ancho de pilar designado en la configuración del modelo. Se ha determinado que estas dimensiones son más que suficientes para la aplicación en los datos de Waymo Open Dataset como se muestra en la figura 5.3. Dicha figura muestra el primer fotograma de uno de los escenarios recogidos en el conjunto de datos. En ella se han representado las entidades de interés marcadas en los datos del fotograma. El vehículo más lejano que aparece en la sección frontal del vehículo de toma de datos, en la zona derecha de la representación de la nube de puntos (representado con una bounding box de color azul oscuro), está ubicado a una distancia de 44m en el plano horizontal medida desde el centro de la nube de puntos. Dada la representación de la nube de puntos y conocida esta distancia se puede asumir que las dimensiones del área de trabajo serán más que suficientes para contener las entidades diferenciables en la nube de puntos.

Figura 5.3 – Fotograma de Waymo Open Dataset.

Una vez definido el área de trabajo que definirá la pseudo imagen bidimensional de PointPillars, los parámetros que definen la velocidad de aprendizaje se definen basándose en los parámetros empleados en el artículo original dado que se ha demostrado que aportan buenos resultados. Dichos parámetros son una tasa de aprendizaje inicial de 0.0002 y un factor de disminución de la tasa de aprendizaje de 0.8 cada 15 épocas. De igual modo se emplea el optimizador Adam para la optimización de la función de error de PointPillars.

Una vez presentados los parámetros comunes a todos los modelos de entrenamiento, se abordan los parámetros de ensayo y las diferentes configuraciones que definen estos modelos. En la figura 5.4 se muestra una tabla que contiene las configuraciones de los modelos que serán entrenados. Se analizará la clase automóvil por separado, se analizarán las clases peatón y ciclista en otro entrenamiento y se hará una prueba con las tres clases juntas para obtener una comparación del rendimiento con el número de clases detectables. Por otra parte, se realizarán ensayos con diferentes tamaños de pilar, siendo este de sección cuadrada y variando el ancho de esta. Finalmente se variará el número máximo de puntos por pilar y se estudiarán sus efectos en la precisión y tiempo de inferencia.

Configuración	Clases	Ancho de pilar	Puntos por pilar
xyres_16_car	Automóvil	16	100
xyres_16_ped_cycle	Peatón Ciclista	16	100
xyres_16_car_ped_cycle	Automóvil Peatón Ciclista	16	100
xyres_20_car	Automóvil	20	100
xyres_24_car	Automóvil	24	100
xyres_28_car	Automóvil	28	100
xyres_16_car_50p	Automóvil	16	50
xyres_16_car_150p	Automóvil	16	150

Figura 5.4 – Configuraciones de entrenamiento.

Como se ha mencionado en apartados anteriores, el conjunto de datos empleado para los entrenamientos de los diferentes modelos es el conjunto de datos de Kitti Dataset dedicado a la detección de objetos 3D [22]. El set de datos comprende fotogramas de diferentes localizaciones y diferentes condiciones climáticas con el fin de proporcionar un entrenamiento rico en tipología de datos que permita conseguir un modelo lo más generalista posible. Kitti proporciona una clasificación de dificultad de todos los objetos en función del porcentaje de visibilidad de estos con el fin de evaluar el modelo posteriormente para cada dificultad.

5.3.2 Resultados del entrenamiento

Una vez han sido entrenados los 8 modelos con el conjunto de datos Kitti se reúnen las condiciones suficientes para realizar una evaluación de estos. Para realizar la evaluación se sigue la metodología oficial marcada por Kitti en [38] para detección de objetos 3D, representando los resultados en AP (Average Precision) o precisión media. Los datos se han evaluado utilizando el conjunto de validación, dado que el conjunto de datos de prueba no contiene etiquetado de las entidades pertenecientes a cada fotograma. Dado que el conjunto de validación no se ha usado para el entrenamiento este ensayo es totalmente válido.

Los resultados de precisión se han obtenido para diferentes valores de solapamiento utilizando el parámetro intersección sobre unión IoU. En la figura 5.5 y 5.6 se muestra una tabla comparativa de los resultados obtenidos en la evaluación de velocidad de inferencia y precisión media de detección 3D aplicada a vehículos. Análogamente en las figuras 5.7 y 5.8 se pueden apreciar los resultados para la clase de detección peatones y en las figuras 5.9 y 5.10 los resultados de la clase ciclistas.

Configuración	Velocidad de inferencia (Hz)	Automóvil (IoU = 0.7)		
		Fácil (AP)	Moderado (AP)	Difícil (AP)
xyres_16_car	23.08	86.83	76.61	70.26
xyres_16_car_ped_cycle	19.41	81.05	74.76	68.99
xyres_20_car	29.02	82.42	75.73	69.29
xyres_24_car	32.47	86.17	76.41	69.91
xyres_28_car	35.48	79.05	69.34	67.18
xyres_16_car_50p	25.89	81.48	76.34	70.32
xyres_16_car_150p	25.73	83.93	75.42	69.01

Figura 5.5 – Resultados de entrenamiento en automóviles (IoU = 0.7).

Configuración	Velocidad de inferencia (Hz)	Automóvil (IoU = 0.5)		
		Fácil (AP)	Moderado (AP)	Difícil (AP)
xyres_16_car	23.08	90.71	89.64	88.85
xyres_16_car_ped_cycle	19.41	90.76	89.71	89.08
xyres_20_car	29.02	90.68	89.62	88.82
xyres_24_car	32.47	90.77	89.76	88.88
xyres_28_car	35.48	90.69	89.56	88.25
xyres_16_car_50p	25.89	90.81	89.88	89.09
xyres_16_car_150p	25.73	90.66	89.69	88.50

Figura 5.6 – Resultados de entrenamiento en automóviles (IoU = 0.5).

Configuración	Velocidad de inferencia (Hz)	Peatón (IoU = 0.5)		
		Fácil (AP)	Moderado (AP)	Difícil (AP)
xyres_16_ped_cycle	18.53	64.80	59.11	55.53
xyres_16_car_ped_cycle	19.41	16.32	14.18	13.57

Figura 5.7 – Resultados de entrenamiento en peatones (IoU = 0.5).

Configuración	Velocidad de inferencia (Hz)	Peatón (IoU = 0.25)		
		Fácil (AP)	Moderado (AP)	Difícil (AP)
xyres_16_ped_cycle	18.53	83.50	79.49	76.45
xyres_16_car_ped_cycle	19.41	42.08	38.38	35.98

Figura 5.8 – Resultados de entrenamiento en peatones (IoU = 0.25).

Configuración	Velocidad de inferencia (Hz)	Ciclista (IoU = 0.5)		
		Fácil (AP)	Moderado (AP)	Difícil (AP)
xyres_16_ped_cycle	18.53	81.23	60.88	57.41
xyres_16_car_ped_cycle	19.41	46.07	28.74	27.50

Figura 5.9 – Resultados de entrenamiento en ciclistas (IoU = 0.5).

Configuración	Velocidad de inferencia (Hz)	Ciclista (IoU = 0.25)		
		Fácil (mAP)	Moderado (mAP)	Difícil (AP)
xyres_16_ped_cycle	18.53	84.63	64.96	60.47
xyres_16_car_ped_cycle	19.41	66.17	46.03	43.91

Figura 5.10 – Resultados de entrenamiento en ciclistas (IoU = 0.25).

Como puede observarse en el caso de la clase automóvil, para un IoU de valor 0.5, todos los modelos obtienen un AP en torno al 90%, por lo que para esta clase es más adecuado estudiar los resultados en un IoU de 0.7 (figura 5.5). En la figura 5.11 se muestra una representación de los resultados de precisión obtenidos con IoU de 0.7 para cada tamaño de columna en la clase automóvil divididos en tres grupos según su dificultad. Como se esperaba se aprecia una disminución de la precisión media con el aumento del tamaño de sección, con el caso atípico de la sección de 24 cm de lado. Para la dificultad moderada se observa un resultado bajo en la sección de 28 cm.

La variación del número de puntos máximos por cada pilar no parece tener efectos significativos en los resultados, por tanto, se puede asumir que al menos para la clase automóvil no resulta beneficioso realizar el aumento de este umbral, ni en términos de precisión ni en términos de velocidad de inferencia.

En cuanto a la velocidad de inferencia, como era de esperar, se aprecia en la figura 5.12 un claro aumento de la velocidad a medida que aumenta el tamaño de sección, dado que la pseudo imagen generada adquiere un tamaño menor. Por su parte, la velocidad de inferencia a 16 mm de 23.08 Hz resulta muy inferior a la velocidad mencionada en la publicación de PointPillars [21] de 42.4 Hz obtenida con Pytorch, siendo 61.7 Hz con TensorRT. Este cambio de velocidad puede deberse al hardware empleado, dado que en el artículo original se emplea una GPU NVIDIA GTX 1080Ti con 11 GB de VRAM, mayor memoria gráfica que la empleada en los ensayos de este trabajo de fin de máster.

