ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Detección e identificación de obstáculos a partir de una nube de puntos

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación
Mención: Telemática
Autor: Fernando Santana Falcón
Tutores: D. David de la Cruz Sánchez Rodríguez
Dª. Itziar G. Alonso González

Fecha: Septiembre 2021

ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

Detección e identificación de obstáculos a partir de una nube de puntos

Calificación: _____

Presidente:

Vocal: **Secretario:**

Fecha: Septiembre 2021

Agradecimientos

Quisiera agradecer a las personas que me han rodeado durante estos años cuyo apoyo ha sido esencial para conseguir la culminación de estos estudios. Con los que se cierra una etapa muy enriquecedora en mi vida.

Especialmente a mis abuelos por haberme "adoptado" y apoyado durante este camino. A mis padres, por haber sentado las bases de lo que soy como persona, por apoyarme en mis decisiones, por su lucha diaria para darnos lo mejor, por ser los mejores y mi gran orgullo. A mi hermana por ser mi motivo de superación. Al resto de mi familia como son mis tías y mis primas, tan importantes en mi vida.

A mis compañeros, sin los que no podría haber conseguido este objetivo, por haberme apoyado tanto en la biblioteca como fuera de ella, especialmente a los que puedo considerar amigos.

Para finalizar, agradecer a mis tutores Dª. Itziar G. Alonso González y D. David de la Cruz Sánchez Rodríguez, cuyo apoyo ha sido esencial para completar este trabajo y a la universidad por haberme permitido vivir múltiples experiencias con la cercanía que caracteriza a la comunidad de la EITE.

Resumen

Describir una escena del mundo real, tal y como la percibe el ser humano no es una tarea fácil, requiere de algunos factores importantes como son el conocimiento y la capacidad de reconocimiento.

Una situación que a las personas nos parece tan trivial como distinguir entre diferentes objetos, es más complicado de realizar mediante un ordenador. Los computadores no poseen la capacidad que tiene nuestro cerebro de unir los puntos que pertenecen al mismo objeto y descartar aquellos que corresponden al entorno.

En la robótica actual se están realizando grandes avances en el ámbito de la inteligencia artificial (IA) y la adquisición de imágenes. Dotar a los robots de la capacidad de interpretar el entorno y reconocer los objetos que componen la escena ha llevado al desarrollo de aplicaciones como conducción autónoma, detección de caídas y detección de intrusos, entre otras.

El actual auge de los sensores LIDAR provocado por su desarrollo, ha causado una revolución en el sector. Con aplicaciones en múltiples industrias, por ejemplo, agricultura, sanidad o transporte y haciendo uso de distintos dispositivos desde drones hasta smartphones. Esta tecnología ha posibilitado el desarrollo de nuevos enfoques en el ámbito de la visión por computador.

Siguiendo esta línea de trabajo, este trabajo de fin de grado se centra en la detección e identificación de objetos en nubes de puntos con un sistema de bajo costo computacional.

Abstract

Describing real-world scenes, as perceived by human beings, is not an easily accomplished task, since it requires some important factors such as perception and the ability to recognize external factors.

Something that seems trivial to humans, like distinguishing between different objects, becomes more complex when using a computer. Computers do not have the same ability as our brain has to connect points that belong to the same object, while discarding those that correspond to the environment.

In today's robotics, great advances are being made in the field of Artificial Intelligence and image acquisition. Providing robots, the ability to interpret the environment and recognize several objects in a given scene has led the development of applications such as autonomous driving, fall detection or intrusion detection.

The current boom in LIDAR sensors, triggered by their massive growth, has caused a revolution in the sector. With applications in multiple industries, such as agriculture, health and transport, these sensors have been manifested in different devices, from drones to smartphones. This technology has enabled the development of new approaches in the field of computer vision.

Following this line of research, this project focuses on object detection and identification from point clouds, using a low-cost computational system.

Índice de contenidos

Índice de figuras

Índice de tablas

Listado de siglas y acrónimos

2D	2 Dimensiones
3D	3 Dimensiones
API	Application Programming Interface (Interfaz de programación de aplicaciones)
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange (Código Estándar estadounidense para el Intercambio de Información)
BIM	Building Information Modelling (Modelado de información de construcción)
CAD	Computer Aided Design (Diseño Asistido por Ordenador)
CAM	Computer Aided manufacturing (Fabricación asistida por computadora)
CNN	Convolutional Neural Networks (Red Neuronal Convolucional)
CPU	Central Processing Unit (Unidad Central de Procesamiento)
CUDA	Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo)
DBSCAN	Density-based spatial clustering of applications with noise (agrupamiento espacial basado en densidad de aplicaciones con ruido)
EITE	Escuela de Ingeniería de Telecomunicación y Electrónica
FP	False Positive (Positivos Falsos)
FN	False Negative (Negativos Falsos)
GPU	Graphics Processing Unit (unidad de procesamiento gráfico)
GFLOPS	Operaciones de coma flotante por segundo
HTML	HyperText Markup Language
IGIC	Impuesto General Indirecto Canario
IA	Inteligencia Artificial
IDE	Integrated Development Environment (entorno de desarrollo integrado)
IEEE	Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos)
Kd	k-dimensiones
kNN	k-Nearest Neighbors (k vecinos más cercanos)
LIDAR	Light Detection and Ranging o Laser Imaging Detection and Ranging
MB	MegaBytes
MLP	Multilayer Perceptron (Perceptron multicapa)
MSER	Maximally Stable Extremal Regions
OpenCV	Open Computer Vision Library
PDF	Portable Document Format
PMD	Photonic Mixer Devices
RAM	Random Access Memory (memoria de acceso aleatorio)
RANSAC	Random Sample Consensus
ReLU	Rectified Linear Unit (Rectificador)
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
RMSProp	Root Mean Square Propagation (propagación de la raíz cuadrática media)
RNA	Redes Neuronales Artificiales
ROS	Robot Operating System
SBC	Single Board Computer
SD	Secure Digital

SDK	Software Development Kit (kit de desarrollo de software)
SGD	Stochastic Gradient Descent (descenso de gradiente estocástico)
SIFT	Scale-Invariant Feature Transform
TFG	Trabajo Fin de Grado
TN	True Negative (Negativos Verdaderos)
T-NET	Transformation Network
ToF	Time of Flight
TP	True Positive (Positivos Verdaderos)
USB	Universal Serial Bus

I MEMORIA

Capítulo 1 – Introducción

1.1 Antecedentes

En la última década, los algoritmos, métodos y sistemas de percepción basados en información tridimensional han desarrollado un importante auge revolucionando la creación de aplicaciones robóticas, tales como el guiado, localización y navegación de robots o como la detección y reconocimiento de objetos para la interacción con robots [1].

Este avance ha estado provocado, principalmente, por dos factores: el desarrollo (tanto espacial como computacional) de nuevos sensores capaces de adquirir información tridimensional en tiempo real a un bajo coste, y el progreso que ha sufrido el hardware destinado al procesado de estos datos. Por consiguiente, en los últimos años la cantidad de enfoques y métodos para realizar estas tareas con información tridimensional han aumentado. Estas mejoras se encuentran presentes en una gran variedad de tareas tales como la reconstrucción de escenas 3D, extracción de características, segmentación o las nuevas técnicas de clasificación.

En el Trabajo de Fin de Grado (TFG) "Evaluación de cámaras de tiempo de vuelo para la definición de entornos de interior" [2], se expone un análisis de un sistema de adquisición de datos tridimensionales a través de sensores y computadores de bajo coste. Se parte de este proyecto para realizar un sistema de bajo costo capaz de adquirir, procesar y reconocer objetos en escenas tridimensionales utilizando una de las herramientas más poderosas de la visión artificial, el aprendizaje profundo, conocido también como deep learning.

1.2 Objetivo

El objetivo de este proyecto es la implementación de una solución de bajo coste para la identificación de objetos en interiores basada en un sensor que adquiere una nube de puntos. Este objetivo general se desglosa en los siguientes objetivos específicos según se recoge en el anteproyecto:

O1: Análisis de las soluciones de identificación de objetos existentes en la actualidad.
O2: Estudio de la librería OpenCV.
O3: Parametrización y extracción de características.
O4: Programación de una aplicación para la identificación de objetos.

Tras estudiar la librería OpenCV [3], [4], se decidió no utilizarla en virtud de los mejores resultados obtenidos por la librería Open3D [5] para el tratamiento de nubes de puntos tal y como veremos en el Capítulo 3 – Tecnologías utilizadas.

1.3 Estructura del documento

El presente documento consta de cuatro apartados, el primero de ellos, la **Memoria** contiene seis capítulos cuyo contenido se describe a continuación:

- **Capítulo 1, Introducción**: consta de un breve resumen de los antecedentes en los que se basa el proyecto y los objetivos a cumplir.
- **Capítulo 2, Conceptos teóricos y estado del arte**: desarrolla de forma introductoria los conceptos teóricos presentes en el sistema para concluir con las investigaciones recientes en el reconocimiento de objetos en nubes de puntos.
- **Capítulo 3, Tecnologías utilizadas**: describe los elementos hardware y herramientas software utilizados en la elaboración del trabajo, así como sus características y la justificación de su uso.
- **Capítulo 4, Algoritmos y dataset**: introduce las bases de datos creadas y explica el funcionamiento de los algoritmos realizados para cada fase del sistema.
- **Capítulo 5, Entrenamiento y clasificación**: explica el proceso de entrenamiento y clasificación que realiza el sistema para finalizar evaluando los resultados obtenidos.
- **Capítulo 6, Conclusiones y líneas futuras**: se exponen las conclusiones extraídas de este proyecto y las posibilidades futuras que ofrece.

Para finalizar la memoria, se presenta la **Bibliografía** donde se recogen las referencias utilizadas en la elaboración del TFG con formato IEEE.

Tras la memoria, se encuentran los apartados: **Presupuesto** donde se desglosan los costes pertenecientes a la elaboración de este proyecto; los **Anexos** 1, 2 y 3 aportan información adicional del proyecto, incluyen una guía de configuración del hardware y puesta a punto del sistema, el código desarrollado y la estructura de los conjuntos de datos.

Capítulo 2 – Conceptos teóricos y estado del arte

2.1 Visión Artificial

La visión artificial es un campo de la Inteligencia Artificial (IA) que, mediante la utilización de las técnicas adecuadas, permite la obtención, procesamiento y análisis de cualquier tipo de información especial obtenida a través de imágenes digitales. [6]

Los seres humanos percibimos el mundo que nos rodea de forma tridimensional. Somos capaces de captar un objeto y sus propiedades en el espacio gracias a la visión. El objetivo principal de la visión artificial o visión por computadora es dotar a los ordenadores de la capacidad de comprender una imagen al igual que lo hace el ser humano.

La visión artificial la componen un conjunto de procesos destinados a realizar el análisis de imágenes. Estos procesos son: captación de imágenes, memorización de la información, procesado e interpretación de los resultados. Las actividades principales de la visión artificial se pueden clasificar en:

- **Procesamiento de imágenes digitales**: tiene como objetivo la descripción y reconocimiento del contenido de una imagen digital.
- **Visión computacional**: pretende dotar a los ordenadores de la capacidad de poder simular la visión humana.

Diariamente se generan millones de imágenes y vídeos, esta cantidad ha ido aumentando año tras año con el avance tecnológico. Para las computadoras, una imagen es sólo un conjunto de números que por sí mismo no tienen ningún significado. La necesidad de controlar y analizar estos datos hace que el procesamiento de imágenes sea una investigación en constante evolución. Según la Universidad de Cambridge [7][8], ver Figura 1, la visión artificial se especializa en tres estrategias para analizar estos datos:

- **Reconocimiento**: consiste en saber donde están los objetos en una imagen e identificar qué son.
- **Reconstrucción**: es una técnica que proporciona forma tridimensional a los elementos de una imagen.
- **Registro**: consiste en la alineación de modelos. Compara los elementos de una imagen con modelos o patrones ya definidos en busca de coincidencias.

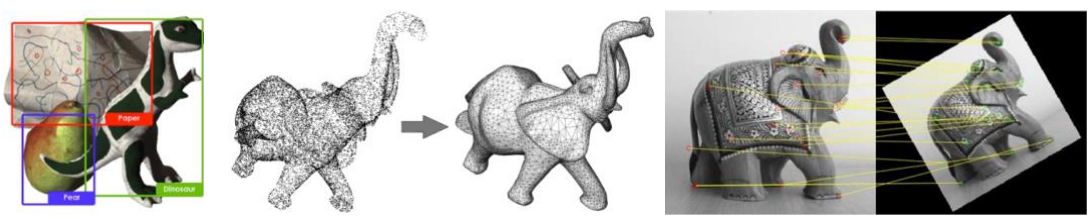


Figura 1: Tareas de reconocimiento, reconstrucción y registro sobre nubes de puntos [7][8]

Desde su inclusión en la industria, la visión artificial ha ido desarrollando sus técnicas para capturar figuras tridimensionales y la apariencia de los objetos a partir de imágenes. Estos avances han permitido que hoy en día podamos realizar tareas como son: el reconocimiento facial, la conducción autónoma, el seguimiento de personas o la recreación de objetos 3D con imágenes bidimensionales.

La versatilidad y fácil implementación de esta tecnología, ha favorecido su asentamiento en múltiples sectores, siendo la industria donde se puede encontrar un mayor número de aplicaciones. Las principales aplicaciones de la visión artificial en la industria actual son:

- Identificación e inspección de objetos.
- Determinación de la posición de los objetos en el espacio.
- Establecimiento de relaciones espaciales entre varios objetos (guiado de robots)
- Determinación de las coordenadas de un objeto.
- Realización de mediciones angulares y tridimensionales.

En [9] se observan los múltiples sectores que abarca la visión por computador, sus variadas aplicaciones y la gran importancia de estas.

Como hemos comentado anteriormente, el principal objetivo de la visión por computador es tener la capacidad de analizar una imagen, sin embargo, existen factores que provocan que esta tarea sea compleja. Para ello, en los sistemas de visión artificial la información es descompuesta en distintos niveles de visión para reducir su complejidad [10]:

- **Nivel bajo:** trata con la imagen digitalizada (píxeles o puntos). Se extraen características de la imagen como son el color, textura, gradiente, etcétera.
- **Nivel intermedio:** agrupa la información aportada por el nivel bajo para obtener bordes, líneas, regiones, etcétera. Su finalidad es poder segmentar la imagen.
- **Nivel alto**: se encarga de interpretar los datos obtenidos en los niveles anteriores utilizando modelos del problema.

Pese a que cada sistema de visión artificial tenga una forma distinta de tratar los datos, la adquisición y procesamiento, siguen un proceso similar basado en los niveles vistos anteriormente. La base del sistema es la fase de capturación de imágenes y su respectivo procesado.

La imagen es captada por un sensor, cámara o similar para ser digitalizada y facilitar su manipulación por un ordenador. El procesado de la imagen o imágenes obtenidas pertenece a la parte software del sistema. Se encarga de dotar de lógica al computador mediante algoritmos para transformarla en datos de alto nivel, para ello es necesario un preprocesado para mejorar la calidad de la imagen, como puede ser la eliminación de ruido. Por consiguiente, la estructura general de los sistemas de visión artificial consta de las siguientes fases; ver Figura 2.

1) **Adquisición de datos**: se capturan las imágenes y se digitalizan.
2) **Preprocesado**: se mejora la calidad de las imágenes obtenidas (eliminación de ruido, adaptación al sistema, enriquecimiento de detalles…).
3) **Segmentación**: divide la imagen en áreas de interés.
4) **Representación y descripción**: busca y extrae los patrones de la imagen.
5) **Reconocimiento**: identifica los objetos de la imagen.
6) **Interpretación**: asigna un significado a un conjunto de objetos reconocidos.

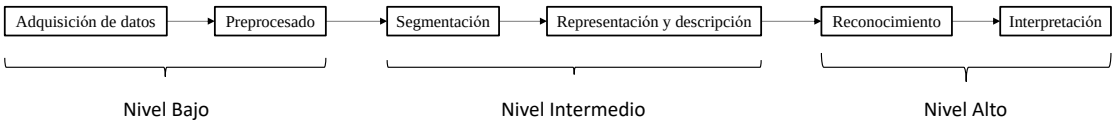


Figura 2: Esquema de niveles y etapas de un sistema de visión artificial

Normalmente, los sistemas de visión artificial tratan con grandes cantidades de datos y los procesos requeridos para tratar las imágenes requieren de grandes costos computacionales. Con los continuos avances tecnológicos, estos problemas se han ido superando.

2.2 Reconocimiento de objetos

El reconocimiento de objetos es en la actualidad el mayor desafío de la visión artificial, por lo que en las últimas décadas se han realizado grandes esfuerzos en esta tarea, especialmente, en las escenas tridimensionales por su aplicación en el entorno real [11].

En la visión artificial, un objeto es una entidad del mundo real con una etiqueta única. El significado de ese objeto depende del nivel de abstracción que se le otorga en el contexto de la aplicación. Szeliski [11] además del reconocimiento facial, declara 4 tipos de reconocimientos; ver Figura 3.

- **Detección de objetos**: consiste en detectar un objeto conocido que se encuentra inmóvil. Se realiza un barrido de la escena para determinar si se produce alguna correspondencia con el objeto a detectar.
- **Reconocimiento de instancias**: se encarga de reconocer un objeto rígido (en 2D o 3D) conocido que se encuentra situado en un fondo con oclusiones o desordenado. Se realiza mediante correspondencia sin depender de las propiedades de la imagen o escena.
- **Reconocimiento de categorías**: es utilizado para reconocer instancias de una clase general.
- **Comprensión de contexto y escena**: afronta el reconocimiento del contexto de la escena donde se encuentra el objeto.

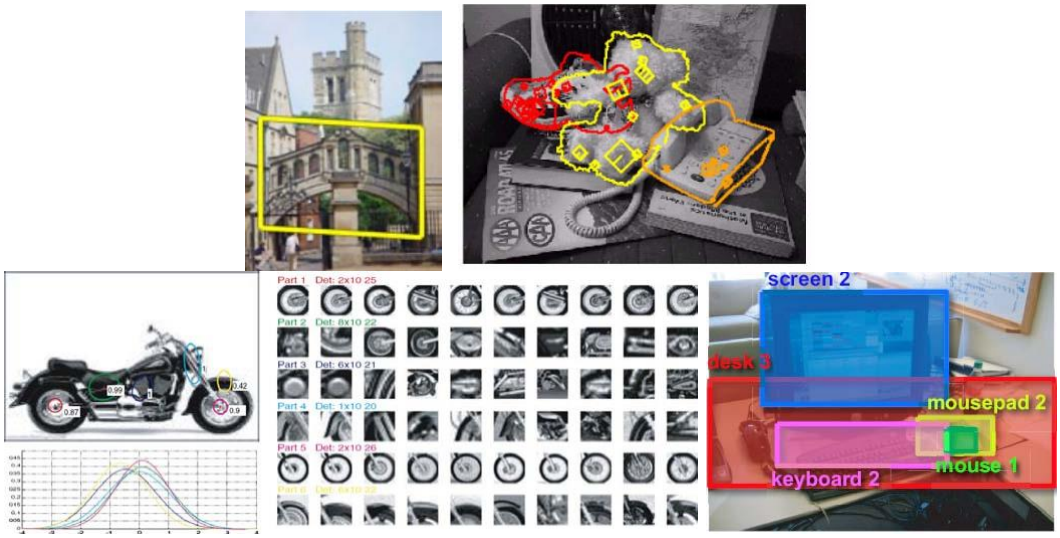


Figura 3: Tareas de detección, reconocimiento de instancias, reconocimiento de categorías y comprensión de contexto y escena sobre imágenes [11]

De estas definiciones se concluye que la detección de objetos se encarga de identificar la localización de un objeto en la escena mientras que el reconocimiento clasifica un objeto presente en una región de la imagen previamente segmentada, en la Figura 4 podemos observar ambas tareas.

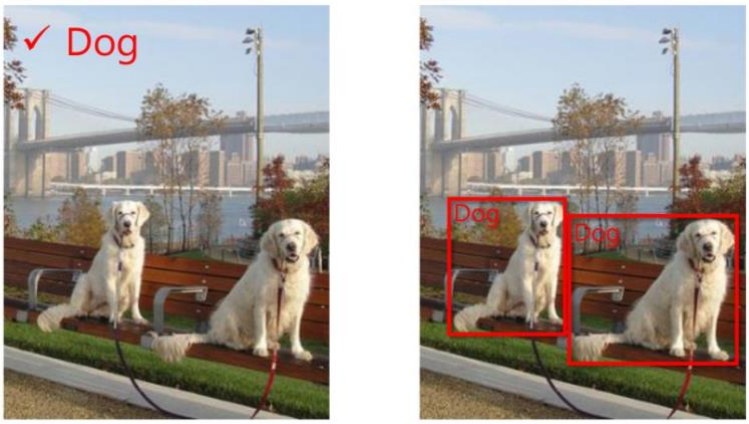


Figura 4: Clasificación vs detección [11]

Según la metodología utilizada en el proceso de reconocimiento podemos clasificar los métodos como:

2.2.1 Basado en modelos

Estos métodos, consideran que el conocimiento de la apariencia de un objeto proviene del modelado explícito de su forma sin considerar el contexto en el que se encuentra.

Su reconocimiento se realiza mediante la búsqueda de correspondencia entre ciertas características de la imagen y las características comparables del modelo. Por ello, el problema a resolver por este tipo de métodos es distinguir qué es considerado característica y cómo se puede usar en la comparación de características con el modelo.

Las características se pueden clasificar entre locales o globales, su uso depende de la naturaleza de la imagen:

- **Características globales**: utilizadas en métodos donde se puede realizar una segmentación perfecta entre objetos y fondo. Por ejemplo, el área, el perímetro, etcétera.
- **Características locales**: de mayor complejidad, como son las intersecciones o las esquinas.

La secuencia de pasos a seguir por la mayoría de los métodos es:

- **Detección de primitivas**: localizar las primitivas y representarlas como símbolos.
- **Organización perceptual**: identificar agrupaciones estables de primitivas.
- **Indexación**: seleccionar los modelos más similares de la base de datos mediante las primitivas.
- **Emparejamiento**: realizar la correspondencia entre las primitivas de la imagen y los modelos seleccionados.
- **Verificación**: decidir si el resultado del emparejamiento es correcto.

Este proceso de reconocimiento es muy costoso computacionalmente dado que cada vez que se reconoce un objeto, sus primitivas son eliminadas de la imagen convirtiéndose en una búsqueda constante a la hora de reconocer otro objeto [12]. En la Figura 5 se describe un proceso de reconocimiento facial basado en modelos.

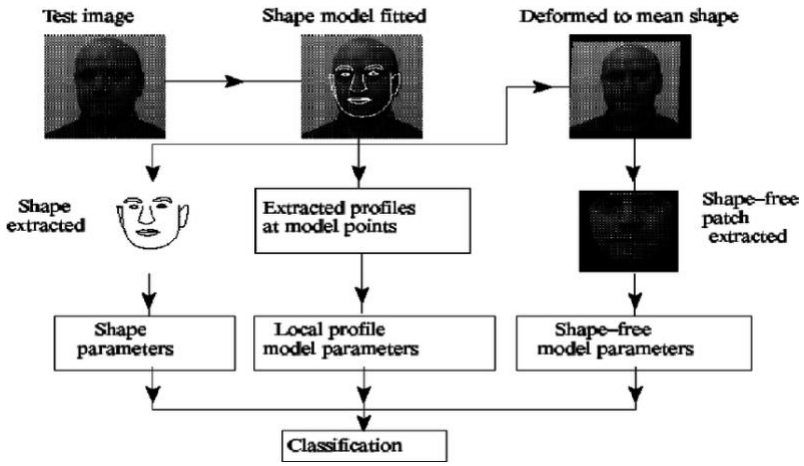


Figura 5: Método de reconocimiento basado en modelo [12]

2.2.2 Basado en apariencias

En este tipo de métodos el objeto de interés es capturado mediante diferentes vistas bidimensionales. Según el tipo de características que se utilice, podemos distinguir entre modelos de enfoque global o enfoque local.

Las características globales tratan la información de la imagen en su totalidad, permiten reconstruir la imagen original ofreciendo robustez a ampliaciones. Estas técnicas pueden ser desde histogramas de características hasta medidas estadísticas.

Por otro lado, las características locales son propiedades situadas en un solo punto de la imagen o una pequeña región. Aportan información distintiva del objeto como puede ser el color, valor de gris de un píxel o el gradiente. La gran ventaja de este tipo de características es que permite trabajar con objetos parcialmente ocluidos.

Los sistemas basados en características locales, en primer lugar, deben detectar las regiones de interés de la imagen para poder compararlos con las regiones de interés de los objetos conocidos. Cuando esta región de interés esta constituida por un único punto, este punto es denominado punto de interés. Los detectores de regiones se pueden clasificar en tres grupos:

- **Basados en regiones**: se encargan de detectar zonas con brillo uniforme. MSER [13] es uno de los métodos más usados.
- **Basados en esquinas**: buscan las zonas de transición de la imagen a través de derivadas, con ello obtienen el gradiente que permite encontrar las zonas de mayor contraste. Uno de los métodos más populares es el detector de esquinas Harris [14].
- **Otros**: que toman como referencia otras características como puede ser la entropía de las regiones.

Tras detectar la región de interés, es necesario representarla mediante descriptores de características que sean robustos a variaciones de la imagen como ruido o cambios de iluminación. Los descriptores pueden ser:

- **Basados en distribuciones**: representan las propiedades de la región, normalmente de carácter geométrico, mediante histogramas. El descriptor más destacado es SIFT [15], ver Figura 6, encargado de detectar regiones mediante la diferencia de gaussianas.
- **Basados en filtros**: usan operadores diferenciales locales con la finalidad de obtener invariancia a la rotación. Uno de los métodos más populares es [16].
- **Otros**: donde destaca la correlación cruzada, encargada de describir las intensidades de los píxeles de la región mediante una media estadística [17].

Tras extraer los descriptores de la región es necesario una etapa final de clasificación que permita el reconocimiento final del objeto. Existen numerosos tipos de clasificadores, el más básico es el de K-vecinos más cercanos, encargado de asignar al objeto la clase que predomina entre sus vecinos.

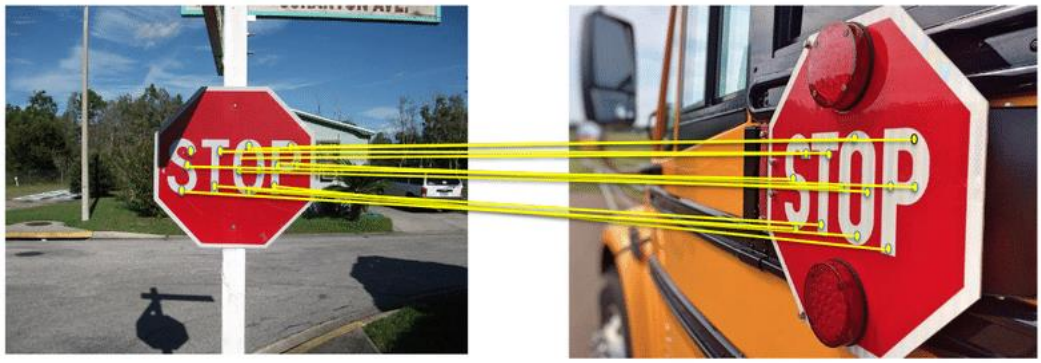


Figura 6: Método de reconocimiento, SIFT, basado en apariencias [15]

El desarrollo del aprendizaje automático, también conocido como machine learning, ha permitido mejorar las tareas de clasificación. Podemos realizar esta labor con dos distintos métodos de clasificación:

- **Clasificación supervisada**: el aprendizaje supervisado, aprende una función a partir de ejemplos de sus entradas y salidas. Utiliza un gran conjunto de datos de aprendizaje con el que se entrena el sistema para asociar patrones a clases.
- **Clasificación no supervisada**: el aprendizaje no supervisado, aprende a partir de ejemplos de entrada para los que no se especifica su valor de salida. Usa cálculos estadísticos para aprender a clasificar la información en lugar de usar conjuntos de datos.

2.2.3 Basado en Deep Learning

Las investigaciones relacionadas con la tarea del reconocimiento han conseguido remplazar los descriptores de características manuales. Como hemos visto en los apartados anteriores su creación requiere de experiencia en el dominio de la aplicación. En sustitución, se ha conseguido crear redes multicapas capaces de aprenderlos automáticamente mediante un algoritmo de entrenamiento.

Esta solución fue propuesta por [18] en los años 70 provocando la creación de una nueva rama dentro del aprendizaje automático, el Deep Learning. En el apartado 2.3 Deep Learning, se explica esta técnica y sus aplicaciones.

Como podemos observar, el reconocimiento de objetos ha ido evolucionado, los estudios comenzaron centrados en la búsqueda de objetos en imágenes bidimensionales. Sin embargo, el boom del Deep Learning ha favorecido el tratamiento de escenas tridimensionales provocando una gran revolución por su similitud con el mundo real.
El proceso de reconocimiento de objetos en escenas tridimensionales requiere de las siguientes etapas tal y como aparece en la Figura 7:

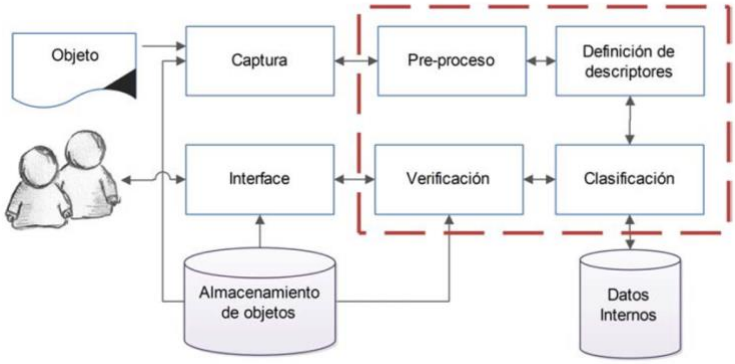


Figura 7: Esquema de un sistema de reconocimiento de objetos tridimensionales [19]

1) **Adquisición**: la captura de datos tridimensionales se realiza escaneando o muestreando una superficie con tecnología láser. Como resultado proporciona una gran cantidad de puntos en un corto periodo de tiempo sobre la apariencia de la escena. Dado que los dispositivos láser son instrumentos de línea de vista, es necesario realizar múltiples capturas desde diferentes perfiles para poder obtener una cobertura completa de la escena.

2) **Preprocesado**: se encarga de mejorar y adaptar la escena obtenida. Modifica las propiedades de la nube de puntos obtenida en la fase anterior para su posterior análisis. Su función es destacar la información relevante y eliminar el ruido.
3) **Definición de descriptores**: para reconocer el objeto, poder representarlo matemáticamente y simplificar la información esencial. Esta fase se realiza mediante deep learning.
4) **Clasificación**: según las características del objeto, se le atribuye una clase [20]. Los datos brutos son clasificados en información útil.
5) **Verificación**: comprueba los resultados obtenidos por el clasificador.
6) **Interfaz**: dedicada a mostrar al usuario los resultados.

2.2.4 Adquisición de datos tridimensionales

La adquisición de imágenes tridimensionales y el procesado de datos en tiempo real son las principales limitaciones de la visión artificial hoy en día. Desde la invención de la cámara, la información visual se ha proyectado en imágenes bidimensionales. Sin embargo, este tipo de imágenes no contienen información de profundidad, esencial para identificar los objetos.

En los últimos años, gracias al desarrollo tecnológico, hemos conseguido adquirir los datos de profundidad de una escena mediante sensores 3D. Esta tecnología se encuentra disponible en el mercado en dispositivos como son Microsoft Kinect, Intel RealSense o smartphones de alta gama como es el caso de la cámara LIDAR de los teléfonos de Apple, presente en la Figura 8.

