Trabajo Final de Máster

# Estrategia de movimiento basada en Deep Learning y Visión por Ordenador para Robot Cuadrúpedo

**Màster Universitari en Sistemes Automàtics i Electrònica Industrial**

Gabriel Mourad

31 Enero 2022


Universitat Politècnica de Catalunya
Escola Politècnica Superior d'Enginyeria de Vilanova i la Geltrú

**Director**
Prof. Dr. José Antonio Soria Pérez

# Contents

# List of Figures

# Nomenclatura

CNN Convolutional Neural Network

CNN Convolutional Neural Network

FSM Finite State Machine

GB Giga Bytes

IA Inteligencia Artificial

ILSVRC ImageNet Large Scale Visual Recognition Challenge

ML Machine Learning

RAM Random Acces Memory

# Resumen

La locomoción es uno de los principales desafíos presentes en el campo de la robótica. Esta habilidad expande el alcance de los robots, y les permite resolver un gran abanico de tareas, desde aplicaciones cotidianas del día a día, hasta exploraciones en terrenos desconocidos. En este documento se detalla el desarrollo de una estrategia de movimiento, basada en inteligencia artificial, para un robot cuadrúpedo. El algoritmo desarrollado tiene como objetivo hacer que el robot alcance (de forma autónoma) un determinado objeto. El sistema propuesto está compuesto por varias entidades: un sensor cámara instalado en la parte frontal del robot para capturar imágenes del entorno, una red neuronal convolucional para clasificar la posición relativa del objetivo, y una estructura de robot cuadrúpedo encargada de realizar las acciones de movimiento determinadas por el algoritmo de control.

El desarrollo de este proyecto se ha llevado a cabo siguiendo una metodología ascendente. Resolviendo en primer lugar una simplificación del problema planteado, se ha aumentado la complejidad de cada estrategia de movimiento hasta alcanzar resultados satisfactorios. Cada etapa del desarrollo sigue la misma estructura: diseño del algoritmo de control, implementación y análisis de resultados. Gracias a ello, ha sido posible observar cuáles son las virtudes y carencias de cada estrategia de movimiento, permitiendo mejorar el desempeño del sistema en etapas posteriores. El resultado final es una estrategia de movimiento funcional, capaz de conducir el robot hasta el objetivo deseado gracias a los excelentes resultados obtenidos en el modelo de clasificación.

# Abstract

Locomotion is one of the main challenges to overcome in robotic technologies, this skill enable robotics to be applied over a wide range of tasks spanning day-to-day applications up to exploration tasks in unknown and far territories. This document details the development of a movment strategy definition on a four-legged robot based on artificial intelligence. The algorithm developed is made to enable the robot reach the objective autonomously, this system is composed of various subsystems: A camera installed in the front, to capture images of the surrounding. A convolutional neural network allows classification of the object's relative position and finally the robot's structural awareness that will execute the movements to be performed as per tasks generated by the control unit.

This project has been executed following an ascending methodology. Solving by first a simplification of the proposed problem, the complexity of each movement strategy has been increased until achieve satisfactory results. Each development stage follows the next structure: control algorithm design, implementation and analysis of the performed task. As a result, it has been possible to evaluate the strengths and weaknesses of each movement strategy, and then improve the system performance on subsequent stages. Thanks to the excellent accuracy acquired on the classification models, the final result is a functional movement strategy able to drive the robot to the desired object.

# 1. Introducción

El ML se ha convertido en una disciplina ingenieril omnipresente que se extiende más allá de las plataformas informáticas tradicionales (por ejemplo, servidores y ordenadores de sobremesa) hacia dispositivos y tecnologías de última generación, así como teléfonos móviles, sistemas embebidos, IoT, robótica y otros sistemas ciber-físicos [1].

Una aplicación emergente que pone a prueba los desafíos del ML en dispositivos de última generación es el aprendizaje de robots cuadrúpedos. Estas estructuras pueden llegar a ser ligeras y operar en espacios reducidos, convirtiéndolos en una herramienta útil para investigar soluciones a problemas de más calibre como la búsqueda y rescate de emergencia, el monitoreo y mantenimiento de infraestructuras o la supervisión y realización de tareas en el sector agrícola [1, 2].

En el desarrollo de la agricultura moderna, por ejemplo, la aplicación de robots y máquinas inteligentes para actualizar diferentes aspectos de este sector supone uno de los mayores desafíos [3]. La agricultura tradicional conlleva un gran número de tareas repetitivas, intensas y tediosas [4], como cortar, recoger o identificar el punto óptimo del fruto. Por otro lado, también es posible llegar a una agricultura sostenible, capaz de proveer suficiente producto sin afectar al medio ambiente, siendo posible aumentar la productividad a un menor coste [5, 6]. Los robots cuadrúpedos no requieren de superficies planas para su locomoción, debido a su estructura, pueden desplazarse en terrenos irregulares. Además, estas estructuras reducen los daños infligidos al terreno al andar [7], convirtiéndose en candidatos ideales para trabajar en entornos agrícolas.

Las ventajas de éstos robots son la alta capacidad de locomoción junto a la poca superficie de contacto, lo que facilita el desplazamiento y los convierte en herramientas completamente aptas para tareas de inspección y exploración [8]. Los robots cuadrúpedos están diseñados para cumplir tres funcionalidades básicas: alta velocidad locomotriz [9], estabilidad en cualquier terreno [10] y capacidad de carga [11]. Este tipo de robots devienen una tecnología versátil, puesto que son capaces de llevar a cabo un gran número de tareas utilizando técnicas de control simples.

Para llevar a cabo las funcionalidades básicas, un robot necesita información de calidad del entorno exterior. Además, el monitoreo y búsqueda de objetos en entornos caóticos y ruidosos necesita de técnicas de procesado de imagen y visión por computador para ser realizados correctamente [3]. Incorporar sistemas de visión da lugar al diseño y desarrollo de estrategias de control basadas en inteligencia artificial para resolver las distintas tareas planteadas.

## 1.1. Objetivo del proyecto

Se dispone de la estructura de robot cuadrúpedo comercial, *Bittle* [12] y, se pretende diseñar una estrategia de control para que el robot cuadrúpedo se desplace hasta encontrar un determinado objeto. Para ello, se lleva a cabo un desarrollo progresivo partiendo de estrategias simples a estrategias más complejas, dichas estrategias están basadas en el uso de algoritmos de Deep Learning que incluyen visión artificial y procesado de imagen.

## 1.2. Definición del proyecto

Las estrategias propuestas tienen como objetivo conseguir que el robot cuadrúpedo se oriente y encuentre objetos. En este caso, una pelota de tenis. Acorde a la figura 1.1 la estrategia de control se construye sobre los siguientes elementos:

- La cámara actúa como sensor de visión y envía las imágenes, que el modelo de Deep Learning utiliza para localizar y determinar la posición del objetivo, a la unidad de control del robot.
- El algoritmo procesa y clasifica la imagen mediante un modelo de Inteligencia Artificial (IA) con el que determina dónde está el objeto y toma las decisiones correspondientes para dirigirse al mismo.
- Aspectos relacionados con la cinemática del robot, aspectos mecánicos y diseño electrónico quedan fuera del alcance de este proyecto.
- En lo que respecta al proyecto, los elementos del robot se tratan como una entidad que, a partir de unas entradas, procesa información y genera las salidas correspondientes para su movimiento.

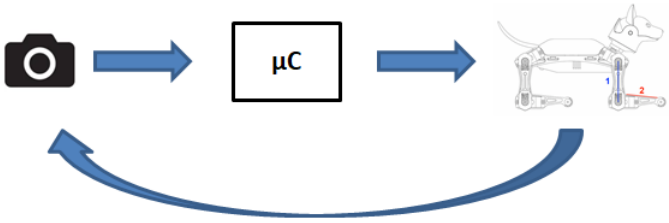


**Figure 1.1.:**

## 1.3. Especificaciones y requerimientos

- **Especificaciones del sensor:** El sensor para la detección de objetos consta de una cámara sensor Sony IMX219 de 8 Megapíxeles, que se sitúa en la parte frontal del robot. Este captura las imágenes del entorno y las almacena en la memoria del microcontrolador. El sensor propuesto ofrece imágenes y vídeo de alta calidad. Dispone también de funciones de control automático como el control de exposición, el balance de blancos y la detección de luminancia [13].
- **Microcontrolador:** Es la unidad a la que se conectan tanto los motores que mueven las articulaciones del robot y la cámara. La unidad de control seleccionada es una Raspberry Pi 4 Modelo B de 2 GB de RAM, dispone de dos puertos micro-HDMI, dos puertos USB, puerto MIPI CSI para la cámara y 40 pines GPIO [14]. La unidad de control debe permitir una comunicación sencilla con los periféricos (cámara y robot) y computar el modelo de clasificación y el algoritmo de control.
- **Software:** El sistema operativo Raspberry Pi OS Buster de 32 bits es el único sistema sistema operativo compatible con las siguientes librerías de Python que son, a su vez, requerimientos indispensables para construir y computar el modelo de Deep Learning utilizado:
  - TensorFlow 1.15.2
  - Keras 2.2.4
  - h5py 2.10
  - python-statemachine 0.9.0
- **Robot cuadrúpedo:** Para llevar a cabo las acciones de control determinadas por el microcontrolador se ha seleccionado el robot cuadrúpedo Bittle creado por Petoi y ThinkerGen. Una estructura de 290g y de pequeñas dimensiones. Dispone de una placa de control NyBorad V1 con un microcontrolador ATMega328PA y pines de conexión a la Raspberry Pi mediante puerto serie UART-RS232 [15]. El sistema permite ser configurado mediante librerías de código abierto compatibles con Raspberry Pi y el propio microcontrolador del robot:
  - **OpenCat**: La plataforma base utilizada es OpenCat, una librería de código abierto que permite configurar los movimientos del robot en distintos lenguajes de programación (C/C++/Python). Dicha plataforma cuenta con un protocolo de comunicación que permite controlar el robot a través del puerto serie, siendo posible obtener distintos tipos de movimientos así como gateos, posturas, giros y piruetas.
- **Fuente de alimentación:** Una batería de Li-ion recargable de 7.4V de salida y 1000mAh se ha utilizado para alimentar todo el sistema. De modo que robot, microcontrolador y cámara son alimentados por la misma fuente de energía.

- **Máquina de estados:** Las distintas estrategias de control desarrolladas en este proyecto están construidas sobre una máquina de estados. Para implementar las distintas máquinas de estados en el microcontrolador se utiliza la librería de código abierto StateMachine [16], esta herramienta permite crear máquinas de estado en lenguaje de programación Python.

## 1.4. Entregables del proyecto

En este trabajo, los siguientes entregables van a ser desarrollados para cumplir con los objetivos propuestos:

- **Modelo de clasificación:** Para llevar a cabo una de las tareas del control, se utiliza un modelo de Deep Learning para estimar la posición relativa de la pelota respecto al robot, de manera discreta (izquierda, derecha o centro). Este entregable incluye los códigos utilizados para construir y entrenar el modelo, así como el código utilizado para realizar inferencias con los distintos modelos de clasificación.
- **Máquinas de estado:** Este entregable incluye los códigos correspondientes a las máquinas de estado construidas para controlar el robot en sus distintas etapas del desarrollo.
- **Repositorio público:** A modo de conjunción de los distintos elementos de este trabajo, se ha creado un repositorio de acceso público en la plataforma GitHub. En este enlace `https://github.com/Gabriel-Mourad/TFM`, se puede encontrar todo el software desarrollado, así como parte de los resultados obtenidos en formato multimedia.

## 1.5. Motivación del trabajo

Este proyecto puede considerarse un trabajo continuista del Trabajo Final de Grado que realicé en el curso 2020-2021, titulado «Técnica de procesado de imagen y Deep Learning para la estimación de madurez de productos agrícolas». En dicho trabajo aprendí a trabajar con redes neuronales y utilizarlas para resolver un problema de segmentación semántica: detectar en la imagen formas de productos agrícolas como el limón y a su vez, estimar su estado de madurez.

Aunque puedan parecer trabajos muy distintos, la motivación en ambos es la misma, la modernización del sector agrícola. Detrás de la gran mayoría de productos y servicios cotidianos en la vida del ser humano, se encuentran procesos automatizados e industrializados. Aun así, uno de los sectores más importantes y necesarios para el sustento de la vida humana, el sector agrícola, sigue dependiendo en su gran mayoría de la mano de obra humana, resultando ser un sector limitado e insostenible debido a los requisitos de alta producción a bajo coste.

Una vez superadas las asignaturas de este Máster, el deseo de seguir explorando materias como control e inteligencia artificial han dado origen a la propuesta de este trabajo. Por otro lado, uno de los desafíos más complejos y ambiciosos que ofrece la automatización del sector agrícola consiste en la detección y recolección del fruto. Se podría decir que el primer trabajo de final de estudios está enfocado a la problemática de detectar e identificar, mediante visión artificial, el producto agrícola. En este trabajo se explora el uso de robots cuadrúpedos y se desarrollan estrategias de movimiento para resolver problemas de búsqueda de objetos, pudiendo servir de base para la problemática de la recolección del fruto.

Así pues, el origen más profundo de este trabajo se halla en el deseo de explorar y aprender sobre el conjunto de técnicas y tecnologías existentes, capaces de contribuir al desarrollo del sector agrícola.

# 2. Robótica y Machine Learning

## 2.1. Robots cuadrúpedos

El concepto «robot» nació en una obra de teatro de ciencia ficción checa en el año 1921 [17]. En esta obra, el escritor Karel Capek describía a humanos artificiales encargados de aligerar la carga a los trabajadores. A dichos humanos artificiales se les llamó «robots», derivado de la palabra checa, «robota», que significa servidumbre [18]. Unas décadas más tarde, el concepto «robot» dejó de ser únicamente un recurso de ciencia ficción y se convirtió en algo real y accesible.

Los primeros avances en el campo de la robótica andante se consiguieron en torno al año 1970 por dos famosos investigadores, Kato y Vukobratovic. En Japón, 1973, se presentó el primer robot antropomórfico, el WABOT 1 por parte de Kato y su equipo en la universidad de Waseda. Utilizando un control simple, el robot era capaz de realizar pequeños pasos en equilibrio estático [19].

En paralelo, Vukobratovic y su equipo del instituto Mihailo Puppin, Belgrado, diseñaron el primer exoesqueleto y otros dispositivos enfocados al apoyo en rehabilitación funcional. En 1972 presentáron el concepto punto de momento-cero (ZMP), un desarrollo para formalizar la necesidad de estabilidad dinámica en robots andantes [20].

Estudios posteriores, como el de Waldron y McGhee, dejaron en evidencia la importancia de desarrollar este ámbito, pues las piernas proveen al robot de una suspensión activa, de modo que el movimiento del cuerpo principal del robot queda disociado del terreno, permitiendo movilidad y locomoción en lugares inalcanzables hasta el momento [21].

Los robots andantes están formados por un cuerpo central (también llamado torso) con piernas unidas a él. Los más comunes son los monópodos, bípedos, cuadrúpedos y hexápodos, con una, dos, cuatro y seis piernas respectivamente [22].

Desde entonces y hasta la actualidad, el campo de la robótica móvil no ha dejado de evolucionar. A continuación se muestra, en forma de imágenes, alguno de los hitos en el desarrollo cronológico que ha seguido este campo:

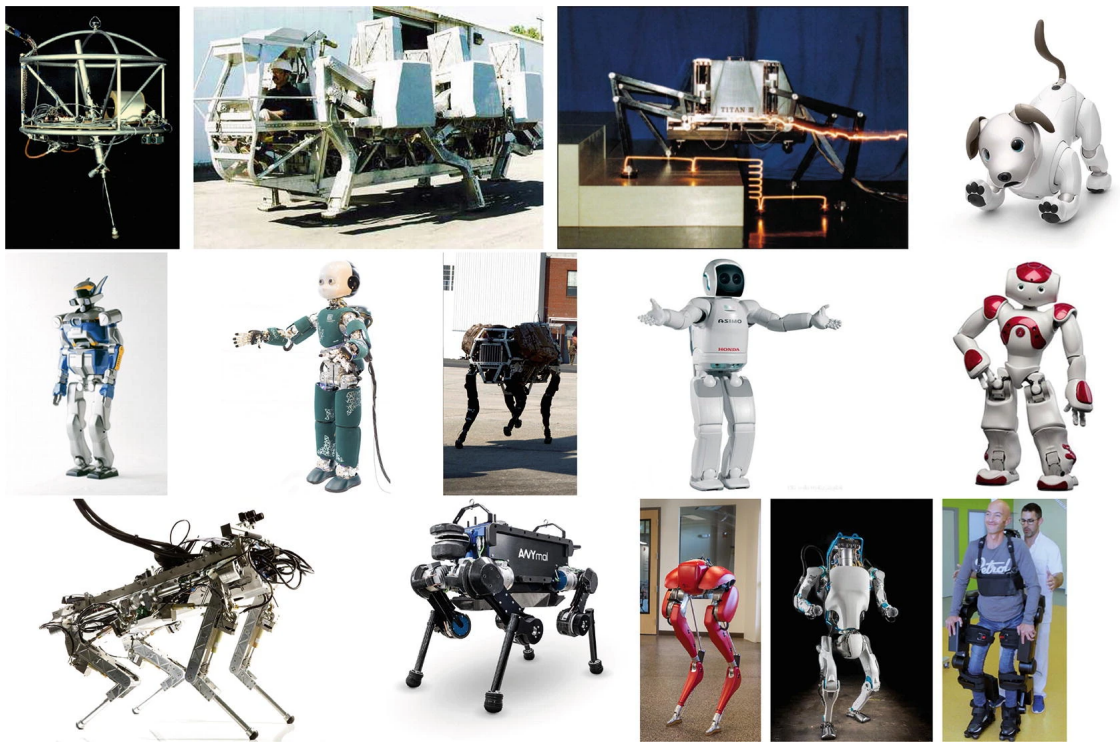


**Figure 2.1.:** Ejemplos de robots andantes. De izquierda a derecha y de arriba a abajo: Raibert's 3D hopper [23]; Adaptive Suspension Vehicle [21]; Titan III [24]; AIBO [25]; HRP-2 [26]; iCub [27]; BigDog [28]; ASIMO [29]; NAO [30]; HyQ [31]; ANYmal [32]; Cassie [33]; ATLAS [34]; self-balancing exo-skeleton [35]

La locomoción de los seres vivos se ha manifestado con gran variedad en la naturaleza, una de las más utilizadas por la gran mayoría de mamíferos es la marcha cuadrúpeda. El andar es el resultado de múltiples movimientos creados por numerosos procesos de extensión y contracción muscular [36]. Los robots cuadrúpedos están inspirados en este fenómeno, de modo que se obtienen sistemas con gran número de grados de libertad, flexibilidad y estabilidad [36].

La principal ventaja del movimiento cuadrúpedo no es únicamente proveer más estabilidad al sistema, también permite alcanzar más velocidad y mayor capacidad de carga, además de ser un sistema más sencillo de controlar, diseñar y mantener en comparación a robots de dos o seis piernas [37]. Por este motivo, el desarrollo de robots cuadrúpedos siempre ha sido el más valorado [11].