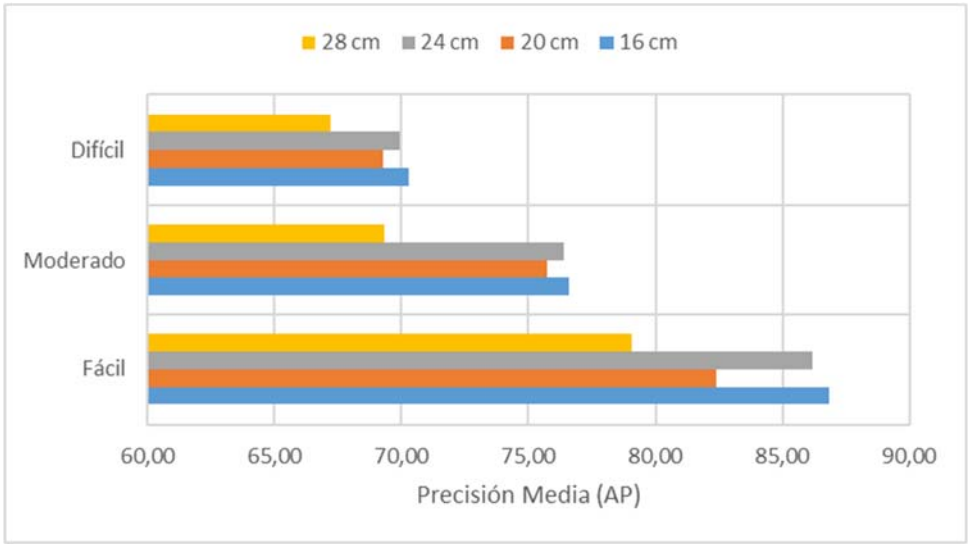

Figura 5.11 – Precisión frente a tamaño de columna (IoU = 0.7).

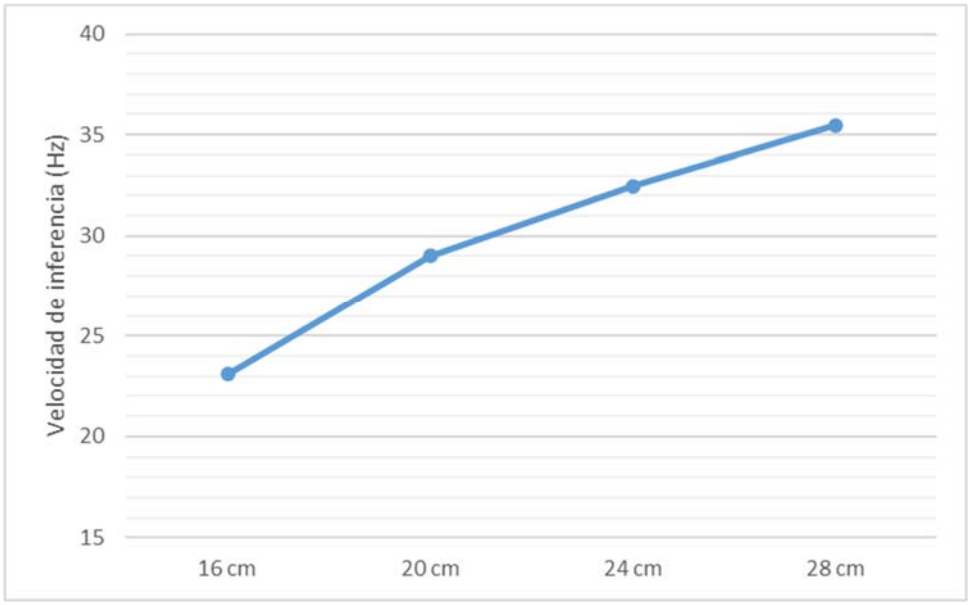

Figura 5.12 – Velocidad de inferencia frente a tamaño de columna.

Otro ensayo interesante es el estudio de la variación de la velocidad de inferencia con el número de clases detectables por el modelo. En la figura 5.13 se puede apreciar una tendencia descendente de la velocidad de inferencia con el número de clases buscadas, con el caso atípico de una velocidad levemente mayor en el modelo de automóviles, peatones y ciclistas que en el modelo de peatones y ciclistas únicamente. Dado que los modelos xyres_16_car y xyres_16_ped_cycle fueron entrenados varios meses antes que el resto es posible que las actualizaciones de los controladores de la GPU hayan mejorado la eficiencia de los núcleos CUDA empleados en las operaciones relacionadas con el aprendizaje profundo.

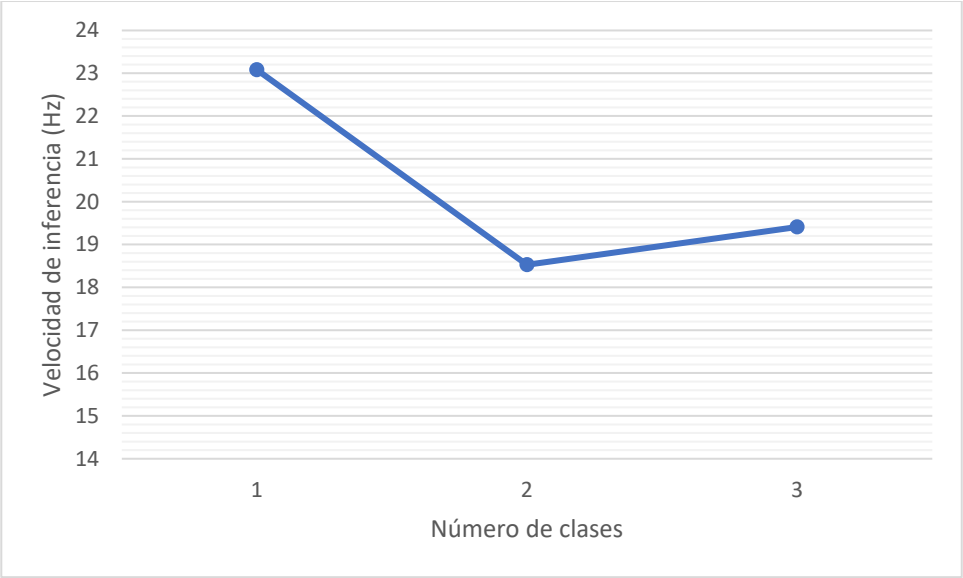

Figura 5.13 – Velocidad de inferencia frente a número de clases.

Se puede apreciar una drástica bajada de precisión media por parte de las clases peatón y ciclista entre los modelos xyres_16_ped_cycle y xyres_16_car_ped_cycle. La particularidad del modelo que abarca las tres clases de detección resulta ser la inclusión de 2 rangos diferentes de medición, uno para automóviles y otro diferente para el resto de las clases. Esta particularidad estaría afectando muy negativamente al modelo conjunto, lo que hace que no se buena idea realizar una implementación conjunta de todas las clases de detección.

Por otra parte, se puede observar cómo aumenta claramente la precisión media con forme disminuye el IoU en todas las clases. En la clase automóvil la AP permanece casi constante para IoU de valor 0.5 pero variable en 0.7. Una posible explicación sería que al ser 0.7 un IoU mucho más exigente que 0.5, el aumento de resolución de la pseudo imagen como consecuencia de la disminución del tamaño de pilar haya permitido un mayor solapamiento entre la bounding box propuesta y la bounding box real. Por parte de ciclistas y especialmente peatones se hace imprescindible emplear tamaños de columna lo más bajos posibles dadas las dimensiones horizontales de sus respectivas bounding boxes, por ello mismo es necesario también rebajar el parámetro IoU a valores bajos como 0.25 para aumentar la tasa de detección.

Tras el análisis realizado se ha llegado a la conclusión de que para la implementación en la simulación con datos de Waymo Open Dataset se van a emplear modelos con secciones de pilar de 16 cm de lado. Por parte del número de clases detectadas en un mismo modelo se ha decidido que se empleará un único modelo para la detección de automóviles. Por tanto, el modelo que se va a emplear es el modelo ya entrenado xyres_16_car. Por otra parte, la selección del parámetro intersección sobre unión (IoU) se realizará en el siguiente apartado junto a otros parámetros de interés.

5.4 Simulación preliminar y ajuste de los parámetros

Una vez que el modelo ha sido entrenado, a la hora de implementarlo en un nuevo conjunto de datos es necesario realizar el ajuste de unos determinados parámetros. Los parámetros mencionados deben ajustarse de manera manual, por lo que es necesario emplear parte del conjunto de datos para esta finalidad. De los 24 escenarios del set de datos de 20GB descargados de Waymo Open Dataset, se van a utilizar 3 para el ajuste de parámetros y los 21 restantes para la simulación y evaluación.

