BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD CIENCIAS DE LA COMPUTACIÓN

DETECCIÓN DE OBJETOS USANDO CÁMARAS Y SENSORES LIDAR

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

Maestro en Ciencias de la Computación

PRESENTA:

Fernando Rojas Ramos

ASESOR:

Ivo Humberto Pineda Torres

Ciudad Universitaria, Puebla, 2020

Dedico este trabajo a mi familia, amigos y quienes me brindaron su tiempo y apoyo para hacer esto posible.

Reconocimientos

Expreso mi más sincero y profundo agradecimiento a mis familiares y amigos que me alentaron a realizar mis estudios. En especial a mis padres quienes me dieron toda su confianza y apoyo.

Mi gratitud a la Benemérita Universidad Autónoma de Puebla la cual me he permitido la realización de mis estudios y donde conocí a personas importantes de mi vida.

Al Consejo Nacional de Ciencia y Tecnología por la beca económica que se me otorgó, con la cual, puede iniciar y culminar con grandes satisfacciones mis estudios de maestría.

Declaración de autenticidad

Por la presente declaro que, salvo cuando se haga referencia específica al trabajo de otras personas, el contenido de esta tesis es original y no se ha presentado total o parcialmente para su consideración para cualquier otro título o grado en esta o cualquier otra Universidad. Esta tesis es resultado de mi propio trabajo y no incluye nada que sea el resultado de algún trabajo realizado en colaboración, salvo que se indique específicamente en el texto.

Fernando Rojas Ramos. Ciudad Universitaria, Puebla, 2020

Resumen

En los siguientes años para que los vehículos autónomos puedan operar exitosamente, necesitan estar al tanto de otros vehículos con tiempo suficiente para hacer planes seguros y estables. Dadas las velocidades cercanas entre dos vehículos, esto requiere la capacidad de detectar con precisión los vehículos cercanos y los distantes.

Muchos detectores de objetos actuales basados en imágenes que utilizan redes neuronales convolucionales exhiben un excelente rendimiento en conjuntos de datos existentes como KITTI, este Dataset se utilizara para realizar el estudio con imágenes y datos reales de sensores LIDAR.

La empresa autonomotriz Tesla por ejemplo no usa detección de objetos con sensores LIDAR solo usan cámaras y radares a comparación de muchas otras empresas dedicadas al desarrollo de autos autónomos donde su enfoque esta en usar sensores LIDAR como medio de visión.

En este proyecto de estudio se propone lo siguiente:

- Usar aprendizaje profundo con redes neuronales profundas usando únicamente imágenes para la detección de objetos (clases disponibles en el conjunto de datos).
- Usar aprendizaje profundo con datos de los sensores LIDAR para detectar objetos (clases disponibles en el conjunto de datos).
- Metodología para comparar la eficiencia de detección de ambos dispositivos (Cámaras y LIDAR). De acuerdo a los parámetros de medición que nos brindan las redes neuronales y al procesamiento de los datos.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Como uno de los proyectos de ingeniería más emocionantes del mundo moderno, la conducción autónoma es una aspiración para muchos investigadores e ingenieros de generación en generación. Es un objetivo que podría redefinir fundamentalmente el futuro de la sociedad humana y la vida cotidiana de todos. Una vez que la conducción autónoma se encuentre en plena madurez, seremos testigos de una transformación del transporte público, la infraestructura y la apariencia de nuestras ciudades. El mundo espera explotar la conducción autónoma para reducir los accidentes de tránsito causados por errores de los conductores, para ahorrarles tiempo y liberar a la fuerza laboral, así como para ahorrar espacios de estacionamiento, especialmente en el área urbana.

Los sistemas de detección y percepción se entrenan utilizando grandes conjuntos de datos de datos verificados por humanos. Esto se clasifica como un sistema de aprendizaje supervisado. Los datos de entrenamiento consisten en una salida de los sensores del automóvil (como imágenes de video de la carretera o una nube de puntos LiDAR 3D del entorno alrededor del vehículo). Estos datos se procesan por un humano, que anota la posición y el tipo de cada objeto que desea el vehículo autónomo (AV) aprenda a "ver". Dado que los datos etiquetados son la entrada principal por la cual la red neuronal del auto está entrenada para percibir el mundo, a menudo vemos pistas de qué tan bien funcionará el sistema de percepción de un AV con solo observar la calidad de los datos de entrenamiento que está generando.

Módulo de detección, para garantizar la confiabilidad, la conducción autónoma generalmente requiere múltiples tipos de sensores. Las cámaras, la detección y el alcance por radio (RADAR), la detección y el alcance de la luz (LIDAR) y los sensores ultrasónicos son los más utilizados. Entre esos sensores, LIDAR es particularmente interesante porque proporciona directamente una representación 3D precisa de una escena. Aunque las técnicas para la reconstrucción en 3D y la estimación de profundidad basadas en imágenes en 2D se han mejorado significativamente con el desarrollo de algoritmos de visión por computadora basados en aprendizaje profundo, las estimaciones resultantes aún no son siempre precisas o confiables.

Los avances recientes en las redes neuronales profundas (DNN) han llevado al desarrollo de automóviles autónomos impulsados por DNN que, utilizando cámaras como

sensores, LIDAR, etc., pueden conducir sin intervención humana. Sin embargo, a pesar de su progreso espectacular, los DNN, al igual que el software tradicional, a menudo demuestran comportamientos incorrectos o inesperados de casos de esquinas que pueden conducir a colisiones potencialmente fatales. Varios accidentes del mundo real que involucran autos autónomos ya han ocurrido.

La mayoría de las técnicas de prueba existentes para vehículos con DNN dependen en gran medida de la recopilación manual de datos de prueba en diferentes condiciones de conducción que se vuelven prohibitivamente costosas a medida que aumenta el número de condiciones de prueba.

1.1. Objetivo

Objetivo general:

- Diseñar y entrenar una arquitectura de red neuronal profunda, tipo Red Convolucional (CNN), para la detección de objetos con datos de sensores LIDAR, y con datos obtenidos de las cámaras monoculares de imágenes.

Objetivos específicos

- Elaborar una metodología de evaluación de resultados que permita comparar rendimiento de las CNN, sobre la base de diferentes tipos de datos.

1.2. Estado del arte

Existe una variedad de estudios dirigidos a la autonomía de los autos y en particular a los dispositivos que sirven como medio de visión, en este caso los sensores LIDAR y Cámaras.

Los estudios relacionados a este proyecto están enfocados en estudiar la detección de objetos por separado, es decir los resultados se basan en detección de objetos usando solo cámaras o sensores LIDAR. En este proyecto abarcamos estas dos tendencias para realizar una metodología de comparación de los resultados. Con la finalidad de determinar cual dispositivo nos brinda mejores resultados de detección.

Manuel Herzog y Klaus Dietmayer (15), proponen en su trabajo la detección de objetos con sensores LIDAR en automóviles sin conductor y con la estrategia de entrenar un modelo utilizando datos de diferentes tipos de sensores LIDAR. En comparación a los autores Simon Chadwick, Will Maddern y Paul Newman (2) su estudio consiste en la detección de objetos con cámaras y el calculo de la distancia usando radar este trabajo usa el dataset KITTI , el cual cuenta con diferentes tipos de datos de sensores. También es importante mencionar el uso de Deep learning para la detección de objetos. Las redes neuronales convolucionales profundas (CNN) representan lo último en detección de objetos y se ha demostrado que funcionan bien en una variedad de

escenarios diferentes. Sin embargo, también se ha demostrado que luchan por detectar con precisión objetos pequeños. Jyothish K. James (16) el enfoque de su trabajo son los sensores LIDAR, el desempeño de los sensores LIDAR se ha vuelto significativamente importante para garantizar su fiabilidad y por lo tanto, garantizan la seguridad del vehículo. Naciones Unidas los subestiman con el rendimiento del sensor puede detectar los objetos de manera confiable. Sin embargo, en condiciones climáticas extremas, el rendimiento se ve fuertemente afectado por manipulaciones en la placa frontal del sensor. Se enfocaron en clasificar diferentes tipos de contaminación usando aprendizaje profundo, entrenaron una red neuronal profunda (DNN) siguiendo un concepto de vista múltiple para la generación de datos de entrenamiento y prueba.

Se han realizado experimentos en los que la placa frontal de un sensor LIDAR ha sido contaminado artificialmente con el objetivo de verificar la eficiencia del sensor para realizar su trabajo con los mismos resultados. Di Feng (9) en su artículo intenta resumir sistemáticamente las metodologías y discutir los desafíos para la detección profunda de objetos multimodales y la segmentación semántica en la conducción autónoma. Con este fin, primero proporcionaron una descripción general de los sensores integrados en vehículos de prueba, conjuntos de datos abiertos e información básica para la detección de objetos y la segmentación semántica en la investigación de conducción autónoma. El enfoque de Fusión de sensores es de gran interés y estudio en la autonomía de los autos, debido a que es la integración de todos los datos provenientes de radares, lidar, cámaras, gps, etc. Yan Wang (27) su trabajo tiene como fundamento el uso de cámaras estéreo para la detección de objetos, estas imágenes tienen la ventaja de mostrar más información que las imágenes de cámaras monoculares; usan la profundidad en las escenas de la imágenes para localizar que tan lejano esta el objeto detectado como ocurre con los sensores LIDAR. Por motivos de costo los LIDAR son mas caros que las cámaras estereo. Con estas imágenes logran simular el funcionamiento de los sensores LIDAR. Sin embargo también prueban que la fusión de las imágenes de las cámaras más la nube de puntos de los LIDAR obtienen mejores resultados de detección, el artículo propone como trabajo futuros hacer la comparación de tiempos de procesamiento de los datos de ambos dispositivos de detección.

1.3. Planteamiento del problema

El presente trabajo propone evaluar dos dispositivos de visión cámaras y sensor LIDAR por separado, usando algoritmos de aprendizaje máquina. Se propone implementar y diseñar un modelo de arquitectura de red neuronal tipo CNN para la detección de objetos a partir de datos reales de imágenes de cámaras y datos de sensores LIDAR. Con el objetivo de analizar los resultados de ambos dispositivos y determinar cual de ellos presenta mejor rendimiento en la detección de objetos.

Muchas empresas y centros de investigación dedicadas al desarrollo de la autonomía de los autos están divididos en usar solamente cámaras y radares o sensores LIDAR en fusión con cámaras y radares, para optar al auto de un sistema de visión completo,

con este estudio se brindará una visión general de ambos dispositivos comparando sus resultados de detección usando herramientas de inteligencia artificial.

Detectar e identificar los objetos es la tarea principal de un auto autónomo para poder tomar decisiones correctas para continuar con su trayectoria, su sistema de visión trata de asemejarse a la vista humana capaz de procesar múltiples eventos y procesos en instantes de tiempo (27).

1.4. Metodología

La visión por computadora es un campo multidisciplinario que ha estado ganando una gran cantidad de atracción en los últimos años y los autos autónomos han tomado el centro del escenario. Otra parte integral de la visión por computadora es la detección de objetos.

La detección de objetos ayuda en la estimación de pose, detección de vehículos, vigilancia, etc. La diferencia entre los algoritmos de detección de objetos y los algoritmos de clasificación es que en los algoritmos de detección, tratamos de dibujar un cuadro delimitador alrededor del objeto de interés para ubicarlo dentro de la imagen, mientras que en la clasificación los algoritmos se dedican a ubicar un objetos en una clase.

Para desarrollar el estudio se utilizarán técnicas de aprendizaje profundo, Deep Learning. Se cuenta con un conjunto de datos llamado “KITTI”, el cual posee diferentes tipos de datos, imágenes de cámaras, datos de radar, LIDAR y valores numéricos para el posicionamiento GPS. En la parte de detección de objetos (clases disponibles en el conjunto de datos) con redes neuronales profundas se usarán imágenes y datos de sensores Lidar.

Diseño de la red convolucional y posibles variantes.

El diseño de la red neuronal se tomo como modelo la arquitectura U-net, esta red convolucional tiene características muy importantes debido a las sucesivas convoluciones en su arquitectura; las cuales hacen posible la segmentación, localización y detección de objetos en imágenes con resultados muy eficaces.

A continuación se describe la arquitectura de la red convolucional U-net:

La arquitectura de la red neuronal profunda consiste en una ruta de contracción y una ruta expansiva. La ruta de contracción sigue la arquitectura típica de una red convolucional. Consiste en la aplicación repetida de dos convolucionales de 3x3(convolucionales sin relleno), cada una seguida de una función de activación RELU y una operación max pooling de 2x2 con stride de 2 para la parte de disminución de resolución(downsampling).

En la parte de la ruta expansiva consiste en un muestreo ascendente del mapa de características seguido de una convolución de 2x2(Up convolution) que reduce a la mitad el número de canales de características, una concatenación con el mapa de características recortado correspondientemente de la ruta de contracción y dos convolucionales de 3x3, cada una seguida por una función de activación RELU. El recorte es necesario debido a la pérdida de pixeles del borde en cada convolución. En la capa final, se utiliza

una convolución 1x1 para asignar cada vector de características de 64 componentes al número deseado de clases. En total, la red tiene 23 capas convolucionales.

Pre-procesamiento de la información

Se analizarán los posibles modelos existentes de redes neuronales profundas que han trabajado con datos de sensores LIDAR como referencia para poder replicar los modelos con sus parámetros previamente estudiados e implementados con éxito, la misma metodología con la red neuronal profunda con imágenes.

Los primeros pasos para la dar solución a este proyecto consisten en manipular y procesar los datos de los sensores LIDAR. El objeto del preprocesamiento es para tener una primera versión de los objetos que están en la escena. Los objetos detectados por lo general representan una nube de puntos las cuales serán delimitadas por una rectángulo conocido en la literatura como bounding box.

Se repite el proceso, utilizando la misma red profunda que se emplea con los datos LIDAR, con el objetivo de estimar coincidencias en la detección de objetos. Cada objeto analizado es igualmente delimitado por el rectángulo.

Metodología

Finalmente se definirá una metodología de evaluación la que permitirá estimar que sensor es preferible para la de detección de objetos. El objetivo es concluir si los sensores LIDAR son necesarios en los autos autónomos como sistema de visión, o los resultados de las cámaras son mejores y suficientes.

Una forma de evaluar los resultados será mediante la utilización de dos redes neuronales convolucionales profundas arquitecturas para el entrenamiento y prueba, existen condiciones extremas como lluvia, granizo, nieve, polvo,luz,etc, donde se ha reportado que los sensores LIDAR trabajan mejor que las cámaras y viceversa, lo que conlleva a que los conjuntos de datos sean diferentes y se requiera de la realización de varias pruebas. Motivo por el cual se considera de la utilización de la infraestructura de super cómputo que brinda la BUAP. Se adoptarán criterios de evaluación similares a los empleados en el artículo de Yan Wang (27).

1.5. Contribuciones

La principal contribución de este trabajo es un estudio en el campo de la inteligencia artificial en especial en Deep Learning con la detección de objetos con dispositivos que tienen como función la visión de los autos autónomos(Cámaras y sensores LIDAR).

Conocer y aprender el estado del arte de el alcance de los dispositivos de visión de las próximas generaciones de los autos autónomos es de suma importancia para evitar lo que se llaman Recalls a causa de las fallas que se detectan durante la liberación de los primeros autos con autonomía nivel 5. Lo cual significaría volver hacer pruebas e invertir en re trabajos que para la industria automovilística son inversiones millonarias.

El estudio esta enfocado en el desarrollo y diseño de algoritmos de aprendizaje máquina usando los datos de los dispositvos de visión de los autos autónomos (LIDAR y Cámara), con el objetivo de alcanzar la autonomía total en los próximos años.

Muchas empresas y centros de desarrollo tecnológico están trabajando en el área de la autonomía de los autos el cual abre la oportunidad de nuevos empleos y sera necesario atender los requerimientos en conocimientos para enfrentar las posibles fallas en un futuro.

1.6. Estructura de la tesis

En el presente proyecto de tesis, se tratan diversos temas de visión por computadora y de aprendizaje profundo, además se realizan distintas pruebas experimentales de los objetivos planteados y se obtienen resultados aceptables. Por lo cual, se plantea la siguiente estructura del proyecto de tesis:

- En el segundo capítulo, se abordan los fundamentos y la teoría que se utilizaron para el desarrollo del proyecto. Se expone la teoría relacionada con el aprendizaje profundo (en inglés, deep learning). Se da una explicación sobre las redes neuronales profundas, los tipos de capas neuronales que existen y los parámetros para su entrenamiento. Teoría general de los sensores LIDAR y Cámaras. Se da una breve introducción a los algoritmos de procesamiento de imágenes, segmentación y detección de objetos.
- En el tercer capítulo, se expone de manera detallada el procedimiento para desarrollar los objetivos del Proyecto: Procesamiento de la nube de puntos de los sensores LIDAR. Procesamiento de las imágenes de las cámaras e imagenes voxel de los sensores Lidar. Diseño de Red neuronal convolucional profunda entrenada con datos de los sensores LIDAR y con imágenes de las cámaras, por último la descripción de los parámetros de entrenamiento de las redes neuronales convolucionales.
- En el cuarto capítulo, Metodología de evaluación de resultados que permite comparar rendimiento de las redes neuronales convolucionales, sobre la base de diferentes tipos de datos, se abordan las pruebas de las CNN, comparando y analizando los resultados de la segmentación y detección de objetos.
- En el quinto capítulo, se exponen las conclusiones del proyecto de tesis y se plantea el trabajo a futuro.

Capítulo 2

Marco teórico

En este capítulo, se presentan los principales fundamentos teóricos sobre los cuales está planteado el presente proyecto de tesis. Se parte con la teoría general de redes neuronales convolucionales y sus principales operaciones para poder explicar el diseño de la red neuronal U-Net. Se da un panorama general del sensor LIDAR y los algoritmos que se usan para el procesamiento de la nube de puntos. Se presenta teoría general de el procesamiento de imágenes y los principales algoritmos que nos ayudan a mejorar la calidad y resolución. Por último se describe la red neuronal convolucional profunda U-Net, la cual es el modelo que se entrenara con datos de los sensores LIDAR e imágenes de las cámaras.

2.1. Introducción a las redes neuronales

Primero, tenemos que hablar sobre las neuronas, la unidad básica de una red neuronal (25). Una neurona toma entradas, hace algunos cálculos con ellas y produce una salida. Así es como se ve una neurona de 2 entradas Figura 2.1:

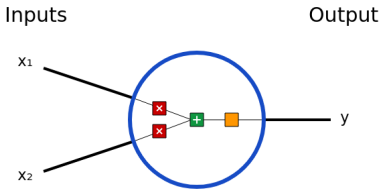


Figura 2.1: Neurona.

3 cosas están sucediendo aquí. Primero, cada entrada se multiplica por un peso:

$$\begin{aligned} x_1 &\rightarrow x_1 * w_1 \\ x_2 &\rightarrow x_2 * w_2 \end{aligned}$$

A continuación, se agregan todas las entradas ponderadas junto con un bias b:

$$(x_1 * w_1) + (x_2 * w_2) + b$$

Finalmente, la suma se pasa a través de una función de activación:

$$y = f\left(x_1 * w_1 + x_2 * w_2 + b\right)$$

La función de activación se utiliza para convertir una entrada sin límites en una salida que tiene una forma agradable y predecible. Una función de activación comúnmente utilizada es la función sigmoide Figura 2.2:

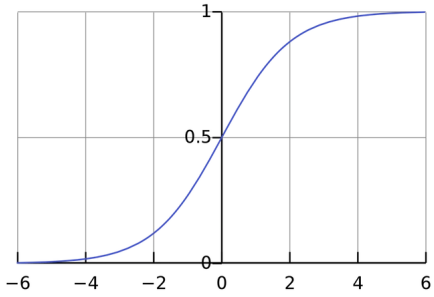


Figura 2.2: Función Sigmoid.

La función sigmoide (30) solo genera números en el rango $(0, 1)$. Puedes considerarlo como una compresión $(-\infty, +\infty)$ a $(0, 1)$: los grandes números negativos se vuelven (~ 0), y los grandes números positivos se vuelven (~ 1).