La dinámica de estos sistemas se basa en mantener la estructura en equilibrio. La locomoción se consigue generando fuerzas en el punto de contacto con el ambiente. La ecuación de Newton muestra claramente la necesidad de generar fuerzas externas $f_i$ para mover el centro de masa $c$, de un cuerpo de masa $m$, en una dirección distinta a la gravedad $g$ [19].

$$m(\ddot{c} - g) = \sum f_i \tag{2.1}$$

Por otro lado, la ecuación de Euler muestra que la posición de los puntos de contacto $s_i$ con respecto al centro de masa son de vital importancia para mantener el momento angular $L$ del robot controlado.

$$\dot{L} = \sum (s_i - c) \times f_i \tag{2.2}$$

La estabilidad durante el movimiento del robot se garantiza si el centro de masa del sistema se proyecta verticalmente dentro del polígono de soporte [38] como muestra la figura 2.2. De esta manera, movimientos cíclicos y puntos de equilibrios son sencillos de calcular, por lo tanto, si el robot es capaz de alcanzar el equilibrio en pocos pasos, entonces su locomoción es viable [39]. De lo contrario, se ha demostrado que si el robot es incapaz de alcanzar el equilibrio en dos pasos, la caída del mismo está garantizada [40].

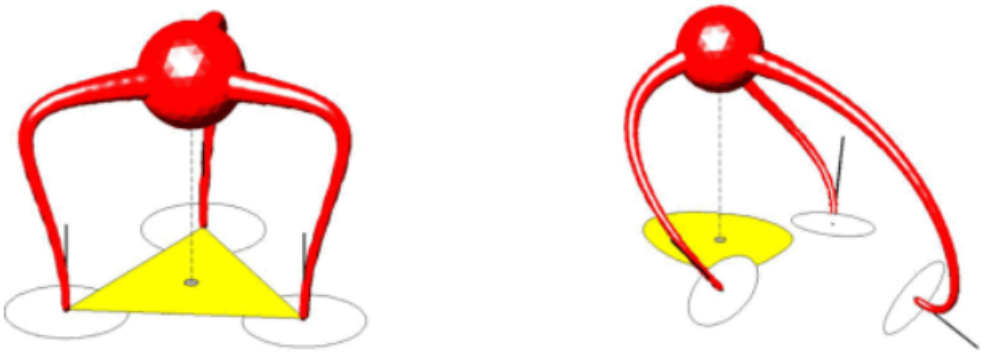

**Figure 2.2.:** Según la posición de los puntos de contacto, la proyección vertical del centro de masa cae dentro (izquierda) o fuera (derecha) del polígono de apoyo [38]

Hay un gran número de aplicaciones para los robots cuadrúpedos, algunas de las más prometedoras son el servicio de emergencias, inspección, mantenimiento, construcción y seguridad [22]. Uno de los avances más pioneros en este sector lo ha llevado a cabo la empresa Boston Dynamics, con el robot Spot. Caminar, trotar, evadir obstáculos y subir escaleras son algunas de las tareas que este robot resuelve con facilidad. Consta de potentes placas de control y cinco módulos sensoriales localizados en cada lado del robot. Esto le permite observar el espacio que le rodea en cualquier dirección. Cada pierna incorpora 3 motores (uno en cada articulación), y es capaz de alcanzar velocidades de 1.6 metros por segundo. Por último, mencionar que puede operar durante 90 min y su capacidad de carga es de 14 kg [41].

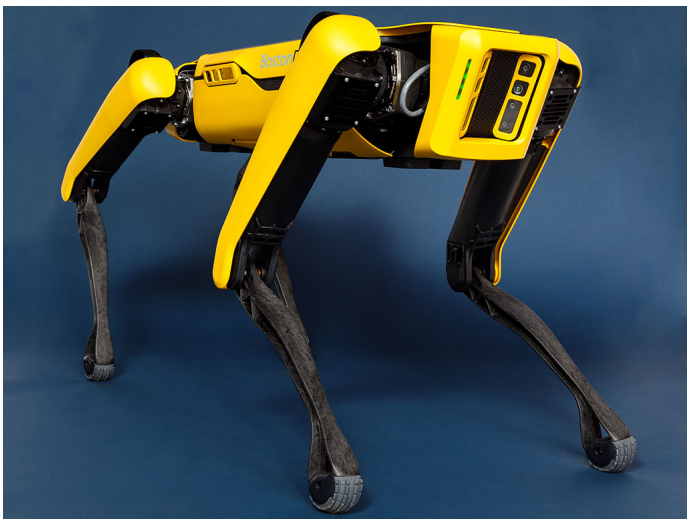

**Figure 2.3.:** Robot cuadrúpedo Spot de Boston Dynamics [41]

El robot Spot tiene incorporado un sistema de visión 3D con un sistema de localización y mapeo (SLAM), que proporciona al robot información del entorno para evitar obstáculos o colisiones [37].

Durante las últimas décadas, los robots se han usado mayoritariamente en aplicaciones industriales, realizando tareas como soldadura, ensamblaje y metalurgia. Todas estas aplicaciones tienen la misma limitación, pues el robot únicamente puede realizar la misma operación una y otra vez, y el éxito en estas, se garantiza siempre y cuando el entorno de trabajo se mantenga invariable [42].

Gracias al desarrollo de la visión por computador y el Deep Learning, hoy en día es posible dotar a los ordenadores del sentido de la vista, permitiendo obtener sistemas capaces de identificar y clasificar patrones encontrados en datos de tipo imagen o video. Cuando un sistema de visión artificial se combina con un robot, se libera a este de las restricciones impuestas por el espacio y el tiempo, permitiendo al robot realizar distintos tipos de tareas según el entorno percibido [42, 43].

Este tipo de investigaciones necesita de una estructura de robot cuadrúpedo para realizar pruebas de funcionamiento y experimentos. Así pues, con fines didácticos y en fases iniciales de investigación, es común el uso de pequeñas estructuras comerciales, que permiten implementar y experimentar los algoritmos de control diseñados a un bajo coste. En las siguientes secciones se muestran algunas estructuras de robot cuadrúpedo disponibles en el mercado.

### 2.1.1. Bittle

Bittle es el robot cuadrúpedo desarrollado por Petoi [12], es una plataforma de código abierto, dispone de una interfaz compatible con Raspberry Pi, permitiendo desarrollar proyectos de alta complejidad. A continuación se detallan algunas características de este sistema.


**Figure 2.4.:** Perspectiva general del robot cuadrúpedo Bittle [12]

La estructura del Bittle está formada por distintas piezas de plástico, que ensambladas entre sí, configuran el cuerpo del robot. Consta de cuatro patas y nueve articulaciones (dos articulaciones en cada pierna y una en el cuello). Los movimientos del Bittle se realizan mediante servomotores incorporados en cada articulación. Al combinar determinadas series de giros en cada servomotor, se construyen cadenas de movimiento, permitiendo al Bittle caminar, gatear, trotar e incluso realizar piruetas o posturas.

**Figure 2.5.:** Ensamblaje estructural del robot Bittle [44]

**NyBoard** NyBoard es la placa de control de movimiento del Bittle y es compatible con Arduino. A ella, están conectados todos los servomotores del robot y dispone de una interfaz compatible con Raspberry Pi mediante su puerto serie. El uso de una Raspberry Pi permite analizar más datos sensoriales, conectar el sistema a Internet y diseñar modelos de control más complejos.

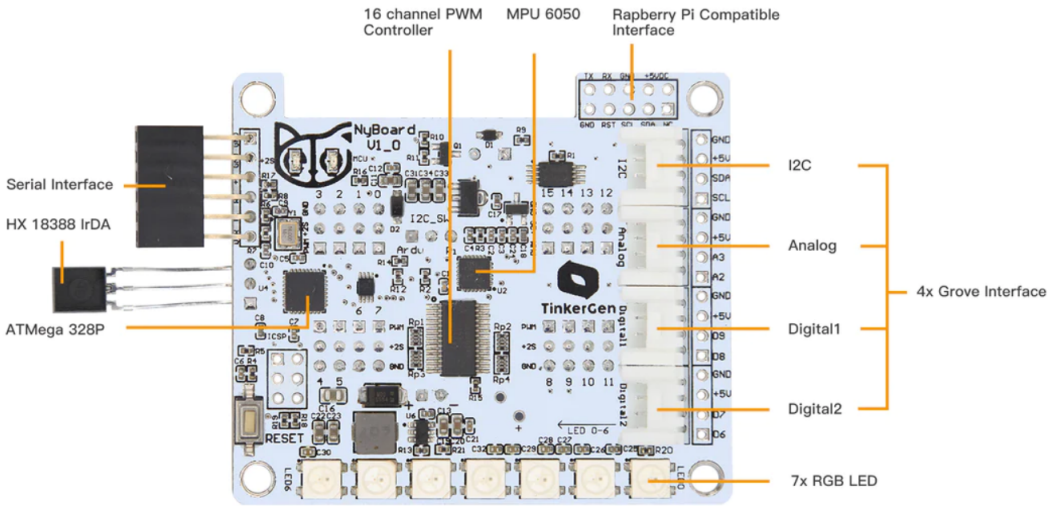


**Figure 2.6.:** Perspectiva general de la placa de control NyBoard [15]

**OpenCat** OpenCat es la librería de código abierto desarrollada por Petoi. A través de ella es posible controlar la locomoción del robot mediante instrucciones simples,

transmitidas a través del puerto serie. Es compatible con Python y C/C++.

**OpenCat**
**Communication Protocol and Parsing**

| Interface | | Token | Encoding | Parameters | Format | Bytes | Function |
|---|---|---|---|---|---|---|---|
| RasPi Serial Port | Arduino Serial Monitor | 'h' | Ascii | | char | 1 | print help information |
| | | 'c' | | idx*,angle** '\n' | string | strlen + 2 | calibrate $servo_{idx}$ by angle |
| | | 'm' | | idx*,angle** '\n' | string | strlen + 2 | move $servo_{idx}$ to angle |
| | | 'j' | | | char | 1 | show all 16 joint angles |
| | | 'd' | | | char | 1 | shut down servos |
| | | 'p' | | | char | 1 | pause motion |
| | | 'a' | | | char | 1 | abandon calibration |
| | | 's' | | | char | 1 | save calibration |
| | | 'k' | | abbreviation '\n' | string | strlen + 2 | load skill |
| | | 'w' | | command '\n' | string | strlen + 2 | some future command words |
| | | 'r' | | | char | 1 | reset board |
| | | 'i' | Binary | $idx_1$ $a_1$ ... $idx_N$ $a_N$ '~' | string | strlen + 2 | list of indexed rotation angles |
| | | 'I' | | $a_1$ $a_2$ ... $a_{DoF}$ '~' | string | DoF + 2 | list of all DoF rotation angles |

* index range: 0 ~ (DoF - 1)

** angle range: -90 ~ 90. fits in the range of signed char (-128 ~ 127). Also depends on the servos' parameters

**Figure 2.7.:** Protocolo de comunicación OpenCat [45]

Es posible acceder al monitor serie de la placa y enviar instrucciones, cada instrucción se inicia indicando un *token*, seguido de los parámetros indicados en la figura 2.7, de esta manera es posible acceder al conjunto de funciones que puede realizar el Bittle.

Por ejemplo, el *token* «m» está reservado para mover cada servomotor a una determinada posición angular. Como se puede observar en la columna de parámetros, junto al *token*, hay que indicar el índice del servomotor, y el ángulo deseado. Por lo tanto, si se envía por el puerto serie la cadena de caracteres «m0 30», el servomotor nº0 se moverá a la posición 30º.

| | Type | Command | Note |
|---|---|---|---|
| Control | Token | d | rest and shutdown all servos |
| | | g | turn on/off gyro to boost speed |
| | | p | pause motion and shut off all servos |
| | | c | enter calibration mode |
| | | m | move a joint to certain angle |
| Skill | Gait | bk | back |
| | | bkL | backLeft |
| | | bkR | backRight |
| | | vt | stepping on the same spot |
| | | crF | crawl |
| | | crL | crawl left |
| | | crR | craw right |
| | | wkF | walk |
| | | wkL | walk left |
| | | wkR | walk right |
| | | trF | trot |
| | | trL | trot left |
| | | trR | trot right |
| | | bdF | bound (not recommanded) |
| | Posture | balance | |
| | | buttUp | buttom up |
| | | calib | calibration |
| | | rest | |
| | | sit | |
| | | sleep | |
| | | str | stretch |
| | | zero | |
| | Behavior | ck | check around |
| | | hi | hi sequence |
| | | pee | pee sequence |
| | | pu | push up sequence |
| | | rc | recovering sequence |
| | | pd | play dead |
| | | bf | back flip |

**Figure 2.8.:** Lista de comandos prediseñados en OpenCat [45]

En la figura 2.7 se ha mostrado el protocolo general de comunicación, en la figura 2.8 se muestra la lista de comandos y sus funciones. Las habilidades o *skills* se indican bajo el *token* «k», a continuación se debe indicar la abreviatura correspondiente al movimiento que se desea realizar con el Bittle. Por ejemplo, si se desea caminar a la derecha, hay que enviar por el puerto serie la cadena de caracteres «kwkR». Siguiendo esta metodología, el robot puede realizar multitud de gateos, posturas y piruetas.

### 2.1.2. Wavego

El robot cuadrúpedo Wavego es una estructura comercial ligera desarrollada por Waveshare. Dispone de servomotores de gran capacidad de par, doce grados de libertad, un sensor cámara incorporado en la parte frontal y una estructura robusta. Utiliza un módulo ESP32 [46] a modo de sub-controlador del sistema, encargado de manejar las funciones locomotrices, así como la comunicación con el controlador principal. Para el desarrollo de controles y funciones de más alto nivel, se dispone de una interfaz compatible con Raspberry Pi, permitiendo incorporar más periféricos al sistema, trabajar con conjuntos de datos más extensos y desarrollar algoritmos de control más complejos [47].

**Figure 2.9.:** Robot cuadrúpedo Wavego

### 2.1.3. Freenove

La marca Freenove también ofrece una estructura de robot cuadrúpedo de código abierto compatible con Raspberry Pi. Es una estructura ligera de doce grados de libertad, tiene incorporado un sensor cámara y un sensor de ultrasonido en la parte frontal. Es un sistema ideal para aplicaciones didácticas de robótica, programación y electrónica. Es una herramienta para convertir ideas y diseños, en prototipos funcionales de bajo coste [48].

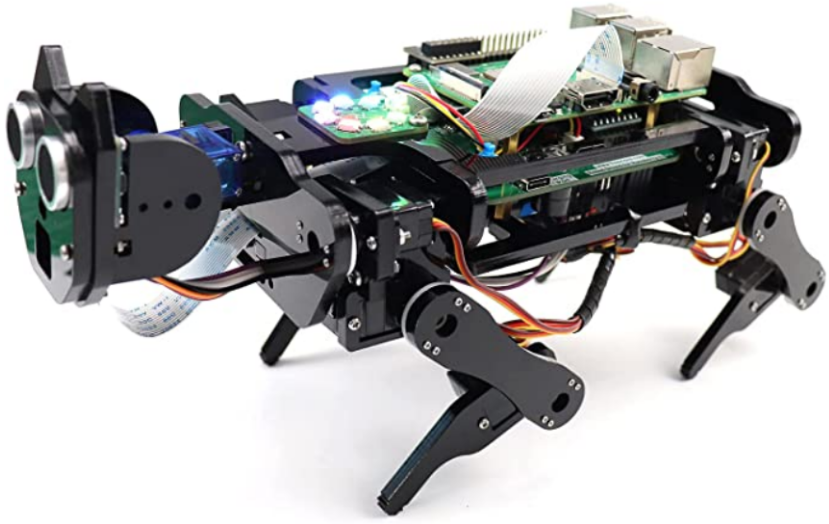

**Figure 2.10.:** Estructura de robot cuadrúpedo Freenove

## 2.2. Visión por ordenador

La inteligencia artificial es la rama de la ciencia computacional que tiene como objetivo dotar a las máquinas de la habilidad de realizar tareas cognitivas e inteligentes. Comparable a la manera en que la Primera Revolución Industrial dio lugar a una era de máquinas capaces de realizar tareas físicas, la inteligencia artificial desarrolla máquinas con habilidades cognitivas [49].

La visión por ordenador es un campo de la inteligencia artificial que permite a los ordenadores extraer información de imágenes y vídeos digitales. Se podría decir que la visión por ordenador permite a los computadores ver, observar y entender, permitiendo clasificar objetos, identificar la distancia a éstos, apreciar su movimiento o incluso detectar si hay algo inusual en la imagen.

La visión humana realiza estas tareas utilizando retinas, nervios ópticos y el córtex visual, la visión por ordenador, en cambio, utiliza cámaras, datos y algoritmos. De este modo es posible analizar gran cantidad de datos visuales en intervalos de tiempo mucho más cortos en comparación a la visión humana [50].

El contenido básico de cada imagen consiste en píxeles, ejes, ángulos, figuras geométricas, colores, formas y texturas [51]. Existen gran variedad de técnicas y algoritmos capaces de extraer e interpretar información de imágenes digitales.

**Figure 2.11.:** Técnica de visión por ordenador *Divide and Conquer* [51]

Dos tecnologías son esenciales para llevar a cabo la visión por ordenador, el Machine Learning y las redes neuronales convolucionales [50]. Ambas tecnologías se encuentran detalladas y profundizadas en las secciones posteriores.

## 2.3. Machine Learning

El Machine Learning es la rama de la inteligencia artificial que permite a los ordenadores aprender sin ser explícitamente programados [49]. Se aplica en modelos estadísticos para detectar patrones en conjuntos de datos sin utilizar comandos directos de programación. En la figura 2.12 se muestra una comparativa conceptual entre ML y la programación clásica.

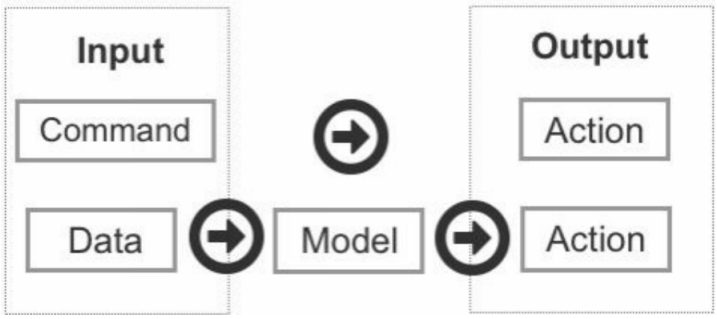


**Figure 2.12.:** Comparativa entre ML y programación clásica

Por un lado, la programación clásica toma como parámetro de entrada una línea de comando, esto conlleva una única salida o respuesta. Es decir, un comando directo con una respuesta directa. En cambio, en ML encontramos como parámetros de entrada conjuntos de datos, que se utilizan para alimentar un modelo previamente configurado y entrenado, que responde con una predicción o estimación sobre el conjunto de datos de entrada [49]. En este caso y a diferencia de la programación clásica, el modelo toma decisiones basándose en la experiencia adquirida durante las fases de entrenamiento, de forma similar al proceso de toma de decisiones de los humanos.

El ML incorpora gran cantidad de modelos estadísticos, y elegir cuál o qué combinación de ellos es el adecuado para resolver un problema, es uno de los primeros desafíos que se encuentran los trabajadores de este campo. En el capítulo *ML Categories* del libro *Machine Learning for absolute begginers* [49] se describen las distintas maneras de llevar a cabo el entrenamiento de un modelo para realizar una determinada tarea.