Uno de los parámetros que es necesario ajustar es el umbral de detección. Cuando el modelo genera detecciones con sus respectivas posiciones, también se genera una puntuación que puede interpretarse como la confianza del modelo sobre esa predicción. Para separar los verdaderos positivos de los falsos positivos es necesario realizar un ajuste experimental de este umbral. Para ello se ha realizado la evaluación del modelo xyres_16_car con diferentes valores de umbral de confianza, calculando valores de precisión y recall totales y así obteniendo los resultados mostrados en la figura 5.14. La precisión se define como el tanto por uno de verdaderos positivos sobre las detecciones totales del modelo, mientras que el recall se define como el tanto por uno de verdaderos positivos sobre los objetos totales en los datos.

$$Precisión = \frac{VP}{VP + FP} \qquad Recall = \frac{VP}{VP + FN}$$

Umbral de confianza	Precisión	Recall	Precisión · Recall
0.05	0.0881	0.6945	0.0612
0.10	0.2405	0.6679	0.1606
0.15	0.3844	0.6504	0.2500
0.20	0.5009	0.6345	0.3178
0.25	0.5942	0.6205	0.3687
0.30	0.6775	0.6066	0.4110
0.35	0.7397	0.5886	0.4354
0.40	0.7939	0.5706	0.4530
0.45	0.8382	0.5500	0.4610
0.50	0.8766	0.5324	0.4667
0.55	0.9056	0.5118	0.4635
0.60	0.9325	0.4915	0.4583
0.65	0.9555	0.4699	0.4489
0.70	0.9747	0.4464	0.4351
0.75	0.9847	0.4142	0.4079
0.80	0.9925	0.3769	0.3740
0.85	1.0000	0.3271	0.3271
0.90	1.0000	0.2344	0.2344
0.95	1.0000	0.0863	0.0863

Figura 5.14 – Tabla de resultados de variación de umbral de confianza.

La precisión da información de cuantos de los objetos detectados son correctos y el recall da la información de cuantos de los objetos totales se han detectado e identificado correctamente. Dado que con forme aumenta el umbral de confianza aumenta la precisión y disminuye el recall, es necesario apoyarse en un parámetro combinado de los dos, que en este caso será el producto de estos. Siguiendo este último parámetro el umbral de confianza óptimo se encuentra en 0.5 con un valor del producto entre precisión y recall de 0.4667. Se ha estimado que 0.5 es un valor aceptable para el parámetro IoU, por tanto, todos los ensayos se han realizado con este valor, de igual modo en futuros apartados se empleará el mismo valor.

Finalmente, para evaluar el modelo correctamente es necesario definir un valor umbral de número de puntos por objeto para elegir que objetos del set de datos son aceptables para utilizarse en los ensayos, dado que en los datos de Waymo Open Dataset se realizan etiquetados de todos los objetos presentes, aunque presenten una cantidad de puntos ínfima. En un caso se ha observado un objeto etiquetado representado únicamente por 4 puntos, lo que justifica la necesidad de establecer el umbral antes de realizar los ensayos. Dicho valor de puntos mínimos se ha establecido en 125 puntos para la detección de automóviles.

CAPÍTULO 6

EVALUACIÓN Y DISCUSIÓN DE LOS RESULTADOS

6.1 Introducción

En este capítulo se aborda la simulación final del modelo entrenado en los datos de Waymo Open Dataset con la evaluación de este mediante los principales parámetros de interés. Se realizará la simulación en los datos restantes para evitar influencias de los escenarios usados para el ajuste del modelo. Se analizarán las principales fuentes de error en la precisión del modelo y finalmente se realizará una discusión de los resultados.

6.2 Evaluación

Una vez realizado el ajuste de parámetros del modelo entrenado se reúnen las condiciones para realizar las simulaciones del modelo en el resto de los datos de Waymo Open Dataset. Los datos empleados constan de 21 recorridos de en torno a 200 fotogramas a una tasa de 10 fotogramas por segundo. En total el ensayo cubre 4166 fotogramas con 30202 automóviles en total.

Al introducir la totalidad de los datos de evaluación en el modelo PointPillars xyres_16_car se obtienen en primer lugar los resultados de tiempos de inferencia mostrados en la figura 6.1. Las etapas por las que pasan los datos comienzan con la carga de estos en la tarjeta gráfica tras transformarse al tipo de dato empleado por Pytorch, seguidamente se realiza un pre procesado que elimina el exceso de datos externo a la zona de trabajo y prepara la entrada a la red neuronal de aprendizaje profundo. Posteriormente los datos son procesados por las etapas de la arquitectura PointPillars que ofrece como salida las detecciones 2D. La última etapa emplea una red de regresión les da a las bounding boxes un carácter tridimensional.

Etapa	Tiempo medio (ms)
Carga de los datos de inferencia a tarjeta gráfica	4.496
Preparación	18.830
PointPillars: Pillar feature net	9.554
PointPillars: Backbone	0.965
PointPillars: Detection head	21.442
Regresión altura y posición Z	2.972
Total	58.259

Figura 6.1 – Tabla de tiempos por etapa del modelo xyres_16_car.

Los resultados de velocidad de la simulación arrojan una velocidad de inferencia media de 16.99 Hz, unos 6 Hz inferior a los 23.08 Hz obtenidos en el mismo modelo con los datos de Kitti dataset. Este aumento del tiempo de procesado por fotograma se achaca a que con los datps Kitti se realiza el preprocesado antes de evaluar la red. Así mismo la velocidad de inferencia resultante se considera totalmente aceptable dado que los datos de Waymo Open Dataset están recogidos a una tasa de 10 fotogramas por segundo, por lo que todos los fotogramas son procesados por la red.

Además de los resultados de velocidad media de inferencia se obtienen los parámetros que determinan la calidad del modelo entrenado. En la tabla 6.2 se muestran los resultados de AP para un IoU de 0.5 y un umbral de confianza de 0.5 comparados con los obtenidos con los datos de Kitti Dataset tras el entrenamiento. El AOS resultante de la evaluación con los datos de Waymo tiene un valor de 0.4227.

Set de datos (IoU = 0.5)	AP (ms)
Waymo Open Dataset – Evaluación	0.5313
Kitti Dataset – Easy	0.9071
Kitti Dataset – Moderate	0.8964
Kitti Dataset – Hard	0.8885

Figura 6.2 – Tabla de AP del modelo xyres_16_car en los diferentes sets de datos.

Claramente hay un descenso en la precisión media de los datos con respecto a los datos de Kitti Dataset empleados para evaluar el modelo tras su entrenamiento. Se esperaba una disminución de la AP, sin embargo, una reducción de 0.35 en el caso más favorable resulta excesiva. En el siguiente apartado se analizarán las posibles causas de este cambio. El AOS de valor 0.4227 es un 20% menor que el AP, lo que es indicador de buen funcionamiento de la estimación de la orientación de los verdaderos positivos.

Una vez determinados los parámetros principales de la calidad de la red pueden presentarse las curvas que dan lugar a ellos. El AP es el área abarcada bajo la curva precisión – recall mostrada en la figura 6.3, mientras que el AOS es el área bajo la curva orientation similarity – recall, mostrada en la figura 6.4.

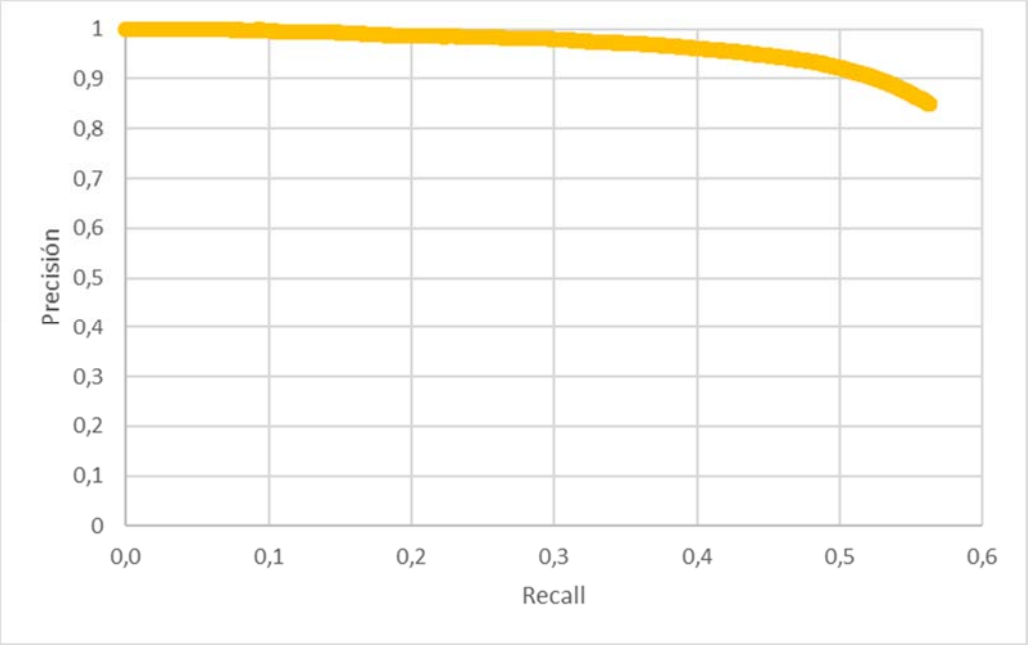

Figura 6.3 – Precisión – Recall con IoU 0.5.