Figura 8: Cámara con sensor LIDAR en el iPhone 12 pro [21]

Sensores 3D

Existen distintos tipos de sensores, dependiendo del tipo de sensor usado, la imagen presentará distintos niveles de resolución, exactitud y ruido [22]. Según su método de adquisición de datos, los sensores de imagen tridimensionales pueden ser:

- **Contacto:** examinan la superficie con un elemento denominado palpador. El palpador, se apoya sobre el objeto a escanear por lo que sólo es válido para obtener

datos sobre un único objeto y no sobre una escena. Al requerir contacto físico, puede modificar o dañar objetos sensibles.

- **Sin contacto:** son escáneres que no requieren contacto con el objeto.
- **Ópticos:** donde la luz porta la información de medición.
- **No ópticos:** acústicos (ultrasónico, sísmico), electromagnéticos (infrarrojo, ultravioleta, microondas, radar…).

Los sensores de imagen tridimensionales que no precisan de contacto se pueden clasificar en:

- **Activos**: obtienen la información del objeto emitiendo y recibiendo cierto tipo de radiación (luz, rayos x, ultrasonidos).
- **Pasivos**: no emiten ningún tipo de radiación, pero detectan la radiación ambiental. Son métodos bastante baratos por lo que suelen utilizarse en cámaras digitales convencionales.
- **Directos:** aportan un intervalo de datos entre la superficie desconocida y el sensor de rango.
- **Indirectos:** las mediciones se infieren a partir de imágenes monoculares y del conocimiento previo de las propiedades del objetivo.

Las técnicas de captación de datos tridimensionales suelen estar basadas en:

- **Triangulación** [23].
- **Retardo temporal** [22]**.**
- **Imágenes monoculares** [22]**.**

Los métodos de adquisición de datos tridimensionales sin contacto más destacados son:

- **Tiempo de vuelo:** calculan la distancia del objeto a partir del tiempo que tarda un pulso de radiación emitida de luz infrarroja, en ir y volver. Son los más antiguos, por lo tanto, los más conocidos para medir profundidad, sin embargo, su principal inconveniente es su alto precio.

- **Interferometría**: los métodos interferométricos operan proyectando un patrón periódico que varía espacial o temporalmente sobre una superficie, seguido de la mezcla de la luz reflejada con un patrón de referencia. El patrón de referencia demodula la señal para revelar la variación en la geometría de la superficie.

- **Triangulación láser:** los métodos de triangulación láser se basan en el principio de triangulación activa [23].

- **Luz estructurada:** comparte el enfoque de triangulación activa, sin embargo, en lugar de escanear la superficie, proyectan patrones bidimensionales de luz.

- **Estereográfico:** representa la versión pasiva de las técnicas de luz estructurada. Necesita dos o más cámaras previamente calibradas capturando simultáneamente la misma escena. Las ventajas notables del enfoque estéreo son la simplicidad y el bajo costo; el principal problema es la identificación de puntos comunes dentro de los pares de imágenes. Están basados en la visión humana.

- **Fotogrametría:** obtiene modelos 3D fiables mediante imágenes digitales. Se realiza un proceso de orientación de la cámara, mediciones de puntos de imagen, generación de nubes de puntos, superficies y mapeo de texturas. La calibración de la cámara es fundamental para obtener modelos precisos.

De acuerdo con lo comentado anteriormente, se pueden clasificar como:

	Triangulación	Retardo temporal	Pasivo	Activo	Directo	Indirecto
Tiempo de vuelo	X			X	X	
Interferometría	X			X	X	
Triangulación láser	X		X		X	
Luz estructurada	X		X		X	
Estereográfico		X		X	X	
Fotogrametría		X		X	X	

Tabla 1: Clasificación de los métodos de adquisición de datos tridimensionales [22]

Nubes de puntos

Los datos 3D son el resultado del escaneo de sensores de profundidad como LIDAR, sensores radar o cámaras RGB-D. Generalmente se pueden representar en diferentes formatos, siendo las nubes de puntos el formato más común, ver Figura 9. La ventaja de la captura tridimensional mediante el escaneado láser es que se puede obtener una gran cantidad de puntos con alta precisión en un periodo corto de tiempo.

La nube consta de puntos con vectores tridimensionales no estructurados. Cada punto es representado mediante un vector, que indica su coordenada 3D. Como formato de uso común, la representación de nubes de puntos conserva la información geométrica original en el espacio 3D sin ninguna discretización.

Supone la representación más simple de objetos 3D: sólo puntos en el espacio tridimensional, sin conectividad. Sin embargo, puede contener información adicional de cada punto como los componentes de color RGB, la intensidad y la componente normal. Los objetos descritos por nubes de puntos son capaces de representar desde milímetros hasta ciudades enteras.

Al proporcionar información detallada para objetos y entornos, la nube de puntos se usa ampliamente en diversas aplicaciones, como ingeniería inversa, preservación digital, topografía, conducción autónoma, arquitectura, juegos 3D, robótica y realidad virtual.

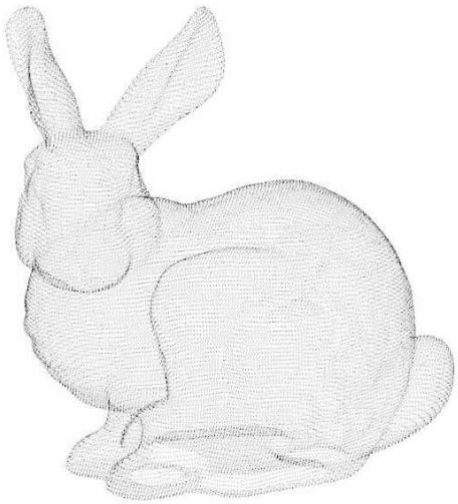


Figura 9: Nube de puntos [24]

Este formato ha aumentado su popularidad como resultado del aumento de dispositivos de capturación y su vital función en las áreas comentadas anteriormente. Existen distintos estándares de almacenamiento de nubes de puntos; ver Tabla 2.

Estándar	Estructura	Descripción
xyz	[x, y, z]	Donde x, y, z son las coordenadas geométricas de cada punto.
xyzn	[x, y, z, nx, ny, nz]	Donde x, y, z son las coordenadas geométricas de cada punto y nx, ny, nz las componentes normales.
xyzrgb	[x, y, z, r, g, b]	Donde x, y, z son las coordenadas geométricas de cada punto y r, g, b son los componentes de color de rango [0, 1].
pts	[x, y, z, i, r, g, b]	La primera línea contiene el numero de puntos de la nube. El resto de las líneas está en el formato indicado anteriormente donde i indica la intensidad del píxel y las componentes de color se encuentran en formato uint8.
ply	Estructura disponible en [25]	Polygon
pcd	Estructura disponible en [26]	Point Cloud Data

Tabla 2: Formatos comunes de las nubes de puntos

2.2.5 Procesado de datos tridimensionales

La imagen resultante de la adquisición de datos tridimensionales puede poseer gran cantidad de información según las características vistas anteriormente. Toda esta información puede no ser útil, la adquisición de datos indiferentes para el sistema provoca un mayor tamaño en el conjunto de datos a tratar, mayor posibilidad de error y un mayor retraso en los cálculos. Por ello, existen numerosos métodos de filtrado de datos de la imagen como puede ser el procesado de las características de la escena, filtrando según la distancia, densidad, geometría del objeto, etcétera. A continuación, se detallan dos métodos de procesado de datos utilizado en este TFG cuya función son la detección de planos y el agrupamiento de datos:

RANSAC (Random Sample Consensus)

Los sensores tridimensionales al trabajar en el mundo real no obtienen valores totalmente perfectos, se obtienen algunos valores indeseados. RANSAC es un algoritmo simple pero bastante efectivo con datos afectados por valores atípicos.

Este algoritmo es un método iterativo que se utiliza para estimar los parámetros de un modelo matemático a partir de un conjunto de datos que contienen valores denominados como atípicos u outliers. Fue publicado por Fischler y Bolles en 1981 [27]. El algoritmo asume que todos los datos de entrada están compuestos tanto de inliers como de outliers. Los valores inliers se pueden explicar mediante un modelo con un conjunto particular de valores de sus parámetros, mientras que los valores outliers no se ajustan a ese modelo en ninguna circunstancia, ver Figura 10.

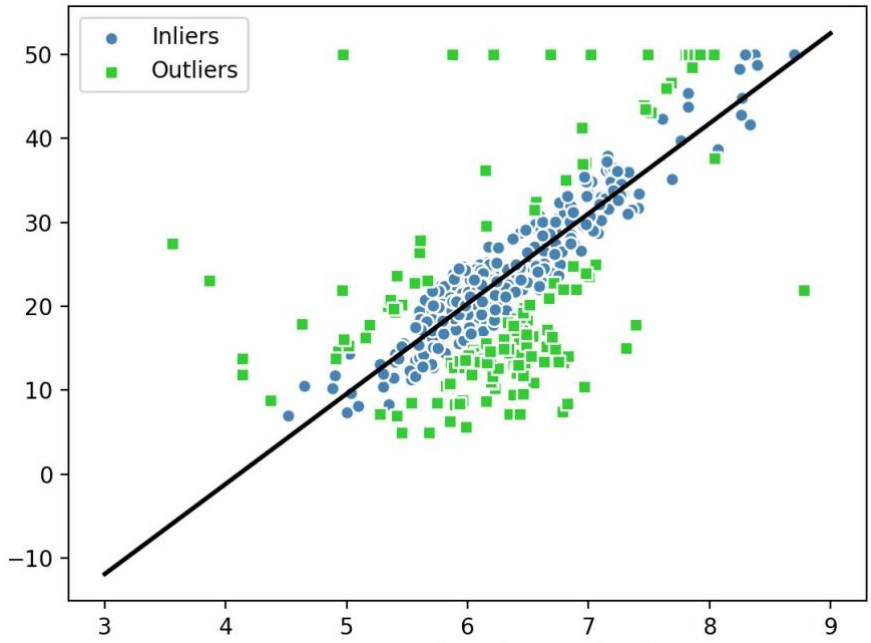


Figura 10: Valores inliers y atípicos (outliers) del método RANSAC [28]

La ventaja de RANSAC es su capacidad para hacer una estimación robusta de los parámetros del modelo, puede estimar los parámetros con un alto grado de precisión incluso cuando el conjunto de datos presenta un número significativo de valores atípicos.

Procedimiento

El procedimiento de este método a diferencia de los algoritmos de estimación tradicionales no utiliza la mayor cantidad de datos posibles para obtener una solución y eliminar los datos erróneos posteriormente. RANSAC utiliza el menor conjunto de datos que permita calcular una solución inicial, aumentando dicho conjunto con datos consistentes que se adapten al modelo obtenido.

El algoritmo primero crea un plano a partir de los datos, para ello, selecciona aleatoriamente 3 puntos de la nube necesarios para establecer un plano. Luego, verifica cuántos de los puntos restantes están situados en el plano (hasta un cierto umbral), lo que

le dará una puntuación a la propuesta. Este proceso es repetido con 3 nuevos puntos aleatorios (N iteraciones), al calcular un mayor número de iteraciones, aumenta la probabilidad de que se produzca un modelo razonable. Finalmente se selecciona el modelo con mayor puntuación. Obteniendo de resultado final: los puntos situados en el plano, más los tres puntos que hemos muestreado constituyendo nuestro conjunto de puntos inliers, y el resto es nuestro conjunto de puntos outliers.

En la Figura 11, una nube de puntos perteneciente a una cocina ha sido sometida a este algoritmo. El método tiene como objetivo identificar el mayor plano de la escena aplicando RANSAC, en este caso, el techo ha sido perfectamente identificado.

Figura 11: Plano predominante de la escena detectado mediante RANSAC [29]

El procedimiento no es determinista, pese a que con cada iteración la probabilidad de que los resultados sean correctos aumenta, nunca obtendrá resultados exactos. Cuando el número de iteraciones calculadas es limitado, la solución obtenida puede no ser óptima, y que ni siquiera se ajuste a los datos de una buena manera. Otra desventaja de RANSAC es que no tiene límite en el tiempo de cálculo de los parámetros, además de requerir del establecimiento de umbrales específicos para el problema.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN, cuya traducción en castellano sería agrupamiento espacial basado en densidad de aplicaciones con ruido, tras su invención en 1996 [30] se ha convertido en uno de los algoritmos de agrupamiento de datos más utilizados.

DBSCAN itera sobre los puntos del conjunto de datos. Para cada punto que analiza, construye el conjunto de puntos alcanzables por densidad desde este punto: calcula la vecindad de este punto, y si esta vecindad contiene más de una cierta cantidad de puntos, se incluye en la región. Cada uno de los puntos vecinos pasa por el mismo proceso hasta que ya no puede expandir el clúster. Si el punto considerado no es un punto

inlier, es decir, no tiene suficientes vecinos, se etiquetará como ruido, ver Figura 12. Esto permite que DBSCAN sea robusto ante valores outliers, ya que los aísla.

El algoritmo requiere de dos parámetros (ϵ para la vecindad y n_min para el número mínimo de puntos), cada parámetro tiene una influencia especifica en el algoritmo por lo que su elección es una tarea compleja. Se debe tener cuidado al configurar los parámetros para crear suficientes puntos interiores (lo que no sucederá si n_min es demasiado grande o ϵ demasiado pequeño). En particular, esto significa que DBSCAN tendrá problemas para encontrar grupos de diferentes densidades. Pero, por otro lado, DBSCAN posee la gran ventaja de ser computacionalmente eficiente sin necesidad de predefinir el número de clústeres, a diferencia de otro método muy popular como es Kmeans.

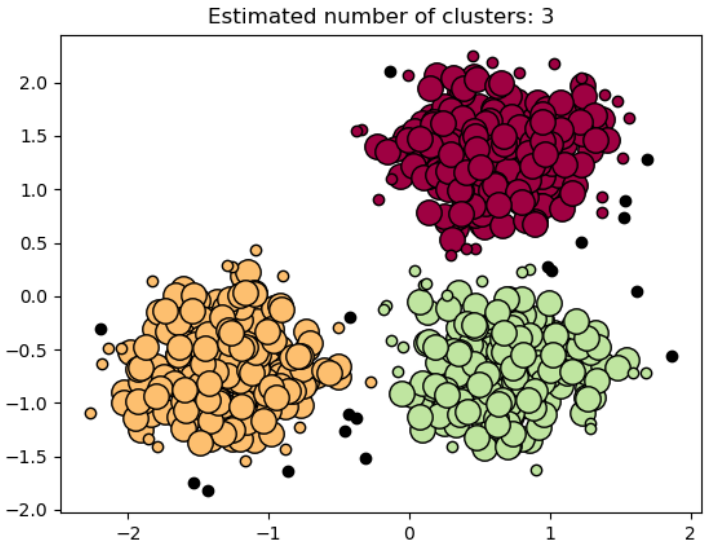


Figura 12: Funcionamiento del algoritmo de agrupamiento DBSCAN [31]

2.2.6 Definición de descriptores

Un descriptor visual es definido como la descripción detallada de un objeto, la cual es considerada para su identificación o análisis en un proceso de reconocimiento de objetos [32].

Los descriptores están constituidos por estructuras de datos que contienen información sobre el objeto o escena de interés, de forma ordenada, normalmente en forma de vector o matriz. En algunas ocasiones, la escena o imagen, en sí misma, puede considerarse descriptor.

Conforme a las características de los descriptores visuales, pese a que suelen fundamentarse en características diferenciadoras del objeto como, por ejemplo: el color, la intensidad o la forma, es difícil realizar una clasificación según su naturaleza. Así pues, según su nivel de abstracción se suelen clasificar en:

- **Descriptores locales**: actúan sobre regiones previamente calculadas o identificadas de la escena o imagen. Construyen un vector de características con la información aportada por la región de interés y su vecindario. En reconocimiento tridimensional la región e interés es conocida como puntos destacados o keypoints. El descriptor es el resultado de la totalidad de vectores de características calculados.
- **Descriptores globales**: resumen la información de la escena o imagen en un único vector o matriz de características. A pesar de su simplicidad, son utilizados para múltiples tareas dado su bajo coste computacional. Esto se debe a la capacidad que poseen de agrupar una gran cantidad de información en un pequeño conjunto de datos.

Como podemos apreciar en la Figura 13, los sistemas de aprendizaje profundo no requieren de la interacción humana para realizar la extracción de características, sin embargo, los sistemas de aprendizaje automático necesitan una definición previa de descriptores.

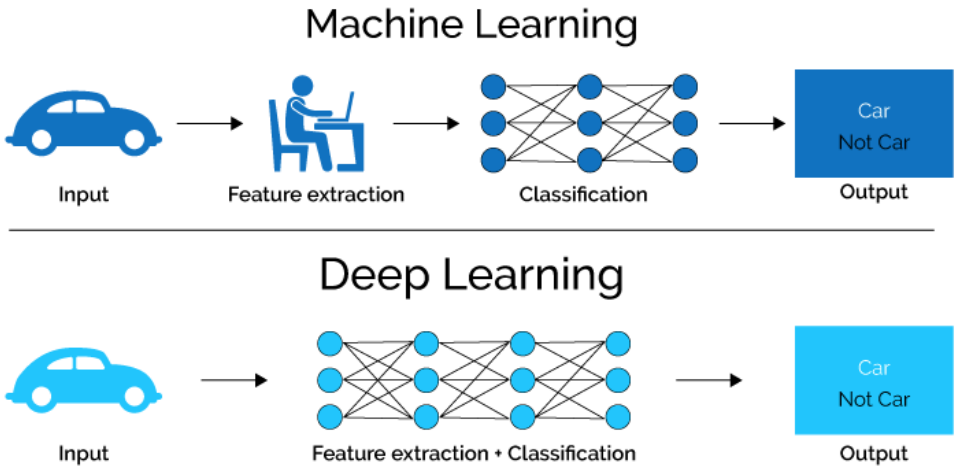


Figura 13: Extracción de características en sistemas de aprendizaje automático y aprendizaje profundo

2.2.7 Modelización de datos

La representación de la información tridimensional consta de cuatro tipos de representación básicos. La elección del formato correcto para cada sistema depende del objetivo a conseguir o el formato de salida del dispositivo de adquisición. Las nubes de puntos comentadas en una sección anterior son el modelo de representación más utilizado, este formato nos permite crear otros tipos de modelos 3D de gran utilidad para varios campos, incluyendo realidad virtual, imágenes médicas o arquitectura. Los principales modelos de datos tridimensionales son:

Imagen de profundidad e imagen RGB-D

La imagen RGB-D es una representación en un plano bidimensional de los tres canales de color (RGB) y un canal adicional para la profundidad (D) de la escena. Por otro lado, la imagen de profundidad sólo contiene la información relacionada con la

distancia de las superficies de los objetos de la escena. Sin embargo, la sencillez de estas representaciones provoca una pérdida de precisión e información tridimensional de gran validez motivo por el cual no suelen ser utilizadas.

Figura 14: Imagen de profundidad, Imagen RGB e Imagen RGBD [33]

Malla

Es uno de los tipos de representación más utilizados debido a su precisión, puede aproximar cualquier tipo de superficie sin incrementar en gran cantidad el volumen de información. Su representación consta de un conjunto de vértices y triángulos o polígonos encargados de conectarlos.

Su representación aproxima una superficie 3D sin conocer el interior de la figura, aportando detalles sólo de su superficie por lo que es considerada una representación 2,5D, ver Figura 15. Se puede considerar que el resultado es una representación volumétrica ficticia, debido a que no existe una figura sólida sino un conjunto de planos interconectados cuyo aspecto volumétrico es aportado por su topología.

No podría ser una representación de entrada en una red neuronal debido al número variable e irregular de vértices y primitivas. Añadido a la dificultad que supone la generación de mallas con datos del mundo real, hace que sea la representación por antonomasia en la informática gráfica, estando meramente presente en aplicaciones relacionadas con videojuegos, CAD, CAM o BIM.

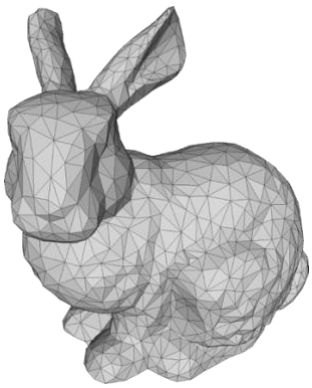


Figura 15: Malla tridimensional [24]

Volumétrica

La representación volumétrica consiste en un array tridimensional conocido como grid, donde cada celda corresponde a un elemento llamado vóxel. El vóxel es considerado como la extensión tridimensional del píxel. Pese a su gran similitud, a diferencia de las mallas, la representación volumétrica es tridimensional y carece de vértices y primitivas que son sustituidos por cubos.

Cada celda del grid almacena un valor que indica si la celda contiene vóxel o no, ver Figura 16. Todos los vóxeles que componen el grid deben tener el mismo tamaño, la representación de los vóxeles es tan sencilla que no se ha desarrollado ningún estándar para su almacenamiento.

La sencillez de esta representación tiene como consecuencia una pérdida proporcional de precisión, por ello es necesario definir un valor de precisión ρ para el vóxel que encerrará una porción del espacio de dimensiones $\rho \times \rho \times \rho$ unidades. La dependencia de la calidad de la representación con el valor de precisión del vóxel se debe a que toda variación espacial dentro del vóxel se pierde, por ejemplo, una figura cuyas dimensiones son menores a la precisión del vóxel, perderá todos los detalles siendo representada por un único cubo.

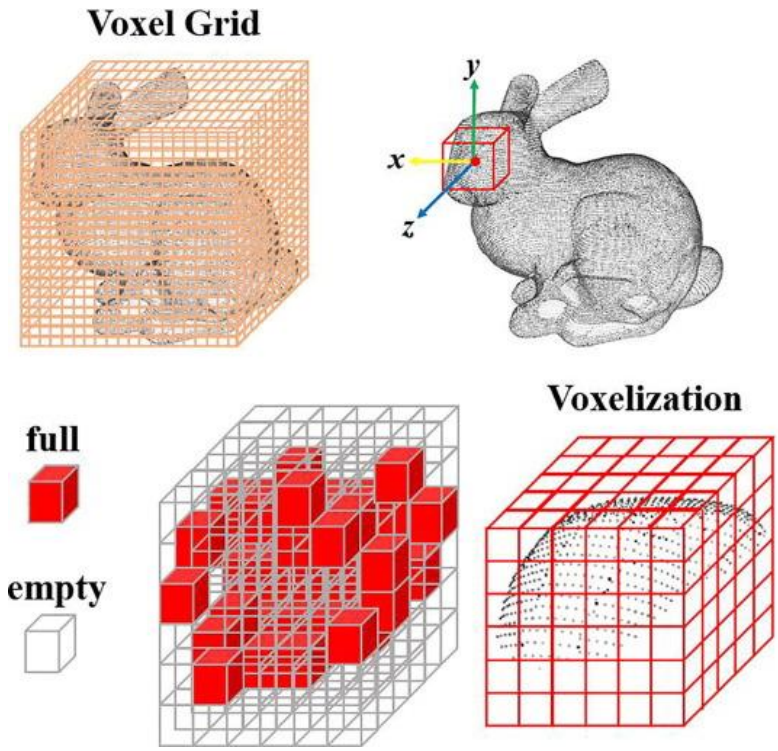


Figura 16: Estructura de un grid compuesto de vóxeles [34]

La representación volumétrica no se suele generar captando directamente la información de la escena, es fruto de la transformación de otros tipos de representaciones, siendo el proceso más común la transformación de una nube de puntos en una representación regular volumétrica. El objeto obtenido presenta una gran regularidad y uniformidad, ver Figura 17, debido a que los vóxeles que lo componen son del mismo tamaño. Estas características permiten establecer una dimensión fija del grid y usarlo para entrenar redes neuronales convolucionales.

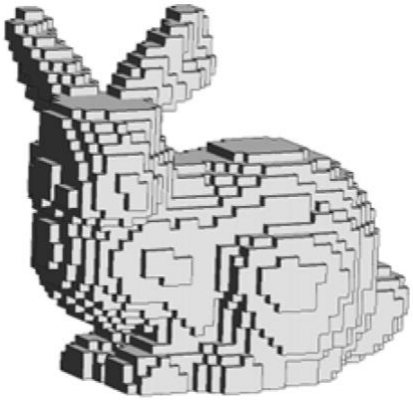


Figura 17: Grid compuesto por vóxeles [24]

2.3 Deep Learning

El aprendizaje automático ha supuesto grandes avances en el ámbito de la Inteligencia Artificial con la creación de algoritmos encargados de tomar decisiones sobre tareas del mundo real.

El Deep Learning o aprendizaje profundo, es un nuevo paradigma del aprendizaje automático, que se ha hecho muy popular en los últimos años gracias al uso de GPUs para el procesamiento en paralelo. Esta tecnología ha posibilitado el aumento del desarrollo de la IA permitiendo usar técnicas conocidas de forma más rápida y con menor costo.

A diferencia del Machine Learning, el Deep Learning está sometido a una menor supervisión. El sistema, en lugar de necesitar una especificación formal de todo el conocimiento que requiere, comienza su tarea a partir de conceptos simples que mediante su relación van creando conceptos de mayor complejidad, formando una jerarquía de conceptos [35]. Si se creara un grafo de esta jerarquía los conceptos más complejos se situarían sobre lo más simples creando una estructura muy profunda, por este motivo se decidió nombrar este aprendizaje como profundo.

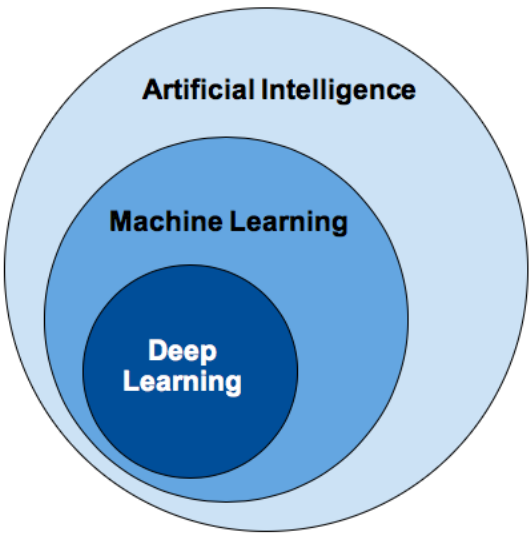


Figura 18: Ámbito del Deep Learning [36]

El Deep Learning surge como la necesidad de que los ordenadores extraigan su propio conocimiento, con su propia representación de este y utilizando su propio esquema de razonamiento, limitando el uso del conocimiento humano.

Los algoritmos de aprendizaje profundo están estructurados por capas anidadas mediante interconexión de sus nodos. Por lo que cada vez que el sistema se enfrenta a una nueva experiencia, aprende de ella reorganizando las conexiones de los nodos. Esto es conseguido mediante el uso de redes neuronales a gran escala, permitiendo simular el funcionamiento del cerebro para que el sistema aprenda y reconozca patrones por si mismo.

Las redes neuronales fueron descubiertas en los años 50 con la finalidad de reconocer patrones de imágenes. Sin embargo, la imposibilidad de simular un gran número de neuronas simultáneamente fue motivo para dejar esta tecnología en el olvido al no poder resolver problemas complejos de la IA. En la década de los 80, esta tecnología fue retomada con los primeros modelos de Deep Learning. En la última década, se ha conseguido su mayor eficiencia con los avances tecnológicos comentados anteriormente.

La primera capa de la máquina aprende las características más primitivas de la imagen como pueden ser los bordes. El principal beneficio del aprendizaje profundo es que permite resolver uno de los principales problemas de la IA como es la dependencia de la representación utilizada para los datos, añadiendo representaciones que son expresadas de forma más simple. En el aprendizaje automático para detectar estas características es necesario un conocimiento humano de alto nivel.

Una vez conocidas estas características en la primera capa, se envía la información extraída a la siguiente capa para extraer datos de mayor complejidad. Este proceso es repetido durante toda la red hasta obtener un resultado fiable de reconocimiento. La imagen que se encuentra a continuación muestra el funcionamiento de un sistema de reconocimiento de una persona usando métodos de complejidad simple como son la detección de bordes, esquinas y contornos.

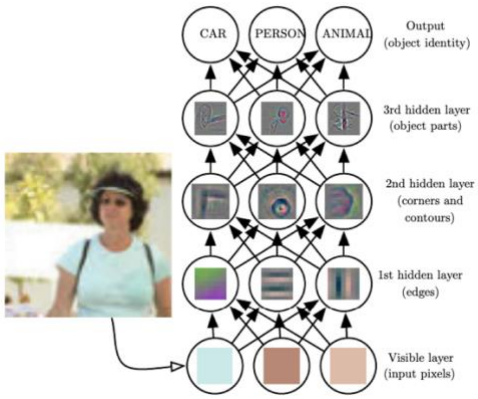


Figura 19: Esquema de funcionamiento de un sistema de aprendizaje profundo [35]

El ejemplo por antonomasia de un modelo de deep learning es la red neuronal artificial (RNA) conocida como perceptrón multicapa o multi layer perceptron (MLP). Es un aproximador para funciones matemáticas que mapea valores de entrada a valores de

salida. La función es el resultado de la composición de funciones más simples. Está formado por al menos tres capas de neuronas en las que cada neurona está conectada con todas las neuronas de la capa anterior, ver Figura 20.

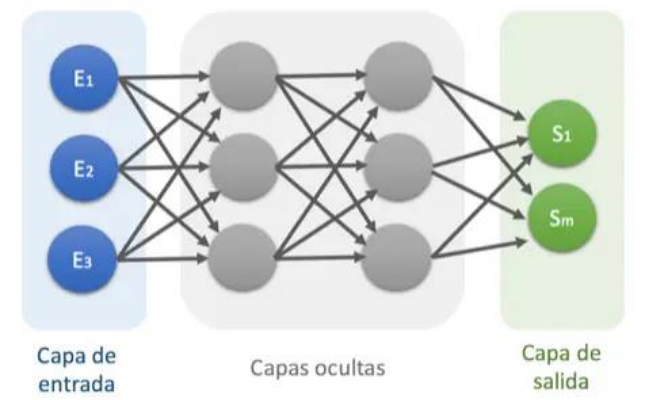


Figura 20: Estructura perceptrón multicapa (MLP) [37]

En conclusión, el aprendizaje profundo es un tipo de aprendizaje automático que permite obtener conocimiento de alto nivel, en un formato de difícil representación para las computadoras, siguiendo pasos similares a la detección de patrones realizada por los humanos. Las características de este paradigma le permiten representar el mundo como una jerarquía de conceptos interconectados, donde cada concepto es definido en relación con otros más simples.

2.3.1 Redes Neuronales Artificiales

Las Redes Neuronales Artificiales (RNA) son un sistema de procesamiento de información mediante unidades básicas de procesamiento que se basan en el funcionamiento de la célula fundamental del sistema nervioso humano: la neurona [38]. Al igual que las neuronas humanas, las neuronas artificiales se interconectan entre sí, intercambiando información y formando redes neuronales. Estas redes producen una salida a partir del conjunto de datos de entrada proporcionados.

El comportamiento de estas neuronas sigue el funcionamiento de la neurona humana, al recibir las señales de entrada, activarán la neurona según una función de activación determinada. Cuando esto ocurre, se genera una señal de salida, que será transmitida a otra neurona. Mediante la interconexión de grandes cantidades de neuronas, se conforman las redes neuronales. La información aportada a las neuronas conformará los datos de entrada de la red, que es procesada en cada una de sus neuronas de distinta forma, obteniendo una información de salida.