- **Aprendizaje supervisado:** Esta rama del ML aprende patrones relacionando variables de entrada con valores de salida esperados. Este conjunto de datos está etiquetado, por lo tanto, el modelo conoce el valor real de estos. Así pues, el aprendizaje supervisado funciona alimentando el modelo con un conjunto de datos (con varias características) y su correcto valor de salida. El algoritmo, entonces, descifra patrones ocultos en el conjunto de datos y se configura para reproducir el mismo tipo de criterio y así, predecir valores de salida con nuevos datos de entrada.
- **Aprendizaje no supervisado:** En este tipo de aprendizaje, no todas las variables y patrones son clasificados. En cambio, el modelo es capaz de encontrar patrones ocultos y crear sus propias etiquetas. La ventaja del aprendizaje no supervisado es la posibilidad de encontrar patrones, desconocidos hasta el momento, en el conjunto de datos.
- **Aprendizaje reforzado:** A diferencia del aprendizaje supervisado y no supervisado, esta metodología de aprendizaje mejora el modelo continuamente aprovechando el resultado de anteriores iteraciones. El modelo se centra en

aprender el valor de distintas acciones realizadas bajo diferentes condiciones. Según el tipo de problema que se desee resolver, el algoritmo se configurará para maximizar o minimizar determinados valores, obteniendo así mejora constante basada en el aprendizaje realizado y las anteriores iteraciones.

### 2.3.1. *Pattern recognition*

El reconocimiento de patrones se ocupa del descubrimiento automático de regularidades en conjuntos de datos usando algoritmos computacionales, y del uso de estas regularidades para llevar a cabo acciones de clasificación o regresión de datos [52].

Las técnicas de ML utilizan grandes conjuntos de datos organizados en forma de vector $X = \{x_1, ..., x_N\}$, llamados conjunto de entrenamiento, para configurar los parámetros de un modelo. Las diferentes categorías del conjunto de datos de entrenamiento se conocen de antemano, normalmente utilizando etiquetas en cada dato. El resultado de compilación del algoritmo puede expresarse como una función $Y(X)$ que toma un vector de datos como entrada, y devuelve un vector de salida codificado según determine el problema de clasificación. Una vez el modelo es entrenado, puede clasificar nuevos datos de entrada, que conforman el conjunto de datos de *test*. La habilidad de categorizar nuevas entradas que difieren de las usadas en el conjunto de datos de entrenamiento se llama *generalización*. En la práctica, la variabilidad de los datos de entrada será mucho más grande que el conjunto de entrenamiento, así pues, la generalización es un objetivo esencial en el reconocimiento de patrones [52].

En la mayoría de casos, se realiza un pre-procesado de los datos de entrada, transformándolos en un nuevo espacio de variables, donde se espera que el reconocimiento de patrones sea más sencillo. Es importante remarcar que el nuevo conjunto de datos que se utilice posteriormente al entrenamiento, también deberá ser pre-procesado del mismo modo.

También es recomendable realizar preprocesamiento de datos con el objetivo de agilizar la computación. Por ejemplo, en aplicaciones de clasificación de imagen en tiempo real, el ordenador deberá analizar un gran número de píxeles por segundo y alimentar al modelo de ML utilizado, llegando a ser una tarea computacionalmente inviable. En cambio, aplicando un pre-procesado de datos es posible extraer las características más representativas del conjunto de datos. Alimentar al modelo con dichas características aumenta extremadamente la eficiencia sin perder precisión en la clasificación [53].

Existen varios modelos capaces de analizar conjuntos de datos y extraer patrones o regularidades en ellos. Uno de los más extendidos actualmente, se basa en aplicar conceptos biológicos para encontrar patrones, estos modelos se conocen como redes neuronales artificiales [54].

## 2.4. Redes neuronales artificiales

Como se ha explicado anteriormente, el modelo matemático en ML tiene la tarea de aprender o encontrar los patrones escondidos en el conjunto de datos. El aprendizaje de una red neuronal puede entenderse como un problema de optimización no lineal donde se busca la configuración de parámetros que minimiza la función de coste. Esta búsqueda de parámetros es lo que se conoce como entrenamiento.

Las redes neuronales artificiales suelen entrenarse por épocas, en cada una de ellas, todo el conjunto de entrenamiento se ha procesado utilizando el modelo una sola vez. Al finalizar el entrenamiento, es decir, un determinado número de épocas, el modelo ha obtenido la habilidad de generalizar [55] (concepto definido en la sección 2.3.1).

En esta sección se explora el mundo de las redes neuronales. A continuación se presenta su unidad básica, así como sus arquitecturas y los algoritmos de aprendizaje.

### 2.4.1. Neuronas

Puesto que las redes neuronales artificiales son modelos matemáticos que mimetizan y replican los procesos biológicos que se llevan a cabo en el cerebro humano, es conveniente entender como funciona nuestro cerebro y de qué manera lleva a cabo tareas tan instintivas como el aprendizaje.

El cerebro humano lo forman alrededor de $10^{11}$ neuronas. La neurona, o nervio celular, es la unidad fundamental del sistema nervioso. Las neuronas son extensiones celulares formadas por cuatro componentes: las dendritas, el soma, el axón y la sinapsis. El soma contiene el núcleo celular. Las dendritas se ramifican alrededor del soma y sirven para recibir las señales de entrada procedentes de otras neuronas. El axón sirve de canal de salida, este se conecta a las dendritas de otras neuronas. Dicha conexión se conoce como sinapsis.

Las neuronas reciben señales de otras neuronas a través del soma y las dendritas, se realiza un cálculo integral sobre ellas y se envía la señal de salida a otras neuronas a través del axón. Recibiendo señales de distintas neuronas vecinas y transmitiendo el cálculo a nuevas neuronas, se forma una red neuronal [56]. Un diagrama esquemático de la estructura de una neurona se muestra en la figura 2.13.

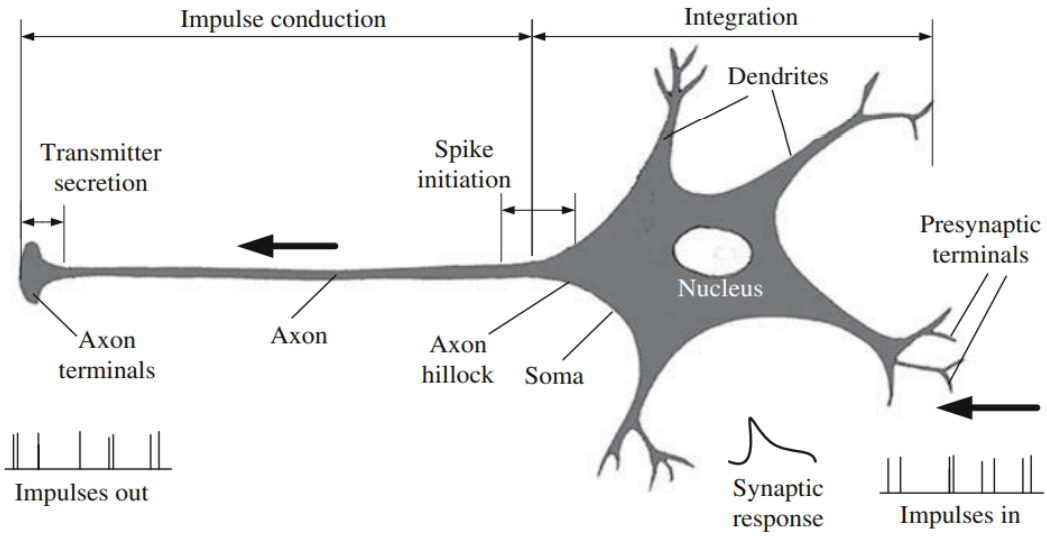


**Figure 2.13.:** Esquema de una neurona

Las conexiones sinápticas muestran plasticidad, es decir, aquellas conexiones utilizadas con más frecuencia en respuesta a determinados patrones adquieren más fuerza, por otro lado, aquellas conexiones menos utilizadas se vuelven más débiles [57]. Este mecanismo forma la base del aprendizaje en el cerebro humano y es en el que se inspira el mecanismo de aprendizaje de una red neuronal artificial.

### 2.4.2. El perceptrón

Las redes neuronales artificiales, al igual que el cerebro humano, están compuestas de unidades básicas de procesamiento. En el cuerpo humano, este papel lo llevan a cabo las neuronas, y en las redes neuronales artificiales, los perceptrones. Aunque el ML se podría considerar una tecnología de vanguardia, el pereceptrón es un elemento matemático desarrollado en torno al año 1960 por Rosenblatt. En el artículo *The perceptron: a probabilistic model for information storage and organitzation in the brain* [58] se diseña un hipotético sistema nervioso, llamado perceptrón, para ilustrar algunas propiedades fundamentales de los sistemas inteligentes.

En este primer diseño, el perceptrón consiste en un elemento matemático que recibe un conjunto de señales o datos de entrada, los procesa, y envía una respuesta. Según el cálculo procesado, el conjunto de respuestas puede ser excitatorio o inhibitorio, dicha característica es adquirida según los patrones o regularidades existentes en los datos de entrada. En la figura 2.14 se puede apreciar un esquema representativo del perceptrón de Rosenblatt.

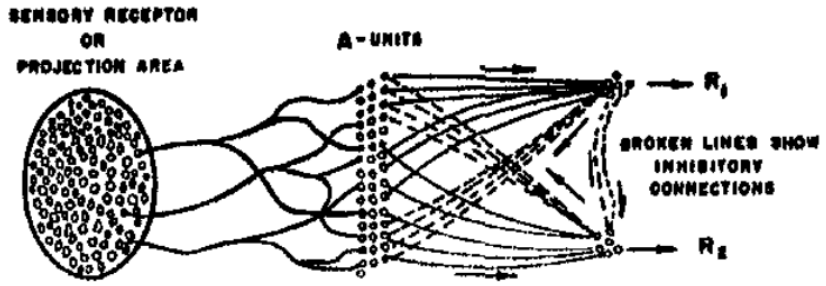


**Figure 2.14.:** Esquema del perceptrón planteado por Rosenblatt [58]

Con el desarrollo del ML, estas investigaciones se han formalizado y han dado lugar a la red neuronal más simple:

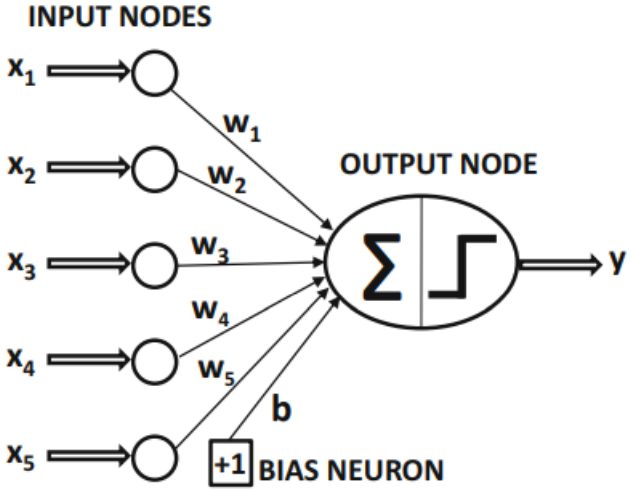


**Figure 2.15.:** Modelo matemático del perceptrón [59]

En la figura 2.15 puede apreciarse el modelo matemático de la red neuronal más simple. Contiene una única capa de datos de entrada y un nodo de salida. La capa de entrada contiene $d$ nodos que transmiten $d$ características $\bar{X} = [x_1...x_d]$ con pesos ponderados $\bar{W} = [w_1...w_d]$ al nodo de salida. A continuación se computa la función lineal $\bar{W} \cdot \bar{X} = \sum_{i=1}^{d} w_i x_i$, para finalmente, evaluar el resultado en una función no lineal:

$$\hat{y} = \Phi\left(\sum_{j=1}^{d} w_j x_j\right) \tag{2.3}$$

Debido a la uso de una no linealidad es posible convertir el producto $\bar{W} \cdot \bar{X}$ en un identificador de clase. Es por este motivo que a este tipo de funciones se las conoce como funciones de activación. Existen diferentes funciones de activación para simular distintos tipos de modelo, elegir la función adecuada es crucial para obtener buenos resultados.

Como se puede ver en la figura 2.15, es común encontrarse con un sesgo o *bias*, éste aporta una parte de invarianza a la predicción. En los casos en que la media del conjunto de características de entrada está desviada de la media de la función de activación, incorporar un sesgo constante que capture dicha desviación es de gran utilidad para obtener un buen resultado en las predicciones del modelo [59]. En estos casos la ecuación 2.3 debe ser modificada de la siguiente manera:

$$\hat{y} = \Phi\left(\sum_{j=1}^{d} w_j x_j + b\right) \tag{2.4}$$

Las funciones de activación propagan la salida del nodo a la siguiente capa de entrada. Son funciones escalares que determinan un valor específico de salida y permiten introducir no linealidades en los modelos utilizados [60]. Así pues, la función de activación de una neurona define la manera en que dicho nodo se activa. Es importante saber que existen varias funciones de activación capaces de aportar distintos comportamientos a las neuronas, las que se presentan a continuación son algunas de las más utilizadas:

- **Lineal:** La figura 2.16 muestra la función de activación lineal, se define como $\Phi(z) = Wz$, donde la variable dependiente tiene una relación directamente proporcional con la variable independiente. El problema de este tipo de funciones de activación es que no permiten el aprendizaje de patrones no lineales [61].

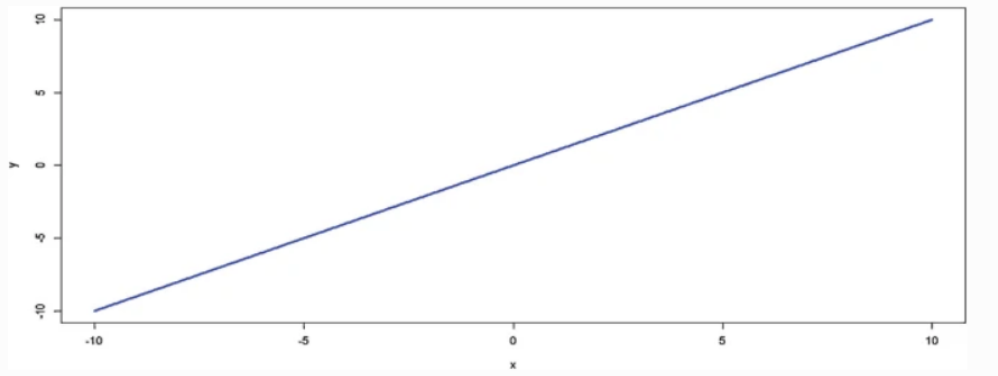


**Figure 2.16.:** Representación gráfica de una función de activación lineal

- **ReLU:** La función de activación ReLU (*Rectifier Linear Unit*) es una de las más populares. La función ReLU es plana para valores menores a un determinado sesgo (normalmente cero) y linear para valores superiores a él. Es decir, la función ReLU activa la neurona sólo si el valor de entrada es superior a una cierta cantidad, en dicho caso, la salida y la entrada tienen una relación lineal. En caso contrario, a la salida de la neurona se obtiene el valor cero. La función ReLU se define como $\Phi(z) = max(0, z)$ y su representación gráfica puede apreciarse en la figura 2.17. A pesar de su simplicidad, la función de activación ReLU aporta no linealidad al sistema [61]. La función de activación ReLU forma parte del estado del arte actual, ya que ha demostrado un mejor desempeño en el aprendizaje que la función de activación Sigmoide. La función ReLU es común encontrarla en las capas ocultas de la red neuronal, así como en las capas de salida cuando la variable de respuesta es continua y positiva [62].

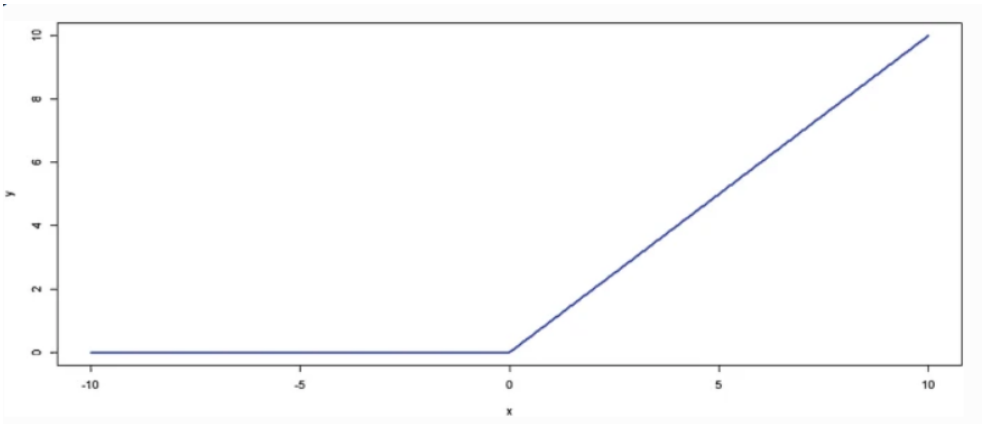


**Figure 2.17.:** Representación gráfica de la función de activación ReLU

- **Sigmoide:** La función de activación Sigmoide es una máquina que convierte una variable independiente de rango infinito en simples probabilidades de rango [0,1] donde la mayoría de salidas serán cercanas a 0 y 1. La función Sigmoide tiene la habilidad de reducir valores extremos o fuera de rango a datos aptos para alimentar el modelo, sin la necesidad de eliminar dichos valores. Esta función de activación se define como $\Phi(z) = (1 + e^{-z})^{-1}$ [60, 61]. Es una de las más utilizadas para resolver problemas de respuesta binaria [62]. En la figura 2.18 puede apreciarse el comportamiento de la función Sigmoide, tomando valores de salida cercanos a 1 o 0 siempre que la variable de entrada toma valores extremadamente positivos o negativos respectivamente.

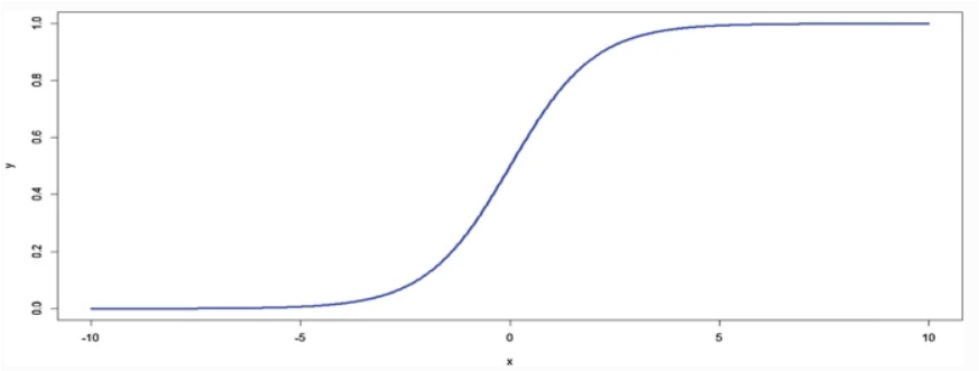


**Figure 2.18.:** Representación gráfica de la función de activación Sigmoide

- **Softmax:** La función de activación Softmax es una generalización de la función Sigmoide utilizada para llevar a cabo sistemas de respuesta multivariable. Normalmente es la función de activación que se encuentra en la capa de salida cuando el problema de clasificación involucra más de dos categorías. La función Softmax se define como:

$$\Phi(z_j) = \frac{exp(z_j)}{1 + \sum_{c=1}^{C} exp(z_c)}, j = 1, \ldots, C \tag{2.5}$$

  Extrapolando el comportamiento de la función Sigmoide, la función Softmax convierte un vector de dimensión C de valores reales y arbitrarios en un vector de dimensión C de valores reales comprendidos entre [0,1]. Tanto la función Sigmoide y Softmax permiten interpretar la respuesta del nodo como una distribución probabilística de cada categoría [62].

## 2.4.3. Tipologías de redes neuronales artificiales

En esta subsección se van a describir las tipologías de redes neuronales más populares. La arquitectura de la red neuronal representa el modo en que las neuronas se conectan para formar una red. La topología de la red neuronal juega un papel de vital importancia en su funcionalidad y rendimiento [62].