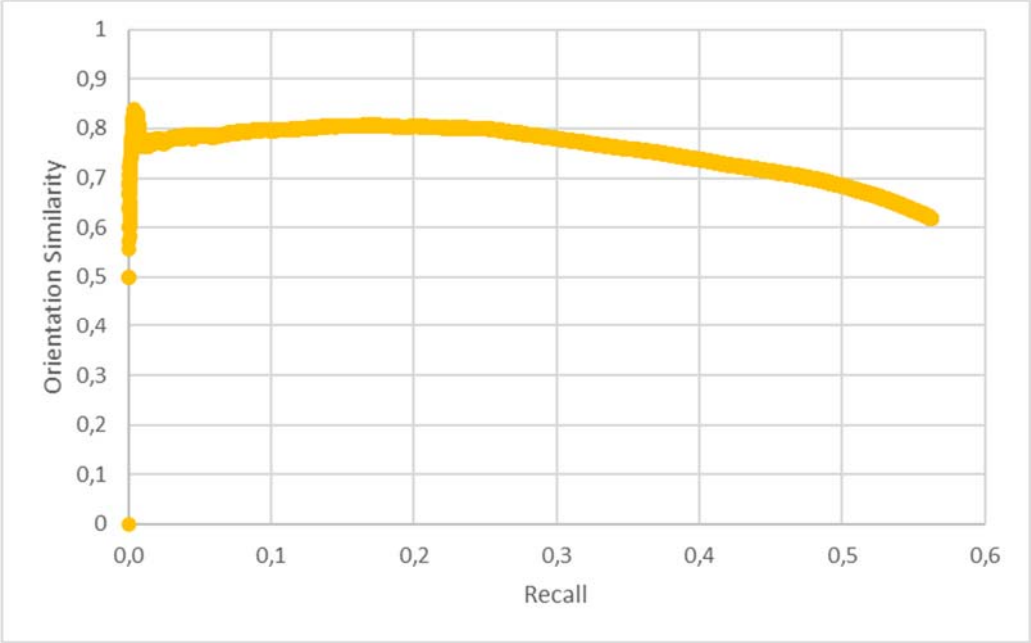

Figura 6.4 – Orientation similarity – Recall con IoU 0.5.

Finalmente, resulta de interés obtener las mismas gráficas anteriores para diversos valores de IoU, lo que permite ver la variación de los parámetros de calidad AP y AOS. Dichas gráficas se presentan en las figuras 6.5 y 6.6.

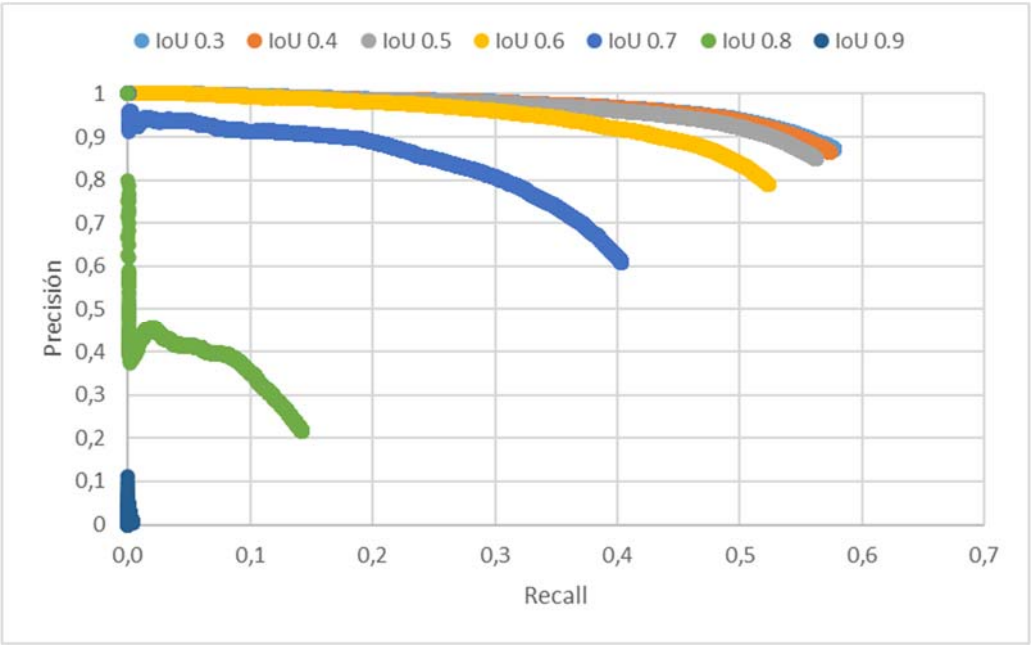

Figura 6.5 – Precisión – Recall con variación de IoU.

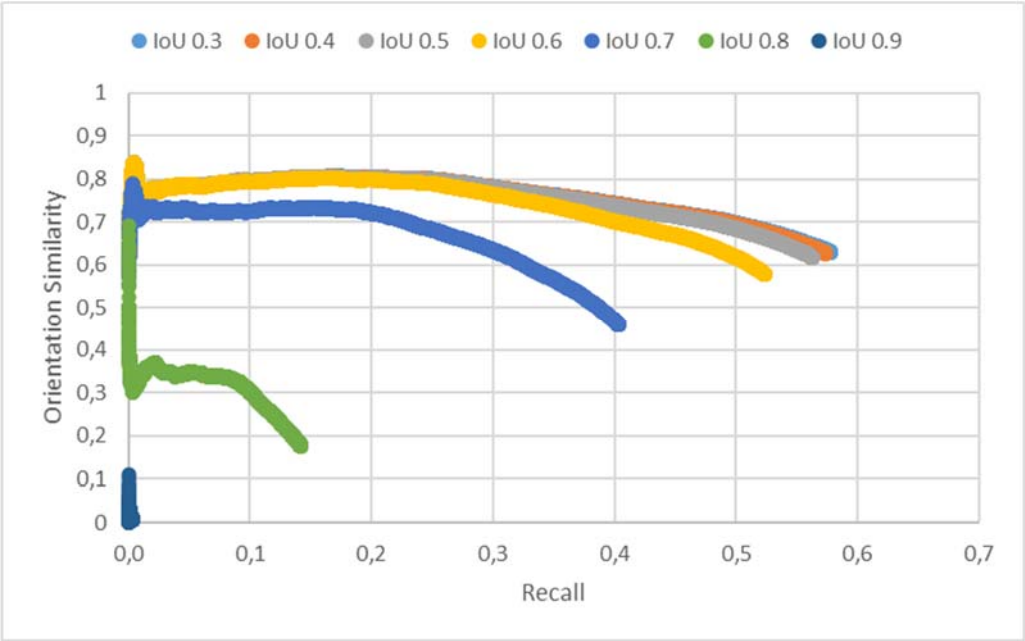

Figura 6.6 – Orientation similarity – Recall con variación de IoU.

6.3 Fuentes de error

Con el fin de encontrar las principales causas de disminución de AP del modelo se realizan ensayos por separado de escenarios de características particulares, variando las condiciones climáticas y las condiciones del tráfico. El primer escenario consiste en un entorno de conducción con una calzada que sufre sucesivos cambios de nivel y curvas. El recorrido se ha grabado de día con cielo nuboso y lluvia leve. Tras la realización de la evaluación se obtiene un AP de 0.5367 y un AOS de 0.5194 con una velocidad de inferencia de 13.98 Hz. El fotograma mostrado en la figura 6.7 se ha considerado uno de los más desfavorables, dando como resultado de la detección 3 verdaderos positivos, 0 falsos positivos y 6 falsos negativos. Como se puede observar en la figura mencionada, este fotograma contiene una serie de vehículos aparcados en batería en la calle perpendicular a la transitada, produciéndose en este momento un solapamiento entre los mismos. Los elementos detectados han tenido puntuaciones de confianza superiores a 0.77. Los dos vehículos negros mostrados en la figura 6.7 no han sido detectados a pesar de encontrarse a una distancia razonablemente cercana al vehículo de toma de datos. Otros dos de los falsos negativos presentan su nube de puntos cortada por encontrarse en la zona alta de un cambio de nivel y cortar el límite de la zona de trabajo la bounding box de estos. Oros dos de los vehículos no detectados son vehículos que presentan una alta cercanía al vehículo de toma de datos pero que se encuentran muy solapados por vehículos cercanos. El sistema de evaluación detecta que el número de puntos de estos supera el umbral y por tanto los considera candidatos detectables a pesar de dejar ver una porción muy pequeña de los mismos.