Como hemos visto, las funciones de activación son una parte vital del éxito de las redes neuronales convolucionales. Por lo tanto, dada su importancia, procedemos a examinar algunas de las funciones más comunes utilizadas como activaciones 2.3.

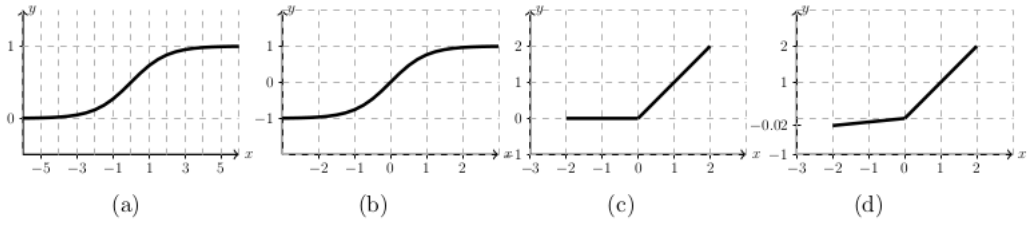


Figura 2.3: Funciones de activación comunes. La figura (a) ilustra la función sigmoidea, la figura (b) ilustra la función tangente hiperbólica, la figura (c) la unidad lineal rectificada y (d) la unidad lineal rectificada con fugas.

2.1.1. Combinando neuronas en una red neuronal

Una red neuronal no es más que un grupo de neuronas conectadas entre sí. Así es como se vería una red neuronal simple Figura 2.4:

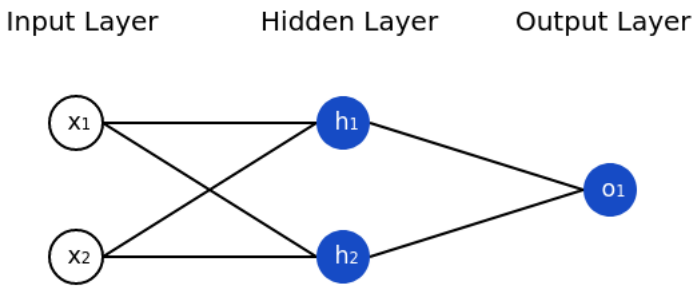


Figura 2.4: Red neuronal.

Esta red tiene 2 entradas, una capa oculta con 2 neuronas (h_1 *y* h_2) y una capa de salida con 1 neurona (o_1) observe que las entradas para o_1 son las salidas de h_1 y h_2 eso es lo que hace de esto una red. Una capa oculta es cualquier capa entre la capa de entrada (primera) y la capa de salida (última). Puede haber múltiples capas ocultas.

2.1.2. Función de pérdida

Antes de entrenar nuestra red, primero necesitamos una forma de cuantificar qué tan bien lo está haciendo para que pueda intentar mejorar. Esa es la función de pérdida. Utilizaremos la función de pérdida de error cuadrático medio (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{true} - y_{pred})^2 \tag{2.1}$$

Analicemos esto:

n: es el número de muestras.

y: representa la variable que se predice.

y_{true}: es el valor verdadero de la variable (la respuesta correcta).

y_{pred}: es el valor predicho de la variable. Es lo que sea que produzca nuestra red.

$(y_{true} - y_{pred})^2$: se conoce como el error al cuadrado. Nuestra función de pérdida es simplemente tomar el promedio sobre todos los errores al cuadrado (de ahí el nombre error cuadrático medio). ¡Cuanto mejores sean nuestras predicciones, menor será nuestra pérdida! Mejores predicciones = menor pérdida.

2.2. Redes neuronales convolucionales

Las redes neuronales convolucionales (ConvNets o CNN) (28) son una categoría de redes neuronales que han demostrado ser muy efectivas en áreas como el reconocimiento y la clasificación de imágenes. ConvNets ha tenido éxito en la identificación de caras, objetos y señales de tráfico, además de potenciar la visión en robots y automóviles sin conductor.

Básicamente son redes neuronales que usan capas Convolucionales, también conocidas como capas Conv, que se basan en la operación matemática de la convolución. Las capas de convolucionales consisten en un conjunto de filtros, que puedes considerarse matrices 2D de números.

Podemos usar una imagen de entrada y un filtro para producir una imagen de salida haciendo girar el filtro con la imagen de entrada. Esto consiste en:

1.- Superposición del filtro en la parte superior de la imagen en algún lugar. 2.- Realizando la multiplicación por elementos entre los valores en el filtro y sus valores correspondientes en la imagen. 3.- Resumiendo todos los productos basados en elementos. Esta suma es el valor de salida para el píxel de destino en la imagen de salida. 4.- Repetir para todas las ubicaciones.

Esa descripción de 4 pasos fue un poco abstracta, así que hagamos un ejemplo. Se considera esta pequeña imagen en escala de grises de 4x4 y un filtro de 3x3 como se muestra en la Figura 2.5:

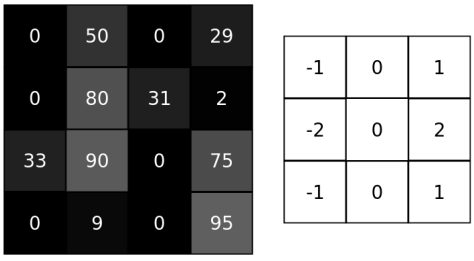


Figura 2.5: Imagen de 4x4 (izquierda) y un filtro de 3x3a (derecha)

Los números en la imagen representan intensidades de píxeles, donde 0 es negro y 255 es blanco. Envolveremos la imagen de entrada y el filtro para producir una imagen de salida de 2x2.

A continuación, realizamos una multiplicación por elementos entre los valores de imagen superpuestos y los valores de filtro. Aquí están los resultados, comenzando desde la esquina superior izquierda, yendo hacia la derecha, luego hacia abajo 2.1.

Valor imagen	Valor filtro	Resultado
0	-1	0
50	0	0
0	1	0
0	-2	0
80	0	0
31	2	62
33	-1	-33
90	0	0
0	1	0

Tabla 2.1: Ilustración del producto punto de la operación convolución

A continuación, resumimos todos los resultados $62 - 33 = 29$.

Finalmente, colocamos nuestro resultado en el píxel de destino de nuestra imagen de salida. Dado que nuestro filtro se superpone en la esquina superior izquierda de la imagen de entrada, nuestro píxel de destino es el píxel superior izquierdo de la imagen de salida 2.6:

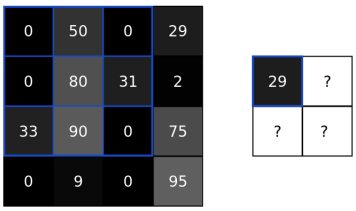


Figura 2.6: Producto punto de la imagen con el Filtro

Hacemos lo mismo para generar el resto de la imagen de salida 2.7

2.2.1. Relleno

Muchas veces, preferimos que la imagen de salida sea del mismo tamaño que la imagen de entrada. Para hacer esto, agregamos ceros alrededor de la imagen para que

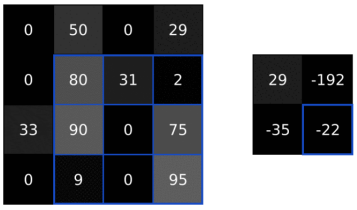


Figura 2.7: Resultado final del producto punto de la imagen con el Filtro

podamos superponer el filtro en más lugares. Un filtro 3x3 requiere 1 píxel de relleno 2.8

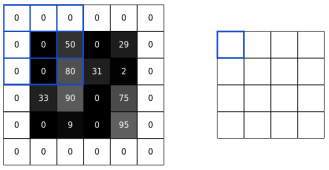


Figura 2.8: Una entrada 4x4 convolucionada con un filtro 3x3 para producir una salida 4x4 usando el mismo relleno

Esto se llama "mismo relleno", ya que la entrada y la salida tienen las mismas dimensiones. No usar ningún relleno, que es lo que hemos estado haciendo y continuaremos haciendo para esta publicación, a veces se denomina relleno valido".

2.2.2. Capas de convolución

Como se mencionó anteriormente, las CNN incluyen capas conv que usan un conjunto de filtros para convertir las imágenes de entrada en imágenes de salida. El parámetro principal de una capa de convolución es la cantidad de filtros que tiene. Por ejemplo para una CNN, usaremos una pequeña capa de convolución con 8 filtros como capa inicial en nuestra red. Esto significa que convertirá la imagen de entrada de 28x28 en un volumen de salida de 26x26x8, la salida es 26x26x8 y no 28x28x8 porque estamos usando un relleno valido, lo que disminuye el ancho y la altura de la entrada en 2 2.9

Cada uno de los 8 filtros en la capa conv produce una salida de $26x26$, por lo que apilados juntos forman un volumen de $26x26x8$. Todo esto sucede debido a 3x3 (tamaño del filtro) x 8 (número de filtros) = solo 72 pesos

2.2.3. Capa de agrupación

Los píxeles vecinos en las imágenes tienden a tener valores similares, por lo que las capas convolucionales generalmente también producirán valores similares para los

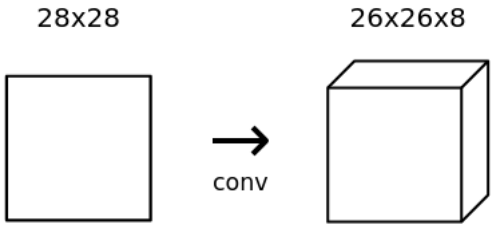


Figura 2.9: Conv Layers

píxeles vecinos en las salidas. Como resultado, gran parte de la información contenida en la salida de una capa convolucional es redundante. Por ejemplo, si usamos un filtro de detección de bordes y encontramos un borde fuerte en una ubicación determinada, es probable que también encontremos bordes relativamente fuertes en ubicaciones con 1 píxel desplazado del original. Sin embargo, estos son todos del mismo borde. No estamos encontrando nada nuevo.

Capas de agrupación (pooling) resuelven este problema. Todo lo que hacen es reducir el tamaño de la entrada, agrupando por valores. La agrupación generalmente se realiza mediante una operación simple como valor máximo, mínimo o el promedio. Aquí hay un ejemplo de una capa de agrupación máxima con un tamaño de agrupación de 2 2.10

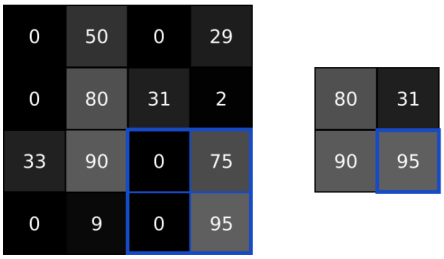


Figura 2.10: Agrupación máximo (tamaño de agrupación 2) en una imagen 4x4 para producir una salida 2x2

Para realizar agrupación máximo, atravesamos la imagen de entrada en bloques de 2x2 (porque el tamaño de la agrupación = 2) y colocamos el valor máximo en la imagen de salida en el píxel correspondiente.

La agrupación divide el ancho y la altura de la entrada por el tamaño de la agrupación. Para nuestro ejemplo, colocaremos una capa de agrupación máximo con un tamaño de 2 justo después de nuestra capa de conversión inicial. La capa de agrupación transformará una entrada de 26x26x8 en una salida de 13x13x8, como se muestra en la Figura 2.11

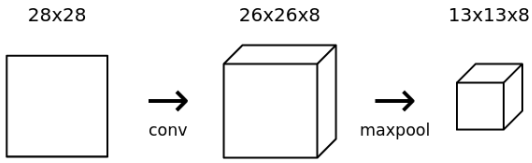


Figura 2.11: Convolución + Agrupación

2.2.4. Softmax

Softmax convierte valores reales arbitrarios en probabilidades, que a menudo son útiles en aprendizaje maquina. Escrito de manera elegante, Softmax realiza la siguiente transformación en n números (19).

$$s\left(x_i\right) = \frac{e^{x^i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.2}$$

Las salidas de la transformada Softmax siempre están en el rango [0,1] y suman hasta 1. Por lo tanto, forman una distribución de probabilidad. Se muestra el diagrama de bloques con la capa softmax en la siguiente Figura 2.12

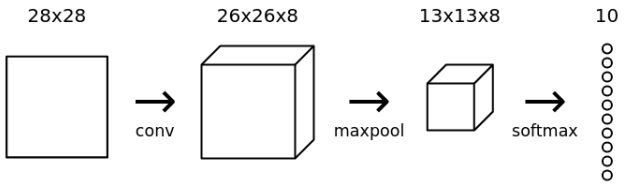


Figura 2.12: Convolución + Agrupación + Softmax

2.2.5. Propagación hacia adelante

La propagación hacia adelante (forward propagation), 2.13 es un proceso de alimentar valores de entrada a la red neuronal y obtener una salida que llamamos valor predicho. A veces nos referimos a la propagación hacia adelante como inferencia. Cuando alimentamos los valores de entrada a la primera capa de la red neuronal, no se realiza ninguna operación. La segunda capa toma valores de la primera capa y aplica operaciones de multiplicación, suma y activación y pasa este valor a la siguiente capa. El mismo proceso se repite para las capas posteriores y finalmente obtenemos un valor de salida de la última capa.

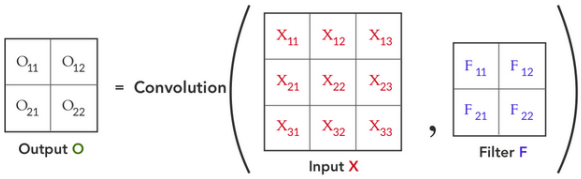


Figura 2.13: Propagación hacia adelante, con la convolución entre X y F nos da la salida O

2.2.6. Propagación hacia atrás

Propagación hacia atrás (Back-Propagation), 2.14 después de la propagación hacia adelante, obtenemos un valor de salida que es el valor predicho. Para calcular el error, comparamos el valor predicho con el valor de salida real. Utilizamos una función de pérdida (mencionada a continuación) para calcular el valor del error. Luego calculamos la derivada del valor de error con respecto a cada peso en la red neuronal. La propagación hacia atrás utiliza la regla de la cadena del cálculo diferencial. En la regla de la cadena, primero calculamos las derivadas del valor de error con respecto a los valores de peso de la última capa. Llamamos a estos derivados, gradientes y usamos estos valores de gradiente para calcular los gradientes de la segunda última capa. Repetimos este proceso hasta que obtengamos gradientes para cada peso en nuestra red neuronal. Luego restamos este valor de gradiente del valor de peso para reducir el valor de error. De esta manera nos acercamos (descenso) a los mínimos locales (significa pérdida mínima).

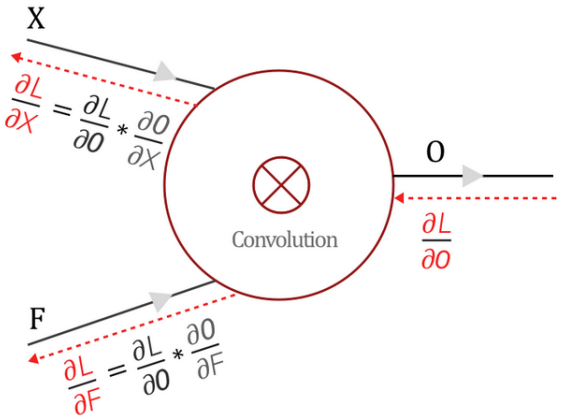


Figura 2.14: Regla de la cadena y Propagación hacia atrás

2.2.7. Optimizadores

Optimizadores del modelo: el optimizador es una técnica de búsqueda que se utiliza para actualizar los pesos en el modelo.

- SGD: Descenso de gradiente estocástico, con soporte para el impulso.
- RMSprop: método de optimización de la tasa de aprendizaje adaptativo propuesto por Geoff Hinton.
- Adam: Estimación adaptativa del momento (Adam) que también utiliza tasas de aprendizaje adaptativo.

2.3. Métricas de evaluación

- True Positive (TP): el número de píxeles clasificados correctamente que pertenecen a la clase X.
- False Positive (FP): el número de píxeles que no pertenecen a la clase X en ground truth pero el algoritmo los clasifica que pertenecen a la clase.
- False negative (FN): el número de píxeles que pertenecen a la clase X en la ground truth, pero el algoritmo no los clasifica como pertenecientes a la clase.

$$IOU_{class} = \frac{TP}{TP + TF + FN} \tag{2.3}$$

La segmentación semántica (19) consiste en proporcionar una etiqueta clase para cada píxel en una imagen 2D, se pueden evaluar los modelos de segmentación semántica usando la clase IOU. Se muestra en la Figura 2.15 un ejemplo de como se evalua la clase Road de acuerdo a la formula anterior.

2.4. Autoencoder

Los autoencoders son redes neuronales con el objetivo de generar nuevos datos primero comprimiendo la entrada en un espacio de variables latentes y luego reconstruyendo la salida en base a la información adquirida. Este tipo de red consta de dos partes, Se muestra en la Figura 2.16

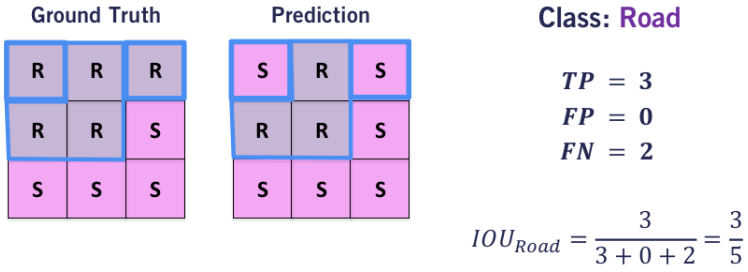


Figura 2.15: Métrica de evaluación IOU

El principio básico para todos los métodos implica recibir una entrada x y asignarla a una representación latente h, utilizando una función $h = \sigma(Wx + b)$, donde σ es una función de activación no lineal, W es una matriz de pesos entre las dos capas, y b es sesgo(bias). El autoencoder intenta reconstruir la entrada original por $y = \sigma\left(W'h + b'\right)$. Por lo tanto, cada muestra de entrenamiento x_i se asigna primero a una capa oculta h_i y luego se reconstruye a y_i. El autoencoder (4) se entrena utilizando la retropropagación para reducir este error de reconstrucción.

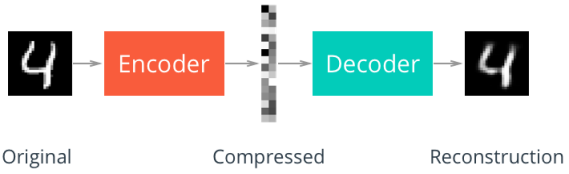


Figura 2.16: Arquitectura de un autoencoder

Encoder: la parte de la red que comprime la entrada en un espacio de variables latentes y que puede representarse mediante la función de codificación h = f (x). Decoder: la parte que trata de reconstruir la entrada en base a la información recolectada previamente. Se representa mediante la función de decodificación r = g (h).

2.4.1. Autoencoder Convolucional

Para usar CNN como autoencoders (4), para cada capa convolucional, se debe construir una capa deconvolucional correspondiente. Además, las capas de agrupación máxima provocan la pérdida de información, por lo que una capa de agrupación debería intentar restaurar aproximadamente los valores originales. Tenga en cuenta que el submuestreo debido a la agrupación máxima de hecho funciona como un fuerte regularizador.

Las capas de desconvolución pueden ser iguales pero transpuestas a las capas de convolución originales, o aprendidas desde cero. A menudo, ambos enfoques funcionan

igualmente bien en la práctica. Esto es similar a los codificadores automáticos estándar donde los pesos de la capa de decodificador $W^{'}$ pueden aprenderse desde cero o ajustarse a la transposición del codificador capa $W^{'} = W^{T}$, esto se conoce como pesos atados.

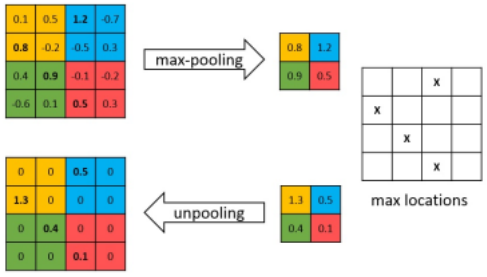


Figura 2.17: Agrupando y desagrupando capas. Para cada capa de agrupación, se almacenan las ubicaciones máximas. Estas ubicaciones se utilizan luego en la capa de desagrupación.