La neurona tiene capacidad de realizar divisiones lineales del espacio de características definido por los valores de entrada, encontrando fronteras de separación de naturaleza lineal, para aproximar los datos que siguen una distribución lineal en el espacio (Figura 21). En cambio, la aplicación más importante se puede apreciar en la imagen de la derecha de la Figura 21, donde es necesario trazar una circunferencia para poder separar los 2 tipos de datos en clases diferentes. Por lo que las neuronas son capaces de encontrar límites de decisión de naturaleza no lineal, aproximando prácticamente cualquier función matemática.

Figura 21: Conjunto de datos separables linealmente y no separables linealmente [39]

Funciones de activación

Las funciones de activación son los operadores necesarios para realizar la aproximación de funciones no lineales. Tras multiplicarse los datos de entrada por los pesos de la capa y sumarse el sesgo de esta, el resultado obtenido sirve de entrada a una nueva función cuyo objetivo es posibilitar a la red el aprendizaje de cualquier tipo de función (lineal o no lineal). Al igual que las neuronas biológicas, la salida de esta función es un valor de activación. Este valor permite controlar que, si se produce un valor negativo, este valor no se propague en adelante desfavoreciendo el ajuste de los pesos. Existen distintos tipos de funciones de activación, se buscan funciones cuyas derivadas sean simples, para minimizar con ello el coste computacional:

- **Lineal:** no aplica ningún operador, el resultado se propaga a la siguiente capa sin modificar.

- **Sigmoide:** es una función natural de la regresión logística. Mapea el espacio de entrada en un rango (0, 1).

$$\boldsymbol{f(x) = \frac{1}{1 - e^{-x}}}$$

- **Tangente hiperbólica:** sigue la misma filosofía que la función Sigmoide a diferencia que el rango de mapeo es (-1, 1).

$$\boldsymbol{f(x) = tanh(x)}$$

- **Unidad de Rectificación lineal (ReLU):** esta función en caso de tener un valor negativo devuelve 0, si el valor es mayor que 0, lo devuelve.

$$\boldsymbol{f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}}$$

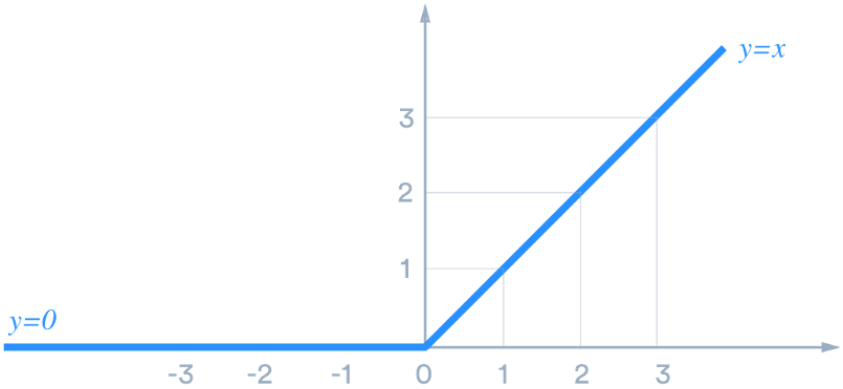


Figura 22: Gráfica función ReLU [40]

Existe una variante, la función **Leaky ReLU**, que multiplica los negativos por un coeficiente rectificativo y sigue manteniendo los positivos con su valor de entrada.

$$f(x) = \begin{cases} 0, & x < 0 \\ a * x, & x \geq 0 \end{cases}$$

- **Softmax:** transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de la salida es 1.

$$f(x)j = \frac{e^x j}{\sum_{k=1}^{K} e^x k}$$

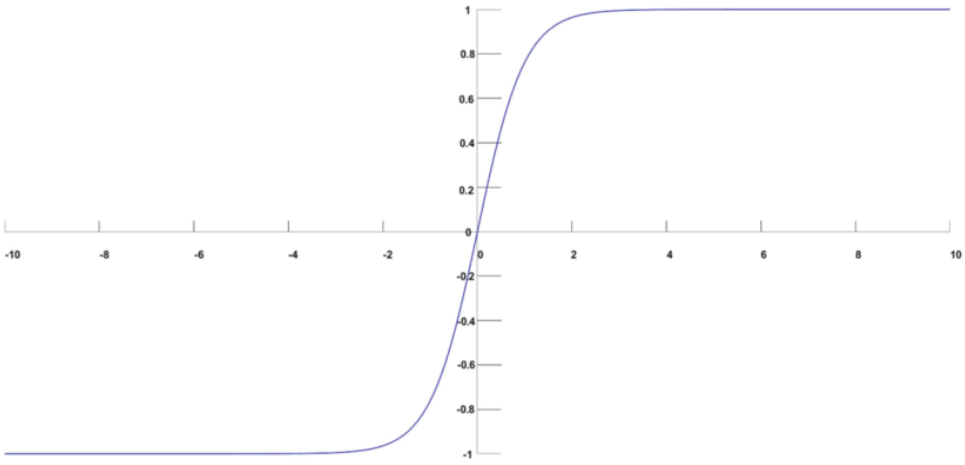


Figura 23: Gráfica función Softmax [41]

Las RNA, son modelos computacionalmente complejos, con un alcance y ámbitos de aplicación aún por conocer. La complejidad de sus procesos provoca un alto costo computacional por lo que es necesario ajustar sus hiperparámetros para obtener un correcto entrenamiento y poseer un hardware especializado para ello como son las GPU. Este tipo de procesadores dada su arquitectura permiten ejecutar varios hilos en paralelo, permitiendo realizar operaciones con matrices de manera eficiente a diferencia de las CPU donde un solo hilo, debe calcular cada posición de la matriz resultando altamente ineficiente.

Arquitectura

Las redes neuronales están estructuradas en capas de neuronas donde cada capa procesa la información recibida de la capa interior, produciendo una salida que será enviada a la siguiente capa [35]. A nivel estructural, como se muestra en la Figura 24 , se puede distinguir entre 3 tipos de capas:

- Capa de entrada.
- Capas ocultas.
- Capa de salida.

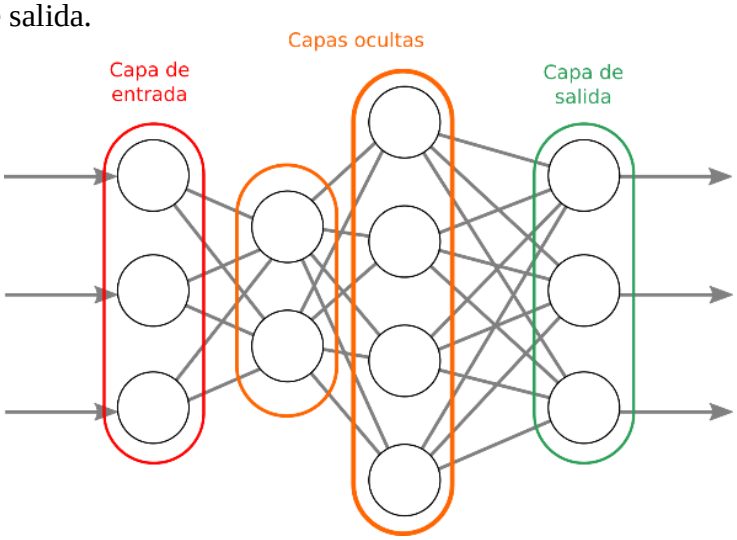


Figura 24: Estructura de las redes neuronales en aprendizaje profundo [42]

Podemos concluir que en las RNA cada capa está constituidas por nodos o neuronas, las cuales están conectados a cada neurona de la siguiente capa. Cada unión entre dos neuronas representa un peso sináptico. A través de la capa de entrada de la red se posee una información que es procesada en la red hasta llegar a la capa de salida. En tareas de clasificación, la capa de salida poseerá tantas neuronas como clases haya que distinguir en el problema. Las capas intermedias son denominas ocultas, especialmente en Deep Learning, es muy común obtener varias capas debido a la profundidad de la red.

El objetivo de las redes neuronales es aprender una función matemática, que para cada valor de entrada X mapee un valor de salida Y. Con un ajuste adecuado de los pesos y un entrenamiento apropiado, es posible obtener una red neuronal que mapee correctamente cada valor de entrada a un valor de salida según la función aprendida.

Entrenamiento

El entrenamiento es una fase fundamental en cualquier problema de aprendizaje automático, para realizar el entrenamiento es necesario un conjunto de datos etiquetados. Es decir, datos con un valor real Y asociados a un valor X de entrada (aprendizaje supervisado, ver Figura 25).

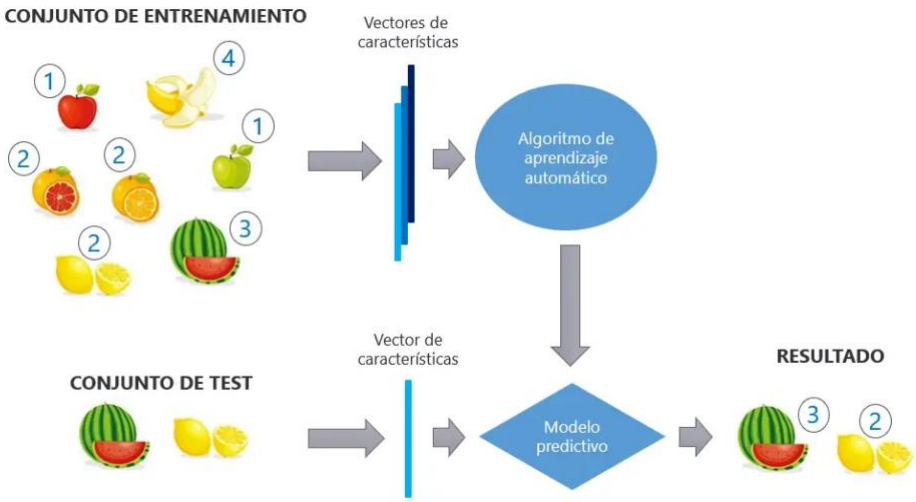


Figura 25: Entrenamiento supervisado [43]

El conjunto de datos debe dividirse en 2 particiones: entrenamiento y test. Los datos de entrenamiento serán utilizados para entrenar el modelo, mientras que el conjunto de test permitirá ir evaluando el entrenamiento y probando la validez de este.

El proceso de entrenamiento consiste en introducir en la red un conjunto de datos de entrada y propagarlos hasta obtener un valor de salida. El valor de salida obtenido debe ser comparado con la etiqueta real asociada al dato de entrada. Además, la predicción aporta una medida de error que sirve para calcular el error cometido en cada capa, lo que permite actualizar los pesos en futuras iteraciones para disminuir el error.

Durante el entrenamiento, teniendo en cuenta el funcionamiento del algoritmo y el conjunto de datos, se deben ajustar los parámetros hasta generar un modelo perfectamente entrenado, que sea capaz de predecir algún fenómeno mediante el aprendizaje de la distribución probabilística de los datos.

Según la tarea a resolver o el esquema de aprendizaje el proceso varía dando lugar a diferentes representaciones del conocimiento. En el caso de las redes neuronales, el aprendizaje parte de un modelo ya constituido compuesto por pesos y sesgos que se han generado aleatoriamente en el intervalo (0,1). El objetivo es ajustar los pesos lo mejor posible hasta obtener el comportamiento deseado, básicamente su función es minimizar la función de error o coste, lo que equivale a aproximar la salida de la red al valor real.

Algunos elementos básicos presentes en el entrenamiento de una red neuronal son:

Back-propagation

El principal objetivo del entrenamiento de redes neuronales es minimizar la función de coste encontrando los pesos adecuados para las aristas de la red. El algoritmo numérico back-propagation, en castellano, propagación hacia atrás del error, es el encargado de regular estos pesos.

El algoritmo, es el más utilizado en entrenamientos, se encarga de tomar datos de entrada y propagarlos por la red hasta obtener un valor de salida. En este punto, actúa en

consecuencia a su nombre retrocediendo a la capa de entrada para ajustar los pesos en función del error cometido en la predicción. En caso de que la predicción haya sido correcta, la medida de error obtenida será nula y los pesos sinápticos no sufrirán variaciones.

De esta forma el algoritmo de entrenamiento seguiría los siguientes pasos:

1. Se toma una muestra perteneciente al dataset de entrenamiento y se propaga por la red hasta obtener una salida.
2. El valor obtenido en la salida es comparado con el valor real, obteniendo como resultado una métrica con el error de coste producido por la red.
3. Se aplica el gradiente descendente hasta la primera capa, calculando la derivada parcial de la función de coste en cada capa con respecto a los pesos que la componen.
4. Los pesos y sesgos de cada capa se actualizan según el error cometido por la red.
5. Este proceso se debe repetir para todos los elementos del conjunto de entrenamiento. Para conseguir una solución óptima es necesario repetir el proceso varias veces a través del conjunto de datos, cada ciclo es denominado epoch.

Hiperparámetros

El gran número de ajustes que requiere una red neuronal provocan que su manejo no sea trivial. De todas las configuraciones de parámetros posibles, sólo es posible influir en una pequeña parte. El ajuste de pesos y sesgo se realiza automáticamente en el entrenamiento por el algoritmo de propagación hacia atrás del error. Los hiperparámetros, son un tipo de ajustes que deben ser configurados manualmente antes del entrenamiento de cualquier red neuronal. Algunos ejemplos son:

- **Diseño de la red**: escoger el número adecuado de unidades ocultas, capas, unidades de entradas y salidas. Es el ajuste más importante y posiblemente el más complejo.
- **Ratio de aprendizaje**: determina como se ve afectado cada peso de la red según el error cometido en cada capa tras el algoritmo de propagación hacia atrás del error.
- **Decaimiento del ratio de aprendizaje**: para evitar el sobreajuste se suele reducir en cada epoch el ratio de aprendizaje una fracción de su total. Comúnmente suele ser un 90 o 95 por ciento, o hacer uso de escalas logarítmicas.
- **Algoritmo de optimización**: más óptimo considerando el tamaño y la complejidad del conjunto de datos.
- **Número de epochs:** este parámetro hace referencia a cuantas veces va a procesarse el conjunto de entrenamiento en su totalidad. Un valor muy elevado puede producir sobreajuste.
- **Función de coste**: su labor es determinar el error entre el valor real y estimado, según la naturaleza de la tarea se selecciona una función distinta. Las funciones más comunes son la entropía cruzada para problemas de clasificación y el error cuadrático medio para problemas de regresión.

Optimizadores

Los optimizadores son métodos con un papel crucial en el proceso de entrenamiento, son los encargados de generar pesos cada vez mejores. Su funcionamiento se basa en calcular el gradiente de la función de coste por cada peso (parámetros) de la red. La intención principal es de minimizar el error, para ello se modificará cada peso en la dirección negativa del gradiente. El conjunto de métodos en busca de un mínimo local es conocido como métodos de optimización basados en el gradiente descendente. A continuación, se describen brevemente los principales optimizadores disponibles en Keras:

- **SGD (Stochastic Gradient Descent):** o descenso de gradiente estocástico, es un algoritmo variante del descenso de gradiente, encargado de actualizar los parámetros del modelo uno por uno.

- **Mini-Batch Gradient Descent:** en castellano, gradiente descendente mini-batch, es una combinación de los métodos SGD y de descenso de gradiente por lotes. Se encarga de dividir el conjunto de datos de entrenamiento en pequeños lotes y realizar una actualización para cada uno de los lotes. Este procedimiento crea un equilibrio entre la robustez del SGD y la eficiencia del descenso del gradiente por lotes.

- **SGD with Momentum:** en castellano, gradiente descendente con momentum. La idea básica de este método es calcular la media ponderada exponencial y usarla para actualizar los pesos. Este paradigma agrega un término de momento al descenso de gradiente estocástico regular, momemtum, es un parámetro de fricción encargado de ir midiendo la velocidad con la que converge el gradiente descendente. De esta manera, puede aumentar la estabilidad hasta cierto punto, para que pueda aprender más rápido, y también tener la capacidad de deshacerse de la optimización local.

- **RMSProp (Root Mean Square Propagation):** en castellano, propagación de la raíz cuadrática media. Es un paradigma que pretende acelerar la convergencia del gradiente, aspirando a moverse más rápido por el espacio de los pesos, que el espacio de los sesgos.

- **Adam (Adaptive moment estimation):** es un algoritmo muy confiable para acelerar la convergencia del gradiente descendente, combina las mejores características de los métodos RMSProp y SGD con momentum [44].

Sobreajuste

El sobreaprendizaje consiste en extraer un conocimiento que realiza la tarea de manera excelente con el conjunto de entrenamiento, sin embargo, con los datos de testeo y validación el resultado empeora significativamente. Este fenómeno es muy casual en los algoritmos basados en redes neuronales.

Para evitar este problema es recomendado utilizar un dataset lo más abundante posible, balanceado y si es viable aplicar técnicas de aumento de datos para incrementar el número de datos.

Para detectar este fenómeno hay que comparar la función de coste de entrenamiento con la de validación. Si en algún epoch las gráficas comienzan a separarse de manera que el coste de entrenamiento se reduce y el de validación aumenta a partir de ese punto, existe un claro signo de sobreajuste. En caso de sobreajuste, es necesario reconfigurar los hiperparámetros o la red no podrá mejorar el entrenamiento a partir de ese epoch.

Han sido varios los esquemas de redes neuronales profundas propuestas: Redes Neuronales Convolucionales, Redes Generativas Adversarias, Redes Neuronales Recurrentes. Dadas las características de este trabajo se hará uso de las Redes Neuronales Convolucionales.

Redes Neuronales Convolucionales (CNN)

Las redes neuronales convolucionales (en inglés, Convolutional Neural Networks), también conocidas como CNN, son un tipo de redes neuronales artificiales con aprendizaje supervisado. Las redes CNN procesan sus capas siguiendo el funcionamiento del ojo humano para identificar características en las entradas que permitan la identificación de objetos. Surgen debido a la gran dimensión y cantidad de parámetros necesarios a la hora de diseñar una red neuronal para la clasificación de imágenes [35]. Para comprender esta problemática, las imágenes actuales suelen tener dimensiones de hasta 40 megapíxeles. La idea existente daba como entrada a la red cada píxel de la imagen por cada uno de los canales que constituyen la imagen, por lo que una imagen de 40 megapíxeles tendría 40 millones de entradas por cada canal, en el caso de las imágenes RGB, serían 120 millones de entradas.

El principal beneficio de las redes CNN es la compartición de parámetros para todos los píxeles de la imagen con el objetivo de extraer características de las imágenes. Se diferencia de las demás redes por el hecho de que cada una de las neuronas de las capas que la conforman no tienen que recibir conexiones de todas las neuronas de la capa anterior, solo algunas de ellas. Permitiendo simplificar la estructura de la red y disminuyendo los costes computacionales.

La extracción de características es realizada de forma jerárquica, partiendo de características más simples hasta conseguir características de alto nivel para alimentar la entrada de la última parte de la red cuya estructura es tipo MLP. Este proceso se puede observar en la Figura 19, donde las primeras capas de convolución extraen características muy simples como pueden ser los bordes, para que las capas posteriores extraigan formas. Con este procedimiento se realiza un procesamiento incremental, característico del aprendizaje profundo, en el que cada capa posterior maneja características de más alto nivel. Este tipo de redes están compuestas principalmente de 3 tipos de capas:

- **Capas de convolución**: se realizan operaciones de productos y sumas entre la capa de partida y los filtros (o kernels), generando un mapa de características [45]. Donde las características extraídas corresponden a la posible ubicación del filtro en la escena. Por estas condiciones, un filtro permite extraer las mismas características en cualquier parte de la imagen simplificando la complejidad de la red.

- **Capas de reducción o pooling**: la función de estas capas es disminuir la dimensión de la entrada, al reducir se pierde precisión, pero se gana compatibilidad. Esto permite que la red pueda clasificar mejor pese a que haya ejemplos de entrada que no ha visto en su totalidad. La reducción se realiza a través de medios estadísticos como son el máximo y la media, por ello que existan 2 tipos de reducción, ver Figura 26:

    - **Max-pooling**: recorre bloques de píxeles del cubo resultante de la capa anterior para agruparlos y elegir el valor mayor de cada vecindad.
    - **Average pooling**: este filtro también recorre los píxeles producidos por la capa de convolución para obtener el promedio de píxeles de cada vecindad y generar un único píxel.

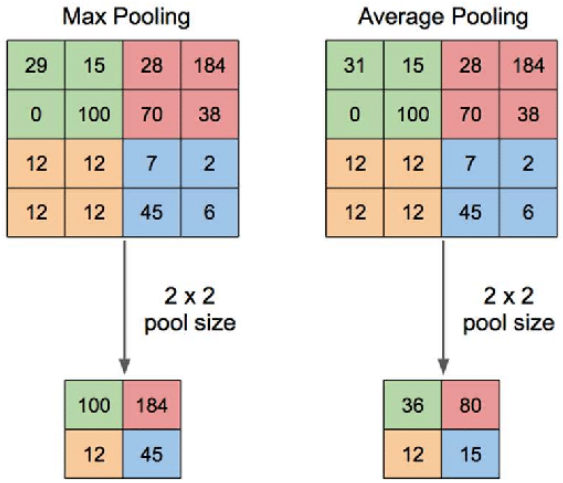


Figura 26: Comparación del funcionamiento de los métodos de la capa de reducción [46]

- **Capas densas o fully-connected**: al estar formada por al menos 3 capas es un tipo de MLP. La diferencia respecto a la versión clásica es que la red es alimentada con características en vez de píxeles, por lo que se ha descartado el contenido no relevante de la imagen. Al tener características como entradas se consigue reconocer entidades independientemente de su localización en la imagen. Es denominada capa densa porque todos sus nodos están conectados entre sí y su objetivo es realizar la clasificación de la imagen en base a los resultados de las capas anteriores.

En este trabajo dada la tridimensionalidad de los datos, se hace uso de un enfoque cuya arquitectura consta de una red CNN 3D, este tipo de redes suele utilizar datos modelados en forma de grids o nubes de puntos.

Las operaciones de las redes neuronales convolucionales tridimensionales se realizan de la misma manera que las redes neuronales convolucionales comunes con la única novedad de que cuentan con una dimensión más, proveniente de la profundidad de la escena.

2.3.2 Nubes de puntos en Deep Learning

En las tareas de reconocimiento, clasificación y detección de objetos los sistemas basados en aprendizaje profundo son los que mejores resultados aportan. De hecho, se han conseguido resultados por encima del 90% de éxito. Sin embargo, el reconocimiento de objetos tridimensionales sigue suponiendo un desafío para este paradigma, especialmente la búsqueda del método de representación más adecuado para el aprendizaje.

Las técnicas del aprendizaje profundo están aplicadas en mayor parte a datos estructurados, sin embargo, la naturaleza de las nubes de puntos dificulta los procesos, ver Figura 27. Los desafíos que provocan las características de estos datos al aprendizaje profundo son:

- **Irregularidad**: los datos de la nube de puntos son irregulares, lo que significa que los puntos no se muestran de forma uniforme en la escena, por lo que algunas regiones pueden tener puntos densos mientras que en otras estén dispersos. Esta irregularidad puede ser atenuada mediante técnicas de submuestreo, pero no puede ser eliminada al completo.
- **Carencia de estructura**: A diferencia de los píxeles, los puntos no están colocados en una cuadrícula regular. Cada punto es escaneado de forma independiente y por lo tanto la distancia entre puntos vecinos no es constante. Esto provoca que los filtros CNN existentes sean insuficientes.
- **Invariancia a la permutación**: los puntos de la nube son almacenados en una lista sin un orden específico. El orden de los puntos en la lista no tiene dependencia geométrica dado que representa la misma forma, por lo tanto, es invariante a la permutación. Sin embargo, cambiar el orden de los puntos modifica la estructura de su matriz subyacente. Esto permite que una nube de puntos pueda ser representada por dos matrices muy diferentes.
- **Distinta cantidad de puntos:** en las imágenes, dependiendo de la resolución de la cámara el número de píxeles es constante. Por otro lado, el número de puntos que contiene una nube puede variar dependiendo del sensor y escena capturada.
- **Invariante a la traslación**: los datos tridimensionales permiten su rotación y traslación. Por este motivo la representación aprendida del objeto debe de ser invariante a este tipo de transformaciones.
- **Falta de datos y oclusión**: provocados en la adquisición por parte del sensor.

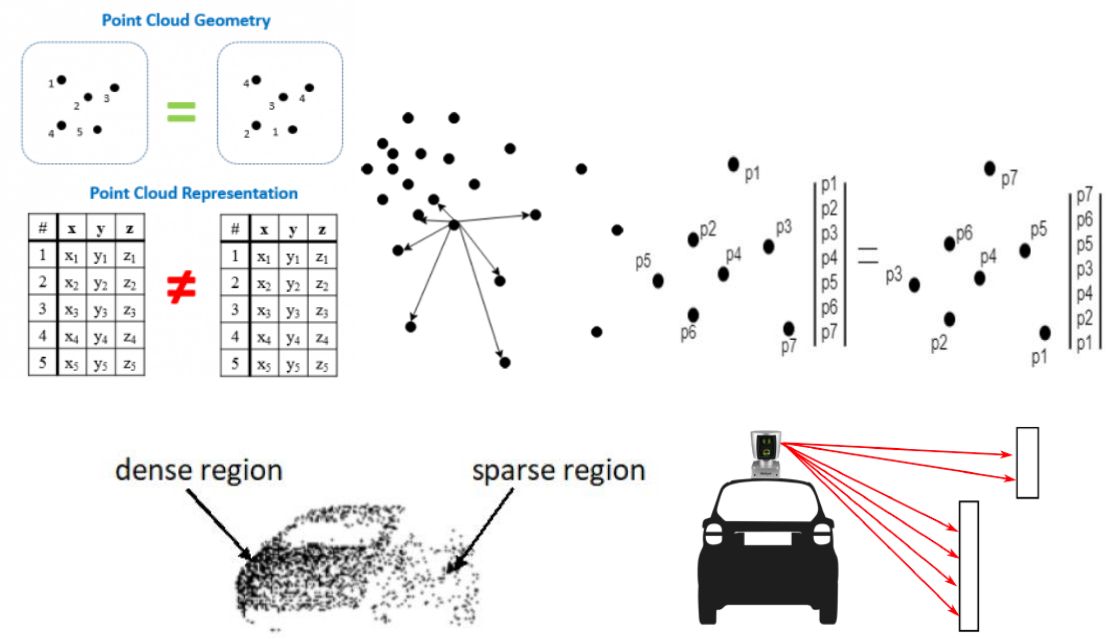


Figura 27: Características de la nube de puntos: Invariancia a la permutación, carencia de estructura, desorden, variedad de densidad y oclusión de datos. [47] [48]

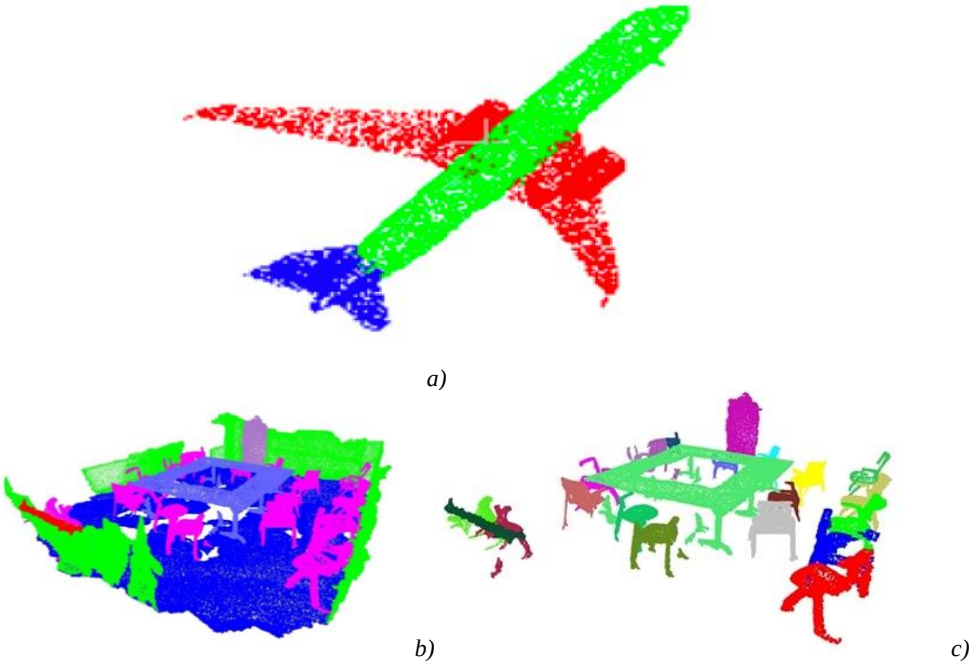
2.3.3 Aplicaciones de las nubes de puntos usando Deep Learning

Hay numerosas metodologías para trabajar con nubes de puntos. Para este TFG nos hemos enfocado en las tres principales y más útiles aplicaciones de estos métodos: segmentación, clasificación y detección.

Segmentación

La segmentación permite dividir una imagen o nube de puntos en distintas regiones de interés. La segmentación de una nube de puntos consiste en organizar, parametrizar y procesar los puntos de la nube, para agruparlos en elementos identificables. Para ello el modelo recopila el contexto global y local de cada punto con la finalidad de asignarle una etiqueta que le relaciona a un grupo con características similares. Hay tres posibles tipos de segmentación en nube de puntos:

- **Segmentación de partes**: etiqueta cada punto en un objeto y los agrupa identificando las diferentes partes de este, ver Figura 28 a.
- **Segmentación semántica**: etiqueta cada punto de una escena asignando a cada punto una clase según su correspondiente categoría semántica, ver Figura 28 b.
- **Segmentación de instancias**: identifica diferentes objetos de una misma categoría semántica para distinguirlos entre sí, ver Figura 28 c.



a) *b)* *c)*

Figura 28: Tipos de segmentación: por partes, semántica y de instancias [48][49]

Clasificación y detección

La clasificación de objetos ha sido una de las principales áreas en las que se utiliza el aprendizaje profundo, esto se debe a la importancia de reconocer los objetos de una imagen o escena en aplicaciones como puede ser la detección de personas en la conducción autónoma. La clasificación en las nubes de puntos es comúnmente conocida como clasificación de formas 3D. Dada una nube de puntos, la red debe clasificarla en una determinada categoría.

La detección de objetos es una extensión de la clasificación en la que se reconocen objetos de la escena y se localizan mediante un cuadro limitador, en la Figura 29 se observan estas diferencias.

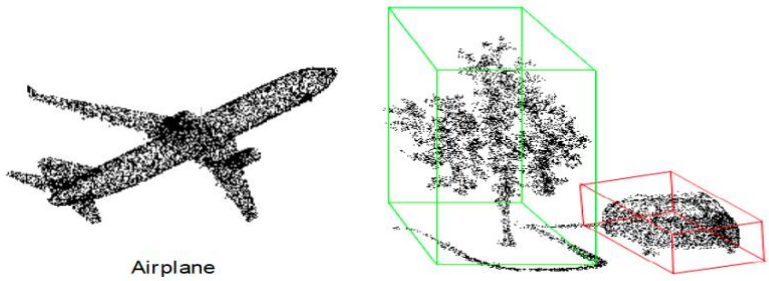


Figura 29: Clasificación y detección de nubes de puntos [48]

Los métodos utilizados para realizar estas tareas se encuentran detallados en el siguiente apartado.

2.4 Métodos de aprendizaje profundo para nube de puntos

Como hemos comprobado en los apartados anteriores, el gran éxito obtenido por el Deep Learning en tareas de detección de objetos, clasificación de objetos, segmentación semántica, etcétera., ha servido de base para abordar las correspondientes tareas con datos tridimensionales.

El aprendizaje de características en nubes de puntos se puede clasificar en dos categorías:

- **Métodos basados en árbol kd**: representan la nube de puntos de forma regular antes introducir la información a los modelos.
- **Métodos basados en nubes de puntos sin procesar**: la entrada está compuesta por nubes de puntos no estructuradas y desordenadas.