El factor de cambios de nivel en el entorno de conducción limita la detección de vehículos al tener unas alturas de trabajo muy limitadas. Por otra parte, se encuentran errores

de detección de vehículos muy próximos y errores de detección en vehículos que presentan una alta dificultad de detección.

Figura 6.7 – Escenario con desniveles.

El segundo escenario de conducción consta de un cruce con alta densidad de vehículos cercanos encontrándose el vehículo de toma de datos detenido por un semáforo en rojo mientras se produce la circulación en el carril perpendicular de escasos vehículos y un autobús urbano doble. Los resultados obtenidos en este escenario son AP de valor 0.2723, AOS de valor 0.2560 y velocidad de inferencia de 14.41 Hz. En el análisis de este escenario se particulariza en el fotograma mostrado en la figura 6.8, donde hay un verdadero positivo, 7 falsos negativos y 0 falsos positivos. De los 7 falsos positivos es razonable el fallo en la detección de un autobús de 2 vehículos y una excavadora, sin embargo, también se produce el fallo de detección de los 3 vehículos cercanos al vehículo de detección. El vehículo restante, ubicado a 48m de distancia se había detectado con una puntuación de confianza de 0.13, por esa razón no se ha marcado como positivo.

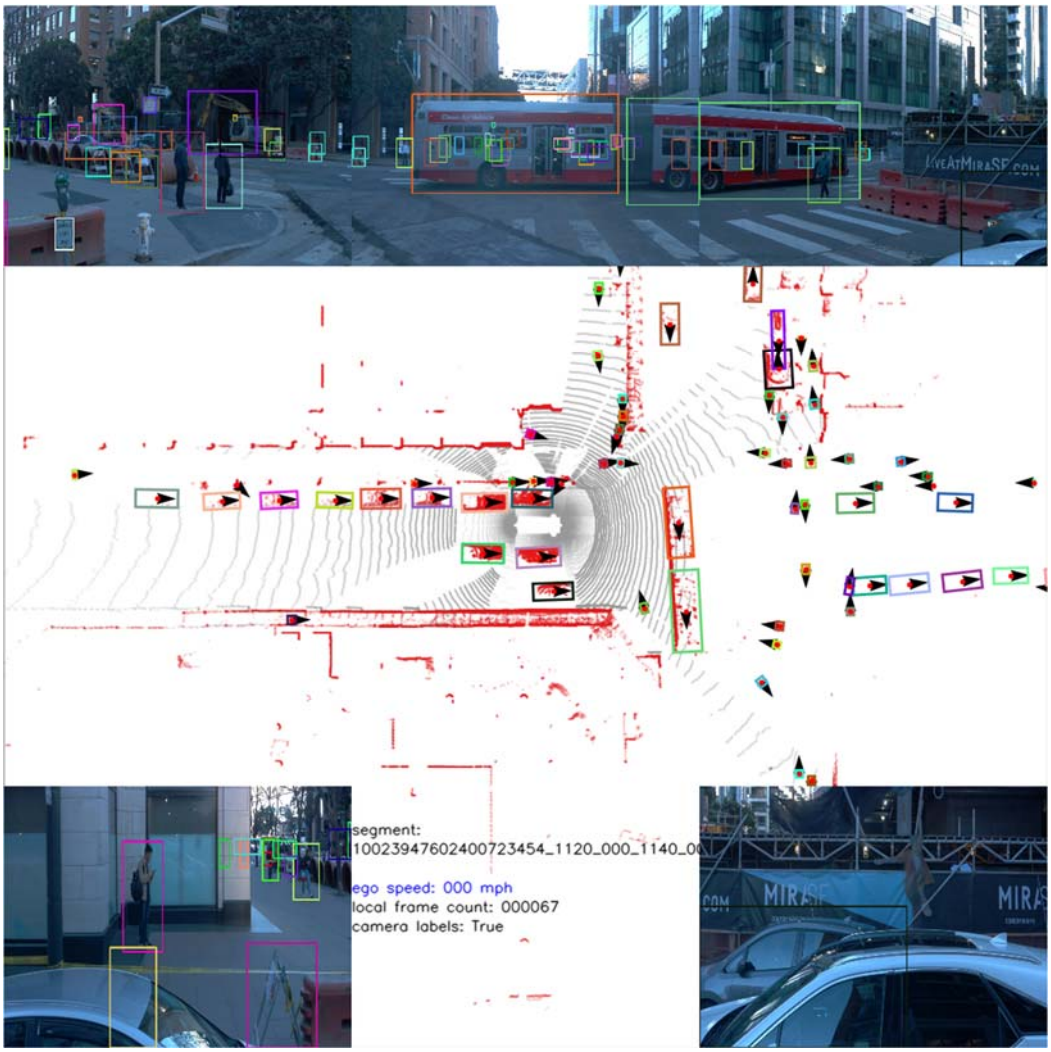

Figura 6.8 – Escenario de espera en cruce.

El tercer escenario analizado tiene unas condiciones climáticas lluviosas y los datos están tomados en una zona residencial con vehículos aparcados en batería, lo que pone a prueba la efectividad de la red con un alto solapamiento de los vehículos. El AP del ensayo tiene un valor de 0.4447, el AOS de 0.3857 y la velocidad de inferencia resulta de 13.89 Hz. En este escenario se ha elegido el fotograma mostrado en la figura 6.9 por considerarse el más desfavorable, presentando 3 verdaderos positivos, 0 falsos positivos y 9 falsos negativos. De los 9 elementos no detectados 1 de ellos se encontraba a una distancia de 51m, 4 de ellos eran muy cercanos al vehículo de muestreo y otros 3 tenían la mayor parte solapada por otros vehículos o muros de viviendas. La lluvia no parece haber tenido más efectos negativos de los ya conocidos por los análisis de escenarios anteriores, sin embargo, el aparcamiento en batería provoca un solapamiento entre vehículos que dificulta enormemente la detección.

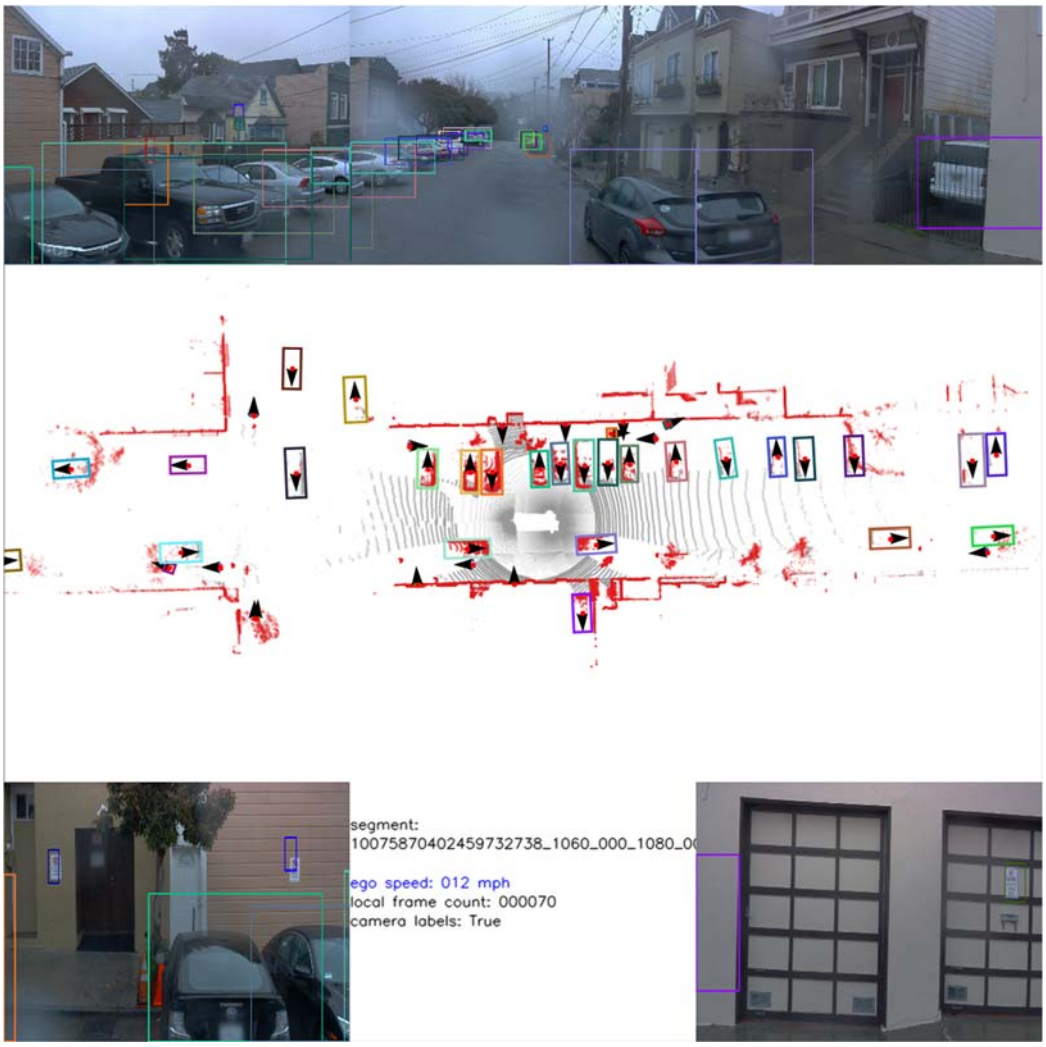

Figura 6.9 – Escenario de lluvia y alto solapamiento.