**Redes *feedforward:*** En este tipo de redes neuronales artificiales, la información fluye en una única dirección, desde las neuronas de entrada, a través de las capas de procesamiento, hasta llegar a la capa de salida de la red neuronal. Es decir, no hay conexión entre neuronas de la misma capa y tampoco hay transmisión de datos de las capas superiores a las inferiores. En la figura 2.19 se muestra el esquema de una tipología *feedforward*, en ella se puede apreciar una única capa oculta, pudiendo haber múltiples de ellas [62].

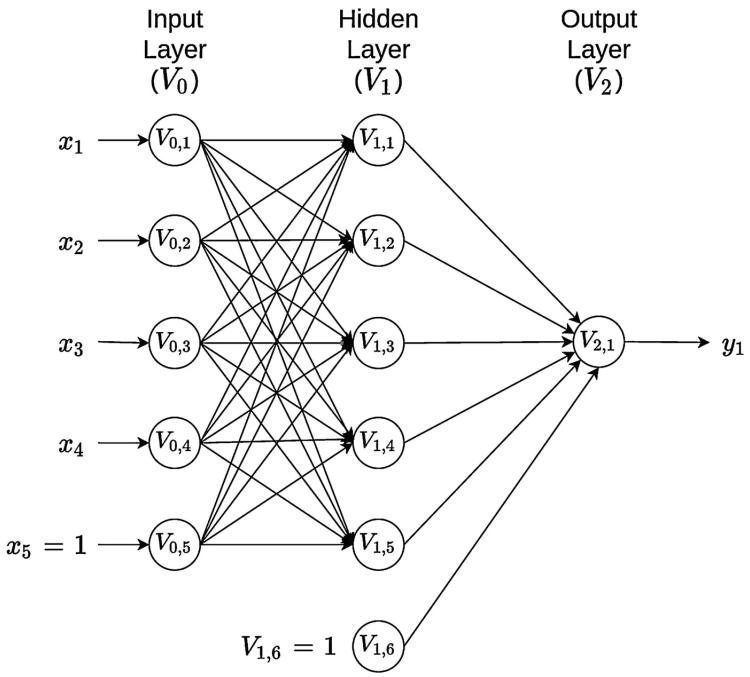


**Figure 2.19.:** Arquitectura de una red neuronal *feedforward* [62]

**Redes neuronales recurrentes:** En este tipo de arquitecturas la información no siempre fluye en una dirección, ya que puede existir realimentación de información a capas anteriores. Todas las neuronas tienen conexiones de entrada (provenientes de capas anteriores), conexiones de salida (dirigidas a capas posteriores) y conexiones recurrentes que propagan información a neuronas de otras capas. Este tipo de redes se usan frecuentemente en predicción de series temporales [62].

### 2.4.4. Algoritmo de aprendizaje: *Backpropagation*

Esta sección se centra en detallar de qué modo una red neuronal artificial aprende. En secciones anteriores se ha explicado que el aprendizaje del modelo se lleva a cabo durante la fase de entrenamiento, donde se ajustan los pesos de las conexiones de la red neuronal. Dado que la técnica de Deep Learning utilizada en el desarrollo práctico de este proyecto forma parte del aprendizaje supervisado, en este apartado del estado del arte únicamente se va a enfatizar el algoritmo de aprendizaje utilizado en este tipo de metodologías.

En aprendizaje supervisado los parámetros de la red neuronal se ajustan comparando directamente el valor de salida y el deseado. Se realiza un lazo de realimentación cerrado con la señal de error. El proceso de aprendizaje es guiado por la medida de error obtenida al comparar la salida calculada y la deseada [55]. En la figura 2.20 se presenta un esquema del proceso de aprendizaje supervisado.

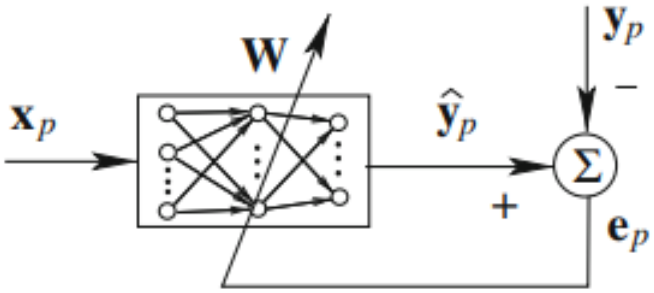


**Figure 2.20.:** Lazo de realimentación en aprendizaje supervisado [55]

Así pues, el aprendizaje de una red neuronal artificial puede entenderse como un problema de optimización en que se desea minimizar la función de error, también llamada función de pérdida u objetivo [62]. En términos generales, la función de pérdida mapea un valor variable o multivariable, en un número real que representa el coste asociado ha dicho evento. Es decir, se intenta cuantificar la cercanía entre el valor predicho por el modelo y el valor real [63]. A medida que la función de pérdida decrece, la estimación del modelo y la realidad se acercan. En otras palabras, un modelo será más preciso cuando su función de coste tome valores más pequeños.

Los procesos de optimización utilizados se basan en funciones del tipo:

$$J(W) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\hat{y}_i, y_i) \tag{2.6}$$

Donde $N$ corresponde al tamaño del conjunto de datos de entrada, $y$ es el valor esperado de salida de la red neuronal e $\hat{y}$ la respuesta de salida predicha por el modelo [55]. Este tipo de funciones depende de la entrada de datos y de la configuración de pesos del modelo $f(x_i, W)$. A continuación, se presentan las funciones de pérdida $\mathcal{L}(\hat{y}_i, y_i)$ más utilizadas para variables de respuesta continua [61]:

- **Error cuadrático medio:** Asumiendo que se deseen predecir $L$ categorías, la función de pérdida MSE (*Mean Square Error*) se describe cómo:

$$\mathcal{L}(\hat{y}_i, y_i) = \sum_{j=1}^{L}(\hat{y}_{ij} - y_{ij})^2 \tag{2.7}$$

- **Sumatorio del error absoluto:** Con el fin de utilizar como parámetro de coste el error absoluto entre la salida predicha y la esperada, para una salida de $L$ categorías la función de pérdida SAPE (*Sum of Absolute Perentage Error*) se describe:

$$\mathcal{L}(\hat{y}_i, y_i) = \sum_{j=1}^{L}\left|\frac{\hat{y}_{ij} - y_{ij}}{y_{ij}}\right| \tag{2.8}$$

- **Error cuadrático logarítmico:** Similar a la función de pérdida MSE, el error cuadrático logarítmico se describe como:

$$\mathcal{L}(\hat{y}_i, y_i) = \sum_{j=1}^{L} (log(\hat{y}_{ij}) - log(y_{ij}))^2 \tag{2.9}$$

Aunque la función de pérdida MSE tiene el inconveniente de ser muy sensible a valores extremos, es una de las más utilizadas en modelos de Deep Learning debido a sus propiedades matemáticas y la sencillez computacional [62].

En modelos de respuesta binaria u ordinal, es decir, modelos con la finalidad de clasificar dos o más categorías del conjunto de datos, es común encontrar funciones de pérdida logísticas de la forma:

$$\mathcal{L}(\hat{y}_i, y_i) = -\sum_{j=1}^{L} [y_{ij} \times \log(\hat{y}_{ij}) + (1 - y_{ij}) \times \log(1 - \hat{y}_{ij})] \tag{2.10}$$

En modelos de clasificación binaria, la ecuación representa la diferencia entre dos distribuciones probabilísticas, una medida de la divergencia entre la distribución estimada y observada [62]. Cuando el número de clases del conjunto de datos es mayor a dos [61], la función de pérdida logística se describe como:

$$\mathcal{L}(\hat{y}_i, y_i) = -\sum_{j=1}^{L} [y_{ij} \times \log(\hat{y}_{ij})] \tag{2.11}$$

Una vez comprendido el concepto de la función de pérdida, el siguiente paso consiste en detallar el mecanismo de aprendizaje de la red neuronal. El entrenamiento de una red neuronal consiste en encontrar la combinación de pesos que consigue el mínimo coste en la función de pérdida [64]. Para un conjunto de pesos $W = (w_0, w_1, ...)$, el problema de optimización puede formalizarse de la siguiente manera:

$$W = argmin(J(W)) \tag{2.12}$$

Una representación gráfica de la función de pérdida de una red neuronal con dos conexiones o pesos, se muestra en la figura 2.21:

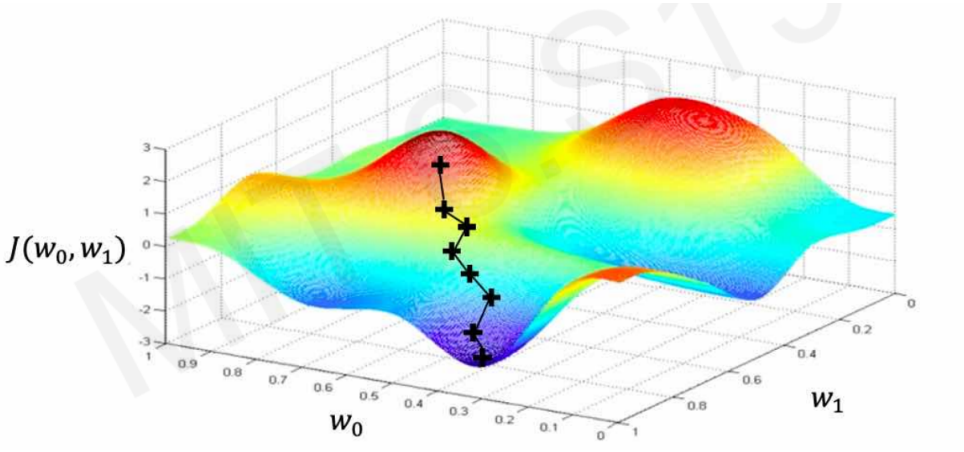


**Figure 2.21.:** En el eje vertical la función de pérdida, y en los ejes horizontales las posibles combinaciones de pesos [64]

En el caso de estudio propuesto en la figura 2.21, el problema de optimización resulta mucho más intuitivo. Basta con encontrar la combinación de pesos que consigue un valor mínimo de la función de pérdida. El proceso de optimización llevado a cabo comienza eligiendo una configuración de pesos aleatoria. A continuación se calcula el gradiente $\frac{\partial J(W)}{\partial W}$, puesto que el gradiente de una función indica la dirección y sentido del máximo local, es posible realizar una pequeña modificación de los pesos en sentido opuesto al gradiente calculado. Repitiendo esta metodología, el algoritmo converge hasta alcanzar un mínimo relativo en la función de coste, hallando la combinación de pesos que consigue una respuesta del modelo más precisa [64, 62].

Debido a que este algoritmo utiliza el gradiente para descender en la función de pérdida, suele llamarse algoritmo de gradiente descendente, y puede presentarse de la siguiente forma:

1. Inicialización aleatoria de la configuración de pesos
2. Bucle hasta converger::
   a) Computar el gradiente, $\frac{\partial J(W)}{\partial W}$
   b) Actualizar los pesos, $W_t \leftarrow W_{t-1} - \eta \frac{\partial J(W)}{\partial W}$
3. Retorno de pesos

En el caso de estudio presentado en la figura 2.21, puede resultar sencillo calcular el gradiente de la función de pérdida cuando únicamente depende de dos pesos. Las redes neuronales utilizadas en los modelos de Deep Learning están formadas por varias capas, cada una de ellas constituida de varias neuronas, así pues, el número de pesos existentes suele ser una cifra de gran magnitud, y el gradiente se convierte en una operación computacional de alta complejidad [65].

Para calcular el gradiente de la función de pérdida respecto a cada peso de la red neuronal se usa un proceso llamado *Backpropagation.* De nuevo, se presenta un

caso de estudio simple, una red neuronal formada por una única neurona en la capa oculta, lo que implica dos conexiones. En la figura 2.22 se muestra la arquitectura de la red neuronal propuesta:


**Figure 2.22.:** Red neuronal simple de dos conexiones

Aplicando la regla de la cadena es posible calcular el gradiente $\frac{\partial J(W)}{\partial W}$:

$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

El algoritmo de *Backpropagation* consiste en repetir este proceso para cada peso de la red neuronal, propagando el gradiente, desde la capa superior de la red (donde se produce la predicción) hacia capas anteriores [64]. La propagación del gradiente requiere gran cantidad de recursos computacionales, y éstos aumentan con la complejidad de la tipología de la red neuronal, es por este motivo que el entrenamiento del modelo suele ser un proceso lento y costoso en términos computacionales [66].

Una vez hallado el gradiente de la función de pérdida, queda definido cómo afectan a la función de pérdida pequeños cambios en cada peso de la red neuronal, siendo posible actualizar el valor de cada peso según se muestra en el paso 2b del algoritmo de aprendizaje.

$$W_t \leftarrow W_{t-1} - \eta \frac{\partial J(W)}{\partial W}$$

En cada iteración del bucle, la configuración de pesos actual se modifica en sentido opuesto al gradiente. El parámetro $\eta$, conocido como *learning rate* cuantifica al ancho de paso en cada iteración. La configuración del parámetro de aprendizaje tiene un papel de vital importancia. Valores pequeños de *learning rate* convergen

lentamente y en falsos mínimos. Por otro lado, parámetros de aprendizaje demasiado grandes provocan inestabilidad y divergencia, pasando por alto los mínimos de la función [64].

Uno de los mayores temores en el uso de algoritmos de gradiente descendente es la convergencia en mínimos relativos no satisfactorios. En la práctica, las soluciones al problema de optimización suelen ser de la misma calidad, es decir, todos los mínimos de la función de pérdida son de la misma magnitud [65]. Siendo irrelevante, en cuál de ellos converger o desde qué condición inicial empezar.

### 2.4.5. Sobredimensionado u *overfiting*

El sobredimensionado es un problema fundamental y uno de los principales desafíos en modelos de ML. El objetivo principal del ML es construir modelos capaces de aprender las características o regularidades de un determinado conjunto de datos, permitiendo generalizar y estimar nuevos datos del conjunto. Por un lado, un modelo poco dimensionado no va a ser capaz de aprender los patrones y características del conjunto de datos, provocando estimaciones pobres o poco precisas. En el otro extremo, sobredimensionar un modelo permite capturar todas las características del conjunto de datos, lo que conlleva una gran pérdida de generalización en la estimación de nuevos datos del conjunto [64].

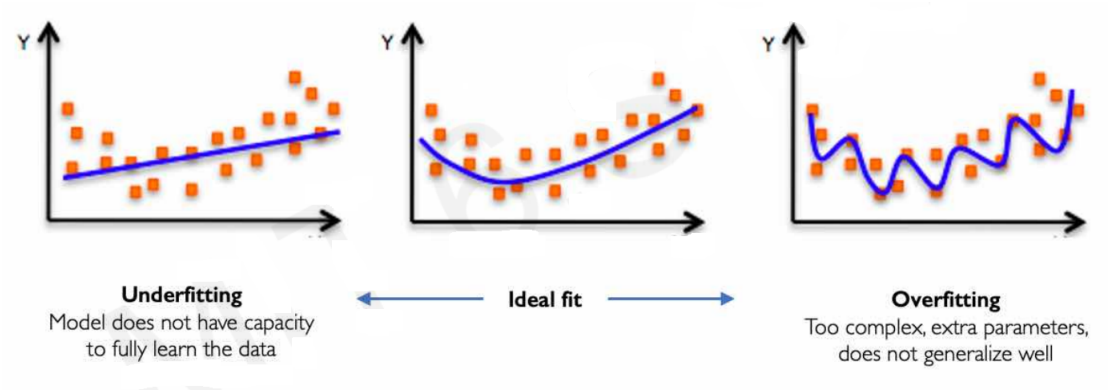


**Figure 2.23.:** Representación gráfica de un conjunto de datos (puntos rojos) y de su modelo de ML (curva azul) [64]

Idealmente, durante el desarrollo de un modelo de ML se desea encontrar un punto intermedio capaz de mantener la habilidad de generalizar y a su vez, capturar la mayoría de características del conjunto. Para resolver este problema se recurre al uso de técnicas de regularización. Con estas técnicas es posible extraer el mayor número de características existentes en el conjunto de datos de entrenamiento sin perder habilidad para generalizar en el conjunto de datos de test. Una de las técnicas de regularización más utilizadas se conoce como Parada Temprana o *Early Stopping*,

y su objetivo es terminar el entrenamiento justo antes de entrar en la etapa de sobredimensionado.

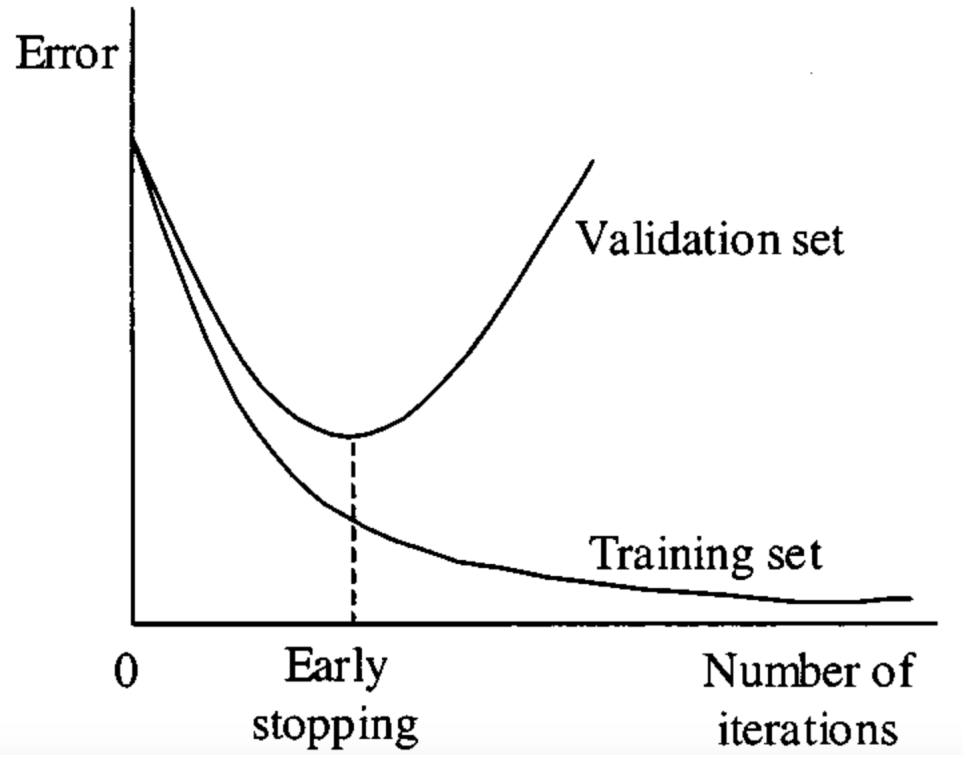


**Figure 2.24.:** Técnica de regularización *Early Stopping* [64]

Para ello, el conjunto de datos de entrenamiento se separa en dos subconjuntos, uno para entrenar el modelo propiamente dicho y otro para validar la habilidad de generalización. Como puede apreciarse en la figura 2.24, al inicio del entrenamiento, optimización y generalización están correlacionadas, a medida que la «pérdida» en el conjunto de entrenamiento se reduce, la «pérdida» en el conjunto de test también. En esta fase del entrenamiento se considera que el modelo está sub-dimensionado, sigue habiendo progreso que realizar [67]. Después de algunas iteraciones en el entrenamiento, se observa una divergencia entre las curvas de error de entrenamiento y validación, indicando el punto en que el modelo empieza a extraer demasiadas características del conjunto de datos de entrenamiento, provocando la pérdida de generalización en nuevas muestras (conjunto de datos de validación) [64].