2.5. LIDAR

LIDAR (Light Detection and Ranging) (10) es un método utilizado para generar representaciones 3D precisas de los alrededores, y utiliza luz láser para lograrlo. Básicamente, el objetivo 3D se ilumina con una luz láser (un haz de luz dirigido y enfocado) y la luz reflejada es recolectada por sensores. Se calcula el tiempo requerido para que la luz se refleje nuevamente en el sensor, se muestra en la Figura 2.18.

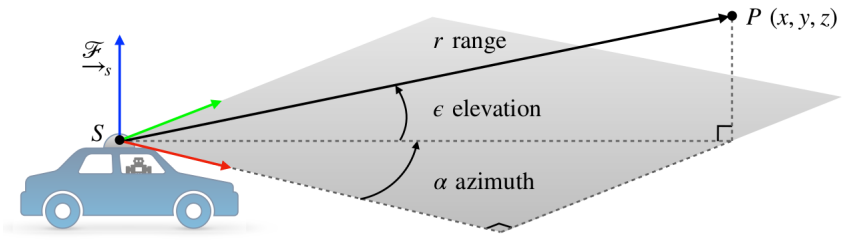


Figura 2.18: Los sensores 3D LIDAR informan el alcance, el ángulo de azimuth y el ángulo de elevación (+ intensidad de retorno).

Diferentes sensores recogen luz de diferentes partes del objeto, y los tiempos registrados por los sensores serían diferentes. Esta diferencia de tiempo calculada por los

sensores se puede usar para calcular la profundidad del objeto. Esta información de profundidad combinada con la representación 2D de la imagen proporciona una representación 3D precisa del objeto. Este proceso es similar a la visión humana real. Dos ojos hacen observaciones en 2D y estas dos piezas de información se combinan para formar un mapa 3D (percepción de profundidad). Así es como los humanos perciben el mundo que nos rodea.

2.5.1. El algoritmo de punto más cercano iterativo (ICP)

ICP es realmente sencillo matemáticamente. El paso de asociación es bastante simple. Puede ser simplemente una búsqueda de fuerza bruta para los pares de puntos más cercanos entre los escaneos fuente y objetivo. Sin embargo, si está trabajando con escaneos grandes, es una buena idea usar árboles KD para este paso (23).

- ICP es una forma de determinar el movimiento de un automóvil sin conductor alineando nubes de puntos de LIDAR u otros sensores.
- ICP minimiza iterativamente la distancia entre puntos en cada punto nube.
- ICP es sensible a los valores atípicos causados por objetos en movimiento, que pueden ser parcialmente mitigado mediante funciones de pérdida robustas.

Punto a punto ICP minimiza la distancia euclidiana entre puntos $Ps^{'}$ y el punto más cercano Ps 2.19.

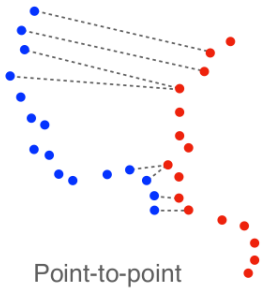


Figura 2.19: Algoritmo de punto más cercano iterativo

Punto a plano ICP minimiza la distancia perpendicular entre punto $Ps^{'}$ y el plano mas cercano Ps, este método trabaja muy bien en ambientes como las ciudades 2.20.

2.5.2. 3D voxelization

Para disfrutar del éxito del procesamiento de imágenes 2D y la visión por computadora, podemos discretizar el espacio 3D en vóxeles y usar una serie de vóxeles para

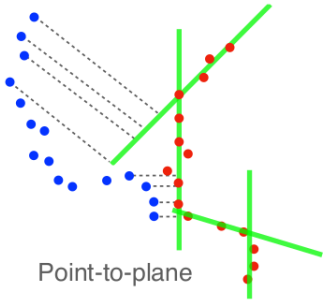


Figura 2.20: Algoritmo de punto más cercano iterativo

representar una nube de puntos 3D. Una discretización directa es dividir el espacio 3D en vóxeles no solapados equidistantes de cada una de las tres dimensiones ver Figura 2.21; Permite un espacio 3D con rango H, W, D a lo largo de los ejes X, Y, Z respectivamente. Cada vóxel es de tamaño h, w, d, respectivamente. El (i, j, k) th vóxel representa un espacio vóxel 3D, $V_{i,j,k} = \{(x, y, z) \mid (i-1)\, h \leq x \langle\, ih, (j-1)\, w \leqslant y < jw, (k-1)\, d \leq z < kd\}$, entonces usamos un tensor de tres modos para representar esta nube de puntos 3D (1) . Tenemos $X_{vox}^{i,j,k} \in \Re^{HXWXD}$,cuyo elemento (i, j, k) es:

$$X_{i,j,k}^{vox} = \begin{Bmatrix} 1, \ cuando\ V_{i,j,k} \cap S \neq \emptyset \\ 0, \ en\ otro\ caso \end{Bmatrix} \tag{2.4}$$

2.5.3. Range view.

Un barrido LIDAR en tiempo real es esencialmente una serie de mediciones de rango desde una sola ubicación con cierto campo de visión angular; ver Figura 2.21. Podemos organizar aproximadamente los puntos 3D en LIDAR en tiempo real a una imagen de vista de rango 2D. Cada píxel en la imagen de vista de rango corresponde a un tronco en el espacio 3D.

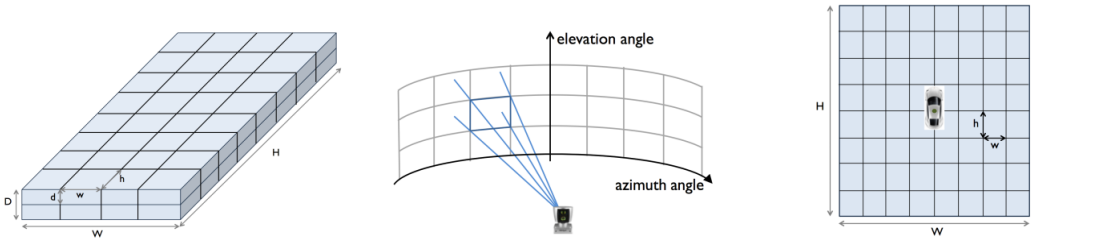
El valor de píxel es el rango de el LIDAR hasta el punto 3D más cercano dentro del tronco. Específicamente, dividimos el espacio 3D a lo largo del ángulo de azimuth $\alpha \in [0, 2\pi)$ y el ángulo de elevación $\theta \in (\pi/2, \pi/2]$ con la resolución del ángulo de azimuth α_0 y la resolución del ángulo de elevación θ_0. El (i, j)th pixel corresponde a un tronco del espacio.
$V_{i,j} = \left\{(x, y, z) \mid \alpha_0\,(i-1) \leq acos\left(\frac{x}{\sqrt{x^2+y^2}}\right) < \alpha_0 i \theta_0\,(j-1) \leq atan\left(\frac{z}{\sqrt{x^2+y^2}}\right) + \frac{\pi}{2} < \theta_0 j\right\}$.
Entonces usamos una matriz 2D para representar una nube de puntos 3D. Tenemos $X^{FV} \in \Re^{HXW}$, cuyo (i, j) th elemento es:

$$X_{FV}^{i,j} = \begin{Bmatrix} min_{x,y,z} \in V_{i,j} \cap S\sqrt{x^2+y^2+z^2}, \ V_{i,j,k} \cap S = \emptyset \\ -1, \ en\ otro\ caso \end{Bmatrix} \tag{2.5}$$

2.5.4. Bird's-eye view.

La representación basada en la vista de pájaro (BEV) es un caso especial de voxelización 3D al ignorar la dimensión de altura. Proyecta vóxeles 3D en una imagen BEV (3); ver Figura 2.21. Permite un espacio 3D con rango H, W a lo largo de los ejes X, Y, respectivamente. Cada píxel es de tamaño h, w respectivamente. El (i, j) th pixel en la imagen BEV representa un espacio de pilar, $V_{i,j} = \{(x, y, z) \mid (i-1)\, h \leq x < ih, (j-1)\, w \leq y < jw\}$. Luego usamos una matriz 2D para representar una nube de puntos 3D. Permite $X^{BEV} \in \Re^{HXW}$ cuyo (i, j) th elemento es:

$$X_{i,j}^{BEV} = \begin{Bmatrix} 1, & cuando\ V_{i,j} \cap S \neq \emptyset \\ 0, & en\ otro\ caso \end{Bmatrix} \tag{2.6}$$



(a) 3D voxel-based representation. (b) Range-view-based representation. (c) Bird's-eye-view-based representation.

Figura 2.21: Enfoques comunes para discretizar el espacio 3D. La representación basada en voxel 3D es para discretizar el espacio 3D en voxels no superpuestos igualmente espaciados de cada una de las tres dimensiones; la representación basada en la vista de rango es discretizar el espacio 3D a lo largo del ángulo de azimuth y el ángulo de elevación; y la representación basada en la vista de pájaro es discretizar el espacio 3D a lo largo de los ejes X, Y, omitiendo la dimensión de altura.

2.6. Segmentación de nube de puntos 3D

El objetivo de la segmentación es clasificar cada punto 3D en una nube de puntos 3D en una categoría predefinida; vea la Figura 2.22 . Como una tarea típica del aprendizaje en la nube de puntos 3D, la segmentación es crítica en el módulo de percepción de la conducción autónoma, donde queremos identificar la clase de cada punto LIDAR, por ejemplo, un vehículo, un árbol o una carretera etc (9).

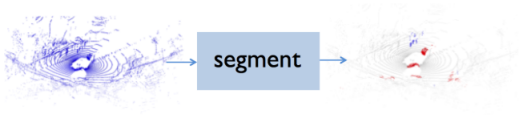


Figura 2.22: La segmentación de la nube de puntos 3D tiene como objetivo clasificar cada punto 3D en una nube de puntos 3D en una categoría predefinida.

2.7. Visión por computadora

La distinción entre el procesamiento de imágenes y la visión por computadora es difícil y subjetiva. Reservamos el nombre de visión por computadora para aquellas partes de los campos donde interpretamos la información visual en términos de las entidades en el mundo 3D que se representan en la imagen.

Comenzamos esta parte con la descripción de la cámara tipo pinhole que es el modelo para la mayoría de nuestros dispositivos de imágenes. La cámara pinhole modela la cámara de fotos estándar, modela el ojo, la cámara de video, etc. Resultará ser un modelo sorprendentemente simple (solo una multiplicación de matriz).

En los gráficos de computadora, el programador (usuario) establece la matriz de la cámara con mayor frecuencia al representar una escena virtual en la pantalla de la computadora. Por otro lado, en visión artificial, la cámara es real y sus parámetros (geométricos y ópticos) y su posición y orientación en el espacio deben aprenderse de las mediciones. La estimación de estos parámetros desconocidos se llama calibración de la cámara y no es una tarea trivial hacer de manera correcta y precisa.

2.7.1. Cámara Pinhole

Cuando miramos a nuestro alrededor, el mundo 3D se proyecta ópticamente sobre la retina (capa fotosensible) en nuestro ojo. La luminancia (energía visual) se mide en muchas células fotosensibles (bastones y conos) en la retina. La colección de todas estas mediciones constituye una imagen de lo que vemos de la misma manera que una colección de píxeles en la pantalla forma una imagen (5). El principio óptico del ojo humano es el mismo que para cualquier cámara óptica, ya sea una cámara fotográfica o una videocámara. El modelo más simple (pero sorprendentemente preciso) para una cámara óptica de este tipo es la cámara Pinhole Figura 2.23.

2.7.2. Distorsión

La distorsión de la imagen ocurre cuando una cámara mira objetos 3D en el mundo real y los transforma en una imagen 2D; Esta transformación no es perfecta. La distorsión en realidad cambia la forma y el tamaño de estos objetos 3D. Por lo tanto, el

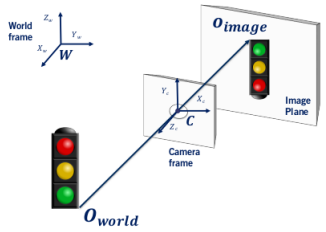


Figura 2.23: Proyección: Mundo a imagen(cámara real)

primer paso para analizar las imágenes de la cámara es deshacer esta distorsión para que pueda obtener información correcta y útil de ellas.

La distorsión puede cambiar el tamaño aparente de un objeto en una imagen, puede cambiar la forma aparente de un objeto en una imagen, puede hacer que cambie la apariencia de un objeto dependiendo de dónde se encuentre en el campo de visión y puede hacer que los objetos parezcan más cercanos o más alejados lejos de lo que realmente son Figura 2.24.

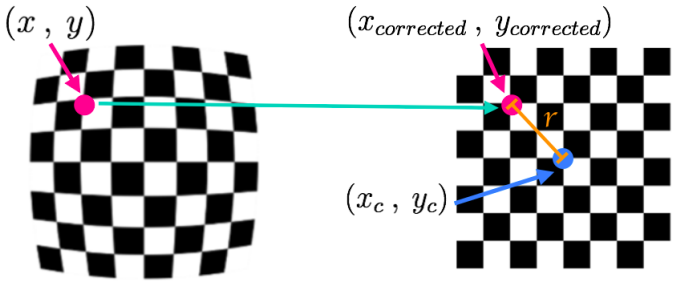


Figura 2.24: Calibración de las cámaras

2.7.3. Tipos de distorsión

Las cámaras reales usan lentes curvadas para formar una imagen, y los rayos de luz a menudo se doblan demasiado o muy poco en los bordes de estos lentes. Esto crea un efecto que distorsiona los bordes de las imágenes, de modo que las líneas u objetos aparecen más o menos curvados de lo que realmente son. Esto se llama distorsión radial Figura 2.25, y es el tipo más común de distorsión (5).

Otro tipo de distorsión, es la distorsión tangencial. Esto ocurre cuando la lente de una cámara no está alineada perfectamente paralela al plano de imagen, donde está la película o el sensor de la cámara. Esto hace que una imagen se vea inclinada para que algunos objetos aparezcan más lejos o más cerca de lo que realmente están.

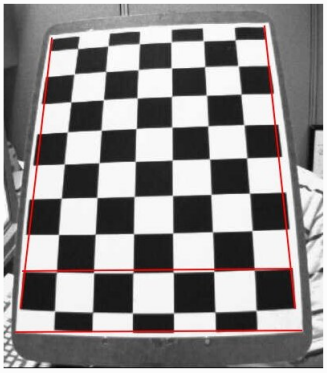


Figura 2.25: Distorsión radial, las líneas rectas aparecerán curvas. Su efecto es mayor a medida que nos alejamos del centro de la imagen, los bordes del tablero de ajedrez están marcados con líneas rojas; Pero puede ver que el borde no es una línea recta y no coincide con la línea roja, todas las líneas rectas esperadas están abultadas.

Corrección de distorsión radial:

$$x_{distorsionada} = x_{ideal}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{2.7}$$

$$y_{distorsionada} = y_{ideal}\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \tag{2.8}$$

Corrección de distorsión tangencial: Hay dos coeficientes más que explican la distorsión tangencial: p1 y p2, y esta distorsión se puede corregir usando una fórmula de corrección diferente.

$$x_{correcto} = x + \left[2p_1 xy + p_2\left(r^2 + 2x^2\right)\right] \tag{2.9}$$

$$y_{correcto} = y + \left[p_1\left(r^2 + 2y^2\right) + 2p_2 xy\right] \tag{2.10}$$

2.8. Detección de características y coincidencia

Detección: identificar el punto de interés.

Descripción: La apariencia local alrededor de cada punto de característica se describe de una manera que (idealmente) es invariable bajo cambios en la iluminación, traslación, escala y rotación en el plano. Por lo general, terminamos con un vector descriptor para cada punto de característica.

Coincidencia: los descriptores se comparan entre las imágenes para identificar características similares. Para dos imágenes podemos obtener un conjunto de pares $(X_i, Y_i) \leftrightarrow \left(X_i', Y_i'\right)$, donde (X_i, Y_i) es una característica en una imagen y $\left(X_i', Y_i'\right)$ su característica correspondiente en la otra imagen.

2.8.1. Descriptor de características

Un descriptor de características es un algoritmo que toma una imagen y genera descriptores de características o vectores de características. Los descriptores de características codifican información interesante en una serie de números y actúan como una especie de "huella digital"numérica que se puede utilizar para diferenciar una característica de otra 2.26.

Figura 2.26: Descriptor de características

2.8.2. Coincidencia de características

La coincidencia de características o generalmente la coincidencia de imágenes 2.27, una parte de muchas aplicaciones de visión por computadora, como el registro de imágenes, la calibración de la cámara y el reconocimiento de objetos, es la tarea de establecer correspondencias entre dos imágenes de la misma escena u objeto. Un enfoque común para la coincidencia de imágenes consiste en detectar un conjunto de puntos de interés, cada uno asociado con descriptores de imágenes a partir de datos de imágenes. Una vez que las características y sus descriptores se han extraído de dos o más imágenes, el siguiente paso es establecer algunas coincidencias de características preliminares entre estas imágenes.

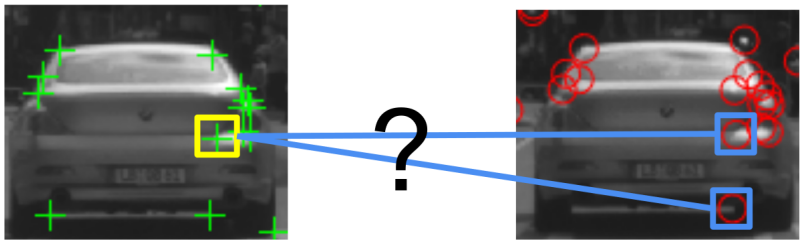


Figura 2.27: Coincidencia de características

2.9. Arquitectura de Red Neuronal U-Net

La arquitectura U-Net 2.28 se basa en la red totalmente convolucional y se modifica de manera que produce una mejor segmentación en imágenes médicas (18). En comparación con FCN-8, las dos diferencias principales son U-net es simétrica y las conexiones de omisión entre la ruta de disminución de muestreo y la ruta de muestreo ascendente aplican un operador de concatenación en lugar de una suma. Estas conexiones de omisión pretenden proporcionar información local a la información global durante el muestreo ascendente. Debido a su simetría, la red tiene una gran cantidad de mapas de características en la ruta de muestreo ascendente, lo que permite transferir información. En comparación, la arquitectura básica de FCN solo tenía varios mapas de características de clases en su ruta de muestreo ascendente.

La U-Net debe su nombre a su forma simétrica, que es diferente de otras variantes de FCN.

La arquitectura U-Net está separada en 3 partes:

- La ruta de contratación o ruta de disminucion de resolución.
- Cuello de botella.
- La ruta de expansión o muestreo ascendente.

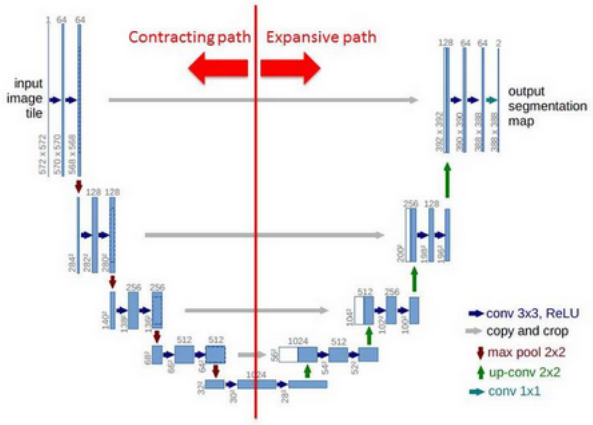


Figura 2.28: Ilustración de la arquitectura de la red neuronal convolucional profunda U-Net

La ruta de contracción se compone de 4 bloques. Cada bloque está compuesto de:

- Capa de convolución 3x3 + función de activación (con normalización por lotes)
- Capa de convolución 3x3 + función de activación (con normalización por lotes)

- 2x2 Max Pooling

Tenga en cuenta que el número de mapas de características se duplica en cada agrupación, comenzando con 64 mapas de característica para el primer bloque, 128 para el segundo, y así sucesivamente. El propósito de esta ruta de contracción es capturar el contexto de la imagen de entrada para poder realizar la segmentación. Esta información contextual aproximada se transferirá a la ruta de muestreo ascendente mediante conexiones de omisión.