2.4.1 Aprendizaje estructurado basado en cuadrículas

Los primeros enfoques del procesado de nube de puntos mediante aprendizaje profundo apuntaban a convertir las nubes de puntos en cuadrículas estructuradas, para un fácil procesamiento por redes neuronales profundas. La lógica de este aprendizaje se debe a lo visto en apartado 2.3 Deep Learning, donde la capa de convolución en las redes CNN son las encargadas del aprendizaje de características. La operación de Convolución requiere tener una cuadrícula estructurada, por ello las nubes de puntos son procesadas en formas estructuradas.

Sin embargo, estos métodos conducen a una pérdida de información de profundidad y requieren un costo computacional más alto. Dada a la importancia del costo computacional y la información de profundidad en este TFG, estos métodos de aprendizaje han sido descartados.

Basado en vóxeles

En este método se pueden distinguir 2 etapas: el procesamiento de la nube y el aprendizaje. El procesamiento se encarga de convertir las nubes de puntos en grids con un número de vóxeles fijos (*ver representación volumétrica en* 2.2.4 Adquisición de datos tridimensionales). En la etapa de aprendizaje se diseña la red neuronal convolucional profunda, utilizando un número variado de capas convolucionales tridimensionales, agrupadas y conectadas. La operación de convolución sigue un enfoque similar a la convolución 2D, convolucionando los vóxeles de estructura $X \times Y \times Z$ con núcleos de tamaño $x \times y \times z$ con $x, y, z <= X, Y, Z$.

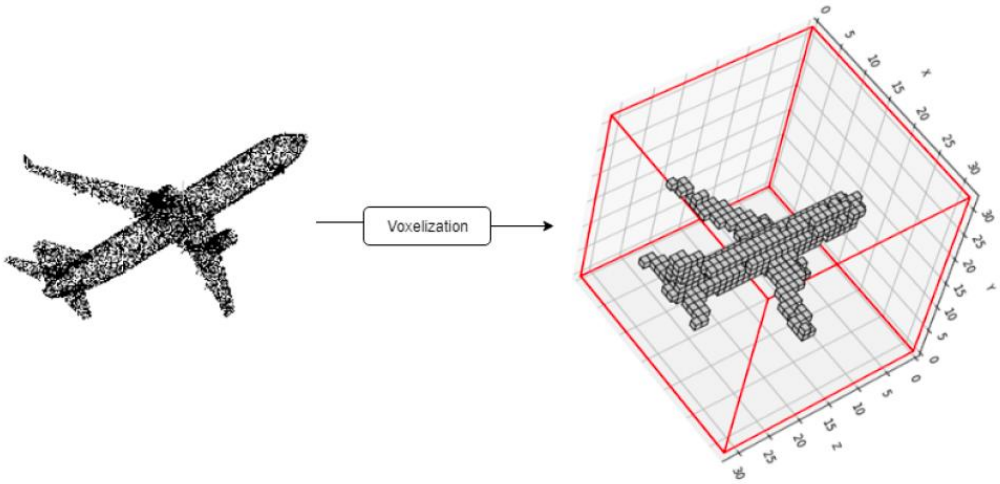


Figura 30: Representación basada en vóxeles [48]

Pese al buen rendimiento de los métodos basados en vóxeles, sufren un alto consumo de memoria debido a la dispersión de los vóxeles. Esta escasez provoca pérdidas de cálculo a la hora de convolucionar en regiones no ocupadas, sumándose a las pérdidas producidas por la representación volumétrica.

Basado en múltiples vistas

Los métodos de estas categorías utilizan las técnicas ya conocidas de las redes CNN bidimensionales. Para ello es necesario transformar la nube de puntos en una colección de imágenes bidimensionales desde distintos puntos de vista como se representa en la Figura 31.

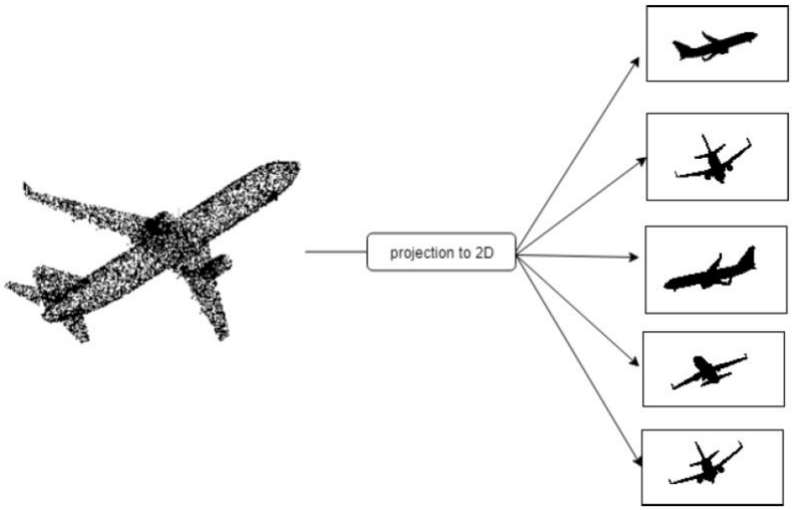


Figura 31: Proyección basada en múltiples vistas [48]

Estas redes muestran mejores resultados que los métodos basados en vóxeles, debido a que utilizan técnicas que ya han sido investigadas y desarrolladas exitosamente. Además, no arrastran las pérdidas producidas en la representación volumétrica.

2.4.2 Aprendizaje directamente con nube de puntos

Recientemente, se ha prestado una mayor atención al aprendizaje profundo con nubes de puntos sin procesar, sin perder información. PointNet [24] es el trabajo pionero en la utilización de este método y la base de la mayoría de los enfoques posteriores.

PointNet

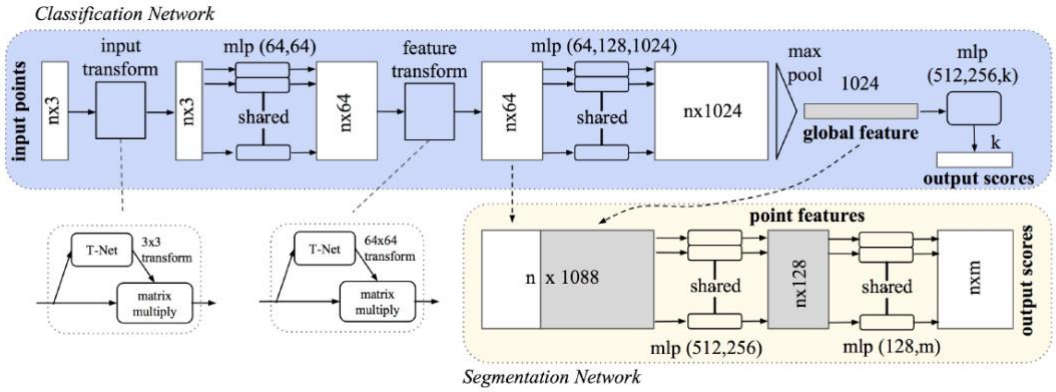


Figura 32: Estructura de PointNet [24]

PointNet [24] es una red neuronal convolucional tridimensional elaborada por los investigadores de la Universidad de Stanford. Surge de la necesidad de evitar la transformación de la nube de puntos en otro tipo de representación para poder servir de entrada a la red. Este enfoque es utilizado para tareas de clasificación y segmentación semántica. En la Figura 32 , se observa su arquitectura donde se distingue entre la red de clasificación y segmentación.

- **Clasificación**: tiene como entrada la nube de un objeto segmentado en una escena. La salida aporta k valoraciones para las k posibles clases del sistema.
- **Segmentación semántica**: tiene como entrada un objeto para segmentar sus partes o una región de una escena para segmentar sus objetos. La salida es una matriz de $n\ x\ m$ posiciones donde n es el número de puntos de entrada y m las posibles categorías.

Arquitectura

Dado que PointNet [24] consume datos de nubes de puntos sin procesar, los autores desarrollaron una arquitectura que se ajustara a las propiedades de las nubes de puntos. Teniendo en cuenta las características observadas en el apartado 2.3.2 Nubes de puntos en Deep Learning, las nubes poseen invariancia a la permutación.

La nube de puntos es representada como un conjunto de puntos $(P_i \mid i = 1, 2, \dots, n)$ donde cada punto P_i es un vector de coordenadas tridimensional (x, y, z) que puede ver aumentado su tamaño según la información adicional deseada en cada punto (formatos de la Tabla 2).

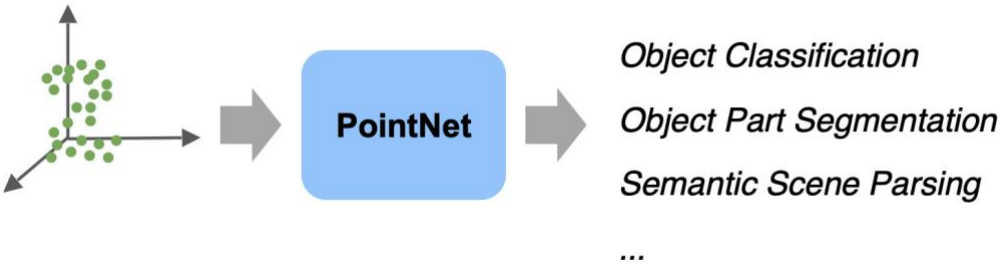


Figura 33: Esquema de los objetivos de PointNet [24]

En la fase inicial, cada punto es representado por sus dimensiones (x, y, z) y procesado independientemente. La capa de entrada de la red es de tamaño n x 3 donde n representa el número de nubes de puntos, ver Figura 34.

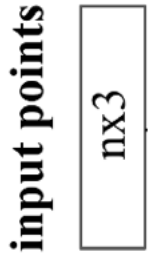


Figura 34: Fase de entrada de PointNet [24]

A las nubes de entrada, se les aplica una transformación consistente en multiplicar la matriz de entrada por una matriz de transformación afín, predicha por una red T-net ortogonal. Esta primera fase, presente en la Figura 35, se realiza para que la red sea invariante a cualquier transformación respetando las características de la nube de puntos.

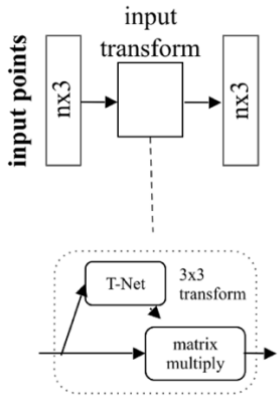


Figura 35: Fase de transformación de la matriz de entrada por red T-net en PointNet [24]

Tras transformar las nubes de puntos, se introducen en la siguiente fase, consistente en un MLP (2 capas de 64 neuronas cada una) donde se obtiene 64 características de cada nube, resultando una salida de n x 64.

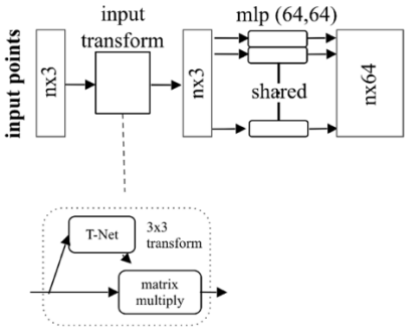


Figura 36: MLP de 2 capas de PointNet [24]

A estas matrices se le vuelve a aplicar una transformación como la usada en la primera fase, para que la nube sea invariante a cualquier transformación rígida. Con la única diferencia en las dimensiones de la matriz, que en vez de ser 3 x 3 será 64 x 64.

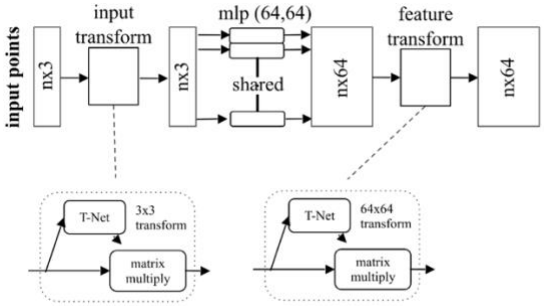


Figura 37: Segunda transformación con T-net PointNet [24]

El proceso se repite pasando las nubes transformadas a través de un bloque MLP. Sin embargo, en esta fase contará con 3 capas de 64, 128 y 1024 neuronas, obteniendo unas salida de 1024 características por cada nube.

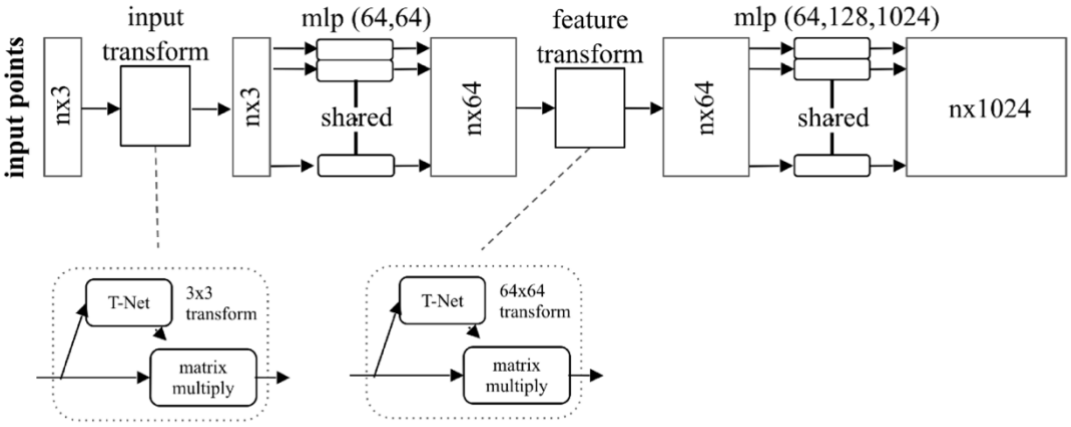


Figura 38: MLP de 3 capas de PointNet [24]

Finalmente, se aplica una función simétrica, la función max pooling, que permite que la nube sea invariante a permutaciones. La salida se pasa por otro bloque MLP de

tres capas fully-connected de tamaños de salida 512, 256 y k, donde k indica los resultados obtenidos en la predicción de los k objetos detectados.

Todas las capas, excepto la última, poseen ReLU como función de activación y un operador de normalización de batch. En esta última capa fully-connected es utilizado dropout, un método de regularización inteligente que reduce el sobreajuste del dataset, haciendo el modelo más robusto. Está configurado con una probabilidad de 0.7, es decir, la probabilidad de que se anule la salida de una neurona es de 0.3.

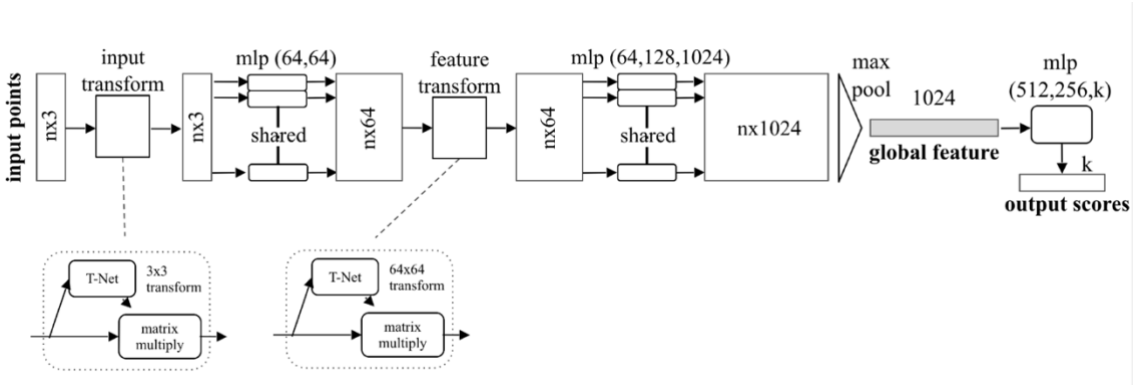


Figura 39: Arquitectura de la red de clasificación de PointNet [24]

La clave de este enfoque es el uso de una sola función simétrica, max-pooling. La red aprende un conjunto de funciones de optimización que seleccionan puntos de interés de la nube y codifican el motivo de su elección. Las capas fully-connected de la etapa final agregan estos valores aprendidos en un descriptor global para predecir la etiqueta correspondiente del punto (segmentación) o la forma del objeto (clasificación). Lo más destacable es que la red resume la nube en un menor conjunto de puntos, considerados de interés, aproximando las estructuras de los objetos.

Sin embargo, el uso de MLP sólo permite aprender las características locales de cada punto e ignora las conexiones entre puntos. PointNet no representa las características locales de los puntos vecinos, lo que limita su labor en escenas complicadas.

Por esto, se propuso otro enfoque denominado PointNet++ [50] para mejorar la extracción de características locales, también encargado de realizar las tareas de segmentación semántica y clasificación. PointNet++ propone el aprendizaje de características jerárquicas para dividir la nube de puntos localmente. Consta de 3 componentes visibles en la Figura 40: capa de muestreo, capa de agrupación, capa PointNet.

La capa de muestreo selecciona una serie de puntos de la nube de entrada para definir el centro del área local. La capa de agrupación emplea PointNet para extraer características locales después de agrupar las nubes por lo que PointNet se convierte en una especie de subred de PointNet++.

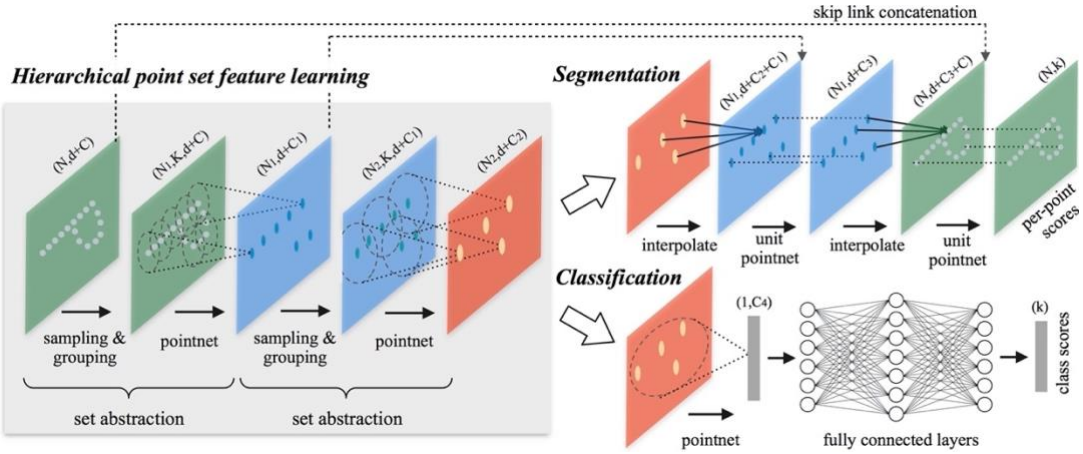


Figura 40: Arquitectura de PointNet++ [50]

Enfoques con cálculo de estructura local

Tras PointNet, se han desarrollado múltiples enfoques de vanguardia para capturar las estructuras locales. Estos métodos capturan la estructura local jerárquicamente de una forma similar a la convolución de grids, con cada jerarquía codificando una representación más rica.

Debido a la desestructuración inherente de la nube de puntos, el modelado de estructuras locales se basa en tres etapas básicas: muestreo, agrupación y función de mapeo, visibles en la Figura 41. El muestreo es utilizado para reducir la resolución de puntos de la misma manera que la operación de convolución reduce la resolución de mapas de características a través de capas convolucionales agrupadas. En la operación de agrupamiento, a medida que se muestrean los puntos vecinos representativos o centroides, se utiliza el algoritmo de k-vecino más cercano (kNN) para seleccionar los puntos vecinos más cercanos y agruparlos en un parche local. Los puntos en estos parches se utilizan para calcular la representación de características locales del vecindario. Tras obtener los puntos más cercanos a cada punto, la función de mapeo no lineal se encarga de mapear las características de los puntos vecinos y convertirlos en un vector de características que represente la estructura local. Dada la característica desestructurada de la nube de puntos, la mayoría de los enfoques aproximan la función utilizando un método basado en PointNet que se compone de perceptrones multicapas (MLP) y una función simétrica de agrupación máxima.

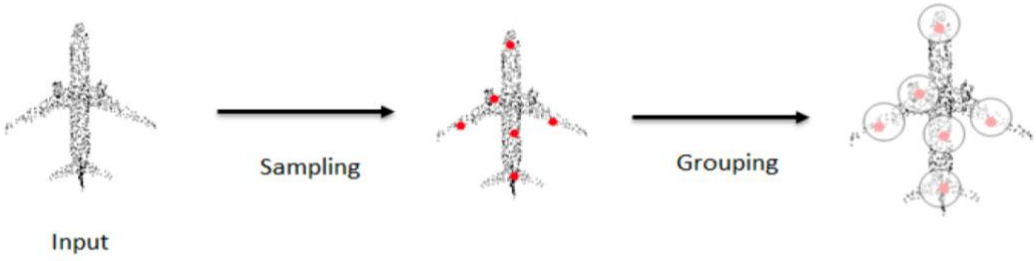


Figura 41: Muestreo y agrupación de nubes de puntos teniendo en cuenta estructuras locales [48]

Enfoques que no exploran la correlación local

Varios métodos siguen un enfoque similar a PointNet sin considerar la correlación de puntos dentro de las regiones locales. Sin embargo, aprenden las características de los puntos a través de un MLP compartido y las características de la región local se agregan mediante una función de agrupación máxima con el principio donde el ganador se lo lleva todo.

Enfoques que exploran la correlación local

Varios enfoques exploran las correlaciones entre puntos en una región local para mejorar la capacidad discriminativa. Esto es intuitivo, dado que los puntos no existen de forma aislada; más bien, se necesitan varios puntos juntos para lograr una forma significativa.

Enfoques basados en gráficos

Los enfoques basados en gráficos representan las nubes de puntos con una estructura gráfica al tratar cada punto como un nodo. Esta estructura es buena para capturar la correlación entre los puntos utilizando los bordes.

En conclusión, los trabajos iniciales sobre el aprendizaje profundo directamente con datos de nubes de puntos sin procesar no modelaron las regiones locales; los enfoques posteriores modelaron las regiones locales a través del muestreo y la agrupación. Más recientemente, se han propuesto varios enfoques que no solo modelan las regiones locales, sino que también exploran la correlación entre puntos en las regiones locales. De las investigaciones realizadas se puede concluir que los enfoques que modelan las regiones locales y tienen en cuenta la correlación entre puntos en estas, funcionan mejor. Por lo tanto, podemos concluir que los métodos actuales de aprendizaje profundo sobre nubes de puntos se pueden clasificar de la siguiente manera:

<table>
<tr><td colspan="6">Aprendizaje profundo en nubes de puntos</td></tr>
<tr><td colspan="6">Enfoques basados en</td></tr>
<tr><td colspan="2">Cuadrículas (grid)</td><td colspan="4" rowspan="2">Nubes de puntos crudas</td></tr>
<tr><td colspan="2">Basados en</td></tr>
<tr><td>Vóxeles</td><td>Múltiples Vistas</td><td colspan="3">Región local</td><td>PointNet</td></tr>
<tr><td>VoxNet</td><td>VMCNN</td><td colspan="2">Correlación local</td><td>Sin correlación local</td><td></td></tr>
<tr><td>NormalNet</td><td>ShapePFCN</td><td>PointCNN</td><td>Basado en Gráficos</td><td>PointNet++</td><td></td></tr>
<tr><td></td><td></td><td>PointWeb</td><td>Kd-Network</td><td>VoxelNet</td><td></td></tr>
<tr><td></td><td></td><td></td><td>DGCNN</td><td></td><td></td></tr>
</table>

Tabla 3: Clasificación de los enfoques del aprendizaje profundo en nubes de puntos y algunos ejemplos [48]

Capítulo 3 – Tecnologías utilizadas

3.1 Recursos hardware

Para la realización de este Trabajo de Fin de Grado, se ha utilizado la cámara CamBoard pico flexx (Figura 44) para la captura de datos tridimensionales, el ordenador de bajo costo NVIDIA Jetson Nano (Figura 42) para la creación y ejecución del sistema de visión artificial y un ordenador MacBook Pro (13-inch, 2018) [51] para realizar con mayor eficacia el entrenamiento del modelo.

3.1.1 NVIDIA Jetson Nano

Figura 42: NVIDIA Jetson Nano [52]

La computadora Jetson Nano es una SBC desarrollada por NVIDIA cuya finalidad es el desarrollo de aplicaciones de IA. Los avances en el ámbito de la Inteligencia artificial han incrementado la utilización de computadores en aplicaciones como tratamiento de imágenes, detección de objetos y reconocimiento facial entre otras tantas. Este dispositivo dado sus características [53], pretende desarrollar sistemas de IA de pequeño tamaño, económicos, de bajo costo y reducir el tiempo de desarrollo, ya que permite actualizar el rendimiento y las capacidades incluso después de implementar un sistema.

NVDIA provee de un kit de desarrollo abierto que permite la ejecución de múltiples redes neuronales en paralelo, proyectos listos para ejecutar, comunidad de desarrolladores activa, compatibilidad con los frameworks de IA más populares como son TensorFlow, PyTorch, Caffe y Keras.

La placa utiliza una arquitectura ARM de 64 bits donde el sistema operativo proporcionado es derivado de Ubuntu, pero permite compatibilidad con otros sistemas operativos de Linux. En Anexo 1. Guía de instalación, se describe el proceso de instalación del sistema operativo y los primeros pasos a realizar en la configuración de la placa.

Las característica más destacada de este dispositivo son su CPU de 4 núcleos, con capacidad de proporcionar 472 GFLOPS, permitiendo procesar sensores de alta resolución y ejecutar varias redes neuronales en paralelo simultáneamente y la GPU de 128 núcleos capaz de ejecutar la librería de procesado de datos CUDA que permite el acelero de aplicaciones gracias a la GPU. Estas funcionalidades, añadidas a su bajo consumo (entre 5 y 10 vatios), lo convierten en un dispositivo ideal para gateways inteligentes o robots domésticos.

La Figura 43 muestra los conectores disponibles en el computador:

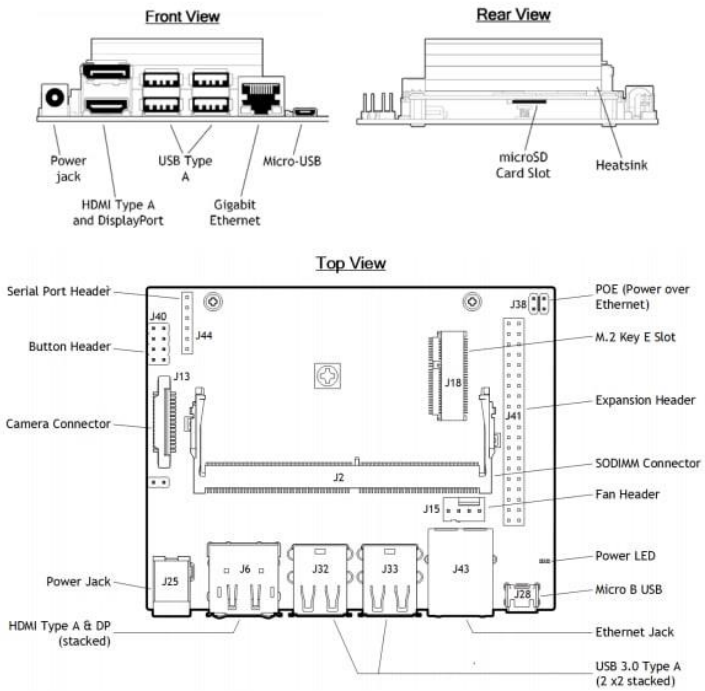


Figura 43: Estructura del computador Jetson Nano [53]

3.1.2 CamBoard pico flexx

Figura 44: CamBoard pico flexx [54]

La CamBoard pico flexx [54], es un kit de desarrollo de cámara 3D alimentado por USB. Desarrollado por la empresa pmd. Está basado en la tecnología PMD ToF y utiliza un láser clase 1 (VCSEL) como fuente de iluminación. Contiene un software multiplataforma para operar la cámara, una herramienta de visualización y un SDK

compatible con múltiples lenguajes y librerías. En el caso de utilizar Linux/ARM, es necesario instalar los controladores descritos en Anexo 1. Guía de instalación. La cámara es capaz de captar:

- **Nubes de puntos**: construida mediante los valores tridimensionales de cada punto de la escena.
- **Valores de confianza**: que indican si el valor de profundidad de un punto es fiable o su calidad ha sido degradada, por ejemplo, por subexposición o saturación. Los valores van de 0 a 255, siendo 0 un valor de baja credibilidad.
- **Valores de grises**: el valor de profundidad de cada punto también puede ir acompañado de un valor de gris que indica la amplitud de la señal infrarroja emitida.

Funcionamiento

La cámara posee sensores PMD, estos sensores tridimensionales basados en el principio de tiempo de vuelo permiten capturar imágenes tridimensionales sin componentes electrónicos complejos [55].

En lugar de utilizar un único rayo láser para obtener la escena tridimensional, toda la escena es iluminada con luz modulada. La ventaja de los dispositivos PMD es que permite observar la escena iluminada con una matriz de píxeles inteligente, donde cada píxel puede medir individualmente el tiempo de respuesta de la luz modulada. Con esta tecnología la cámara capta a la vez la intensidad y la distancia de cada píxel a diferencia de las cámaras que solo captan la intensidad.

La cámara proporciona una tasa de fotogramas configurables desde 5 hasta 45fps, el fabricante proporciona 8 modos de configuración recomendados para cada función. El usuario de la cámara puede ajustar estos parámetros según sus necesidades, teniendo en cuenta que, a mayor tasa de muestreo, menor es la distancia máxima que puede captar la cámara.

Modos

A continuación, se muestran en la Tabla 4 los modos de uso configurados por el fabricante:

Modo	Nombre	Aplicación Recomendada	Rango (m)	Tasa de Fotogramas (fps)	Tiempo de Exposición Máximo (us)
1	MODE_9_ 5FPS_2000	Reconstrucción de habitación interior	1 - 4	5	2000
2	MODE_9_ 10FPS_1000	Escaneo de habitación interior	1 - 4	10	1000
3	MODE_9_ 15FPS_700	Reconstrucción de objetos 3D	0.5 - 1.5	15	700
4	MODE_9_ 25FPS_450	Reconocimiento de objetos medianos o reconstrucción facial	0.3 - 2	25	450
5	MODE_5_ 35FPS_600	Colaboración remota	0.3 - 2	35	600
6	MODE_5_ 45FPS_500	Reconocimiento de objetos pequeños o seguimiento manual	0.1 - 1	45	500
7	MODE_ MIXED_30_5	Mezcla modos		30/5	300/1300
8	MODE_ MIXED_50_5	Mezcla modos		50/5	250/100

Tabla 4: Modos de configuración de la cámara CamBoard pico flexx

3.2 Recursos Software

Este TFG se ha implementado en el sistema operativo proporcionado por NVIDIA, derivado de Ubuntu haciendo uso del lenguaje de programación Python. El uso de este lenguaje se debe a la sencillez con respecto a otros lenguajes utilizados para sistemas de visión artificial como pueden ser C++ o ROS [56]. En la creación de los algoritmos se han utilizado las librerías descritas a continuación.

3.2.1 Bibliotecas

OpenCV

OpenCV [3] es una librería desarrollada por Intel, considerada como la biblioteca más popular en el ámbito de la visión artificial. Es una herramienta de código abierto y multiplataforma que ofrece cientos de funciones que abarcan un gran gama de tareas como son la detección de movimientos, calibración de cámaras, reconocimiento facial entre otras.

Si bien en el anteproyecto se propuso el uso de la biblioteca OpenCV [3], a medida que se estudiaron las librerías disponibles para nubes de puntos en Python se descartó su uso.