### 2.4.6. Matriz de confusión

Una matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Básicamente, es una tabla que contabiliza las distintas combinaciones de valores reales y predichos [68]. De manera general, en

un problema de clasificación simple (positivo y negativo), en la matriz de confusión se encuentran cuantificadas las siguientes combinaciones:

- Verdaderos positivos (VP): Predicciones positivas que realmente lo son.
- Verdaderos negativos (VN): Predicciones negativas que realmente son.
- Falsos positivos (FP): Predicciones positivas que realmente no lo son.
- Falsos negativos (FN): Predicciones negativas que realmente no lo son.

La matriz de confusión permite evaluar de manera simple y eficiente algunas medidas de rendimiento del modelo [69]. Algunas de las más comunes son:

- **Precisión:** Esta medida muestra la precisión general del modelo. Es la fracción del total de muestras clasificadas correctamente.

  $$precisión = \frac{VP + VN}{VP + VN + FP + FN}$$

- **Exactitud:** Muestra la fracción de predicciones positivas clasificadas correctamente.

  $$exactitud = \frac{VP}{VP + FP}$$

- **Recall:** También llamado *True Positive Rate*, muestra la fracción de muestras positivas clasificadas correctamente.

  $$recall = \frac{VP}{VP + FN}$$

## 2.5. Redes neuronales convolucionales

### 2.5.1. Introducción

Las redes neuronales convolucionales han ganado recientemente un papel crucial en el campo de la visión por ordenador en grandes conjuntos de imagen y video [70, 71]. Este auge ha sido posible gracias a los grandes repositorios públicos de datos, como ImageNet [70] y al desarrollo de dispositivos de alta capacidad computacional [72].

Las redes neuronales convolucionales son modelos diseñados para trabajar con datos estructurados en forma de matriz o de múltiples vectores, datos con fuerte dependencia de su posición espacial. Esto las convierte en modelos extremadamente útiles para trabajar con imágenes digitales. Este tipo de datos exhiben dependencia espacial, porque es común encontrar regiones adyacentes con las mismas propiedades [59].

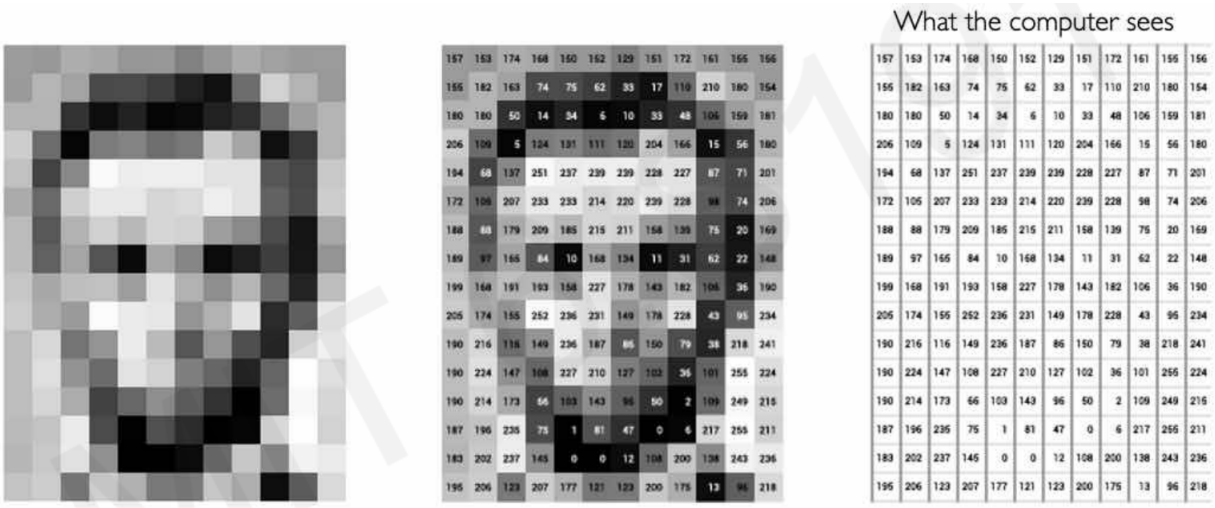


**Figure 2.25.:** A la izquierda una imagen en blanco y negro, en el medio la intensidad de cada píxel en el rango [0, 255] y a la derecha, la matriz equivalente que percibe un computador [73]

En el caso de representar la imagen en escala de grises, cada píxel almacena la intensidad equivalente comprendida entre [0, 255], como puede verse en la figura 2.25. Una imagen en color, almacena en cada píxel tres intensidades, una para cada escala de color RGB (Rojo, Verde y Azul) [65].

Para resolver problemas de clasificación de imágenes, el modelo debe basar su elección en los conjuntos de características encontrados en la imagen. Similar al modo en que el ser humano es capaz de identificar una cara si percibe una nariz, unos ojos y una boca, o detectar una casa siempre que percibe puertas, ventanas o escaleras [73].

La tipología de las redes neuronales convolucionales sigue las mismas directrices, puesto que uno de los objetivos es conservar la distribución espacial del conjunto de datos, la matriz de entrada se subdivide en conjuntos llamados parches, y cada uno de ellos alimenta una única neurona de la siguiente capa. De este modo se consigue que cada neurona de la red neuronal perciba una pequeña región del conjunto de datos de entrada. Esta arquitectura resulta útil en el procesado de imágenes, porque píxeles adyacentes suelen compartir información entre ellos, hecho que facilita la extracción de características de la imagen [73].

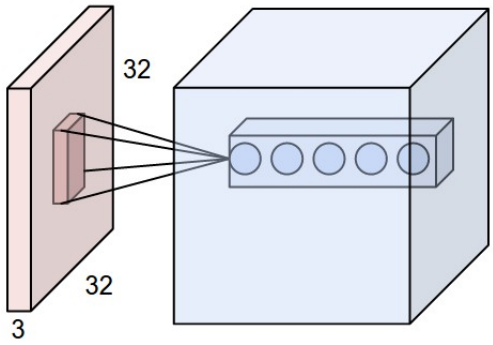


**Figure 2.26.:** Representación gráfica de una única conexión en una red neuronal convolucional

Así pues, cada píxel contenido en la región del parche, se multiplica por un peso y el resultado se computa a través de la neurona (ver sección 2.4.2). Al conjunto de pesos configurado en cada parche se le llama filtro. La idea es que cada filtro identifique un patrón espacial particular en una pequeña región de la imagen, por lo tanto, un gran número de filtros es necesario para capturar la gran variedad de formas presentes en una imagen [59].

De este modo, una red neuronal convolucional es una secuencia de capas. Cada una de ellas transforma el volumen de datos de entrada a través de una función. Se utilizan tres tipos de capas en este tipo de arquitectura: capas convolucionales, capas *Pooling* y capas de neuronas completamente conectadas (ver figura 2.19) [74]:

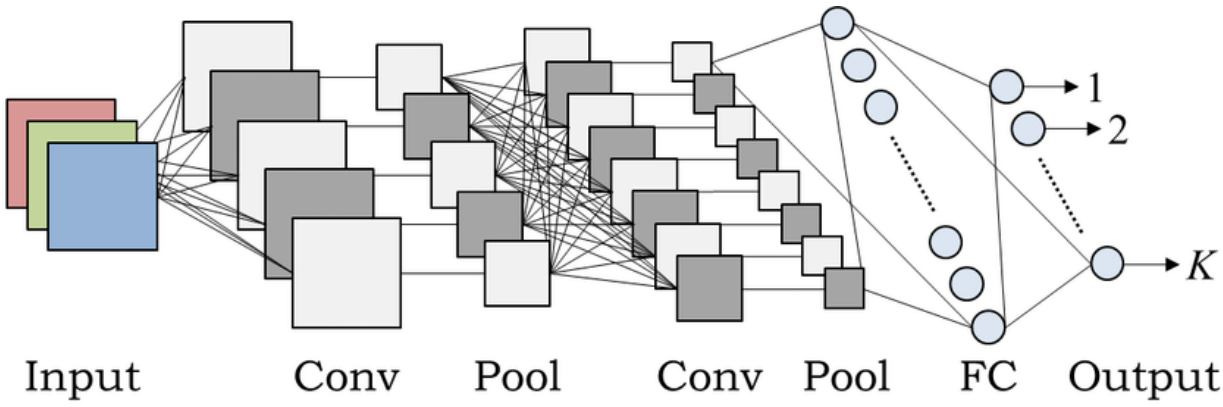


**Figure 2.27.:** Arquitectura general de una red neuronal convolucional [73]

- **Capa convolucional:** Las capas convolucionales computan la salida de las neuronas conectadas a cada región del volumen de entrada. Cada una calcula el producto escalar entre la matriz de datos de entrada y la matriz de pesos del filtro.

- **ReLU:** Como se ha explicado en la sección 2.4.2, la salida de cada neurona se mapea mediante una función de activación que aplica una no linealidad. Es común utilizar la función de activación ReLU debido a su rapidez y eficiencia computacional [73, 74].

- **Capa *Pooling*:** Estas capas permiten reducir el tamaño del volumen de entrada, uniendo o agregando datos de la matriz de datos. Cada vez que el volumen se reduce, la siguiente capa de filtros es capaz de abarcar una región más grande de características.

- **Red neuronal clásica:** Por último se utiliza una red neuronal completamente conectada para predecir la clase del vector de entrada según se ha detallado en secciones anteriores.

En las redes neuronales convolucionales, únicamente las capas de neuronas clásicas y convolucionales computan funciones dependientes únicamente de sus parámetros o pesos, por lo tanto, el algoritmo de entrenamiento se aplica en dichas capas. Las capas *Pooling* implementan funciones fijas, por lo que no implican un aumento de parámetros del modelo [74].

A continuación, se describe de forma individual cada capa de una red neuronal convolucional.

### 2.5.2. Capa convolucional

Este tipo de capas consiste en un conjunto de filtros de aprendizaje, cada filtro abarca una región del volumen de entrada, se desliza (convuelve) a través del ancho y largo de la matriz y se computa el producto escalar entre la matriz abarcada y los parámetros (pesos) del filtro. A medida que el filtro desliza a lo largo de la matriz, se produce un mapa de activación bidimensional que corresponde a la respuesta del filtro en cada posición del espacio. De este modo, la red neuronal va a aprender los patrones hallados en los filtros que se activen a lo largo de la imagen. En las primeras capas se detectarán patrones como ejes, orientaciones, manchas, y patrones más elaborados en capas de más alto nivel [74].

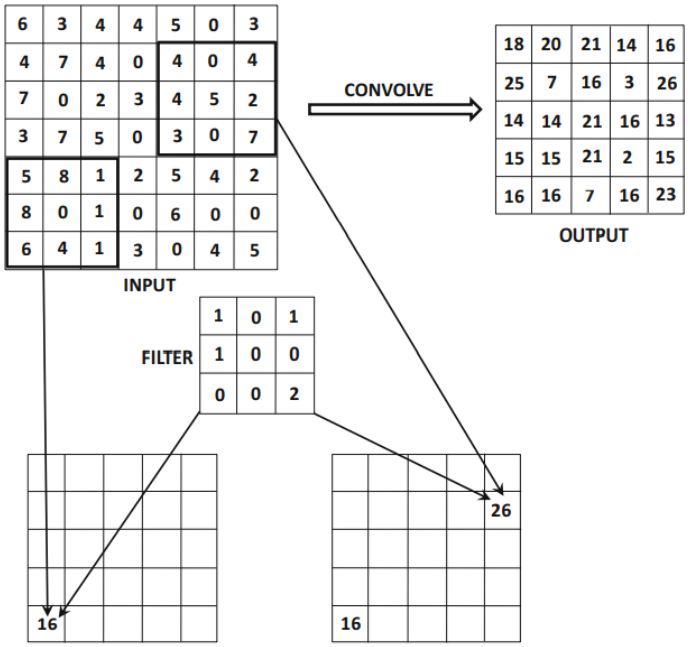


**Figure 2.28.:** Ejemplo de una convolución entre una entrada de dimensión $7 \times 7 \times 1$ y un filtro de dimensión $3 \times 3 \times 1$ con un paso de 1 [59]

Como se puede ver en la figura 2.28 la operación de convolución consiste en aplicar un simple producto escalar entre la matriz de entrada abarcada por el filtro y los parámetros del propio filtro, y repetirlo a través de todas las posiciones válidas del volumen. La operación convolucional se presenta formalizada a continuación:

Siendo $p$ el número de filtros de la capa $q$, los parámetros que lo conforman se pueden denotar con el tensor $W^{(p,q)} = [w_{ijk}^{(p,q)}]$. Los índices $i, j, k$ indican la posición a través de la altura, anchura y profundidad del filtro. Y siendo el volumen de datos de entrada representado por el tensor $X^{(q)} = [x_{ijk}^{(q)}]$, el mapa de características generado se describe de la siguiente manera:

$$x_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} x_{i+r,j+s,k}^{(q)} \tag{2.13}$$

Donde $F_q$ corresponde al ancho y alto de la capa, y $d_q$ corresponde a la profundidad de la misma [59].

En el diseño de capas convolucionales hay que tener en cuenta tres hiperparámetros, la profundidad de la capa, el paso y el parámetro llamado *zero-padding*. En primer lugar, la profundidad del volumen de salida corresponde al número de filtros utilizados en la capa, cada uno de ellos busca un patrón distinto. También es necesario especificar el número de píxeles que se va a desplazar el filtro en cada paso, el

parámetro que representa esta cantidad se conoce como *stride*. Y por último, para controlar las dimensiones de la matriz de salida, es común añadir filas y columnas de ceros en los bordes de la matriz, siendo el parámetro *padding* equivalente al número de filas o columnas añadidas en cada lado [74].

### 2.5.3. Capa *Pooling*

Es común insertar periódicamente capas *Pool* entre capas sucesivas de convolución. Su función es reducir el tamaño del volumen de entrada, de modo que la cantidad de parámetros de la red, también se reduce. A su vez, también se utiliza para controlar el sobredimensionado del modelo [74]. La operación *Pool* trabaja en pequeñas regiones de la matriz correspondiente a cada capa del volumen de entrada, y produce un volumen con la misma profundidad, pero de superficie (alto y ancho) reducida. Por cada región en la capa de entrada de tamaño $P_q \times P_q$, el máximo de estos valores se retorna. Esta operación se conoce como *max-pooling*. Al reducir el tamaño de la superficie de entrada, se aumenta la cantidad de región abarcada por la siguiente capa de filtros, permitiendo el aprendizaje de patrones más complejos [59].

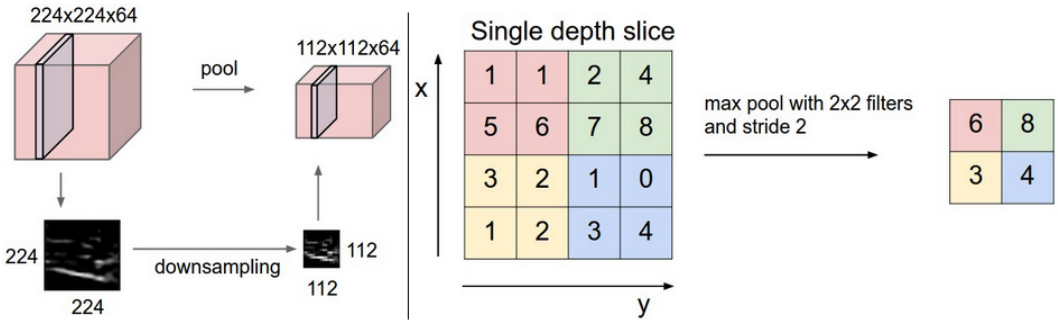


**Figure 2.29.:** Representación gráfica y ejemplo de una operación *Pool* [74]

La capa *Pool* reduce el tamaño de cada capa del volumen de entrada. A la izquierda de la figura 2.29, se muestra un ejemplo de una operación *Pool* realizada sobre un volumen de entrada de dimensión $[224 \times 224 \times 64]$, con un filtro de tamaño 2 y un paso de 2. El volumen de salida correspondiente tiene dimensiones $[112 \times 112 \times 64]$. A la derecha de la figura, se muestra un ejemplo del uso de la operación *max-pooling*, de modo que se toma el valor máximo de la región abarcada por el filtro.

### 2.5.4. Arquitecturas

A lo largo del desarrollo de las redes neuronales convolucionales aplicadas a resolver problemas de visión por computador, algunas arquitecturas han mostrado excelentes resultado y forman parte del estado del arte actual en este campo:

**LeNet** El primer éxito de estos modelos se desarrolló en los años 90 por Yann LeCun, consistió en una aplicación para leer códigos numéricos y dígitos [74].

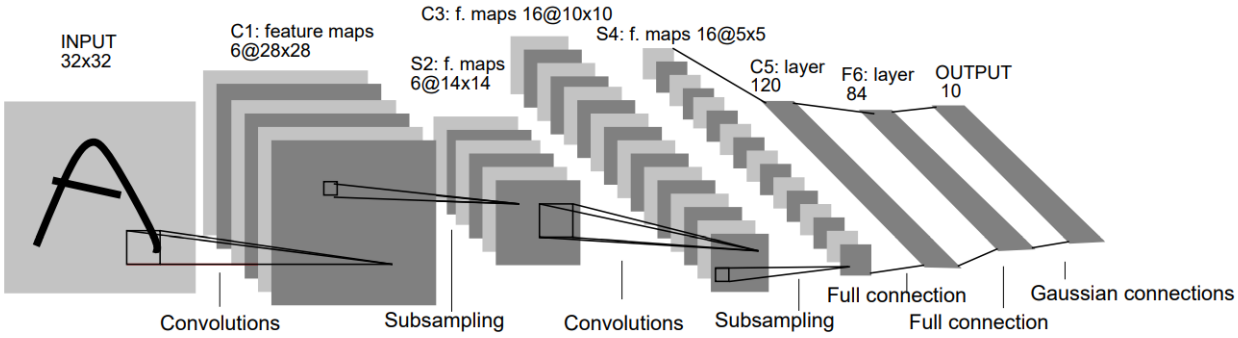


**Figure 2.30.:** Arquitectura LeNet [75]

En la figura 2.30 se puede apreciar la arquitectura de LeNet. Las capas denotadas por $C_i$ corresponden a capas convolucionales. Las capas denotadas por $S_i$ corresponden a capas *Pool*. Los filtros utilizados en las capas convolucionales son de dimensión $5 \times 5$ y los utilizados en las capas *Pool*, de dimensión $2 \times 2$ [75].

**AlexNet** La red AlexNet ganadora del desafío ImageNet ILSVRC en 2012, tiene una arquitectura muy parecida a LeNet, es más grande y más profunda [74]. Esta arquitectura utiliza dos unidades de cálculo para analizar paralelamente la mitad superior e inferior de la imagen. A continuación, se muestra la tipología de la red AlexNet sin partición:

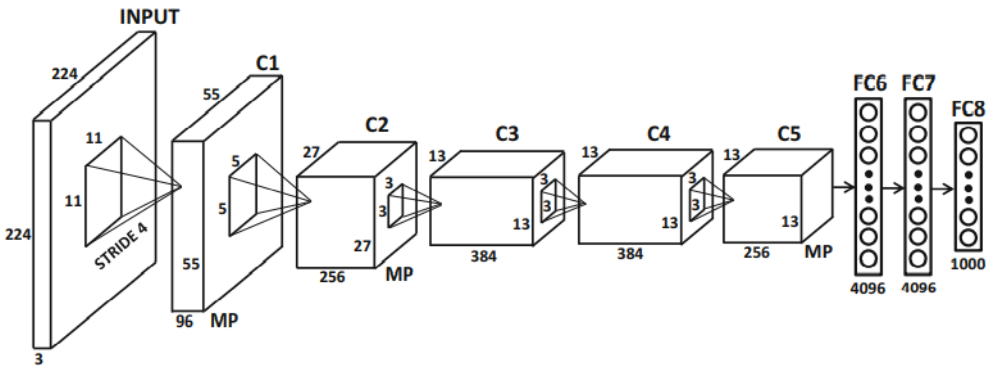


**Figure 2.31.:** Arquitectura Alexnet sin partición computacional [59]

En esta tipología, después de cada convolución, la función de activación ReLU (ver sección 2.4.2) se aplica [76], seguido de una capa *max-pooling*, denotada como MP

en la figura 2.31. La elección de la función de activación ReLU, sirvió de precedente para las arquitecturas desarrolladas posteriormente [59].