Embotellamiento

Esta parte de la red se encuentra entre las rutas de contracción y expansión. El cuello de botella se construye a partir de simplemente 2 capas convolucionales (con normalización por lotes), con dropout.

Ruta de expansión o muestreo ascendente

La ruta de expansión también se compone de 4 bloques. Cada uno de estos bloques está compuesto de:

- Capa de desconvolución con stride 2.
- Concatenación con el mapa de características recortado correspondiente a la ruta de contracción.
- Capa de convolución 3x3 + función de activación (con normalización por lotes).
- Capa de convolución 3x3 + función de activación (con normalización por lotes).

El propósito de esta ruta de expansión es permitir una localización precisa combinada con información contextual de la ruta de contracción.

2.9.1. Clasificación con localización

El bloque de construcción más fundamental en visión por computadora es el problema de clasificación de imágenes donde, dada una imagen, esperamos que la computadora muestre una etiqueta discreta, que es el objeto principal de la imagen. En la clasificación de imágenes, suponemos que solo hay un objeto (y no varios) en la imagen 2.29.

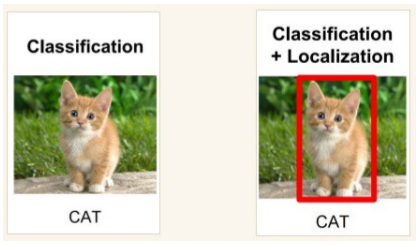


Figura 2.29: Clasificación y localización

En la localización junto con la etiqueta discreta, también esperamos que el cálculo localice exactamente dónde está presente el objeto en la imagen. Esta localización se implementa típicamente usando un cuadro delimitador que puede identificarse mediante algunos parámetros numéricos con respecto al límite de la imagen. Incluso en este caso, la suposición es tener solo un objeto por imagen.

2.9.2. Detección de objetos

La detección de objetos extiende la localización al siguiente nivel donde ahora la imagen no está limitada a tener un solo objeto, sino que puede contener múltiples objetos. La tarea es clasificar y localizar todos los objetos en la imagen. Aquí nuevamente la localización se realiza utilizando el concepto de cuadro delimitador 2.30.

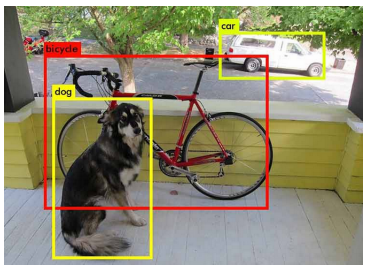


Figura 2.30: Detección de objetos con cuadro delimitador

2.9.3. Segmentación Semántica

El objetivo de la segmentación semántica de imagen es etiquetar cada píxel de una imagen con una clase correspondiente de lo que se representa. Debido a que estamos prediciendo para cada píxel en la imagen, esta tarea se conoce comúnmente como predicción densa (20).

Tenga en cuenta que, a diferencia de las tareas anteriores, la salida esperada en la segmentación semántica no son solo etiquetas y parámetros de cuadro delimitador. La salida en sí es una imagen de alta resolución (típicamente del mismo tamaño que la imagen de entrada) en la que cada píxel se clasifica en una clase particular. Por lo tanto, es una clasificación de imagen a nivel de píxel 2.31.

2.9.4. Segmentación de instancias

La segmentación de instancias está un paso por delante de la segmentación semántica en la que, junto con la clasificación de nivel de píxeles, esperamos que la computadora clasifique cada instancia de una clase por separado. Por ejemplo, en la imagen de abajo

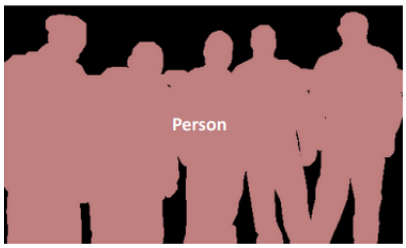


Figura 2.31: Segmentación Semántica

2.32 hay 3 personas, técnicamente 3 instancias de la clase "Persona". Los 3 se clasifican por separado (en un color diferente). Pero la segmentación semántica no diferencia entre las instancias de una clase particular (21).

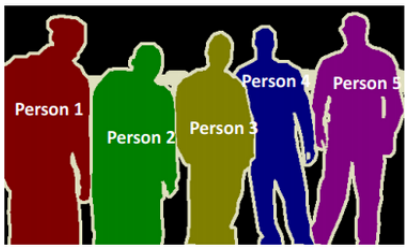


Figura 2.32: Segmentación de instancias

Capítulo 3

Diseño del experimento

El objetivo principal del presente proyecto de tesis es: implementar y diseñar dos redes neuronales convolucionales profundas (U-Net), las cuales son capaces de segmentar y detectar los objetos etiquetados en el conjunto de datos (KITTI). Para lograr esto es necesario procesar los datos de los sensores LIDAR y las imágenes de las cámaras.

La primera red neuronal convolucional profunda (U-Net) se entrenara con datos de los sensores Lidar, estos datos son accesibles en el conjunto de entrenamiento KITTI, con los archivos y parámetros necesarios para la calibración y transformación de coordenadas.

La segunda red neuronal profunda convolucional profunda se entrena con las imágenes de las cámaras que también son accesibles en el conjunto de datos KITTI, los archivos de calibración de las cámaras y las etiquetas de los objetos a detectar vienen adjuntos con su respectivo barrido de puntos de los sensores lidar, con el mismo procedimiento se usa el modelo de red neuronal convolucional profunda U-Net.

3.1. Conjunto de Datos KITTI

KITTI (11) es un conjunto de datos disponible para realizar el estudio con datos reales de un auto autónomo prototipo. EL vehículo es equipado con cuatro cámaras: 1 par estéreo de cámara a color y 1 par estéreo de cámara en escala de grises. Las cámaras en color y en escala de grises están montadas una cerca de la otra (6 cm), la línea base de ambas plataformas estéreo es de aproximadamente 54 cm. Esta configuración permite obtener de la cámara izquierda y derecha información tanto en color como en escala de grises. Mientras que las cámaras de color (obviamente) vienen con información de color, las imágenes de la cámara en escala de grises tienen un mayor contraste y un poco menos de ruido.

Todas las cámaras están sincronizadas a aproximadamente 10 Hz con respecto al escáner láser Velodyne. El gatillo está montado de tal manera que las imágenes de la cámara coinciden aproximadamente con los láseres Velodyne mirando hacia adelante (en la dirección de conducción).

Todas las imágenes de la cámara se proporcionan como secuencias png comprimidas y rectificadas sin pérdidas. La resolución de la imagen nativa es de 1382x512 píxeles y un poco menos después de la rectificación.

El ángulo de apertura de las cámaras (izquierda-derecha) es de aproximadamente 90 grado.

Datos de escaneo láser Velodyne 3D

Los datos contienen 4 * valores numéricos, donde los primeros 3 valores corresponden a x, y, z, el último valor es la información de reflectancia. Todos los escaneos se almacenan alineados en fila, lo que significa que los primeros 4 valores corresponden a la primera medición.

Las categorías de objetos se clasifican de la siguiente manera

- 'Coche'
- 'Camioneta'
- 'Camión'
- 'Peatón'
- 'Persona'
- 'Ciclista'
- 'Tranvía'
- 'Varios'

3.2. Pre-procesamiento de la nube de puntos sensores LIDAR

Está claro para nosotros que la odometría de la rueda de un auto autónomo para nuestro proyecto, no es suficiente para estimar su movimiento. Sabemos esto porque podemos superponer los escaneos LIDAR del auto en nuestras mentes y tener una idea de cómo el movimiento estimado del auto se está desviando de su movimiento real. Si podemos hacer esto en nuestras mentes, ¿podríamos decirle al auto cómo hacerlo? ¿Puede el auto autónomo usar sus escaneos LIDAR para estimar su propio movimiento?, estas preguntas se contestan a continuación ya que son el principio de la construcción de los objetos los cuales reciben los pulsos de laser del sensor LIDAR.

A través de un proceso llamado coincidencia de escaneo. Básicamente, el objetivo es tomar un nuevo escaneo del LIDAR del auto y encontrar la transformación que mejor alinee el nuevo escaneo con escaneos anteriores o algún tipo de mapa abstracto. Hay muchas formas de implementar esta idea para este proyecto se muestra el método más simple: usar el algoritmo de punto más cercano iterativo (ICP) 3.1 para alinear el escaneo LIDAR más nuevo con el escaneo anterior.

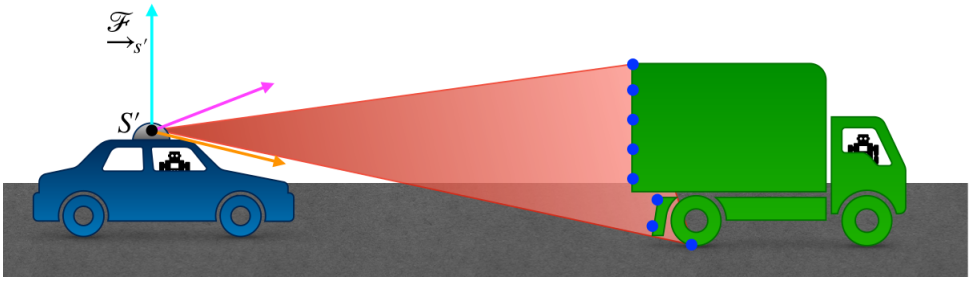


Figura 3.1: Estimación de estado mediante registro de conjunto de puntos.

3.2.1. Punto iterativo más cercano nube de puntos

El algoritmo ICP (23) implica 3 pasos: asociación, transformación y evaluación de errores. Estos se repiten hasta que los escaneos estén alineados satisfactoriamente. Primero imaginemos que queremos encontrar la transformación que alinea los dos escaneos que se muestran a continuación. El escaneo anterior, conocido como el objetivo. Mientras que el nuevo escaneo, también llamado fuente, está en magenta. El objetivo es encontrar la transformación rígida (rotación y traslacion) que mejor alinee la fuente con el objetivo 3.2.

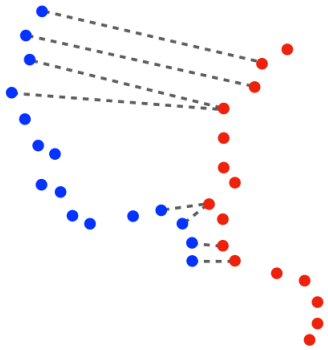


Figura 3.2: Asociar cada punto en $p_s{}'$ con el punto más cercano en p_s.

El algoritmo en cada paso de iteración, primero calcula las correspondencias de puntos usando el criterio del punto más cercano, y después calcula la transformación rígida que minimiza la función de coste para dichas correspondencias, utilizando uno de los métodos en forma cerrada, a continuación se muestra el diagrama de flujo 3.3.

Intuición: cuando se encuentra el movimiento óptimo, los puntos correspondientes estarán más cerca uno del otro que de otros puntos.

Heurística: para cada punto, el mejor candidato para un punto correspondiente es el punto más cercano en este momento.

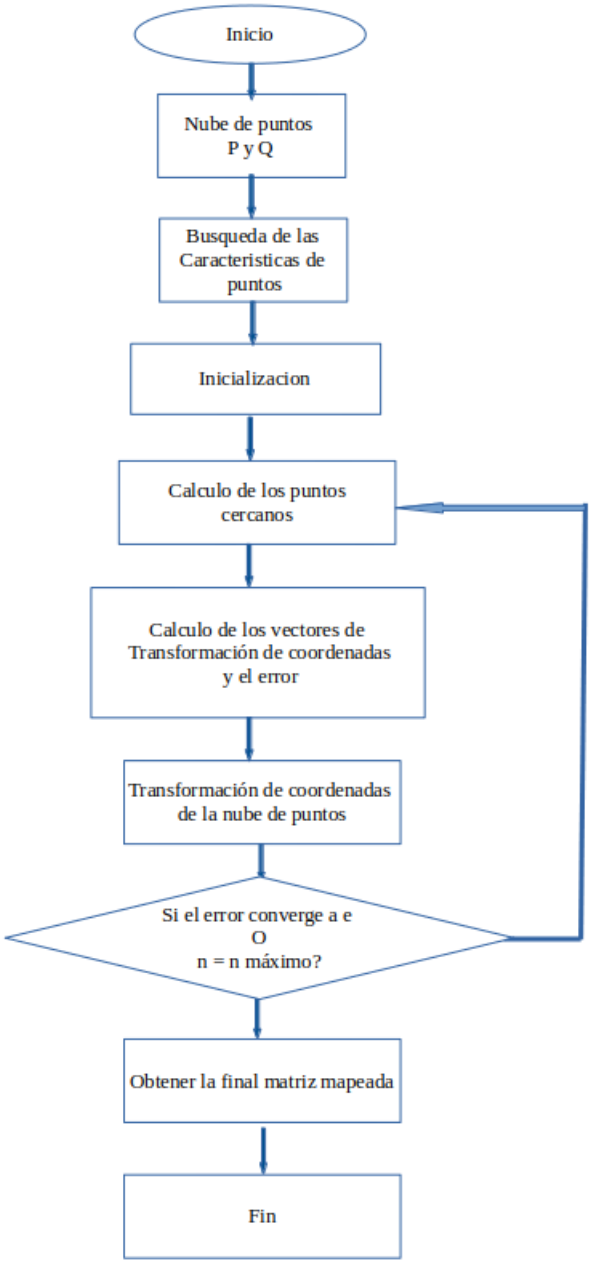


Figura 3.3: Diagrama de flujo del algoritmo ICP.

Nuestro primer paso para estimar esta transformación es decidir qué puntos en el escaneo fuente corresponden a las mismas características físicas que los puntos en el escaneo objetivo. La forma más sencilla de hacerlo es a través de una búsqueda de vecino más cercano: los puntos en el escaneo de origen se asocian al punto más cercano

en el escaneo de destino. En la imagen a continuación, he encontrado los vecinos más cercanos de cada punto en el escaneo objetivo. Los puntos asociados están conectados con las lineas azules 3.4.

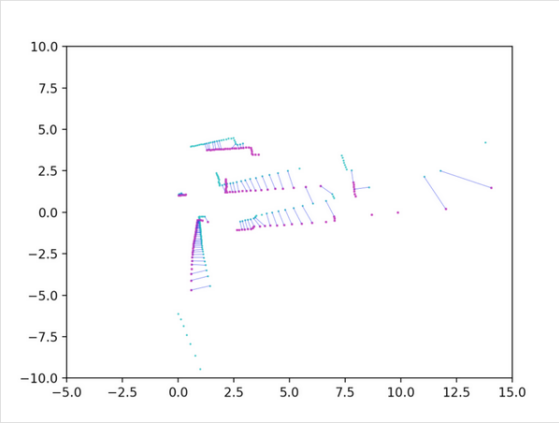


Figura 3.4: Aplicando algoritmo de búsqueda de vecino más cercano.

Podemos ver de inmediato algunos errores en la búsqueda de vecinos más cercanos, pero en general, las asociaciones representadas tirarán de los puntos de origen en la dirección correcta. El siguiente paso en el proceso es la transformación. Encontramos la transformación que, cuando se aplica a los puntos de origen, minimiza la distancia cuadrática media entre los puntos asociados: $T^{'} = argmin\frac{1}{N}\sum_{i}^{N}\|t_i - Ts_i\|^2$, donde $T^{'}$ es la transformación final estimada y t_i y s_i son los puntos objetivos y la fuente de puntos, respectivamente. El resultado de esta estimación se muestra a continuación 3.5.

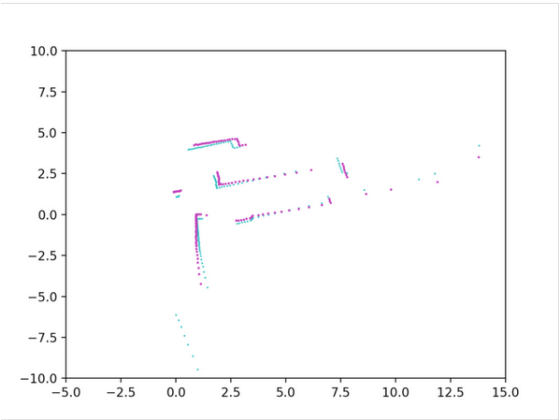


Figura 3.5: Con el proceso de transformación las aproximaciones y coincidencia de los puntos es más exacta.

Después de esto, evaluamos el error en la alineación como $e = \frac{1}{N}\sum_i^N \|t_i - Ts_i\|^2$ y decidimos si necesitamos repetir el proceso anterior. En este caso, nuestros escaneos aún no están bien alineados, por lo que rehacemos las asociaciones con los puntos fuente transformados y repetimos el proceso. Después de cinco iteraciones nuestros escaneos, el algoritmo encuentra una alineación bastante buena 3.6.

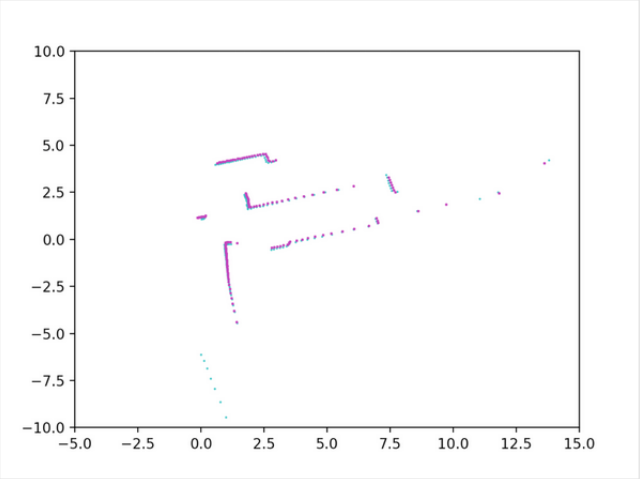


Figura 3.6: Repitiendo el algoritmo ICP la aproximación de los puntos es mejor en cada iteración, con la coincidencia de los puntos rojos con los azules.

3.2.2. Visualización nube de Puntos LIDAR

El conjunto de datos y su respectivo preprocesamiento son los primeros pasos que se realizan para poder llevarlos a la entrada de una red neuronal convolucional profunda, la teoría demuestra que se necesita una gran cantidad de datos para obtener mejores y precisos resultados en nuestra tarea detección y segmentación.

Cada escaneo de datos LIDAR se almacena como una nube de puntos tridimensional. El procesamiento eficiente de estos datos mediante una indexación y búsqueda rápidas es clave para el rendimiento de la pila de puntos para su procesamiento. Esta eficiencia se logra utilizando la nube de puntos como un objeto (pointcloud), que organiza internamente los datos utilizando una estructura de datos K-d tree. En la Figura 3.7 se muestra la estructura de una nube de puntos de los sensores LIDAR, esta estructura es característica para los sensores de la marca velodyne, los cuales son usados en el conjunto de datos que estamos trabajando (15).

```
ptCloud =

  pointCloud with properties:

     Location: [32×1083×3 single]
        Color: []
       Normal: []
    Intensity: [32×1083 single]
        Count: 34656
      XLimits: [-80.0444 87.1780]
      YLimits: [-85.6287 92.8721]
      ZLimits: [-21.6060 14.3558]
```

Figura 3.7: Localización es la estructura de los datos en celdas, es decir se divide cada barrido en columnas, filas y el número de canales, el color es un atributo del sensor, la normal es un atributo asignado a el etiquetamiento del sensor y la nube de puntos, intensidad describe la distribución de los puntos para cada lote del barrido, el contador es el número de puntos totales en la nube de puntos, los últimos parámetros son las coordenadas que limitan la nube de puntos, un barrido de puntos puede tener un periodo de hasta 20 segundos, esto depende de la configuración y calibración del sensor.