Figura 45: OpenCV [4]

Open3D

Open3D [5], es la principal herramienta de procesado de nube de puntos que se ha utilizado en este TFG. Es una biblioteca de código abierto con compatibilidad tanto para Python como para C++. Al tener apenas tres años de vida, a diferencia de otras librerías más populares no dispone de un gran número de ejemplos y experiencias de otros usuarios. Sin embargo, la buena estructura de su documentación, la compatibilidad con arquitecturas ARM, las múltiples herramientas para el procesado de datos tridimensionales y su código limpio e intuitivo han sido esencial para su elección.

Pese a que por ahora no sea compatible con arquitecturas ARM, cuenta con un módulo dedicado a las tareas relacionadas con el aprendizaje automático que permite la compatibilidad con otros frameworks como son TensorFlow [57] y PyTorch [58].

Su instalación en Jetson Nano requiere de una serie de procesos para poder dotar de compatibilidad con arquitecturas ARM y habilitar el uso del módulo CUDA para

obtener mejores resultado. En el Anexo 1. Guía de instalación, se encuentra detallada una guía para su instalación. Su documentación se encuentra disponible en [59].

Figura 46: Open3D [59]

Trimesh

Trimesh [60] es una biblioteca y conjunto de herramientas para leer, escribir y manipular mallas triangulares. Se encuentra disponible solo para Python ofreciendo disponibilidad para versiones superiores a Python 2.7. Se ha utilizado en este proyecto para leer y muestrear las mallas con los puntos deseados.

Figura 47: Trimesh [60]

TensorFlow

TensorFlow [57] es una biblioteca de código abierto creada por Google para computación numérica y orientada a la creación de modelos de Machine Learning. La API de Python es la más estable, sin embargo, también ofrece compatibilidad para C++, Java y Go.

La elección de esta librería se debe a su posibilidad de ejecución de manera eficiente en múltiples dispositivos (desde móviles hasta grandes centros de procesamiento de datos), aspecto clave en este proyecto. Cabe destacar la gran popularidad que posee, lo que le ha permitido tener una gran comunidad de usuarios activos que colaboran con ella.

Además, existe una versión TensorFlow Lite cuyos modelos son más pequeños, más rápidos y menos costosos computacionalmente permitiendo ser ejecutado en dispositivos con menos recursos (teléfonos, dispositivos embebidos, etcétera). Sin embargo, esta versión no permite el entrenamiento del sistema.

Figura 48: TensorFlow [57]

Keras

Es una biblioteca de Python diseñada para trabajar con redes neuronales de forma rápida, con el único requerimiento de un motor computacional que corra debajo. Fue creada en el año 2015 por un ingeniero de Google, permitiendo correr sobre diversas bibliotecas de Deep Learning. En el año 2017, se convirtió en parte de la librería TensorFlow [57], aunque continúa ofreciendo soporte a bibliotecas como Theano o Microsoft CNTK.

Su elección se debe a la compatibilidad con la librería TensorFlow donde permite implementar una red neuronal en pocas líneas gracias a su modularidad, minimalismo y extensibilidad.

Figura 49: Keras [57]

3.2.2 Otras librerías

API Royale (libroyale)

El fabricante de la cámara ToF utilizada, proporciona en el SDK de las cámaras una herramienta de visualización llamada RoyaleViewer y una API denominada Royale junto a su correspondiente documentación. La información provista permite al usuario adaptar la API a su plataforma y lenguaje correspondiente, calibrar y entender el funcionamiento de la cámara con scripts de ejemplos e información sobre los métodos y clases disponibles. El uso de esta librería permitió la configuración y capturación con la cámara.

NumPy

Es una librería de Python que añade un mayor soporte para vectores y matrices. Posee un gran número de funciones matemáticas de alto nivel, aportando así mayor rapidez y eficiencia a la hora de trabajar con este tipo de estructura de datos. Esta librería se utilizó en distintas ocasiones, destacando su uso para el manejo de los datos resultantes de las capturas.

Matplotlib

Es una librería para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación de Python, además, ofrece compatibilidad con la librería NumPy [61]. Esta librería se utilizó para la representación en 2D de los puntos y la creación de gráficas.

Scikit-learn

Es una biblioteca de software libre para Python para aprendizaje automático que contiene algoritmos de gran popularidad. Ha sido creada sobre las bibliotecas NumPy, SciPy y matplotlib propias de Python. Surgió como un proyecto de Google que ha ido evolucionando hasta la actualidad donde posee una gran comunidad de contribuidores [62]. En este TFG se ha utilizado para realizar los reportes relacionados con en el entrenamiento y la clasificación.

Seaborn

Es una librería de Python basada en la biblioteca matplotlib y cuya finalidad es la visualización de datos [63]. Ha sido utilizada para mostrar la matriz de confusión.

os

El módulo os nos ha permitido realizar operaciones mediante Python que son dependientes del sistema operativo.

glob

El módulo glob encuentra todos los nombres de ruta que coinciden con un patrón especificado de acuerdo con las reglas utilizadas por el shell de Unix, aunque los resultados se devuelven en orden arbitrario.

argparse

La librería argparse de Python fue utilizada para crear interfaces de línea de comandos con la misión de facilitar la ejecución de los scripts.

queue

El módulo queue ha sido usado para crear y trabajar con colas de manera sencilla, permitiendo intercambiar información entre hilos de manera segura.

time

El módulo time pertenece a librería estándar de Python, ha posibilitado conocer el tiempo real para hacer cálculos de tiempo.

Capítulo 4 – Algoritmos y datasets

4.1 Consideraciones previas y solución diseñada

El objetivo principal del proyecto es la detección e identificación de obstáculos en las nubes de puntos captadas por la CamBoard pico flexx y procesadas en el computador NVIDIA Jetson Nano.

Para conseguir el objetivo se ha estudiado los métodos actuales de reconocimiento de objetos (2.2 Reconocimiento de objetos), en especial los métodos basados en Deep Learning (2.3 Deep Learning). En lo que incumbe a este trabajo, el estado del arte de las tareas de aprendizaje profundo con nubes de puntos ha sufrido una revolución tras la publicación de PointNet. Por ello, se ha escogido como el enfoque a seguir para el reconocimiento de objetos en este TFG.

Teniendo en cuenta que la estructura general de los sistemas de visión artificial es la abordada en 2.1 Visión Artificial y considerando las características del aprendizaje profundo seleccionadas para este trabajo, podemos rediseñar la Figura 2 siguiendo el siguiente proceso:

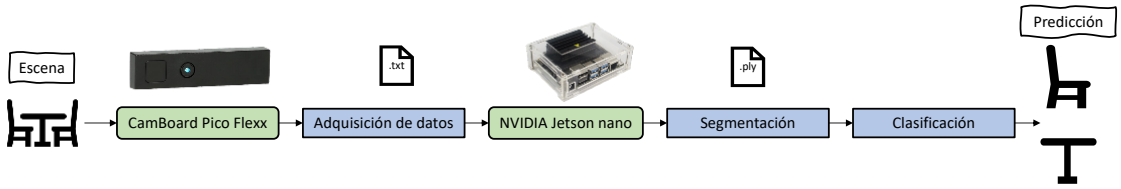


Figura 50: Sistema de visión artificial implementado

La captura de datos se realiza mediante la CamBoard pico flexx. La cámara, envía la información de la escena en tiempo real al computador donde puede ser mostrada o procesada.

La nube de puntos adquirida por la cámara posee más de treinta mil puntos por lo que necesita un procesado, para ello se han utilizado varios algoritmos para reducir el tamaño de la nube, con la finalidad de encontrar objetos de interés.

Una vez detectados los posibles objetos de interés se procede a clasificarlos mediante el modelo entrenado siguiendo el enfoque PointNet.

Para poder entrenar el modelo ha sido necesario crear un conjunto de datos, también conocido como dataset, con los objetos de interés para nuestra tarea de clasificación. La solución aportada pretende que el sistema compare las nubes captadas por la cámara con las almacenadas en la base de datos. Dada las características tridimensionales de la captura, para dotar de mayor precisión al sistema, ha sido necesario tomar capturas de los objetos desde diferentes ángulos y posiciones. A continuación, en la Figura 51, se detalla un esquema con los algoritmos y conjuntos de datos creados donde las cajas azules representan los scripts creados y las cajas verdes los datasets.

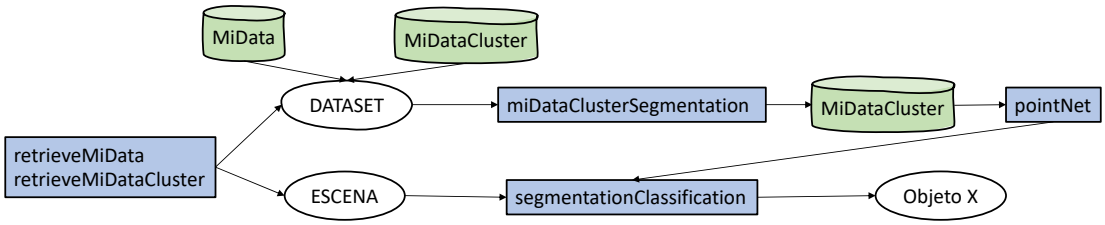


Figura 51: Esquema del sistema creado

4.2 Algoritmos

Se han desarrollado cinco algoritmos para este TFG, dos para la adquisición de datos, uno para la etapa de entrenamiento y dos para la fase de detección y clasificación.

4.2.1 Adquisición de datos

Se han creado dos algoritmos de adquisición de datos, uno para cada dataset creado. No obstante, se realizaron algunas pruebas iniciales para conseguir la mejor calidad de captura posible.

Estudio del mejor modo de capturación

Como se comentó en el Capítulo 3, la cámara posee 9 modos de configuración para captar la escena. Se procedió a analizar capturando un objeto desde distintas posiciones y distancias, obteniendo el resultado que se muestra a continuación:

Captura	Modo	Aplicación Recomendada	Rango (m)	Resultado
1	MODE_9_ 5FPS_2000	Reconstrucción de habitación interior	1 – 4	Buen resultado
2	MODE_9_ 10FPS_1000	Escaneo de habitación interior	1 – 4	Regular
3	MODE_9_ 15FPS_700	Reconstrucción de objetos 3D	0.5 - 1.5	Al aumentar la distancia no capta bien el objeto.
4	MODE_9_ 25FPS_450	Reconocimiento de objetos medianos o reconstrucción facial	0.3 – 2	Al aumentar la distancia no capta bien el objeto.

Tabla 5: Comparación de los resultados obtenidos por los métodos de adquisición

Atendiendo a la información del fabricante de la cámara, se testearon los modos configurados que cubren la distancia de la escena requerida con múltiples objetos desde distintos perfiles y distancias. Se concluye que el Modo 1: "**MODE_9_5FPS_2000**", consigue una mayor calidad en la captura de datos para la escena y tarea en cuestión.

Valor de confianza vs nube original

En el Capítulo 3, se mostró la información que puede captar la CamBoard pico flexx. Se ha utilizado el valor de confianza de los puntos adquiridos para disminuir el tamaño de la nube de puntos. El valor de confianza con valor 0, indica que la calidad del punto no es deseada, por lo que eliminando estos puntos eliminamos información residual de nuestra captura. Este filtrado favorece los cálculos posteriores teniendo menor cantidad de puntos y por lo tanto menor coste computacional.

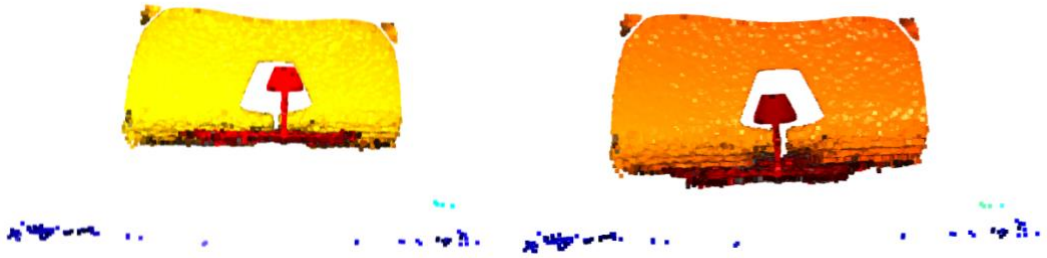


Figura 52: Comparación entre una nube original y una nube filtrada por valor de confianza

La imagen superior izquierda muestra una nube de 38304 puntos procedente de la capturación de una escena. A la derecha, esa nube ha sido filtrada eliminando los puntos cuyo valor de confianza es 0 obteniendo como resultado una nube de 33177 puntos. Pese a que a simple vista sea muy similar, en este caso, supone una reducción del 13,4% de los puntos de la nube.

Algoritmo de capturación

Una nube de puntos es una estructura de datos utilizada a menudo para representar de forma inmediata la información tridimensional, así como del mapa de profundidad, elaborado en imágenes RGB-D. Por ello, para dotar de toda información útil a nuestra base de datos, el algoritmo encargado de capturar la escena nos devuelve archivos .txt con los que podemos obtener las nubes de puntos e imágenes de profundidad.

Para realizar este script ha sido necesario utilizar el SDK de la cámara y las librerías argparse, time, queue, numpy y matplotlib propias de Python. Mediante los argumentos indicados en la consola de Python, el script captura durante un intervalo de tiempo --seconds, con la configuración del modo --mode, para finalmente almacenar el resultado obtenido en el directorio indicado en los parámetros: --directorio/--clase_objeto/--objeto/--n_captura.

El script se encarga de activar la cámara y procesar los datos con la librería numpy, almacenar los archivos y finalmente dibujar las imágenes de profundidad obtenidas con matplotlib. El funcionamiento completo se puede describir mediante el siguiente diagrama de flujo:

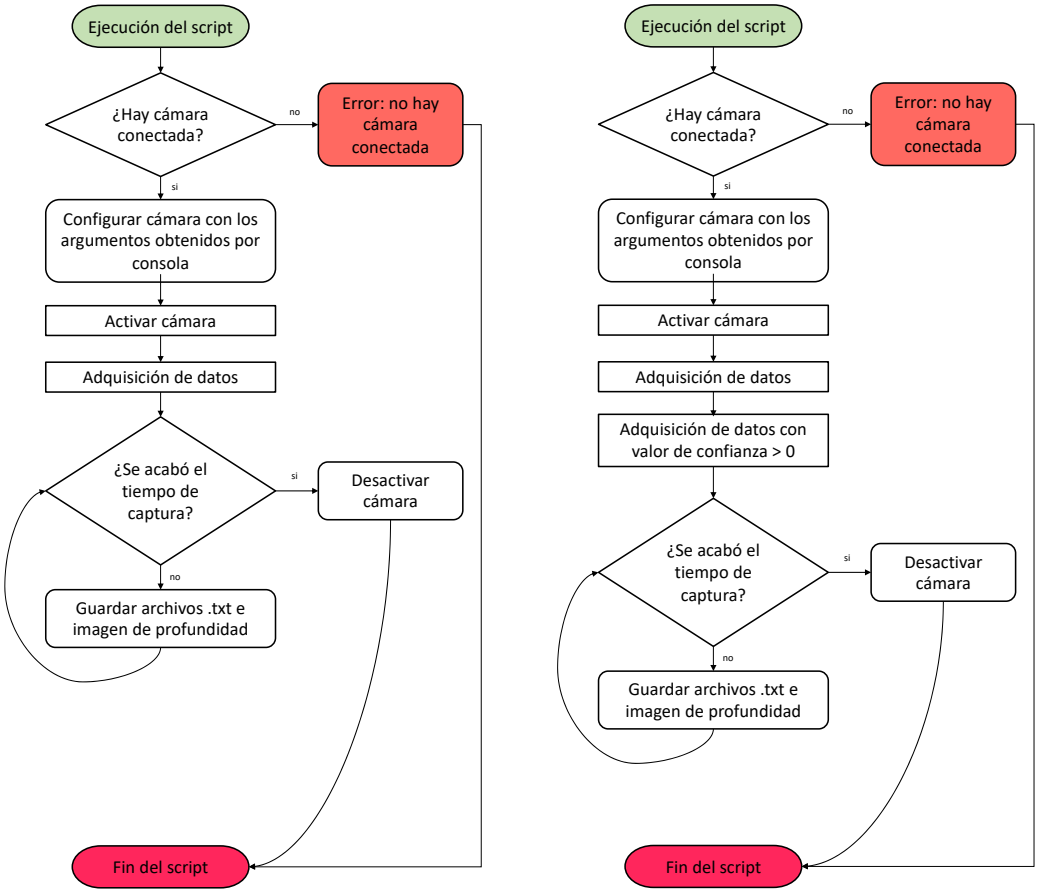


Figura 53: Diagrama de flujos de los scripts: retrieveMiData.py y retrieveMiDataCluster.py

Se puede apreciar como en el diagrama del script retrieveMiDataCluster.py, además de obtener las matrices originales, realiza un filtrado para obtener las matrices cuyo valor de confianza es mayor a 0.

4.2.2 Preprocesado y segmentación

Una vez capturada la escena y filtrada la matriz según el valor de confianza de sus puntos, se procede a convertir en nube de puntos y procesarla mediante la librería Open3D. La finalidad del procesado es limpiar las nubes hasta poder identificar posibles objetos para posteriormente añadirlos al dataset o clasificarlos. A continuación, se detallan los pasos seguidos:

Filtrado de fondo

En primer lugar, se realiza la segmentación del plano predominante en la escena haciendo uso del algoritmo RANSAC (Random Sample Consensus). Este plano suele coincidir con el fondo de la imagen o el suelo, permitiendo eliminar gran parte de la nube que no es de nuestro interés. En las figuras adjuntadas a continuación se observan ambos casos.

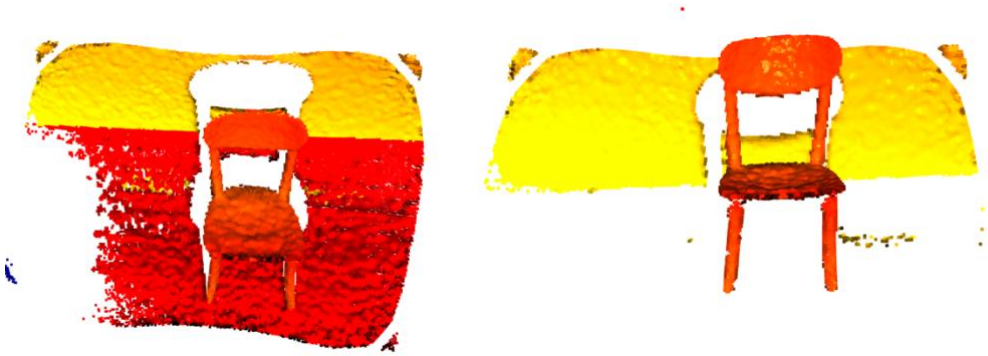


Figura 54: Resultado de la segmentación del suelo de la nube de puntos

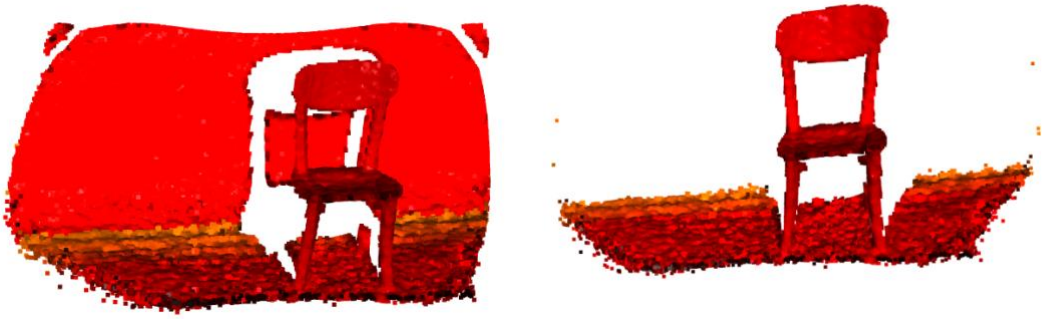
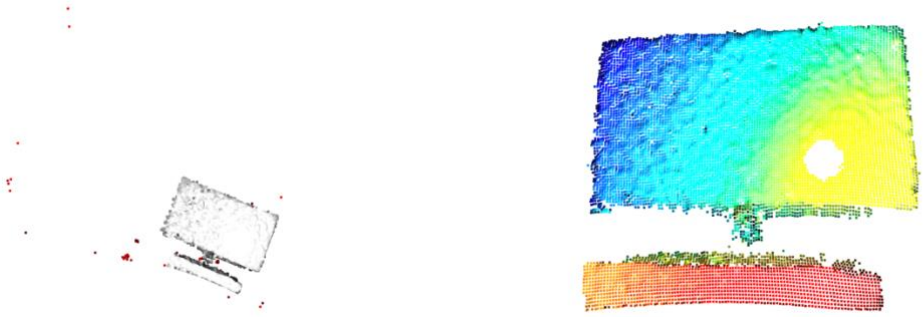


Figura 55: Resultado de la segmentación de la pared de una nube de puntos

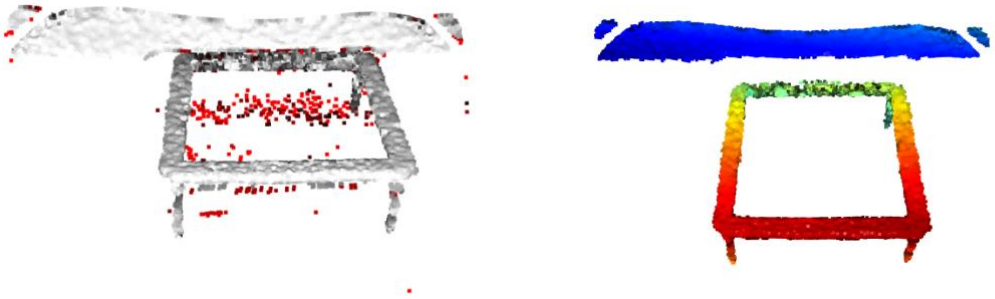
Filtrado de ruido

Tras eliminar el plano predominante, quedan algunos puntos perdidos en la nube que pueden llevar a confusión a los procedimientos posteriores. Por ello se realiza un filtrado de ruido mediante un método que elimina los puntos que no poseen 16 puntos vecinos dentro del radio indicado.



a) *b)*

Figura 56: Resultado de la segmentación del ruido de la nube de puntos de un monitor según un radio dado



a) *b)*

Figura 57: Resultado de la segmentación de la nube de puntos de una mesa con cristal según un radio dado

Las figuras 55 y 56, están compuestas por dos imágenes. La imagen a) destaca en color rojo, los puntos detectados como ruido para nuestro sistema. La figura b) muestra el resultado de la nube sin este ruido, ofreciendo una visualización más nítida del objeto de interés.

Agrupación de la nube

Los puntos resultantes son agrupados mediante el algoritmo DBSCAN. Cada punto es etiquetado en función de su densidad como se aprecia en la Figura 58.

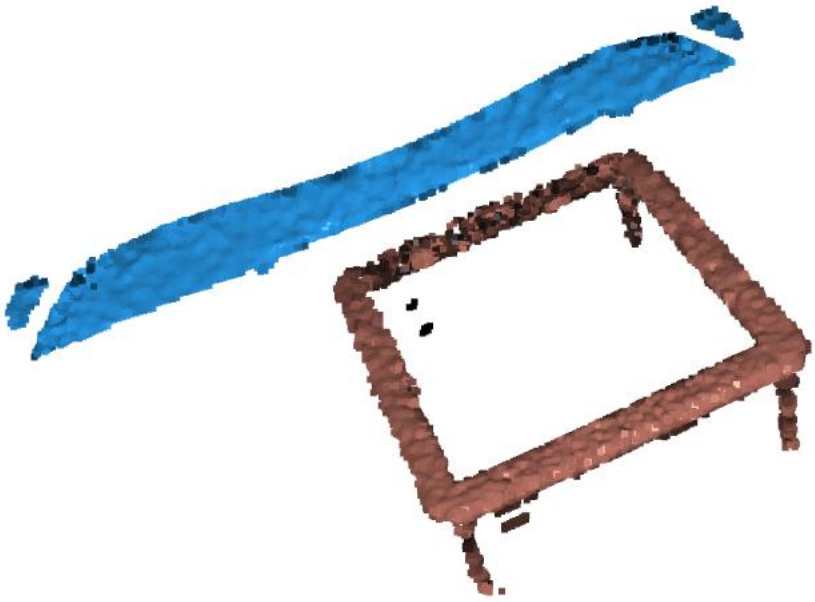


Figura 58: Nube de puntos resultante de la agrupación mediante DBSCAN

Filtrado de tamaño

Los clústeres obtenidos se someten a un filtrado teniendo en cuenta su tamaño con respecto a un umbral deseado. En la Figura 59 se ha filtrado la nube de la Figura 58 obteniendo como resultado el cluster deseado.

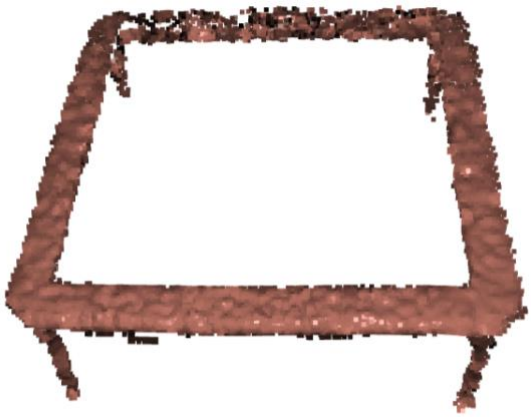


Figura 59: Cluster obtenido tras el filtrado según el umbral de puntos

Una vez llegado a este paso, dependiendo del script ejecutado, el objeto es añadido al conjunto de datos o es clasificado. La clasificación será vista en el próximo capítulo, tras lo visto en este apartado, podemos concluir con que el diagrama de flujo del script miDataClusterSegmentation.py queda de la siguiente manera:

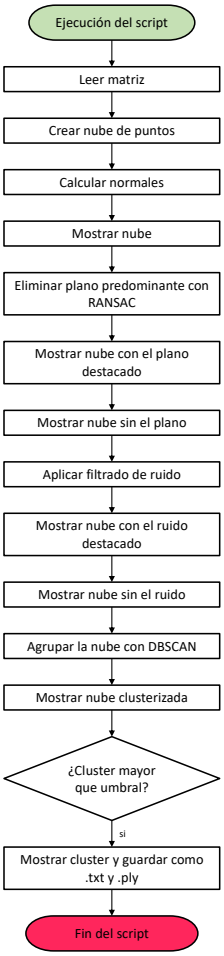


Figura 60: Diagrama de flujo del script miDataClusterSegmentation.py

4.3 Datasets

Se han utilizado tres datasets para comprobar los algoritmos creados. Dos, han sido capturados manualmente con el hardware disponible para este proyecto, el restante ha sido descargado de Internet.

4.3.1 MiData

El dataset MiData ha sido utilizado para observar el comportamiento del algoritmo de segmentación con distintos objetos, en distintos perfiles y distancias. Dentro de las tareas de comprobación del funcionamiento del script, se ha testeado dos métodos de eliminación de ruido distintos y ajustado los parámetros hasta obtener el resultado óptimo para el tipo de escena que estamos capturando.

El conjunto de datos está compuesto por 7 clases que contienen un total de 21 objetos. De cada objeto se ha realizado 15 capturas donde el computador ha obtenido entre 1-4 frames por captura en el tiempo indicado. Esto ha permitido tener un conjunto de aproximadamente 630 nubes de puntos.

De cada frame se han almacenado 4 archivos en formato .txt con las coordenadas espaciales de cada punto. Los datos espaciales de la nube nos permiten realizar múltiples operaciones. En este caso, con la **matrizXYZ** hemos creado las nubes de puntos y la **matrizZ** ha sido utilizada para crear una imagen de profundidad por cada frame. A continuación, se describen y muestran cada uno de los archivos almacenados:

- **matrizXYZ**: Array float estructurado de forma (n,3) con las coordenadas (x,y,z) de cada punto de la nube.

```
0.00000000000000000e+00 -0.00000000000000000e+00 -0.00000000000000000e+00
0.00000000000000000e+00 -0.00000000000000000e+00 -0.00000000000000000e+00
-8.6453825235366821 29e-01 6.91688358783721923 8e-01 -1.55290126800537109 4e+00
-8.6478060483932495 12e-01 6.98274016380310058 6e-01 -1.56656527519226074 2e+00
-8.8435053825378417 97e-01 7.20734179019927978 5e-01 -1.61617529392242431 6e+00
-8.5091972351074218 75e-01 7.00015783309936523 4e-01 -1.56939625740051269 5e+00
-8.3543640375137329 10e-01 6.93808495998382568 4e-01 -1.55568599700927734 4e+00
-8.2570654153823852 54e-01 6.92306101322174072 3e-01 -1.55316770076751709 0e+00
-8.1851178407669067 38e-01 6.92920804023742675 8e-01 -1.55621874332427978 5e+00
-8.1959402561187744 14e-01 7.00622916221618652 3e-01 -1.57630503177642822 3e+00
```

Figura 61: Estructura de los archivos .txt que forman las nubes de puntos

- **matrizX**: Array float de estructura (n, 1) que contiene las coordenadas X de cada punto.

```
0.00000000000000000e+00
0.00000000000000000e+00
-8.6453825235366821 29e-01
-8.6478060483932495 12e-01
-8.8435053825378417 97e-01
-8.5091972351074218 75e-01
-8.3543640375137329 10e-01
-8.2570654153823852 54e-01
-8.1851178407669067 38e-01
-8.1959402561187744 14e-01
```

Figura 62: Estructura de los archivos que almacenan las coordenadas del eje X

- **matrizY**: Array float de estructura (n, 1) que contiene las coordenadas Y de cada punto.

```
-0.00000000000000000e+00
-0.00000000000000000e+00
 6.91688358783721923e-01
 6.98274016380310058e-01
 7.20734179019927978e-01
 7.00015783309936523e-01
 6.93808495998382568e-01
 6.92306101322174072e-01
 6.92920804023742676e-01
 7.00622916221618652e-01
```

Figura 63: Estructura de los archivos que almacenan el eje Y

- **matrizZ**: Array float de estructura (n, 1) que contiene las coordenadas Z de cada punto.

```
-0.00000000000000000e+00
-0.00000000000000000e+00
-1.55290126800537109e+00
-1.56656527519226074e+00
-1.61617529392242432e+00
-1.56939625740051270e+00
-1.55568599700927734e+00
-1.55316770076751709e+00
-1.55621874332427979e+00
-1.57630503177642822e+00
```

Figura 64: Estructura de los archivos que almacenan el eje Z para crear imágenes de profundidad

- **ImagenRGBD**: Imagen de profundidad realizada con la matriz de valores Z de cada frame.

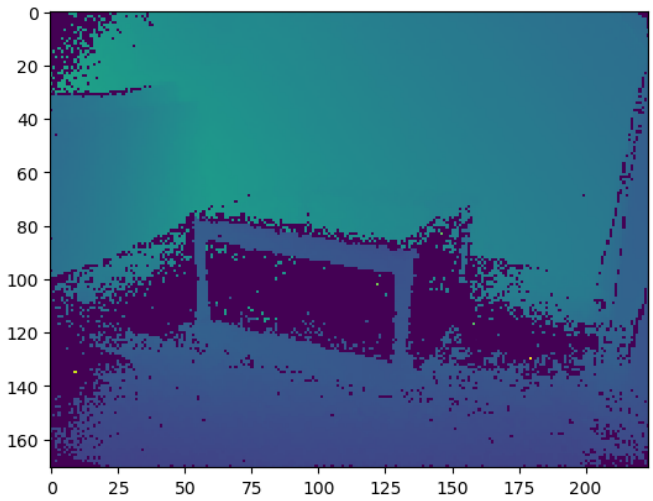


Figura 65: Imagen de profundidad

De cada frame se crea una imagen de profundidad (2D) como la adjuntada, donde se proporciona la información de cada punto, para ello se utiliza la **matrizZ** capturada

del frame. A continuación, se muestra un ejemplo de una nube perteneciente a este dataset:

Figura 66: Nube de puntos modelo de MiData

4.3.2 MiDataCluster

MiDataCluster, ha sido utilizado para comprobar la diferencia con los scripts creados entre una nube original o una nube filtrada, atendiendo el valor de confianza de sus puntos.