**ZFNet** Este modelo ganó el desafío ILSVRC en 2013. Consiste en una mejora de la red Alexnet, ajustando los hiperparámetros. Específicamente, este modelo expande las capas convolucionales y reduce el tamaño de los filtros que se usan en AlexNet [74].

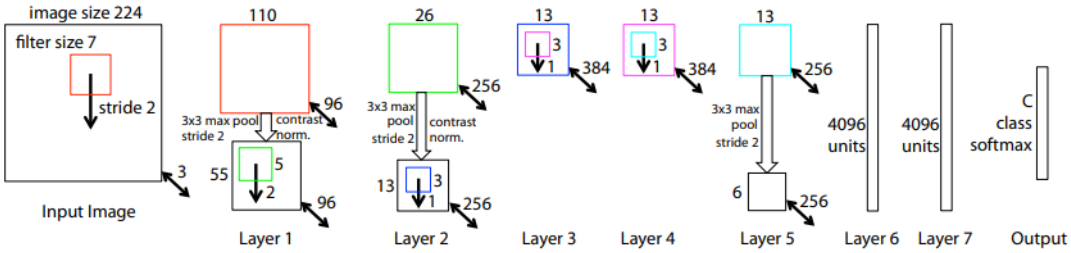


**Figure 2.32.:** Arquitectura del modelo ZFnet[77]

La arquitectura del modelo toma como entrada imágenes de dimensión $224 \times 224 \times 3$. En la primera capa convolucional, se aplican 96 filtros de dimensión $7 \times 7$. El mapa de características resultante, pasa a través de la función de activación ReLU y se le aplica la operación *max-pooling* (mediante regiones de dimensión $3 \times 3$). Esta secuencia se repite hasta 5 veces, únicamente modificando la cantidad de filtros utilizados en cada capa convolucional. Las últimas capas están formadas por neuronas completamente conectadas, como puede verse en la figura 2.32.

**GoogLeNet** El modelo ganador del desafío ILSVRC en 2014 fue GoogLeNet. Este modelo se caracteriza por el uso de sub-estructuras llamadas *Inception modules* (figura 2.33). Además, en lo alto de la red, se utilizan capas *Pool*, en cambio de únicamente capas de neuronas completamente conectadas.

El uso de estos módulos en capas iniciales de la red permite realizar un entrenamiento más eficiente. Al reducir las dimensiones del volumen de entrada es posible añadir más parámetros a la red sin incrementar la complejidad computacional del modelo [78]. La arquitectura general de la red se muestra en la figura 2.1.

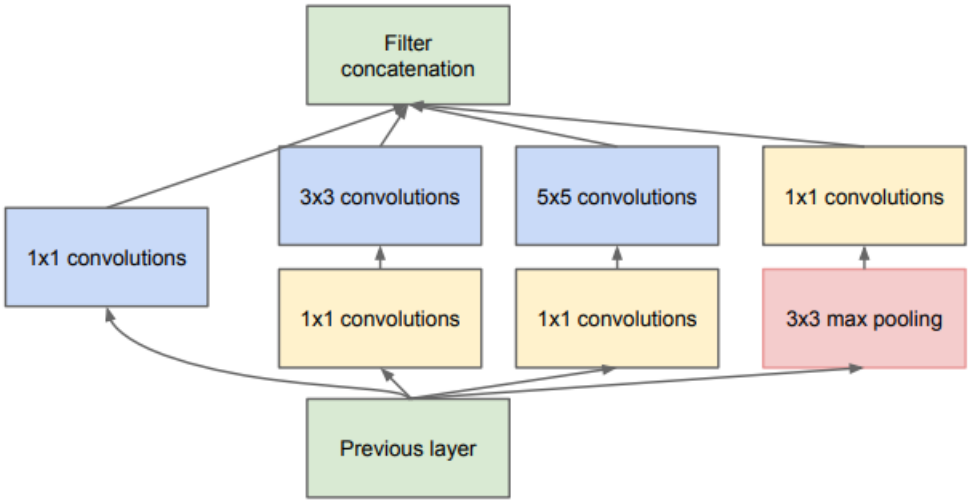


**Figure 2.33.:** Arquitectura del *Incepcion module* utilizado en la red GoogLeNet [78]

| Tipo de capa | Volumen de salida |
|---|---|
| convolución | $112 \times 112 \times 64$ |
| max-pool | $56 \times 56 \times 64$ |
| convolución | $56 \times 56 \times 192$ |
| max-pool | $56 \times 56 \times 192$ |
| inception | $28 \times 28 \times 256$ |
| inception | $28 \times 28 \times 480$ |
| max-pool | $14 \times 14 \times 480$ |
| inception | $14 \times 14 \times 512$ |
| inception | $14 \times 14 \times 512$ |
| inception | $14 \times 14 \times 512$ |
| inception | $14 \times 14 \times 528$ |
| inception | $14 \times 14 \times 832$ |
| max-pool | $7 \times 7 \times 832$ |
| inception | $7 \times 7 \times 832$ |
| inception | $7 \times 7 \times 1024$ |
| avg-pool | $1 \times 1 \times 1024$ |
| dropout(40%) | $1 \times 1 \times 1024$ |
| linear | $1 \times 1 \times 1000$ |
| softmax | $1 \times 1 \times 1000$ |

**Table 2.1.:** Arquitectura del modelo GoogLeNet [78]

**VGGNet** Este modelo fue segundo ganador de la competición ILSVRC en 2014. Sus desarrolladores demostraron que la profundidad de la red es un elemento crítico a tener en cuenta para obtener un buen funcionamiento [74]. Existen varias configuraciones de la tipología VGG, la configuración usada en el desafío fue de 16 capas. A continuación, se presentan las distintas configuraciones de VGGNet:

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

**Figure 2.34.:** Arquitectura de las distintas configuraciones de VGGNet [79]

El volumen de entrada a este modelo es de dimensión $224 \times 224 \times 3$. Cada imagen alimenta un conjunto de capas convolucionales secuenciales, los filtros utilizados son de dimensión $3 \times 3$ (el mínimo tamaño necesario para capturar patrones básicos [79]). El paso del filtro esta fijado a 1 píxel. No todas las capas convolucionales están seguidas de una capa *Pool* (que computa la función *Maxpooling* utilizando ventanas de $2 \times 2$) [79].

### 2.5.5. Auge de las CNN

En las secciones anteriores se han descrito y detallado los principios fundamentales de las redes neuronales convolucionales . De forma general, estos modelos identifican patrones de características existentes en un conjunto de datos de entrada, de manera que en cada capa de la red neuronal se analiza un patrón de más alto nivel que en la capa anterior.

El entrenamiento de estos modelos, consiste pues, en determinar qué patrones o filtros se analizan en cada capa. En la práctica, llevar a cabo el entrenamiento de todas las capas de una red neuronal convolucional requiere de gran cantidad de tiempo, energía y recursos computacionales [80], siendo el inconveniente más significativo de esta tecnología.

El interés en las redes neuronales aumentó drásticamente entorno al año 2006, cuando un grupo de investigadores del CIFAR (*Canadian Institute for Advanced Research*), realizó un pre-entrenamiento en sus modelos. Esta metodología consiste en entrenar las capas iniciales del modelo, de manera que los filtros y patrones más complejos de la red quedan configurados. De este modo, el usuario únicamente debe configurar las últimas capas de la red. El entrenamiento de las últimas capas es mucho más simple a nivel computacional, además de ser en estas capas donde se determina la aplicación final del modelo [65].

## 2.6. Máquinas de estado finito

Las máquinas de estado finito (FSM) son dispositivos secuenciales donde la salida depende del historial de operaciones realizadas anteriormente [81, 82]. Una FSM básicamente está compuesta de estados y transiciones [83].

El modelo matemático puede definirse con la matriz: $\{X, Y, S, \delta, \lambda\}$, donde: $X$ corresponde al vector finito de entradas, $Y$ el vector finito de salida, $S$ es el conjunto de estados, $\delta$ es la función de transición, y $\lambda$ es la función de salida [84].

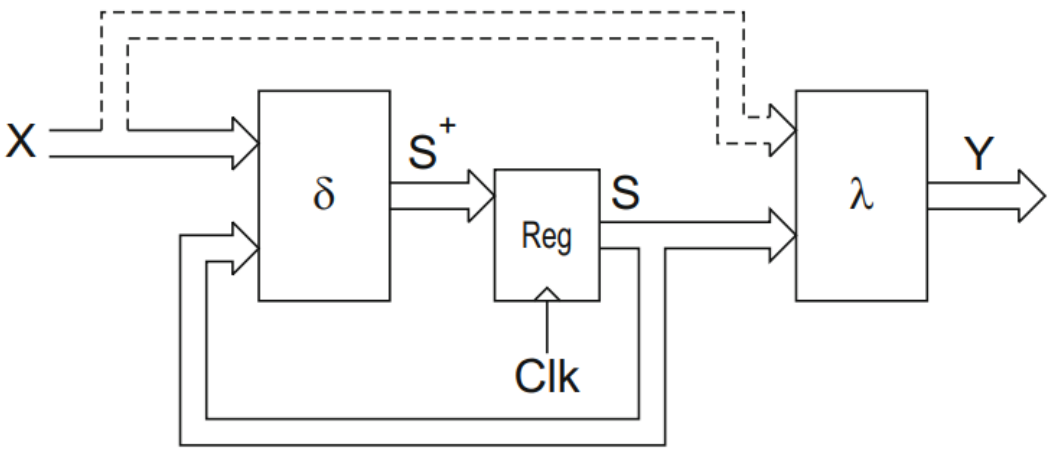


**Figure 2.35.:** Estructura de una máquina de estados [84]

Siendo $X$ e $Y$, conjuntos de datos registrados según el instante de tiempo definido por cada pulso de reloj, cada salida $y_n \epsilon Y$ se representa como [85]:

$$y_n(t) = \lambda_n(X(0), X(1), ..., X(t-1), X(t)) \tag{2.14}$$

En la práctica, es necesario simplificar la ecuación 2.14. De modo que se utilizan los estados internos $S$, para representar el historial de operaciones de la máquina de estado [86].

Así pues, como norma general la salida de una máquina de estados depende el conjunto de datos de entrada y del estado actual del sistema:

$$Y = \lambda(S, X) \tag{2.15}$$

Los modelos de máquinas de estados basados en la ecuación 2.15 y en la estructura de la figura 2.35, se conocen como máquinas de Mealy. Por otro lado, si la salida de la máquina de estado únicamente depende del estado actual del sistema $Y = \lambda(S)$, se le llama máquina de Moore [85].

Para representar máquinas de estados es común utilizar diagramas de estados, en ellos, los estados se denotan por círculos. Las transiciones se representan con flechas, que indican el sentido del cambio de estado. Las transiciones se etiquetan con la condición de activación y la acción correspondiente. En la figura 2.36 se muestra un diagrama simple de una máquina de estados:

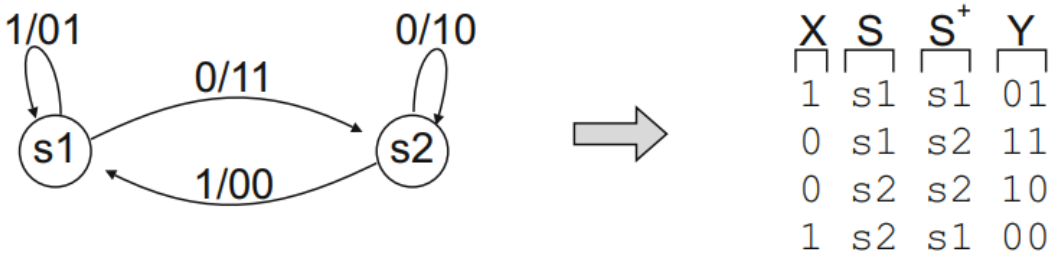


**Figure 2.36.:** Diagrama de una máquina de estados simple [84]

El diagrama mostrado representa una FSM de dos estados y cuatro transiciones. La etiqueta de cada transición indica el valor de las entradas que provoca la transición (desde el estado origen al estado apuntado por la flecha).

En una máquina de Mealy, los valores de salida también se muestran en las flechas (puesto que la salida depende tanto del estado como de las entradas). En una máquina de Moore, la salida se representa en el interior de cada estado (puesto que la salida del sistema depende únicamente del estado actual) [87].

Las máquinas de estado finito se utilizan en gran variedad de sistemas del mundo real, desde simples aplicaciones de calculadora hasta en el diseño de complejas piezas de software. El uso de máquinas de estado es una herramienta muy útil en el desarrollo de controles para sistemas automáticos como lavadoras, drones y todo tipo de autómatas. El flujo de control es simple, basta con identificar los estados del sistema y las acciones de control a llevar a cabo en cada uno de ellos [88]. Convirtiéndolos en sistemas sencillos de implementar, extender (añadiendo más estados o transiciones) y de reutilizar.

## 2.7. Estrategias de movimiento

### 2.7.1. Introducción

Esta sección se describe brevemente algunas de las técnicas utilizadas en la navegación y planificación de movimientos en robótica móvil. La búsqueda de objetos es una tarea compleja para un robot, resolver este tipo de tareas conlleva moverse y explorar el ambiente, para ello es necesario seguir una planificación o estrategia de movimiento, de lo contrario, pueden producirse resultados infructuosos, así como derroche de tiempo y energía [89]. Cómo un robot decide qué movimientos ejecutar al interactuar con el medio externo, es uno de los desafíos que resuelve la planificación de movimientos [90].

### 2.7.2. Navegación basada en geometría

La navegación en la robótica móvil tradicionalmente se ha considerado un problema geométrico, donde el objetivo del robot es percibir la geometría del entorno y planear un camino libre de colisiones hasta la meta deseada. El problema relacionado con el mapeo del entorno y la planificación de movimientos, se conoce como SLAM (*Simultaneous Localisation and Map Builidng*) [91]. Abordar esta problemática desde un punto de vista puramente geométrico puede ser insuficiente en varios escenarios de navegación. Aunque este tipo de técnicas ha producido excelentes resultados en entornos controlados, la navegación en ambientes reales presenta desafíos difícilmente abordables desde un punto de vista puramente geométrico [92].

### 2.7.3. Navegación basada en aprendizaje

El auge de los algoritmos de Deep Learning y las técnicas de aprendizaje reforzado han fomentado la navegación basada en aprendizaje [93]. Este tipo de estrategias permiten aprender de la experiencia e interpretar el entorno actual para planificar el movimiento del robot [94]. Además, este tipo de técnicas no necesita una resolución en dos pasos (mapeo y planificación), sino que se lleva a cabo una resolución directa, a partir de un dato de entrada se determina una acción de control [95]. El estado del arte actual presenta excelentes resultados en este campo: En el artículo *Cad2rl: Real single-image flight without a single real image* [96], se entrena una red neuronal mediante aprendizaje reforzado para la navegación autónoma de un dron. Basando el aprendizaje en la evasión de obstáculos, el dron es capaz de desplazarse de manera autónoma en entornos de interior. En *Deepnav: Learning to navigate large cities* [97], se desarrolla un algoritmo de navegación en ciudades basado en redes neuronales convolucionales. El algoritmo que presentan clasifica imágenes capturadas en las intersecciones de las calles para determinar cuál es la dirección óptima para llegar al destino.

# 3. Diseño y desarrollo

## 3.1. Perspectiva general

En este capítulo, se presenta el diseño y desarrollo de los distintos algoritmos de control, considerando los componentes y tecnologías seleccionadas en capítulos anteriores. Como se ha explicado en la sección 1.1, el objetivo del proyecto es diseñar y desarrollar un algoritmo de control capaz de conducir al robot Bittle hasta una pelota de tenis. La idea general consiste en utilizar un algoritmo de Deep Learning para determinar la posición relativa de la pelota de manera discreta, es decir, si el objetivo se encuentra a la izquierda, a la derecha o delante del robot. Esta información sirve de entrada para una máquina de estados, que determina la acción de control correspondiente.

Este capítulo empieza describiendo el montaje del sistema y la interconexión de sus elementos. A continuación, en las secciones 3.4, 3.5, 3.6 y 3.7, la problemática propuesta se ha desarrollado en distintos pasos. Partiendo de un desafío simple, en que únicamente se desea conseguir que el robot se oriente al objeto seleccionado, hasta desarrollar e implementar una estrategia de movimiento de búsqueda y captura.

Aspectos relacionados con la configuración del sistema se encuentran detallados en el apéndice A. Los códigos utilizados durante el entrenamiento de la red neuronal, así como los algoritmos de control se muestran en los apéndices B, C, D.

## 3.2. Montaje

El sistema propuesto para el desarrollo está compuesto de tres elementos:

- Estructura del robot Bittle
- Raspberry Pi
- Sensor cámara

La estructura del robot seleccionado se ha obtenido en su versión ensamblada, donde todos los componentes (cuerpo, servomotores, placa de control y fuente de alimentación) ya se encuentran montados. Así pues, el montaje del sistema consiste en la conexión del sensor cámara a la unidad de control (Raspberry Pi) y la conexión, a través del puerto serie, de la unidad de control y la placa base del robot.

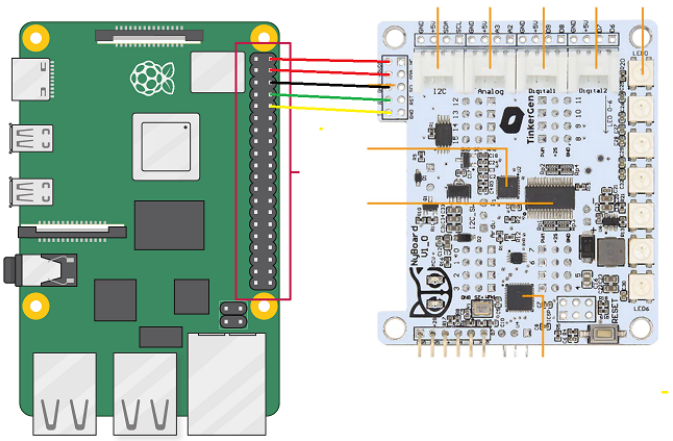

**Figure 3.1.:** Conexionado de la Raspberry Pi con la placa de control Nyboard

La conexión de la cámara a la unidad de control se ha realizado mediante un cable plano *display* de 15 pines. Basta con conectar correctamente el cable plano en su conector.

El conexionado de la Raspberry Pi y la placa de control del robot se realiza a través del puerto serie. La placa de control NyBoard (ver sección 2.1.1) está diseñada para encajar con los pines correspondientes de la Raspberry Pi. El conexionado se muestra en la figura 3.1.

Los pines 2, 4 y 6 de la Raspberry Pi (ver figura A.1) se conectan a la fuente de alimentación de 5V del robot. Los pines 8 y 10 (transmisor y receptor) de la Raspberry Pi se conectan al puerto serie de la placa NyBoard como muestran las líneas verde y amarillo de la figura 3.1.