El lenguaje de programación utilizado para el procesamiento y visualización de los datos es Python y sus librerías de cálculo matemático. El formato de extensión de los datos de los sensores LIDAR es .bin es una representación comprimida de todos los puntos que componen un barrido 3.8.

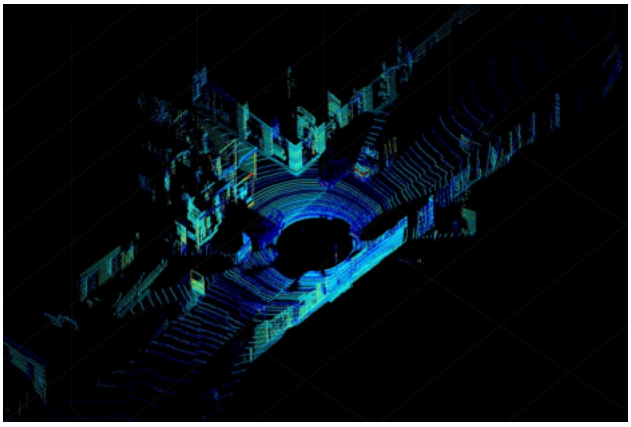


Figura 3.8: Ilustración de un barrido de puntos con extensión .bin de los sensores LIDAR y proyección visual 3D.

3.2.3. Segmentación de la nube de puntos con métodos probabilísticos

Este ejemplo muestra cómo rastrear vehículos utilizando mediciones de un sensor LIDAR montado en la parte superior de un vehículo ego. Los sensores LIDAR informan las mediciones como una nube de puntos. El ejemplo ilustra el flujo de trabajo para procesar la nube de puntos y rastrear los objetos. Los datos lidar utilizados en este ejemplo se registran en un escenario de conducción en carretera del conjunto de datos KITTI. En este ejemplo, utiliza los datos registrados para rastrear vehículos con un rastreador conjunto de asociación de datos probabilísticos (JPDA) (14) y un enfoque de modelo múltiple interactivo (IMM) (29).

El objetivo de presentar este método de segmentación de la nube de puntos es para comparar los métodos que usan algoritmicos probabilisticos con métodos sofisticados usando redes neuronales profundas.

Debido a las capacidades de alta resolución del sensor LIDAR, cada escaneo desde el sensor contiene una gran cantidad de puntos, comúnmente conocidos como una nube de puntos. Estos datos sin procesar deben preprocesarse para extraer objetos de interés, como automóviles, ciclistas y peatones. Este procesamiento previo requiere de los siguientes pasos, lectura de los datos para poder visualizarlos, dimensionar el ambiente de visualización para tener una perspectiva completa del barrido de los puntos, etiquetar los puntos que correspondan a la distribución de un conjunto de puntos de interés, eliminar el ruido y puntos que no formen parte de la segmentación de nuestro objeto. Estos métodos probabilisticos separan en varias regiones de interés la nube de puntos, con ello consiguen separar por ejemplo los bordes, el objeto segmentado, los obstáculos a su alrededor 3.9.

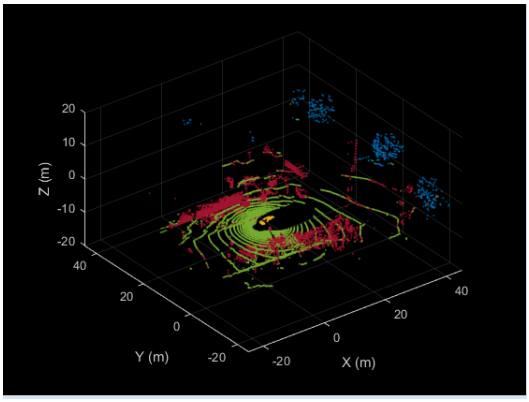


Figura 3.9: Se divide por colores las regiones de interés la nube de puntos, este preprocesamiento es necesario para aplicar algoritmos probabilisticos en un conjunto de barrido de puntos, este ejemplo solo aplica a un barrido.

3.3. Algoritmo Bird eye view sobre la nube de puntos

Modelo de transformación de vista de pájaro, para la transformación de coordenadas de la nube de puntos de los sensores LIDAR, implica transformar un conjunto de puntos de proyección en perspectiva de pájaro correspondiente.

Un algoritmo de búsqueda de parámetros de ajuste estima los parámetros que se utilizan para transformar las coordenadas de la nube de puntos, este algoritmo se describe con detalle enseguida.

Coordenadas de la región para transformarse en una imagen a vista de pájaro (3), especificada como un vector de cuatro elementos de la forma [xmin xmax ymin ymax]. Las unidades están en coordenadas mundiales, como metros o pies, según lo determinado por la propiedad sensor LIDAR. Las cuatro coordenadas definen el espacio de salida en el sistema de coordenadas del vehículo(x_v, y_v). La proyección del sensore LIDAR montado en el auto autónomo se muestra en la siguiente Figura 3.10.

La transformación de perspectiva que nos interesa es una transformación a vista de pájaro que nos permite ver un carril desde arriba. Además de crear una representación de una imagen a vista de pájaro, una transformación de perspectiva también se puede utilizar para todo tipo de puntos de vista diferentes.

El algoritmo para crear una representación en vista de pájaro necesita de los datos de la nube de puntos, los cuales estan en 3D.

- Los datos o archivos .bin o .pcd los cuales contienen los puntos en las cordenadas x, y,z.
- Resolución en x, y es la representación en metros equivalentes a los pixeles de la imagen voxel.

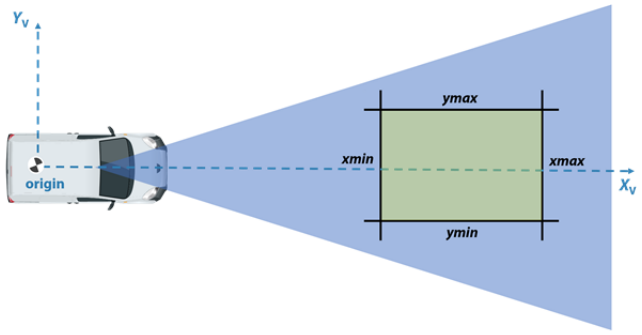


Figura 3.10: Es la proyección de la región con las dimensiones del voxel imagen del barrido de puntos de los sensores LIDAR.

- Resolución en z representación en metros en el eje correspondiente.
- Rango de la imagen voxel (lado izquierdo y lado derecho) los cuales delimitan al rectángulo de la vista.
- Rango longitudinal, definir los límites a lo largo de la vista, se toma como referencia el centro del vehículo y la dirección del etiquetamiento de los objetos a detectar.
- Rango en el eje z para definir la altura con un mínimo y máximo valor en metros.
- Retorno de un array con las características de arriba mencionadas, con una densidad e intensidad.

La perspectiva de transformación vista de pájaro necesitamos de los siguientes archivos:

- Imágenes de entrenamiento.
- Archivos de calibración de las cámaras y sensores Lidar.
- Archivos .bin con los puntos de los sensores Lidar.
- Directorio para guardar las imágenes voxel con vista de pájaro.

A continuación se muestran los datos necesarios para realizar la transformación de coordenadas en la siguiente Figura 3.11.

Con los archivos de calibración podemos transformar, rotando, trasladando las coordenadas 3D en una proyección 2D de los puntos que limitan la etiqueta de los objetos.

Se definen las clases que se desea realizar su proyección en la imagen voxel, estos datos nos ayudan para crear una nueva base de datos, con las nuevas coordenadas de las etiquetas correspondientes a las clases a procesar.

	type	truncated	occluded	alpha	bbox_left	bbox_top	bbox_right	\
0	Car	0.8	0	-2.09	1013.39	182.46	1241.00	
1	Car	0.0	0	1.95	354.43	185.52	549.52	
2	Car	0.0	0	-1.78	819.63	178.12	926.85	
3	Car	0.0	2	-1.69	800.54	178.06	878.75	
4	Car	0.0	0	1.80	558.55	179.04	635.05	
5	Car	0.0	2	1.77	598.30	178.68	652.25	
6	Car	0.0	1	-1.67	784.59	178.04	839.98	
7	Car	0.0	1	1.92	663.74	175.36	707.21	

	bbox_bottom	height	width	length	pos_x	pos_y	pos_z	rot_y
0	374.00	1.57	1.65	3.35	4.43	1.65	5.20	-1.42
1	294.49	1.43	1.70	3.95	-2.39	1.66	11.80	1.76
2	251.56	1.51	1.60	3.24	5.85	1.64	16.50	-1.44
3	230.56	1.45	1.74	4.10	6.87	1.62	22.05	-1.39
4	230.61	1.54	1.68	3.79	-0.38	1.76	23.64	1.78
5	218.17	1.49	1.52	3.35	0.64	1.74	29.07	1.79
6	220.10	1.53	1.65	4.37	7.88	1.75	28.53	-1.40
7	204.15	1.64	1.45	3.48	4.50	1.80	42.85	2.02

Figura 3.11: Para cada objeto dinámico dentro del campo de visión de la cámara de referencia, proporcionamos anotaciones en forma de tracklets de cuadro delimitador 3D, representados en coordenadas Velodyne del sensor LIDAR. Definimos las clases 'Coche', 'Furgoneta', 'Camión', 'Peatón', 'Persona (sentado)', 'Ciclista', 'Tranvía' y 'Misceláneo' (por ejemplo, Remolques, Segways).

La transformación de coordenadas (7) es limitado por los rangos de la proyección del voxel, como se describió anteriormente, dentro de estos límites de trasformación se limita con las siguientes coordenadas del voxel $[xmin_img, xmax_img, ymin_img, ymax_img]$ 3.12.

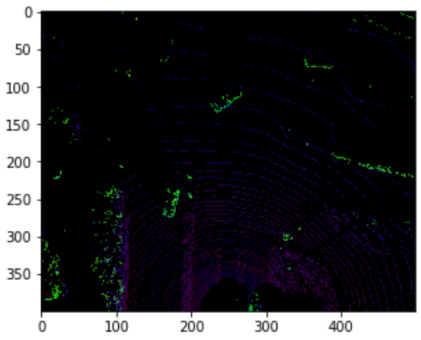


Figura 3.12: El vóxel (del inglés volumetric pixel) es la unidad cúbica que compone un objeto tridimensional. Constituye la unidad mínima procesable de una matriz tridimensional y es, por tanto, el equivalente del píxel en un objeto 2D.

3.3.1. Preparación de los datos

En esta sección se describe la segunda parte del procesamiento de los datos de los sensores LIDAR y las imágenes de las cámaras. El conjunto de datos contiene ahora 7481 imágenes con su correspondiente imagen voxel, con el previo preprocesamiento de las secciones anteriores. El ambiente de las escenas descritas coinciden para ambos archivos, lo necesario para entrenar las redes neuronales convolucionales profundas por separado 3.1.

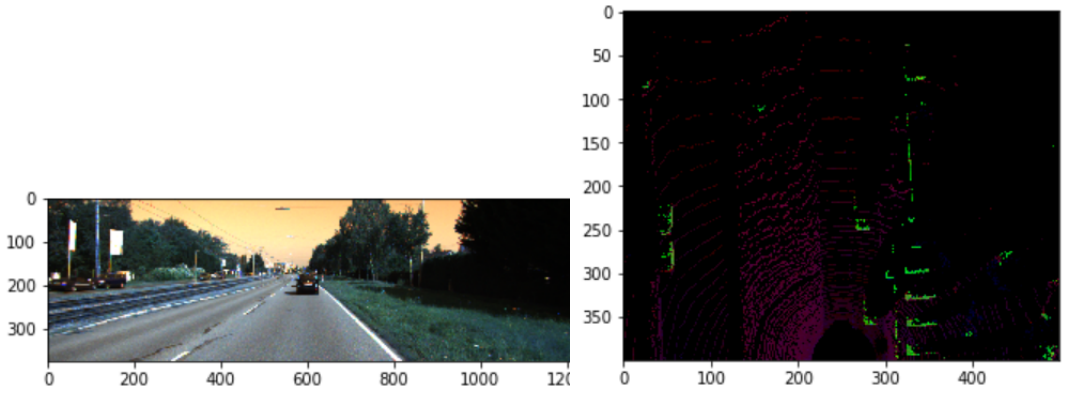
Paso de preparación de los datos.	**Propósito.**
Para evitar problemas debido al pedido de los datos.	Mezcla aleatoria de los datos.
Para evitar características individuales o conjuntos de características dominan la respuesta de nuestro clasificador.	Normalización de las características.
Media y varianza unitaria.	Típicamente a cero.
Estimar la generalización del modelo a nuevos datos.	Dividir los datos en un conjunto de entrenamiento y prueba.
Para evitar tener su algoritmo, simplemente clasifique todo como perteneciente a la clase mayoritaria.	Preparar un conjunto de datos de equilibrio, es decir, tener muchos ejemplos positivos como negativos, o en el caso de problemas de varias clases, aproximadamente el mismo número de casos de cada clase.

Tabla 3.1: Preprocesamiento de datos.

3.3.2. Preprocesamiento Imágenes

El procesamiento de los datos utilizado en este proyecto esta basado en el siguiente artículo,: Predicción del ángulo de dirección del automóvil sin conductor basado en el reconocimiento de imágenes (8), el cual se obtienen buenos resultados en la predicción correcta del ángulo de la dirección del auto para mantenerse dentro de su carril por si solo. El aumento y procesamiento de los datos es necesario para simular diferentes ambientes por ejemplo niebla, lluvia, suciedad de los lentes de las cámaras, etc, este artículo, muestra una gran variedad de técnicas que se aplican a nuestro proyecto de Tesis.

1. Leer imagen: en un principio es tener acceso a los datos para visualizar, en este paso, almacenamos la ruta a nuestro conjunto de datos de imagen de cámaras 3.13a y Lidar 3.13b en una variable y luego creamos una función para cargar carpetas que contienen imágenes en matrices.

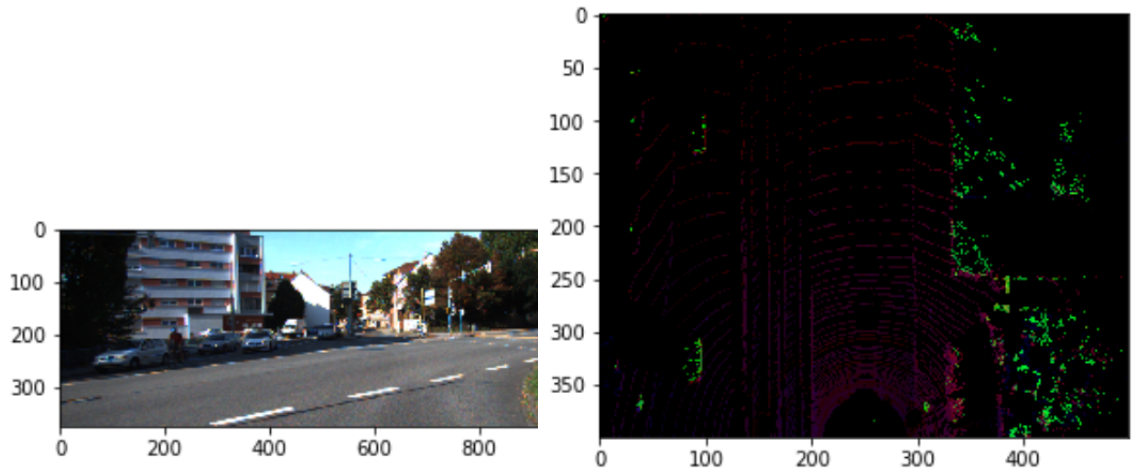


(a) Imagen de cámara. **(b)** Imagen voxel.

Figura 3.13: Lectura de los datos de entrenamiento.

2. Eliminar el ruido: Gaussian blur, (también conocido como suavizado gaussiano) es el resultado de desenfocar una imagen mediante una función gaussiana. Es un efecto ampliamente utilizado en software de gráficos, generalmente para reducir el ruido de la imagen. El efecto visual de esta técnica de desenfoque es un desenfoque suave que se asemeja al de ver la imagen a través de una pantalla translúcida, claramente diferente del efecto bokeh producido por una lente desenfocada o la sombra de un objeto bajo la iluminación habitual. El suavizado gaussiano también se utiliza como una etapa de preprocesamiento en algoritmos de visión por computadora para mejorar las estructuras de imagen a diferentes escalas. A continuación se muestra las imágenes de cámara 3.14a y la imagen voxel 3.14b.

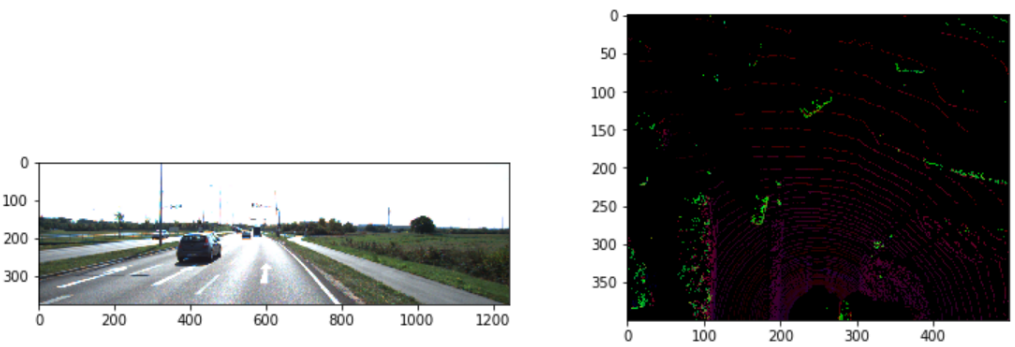
3. Aumento de brillo: el brillo se cambia aleatoriamente para simular diferentes condiciones de luz. Generamos imágenes aumentadas con diferente brillo al convertir



(a) Imagen de cámara. **(b)** Imagen voxel.

Figura 3.14: Eliminación de ruido de los datos.

primero las imágenes a HSV, ampliar o reducir el canal V y volver a convertir al canal RGB. Las siguientes son imágenes típicas aumentadas y aumento de sombras aleatorias proyectadas en las imágenes. La intuición es que incluso si la cámara ha sido sombreada (tal vez por lluvia o lluvia), todavía se espera que el modelo detecte los objetos de manera correcta. Esto se implementa eligiendo puntos aleatorios y sombreando todos los puntos en un lado de la imagen 3.15a. El mismo procedimiento para la imagen voxel 3.15b.

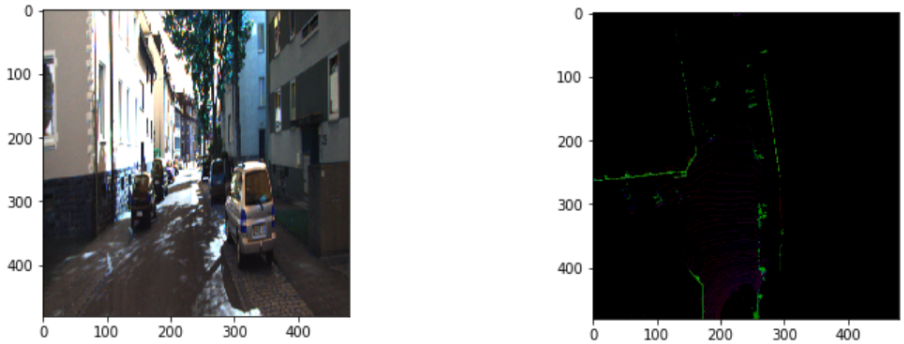


(a) Imagen de cámara. **(b)** Imagen voxel.

Figura 3.15: Aumento de brillo y aumento de sombras aleatorias proyectadas en las imágenes.

4. Escalado de imagen: una vez que nos hemos asegurado de que todas las imágenes

sean cuadradas (o tengan una relación de aspecto predeterminada), es hora de escalar cada imagen de manera adecuada. Hemos decidido tener imágenes con ancho y alto de 480 píxeles 3.21a. Tendremos que escalar el ancho y el alto de cada imagen. Existe una amplia variedad de técnicas de aumento y disminución de la escala y usualmente usamos una función de Numpy de Python para hacer esto por nosotros 3.21b.