Dada la necesidad de segmentar la nube hasta obtener el objeto, se ha realizado un dataset más simple capturando 5 clases de un total de 13 objetos. La segmentación se ha realizado con el algoritmo miDataClusterSegmentation.py visto anteriormente.

Al igual que en el anterior dataset, se han realizado 15 capturas de cada objeto donde se han obtenido entre 1-4 frames de cada captura, completando un total de 622 capturas. Sin embargo, en este conjunto, de cada frame se ha almacenado una imagen de profundidad y dos archivos .txt que nos permiten crear nubes de puntos:

- **matrizXYZ**: Un array float estructurado de forma (n,3) con las coordenadas (x,y,z) de cada punto de la nube.

- **matrizXYZConf**: Un array float estructurado de forma (n,3) con las coordenadas (x,y,z) de cada punto de la nube cuyo valor de confianza es mayor a 0.

- **ImagenRGBD**: Imagen de profundidad realizada con la matriz de valores Z de cada frame.

Una vez comprobado el mejor funcionamiento de las nubes filtradas por valor de confianza, se ha decidido usarlas en el sistema para segmentar, entrenar, detectar y clasificar. La Figura 67, es un cluster perteneciente a este conjunto.

Figura 67: Nube de puntos modelo de miDataCluster

4.3.3 Modelnet10

Modelnet10 es un dataset muy popular que forma parte del proyecto Princeton ModelNet [64], encargado de proveer a investigadores de visión artificial, modelos de objetos tridimensionales.

ModelNet10 contiene 4899 formas pertenecientes a 10 categorías de objetos comunes en interiores (camas, sillas, mesas, escritorios, monitores, retretes, tocadores, bañeras, sofás y mesillas) el 80% de estas, se han seleccionado para entrenamiento y el 20% restante para testeo.

Figura 68: Modelos de cada clase perteneciente a ModelNet10 [64]

La sencillez de los objetos, observables en la Figura 68, disponibles en esta base de datos permite solucionar tareas como el reconocimiento de objetos, segmentación o estimación de pose.

Sin embargo, los modelos proporcionados están en formato Object File Format (.off), cuyo formato almacena las figuras como mallas poligonales descritas mediante caracteres ASCII. Con el uso de la librería Open3D, procedemos a muestrearlas y convertir las mallas en nubes, de N puntos (2048 por defecto).

Este conjunto ha sido utilizado para comprobar el entrenamiento y clasificación del sistema con objetos que poseen una modelización perfecta dada la dificultad de reconstruir una estructura completa del objeto con el hardware de este proyecto.

Capítulo 5 – Entrenamiento y clasificación

Como hemos visto en los apartados anteriores, el enfoque escogido para conseguir la tarea de reconocimiento, PointNet, ha sido desarrollado haciendo uso de las librerías TensorFlow y Keras. El algoritmo resultante, es proceso de una modificación del código original de PointNet [24] [65] y la documentación de Keras [66][67].

5.1 Algoritmos

Para poner en práctica las tareas de entrenamiento y clasificación, se han desarrollado dos scripts, pointNet.py y segmentationClassification.py. En este apartado se encuentra detallada la utilidad y el funcionamiento de ambos.

5.1.1 Entrenamiento

El script PointNet.py es el encargado de crear, entrenar y evaluar el modelo. Además, realiza algunas predicciones para verificar su utilidad. A continuación, se desarrollará paso por paso su funcionamiento, para ello, se tuvo en cuenta que el ciclo de vida de un modelo de aprendizaje profundo consta de las siguientes etapas [68]:

1. **Definir el modelo**
2. **Compilar el modelo(**.*compile***)**
3. **Entrenar el modelo**(.*fit*)
4. **Evaluar el modelo(**.*evaluate***)**
5. **Hacer predicciones(**.*predict***)**

Preparar Hardware

En primer lugar, se ha de preparar el hardware, en caso de ejecutar el script pointNet.py en un computador escaso de memoria, como es el ordenador Jetson Nano, el dispositivo se congelará debido a que no será capaz de almacenar la memoria necesaria. Para ello, si el usuario lo desea, puede limitar el uso de la memoria GPU a 1024MB. Esta simple limitación permitirá realizar el entrenamiento satisfactoriamente, sin embargo, el proceso será más lento.

Preparar Dataset

El algoritmo recorre el dataset indicado con la finalidad de cargar los datos de entrenamiento y validación. Conocidas las características de irregularidad y desestructuración de las nubes, para un correcto funcionamiento del sistema, es necesario que todas posean el mismo número de puntos.

Los datos en el dataset MiDataCluster, se encuentran como nubes de puntos en formato .ply, por lo que son convertidas a mallas para ser muestreadas con la cantidad de

puntos deseados. El conjunto ModelNet10 al estar en formato de malla sólo necesita pasar por el proceso de muestreo.

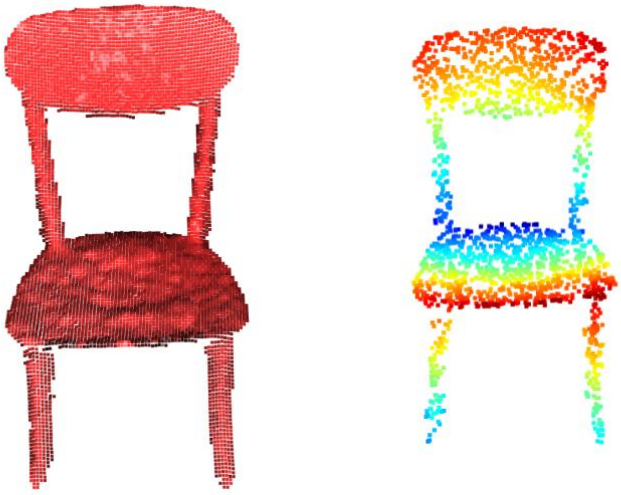


Figura 69: Silla de MiDataCluster muestreada de 5647 puntos a 2048

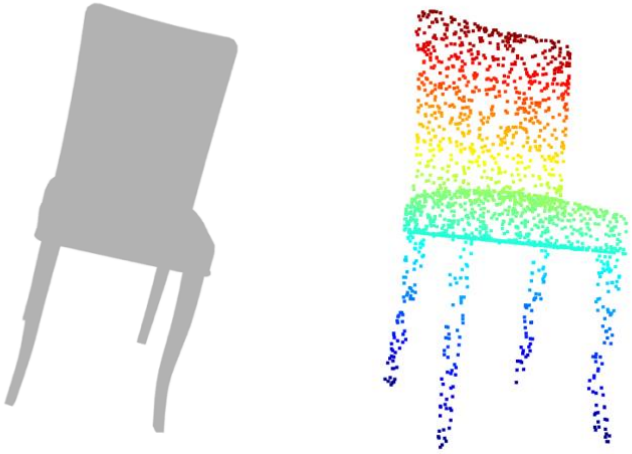


Figura 70: Silla de ModelNet10 muestreada a 2048 puntos

En la Figura 70 y Figura 70, la imagen de la izquierda muestra las capturas originales, las pertenecientes a los datasets propios se encuentran en nube de punto y la de ModelNet10 en malla, ambas son muestreadas con la misma cantidad de puntos para entrar en la red. En las imágenes de la derecha se puede observar la similitud en los resultados pese a que la imagen de MiDataCluster no haya recibido a un modelado tan detallado como ModelNet10.

Tras recolectar los puntos y etiquetas pertenecientes a cada nube, se almacenan según la finalidad del conjunto; entrenamiento o testeo. Una vez almacenados los datos, se muestran los 15 primeros elementos del conjunto de entrenamiento, como se muestra en la Figura 71, para verificar que se han cargado y se visualizan correctamente.

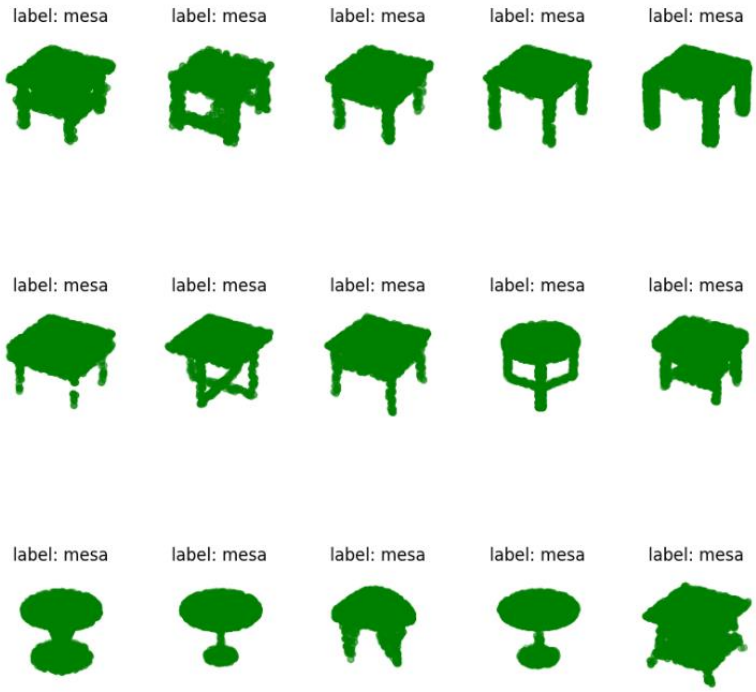


Figura 71: Elementos del dataset de entrenamiento

Con los puntos y etiquetas almacenados se conforman los datasets de entrenamiento y testeo. Se añade ruido aleatorio a estos datasets para asegurarnos de que la representación de los datos no esté sesgada a ningún orden y mantenga así las características de la nube de puntos. Esta acción previene el sobreajuste en el conjunto de datos. Tras este proceso, el conjunto de datos ya se encuentra preparado para comenzar el entrenamiento.

PointNet

La siguiente etapa es crear el modelo y arquitectura necesaria. Es decir, elegir las capas, configurar el número de nodos y función de activación necesaria e interconectar las capas en un modelo cohesivo. Para llevarlo a cabo se ha seguido el modelo presentado en el Capítulo 2, replicando el modelo original de PointNet [65]. Sin embargo, la arquitectura utilizada en este trabajo posee la mitad del número de pesos en cada capa, ya que estamos utilizando un conjunto de datos más pequeño que el utilizado en la investigación original.

Como se ha comentado en el Capítulo 2, todas las capas de PointNet están totalmente interconectadas, a excepción de la capa final. Recordamos que los bloques principales de la arquitectura de este enfoque son las redes MLP y T-Net. La red T-Net es utilizada en dos ocasiones: la primera vez transforma la entrada (n, 3) en una representación canónica y la segunda es una transformación afín para la alineación en el espacio de entidades. Según [24] se puede limitar como una matriz ortogonal. Por ello,

se define en el script una clase regularizadora ortogonal, que facilita la arquitectura de una función general para construir capas T-net.

Los perceptrones multicapas, excepto el último, están formados por una capa convolucional o densa, un operador de normalización de batch y la función de activación ReLU.

Tras definir los elementos pertenecientes a la arquitectura de la red, ya podemos crear la CNN. En primer lugar, es necesario definir la capa de entrada de la red, mostrada en la Figura 72, encargada de representar los puntos de la escena. Esta constituida por la cantidad de puntos deseados (2048 por defecto) y sus correspondientes coordenadas tridimensionales.

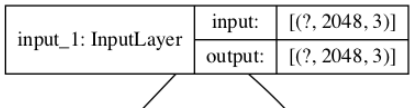


Figura 72: Entrada del modelo de PointNet.py

La entrada es transformada por la red T-net, Figura 73, encargada de convertir los puntos de entrada en un espacio regular con la finalidad de adaptarlo a un procesado secuencial de arquitecturas de aprendizaje profundo similar al de las redes de píxeles.

Esto se realiza mediante la regularización ortogonal que consigue proyectar la nube en forma de matriz. En conclusión, T-net, se encarga de representar las nubes de puntos en un espacio de características más común para técnicas de aprendizaje profundo.

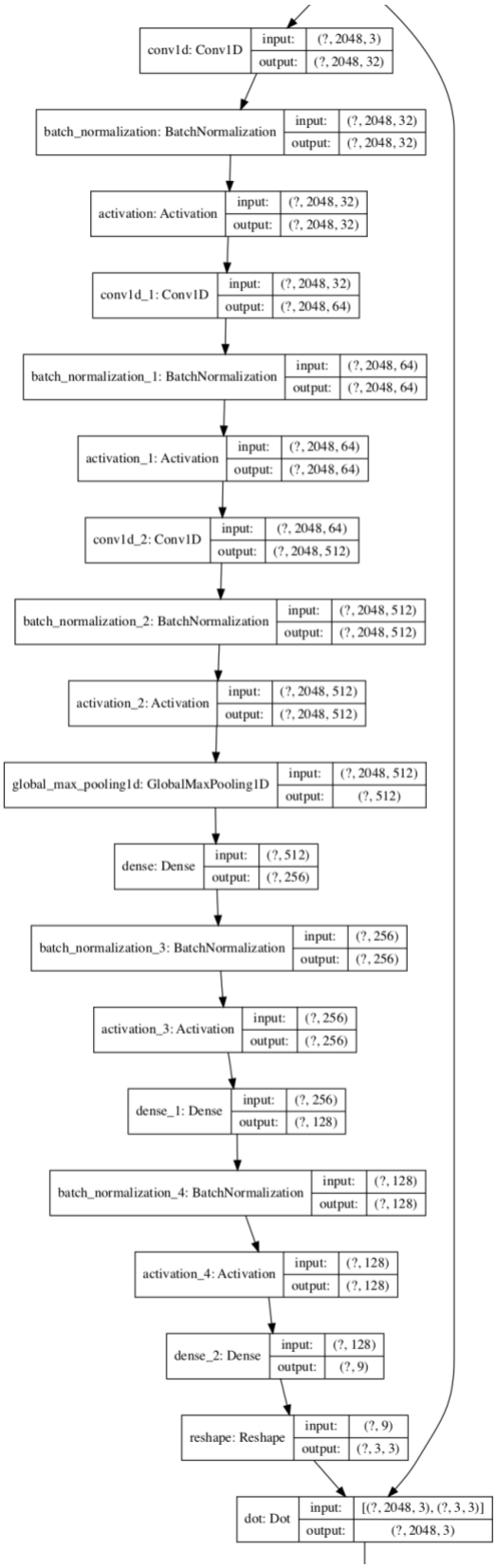


Figura 73: Diagrama de la primera transformación de T-net en pointNet.py

La salida de T-net se introduce en un MLP formado por 2 capas de 32 filtros cada una, dedicados a la extracción de características. Cada convolución es de una dimensión, cuyo núcleo unidimensional provoca un aumento del espacio de características,

obteniendo 32 características de cada nube. La salida de este bloque aumenta la información de 2048 x 3 a 2048 x 32, visible en la Figura 74.

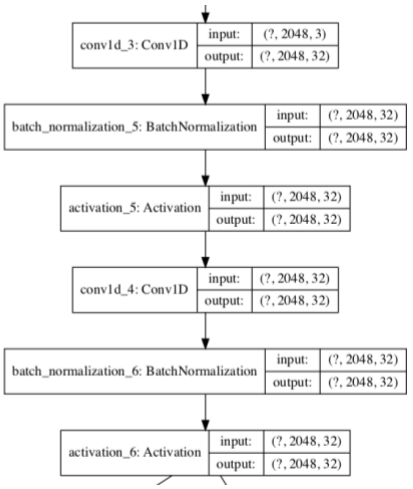


Figura 74: Diagrama del bloque MLP de 2 capas en pointNet.py

Las nuevas matrices son transformadas por otra red T-net con la finalidad de dotar de invariancia a cualquier transformación rígida.

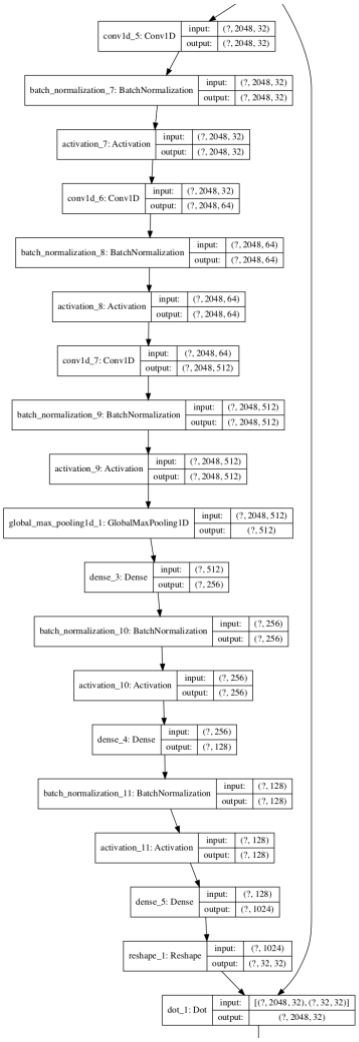


Figura 75: Diagrama de la segunda transformación de T-net en pointNet.py

Se repite el proceso realizado anteriormente, introduciendo las nubes transformadas en otro bloque MLP descrito en la Figura 76. Este bloque está formado por 3 capas de 32, 64 y 512 neuronas respectivamente, obteniendo usa salida de 512 características por nube.

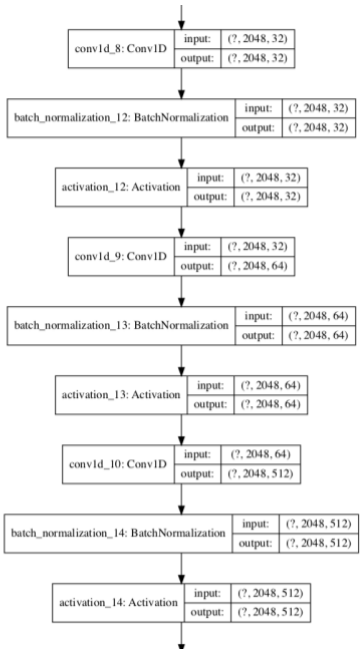


Figura 76: Bloque MLP de 3 capas en pointNet.py

Se aplica una función simétrica max pooling, para dotar a la nube de invariancia a permutaciones.

Figura 77: Capa max pooling en pointNet.py

La salida obtenida pasa por un último bloque MLP de tres capas fully-connected de tamaños de salida 256, 128 y k, donde k hace referencia a los resultados obtenidos en la predicción de los k objetos detectados.

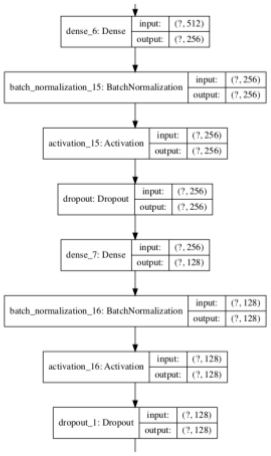


Figura 78: Bloque MLP de 3 capas fully-connected de pointNet.py

En las capas fully-connected, es utilizado dropout para evitar el sobreajuste. En este caso, con una probabilidad de permanencia de 70%, es decir, el 30% de las entradas se eliminarán en cada actualización del modelo.

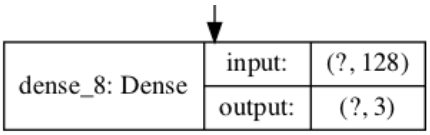


Figura 79: Salida del modelo pointNet.py de 3 clases

La salida esta condicionada al número de clases, dado que el número de neuronas de la salida debe de ser igual a la cantidad de clases existentes en el problema. La función de activación Softmax es utilizada para representar la salida en forma de probabilidades.

La estructura del modelo es presentada mediante el método *summary* y guardada en una imagen mediante el método *plot_model*. La imagen de la arquitectura ha sido utilizada para explicar la arquitectura del modelo.

Compilar modelo

Una vez definido el modelo, se ha de compilar. Al compilar, debemos especificar algunas propiedades requeridas al entrenar la red. Estas propiedades son: la selección de un algoritmo para realizar el procedimiento de optimización, una función de pérdida a optimizar y una métrica adecuada para poder evaluar el modelo correctamente.

Función de pérdida

La función de perdida mide la exactitud del modelo durante el entrenamiento, es necesario minimizar esta función para dirigir el modelo en la dirección adecuada. La función seleccionada para este modelo es **'sparse_categorical_crossentropy'**, considerada apropiada para etiquetado de clases codificadas en enteros (por ejemplo, 0 para una clase, 1 para la siguiente clase, etcétera). La función produce el índice de la categoría más probable tras calcular la pérdida de entropía cruzada entre las etiquetas y las predicciones.

Optimizador

El optimizador indica como el modelo se actualiza basado en el set de datos y la función de pérdida. El optimizador escogido, es el eficiente algoritmo de descenso de gradiente estocástico, **'adam'**. Es una opción muy popular porque se sintoniza automáticamente y produce buenos resultados en una amplia gama de problemas.

Métrica

Las métricas, son utilizadas para monitorear los pasos de entrenamiento y de pruebas. Puesto que tenemos un problema de clasificación, las métricas se encargan de recopilar e informar de la precisión obtenida en la clasificación. La métrica escogida es **'sparse_categorical_accuracy'**, cuyo funcionamiento se basa en calcular la tasa de

precisión media en todas las predicciones, lo que la convierte ideal para problemas de clasificación multiclase.

Entrenar modelo

Una vez creado el modelo y compilado para un cálculo eficiente, es necesario entrenarlo. El entrenamiento aplica el algoritmo de optimización elegido para minimizar la función de pérdida seleccionada y actualizar el modelo utilizando el algoritmo de retropropagación de errores. El ajuste del modelo es la parte lenta de todo el proceso, puede tomar de segundos a días, dependiendo de la complejidad del modelo, el hardware utilizado y el tamaño del conjunto de datos. Mientras se ajusta el modelo, una barra de progreso resume el estado de cada epoch y del proceso de entrenamiento general por consola.

El entrenamiento o ajuste del modelo se realiza mediante la función *fit*, es necesario seleccionar la cantidad de epochs para configurar el entrenamiento. Tras el entrenamiento, los pesos son guardados en el directorio indicado, para facilitar la carga del modelo en caso de que sea necesario posteriormente para hacer otras tareas como puede ser la predicción.

Evaluar modelo

Una vez entrenada la red con el conjunto de entrenamiento, se ha de evaluar el modelo creado con un conjunto de datos que no conozca. En caso de hacerlo con el conjunto de entrenamiento, se puede observar lo bien que se ha modelado el conjunto de datos, sin embargo, utilizando el conjunto de testeo permitirá comprobar el rendimiento del modelo para nuevos datos que no conoce. El resultado obtenido es una predicción imparcial del modelo sobre nuevos datos añadidos a la información de pérdida y precisión.

Las librerías Scikit-learn [62] y seaborn [63] han sido utilizadas en este apartado para obtener una mejor visualización del resultado.

Exactitud vs pérdida

Con los resultados de las métricas, se crean dos gráficas (ver Figura 80 y Figura 81) y se almacenan en forma de imágenes. Además, los resultados promedios son mostrados por consola.

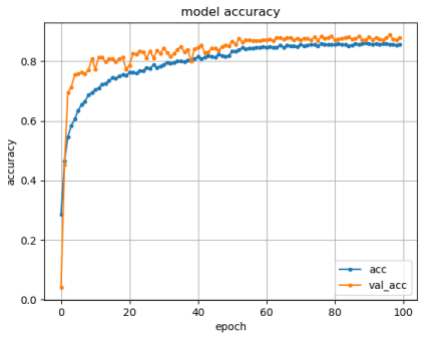


Figura 80: Gráfica de exactitud del entrenamiento con PointNet

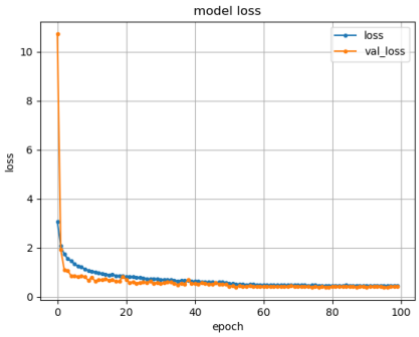


Figura 81: Gráfica de la pérdida en el entrenamiento con PointNet

La mayoría de las veces se observa que la precisión aumenta mientras disminuye la pérdida, pero este no siempre es el caso. La exactitud y la pérdida tienen diferentes definiciones y miden cosas distintas. A menudo, parecen ser inversamente proporcionales, pero no hay una relación matemática entre estas dos métricas.

Matriz confusión

La matriz de confusión, también conocida como matriz de error, es un elemento clave en el rendimiento de la clasificación. Es una visualización tabular de las predicciones del modelo frente a las etiquetas verdaderas. Cada fila de la matriz representa las instancias de una clase predicha y cada columna representa las instancias de una clase real. La matriz nos permite comprobar qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos. La diagonal de arriba a la izquierda a abajo a la derecha contiene las observaciones correctamente predichas.

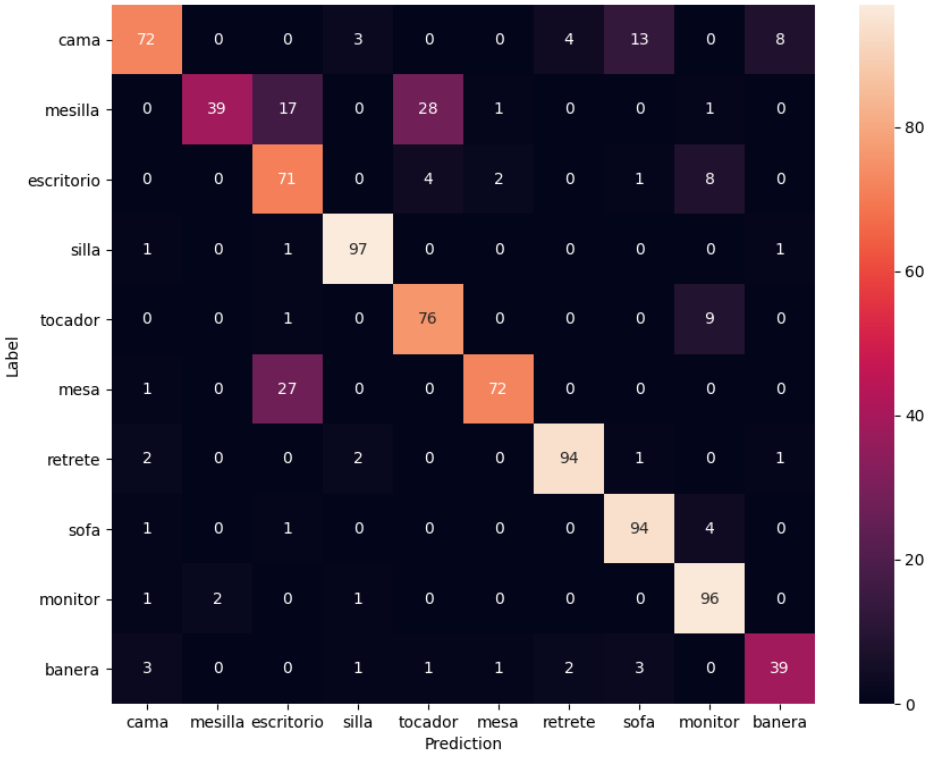


Figura 82: Matriz de confusión de 10 clases

La Figura 82 permite observar como el sistema falla entre objetos similares como son: las mesas y escritorios; retretes y sillas; mesillas y tocadores; sofás y camas. Estos fallos también podrían ser producidos por un humano dada la similitud de su estructura, un aumento en el dataset de estos solucionaría este problema.

Los valores obtenidos por la matriz de confusión se dividen en las siguientes categorías:

- **Positivo verdadero (TP):** el modelo predijo positivo, y el valor real es positivo.
- **Negativo verdadero (TN):** el modelo predijo negativo, y el valor real es negativo.
- **Positivo falso (FP):** el modelo predijo positivo, pero el valor real es negativo.
- **Negativo falso (FN):** el modelo predijo negativo, pero el valor real es positivo.

Considerando una matriz de clasificación de 3 clases estos valores se pueden observar de la siguiente manera:

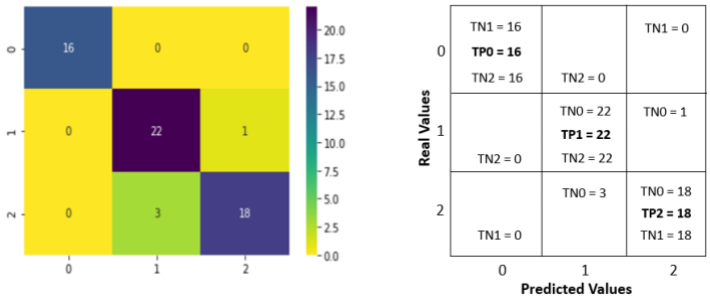


Figura 83: Matriz de confusión de 3 clases y su estructura [69]

Donde cada fila contiene una celda con TP y dos celdas con FN, mientras que cada columna contiene una celda con TP y dos celdas con FP.

Informe de clasificación (precisión por clase)

Un informe de clasificación se utiliza para medir la calidad de las predicciones a partir de un algoritmo de clasificación. El reporte, muestra las principales métricas de clasificación por clase. Las métricas se calculan utilizando verdaderos y falsos positivos, verdaderos y falsos negativos.

	precision	recall	f1-score	support
silla	1.00	0.81	0.90	27
mesa	0.67	1.00	0.80	10
lampara	1.00	1.00	1.00	29
accuracy			0.92	66
macro avg	0.89	0.94	0.90	66
weighted avg	0.95	0.92	0.93	66

Figura 84: Informe de clasificación

Las métricas presentes en el informe indican:

- **Exactitud (accuracy):** es la proporción de predicciones correctas.

$$Exactitud = (TP + TN)/(TP + TN + FP + FN)$$

- **Precisión**: es la proporción de positivos verdaderos de todos los positivos.

$$Precision = TP \ / \ (TP + FP)$$

- **Exhaustividad (recall):** es la proporción de valores positivos correctamente predichos.

$$Exhaustividad = TP \ / \ (TP + FN)$$

- **Valor-F**: es utilizado para comparar clasificadores, es la media armónica de precisión y exhaustividad.

$$ValorF = \frac{(2 \ * precision * exhaustividad)}{(Precision \ + \ exhaustividad)}$$

- **Macro avg**: aritmética media entre dos clases.

$$Macro\ avg = \frac{p0 + p1}{2}$$

- **Weighted avg**: media ponderara.

$$Weighted\ avg = \frac{p0 * s0 + p1 * s1}{s0 + s1}$$

- **Apoyo (support)**: número de observaciones por cada clase.

Predicción

Hacer una predicción es el paso final del ciclo de vida del modelo, es el motivo por el que fue creado el modelo. Necesita nuevos datos para los que se requiera una predicción. En el script se han predicho 10 nubes del conjunto de testeo que son mostrados junto a su etiqueta real y predicha. La figura a continuación muestra un ejemplo realizado con el dataset ModelNet10.