El cuarto escenario tiene la particularidad de haberse muestreado de noche en condiciones de muy baja iluminación. A pesar de la alta oscuridad del escenario el modelo ha conseguido unos resultados de AP de 0.7240 y 0.4923 de AOS, con una velocidad de inferencia de 13.07 Hz. Resulta atípico encontrar una diferencia tan grande entre el AOS y el AP, puede decirse que en este escenario se ha cometido gran error en la asignación de la orientación de las detecciones. Aunque la mayor parte del recorrido transcurre con una baja cantidad de falsos negativos, la figura 6.10 muestra el fotograma más desfavorable con 4 verdaderos positivos, 0 falsos positivos y 6 falsos negativos. Igual que en los casos anteriores vuelve a haber problemas de detección con los automóviles más cercanos, encontrándose en este problema 3 de los 6 falsos negativos. El resto de los elementos no detectados han obtenido puntuaciones de confianza de 0.46, 0.48 y 0.49, muy cercanas al 0.5 necesario para asignarlos como verdaderos positivos. Estos 3 últimos candidatos no han obtenido mayores puntuaciones debido a solapamiento de otros objetos para dos de ellos y por encontrarse el tercero a una distancia de 47.5m del vehículo de toma de datos.

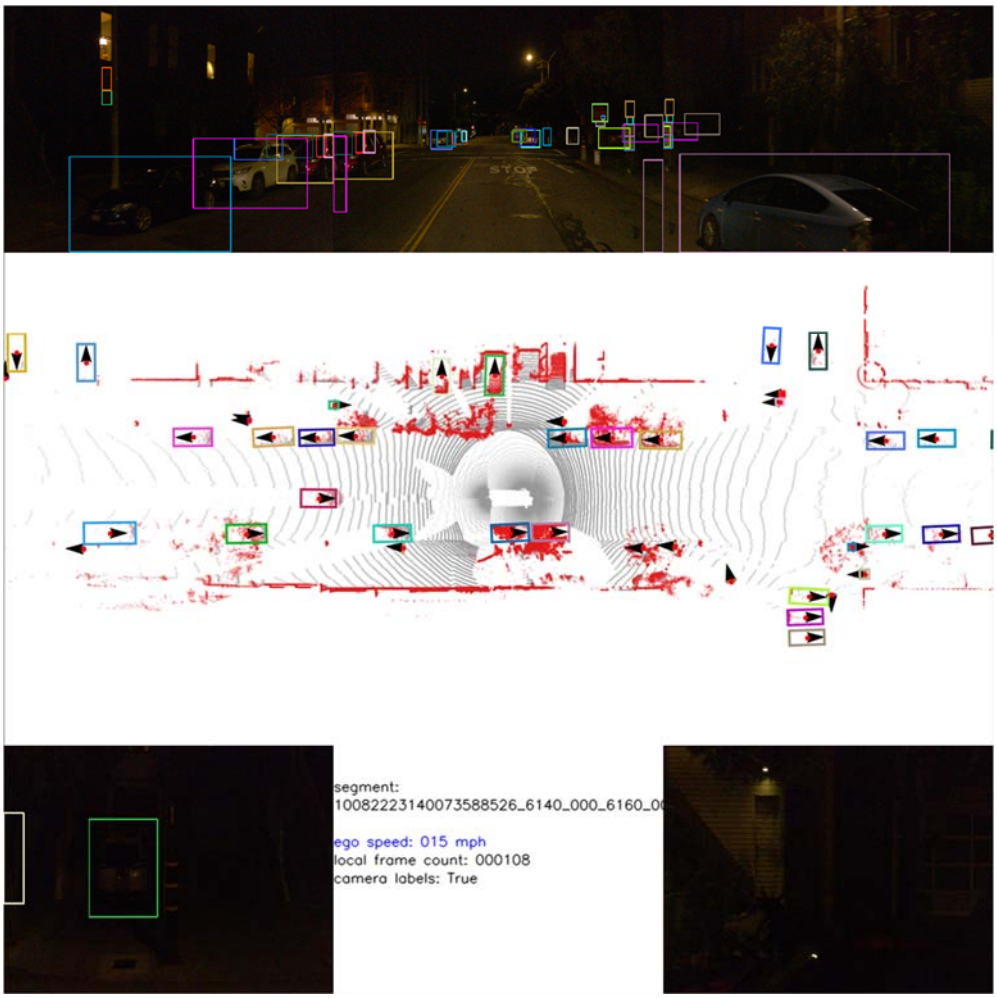

Figura 6.10 – Escenario nocturno de baja iluminación.

El siguiente escenario pone a prueba el modelo en un entorno de funcionamiento diferente a los anteriores como es el caso de una autopista. Este escenario no resulta adecuado para la determinación de AP y AOS dado que los vehículos que circulan por el sentido contrario no han sido etiquetados en los datos. La cantidad total de falsos positivos como consecuencia de esta falta de etiquetado es de 668 en todos los fotogramas del escenario, con 259 falsos positivos y 52 falsos negativos. No obstante, se ha obtenido un AP de 0.6472 y AOS de 0.6470, que indican un muy buen rendimiento conociendo la gran cantidad de falsos positivos como consecuencia de la falta de etiquetado. El número de falsos positivos es muy superior al de falsos negativos sumado al de falsos positivos debido a que en sentido contrario por la diferencia de velocidad se observa una mayor cantidad de vehículos. En la figura 6.11 se muestra un fotograma de este escenario en el que se han encontrado 2 verdaderos positivos, 3 falsos positivos pertenecientes a vehículos del sentido inverso y 0 falsos negativos. Resulta de interés en este escenario observar los efectos de la velocidad en los puntos del LIDAR que gira de modo continúo refrescando los puntos del ángulo orientado por este sensor mientras que el resto se mantiene constante.

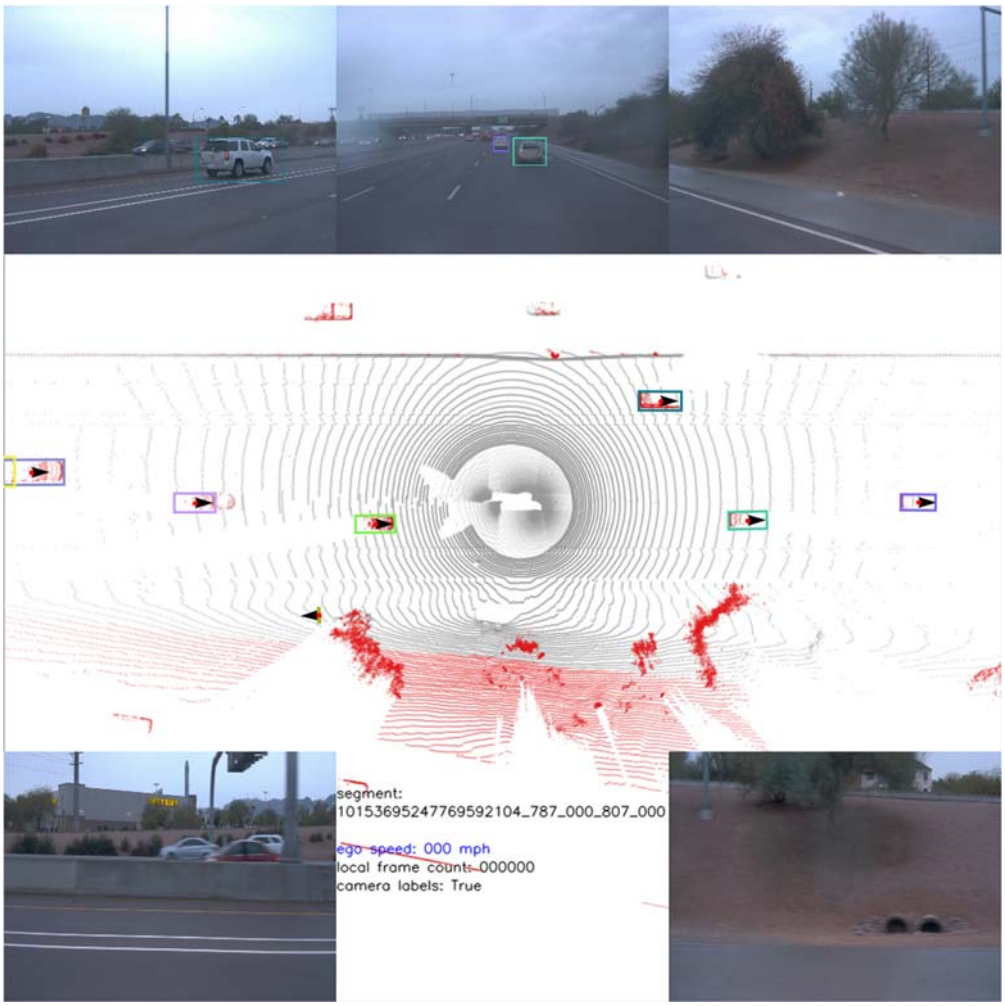

Figura 6.11 – Escenario de autopista.