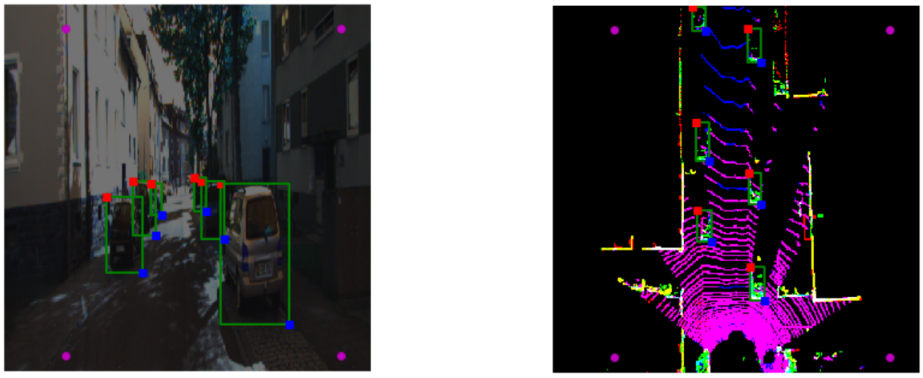


(a) Imagen de cámara. **(b)** Imagen voxel.

Figura 3.16: Dimensionando los datos al mismo tamaño.

5. Transformación de perspectiva (17): corrección de distorsión y coincidencia de características de la imagen, usualmente se eligen cuatro puntos de origen manualmente, en nuestro caso tomamos las coordenadas de los cuadros delimitadores. Por ejemplo, muchos algoritmos de transformación de perspectiva detectarán mediante programación cuatro puntos de origen en una imagen 3.17a en función de la detección de bordes o esquinas, analizarán atributos como el color y los píxeles circundantes.

   Los puntos de interés deben tener las siguientes características: Prominencia, distintiva, identificable y diferente de su inmediato vecino, Repetibilidad, se puede encontrar en varias imágenes usando las mismas operaciones, Localidad, ocupa un subconjunto relativamente pequeño de espacio de imagen, Cantidad: suficientes puntos representados en la imagen, eficiencia: tiempo de cálculo razonable, aplicamos el mismo procedimiento a la imagen voxel 3.17b.



(a) Imagen de cámara **(b)** Imagen voxel

Figura 3.17: Buenas características de imagen deben ser Prominentes, Repetibles, Locales, Eficientes y Numerosas.

La detección y coincidencia de características 3.2 es una tarea importante en muchas aplicaciones de visión por computadora, como la estructura del movimiento, la recuperación de imágenes, la detección de objetos y más.

Rasgos	Características
Forma únicamente	Gradiente de la intensidad de pixel
Color y forma.	intensidad bruta del pixel.
Color únicamente.	Histograma de la intensidad del pixel.

Tabla 3.2: Los rasgos, describen las características de un objeto, y con imágenes, todo se reduce a la intensidad y gradientes de intensidad, y cómo estas características capturan el color y la forma de un objeto.

6. Normalización de entradas de imagen: la normalización de datos es un paso importante que garantiza que cada parámetro de entrada (píxel, en este caso) tenga una distribución de datos similar. Esto hace que la convergencia sea más rápida mientras se entrena la red.

7. Reducción de la dimensionalidad: podríamos optar por contraer los canales RGB en un solo canal de escala de grises. A menudo hay consideraciones para reducir otras dimensiones, cuando se permite que el rendimiento de la red neuronal sea invariable a esa dimensión, o para hacer que el problema de entrenamiento sea más manejable.

Esta técnica es importante para el conjunto de imágenes con las máscaras de los objetos a detectar, el tercer canal es de una sola dimensión.

8. Segmentación: para propósitos generales la segmentación es una técnica muy importante para definir las clases a las cuales pertenece cada pixel dentro de una imagen, en el proyecto, esta clasificación y predicción de la segmentación de clases es una tarea que se encarga la red neuronal (20).

   En la predicción típica en píxeles, dibujamos polígonos alrededor del objeto de interés para dibujar máscaras exactas con la silueta de los objetos de interés. Este trabajo no contaba con esa información, por lo tanto, se utiliza la región dentro de los cuadros delimitadores como máscaras para definir objetos, estas coordenadas delimitan las dimensiones de los objetos etiquetados.

   La creación de las máscaras para cada objeto etiquetado en las proyecciones 2D en las imágenes tipo voxel e imágenes de las cámaras. La red neuronal profunda U-Net necesita la imagen real con su correspondiente imagen con las máscaras de cada objeto, para su posterior predicción de segmentación y detección.

   Para creación de las máscaras correspondientes en la imagen voxel es necesario realizar transformaciones de coordenadas 3D a 2D para cada barrido de puntos, esto es posible con los parámetros de calibración de la cámara como de los sensores LIDAR; también es necesario de los archivos que contienen las coordenadas de la posición de los objetos etiquetados, estas mediciones son validas para proyectar los cuadros delimitadores sobre la imágenes de las cámaras, pero no para las proyecciones de vista de pájaro de la imagen voxel.

   Rotación y traslación de coordenadas se toma un punto en las coordenadas del sensor Velodyne y lo transforma en el sistema de coordenadas de la cámara. Del mismo modo sirve como un representación del marco de coordenadas Velodyne en coordenadas de cámara. Estos archivos son adjuntados al conjunto de datos para su procesamiento.

   Una vez realizado este paso las bases de datos quedan de la siguiente manera, ambos archivos están evaluados para la misma cantidad de clases, se muestra solo algunos canales para describir la estructura 3.18.

	File_Path	Frame	Label	xmin	xmax	ymin	ymax
0	/home/fernando/Descargas/Dataset-KITTI/kitti-o...	004223.png	Car	990.42	1216.43	176.29	273.79
1	/home/fernando/Descargas/Dataset-KITTI/kitti-o...	002429.png	Car	853.57	1139.61	174.42	272.88
2	/home/fernando/Descargas/Dataset-KITTI/kitti-o...	004668.png	Car	339.51	432.79	194.51	253.30
3	/home/fernando/Descargas/Dataset-KITTI/kitti-o...	004668.png	Car	666.79	734.62	174.33	224.69
4	/home/fernando/Descargas/Dataset-KITTI/kitti-o...	004668.png	Car	400.11	471.62	190.95	234.73

Figura 3.18: Base de datos necesaria para el entrenamiento de la Red neuronal U-Net.

El enmascaramiento aparece cuando desea extraer, modificar, contar o manipular valores en una matriz de acuerdo con algún criterio: por ejemplo, es posible que desee contar todos los valores mayores que un cierto valor, o tal vez eliminar todos los valores atípicos que están por encima de algunos límite. En NumPy librería de Python, el enmascaramiento booleano es a menudo la forma más eficiente de realizar este tipo de tareas y es el cual se aplica al conjunto de datos para obtener las máscaras correspondientes.

A continuación se muestra las máscaras correspondientes a la imagen tipo voxel 3.19.

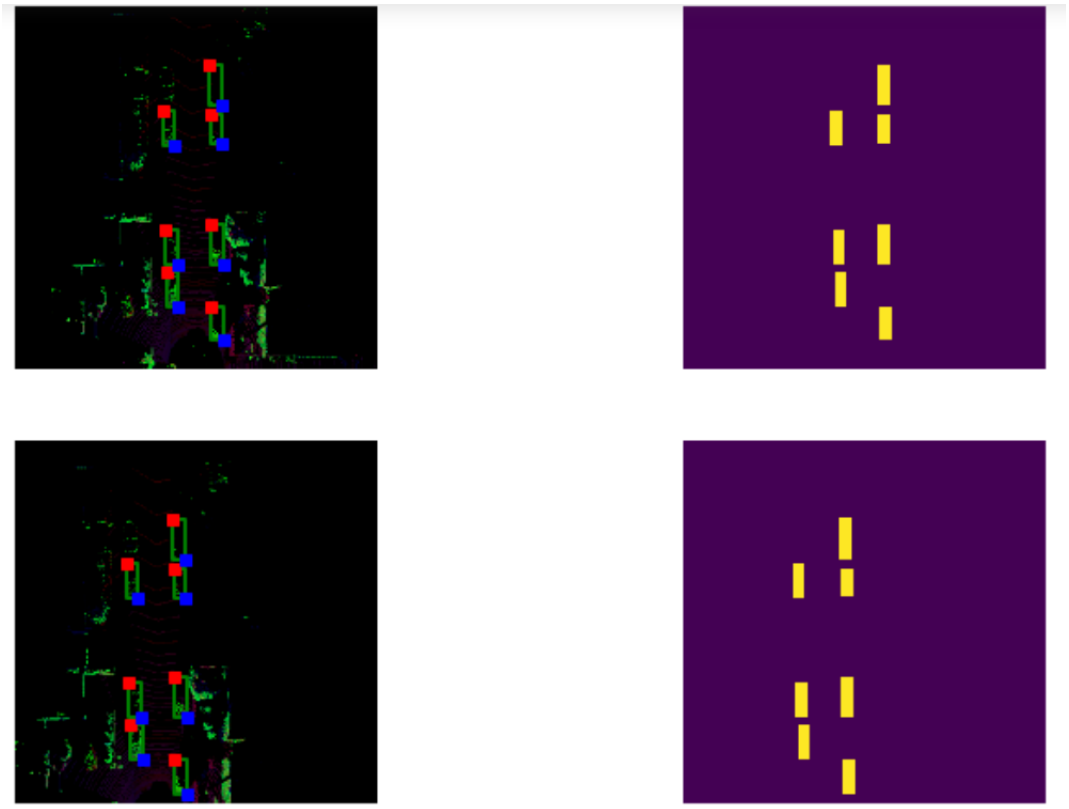
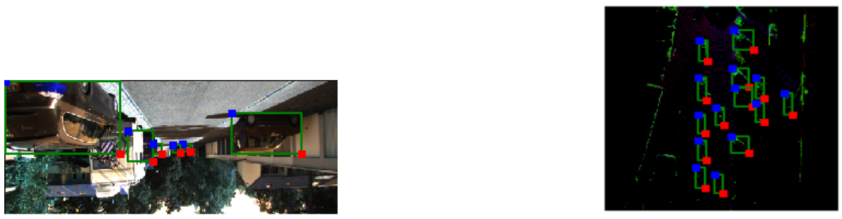


Figura 3.19: Voxel imagen con perspectiva de vista de pájaro con sus correspondientes máscaras, del lado derecho se muestra en rectángulos verdes.

El mismo principio de creación de las máscaras es aplicable al conjunto de imágenes de las cámaras, a continuación se muestra la imagen real con el cuadro delimitador detector de características y su respectiva imagen con las máscaras de todos los objetos etiquetados 3.20.

Figura 3.20: La red neuronal convolucional profunda U-Net necesita la imagen real con su correspondiente imagen con las máscaras de los objetos a predecir su segmentación y detección.

9. Aumento de datos con rotación y traslación (26): otra técnica común de procesamiento implica aumentar el conjunto de datos existente con versiones perturbadas de las imágenes existentes. Escalado, rotaciones, traslaciones y otras transformaciones afines son típicas. Esto se hace para exponer la red neuronal a una amplia variedad de variaciones. Esto hace que sea menos probable que la red neuronal reconozca características no deseadas en el conjunto de datos.



(a) Imagen de cámara **(b)** Imagen voxel

Figura 3.21: Rotación y Traslación

3.4. Modelo de red neuronal profunda en Autos autónomos

Aplicando investigaciones de vanguardia el modelo de red neuronal U-Net (24) se asemeja el proyecto a las redes neuronales que usa Tesla, sus autos equipados con hardware y software instalado para alcanzar la autonomía total. Con el propósito de entrenar redes neuronales profundas en problemas que van desde la percepción hasta el control. Sus modelos de redes neuronales con cámara analizan imágenes en bruto para realizar segmentación semántica, detección de objetos y estimación de profundidad monocular, con vista panorámica (vista de pájaro) toman videos de todas las cámaras para mostrar el diseño de la carretera, la infraestructura estática y los objetos 3D directamente en la vista de arriba hacia abajo. A continuación se muestra el proceso de entrenamiento de un auto autónomo con datos reales el cual es la base del diseño del proyecto 3.22.

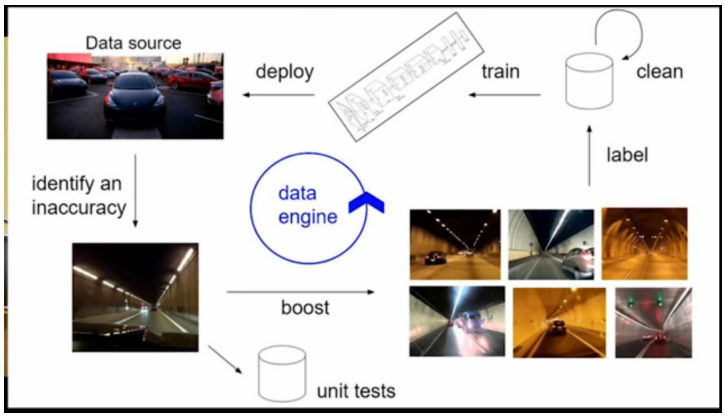


Figura 3.22: Arquitectura de el procesamiento de los datos en un auto autónomo: 1.- Imágenes y datos de sensores para su etiquetamiento usando aprendizaje supervisado, 2.- Entrenamiento de los datos en una red neuronal convolucional profunda, 3.-Pruebas de clasificación y detección para mejorar el rendimiento, 5.- Fuente de generación de nuevos datos(cámaras y sensores(LIDAR o radares).

- Fuente generadora de datos: Estos provenientes de los sensores LIDAR, radares o las cámaras, se cuenta con una base de datos la cual guarda y administra los datos generados para su posterior procesamiento por computadora. Este previo procesamiento es necesario para que la red neuronal pueda interpretar y sea entrenada de manera eficiente.
- Entrenamiento: Al contar con una fuente generadora de una variedad de datos en diferentes escenarios es posible actualizar a la red neuronal de manera constante

para que aprenda a clasificar, detectar, o según sea la tarea a realizar. Los datos de entrenamiento se explican por sí mismos. En general, una mayor cantidad de datos de entrenamiento hará que la red neuronal comprenda mejor su distribución de datos. Más datos harán que su red capacitada funcione mejor. Siempre se pone prioridad en esta parte en la división de los datos.

- Validación: Lo siguiente son los datos de validación. Es la porción de datos que se evaluará durante el proceso de capacitación. Estos datos se usan para estimar el error de predicción.
- Prueba del Modelo: El modelo muestra también una parte muy importante la cual consiste en probar la eficiencia de la red neuronal en escenarios que no ha visto. Esta es la manera de probar la eficiencia de la red neuronal y también de optimizarla con nuevos datos. Los datos de prueba. Estos son los datos utilizados para evaluar el modelo de red neuronal. Si la red funciona bien en los datos de prueba, puede llevar la red al nivel de producción.
- Limpieza y etiquetamiento de los datos es un paso muy importante en la adquisición de los datos en un auto autónomo. Después de obtener un conjunto de datos etiquetados, el modelo de aprendizaje automático se pueden aplicar a los datos para que los nuevos datos no etiquetados se puedan presentar al modelo y se pueda adivinar o predecir una etiqueta probable para esa pieza de datos no etiquetados.

Una simple presentación de una red neuronal profunda 3.23 autónomo para automóvil que toma entradas de la cámara, el sensor de detección de luz y rango (LIDAR) y el sensor IR (infrarrojo), y emite el ángulo de dirección, la decisión de frenado y la decisión de aceleración. Todos estos elementos forman parte del sistema de detección de objetos e interacción con ambientes reales.

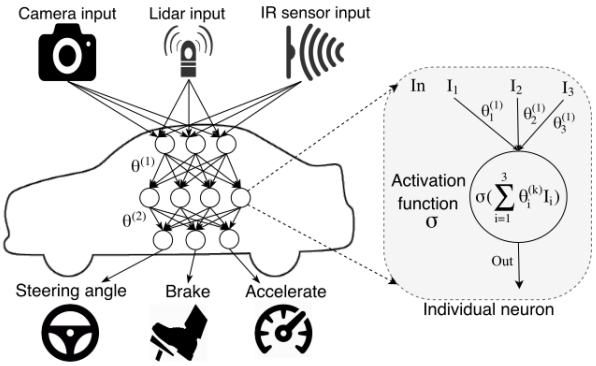


Figura 3.23: Una red convolucional profunda para un automóvil autónomo que toma entradas de la cámara, el sensor de detección de luz y rango (LiDAR) y el sensor IR (enmarcado), para ser capaz de emitir el ángulo para la columna de dirección, la decisión de frenado y de aceleración.

3.5. Entrenamiento de la Redes neuronales U-Net

El U-Net fue desarrollado por Olaf Ronneberger (22), para la segmentación de imágenes biomédicas. La arquitectura contiene dos caminos. La primera ruta es la de contracción (también llamada codificador) que se utiliza para capturar el contexto en la imagen. El codificador es solo una pila tradicional de capas convolucionales y de agrupación máxima. La segunda ruta es la ruta de expansión simétrica (también llamada decodificador) que se utiliza para permitir la localización precisa mediante convoluciones transpuestas. Por lo tanto, es una red totalmente convolucional de extremo a extremo (FCN) (6), es decir, solo contiene capas convolucionales y no contiene ninguna capa densa debido a la cual puede aceptar imágenes de cualquier tamaño. En nuestro proyecto se trabaja con segmentación, localización y detección de objetos con resultados muy eficaces 3.24.

- Las operaciones de convolución y agrupación reducen la muestra de la imagen, es decir, convierten una imagen de alta resolución en una imagen de baja resolución.
- La operación de Max Pooling ayuda a comprender 'QUÉ' hay en la imagen al aumentar el campo receptivo. Sin embargo, tiende a perder la información de 'DÓNDE' están los objetos.
- En la segmentación semántica, no solo es importante saber 'QUÉ' está presente en la imagen, sino que también es importante saber 'DÓNDE' está presente. Por lo tanto, necesitamos una forma de aumentar la muestra de la imagen de baja resolución a alta resolución que nos ayudará a restaurar la información de 'DÓNDE'.

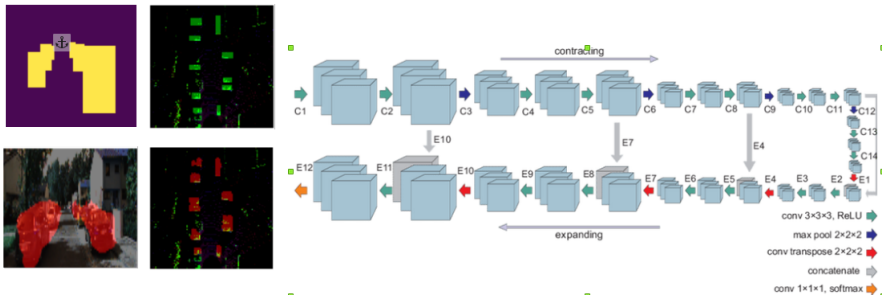


Figura 3.24: La arquitectura de red neuronal convolucional profunda U-Net muestra a su entrada las mascaras iniciales e imágenes reales de la escena, con este conjunto es entrenada la red neuronal con el objetivo de predecir los objetivos con el menor error posible, como se puede visualizar.

- La convolución transpuesta es la opción más preferida para realizar el muestreo, que básicamente aprende los parámetros a través de la propagación inversa para convertir una imagen de baja resolución en una imagen de alta resolución.

Para llevar a cabo el entrenamiento de la red neuronal U-Net, se necesitan los siguientes elementos:

La red neuronal U-Net requiere de los directorios y configuración de las rutas de los archivos de entrenamiento y prueba.

Se procesan los datos para obtener el archivo train.csv, el cual contiene la ruta de todas las imágenes del lote de entrenamiento con las respectivas coordenadas transformadas de los objetos etiquetados.

El conjunto de datos de validación se toma un lote del conjunto de entrenamiento, esto por medio de código se hace una pequeña subdivisión de los datos generados para ser procesados por la red neuronal.

Prueba y resultados de la red neuronal U-Net se toma un subconjunto de los datos del entrenamiento. Para generar un archivo test.csv y generar el archivo con los resultados de las detecciones finales.