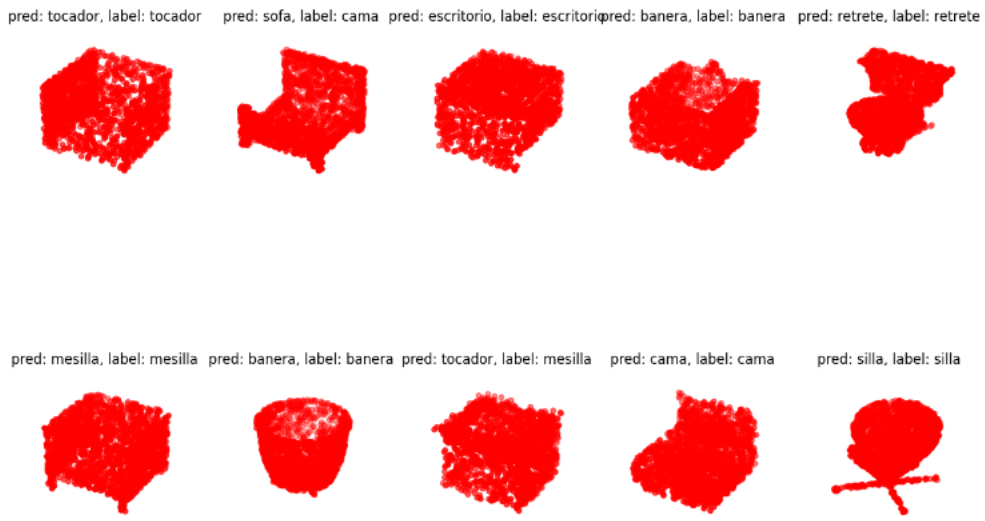


Figura 85: 10 predicciones de un sistema de 10 clases

Otra predicción de una sola nube, nueva para el modelo, se ha realizado para mostrar en una gráfica la correlación con cada clase, siendo resaltada la clase de mayor similitud como se visualiza en la Figura 86.

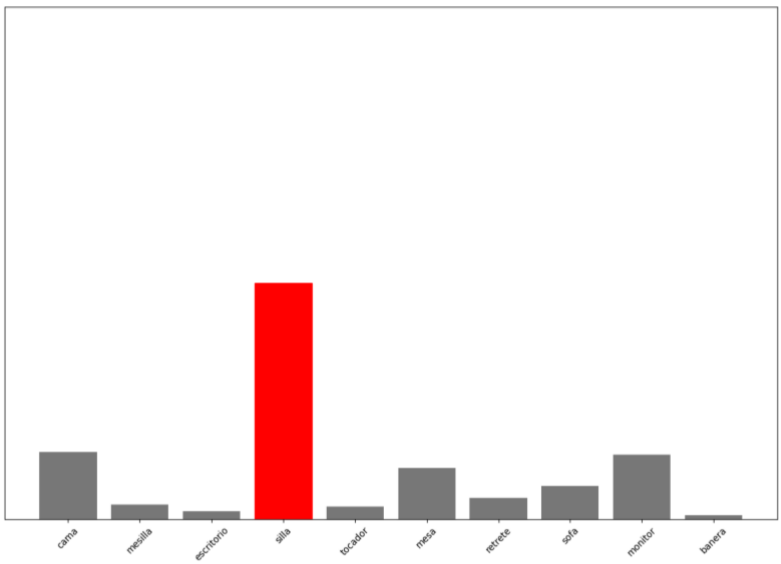


Figura 86: Gráfico con las predicciones de un sólo objeto

Finalmente, se crea un archivo .txt con los parámetros y resultados del entrenamiento para poder almacenarlos y favorecer la comparación en la mejora de parámetros del sistema.

5.1.2 Clasificación

El script final del sistema, segmentationClassification.py, permite cargar una escena, segmentar la nube en busca de posibles objetos y clasificar los posibles objetos detectados, para comparar su similitud con las clases aprendidas por el modelo. Para llevar a cabo este script, se han utilizado las librerías: Open3D, TensorFlow, Trimesh, Keras, matplotlib, numpy y argparse.

En primer lugar, el algoritmo carga una nube seleccionada en formato .txt y la presenta; ver Figura 87.

Figura 87: Nube de puntos cargada de archivo .txt

Además, calcula la componente normal de cada punto (visible con la tecla N), necesaria para cálculos posteriores.

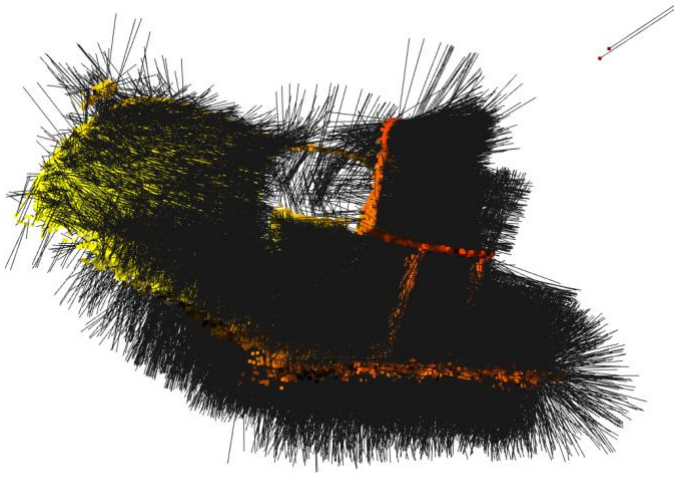


Figura 88: Componentes normales de los puntos de la nube

Tras realizar los procesos de filtrado del plano predominante y ruido vistos en el capítulo anterior, se obtiene el siguiente resultado:

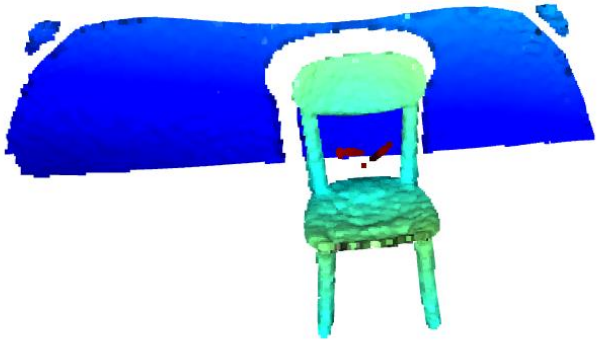


Figura 89: Nube filtrada

Tras este proceso, la nube es sometida al método DBSCAN, encargado de agrupar sus puntos, obteniendo como resultado una nube clusterizada:

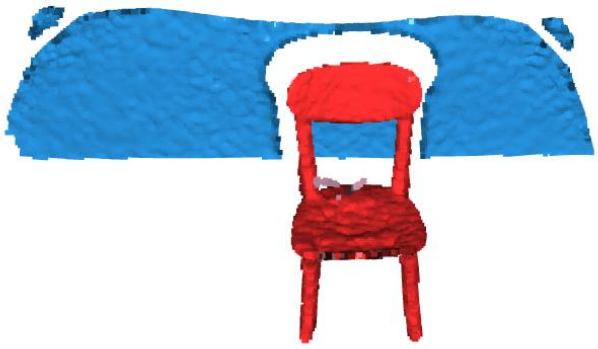


Figura 90: Nube clusterizada

Todos los clusters son limitados mediante cajas, los que sean menor que el umbral de tamaño deseado, son descartados. A los clusters de dimensiones interesantes, se le calcula su distancia con respecto al punto de captura, se muestra por pantalla el cluster, ver Figura 91, y se muestrea con N puntos para poder trasladarlo a la red de clasificación.

Figura 91: Cluster de 5647 puntos situado a 1.1 metros

Tras almacenar los puntos de los clústeres de interés, se muestran todos los clusters obtenidos tras el filtrado y destacados tanto en la nube filtrada (Figura 92) como en la original (Figura 93).

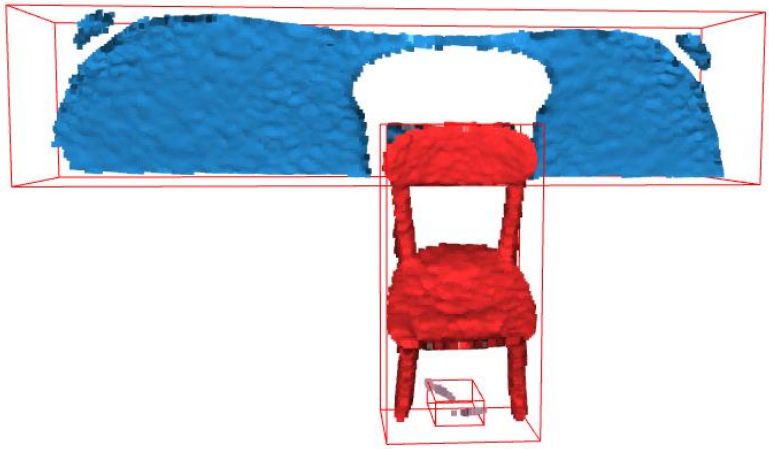


Figura 92: Clusters formados en la nube filtrada

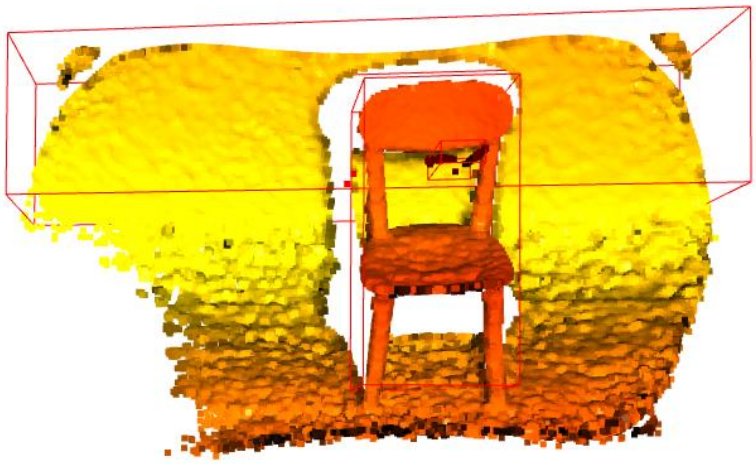


Figura 93: Clusters destacados en la nube original

Esta etapa es similar a la vista en el algoritmo miDataClusterSegmentation.py donde las librerías Open3D y Trimesh han permitido limpiar la nube hasta detectar los posibles objetos. A continuación, los posibles objetos pasan a la siguiente fase, la clasificación, realizada por las librerías TensorFlow y Keras. Para ello, se cargan los pesos guardados tras el entrenamiento (pointNet.py) en un modelo de PointNet y se compila el modelo.

Con los puntos almacenados, se predicen las categorías del cluster o conjunto de clusters obtenidos. En primer lugar, se verifica que la nube posee la cantidad de puntos necesarios (2048 por defecto) y se visualiza.

Figura 94: Nube muestreada con 2048 puntos

Por pantalla, se ofrece la predicción y el valor de confianza que presenta la nube con la clase predicha. Teniendo en cuenta estos valores de confianza, se realiza una gráfica para mostrar visualmente el resultado de la predicción.

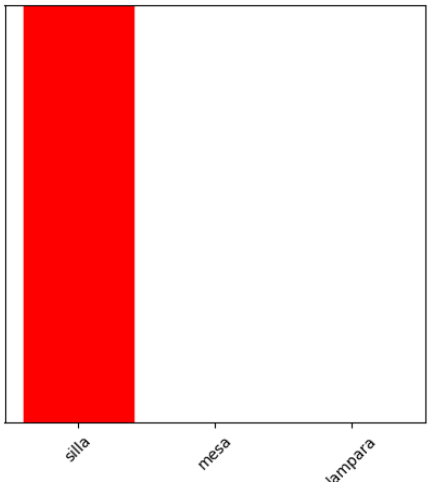


Figura 95: Predicción del cluster detectado de una silla

5.2 Resultados

Para la creación del modelo se realizaron múltiples pruebas con los distintos conjuntos de datos y ajustes. Tras la fase de entrenamiento, se han evaluado los modelos con diferentes herramientas que facilitan su comparación. A continuación, se redactan los tres modelos más concluyentes:

MiDataCluster3

MiDataCluster3, es un modelo realizado con el conjunto de datos MiDataCluster donde las nubes se encuentran en formato .ply. Se encuentran separadas en carpetas de entrenamiento y testeo con una distribución 80-20. Para este modelo se tuvieron en cuenta 3 clases de objetos: silla, mesa y lámpara.

Los parámetros utilizados para la configuración del modelo fueron:

Parámetro	Valor
Número de clases	3
Número de puntos	2048
Tamaño de batch	24
Número de epochs	20

Tabla 6: Parámetros del modelo MiDataCluster3

En la matriz de confusión obtenida podemos observar que la tarea de clasificación ha sido prácticamente excelente. El objeto que produce problemas es la mesa, probablemente provocado por la escasez de imágenes en el conjunto.

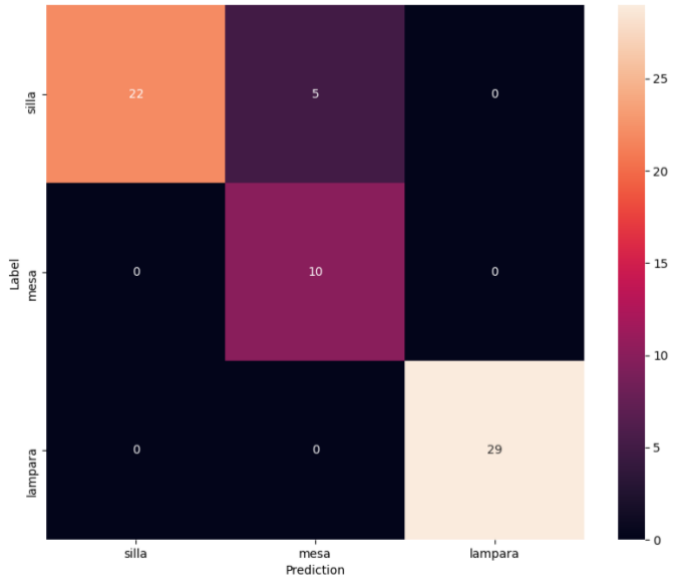


Figura 96: Matriz de confusión del modelo MiDataCluster3

El informe de clasificación proporciona los siguientes resultados:

Clase	Precisión	Exhaustividad	Valor-F	Exactitud
Silla	100%	81%	90%	
Mesa	67%	100%	80%	
Lámpara	100%	100%	95%	
TOTAL				92%

Tabla 7: Informe de la clasificación realizada por el modelo MiDataCluster3

Para mostrar de forma visual el tipo de datos que posee el modelo y su exactitud, se han predicho 10 elementos del conjunto de testeo mediante el modelo MiDataCluster3, mostrándose en la Figura 97, las etiquetas predichas junto a los puntos y etiquetas reales.

Figura 97: 10 Predicciones realizadas por el modelo MiDataCluster3

MiDataCluster4

MiDataCluster4, es un modelo realizado con 4 clases de objetos pertenecientes al conjunto de datos MiDataCluster: silla, monitor, mesa y lámpara.

Parámetro	Valor
Número de clases	4
Número de puntos	2048
Tamaño de batch	24
Número de epochs	20

Tabla 8:Parámetros del modelo MiDataCluster4

En la matriz de confusión creada, se puede observar una mayor cantidad de fallos dada la similitud de estructuras. Los monitores, al igual que la mesa, al tener partes compuestas por cristales tienden a peores resultados provocados por la calidad de sus capturas. Las mesas y especialmente las lámparas, por su forma peculiar, han sido más fáciles de distinguir por el sistema.

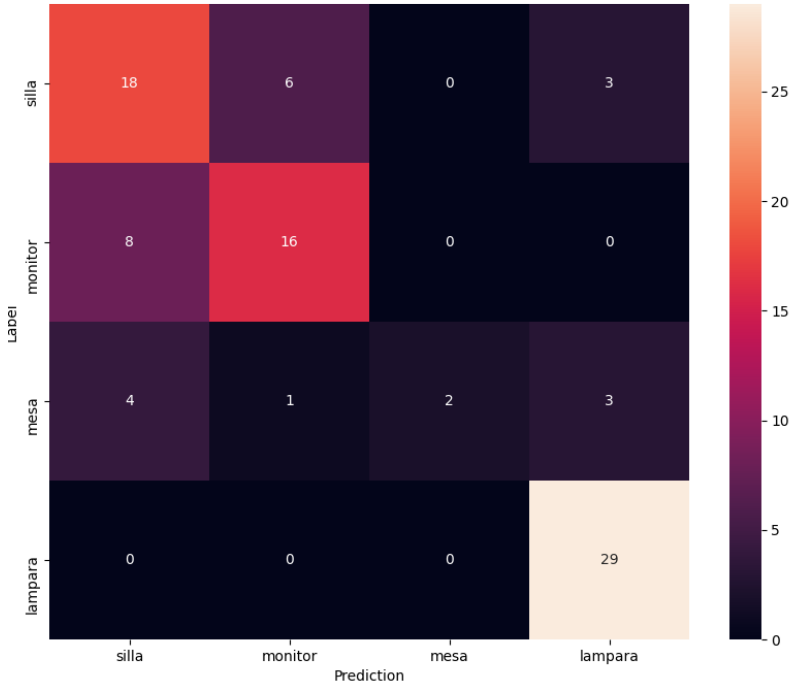


Figura 98: Matriz de confusión del modelo MiDataCluster4

El informe de clasificación proporciona los siguientes resultados en la tarea de clasificación:

Clase	Precisión	Exhaustividad	Valor-F	Exactitud
Silla	67%	73%	67%	
Monitor	70%	67%	68%	
Mesa	100%	60%	77%	
Lámpara	83%	100%	91%	
TOTAL				79%

Tabla 9: Informe de clasificación realizada por el modelo MiDataCluster4

Para mostrar de forma visual el tipo de datos que posee el modelo y su exactitud, se han predicho 10 elementos del conjunto de testeo mediante el modelo MiDataCluster4, mostrándose en la Figura 99, las etiquetas predichas junto a los puntos y etiquetas reales.

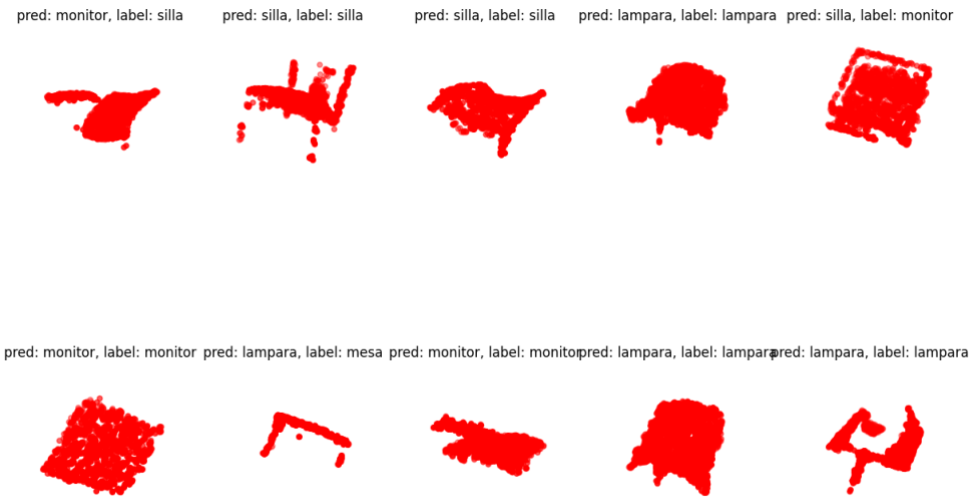


Figura 99: 10 Predicciones del modelo MiDataCluster4

ModelNet10

El modelo ModelNet10, ha sido realizado con el dataset ModelNet10. Este modelo, presenta sus objetos en forma de mallas (.off), tras recibir una perfecto modelado. Consta de 10 clases: cama, mesilla, escritorio, silla, tocador, mesa, retrete, sofá, monitor y bañera.

Los parámetros utilizados para la configuración del modelo fueron:

Parámetro	Valor
Número de clases	10
Número de puntos	2048
Tamaño de batch	24
Número de epochs	30

Tabla 10: Parámetros del modelo ModelNet10

En la matriz de confusión obtenida podemos observar que pese haber aumentado la cantidad de clases y datos en el sistema se ha obtenido una clasificación muy exitosa. En la matriz se reflejan equivocaciones entre elementos de similares características como son las mesas y escritorios o los tocadores y mesillas.

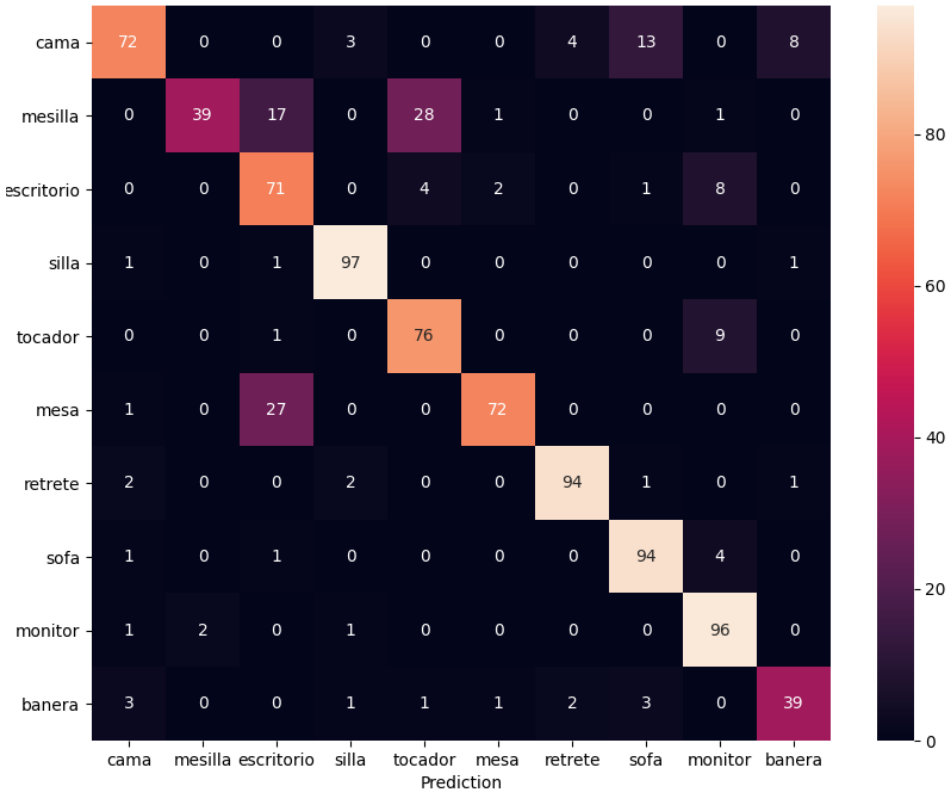


Figura 100: Matriz de confusión del modelo ModelNet10

El informe de clasificación proporciona los siguientes resultados en la tarea de clasificación:

Clase	Precisión	Exhaustividad	Valor-F	Exactitud
Cama	89%	72%	80%	
Mesilla	95%	45%	61%	
Escritorio	60%	83%	70%	
Silla	93%	97%	95%	
Tocador	70%	88%	78%	
Mesa	95%	72%	82%	
Retrete	94%	94%	94%	
Sofá	84%	94%	89%	
Monitor	81%	96%	88%	
bañera	80%	78%	79%	
TOTAL				83%

Tabla 11: Informe de la clasificación realizada por el modelo ModelNet10

Para mostrar de forma visual el tipo de datos que posee el modelo y su exactitud, se han predicho 10 elementos del conjunto de testeo mediante el modelo ModelNet10, mostrándose en la Figura 101, las etiquetas predichas junto a los puntos y etiquetas reales.

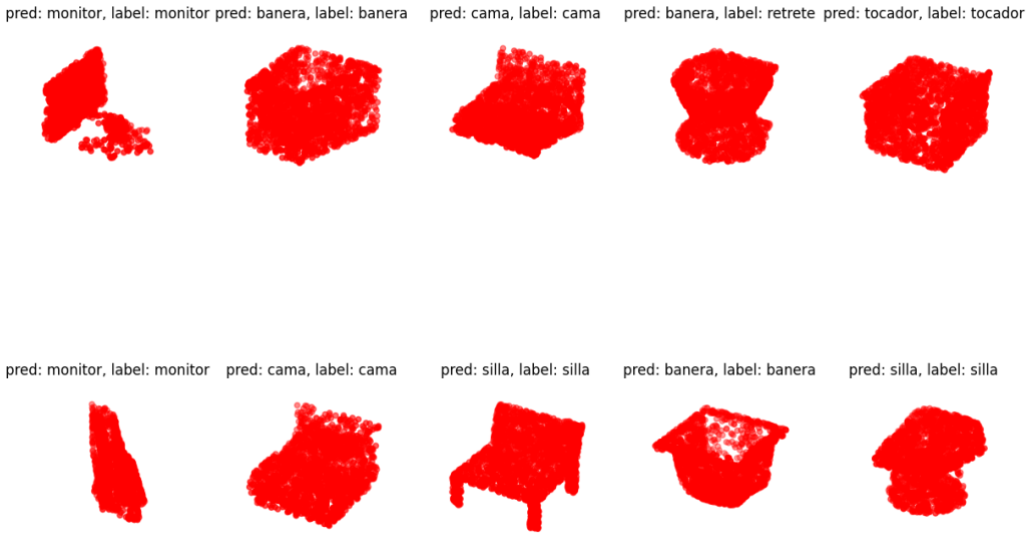


Figura 101: 10 Predicciones del modelo ModelNet10

Con los resultados obtenidos en la etapa de segmentación y filtrado, se ha conseguido detectar e identificar la posición de los posibles objetos satisfactoriamente. Tras aislar estos conjuntos de puntos, se ha conseguido clasificarlos de acuerdo con los modelos entrenados.

Se han entrenado 3 modelos, dos de ellos con nubes de puntos propias y otro con mallas de internet. Los modelos de adquisición propia mostraron un resultado satisfactorio de un 92% y 79% de exactitud respectivamente. Sin embargo, al aumentar la cantidad de clases, que no la cantidad de datos, el sistema comienza a distinguir entre ellas. Se deduce que esto es producido debido a la pérdida de información en las capturas de la cámara y el posterior procesado.

El modelo ModelNet10, ha mostrado una exactitud óptima del 83% al estar constituida por objetos perfectamente modelados. Teniendo en cuenta los casi cinco mil objetos que forman, las 10 clases que lo componen, este modelo ha permitido verificar que la arquitectura de nuestro enfoque funciona correctamente. Sin embargo, dada la distinta naturaleza de los datos, este modelo no es capaz de clasificar con buen rendimiento las imágenes capturadas por la CamBoard pico flexx. A pesar de ello, puede ser una buena solución si se consigue realizar una mejor reconstrucción de la escena.

Capítulo 6 – Conclusiones y líneas futuras

El objetivo principal del proyecto era la identificación y reconocimiento de obstáculos en entornos de interior, a través de nubes de puntos, mediante un sistema de bajo costo. Observando los porcentajes de éxito que se han obtenido en la tarea de reconocimiento de objetos, se puede concluir que se ha conseguido cumplir este objetivo de manera satisfactoria.

Para la consecución del objetivo principal, ha sido necesario superar otros objetivos secundarios como son: el estudio de los métodos actuales de reconocimiento de objetos a través de nubes de puntos, el aprendizaje del lenguaje de programación, Python, la configuración del entorno de programación en el computador de bajo coste, el estudio de librerías compatibles con el lenguaje de programación y la arquitectura del computador y comprender el funcionamiento de la cámara y su SDK.

En cuanto a los algoritmos era necesario seguir los pasos de adquisición, filtrado, segmentación y clasificación. En la parte de la adquisición hemos conseguido obtener los datos de la escena a través de la cámara y filtrar sólo los puntos con un valor de confianza óptimo, permitiendo reducir el número de puntos de la nube para no ralentizar los cálculos posteriores.

En la fase de segmentación, hemos conseguido filtrar el plano predominante de la imagen, siendo este normalmente el suelo o la pared de la escena y filtrado el ruido resultante para eliminar la mayor cantidad de puntos residuales posibles. Una vez limpiada la escena, los puntos restantes han sido agrupados por densidad. Los clusters obtenidos de cada imagen que han superado el umbral de puntos deseados son propuestos para ser clasificados. El sistema de entrenamiento ha sido entrenado para reconocer hasta 4 tipos de objetos tras haber aprendido un dataset de 670 nubes de puntos.

El tratamiento de las nubes, pese a no haber utilizado la librería por antonomasia de la visón por computador, OpenCV, se ha llevado a cabo con una novedosa herramienta, que se encuentra en actual desarrollo y ofrece un gran potencial como es la librería Open3D.

Con relación a las futuras líneas del proyecto, una vez creado el sistema de clasificación y estudiadas las librerías Open3D y TensorFlow, se podría realizar la tarea de clasificación mediante otros enfoques de clasificación, en los que se usan las nubes en otros formatos de entrada a la red, como pueden ser los vóxeles. Por otro lado, se podría estudiar la mejora del procesado de las nubes de puntos, para obtener una reconstrucción óptima, a través de otras librerías donde probablemente se encuentre una mayor cantidad de soluciones compatibles con el sistema actual, usando el lenguaje de programación C, un ejemplo, puede ser la librería PCL [70].

No obstante, considerando las posibilidades que ofrece Python, las futuras actualizaciones de Open3D, ofrecerán nuevas herramientas y soluciones dentro de este ámbito.

Bibliografía

[1] C. Cadena *et al.*, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016, doi: 10.1109/TRO.2016.2624754.

[2] D. Y. Romero Godoy, D. de la C. Sánchez Rodríguez, and I. G. Alonso González, "Evaluación de cámaras de tiempo de vuelo para la definición de entornos de interior," 2020.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] O. Community, "The OpenCV Reference Manual," *October*, 2010.

[5] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," Jan. 2018, [Online]. Available: http://arxiv.org/abs/1801.09847.

[6] C. Generales, "Que es la visión artificial," 1966.

[7] Jitendra Malik, "How Computer Vision Is Finally Taking Off, After 50 Years."

[8] R. Cipolla, C. Hernández, G. Vogiatzis, and B. Stenger, "Recent Advances and New Applications of Computer Vision," 東芝レビュー, vol. 62, 2007.

[9] J. M. de la Cruz Garcia and G. Pajares Martinsanz, *Visión por computador: imágenes digitales y aplicaciones*. 2001.

[10] N. L. Fernández García, "Introducción a la Visión Artificial," 2016.

[11] R. Szeliski, *Computer Vision: Algorithms and Applications*. 2010.

[12] A. R. Pope, "Model-Based Object Recognition - A Survey of Recent Research," 1994.

[13] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions," *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767, Sep. 2004, doi: 10.1016/j.imavis.2004.02.006.

[14] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Procedings of the Alvey Vision Conference 1988*, 1988, pp. 23.1-23.6, doi: 10.5244/C.2.23.

[15] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, pp. 1150–1157 vol.2, doi: 10.1109/ICCV.1999.790410.

[16] P. J. Olver, *Equivalence, Invariants and Symmetry*. Cambridge University Press, 1995.

[17] J. Wang, "Fast Algorithm for the Travelling Salesman Problem and the Proof of P = NP," *Applied Mathematics*, vol. 09, no. 12, pp. 1351–1359, 2018, doi: 10.4236/am.2018.912088.

[18] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences," 1974.

[19] Y. Pérez Gallardo, "Algoritmo Inteligente para el reconocimiento y acotación semántica de primitivas en una nube de puntos 3D."

[20] G. Mountrakis, J. Im, and C. Ogole, "Support vector machines in remote sensing: A review," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 3, pp. 247–259, May 2011, doi: 10.1016/j.isprsjprs.2010.11.001.

[21] Applesfera, "Camera TrueDepth." https://i.blogs.es/910877/app-escaneo-3d-iphone-12-pro-lidar-/1366_2000.jpg.