6.4 Discusión de los resultados

Tras la evaluación del modelo xyres_16_car entrenado con la totalidad de los 21 escenarios restantes de Waymo Open Dataset y el posterior análisis de escenarios que presentan condiciones particulares se reúnen las condiciones suficientes para realizar una correcta discusión de los resultados obtenidos. Por parte de la evaluación realizada en el apartado 6.2 que contenía los 21 escenarios, se obtuvo una precisión media AP de valor 0.5313, mucho menor de lo que se esperaba obtener y mucho menor de las obtenidas para el mismo IoU de valor 0.5 en los datos de Kitti Dataset, que rondaban el 0.9. Esta bajada de precisión se puede interpretar como una pérdida de calidad como consecuencia de la simulación de la implementación de un modelo pre entrenado con los datos de Kitti en un vehículo autónomo diferente. El AOS resultó ser en este ensayo completo de valor 0.4227, en torno a un 20% menor que el AP, lo que indica que en la mayor parte de los casos se consigue una buena predicción de la orientación en los verdaderos positivos, aunque permaneciendo una pequeña parte de ellos con orientaciones inversas. Por parte de la velocidad de inferencia media de 16.99 Hz, menor que los 23.08 Hz obtenidos con los datos de Kitti Dataset, resulta suficiente para la simulación de la implementación en datos tomados a 10 Hz.

A la hora de realizar un análisis detallado en escenarios de condiciones particulares, se obtuvieron dos tipos de resultados. En primer lugar, se observaron unos patrones generales en todos ellos que indican la principal fuente de error de este modelo en los datos de Waymo Open Dataset y en segundo lugar unas conclusiones particulares de cada escenario característico. El principal problema general observado ha sido la ausencia de detecciones en distancias muy cercanas al vehículo de medición, concretamente no se han encontrado verdaderos positivos en distancias menores a los 8m aproximadamente. Este inconveniente puede deberse a 3 factores, el primero es que el modelo no se haya entrenado con vehículos cercanos al vehículo de muestreo, lo cual se desmiente rápidamente al observar las imágenes del set de datos de Kitti. El segundo factor posible es una diferencia significativa en la intensidad recibida por los puntos de los vehículos tan cercanos entre los sets de datos de Kitty y de Waymo, tan grande que ocasione este fenómeno. La tercera posible causa de fallo en elementos cercanos puede ser una alta diferencia de altura entre el LIDAR de los datos de Kitti y el LIDAR empleado por Waymo. Este supuesto parte de la posibilidad de que una altura reducida del LIDAR de Kitti no consiga vistas superiores de los vehículos cercanos, y por tanto una nube de puntos obtenida desde una posición superior con Waymo consiga perspectivas desconocidas para el modelo entrenado con Kitti. En la figura 6.12 se muestran los vehículos de toma de datos de Kitti y Waymo, aunque dado que en la descarga de datos de Waymo no se ofrecen detalles acerca del vehículo empleado no se puede asumir que se trate del mismo. Dada la diferencia de altura de ambos vehículos, este supuesto es el que más probabilidades tiene de explicar el problema de detección de elementos cercanos.

Figura 6.12 – Vehículos de toma de datos de Kitti (izquierda) y Waymo (derecha).

Otras causas de error generales son la evaluación de elementos cuya superficie resulta escasamente visible y que presentan los suficientes puntos para superar el umbral de puntos mínimos que los habilita para ser evaluados, lo que causa que el modelo no identifique correctamente las suficientes como partes de estos vehículos. Por otra parte, Waymo Open Dataset identifica todos los vehículos con la misma etiqueta, por lo que una excavadora o un camión de basura serán marcados como falsos negativos, repercutiendo en las puntuaciones de AP y AOS.

Como observaciones específicas de los escenarios particulares evaluados, se puede decir que en un escenario de continuos cambios de nivel resultan perjudicialel las dimensiones del

entorno de trabajo seleccionado en cuanto al eje Z se refiere. Durante el análisis del fotograma en el escenario con desnivel se pudieron observar dos casos de falsos negativos que no llegaron a identificarse como verdaderos positivos por estar cortada la nube de puntos que los contiene por el límite vertical de la zona de trabajo definida para el modelo. En segundo lugar, se pudo concluir que las condiciones climáticas de lluvia no afectaron de manera desfavorable a la detección de vehículos. Las principales causas de fallo en los escenarios lluviosos fueron comunes al resto de escenarios. En tercer lugar, el análisis del escenario nocturno de baja iluminación permitió confirmar que la detección basada en LIDAR no se ve afectada negativamente por la oscuridad. En este escenario se consiguieron los mejores resultados de AP y AOS, teniendo valores de 0.6472 y 0.6470 respectivamente. Finalmente, en el escenario de autopista, aunque las etiquetas no estuvieran correctamente definidas en los datos de Waymo, una revisión detallada de los datos permitió observar una gran cantidad de acierto en el modelo con una baja tasa de falsos negativos favorecida por el aumento de distancia entre los vehículos en estos entornos de conducción.

CAPÍTULO 7

CONCLUSIONES Y TRABAJOS FUTUROS

7.1 Conclusiones

El presente trabajo de fin de estudios tenía como objetivo principal el estudio de las arquitecturas de aprendizaje profundo en nubes de puntos procedentes de LIDAR y su aplicación a datos recogidos por otros vehículos autónomos a modo de simulación de la implementación de una red pre entrenada. Se ha realizado el estudio del estado del arte sobre las técnicas de detección en nube de puntos aplicadas a vehículos autónomos. Se ha procedido a la selección de la arquitectura de red neuronal de aprendizaje profundo a implementar con unos criterios de selección basados en las exigencias de un entorno de conducción autónoma. Se ha desarrollado software que ha permitido la simulación de la implementación del modelo de aprendizaje profundo en datos procedentes de otros vehículos inteligentes. Se ha entrenado una red neuronal de aprendizaje profundo basada en la arquitectura PointPillars con la configuración que los ensayos han definido como la más adecuada. Se ha realizado el ajuste de parámetros óptimo para la aplicación de ese modelo en los nuevos datos procedentes de Waymo Open Dataset y se ha evaluado ese mismo modelo en 21 escenarios de conducción autónoma de en torno a 200 fotogramas cada uno con una tasa de refresco de 10 fotogramas por segundo. La revisión de los objetivos del proyecto ha permitido verificar que se ha realizado el cumplimiento de dichos objetivos.

Durante el desarrollo de este proyecto se ha realizado el entrenamiento de una red neuronal de aprendizaje profundo de la arquitectura PointPillars mediante el conjunto de datos Kitti Dataset para la detección de objetos 3D generado en 2017. Tras el entrenamiento de modelos de diferentes parámetros se obtuvieron resultados que indica que la arquitectura, en el conjunto de datos de Kitti Dataset, está capacitada para la detección de automóviles, obteniendo para un parámetro de intersección sobre unión (IoU) de valor 0.5 una precisión media (AP) de valores en torno al 90% en las 3 configuraciones de dificultad marcadas por Kitti. Los resultados en peatones y ciclistas permiten verificar que PointPillars es válida para la detección de estas entidades, sin embargo, los resultados sugieren que una disminución del tamaño de las secciones de pilar empleadas aportaría mejores resultados de detección.

La simulación de la implementación del modelo elegido tras el entrenamiento ha arrojado varias conclusiones importantes. En primer lugar, se ha obtenido una gran bajada de AP, de 0.8885 a 0.5313, lo que pone en peligro la viabilidad de la implementación del modelo evaluado en un vehículo inteligente. Los análisis posteriores han permitido concluir que el modelo no realizaba correctamente las detecciones en el área más cercana al vehículo, lo que se ha achacado a una variación en las alturas de los sensores LIDAR empleados en los conjuntos de datos de entrenamiento e implementación. Esta variación de altura habría impedido durante el entrenamiento el correcto aprendizaje de características superiores de vehículos. Otra causa de fallo significativa se ha encontrado en vehículos que presentaban gran porcentaje de la superficie solapada por otros objetos, lo que ha impedido obtener sus detecciones.

En segundo lugar, se han evaluado escenarios de conducción particulares por separado y se ha llegado a la conclusión de que un modelo PointPillars no se ve afectado por condiciones climáticas de lluvia o por condiciones de baja iluminación como son las condiciones nocturnas. De igual modo, se ha apreciado un funcionamiento óptimo en la detección de automóviles en

autopistas. Por otra parte, la velocidad media de inferencia de 16.99 Hz resulta suficiente para realizar el procesado de todos los fotogramas de los datos de simulación.