Configuración de parámetros para la compilación correcta la red neuronal U-Net: La arquitectura de la red neuronal convolucional profunda U-Net recibe a la entrada la información aumentada con la ayuda del preprocesamiento y generación de los datos de entrenamiento y validación.

El código correspondiente a la arquitectura de la red neuronal U-Net contiene las capas convolucionales, las capas de reducción, las capas residuales y las capas de clasificación. También contiene los parámetros necesarios para el entrenamiento:

- En la clase Keras Model, hay tres métodos, fitgenerator, evaluate generator y predict generator. Los tres requieren un generador de datos, pero no todos los generadores se crean por igual.

- Fit generator, requiere dos generadores, uno para los datos de entrenamiento y otro para la validación. Ambos devuelven una tupla (entradas, objetivos) y ambos pueden ser una instancia de la clase Sequence.
- Evaluate generator, el generador de datos aquí tiene los mismos requisitos que en fit generator y puede ser el mismo que el generador de entrenamiento.
- Predict generator, el generador es un poco diferente. Debe devolver solo entradas. Con eso en mente, se construyen algunos generadores de datos, debido a la similitud entre el generador en fit generator y evaluate generator.
- Batch, tamaño del lote de entrenamiento para actualizar los parámetros internos de la red.
- Subdivisions, el número de subdivisiones que tendrá el parámetro batch para nuestro caso 16. Se utiliza para no agotar los recursos de la tarjeta de video.
- Width y height, el tamaño de la imagen de entrada 480x480.
- Channels, el número de canales de la imagen de entrada tres canales para la imagen real y un canal para la imagen con las máscaras.
- Learning rate, la taza de aprendizaje lr=1e-4.
- Optimizer, Adam.
- Max batches, número máximo de lotes que procesa la red neuronal en el proceso de entrenamiento 32.
- Metrics, se pueden evaluar los modelos de segmentación semántica usando la clase IOU.
- Implementación de Callbacks de Keras, disminución de la tasa de aprendizaje si la pérdida de validación no mejora durante 5 épocas continuas, parada temprana si la pérdida de validación no mejora durante 10 épocas continuas, guardado de los pesos solo si hay una mejora en la pérdida de validación.

En la siguiente figura 3.25 se muestra el diagrama de flujo del entrenamiento (12).

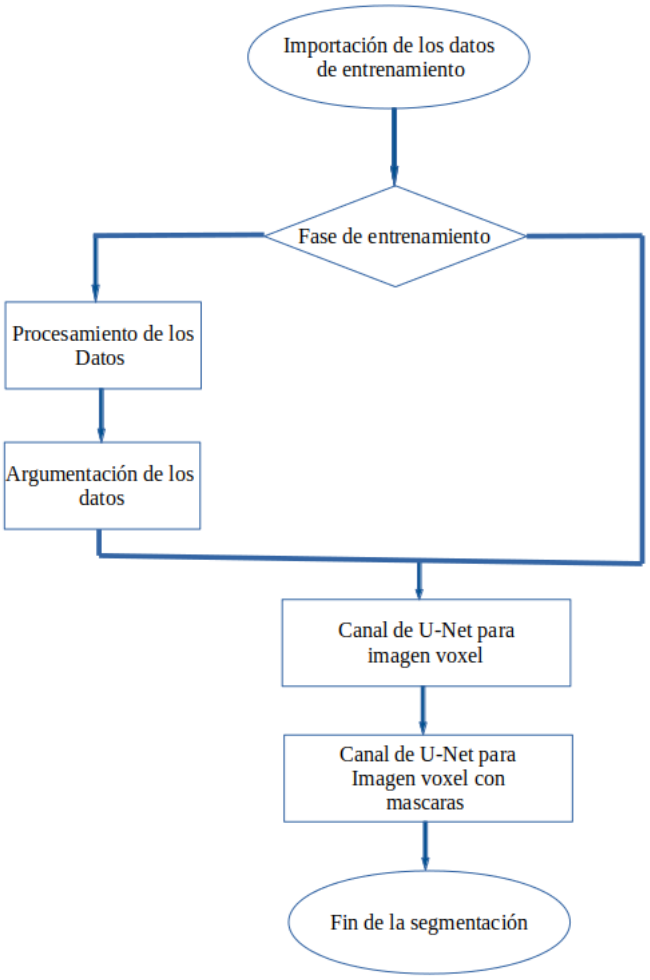


Figura 3.25: Entrenamiento de la red neuronal convolucional profunda U-Net.

Capítulo 4

Análisis de Resultados

En este capítulo se muestran los resultados de los experimentos, los cuales se agruparon en 3 secciones: La primera sección, muestra una introducción a la problemática actual en la detección de objetos en autos con el Hardware instalado para alcanzar la autonomía, enfatizando ¿como el sensor LIDAR podría ayudar a evitar este tipo de accidentes?. En la segunda sección, se presenta una metodología basada en la solución del proyecto propuesto. Esta metodología tiene fases las cuales ya fueron desarrolladas en el Capitulo anterior, por ende solo se presentan las fases por desarrollar. En la tercera sección se muestran las fases restantes de la metodología, las cuales en resumen presentan los resultados de segmentación y detección de objetos, las métricas de medición usadas para evaluar los modelos de Machine Learning.

4.1. Análisis de Resultados

Accidente de un Tesla como se muestra en la figura 4.1 en el que el vehículo se estrella directamente contra la parte superior de un camión. Con el análisis de los resultados de este proyecto se podrá brindar respuesta a los siguientes cuestionamientos y desarrollar las conclusiones finales del proyecto.

1. ¿Por qué el piloto automático de Tesla no percibe con todas sus cámaras y radares un obstáculo tan grande en la carretera como éste?

   Todos los sensores que componen el sistema de percepción envían datos al sistema de control del automóvil o la computadora para ayudarlo a tomar decisiones sobre dónde girar o cuándo frenar. Un automóvil totalmente autónomo necesita un conjunto de sensores que detecten con precisión los objetos, la distancia, la velocidad, etc., en todas las condiciones y entornos, sin que un humano tenga que intervenir. Tesla usa la recolección de millones de imágenes para el entrenamiento de sus redes neuronales, para que estas sean capaces de adquirir la experiencia necesaria para percibir al mundo en diferentes eventos, entonces proponemos que el sistema de percepción no fue capaz de reconocer la escena descrita en su momento como un obstáculo, debido

Figura 4.1: Accidente de un tesla con el equipamiento de un auto autónomo, el planteamiento de nuestro problema en usar o no el sensor LIDAR como solución para evitar este tipo de eventos fatales.

a la falta de experiencia de las redes neuronales de Tesla. La segunda opción es que el piloto automático nunca estuvo activado; lo cual las investigaciones correspondientes lo confirmaran.

2. ¿Los mapas o el LIDAR habrían evitado este accidente?

   La clave es LIDAR. Estos sensores usan láseres para construir un mapa preciso y detallado del mundo alrededor del automóvil y pueden distinguir fácilmente entre un rin y un automóvil de policía. El problema es que, en comparación con el radar, LiDAR es una tecnología joven. Sigue siendo muy caro y no es lo suficientemente robusto para sobrevivir a una vida de baches y recibir lluvia y nieve. Casi todo el mundo que trabaja en un sistema de conducción completamente autónomo, planea usar LIDAR, junto con radares y cámaras.

3. Aprendizaje con imágenes de las cámaras es incompleto y los datos LIDAR pueden ayudar al sistema ser eficiente?

   En este proyecto se hace un análisis de los datos de los sensores LIDAR, los cuales nos brindan una visión en los 360 grados con un solo sensor a diferencia de las cámaras. El sensor LIDAR brinda buenos resultados y la propuesta es usarlo en conjunto con las cámaras para desarrollar un sistema de visión sofisticado y completo. Por ejemplo si las cámaras no detectan algún objeto, se cuenta con el soporte del sensor LIDAR como seguridad. Los grandes retos de hoy en día es reducir el costo y contar en escala con el Hardware capaz de realizar el procesamiento de los datos de los sensores LIDAR en conjunto con los demás sensores que componen el sistema de percepción.

4. Como trabajan las redes neuronales de los autos de Tesla y como se aplica al Proyecto propuesto?

   Las redes neuronales son capaces por las cámaras analizar imágenes sin procesar para realizar segmentación semántica, detección de objetos y estimación de profundidad

monocular. Con perspectiva de vista de pájaro toman video de todas las cámaras para mostrar el diseño de la carretera, la infraestructura estática y los objetos 3D directamente en la vista de arriba hacia abajo. Las redes aprenden de los escenarios más complicados y diversos del mundo, obtenidos de manera iterativa de su flota de casi 1 millón de vehículos en tiempo real. Una compilación completa de redes neuronales piloto automático involucra 48 redes que requieren 70,000 horas de GPU para entrenarse. Juntos, generan 1000 tensores distintos (predicciones) en cada paso de tiempo. El Proyecto cumple con las características de las redes neuronales de Tesla, con el modelo de red neuronal U-net se realiza segmentación semántica, detección y localización de los objetos sobre la imagen, estimación profunda solo aplica para este trabajo para los sensores LIDAR por la característica esencial de este sensor.

4.1.1. Metodología de los Resultados

Con el fin de resolver el problema de la detección de las clases descritas anteriormente en imágenes de cámaras e imágenes tipo voxel de los sensores LIDAR, se propone la metodología mostrada en la figura 4.2. Esto permite segmentar y detectar los objetos etiquetados en las imágenes. Algunas fases de la metodología propuesta se han implementado con éxito anteriormente en el desarrollo del proyecto por consiguiente solo se desarrollan las fases faltantes. También como se ha hecho mucho hincapié se implementan dos redes neuronales convolucionales U-net para la obtención de resultados.

4.2. Generación de Bases de datos de entrenamiento y prueba

Entrenamiento de la red neuronal U-net con el conjunto de datos KITTI, es un conjunto de datos disponible con datos de sensores LIDAR e imágenes de las cámaras. El conjunto de datos nos proporciona las etiquetas de los objetos de las clases disponibles en las imágenes. Para realizar la transformación de las etiquetas de la nube de puntos de los sensores LIDAR se utilizan los parámetros de calibración de los sensores para su conversión de coordenadas 3D a 2D sobre la imagen tipo voxel que generada a partir del algoritmo Bird eye view.

Todas las imágenes de las cámaras se proporcionan como secuencias png rectificadas y comprimidas sin pérdidas. La resolución de la imagen nativa es de 1382 x 512 píxeles y un poco más baja después de la rectificación.

La elaboración de base de datos para el entrenamiento de la red neuronal U-net, fue con la clases Car, con un total de instancias igual a Car = 28614. Este conjunto de datos se encuentra des-balanceado en su total de instancias por clase. De acuerdo a la descripción del conjunto de datos KITTI.

Las dimensiones de las imágenes de las cámaras es de (375, 1242, 3) y de la imagen tipo voxel es de (480, 480, 3).

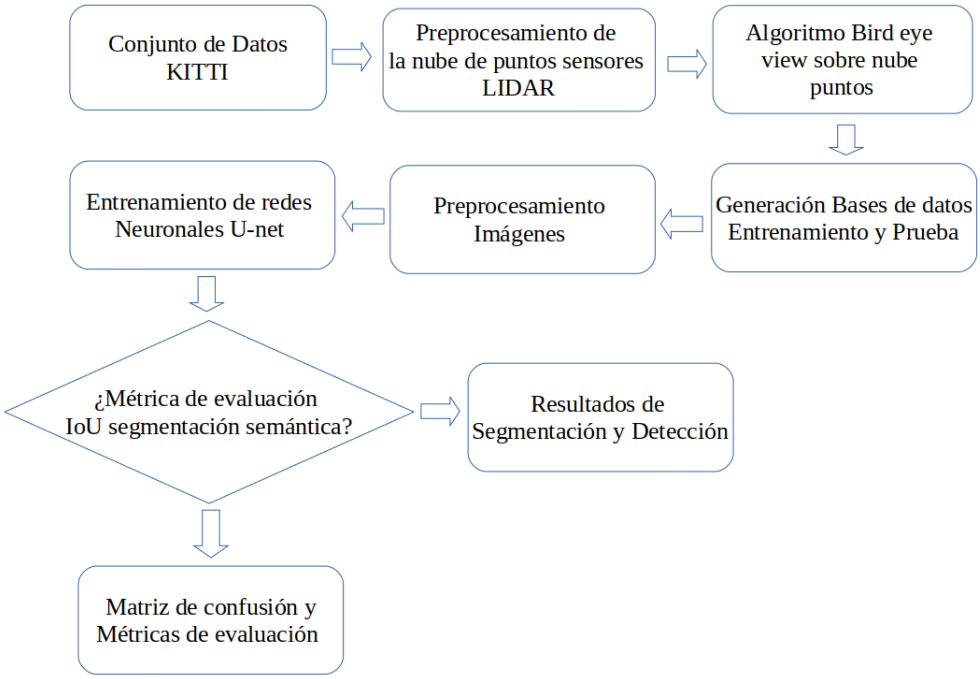


Figura 4.2: La metodología propuesta describe las fases del proceso general para la solución del problema del proyecto.

División de los datos en el entrenamiento, validación y prueba, para el proyecto el conjunto de datos KITTI viene dividido en entrenamiento y prueba, para el caso de la validación estamos usando el 15 por ciento de los datos del conjunto de entrenamiento. Sin embargo para el conjunto de Prueba los datos no vienen con los parámetros de calibración de los sensores y las etiquetas correspondientes para poder realizar la transformación de la nube de puntos a una imagen voxel para la Red U-net con datos LIDAR. El conjunto de entrenamiento está compuesto por 7381 imágenes y 100 imágenes para la prueba. Es por este motivo la distribución de las bases de datos para este proyecto ver Tabla 4.1.

La proyección a vista de pájaro está limitada por la resolución del sensor LIDAR, en este trabajo se toman las siguientes dimensiones de rango lateral -25m a 25m y rango de avance 0 a 40m, es decir, para los objetos en un instante de tiempo incluidos con el La imagen y las etiquetas de los objetos más alejados a esa distancia no caen dentro del rango del sensor LIDAR. Si la proyección a vista de pájaro se genera con mayores dimensiones, se generan máscaras de objetos que para no salir de la información de la nube de puntos de los sensores LIDAR, lo que afecta el periodo de aprendizaje de la red neuronal profunda generando sobreajustes y por tanto malas predicciones. Base de datos de entrenamiento y prueba para datos de sensores LIDAR ver Tabla 4.2.

Base de datos Cámara	Número de clases	Número de imágenes	Número de instancias
Entrenamiento	1	7381	28460
Prueba	1	100	154

Tabla 4.1: Bases de datos Entrenamiento y Prueba para la clase Car con imágenes de las cámaras.

Base de datos LIDAR	Número de clases	Número de imágenes	Número de instancias
Entrenamiento 1	1	7381	22687
Prueba 1	1	100	129

Tabla 4.2: Bases de datos Entrenamiento y Prueba para la clase Car con datos de los sensores LIDAR.

4.3. Métricas de evaluación IoU Segmentación semántica

Se mide el rendimiento durante el proceso de entrenamiento de las redes neuronales convolucionles U-net para determinar la combinación de características que brinden los mejores resultados de segmentación y detección. Esta fase consiste en aplicar métricas de evaluación como: Índice Jaccard (13). Para ello, se llevan a cabo varios experimentos con las bases de datos descritas anteriormente.

La intersección sobre la Unión (Índice Jaccard) es una medida de la magnitud de la superposición entre dos cuadros delimitadores (o, en el caso más general, dos objetos). Calcule el tamaño de la superposición entre dos objetos, dividido por el área total de los dos objetos combinados Fig. 4.3).

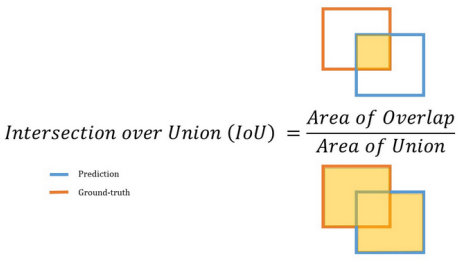


Figura 4.3: Intersección sobre la Unión

Estas métricas de evaluación 4.4 muestran el entrenamiento de la clase Car. Las dos graficas representan el entrenamiento por separado de los dos modelos de red neuronal U-net, una entrenada con imágenes de las cámaras y otra entrenada con imágenes tipo voxel provenientes de los sensores LIDAR.

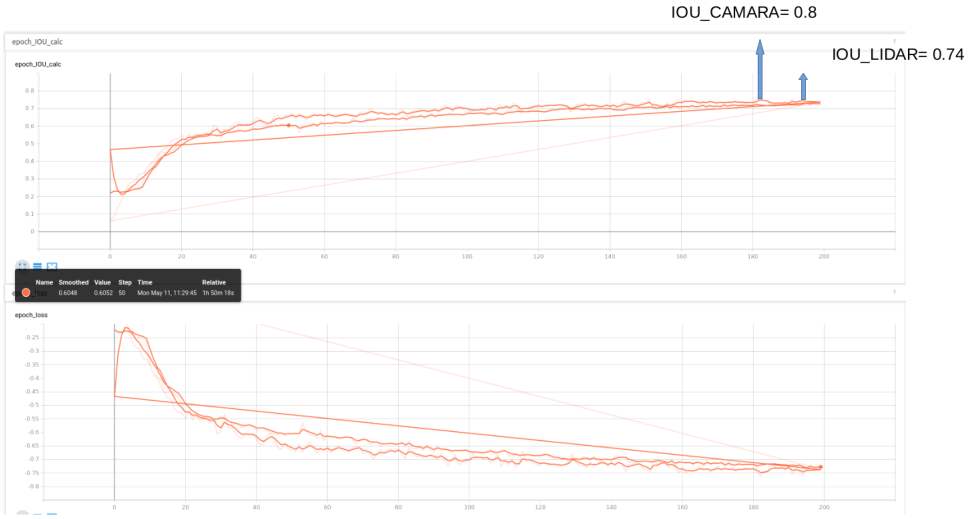


Figura 4.4: La grafica muestra el entrenamiento de las dos redes neuronales (Cámara vs LIDAR), los valores máximos correspondientes para cada red neuronal se ubican en el punto sobre el grafico, el error es inversamente proporcional al valor de la métrica de evaluación de segmentación semántica para este tipo de tareas.

Tiempos de predicción de la clase Car de la red U-net con 100 muestras aleatorias con imágenes de las cámaras 4.3.

Número de imágenes de las cámaras con red neuronal U-net	**Tiempo de predicción**
100.	1.40 s.

Tabla 4.3: Tiempos de predicción clase Car

Tiempos de predicción de Clase Car de la red U-net con 100 muestras aleatorias con imágenes voxel de los sensores LIDAR 4.4.

Número de imágenes voxel con red neuronal U-net	**Tiempo de predicción**
100.	0.34 s.

Tabla 4.4: Tiempos de predicción

4.3.1. Predicciónes de la clase Car con Cámara

La Figura 4.5 muestra en la primera fila la imagen original elegida de manera aleatoria por el generador sobre el conjunto de datos, la segunda columna muestra la predicción de la segmentación de los objetos dependiendo la clase disponible en la base de datos, la tercera columna muestra los objetivos correspondientes a las clases a predecir.