**Figure 3.2.:** Montaje del sistema: sensor, unidad de control y cuerpo del robot

## 3.3. Funcionamiento general

Como se ha explicado en secciones anteriores, el desafío del proyecto se ha resuelto progresivamente en diferentes desarrollos, cada uno de ellos más complejo que el anterior. Teniendo como objetivo final, desarrollar un control de movimiento para que el robot Bittle consiga buscar y encontrar una pelota, se ha empezado diseñando e implementado un objetivo mucho más simple. De esta manera, en las primeras fases del desarrollo se resuelven los aspectos generales del control, que sirven como base para desarrollos posteriores y de más complejidad.

Aunque los desarrollos propuestos tienen distintos comportamientos y estructuras internas, el funcionamiento general del control es el mismo:


**Figure 3.3.:** Esquema general del algoritmo de control

Como puede apreciarse en la figura 3.3, el control del Bittle tiene una estructura simple. Después de capturar una imagen mediante el sensor cámara incorporado en la estructura del robot, se utiliza una red neuronal convolucional VGG-16 para clasificar la posición relativa del objetivo. Dicha clasificación sirve como dato de entrada a una máquina de estados finitos. Al determinar el siguiente estado del sistema, una acción de movimiento se lleva a cabo.

## 3.4. Desarrollo I

El objetivo del primer desarrollo consiste en crear un lazo de control simple para conseguir que el robot se oriente hacia la pelota. A continuación se describe de manera sencilla el algoritmo de control propuesto:

1. Capturar imagen
2. El modelo de Deep Learning clasifica la posición relativa de la pelota: derecha, izquierda o centro
3. Según el resultado, la máquina de estados realiza una transición y su correspondiente acción de control: giro a la derecha, giro a la izquierda o fin de la rutina (en caso de recibir la clasificación «centro»)
4. Regresar a paso 1 en caso de no obtener clase «centro»

### 3.4.1. Dataset

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 450 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 56 imágenes que no han alimentado el modelo durante el entrenamiento.

Es importante adaptar el conjunto de datos al objetivo del proyecto, en este primer desarrollo únicamente se pretende orientar el robot al objetivo desde una posición inicial. No se pretende conseguir ningún acercamiento al objetivo. Por lo tanto, las imágenes se han tomado todas desde la misma distancia inicial.

### 3.4.2. Entrenamiento

Para el entrenamiento del modelo se ha configurado una red neuronal convolucional VGGnet (figura 2.34), se ha utilizado la tipología de 16 capas para clasificar tres clases, derecha, izquierda y centro. Usando las librerías de Tensorflow y Keras [98] se ha construido el siguiente modelo VGG-16:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 512) | 0 |
| dense (Dense) | (None, 4096) | 2101248 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dense_2 (Dense) | (None, 1000) | 4097000 |
| dense_3 (Dense) | (None, 3) | 3003 |

Total params: 37,697,251
Trainable params: 22,982,563
Non-trainable params: 14,714,688

**Figure 3.4.:** Arquitectura del modelo construido

Como se muestra en la figura 3.4, la red neuronal VGG-16 debe ser alimentada con imágenes de dimensión 224x224. Así pues, las imágenes deben ser pre-procesadas antes de alimentar el modelo, tanto en la fase de entrenamiento, como en las fases posteriores de inferencia. El pre-procesado consiste en redimensionar cada imagen del conjunto de datos. Para ello se ha utilizado la función *resize* de la librería de código abierto OpenCV [99].

Puesto que el modelo debe clasificar la posición discreta del objetivo en tres categorías, la última capa del modelo debe contener 3 neuronas, cada una de ellas retorna la probabilidad de que el objetivo se encuentre en una determinada categoría.

Como se ha explicado en la sección 2.4, el entrenamiento de una red neuronal consiste en ajustar los pesos que la conforman. En la sección 2.5.5 se ha visto que el auge de estos modelos, en parte, es debido al uso de modelos pre-entrenados. Así pues, se utiliza la configuración de pesos de Imagenet [100] en las capas inferiores del

modelo. Las cuatro capas superiores (correspondientes a la red neuronal clásica) se han entrenado con el conjunto de datos generado manualmente.

El entrenamiento realizado es de cuatro épocas y se ha utilizado la función de pérdida logística (ecuación 2.11) a modo de señal de error.

En la figura 3.5, puede observarse como el fenómeno de sobredimensionado aparece en la cuarta época del entrenamiento. Una vez el modelo ha sido entrenado, es posible utilizar la red neuronal como una entidad que convierte la imagen de entrada en una clase numérica.

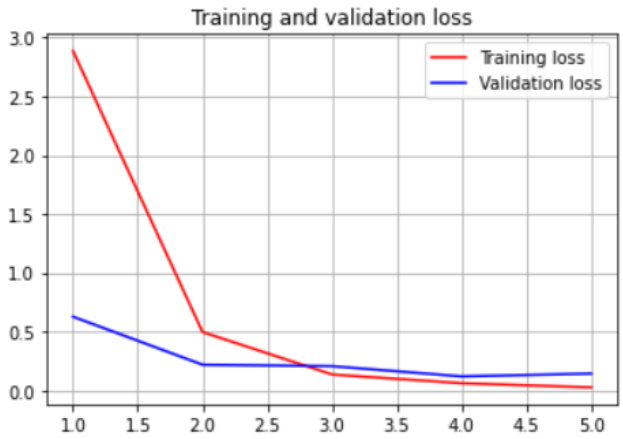


**Figure 3.5.:** Gráfico de la función de pérdida en el entrenamiento del modelo del desarrollo I

### 3.4.3. Máquina de estados

El control del robot se ha desarrollado sobre una máquina de estados finitos. Mediante la librería *StateMachine* [16] de Python es posible describir de manera simple los estados y transiciones de la máquina de estados. En la figura 3.6 se muestra el diagrama de estados implementado en este desarrollo:

Como puede apreciarse en la figura 3.6, se ha diseñado una máquina de Moore. La entrada es el resultado de la inferencia realizada por la red neuronal, y la salida es una respuesta motriz del robot. La máquina de estados diseñada está compuesta de los siguientes estados y transiciones:

- Estados:
  - **Inicio**: El estado inicial sirve como punto de partida del control, cada vez que se entra en dicho estado, el robot toma una postura inicial adecuada para capturar la primera imagen correctamente. Se ha elegido la posición de calibración (ver figura 2.8), para ello se envía por el puerto serie la cadena de caracteres «*kcalib*».

  - **Izquierda**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la izquierda. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrL*» y el tiempo de ejecución de dicho giro (configurado a 1 segundo para todas las instrucciones de giro del proyecto).
  - **Derecha**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la derecha. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrR*» y el tiempo de ejecución de dicho giro.
  - **Centro**: Corresponde al estado final del control, cada vez que se entra en dicho estado, el robot toma una postura para indicar el fin. Para ello se envía por el puerto serie la cadena de caracteres «*ksit*».

- Transiciones:
  - **Transición a derecha**: cuando la entrada a la máquina de estados sea una clase 2, el siguiente estado será «derecha».
  - **Transición a izquierda**: cuando la entrada a la máquina de estados sea una clase 1, el siguiente estado será «izquierda».
  - **Transición a centro**: cuando la entrada a la máquina de estados sea una clase 0, el siguiente estado será «centro».

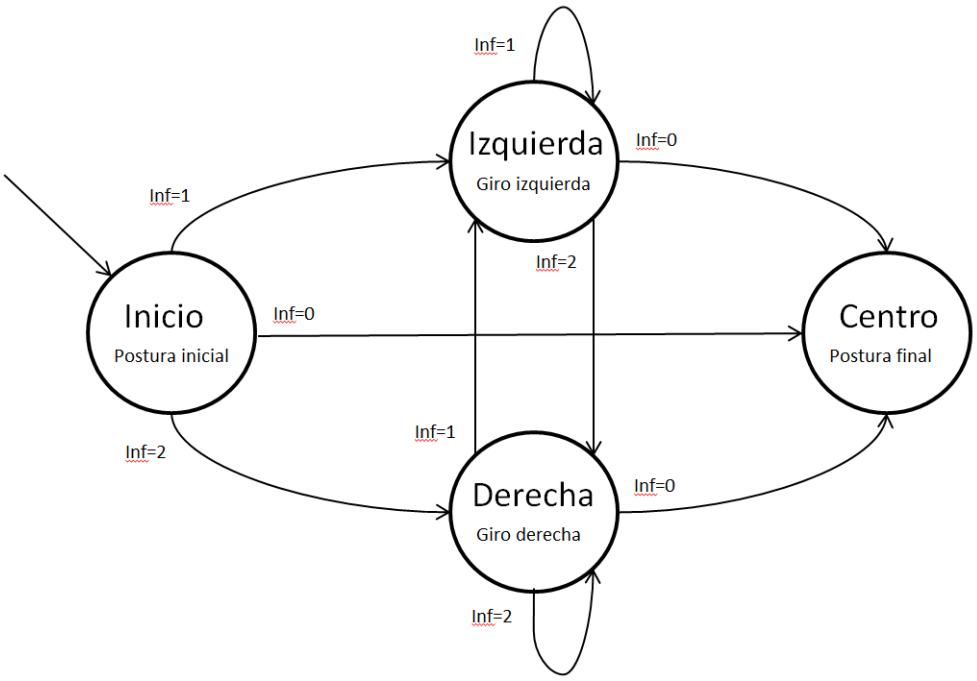


**Figure 3.6.:** Diagrama de estados propuesto para el primer desarrollo del proyecto

### 3.4.4. Algoritmo de control

Una vez desarrolladas las entidades del sistema, se ha creado un algoritmo de control siguiendo el esquema mostrado en la figura 3.7.

Una vez inicializada la máquina de estados, el lazo de control se activa. En cada iteración el algoritmo analiza si el estado actual equivale a «centro», en tal caso, el objetivo del control ha sido cumplido y se termina la rutina. En caso contrario, la red neuronal convolucional clasifica la imagen capturada para determinar la posición del objetivo. Esta información alimenta la máquina de estados, que decidirá cuál es el siguiente estado (ver figura 3.6) y se ejecutará la acción de control correspondiente.

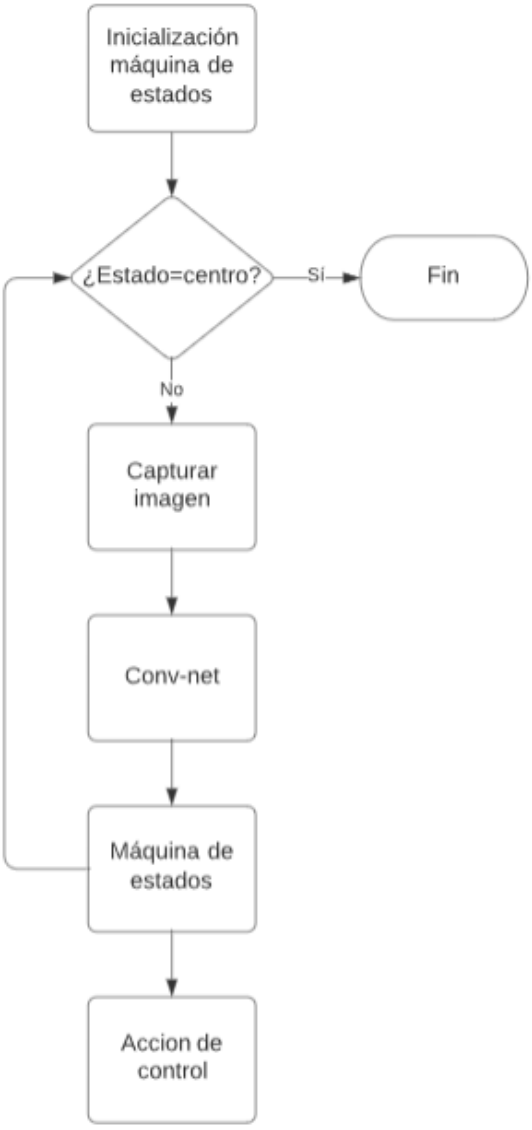


**Figure 3.7.:** Diagrama de flujo del algoritmo de control del desarrollo I

## 3.5. Desarrollo II

El objetivo del segundo desarrollo consiste en crear un lazo de control simple para conseguir que el robot busque y encuentre la pelota. Para ello, el modelo de Deep Learning deberá, a parte de discretizar la posición de la pelota (izquierda, derecha o centro), determinar la clase «destino» (objetivo alcanzado). A continuación se describe de manera sencilla el algoritmo de control propuesto:

1. Capturar imagen
2. El modelo de Deep Learning clasifica la posición relativa de la pelota: derecha, izquierda, centro o destino
3. Según el resultado, la máquina de estados realiza una transición y su correspondiente acción de control: giro a la derecha, giro a la izquierda, andar al frente o fin de la rutina (en caso de recibir la clasificación «destino»)
4. Regresar a paso 1 en caso de no obtener clase «destino»

### 3.5.1. Dataset

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 400 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 80 imágenes que no han alimentado el modelo durante el entrenamiento.

Es importante adaptar el conjunto de datos al objetivo del proyecto, en el segundo desarrollo el robot debe buscar y acercarse al objetivo, por lo tanto, el conjunto de datos de entrenamiento contiene imágenes tomadas desde distintas distancias del objetivo, en sus distintas posiciones.

### 3.5.2. Entrenamiento

Para el entrenamiento del modelo se ha configurado una red neuronal convolucional VGGnet (figura 2.34), se ha utilizado la tipología de 16 capas para clasificar tres clases, derecha, izquierda y centro. Usando las librerías de Tensorflow y Keras [98] se ha construido el modelo VGG-16:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 512) | 0 |
| dense (Dense) | (None, 4096) | 2101248 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dense_2 (Dense) | (None, 1000) | 4097000 |
| dense_3 (Dense) | (None, 4) | 4004 |

Total params: 37,698,252
Trainable params: 22,983,564
Non-trainable params: 14,714,688

**Figure 3.8.:** Arquitectura del modelo de Deep Learning del segundo desarrollo

Puesto que el modelo debe clasificar la posición discreta del objetivo en tres categorías, la última capa del modelo debe contener 4 neuronas, cada una de ellas retorna la probabilidad de que el objetivo se encuentre en una determinada categoría.

El entrenamiento realizado es de cuatro épocas y se ha utilizado la función de pérdida logística (ecuación 2.11) a modo de señal de error.

En la figura 3.9, puede observarse como el fenómeno de sobredimensionado aparece en la cuarta época del entrenamiento. Una vez el modelo ha sido entrenado, es posible utilizar la red neuronal como una entidad que convierte la imagen de entrada en una clase numérica, según muestra la figura 3.10.


**Figure 3.9.:** Representación gráfica de la función de pérdida en el entrenamiento del modelo del desarrollo II

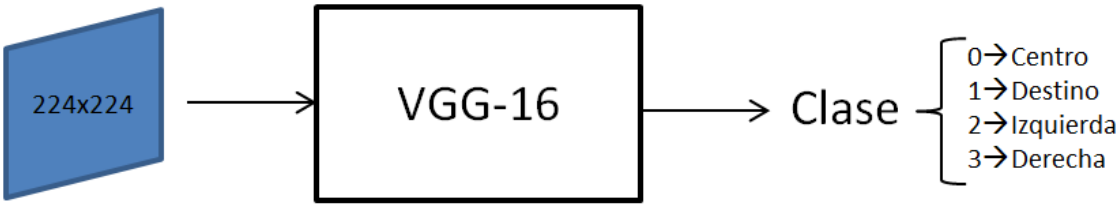


**Figure 3.10.:** Representación funcional del modelo VGG-16 entrenado en el desarrollo II

### 3.5.3. Máquina de estados

El control del robot se ha desarrollado sobre una máquina de estados finito, mediante la librería *StateMachine* [16] de Python es posible describir de manera simple los estados y transiciones de la máquina de estados. En la figura 3.6 se muestra el diagrama de estados implementado en este desarrollo:

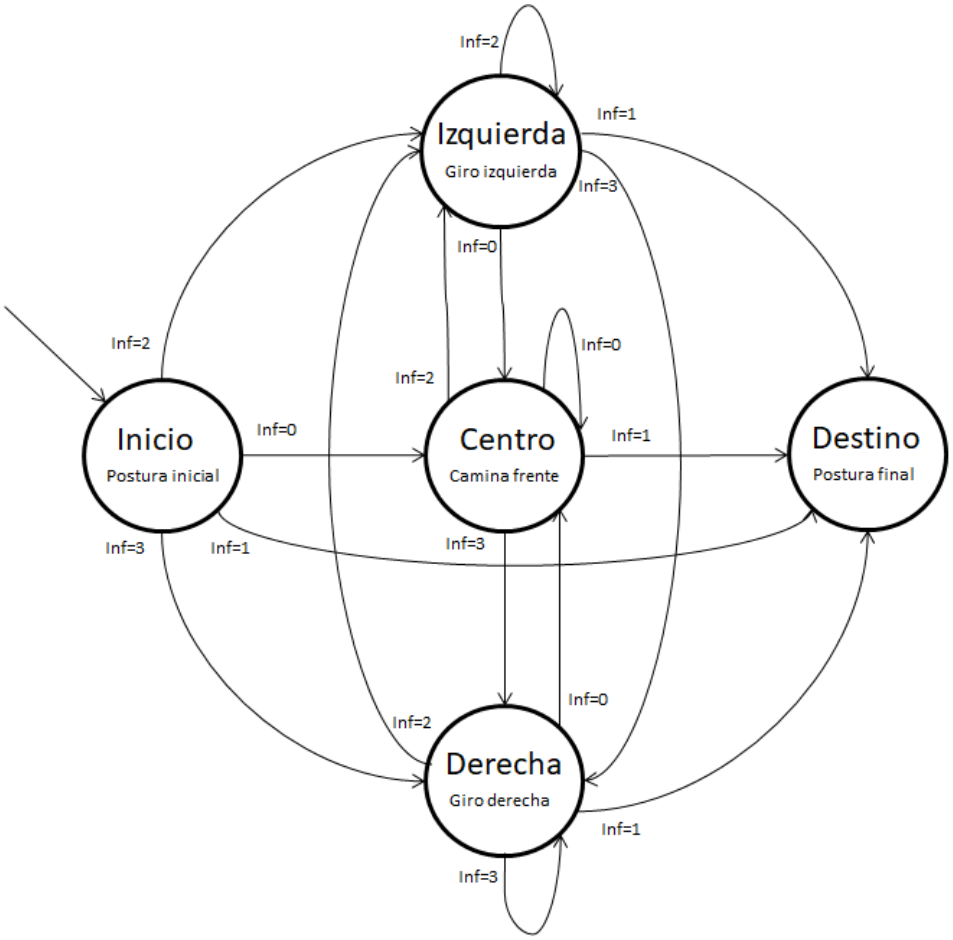


**Figure 3.11.:** Diagrama de estados del segundo desarrollo

Como puede apreciarse en la figura 3.11, se ha diseñado una máquina de Moore, la entrada es el resultado de la inferencia realizada por la red neuronal, y la salida es una respuesta motriz del robot. La máquina de estados diseñada está compuesta de los siguientes estados y transiciones:

- Estados:
  - **Inicio**: El estado inicial sirve como punto de partida del control, cada vez que se entra en dicho estado, el robot toma una postura inicial adecuada para capturar la primera imagen correctamente. Se ha elegido la posición de calibración (ver figura 2.8), para ello se envía por el puerto serie la cadena de caracteres «*kcalib*».
  - **Izquierda**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la izquierda. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrL*» y el tiempo de ejecución de dicho giro (configurado a 1 segundo para todas las instrucciones de giro del proyecto).

  - **Derecha**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la derecha. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrR*» y el tiempo de ejecución de dicho giro.

  - **Centro**: Cada vez que el sistema entra en este estado, el robot camina brevemente hacia delante. Para ello, se envía por el puerto serie la cadena de caracteres «*kcr*F» y el tiempo de ejecución de dicho gateo.