Como conclusión, un modelo de aprendizaje profundo pre entrenado basado en PointPillars podría conseguir condiciones de funcionamiento aceptables al implementarse en otros vehículos autónomos. Sin embargo, se debería realizar un entrenamiento con datos más generalistas que abarquen fotogramas captados por vehículos de muestreo de diferentes condiciones para obtener modelos que consigan una mayor generalización.

7.2 Trabajos futuros

Tras el estudio y aplicación de la arquitectura PointPillars al entorno de la conducción autónoma mediante la implementación de modelos pre entrenados, quedan abiertas diferentes vías adicionales de trabajo.

Para empezar, se han evidenciado las necesidades de mejorar los modelos pre entrenados con esta arquitectura para que puedan ser aplicables a otros vehículos inteligentes. Por tanto, se propone una serie de modificaciones para mejorar la generalización de los modelos pre entrenados. En primer lugar, es necesario recurrir a una modificación de los algoritmos para agregar normalización a los valores de intensidad de retorno de LIDAR, de tal modo que se asegure la compatibilidad entre diferentes modelos de estos sensores. En segundo lugar, resulta imprescindible la ampliación del conjunto de datos de entrenamiento con fotogramas proporcionados por vehículos de diferentes alturas para reducir el problema de detección en la zona cercana al vehículo. Finalmente, una ampliación del rango de altura de la zona de detección mejorará el funcionamiento de los modelos en escenarios con cambios de pendiente.

Adicionalmente, puede realizarse el análisis del funcionamiento de los modelos pre entrenados en otras clases de detección como peatones y ciclistas, lo cual resulta imprescindible para la conducción autónoma en entornos urbanos. En este caso deberá realizarse un replanteamiento de determinados parámetros espaciales como tamaño de columna y área de medición y por otra parte estudiar la viabilidad de aplicar un modelo de detección común a todas las clases o implementar diferentes modelos que trabajen en paralelo para cada clase detección.

Finalmente, tras el estudio de la implementación de modelos pre entrenados de PointPillars en otros vehículos, queda abierta la posibilidad de analizar arquitecturas diferentes de detección en nube de puntos o arquitecturas que emplean fusión de datos procedentes de diferentes sensores, como es el caso de Frustum PointNet [20]. Este análisis planteado permitiría averiguar que arquitecturas resultan más generalistas y por tanto cuales son más adecuadas a la hora de implementar modelos de detección pre entrenados en un entorno de conducción autónoma.

CAPÍTULO 8

BIBLIOGRAFÍA

[1] "Machine Learning vs Deep Learning." https://www.neoland.es/blog/machine-learning-vs-deep-learning (accessed Nov. 30, 2020).

[2] M. Jordan, J. Kleinberg, and B. Schölkopf, "Pattern Recognition and Machine Learning."

[3] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015, doi: 10.1038/nature14539.

[4] S. Sharma, S. Sharma, and A. Athaiya, "ACTIVATION FUNCTIONS IN NEURAL NETWORKS," 2020. Accessed: Dec. 02, 2020. [Online]. Available: http://www.ijeast.com.

[5] R. HECHT-NIELSEN, "Theory of the Backpropagation Neural Network**Based on 'nonindent' by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE.," in *Neural Networks for Perception*, Elsevier, 1992, pp. 65–93.

[6] "A Basic Introduction to Convolutional Neural Network | by Himadri Sankar Chatterjee | Medium." https://medium.com/@himadrisankarchatterjee/a-basic-introduction-to-convolutional-neural-network-8e39019b27c4 (accessed Sep. 11, 2020).

[7] N. Highway Traffic Safety Administration and U. Department of Transportation, "TRAFFIC SAFETY FACTS Crash • Stats Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey," 2015.

[8] "J3016A: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International." Sep. 30, 2016, Accessed: Apr. 16, 2021. [Online]. Available: https://www.sae.org/standards/content/j3016_201609/.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," 2016. Accessed: Apr. 21, 2021. [Online]. Available: http://pjreddie.com/yolo/.

[10] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, Apr. 2018, Accessed: Apr. 21, 2021. [Online]. Available: http://arxiv.org/abs/1804.02767.

[11] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 2017. Accessed: Apr. 21, 2021. [Online]. Available: http://pjreddie.com/yolo9000/.

[12] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2015, doi: 10.1007/978-3-319-46448-0_2.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," Feb. 2017. Accessed: Apr. 21, 2021. [Online]. Available: www.aaai.org.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016. Accessed: Apr. 21, 2021. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," 2017. Accessed: Apr. 21, 2021. [Online]. Available: https://github.com/liuzhuang13/DenseNet.

[16] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition.”

[17] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view Convolutional Neural Networks for 3D Shape Recognition.” Accessed: Sep. 02, 2020. [Online]. Available: http://vis-www.cs.umass.edu/mvcnn.

[18] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” 2017.

[19] J. Kossaifi, A. Bulat, G. Tzimiropoulos, and M. Pantic, “T-Net: Parametrizing Fully Convolutional Nets with a Single High-Order Tensor,” Apr. 2019, Accessed: Sep. 07, 2020. [Online]. Available: http://arxiv.org/abs/1904.02698.

[20] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum PointNets for 3D Object Detection from RGB-D Data,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 918–927, Nov. 2017, Accessed: Sep. 22, 2020. [Online]. Available: http://arxiv.org/abs/1711.08488.

[21] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, “PointPillars: Fast Encoders for Object Detection from Point Clouds,” 2019. Accessed: Sep. 30, 2020. [Online]. Available: https://github.com/nutonomy/second.pytorch.

[22] “The KITTI Vision Benchmark Suite.” http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d (accessed Dec. 10, 2020).

[23] “TensorFlow.” https://www.tensorflow.org/ (accessed Mar. 09, 2021).

[24] “PyTorch.” https://pytorch.org/ (accessed Mar. 09, 2021).

[25] “Keras: the Python deep learning API.” https://keras.io/ (accessed Mar. 09, 2021).

[26] “mAP (mean Average Precision) for Object Detection | by Jonathan Hui | Medium.” https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 (accessed Apr. 26, 2021).

[27] “The KITTI Vision Benchmark Suite.” http://www.cvlibs.net/datasets/kitti/ (accessed Apr. 27, 2021).

[28] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, “Multi-View 3D Object Detection Network for Autonomous Driving,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6526–6534, Nov. 2016, Accessed: Sep. 28, 2020. [Online]. Available: http://arxiv.org/abs/1611.07759.

[29] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, “Joint 3D Proposal Generation and Object Detection from View Aggregation,” in *IEEE International Conference on Intelligent Robots and Systems*, Dec. 2018, pp. 5750–5757, doi: 10.1109/IROS.2018.8594049.

[30] M. Liang, B. Yang, S. Wang, and R. Urtasun, “Deep Continuous Fusion for Multi-Sensor 3D Object Detection,” 2018.

[31] Y. Zhou and O. Tuzel, “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,” 2018.

[32] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct. 2018, doi: 10.3390/s18103337.

[33] B. Yang, M. Liang, and R. Urtasun, "HDNET: Exploiting HD Maps for 3D Object Detection," PMLR, Oct. 2018. Accessed: Apr. 27, 2021. [Online]. Available: http://proceedings.mlr.press/v87/yang18b.html.

[34] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks," in *Proceedings - IEEE International Conference on Robotics and Automation*, Jul. 2017, pp. 1355–1361, doi: 10.1109/ICRA.2017.7989161.

[35] B. Li, T. Zhang, and T. Xia, "Vehicle Detection from 3D Lidar Using Fully Convolutional Network," *Robot. Sci. Syst.*, vol. 12, Aug. 2016, Accessed: May 05, 2021. [Online]. Available: http://arxiv.org/abs/1608.07916.

[36] B. Yang, W. Luo, and R. Urtasun, "PIXOR: Real-time 3D Object Detection from Point Clouds," 2018.

[37] 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-Decem, pp. 5100–5109, Jun. 2017, Accessed: Sep. 28, 2020. [Online]. Available: http://arxiv.org/abs/1706.02413.

[38] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361, doi: 10.1109/CVPR.2012.6248074.

[39] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," 2017.

[40] "Open Dataset – Waymo." https://waymo.com/open/ (accessed May 11, 2021).

[41] "nutonomy/second.pytorch: PointPillars for KITTI object detection." https://github.com/nutonomy/second.pytorch (accessed Jun. 22, 2021).

[42] "traveller59/second.pytorch: SECOND for KITTI/NuScenes object detection." https://github.com/traveller59/second.pytorch (accessed Jun. 29, 2021).

[43] "srikanthmalla/waymo_data_utils: waymo open data utils." https://github.com/srikanthmalla/waymo_data_utils (accessed Jun. 29, 2021).