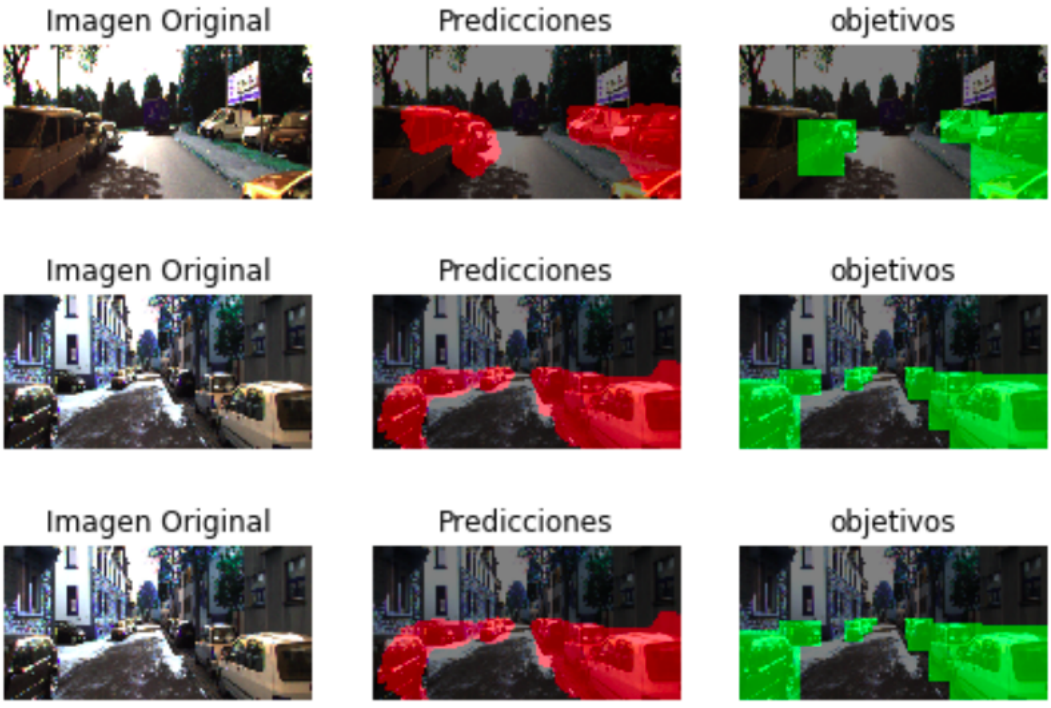


Figura 4.5: Resultados de predicción de la red neuronal entrenada con imágenes de las cámaras.

4.3.2. Predicciónes de la clase Car con LIDAR

El mismo procedimiento se aplica para la red neuronal entrenada con las imágenes tipo voxel y se grafica sobre la misma proyección para realizar la comparativa correspondiente en el entrenamiento 4.6. En la primera columna se observa la imagen voxel con vista de pájaro, la segunda columna muestra las predicciones de segmentación de las clases disponibles en nuestra base de datos de entrenamiento y la ultima columna muestra los objetivos totales.

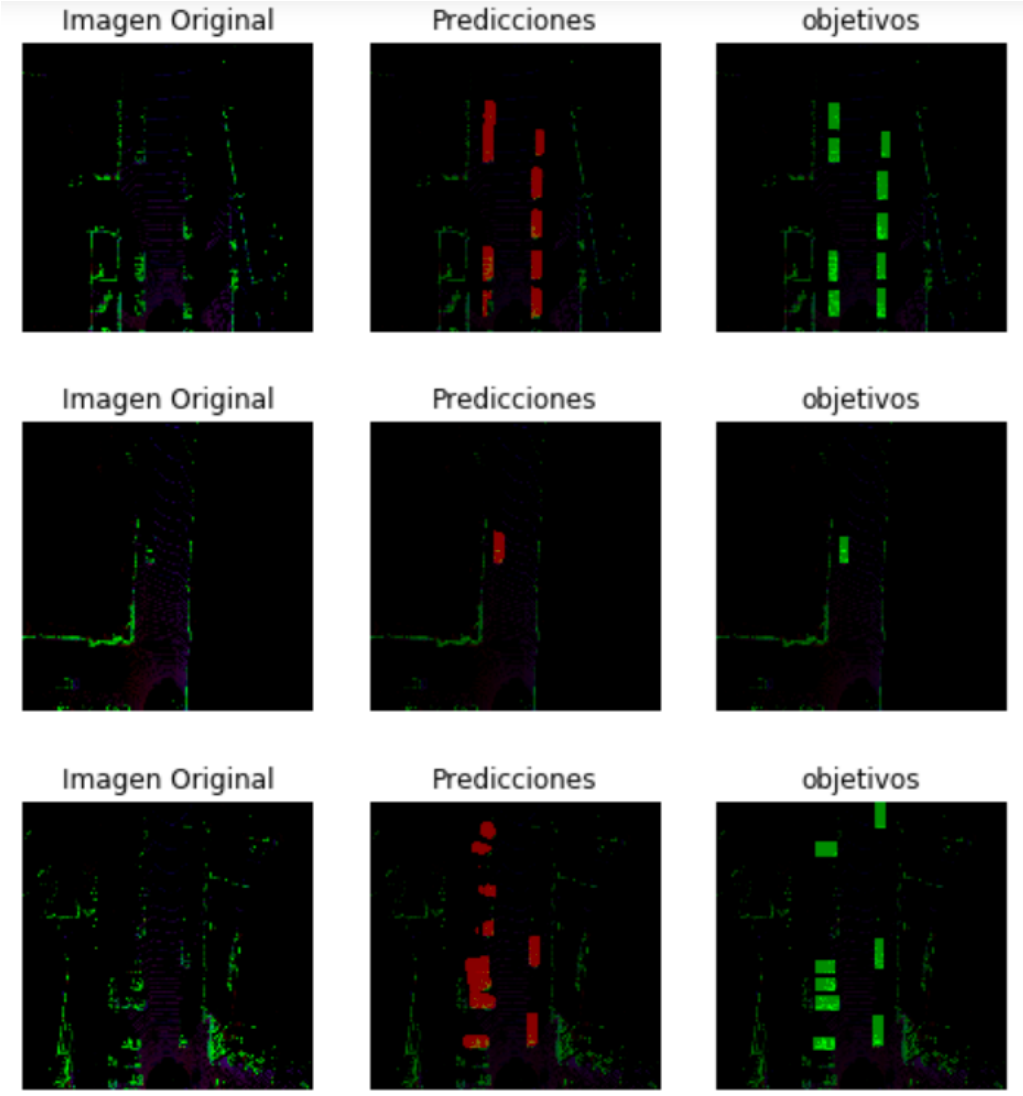


Figura 4.6: Resultados de predicción de la red neuronal entrenada con imágenes voxel de los sensores LIDAR.

4.4. Resultados de segmentación y detección

De acuerdo al diseño de la metodología, se han implementado las fases necesarias para obtener los resultados de segmentación y detección en imágenes de las cámaras Fig. 4.7 e imágenes a partir de los datos de los sensores LIDAR Fig. 4.8 en esta fase se realizan los experimentos con éxito con la clase de Car que tiene el mayor número de etiquetas y por lo tanto presenta equilibrio con respecto al resto de las clases disponibles en el conjunto de datos KITTI.

Los procesos descritos en la metodología se han ejecutado en una computadora con tarjeta gráfica GeFORCE GTX 1050 y 24 GB de RAM necesarios para el proceso de generación de las imágenes a vista de pájaro de la nube de puntos de los sensores LIDAR; esto conlleva más tiempo en comparación con las imágenes de las cámaras. Esta fase de la metodología fue en un inicio de gran interés: trabajar con otros tipos de datos de sensores para entrenar una red neuronal. Los resultados presentan cuatro imágenes por fila de la siguiente manera: La primera imagen representa la imagen de prueba original; la segunda imagen muestra la predicción de la segmentación del objeto por la red neuronal U-net; la tercera imagen muestra los objetivos totales a predecir; En definitiva, la cuarta imagen con un cuadro delimitador representa la detección de la segmentación predicha por la red neuronal con un umbral mayor o igual a 0,5. Los objetos evaluados con la métrica de segmentación de IoU que se evaluaron correctamente para una clase en particular están delimitados por un rectángulo que representa la ubicación del objeto dentro de la escena. Se indica el nombre de la muestra; tiene un área delimitada con una confianza, xmin, xmax, ymin, ymax y el nombre de la clase "Car".

Una métrica buena pero simple que siempre debe usarse cuando se trata de un problema de clasificación es la matriz de confusión. Esta métrica ofrece una descripción general interesante de qué tan bien está funcionando un modelo. Por lo tanto, es un excelente punto de partida para la evaluación de cualquier modelo de clasificación

4.5. Matriz de confusión y Métricas de evaluación

Hemos utilizado tres métricas, precisión, recuperación y medida F1, que evalúa el equilibrio entre la precisión y la recuperación.

$$precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{4.1}$$

$$recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{4.2}$$

$$F1measure = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.3}$$

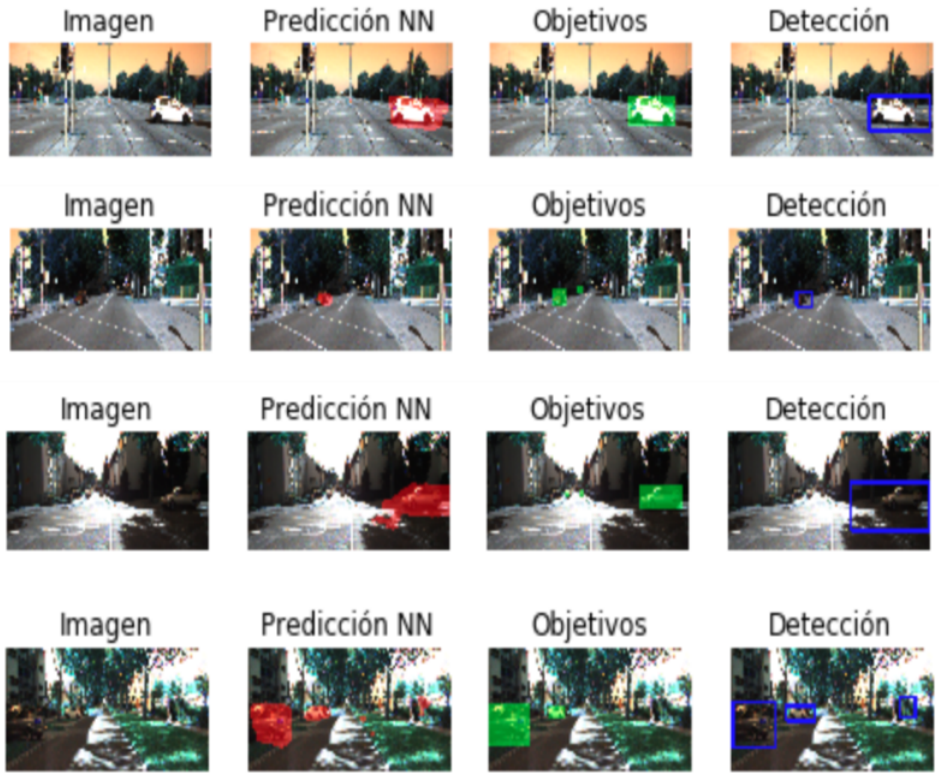


Figura 4.7: Resultados de detección de la red neuronal entrenada con imágenes de las cámaras.

A partir de la matriz de confusión, se calculan las métricas de evaluación tabla 4.9 correspondientes a las predicciones de las dos redes neuronales profundas. La siguiente tabla 8 muestra las métricas de evaluación con respecto a las predicciones realizadas por redes neuronales profundas por separado con el mismo conjunto de datos de entrenamiento y prueba. Es importante mencionar que en este punto la única clase que se está evaluando es Çar", como se describe es la clase con mayor número de instancias, a diferencia de las otras clases. disponible en el conjunto de datos.

La métrica F1 muestra un mejor resultado con los datos de los sensores LIDAR, brevemente esto ocurre por el pre-procesamiento de la nube de puntos, el cual ayuda a reducir demasiado ruido y puntos que rodean a los objetos de interés. Lo cual no pasa con las imágenes de las camaras, debido a la gran variedad de colores y formas que forman parte del ambiente visual de la imagen. Los resultados que muestra el LIDAR son buenos e igual al de las imágenes sin embargo se pueden mejorar con otro tipo de arquitectura de red neuronal por ejemplo que realice solo la detección.

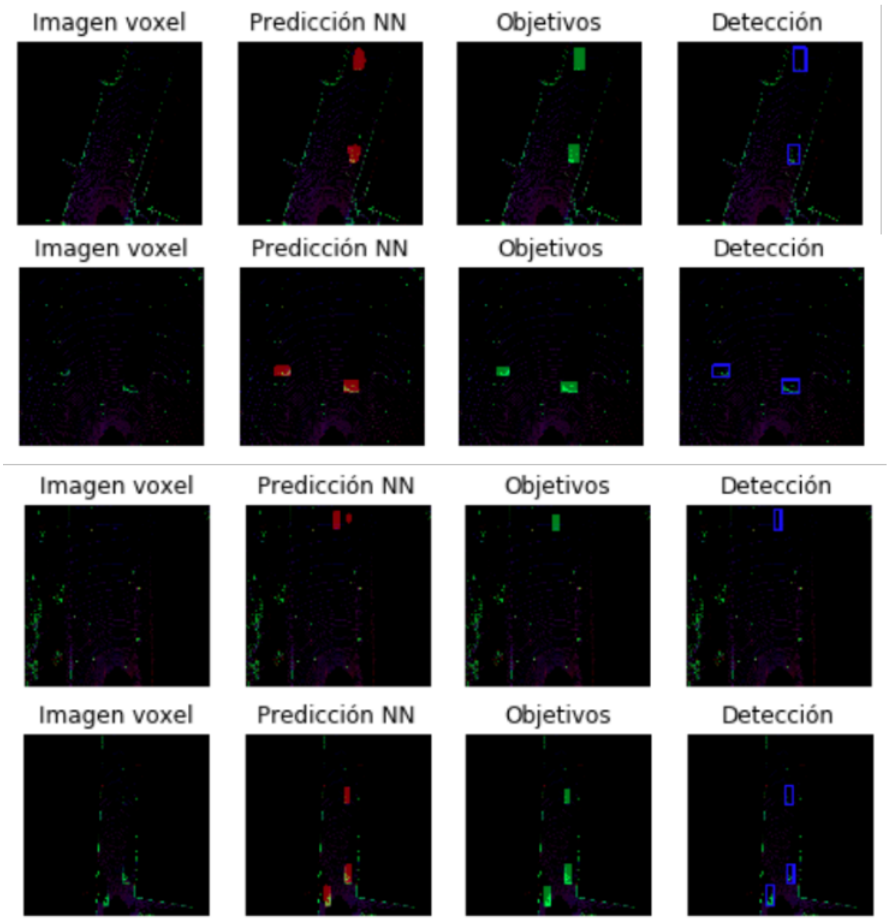


Figura 4.8: Resultados de detección de la red neuronal entrenada con imágenes voxel de los sensores LIDAR.

Database	# TP	# FN	# TN	# FP	Precision	Recall	F1 measure
Cámara	113	47	1	4	96.50%	70.60%	81.50%
LIDAR	105	17	1	19	84.00%	86.00%	85.00%

Figura 4.9: Métricas de evaluación

Capítulo 5

Conclusiones

Con base a los resultados obtenidos en el capitulo 4, se tienen las siguientes conclusiones en relación con los dos modelos de red neuronal profunda entrenadas con imágenes de las cámaras y con datos de los sensores LIDAR.

- Los resultados de la red U-net entrenada con datos de los sensores LIDAR, su eficiencia de predicción es optima como la de las imágenes, sumado las ventajas que ofrece el sensor LIDAR como, la estimación profunda, localización y mapeo es posible con este trabajo obtener buenos resultados.
- Los tiempos de predicción para el proceso de Prueba la red entrenada con datos LIDAR, responde con mayor velocidad a comparación con la red de imágenes, este dato es sumamente importante en respuesta en la detección de objetos en un auto autónomo.
- La estimación de profundidad y creación de mapas de los sensores LIDAR ayuda a detectar objetos, para evitar accidentes como los que han ocurrido con los autos de la marca Tesla, concluimos con este trabajo que el sensor LIDAR es esencial su instrumentación en los autos autónomos.
- Los resultados de segmentación y detección con LIDAR y cámara, pueden mejorar con la implementación en diferentes arquitecturas de red neuronal profunda.
- Asimismo, el desarrollo de sensores LIDAR con señales de mejor resolución y calidad contribuirá directamente a una mejor detección de objetos en el futuro y tendrá un precio más accesible; un factor que es más deseado por las empresas de automovilisticas.

5.0.1. Trabajo Futuro

Como trabajo futuro, se plantean las siguientes mejoras al proyecto propuesto:

- Recientes publicaciones estan usando cámaras monoculares para el área de visión por computadora con la finalidad de generar estimación profunda con este tipo de imágenes digitales, este trabajo se venia realizando con imágenes provenientes de cámaras tipo estéreo que requieren un procesamiento costoso computacional similar al de los sensores LIDAR.
- Se propone actualizar la metodología propuesta probando con diferentes arquitecturas de redes neuronales profundas y utilizando visión estéreo o monocular con estimación profunda vs LIDAR. Se propone analizar la posibilidad de equipar un automóvil con sensor LIDAR y cámaras para generar un conjunto de datos propio en las carreteras de México.

Bibliografía

[1] Asvadi, A., Premebida, C., Peixoto, P., and Nunes, U. (2016). 3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes. *Robotics and Autonomous Systems*, 83:299–311. 20

[2] Chadwick, S., Maddetn, W., and Newman, P. (2019). Distant vehicle detection using radar and vision. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8311–8317. IEEE. 2

[3] Chen, X., Ma, H., Wan, J., Li, B., and Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1907–1915. 21, 39

[4] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*. 17

[5] Chuang, Y.-Y. (2005). Camera calibration. 22, 23

[6] Dai, J., Li, Y., He, K., and Sun, J. (2016). R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387. 52

[7] Deakin, R. E. (1998). 3-d coordinate transformations. *Surveying and land information systems*, 58(4):223–234. 41

[8] Du, S., Guo, H., and Simpson, A. (2019). Self-driving car steering angle prediction based on image recognition. *arXiv preprint arXiv:1912.05440*. 43

[9] Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Glaeser, C., Timm, F., Wiesbeck, W., and Dietmayer, K. (2020). Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*. 3, 21

[10] Gao, H., Cheng, B., Wang, J., Li, K., Zhao, J., and Li, D. (2018). Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. *IEEE Transactions on Industrial Informatics*, 14(9):4224–4231. 18

[11] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE. 31

[12] Gulli, A. and Pal, S. (2017). *Deep learning with Keras*. Packt Publishing Ltd. 54

[13] Gunawardana, A. and Shani, G. (2009). A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, 10(12). 61

[14] Hamid Rezatofighi, S., Milan, A., Zhang, Z., Shi, Q., Dick, A., and Reid, I. (2015). Joint probabilistic data association revisited. In *Proceedings of the IEEE international conference on computer vision*, pages 3047–3055. 38

[15] Herzog, M. and Dietmayer, K. (2019). Training a fast object detector for lidar range images using labeled data from sensors with higher resolution. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2707–2713. IEEE. 2, 37

[16] James, J. K., Puhlfürst, G., Golyanik, V., and Stricker, D. (2016). Classification of lidar sensor contaminations with deep neural networks. 3

[17] Lin, C.-C. and Wang, M.-S. (2012). A vision based top-view transformation model for a vehicle parking assistant. *Sensors*, 12(4):4431–4446. 45

[18] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440. 26

[19] Murugan, P. (2017). Feed forward and backward run in deep convolution neural network. *arXiv preprint arXiv:1711.03278*. 14, 16

[20] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528. 28, 47

[21] Ren, M. and Zemel, R. S. (2017). End-to-end instance segmentation with recurrent attention. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6656–6664. 29

[22] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer. 52

[23] Rusinkiewicz, S. and Levoy, M. (2001). Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE. 19, 33

[24] Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314. 50

[25] Traore, B. B., Kamsu-Foguem, B., and Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, 48:257–268. 7

[26] Van Dyk, D. A. and Meng, X.-L. (2001). The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50. 49

[27] Wang, Y., Chao, W.-L., Garg, D., Hariharan, B., Campbell, M., and Weinberger, K. Q. (2019). Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8445–8453. 3, 4, 5

[28] Wen, T. and Zhang, Z. (2018). Deep convolution neural network and autoencoders-based unsupervised feature learning of eeg signals. *IEEE Access*, 6:25399–25410. 10

[29] Wu, S. and Hong, L. (2005). Hand tracking in a natural conversational environment by the interacting multiple model and probabilistic data association (imm-pda) algorithm. *Pattern Recognition*, 38(11):2143–2158. 38

[30] Yoo, H.-J. (2015). Deep convolution neural networks in computer vision: a review. *IEIE Transactions on Smart Processing & Computing*, 4(1):35–43. 8