[22] G. Sansoni, M. Trebeschi, and F. Docchio, "State-of-the-art and applications of 3D imaging sensors in industry, cultural heritage, medicine, and criminal

investigation," *Sensors*, vol. 9, no. 1. pp. 568–601, Jan. 2009, doi: 10.3390/s90100568.
[23] F. Blais, M. Rioux, and J.-A. Beraldin, "Practical Considerations For A Design Of A High Precision 3-D Laser Scanner System," Nov. 1988, p. 225, doi: 10.1117/12.947787.
[24] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 77–85, doi: 10.1109/CVPR.2017.16.
[25] P. Bourke, "PLY - Polygon File Format." http://paulbourke.net/dataformats/ply/.
[26] pointclouds.org, "The PCD (Point Cloud Data) file format." http://pointclouds.org/documentation/tutorials/pcd_file_format.html.
[27] M. A. Fischler and R. C. Bolles, "Random sample consensus," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, doi: 10.1145/358669.358692.
[28] Data Analytics, "RANSAC Regression." https://vitalflux.com/ransac-regression-explained-with-python-examples/.
[29] F. Poux, "3D point cloud segmentation and clustering." https://towardsdatascience.com/how-to-automate-3d-point-cloud-segmentation-and-clustering-with-python-343c9039e4f5.
[30] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," 1996.
[31] Scikit-learn.org, "DBSCAN clustering algorithm." https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html.
[32] J. Wu, C. Geyer, and J. M. Rehg, "Real-time human detection using contour cues," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 860–867, doi: 10.1109/ICRA.2011.5980437.
[33] R. Min, N. Kose, and J.-L. Dugelay, "KinectFaceDB: A Kinect Database for Face Recognition," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 11, pp. 1534–1548, Nov. 2014, doi: 10.1109/TSMC.2014.2331215.
[34] E. Özbay and A. Çinar, "A voxelize structured refinement method for registration of point clouds from Kinect sensors," *Engineering Science and Technology, an International Journal*, vol. 22, no. 2, pp. 555–568, Apr. 2019, doi: 10.1016/j.jestch.2018.09.012.
[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning (Adaptive Computation and Machine Learning series)*. 2016.
[36] Sumologic, "Artificial Intelligence vs. Machine Learning vs. Deep Learning: What's the Difference?" https://www.sumologic.com/blog/machine-learning-deep-learning/.
[37] D. Calvo, "Perceptrón Multicapa – Red Neuronal," 2018. .
[38] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.
[39] TensorFlow Playground, "Linear and no linear regression data." https://playground.tensorflow.org/.
[40] ICHI, "ReLu Graphic." https://ichi.pro/es/una-guia-practica-de-relu-202570656444919.
[41] L. Shen, Q. Zhang, G. Cao, and H. Xu, "Fall Detection System Based on Deep Learning and Image Processing in Cloud Environment," 2019, pp. 590–598.
[42] G. S. Girón Molina, "Deep Learning y su increíble impacto en la realidad." https://revistaecys.github.io/13Edicion/03_ggiron.html.

[43] D. Calvo, “Aprendizaje supervisado.” https://www.diegocalvo.es/aprendizaje-supervisado/.
[44] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” 2015.
[45] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” Dec. 2015, [Online]. Available: http://arxiv.org/abs/1512.00567.
[46] M. Yani, M. T. Budhi Irawan, S, Si., and M. T. Casi Setiningsih, S.T., “Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail,” *Journal of Physics: Conference Series*, vol. 1201, p. 012052, May 2019, doi: 10.1088/1742-6596/1201/1/012052.
[47] S. Itzik Ben, “3D Point Cloud Classification using Deep Learning.” https://www.itzikbs.com/3d-point-cloud-classification-using-deep-learning.
[48] S. A. Bello, S. Yu, C. Wang, J. M. Adam, and J. Li, “Review: Deep learning on 3D point clouds,” *Remote Sensing*, vol. 12, no. 11. 2020, doi: 10.3390/rs12111729.
[49] Z. Liang, M. Yang, and C. Wang, “3D graph embedding learning with a structure-aware loss function for point cloud semantic instance segmentation,” *arXiv*, vol. 5, no. 3, 2019.
[50] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep hierarchical feature learning on point sets in a metric space,” in *Advances in Neural Information Processing Systems*, 2017, vol. 2017-December.
[51] Apple, “MacBook Pro (13 pulgadas, 2018, cuatro puertos Thunderbolt 3).” https://support.apple.com/kb/SP775?locale=es_ES.
[52] NVIDIA Developer, “Getting Started with Jetson Nano Developer Kit.” .
[53] NVIDIA, “Jetson Nano datasheet.” https://elinux.org/Jetson_Nano.
[54] Pmdtechnologies, “Development Kit Brief CamBoard pico flexx,” 2018. https://pmdtec.com/picofamily/wp-content/uploads/2018/03/PMD_DevKit_Brief_CB_pico_flexx_CE_V0218-1.pdf.
[55] T. Ringbeck and B. Hagebeuker, “A 3D time of flight camera for object detection,” *Optical 3D measurement techniques*, 2007.
[56] M. Quigley *et al.*, “ROS: an open-source Robot Operating System,” in *ICRA workshop on open source software*, 2009, vol. 3, no. 3.2.
[57] M. Abadi *et al.*, “TensorFlow: A system for large-scale machine learning,” 2016.
[58] A. Paszke *et al.*, “PyTorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, 2019, vol. 32.
[59] Z. Qian-Yi, P. Jaesik, and K. Vladlen, “Documentation Open3D: A Modern Library for 3D Data Processing,” 2018. http://www.open3d.org/docs/release/.
[60] Dawson-Haggerty, “Trimesh,” 2019. https://trimsh.org/.
[61] J. D. Hunter, “Matplotlib: A 2D graphics environment,” *Computing in Science and Engineering*, vol. 9, no. 3, 2007, doi: 10.1109/MCSE.2007.55.
[62] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, 2011.
[63] M. Waskom, “seaborn: statistical data visualization,” *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, Apr. 2021, doi: 10.21105/joss.03021.
[64] Z. Wu *et al.*, “3D ShapeNets: A deep representation for volumetric shapes,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June-2015, doi: 10.1109/CVPR.2015.7298801.
[65] Stanford University, “PointNet Code.” https://github.com/charlesq34/pointnet.
[66] D. Griffiths, “Point cloud classification with PointNet.” https://keras.io/examples/vision/pointnet/.

[67] D. Griffiths, “PointNet Keras Documentation code.” https://github.com/keras-team/keras-io/blob/master/examples/vision/pointnet.py.
[68] Machine Learning Mastery, “Get Started in Deep Learning With tf.keras.” https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/.
[69] medium, “Confusion matrix and classification report.” https://medium.com/swlh/confusion-matrix-and-classification-report-88105288d48f.
[70] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” 2011, doi: 10.1109/ICRA.2011.5980567.
[71] SD Association, “SD Memory Card Formatter.” https://www.sdcard.org/downloads/formatter/.
[72] NVIDIA Developer, “Jetson Nano Image.” https://developer.nvidia.com/jetson-nano-sd-card-image.
[73] Balena, “Balena Etcher.” https://www.balena.io/etcher/.
[74] Pmdtechnologies, “pmd SDK.” https://pmdtec.com/picofamily/software/.
[75] Rose Web Services LLC, “Python 3.6.4.” https://www.rosehosting.com/blog/how-to-install-python-3-6-4-on-debian-9/.
[76] JetBrains s.r.o., “PyCharm.” https://www.jetbrains.com/es-es/pycharm/.
[77] JetBrains s.r.o., “PyCharm Download.” https://www.jetbrains.com/es-es/pycharm/download/#section=linux.
[78] JetsonHacks, “Jetson Nano memory swap.” https://www.jetsonhacks.com/2019/11/28/jetson-nano-even-more-swap/.
[79] Open3D, “Open3D arm support.” http://www.open3d.org/docs/release/arm.html#nvidia-jetson.

II PRESUPUESTO

En este apartado se presentará el coste total en la realización del proyecto, así como el desglose de los gastos con la siguiente estructura:

- Trabajo tarifado por el tiempo empleado.
- Amortización del inmovilizado material.
  - Amortización del material hardware.
  - Amortización del material software.
- Redacción del documento.
- Derechos de visado del Colegio Oficial de Ingenieros Técnicos de Telecomunicación
- Gastos de tramitación y envío (COITT).

Finalmente, tras analizar todos estos puntos, se aplicarán los impuestos vigentes y se calculará el coste total del proyecto.

P.1 Trabajo tarifado por el tiempo empleado

El trabajo tarifado por tiempo empleado es el coste de la mano de obra asociada al trabajo de un graduado en ingeniería de telecomunicaciones. Por lo que se contabilizan los gastos referentes a la mano de obra, atendiendo al salario correspondiente a la hora de trabajo de un Ingeniero Técnico de Telecomunicación. Para el cálculo de los honorarios totales se hace uso de la siguiente fórmula:

$$H = C * (74{,}88 * Hn + 96{,}72 * He)$$

Donde:

- Hn, número de horas trabajadas dentro de la jornada laboral.
- He, número de horas trabajadas fuera de la jornada laboral.
- C, factor de corrección en función de las horas trabajadas.

Para el desarrollo de este TFG se ha empleado un total de 300 horas, con un horario correspondiente a la jornada laboral. En la siguiente tabla se muestran los factores de corrección, propuestos por el COITT, que se han de aplicar según las horas trabajadas.

Horas	Factor de corrección
Hasta 36 horas	1
De 36 a 72 horas	0,90
De 72 a 108 horas	0,80
De 108 a 144 horas	0,70
De 144 a 180 horas	0,65
De 180 a 360 horas	0,60
De 360 a 540 horas	0,55

Tabla 12: Factores de corrección del COITT

De acuerdo con lo establecido por el COITT, según se indica en la Tabla 12, el factor de corrección C que se debe aplicar es de 0,6. Por tanto, sustituyendo los datos en la fórmula anterior, el trabajo tarificado por el tiempo empleado en este proyecto tiene un coste de trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos.

$$H = 0{,}6 * (74{,}88 * 300 + 96{,}72 * 0) = 13.478{,}40\,€$$

P.2 Amortización del inmovilizado material

Se tendrán en consideración tanto los recursos hardware como softwares empleados para el desarrollo de este TFG. Para calcular el coste de amortización en un periodo de 3 años se utiliza un sistema de amortización lineal, en el que se supone que el inmovilizado material se desprecia de forma constante a lo largo de su vida útil. El cálculo de la cuota de amortización anual se obtiene de la siguiente formula:

$$Cuota\ Anual = (Valor\ de\ adquisición - Valor\ residual)\ /\ Años\ de\ vida\ útil$$

El valor residual es el valor teórico que tendrá el elemento una vez haya pasado su vida útil, por lo cual es considerado nulo.

P.2.1 Amortización del material hardware

El periodo de desarrollo de este TFG ha sido de 7 meses, periodo muy inferior a los 3 años que se estipulan para el coste de amortización. Por dicha razón se calculan los costes sobre la base de los derivados de los primeros 7 meses.

En esta tabla se enumeran los elementos hardware amortizables necesarios para el desarrollo de este TFG. Además de los elementos utilizados, también se indica el valor de adquisición y su amortización para el tiempo de 7 meses.

Elemento	Valor de adquisición (€)	Amortización (€)
Ordenador portátil	1.999,00	388,7
NVIDIA Jetson Nano	109,00	21,20
CamBoard pico flexx	299,00	58,14
Pantalla modelo LR10FHD01	40,00	7,78
Carcasa Jetson	10,00	1,95
Tarjeta MicroSD 32 Gb	7,20	1,4
Batería externa	20,99	4,1
Total		483,27 €

Tabla 13: Amortización de los recursos hardware.

Los gastos totales para el material hardware son de cuatrocientos ochenta y tres euros con veintisiete céntimos.

P.2.2 Amortización del material software

Como se mencionó en el apartado anterior, el periodo de desarrollo de este TFG ha sido de 7 meses, sólo se tendrán en cuenta los costes derivados de dichos meses. En la siguiente tabla se enumeran las aplicaciones necesarias para el desarrollo de este TFG, donde además se indican el valor de adquisición, así como el valor de amortización.

Elemento	Valor de adquisición (€)	Amortización (€)
Sistema operativo Jetson Nano	Gratuito	0,00
Office 365 (licencia ULPGC)	Gratuito	0,00
PyCharm (licencia estudiante)	Gratuito	0,00
Sistema operativo macOS Big Sur	Gratuito	0,00
Total		0,00 €

Tabla 14: Amortización de recursos software

Debido a que la mayoría del software utilizado ha sido proporcionado por la ULPGC mediante licencias, se ha utilizado software de código libre o software proporcionado junto al hardware, los gastos totales por software son nulos.

P.3 Redacción del documento

Para determinar el coste de la redacción del documento se usa la siguiente fórmula:

$$R = 0{,}07 * P * Cn$$

Donde:

- P, es el presupuesto
- Cn, es el coeficiente de ponderación en función del presupuesto

El valor del presupuesto equivale a la suma de los costes de trabajo tarifados por tiempo empleado y de la amortización del inmovilizado material. En la siguiente tabla se muestra el cálculo del presupuesto.

Concepto	Coste (€)
Tarificación por tiempo empleado	13.478,40
Amortización del inmovilizado material (hardware y software)	483,27
Total	13.961,67 €

Tabla 15: Cálculo del presupuesto para la redacción del documento

Según el COITT, el factor de ponderación, para presupuestos cuyo valor es inferior a 30.050,00€ es de 1,00. Por lo que los costes derivados de la redacción de documento se obtienen sustituyendo los valores en la ecuación anterior.

$$R = 0{,}07 * 13.961{,}67 * 1{,}00 = 977{,}32€$$

El coste de la redacción del documento es de novecientos setenta y siete euros con treinta y dos céntimos.

P.4 Derechos de visado del COITT

Para proyectos de carácter general, los derechos de visado se calculan de acuerdo con la siguiente fórmula.

$$V = 0{,}006 * F1 * C1 + 0{,}003 * F2 * C2$$

Donde:

- F1, es el valor del presupuesto para este proyecto.
- C1, es el coeficiente reductor en función del presupuesto.
- F2, es el presupuesto de ejecución material que corresponde a la obra civil.
- C2, es el coeficiente reductor en función del presupuesto correspondiente a obra

Como se señala en el apartado anterior, el coeficiente C1 está fijado a 1,00, ya que el presupuesto es inferior a 30.050,00. En el desarrollo de este TFG no se ha requerido obra civil por lo cual el valor de F2 es 0,00€. En la siguiente tabla se muestra el valor del presupuesto hasta el momento.

Concepto	Coste (€)
Tarificación por tiempo empleado	13.478,40
Amortización del inmovilizado material (hardware y software)	483,27
Redacción del documento	977,32
Total	14.938,99 €

Tabla 16: Cálculo del presupuesto para los derechos del visado del COITT

Sustituyendo nuestros valores en la fórmula anterior tenemos que:

$$V = 0{,}006 * 14.938{,}99 * 1{,}00 + 0{,}003 * 0{,}00 * 0{,}00 = 89{,}63€$$

Los costes por derecho de visado del presente TFG son de ochenta y nueve euros con sesenta y tres céntimos.

P.5 Gastos de tramitación y envío

Cada documento visado por vía telemática tiene un coste de 6,00€.

P.6 Aplicación de impuestos y coste total

El desarrollo del presente TFG está sujeto por el Impuesto General Indirecto Canario (I.G.I.C.) con un valor del 7 %. Teniendo en cuenta la aplicación de dicho impuesto, en la siguiente tabla se realiza el cálculo total del proyecto.

Concepto	Coste (€)
Tarificación por tiempo empleado	13.478,40
Amortización del inmovilizado material (hardware y software)	483,27
Redacción del documento	977,32
Derechos de visado del COITT	89,63
Gastos de tramitación y envío	6,00
Subtotal	15.034,22
I.G.I.C (7%)	1052,40
Total	16.086,62 €

Tabla 17: Aplicación de impuestos y coste total del presupuesto

El coste total para el desarrollo del TFG, “Detección e identificación de obstáculos a partir de una nube de puntos” tiene un valor de dieciséis mil ochenta y seis euros con sesenta y dos céntimos.

III ANEXOS

Anexo 1. Guía de instalación

1.1 Instalación de la NVIDIA Jetson Nano

1.2.1 Instalación del hardware

Para la elaboración del proyecto, se proporcionó el ordenador NVIDIA Jetson Nano junto a una carcasa plástica y unos ventiladores para dotar de una mejor refrigeración a la SBC. Adicionalmente se conectaron los siguientes periféricos a los puertos USB para mejorar su rendimiento y facilitar el trabajo: un adaptador wifi para tener acceso a Internet inalámbricamente, sin embargo, la conexión ethernet resultó más efectiva; ratón y teclado para poder navegar a través del computador; y la cámara pico flexx para capturar las escenas.

Figura 102: NVIDIA Jetson Nano

1.2.2 Instalación del software

Atendiendo a las advertencias del fabricante [52], para proporcionar al ordenador NVIDIA Jetson Nano de sistema operativo es necesario una tarjeta microSD. Esta tarjeta debe de ser formateada de antemano, para ello es necesario un ordenador con conexión a internet y el software SD Card Formatter [71].

Una vez conectada la tarjeta al ordenador y ejecutado el programa en cuestión, dentro de la interfaz del programa se debe seleccionar la tarjeta a formatear, seleccionar *Quick format* y clicar en *Format*.

Tras realizar el formateo de la tarjeta es necesario montar la imagen Jetson Nano Developer Kit en la tarjeta, para ello se debe descargar la imagen en la pagina web oficial [72] y el programa Balena Etcher [73] que nos permitirá realizar esta acción.

Para montar el sistema operativo, dentro del programa, se debe seleccionar la imagen "*Select image* "y la tarjeta "*Select drive"*. Una vez finalizado el proceso, la tarjeta ya está lista para ser insertada en la Jetson Nano para comenzar su configuración.

Tras insertar la tarjeta en la ranura correspondiente, conectar los dispositivos periféricos y alimentar al dispositivo SBC, se procederá a comprobar y actualizar las herramientas descargadas. Para ello hay que abrir el terminal y escribir los siguientes comandos:

- sudo date -s '<número del día> <mes (Jan, Feb…)> <año> <hora (12:00)>'
- sudo apt update
- sudo apt upgrade

1.2 Instalación API CamBoard pico flexx

Proseguimos a descargar el SDK del fabricante de la cámara [74], disponible sólo para clientes. Al descargar el archivo debemos, descomprimir la capeta "libroyale-3.23.0.86-LINUX-arm-64Bit" para poder ejecutarlo en la Jetson Nano.

Dentro de este directorio, cabe destacar los archivos PDF con la guía de usuario de la cámara, el fichero HTML con la información correspondiente al código de la API y la carpeta Python. La información proporcionada indica que sólo trabaja con Python 3.6 por lo que procedemos a descargarlo siguiendo [75].

El fabricante de la cámara también indica la necesidad de instalar las siguientes librerías:

- sudo apt install libpython3.5
- sudo apt install libpython3.6
- sudo apt install libpython3.7
- sudo pip3.6 install –upgrade pip
- sudo pip3.6 install numpy
- sudo pip3.6 install matplotlib

Para poder detectar la cámara, es necesario instalar los drivers proporcionados y cambiar los permisos del usuario. Dentro del archivo descomprimido acceder al directorio "libroyale-<version_number>-LINUX-arm-64Bit/driver/udev" y ejecutar el siguiente comando:

- sudo cp 10-royale-ubuntu.rules /etc/udev/rules.d

Tras realizar esta acción, es necesario desconectar y volver a conectar el USB de la placa. Una vez realizado estos pasos, la cámara ya está lista para su uso.

Figura 103: CamBoard pico flexx

1.3 Instalación del IDE PyCharm

Para comprobar que la cámara esté conectada se procederá a la instalación del IDE, PyCharm [76]. Tras descomprimir en el directorio de Descargas el fichero descargado de [77], se debe introducir en el terminal el comando:

- sudo mv pycharm-community-2020.X.X /opt/pycharm-community-2020.X.X

Este comando nos permite trasladar el archivo al directorio *opt* que haciendo una analogía con Windows sería como el directorio "Archivos y Programas". Una vez trasladado allí, accedemos a la carpeta *bin*:

- cd /opt/pycharm-community-2020.X.X/bin

Y ejecutar el archivo *pycharm.sh*:

- ./pycharm.sh

Una vez ejecutado, el IDE se inicializa permitiendo ajustar las configuraciones necesarias y creando un acceso directo en el escritorio.

1.4 Instalación Open3D

En primer lugar, es necesario incrementar el espacio de intercambio [78] para evitar que el computador se quede sin memoria RAM durante el proceso. Tras realizar este paso debemos continuar con los procesos preparatorios instalando *cmake*:

- sudo snap install cmake –classic

Comenzamos el tutorial [79] indicado por los desarrolladores de Open3D:

Instalamos dependencias necesarias:

- sudo apt-get update -y
- sudo apt-get install -y apt-utils build-essential git cmake
- sudo apt-get install -y python3 python3-dev python3-pip
- sudo apt-get install -y xorg-dev libglu1-mesa-dev
- sudo apt-get install -y libblas-dev liblapack-dev liblapacke-dev
- sudo apt-get install -y libsdl2-dev libc++-7-dev libc++abi-7-dev libxi-dev
- sudo apt-get install -y clang-7

Clonamos el repositorio de GitHub y accedemos al directorio:

- git clone --recursive https://github.com/intel-isl/Open3D
- cd Open3D
- git submodule update --init --recursive
- mkdir build

- cd build

Ejecutamos *cmake* con la configuración que deseamos para construir la librería C++:

- cmake \
  - -DCMAKE_BUILD_TYPE=Release \
  - -DCMAKE_CUDA_COMPILER=/usr/local/cuda-10.2/bin/nvcc \
  - -DBUILD_SHARED_LIBS=ON \
  - -DUSE_BLAS=ON \
  - -DBUILD_FILAMENT_FROM_SOURCE=ON \
  - -DCMAKE_CXX_FLAGS_RELEASE=ON \
  - -DCMAKE_CXX_FLAGS_RELEASE="-O3 -DNDEBUG -faligned-new" \
  - -DBUILD_GUI=ON \
  - -DGLIBCXX_USE_CXX11_ABI=ON \
  - -DBUILD_CUDA_MODULE=ON \
  - -DOPEN3D_ML_ROOT=https://github.com/intel-isl/Open3D-ML.git \
  - -DBUNDLE_OPEN3D_ML=ON \
  - -DBUILD_TENSORFLOW_OPS=OFF \
  - -DBUILD_PYTORCH_OPS=ON \
  - -DBUILD_UNIT_TESTS=ON \
  - -DCMAKE_INSTALL_PREFIX=~/open3d_install \
  - -DPYTHON_EXECUTABLE=$(which python3)

- make -j3

Ejecutamos los test:

- make tests -j3 ./bin/tests --gtest_filter="-*Reduce*Sum*"

Si se desea instalar el paquete C++:

- make install

Instalación del paquete de Python:

- make install-pip-package -j3
- python -c "import open3d; print(open3d)"

Ejecutamos para comprobar que funciona la interfaz gráfica:

- ./bin/Open3D/Open3D

1.4.1 Guía de uso de la interfaz gráfica de Open3D

La interfaz gráfica que proporciona Open3D aporta múltiples herramientas para visualizar y tratar las nubes de puntos mediante Python. A continuación, se adjuntan algunos comandos básicos para comprender su funcionamiento y obtener un mejor resultado.

- Control General
  - Q, Esc: Salir de la ventana
  - H: Imprimir mensaje de ayuda
  - P, ImprPant : Hacer captura de pantalla
  - D: Hacer captura de profundidad
  - O: Hacer captura de los ajustes actuales de representación
  - N: Enseñar las componentes normales si existen

- Control de la vista con el ratón
  - Botón izquierdo + mover: Rotar
  - Ctrl + botón izquierdo + mover: Trasladar
  - Botón de la rueda + mover: Trasladar
  - Shift + botón izquierdo + mover : Rodar
  - Rueda: Zoom

- Control de la vista con el teclado
  - [/]: Incrementar/disminuir punto de vista
  - R: Resetear punto de vista
  - Ctrl/Cmd + C: Copiar el estado actual de la vista en el portapapeles
  - Ctrl/Cmd + V: Pegar el estado de la vista desde el portapapeles

Anexo 2. Código desarrollado

Los scripts y bases de datos creadas para este proyecto se encuentran accesibles en https://github.com/feersantana5/TFG.

Anexo 3. Estructura de los datasets

Para servir de apoyo en la comprensión del proyecto, en especial de los datasets y los argumentos de los scripts, se han adjuntado las estructuras utilizadas para su creación.

MiData
clase_objeto
objeto
captura(n_captura)
ImagenesRGBD(2D)
ObjetoC(n_Captura)F(n_frame).png
MatricesX
ObjetoX(n_Captura)(n_frame).txt
MatricesY
ObjetoY(n_Captura)(n_frame).txt
MatricesZ
ObjetoZ(n_Captura)(n_frame).txt
MatricesXYZ
ObjetoXYZ(n_Captura)(n_frame).txt

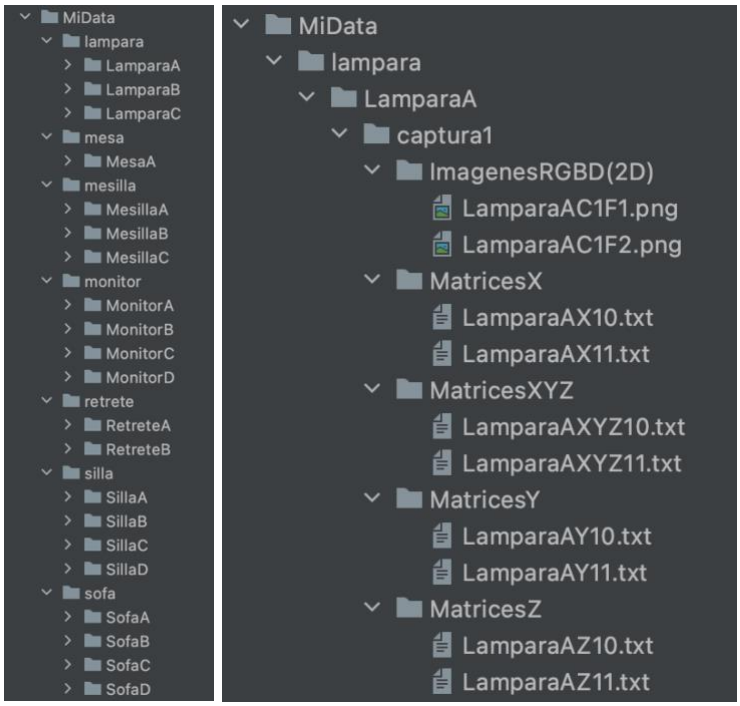


Figura 104: Estructura del dataset MiData

En este dataset, las 15 capturas de cada objeto se han realizado a 1, 1.5 y 2 metros de distancia, adquiriendo 5 imágenes en cada posición.

Figura 105: Escena de captura de MiData

En la imagen, se pueden apreciar las figuras indicando las diferentes distancias. A su vez, en cada distancia, las imágenes han sido sacadas de perfiles diferentes:

Plano inferior centrado Plano altura rodillas centrado Plano altura caderas centrado

Plano altura cadera izquierda Plano altura cadera derecha

MiDataCluster

clustersConf

mesh

test

objeto_C(n_captura)_F(n_frame)_Cluster(i).off

train

objeto_C(n_captura)_F(n_frame)_Cluster(i).off

ply

clase_objeto

objeto_C(n_captura)_F(n_frame)_Cluster(i).ply

txt

clase_objeto

objeto_C(n_captura)_F(n_frame)_Cluster(i).txt

xyzConf

clase_objeto

objeto_C(n_captura)_F(n_frame).txt

xyz

clase_objeto

objeto_C(n_captura)_F(n_frame).txt

ImagenesProfundidad(2D)

clase_objeto

objeto_C(n_captura)_F(n_frame).png

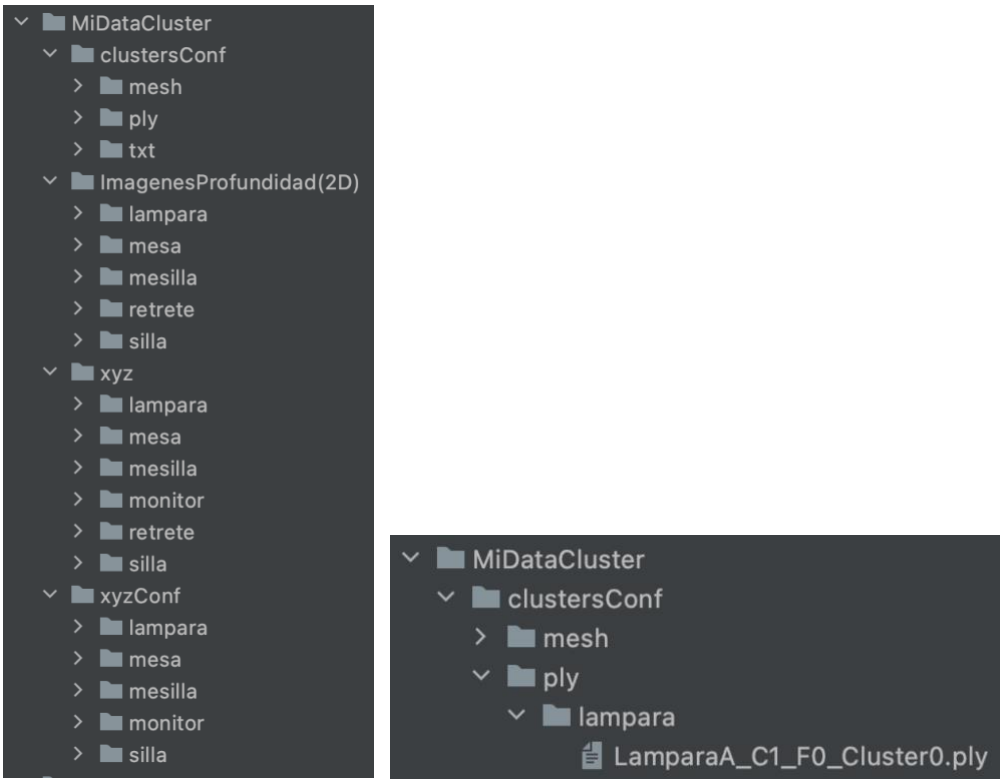


Figura 106: Estructura del dataset MiDataCluster

ModelNet10

clase_objeto

test

objeto_id.off

train

objeto_id.off

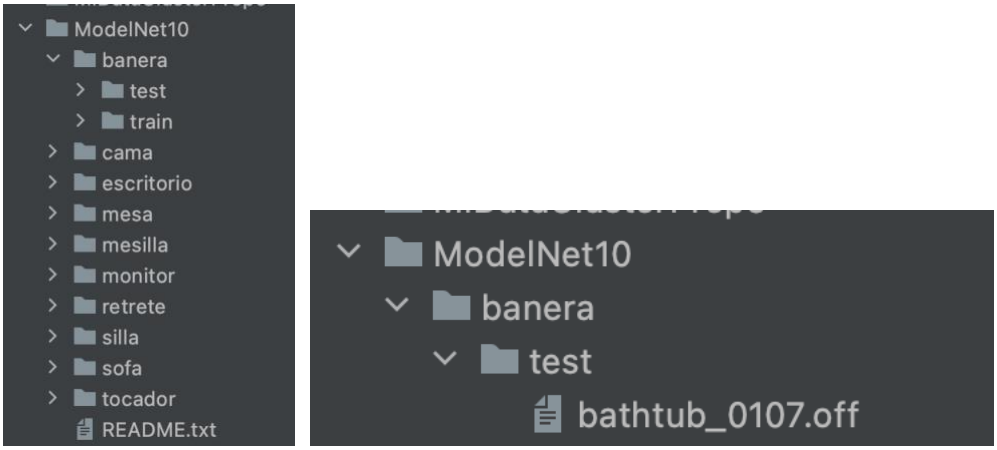


Figura 107: Estructura del dataset ModelNet10