  - **Destino**: Corresponde al estado final del control, cada vez que se entra en dicho estado, el robot toma una postura para indicar el fin. Para ello se envía por el puerto serie la cadena de caracteres «*ksit*».

- Transiciones:

  - **Transición a derecha**: cuando la entrada a la máquina de estados sea una clase 3, el siguiente estado será «derecha».

  - **Transición a izquierda**: cuando la entrada a la máquina de estados sea una clase 2, el siguiente estado será «izquierda».

  - **Transición a centro**: cuando la entrada a la máquina de estados sea una clase 0, el siguiente estado será «centro».

  - **Transición a destino**: cuando la entrada a la máquina de estados sea una clase 1, el siguiente estado será «destino».

### 3.5.4. Algoritmo de control

Una vez desarrolladas las entidades del sistema, se ha creado un algoritmo de control siguiendo el esquema mostrado en la figura 3.12.

Una vez inicializada la máquina de estados, el lazo de control se activa. En cada iteración el algoritmo analiza si el estado actual equivale a «destino», en tal caso, el objetivo del control ha sido cumplido y se termina la rutina. En caso contrario, la red neuronal convolucional clasifica la imagen capturada para determinar la posición del objetivo. Esta información alimenta la máquina de estados, que decidirá cuál es el siguiente estado (ver figura 3.11) y se ejecutará la acción de control correspondiente.

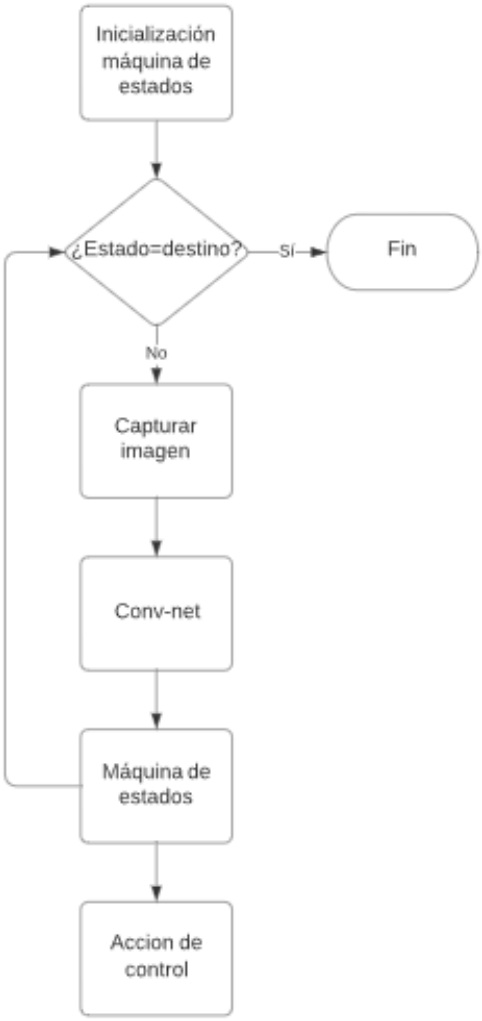


**Figure 3.12.:** Diagrama de flujo del algoritmo de control diseñado en el desarrollo II

## 3.6. Desarrollo III

El objetivo del tercer desarrollo consiste en crear un lazo de control simple para conseguir que el robot busque y encuentre la pelota. A diferencia de la problemática desarrollada en la sección anterior, el problema de clasificación se ha resuelto mediante el uso de dos modelos de Deep Learning consecutivos. El primero de ellos realiza una clasificación binaria (clase «destino» o «no destino»), y el segundo modelo clasifica la posición discreta de la pelota (izquierda, derecha o centro). De esta manera, el problema de clasificación desarrollado en la sección anterior, se ha reducido a dos problemas de más sencillez. A continuación se describe de manera sencilla el algoritmo de control propuesto:

1. Capturar imagen
2. El primer modelo de Deep Learning clasifica la situación final del recorrido: «destino» o «no destino»
3. En caso de «no destino», el segundo modelo de Deep Learning clasifica la posición relativa de la pelota: izquierda, derecha o centro
4. Según el resultado, la máquina de estados realiza una transición y su correspondiente acción de control: giro a la derecha, giro a la izquierda, andar al frente o fin de la rutina (en caso de recibir la clasificación «destino»)
5. Regresar a paso 1 en caso de no obtener clase «destino»

### 3.6.1. Dataset modelo I

El primer modelo de Deep Learning va a realizar una clasificación binaria para conocer si el robot ha llegado a destino o no. Para ello, es necesario entrenar al modelo con imágenes de pelotas en posición destino, y pelotas en cualquier otra posición y distancia.

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 240 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 309 imágenes que no han alimentado el modelo durante el entrenamiento.

### 3.6.2. Dataset modelo II

El segundo modelo de Deep Learning va a realizar una clasificación para discretizar la posición relativa de la pelota (izquierda, derecha o centro). Para ello, es necesario entrenar al modelo con imágenes de pelotas en distintas posiciones y distancias.

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 733 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 180 imágenes que no han alimentado el modelo durante el entrenamiento.

### 3.6.3. Entrenamiento

Para el entrenamiento de ambos modelos se ha configurado una red neuronal convolucional VGGnet (figura 2.34), se ha utilizado la tipología de 16 capas para realizar las clasificaciones correspondientes. Usando las librerías de Tensorflow y Keras [98] se ha construido el modelo VGG-16:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 512) | 0 |
| dense (Dense) | (None, 4096) | 2101248 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dense_2 (Dense) | (None, 1000) | 4097000 |
| dense_3 (Dense) | (None, 2) | 2002 |

Total params: 37,696,250
Trainable params: 22,981,562
Non-trainable params: 14,714,688

**Figure 3.13.:** Arquitectura del modelo de clasificación binario

Puesto que el primer modelo debe clasificar la posición final objetivo en dos categorías, la última capa del modelo debe contener 2 neuronas, cada una de ellas retorna la probabilidad de que el objetivo se encuentre en una determinada categoría.

El segundo modelo, debe clasificar la posición discreta del objetivo en tres categorías: izquierda, derecha o centro. Como este modelo debe realizar la misma función que el modelo presentado en la sección 3.4, se ha reciclado el modelo construido y mostrado en la figura 3.4.

El entrenamiento realizado en el modelo binario y ternario, es de tres y cuatro épocas respectivamente, y se ha utilizado la función de pérdida logística (ecuación 2.11) a modo de señal de error.

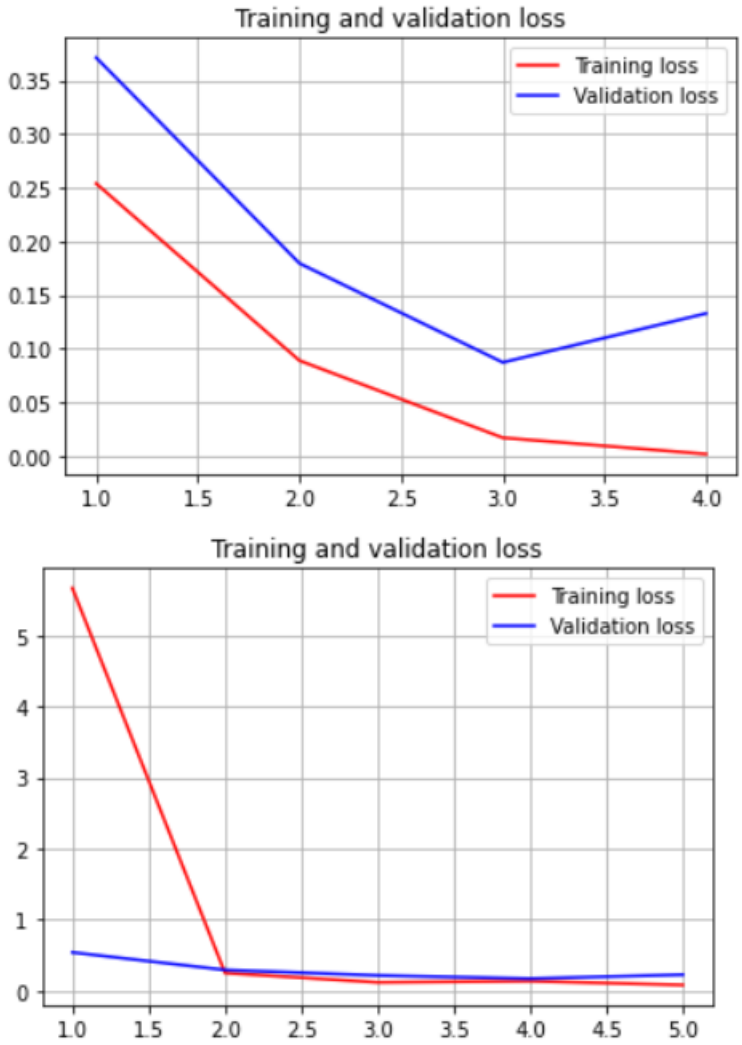


**Figure 3.14.:** Representación gráfica de la función de pérdida en el entrenamiento del modelo binario (grafico superior) y del modelo ternario (grafico interior)

En la figura 3.14, puede observarse como el fenómeno de sobredimensionado aparece en la tercera y cuarta época del entrenamiento.Una vez el modelo ha sido entrenado, es posible utilizar las redes neuronales como entidades que convierten la imagen de entrada en una clase numérica, según muestran las figuras 3.15 y 3.16.

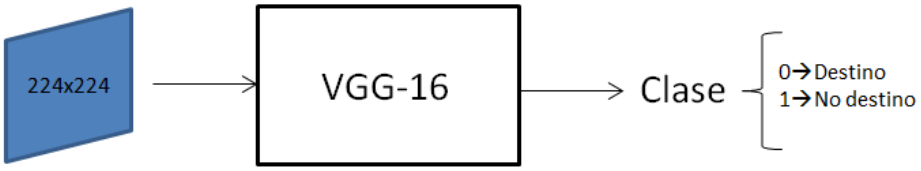


**Figure 3.15.:** Representación funcional del modelo VGG-16 entrenado para determinar el destino

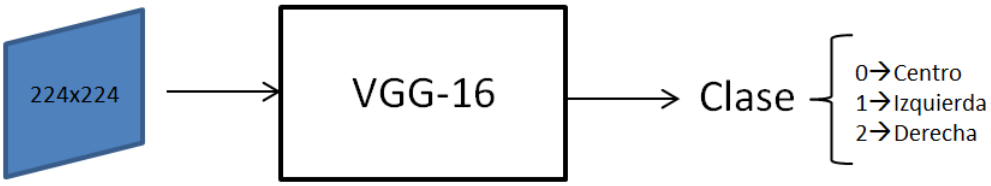


**Figure 3.16.:** Representación funcional del modelo VGG-16 entrenado para clasificar la posición del objetivo (izquierda, derecha o centro)

### 3.6.4. Máquina de estados

Para el desarrollo de este algoritmo de control, se ha usado la misma máquina de estados diseñada en la sección 3.5.3.

### 3.6.5. Algoritmo de control

Una vez desarrolladas las entidades del sistema, se ha creado un algoritmo de control siguiendo el siguiente esquema:

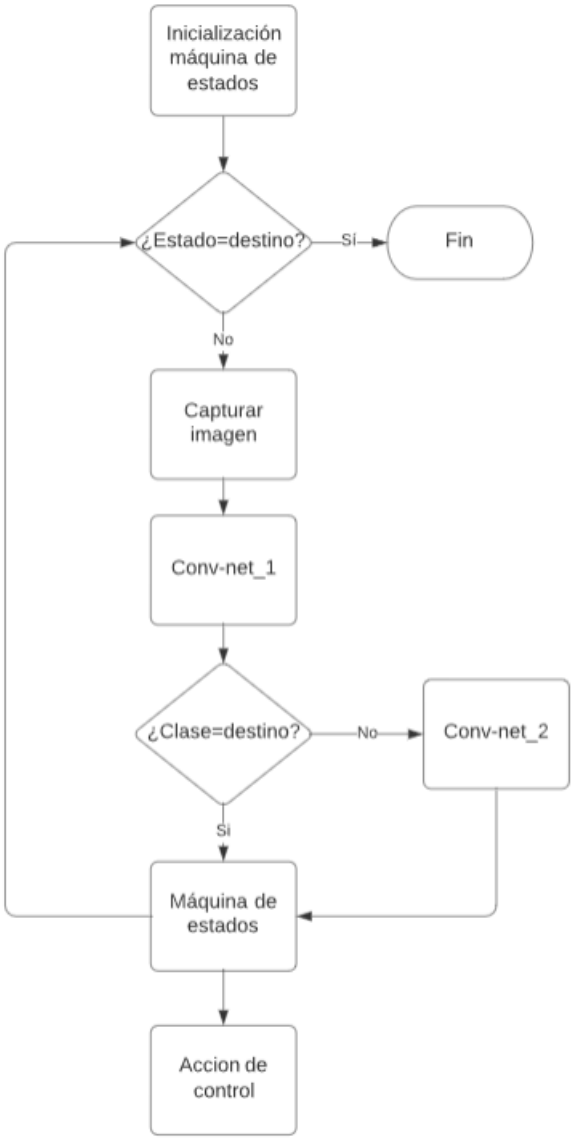


**Figure 3.17.:** Esquema del algoritmo de control del segundo desarrollo

Una vez inicializada la máquina de estados, el lazo de control se activa. En cada iteración el algoritmo analiza si el estado actual equivale a «centro», en tal caso, el objetivo del control ha sido cumplido y se termina la rutina. En caso contrario, la primera red neuronal convolucional clasifica la imagen capturada para determinar si el objetivo ha sido alcanzado, si es así, esta información alimenta directamente a la máquina de estados. En caso contrario (no «destino), la segunda red neuronal clasifica la posición del objetivo en la imagen capturada. La información resultante alimenta la máquina de estados, que decidirá cuál es el siguiente estado (ver figura 3.11) y se ejecutará la acción de control correspondiente.

## 3.7. Desarrollo IV

El objetivo del cuarto desarrollo consiste en crear un lazo de control simple para conseguir que el robot busque y encuentre la pelota. A diferencia de la problemática desarrollada en la sección anterior, ambos modelos realizan una clasificación binaria, pero han sido entrenados para determinar distintas clases. El primero de ellos se utiliza para determinar si el objetivo se ha alcanzado, y el segundo se utiliza para determinar la posición del objetivo (izquierda o derecha). De esta manera, se simplifica el segundo modelo utilizado en el desarrollo anterior. A continuación se describe de manera sencilla el algoritmo de control propuesto:

1. Capturar imagen
2. El primer modelo de Deep Learning clasifica la situación final del recorrido: «destino» o «no destino»
3. En caso de «no destino», el segundo modelo de Deep Learning clasifica la posición relativa de la pelota: izquierda o derecha
4. Según el resultado, la máquina de estados realiza una transición y su correspondiente acción de control: giro a la derecha, giro a la izquierda, andar al frente o fin de la rutina (en caso de recibir la clasificación «destino»)
5. Regresar a paso 1 en caso de no obtener clase «destino»

### 3.7.1. Dataset modelo I

El primer modelo de Deep Learning va a realizar una clasificación binaria para conocer si el robot ha llegado a destino o no. Para ello, es necesario entrenar al modelo con imágenes de pelotas en posición destino, y pelotas en cualquier otra posición y distancia.

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 240 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 309 imágenes que no han alimentado el modelo durante el entrenamiento.

### 3.7.2. Dataset modelo II

El segundo modelo de Deep Learning va a realizar una clasificación para discretizar la posición relativa de la pelota (izquierda o derecha). Para ello, es necesario entrenar al modelo con imágenes de pelotas en distintas posiciones y distancias.

El dataset se ha generado manualmente, utilizando el sensor cámara se han capturado imágenes de la pelota en distintas posiciones. El mecanismo de etiquetación es simple, basta con crear un directorio para cada clase y guardar las imágenes correspondientes en cada uno de ellos. El conjunto de datos de entrenamiento está formado por 480 imágenes en total, de las cuales el 66% se utilizan para el entrenamiento propiamente dicho y el 33% para la validación. Para determinar la precisión del modelo también se ha creado un conjunto de datos de test, formado por 120 imágenes que no han alimentado el modelo durante el entrenamiento.

### 3.7.3. Entrenamiento

Para el entrenamiento de ambos modelos se ha configurado una red neuronal convolucional VGGnet (figura 2.34), se ha utilizado la tipología de 16 capas para realizar las clasificaciones correspondientes. Usando las librerías de Tensorflow y Keras [98] se ha construido el modelo VGG-16:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 224, 224, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 224, 224, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 112, 112, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 112, 112, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 112, 112, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 56, 56, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 56, 56, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 56, 56, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 28, 28, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 28, 28, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 28, 28, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 14, 14, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 14, 14, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 7, 7, 512) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 512) | 0 |
| dense (Dense) | (None, 4096) | 2101248 |
| dense_1 (Dense) | (None, 4096) | 16781312 |
| dense_2 (Dense) | (None, 1000) | 4097000 |
| dense_3 (Dense) | (None, 2) | 2002 |

Total params: 37,696,250
Trainable params: 22,981,562
Non-trainable params: 14,714,688

**Figure 3.18.:** Arquitectura del modelo de clasificación binario

Puesto que ambos modelos deben clasificar la posición del objetivo en dos categorías, la última capa del modelo debe contener 2 neuronas, cada una de ellas retorna la probabilidad de que el objetivo se encuentre en una determinada categoría.

Para determinar si el objetivo se ha alcanzado, se ha utilizado el modelo binario presentado en la sección 3.6, de modo que únicamente se ha entrenado el segundo modelo propuesto:


**Figure 3.19.:** Representación gráfica de la función de pérdida en el entrenamiento del modelo binario (grafico superior) y del modelo ternario (grafico interior)

El entrenamiento realizado es de tres épocas y se ha utilizado la función de pérdida logística (ecuación 2.11) a modo de señal de error .Una vez el modelo ha sido entrenado, es posible utilizar las redes neuronales como entidades que convierten la imagen de entrada en una clase numérica, según muestran las figuras 3.20 y 3.21.

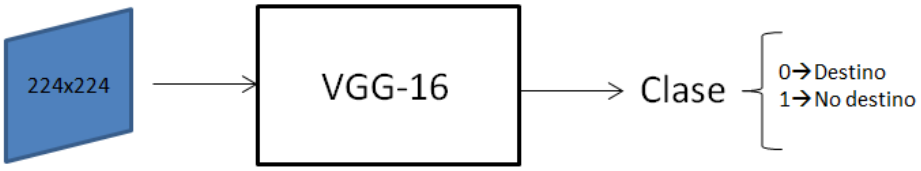


**Figure 3.20.:** Representación funcional del modelo VGG-16 entrenado para identificar el destino

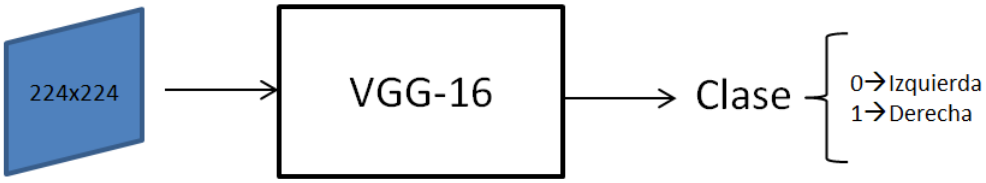


**Figure 3.21.:** Representación funcional del modelo VGG-16 entrenado para clasificar la posición del objetivo

### 3.7.4. Máquina de estados

Para el desarrollo de este algoritmo de control, se ha modificado la máquina de estados diseñada en la sección 3.5.3, eliminando el estado «centro» y modificando las condiciones de cada transición.

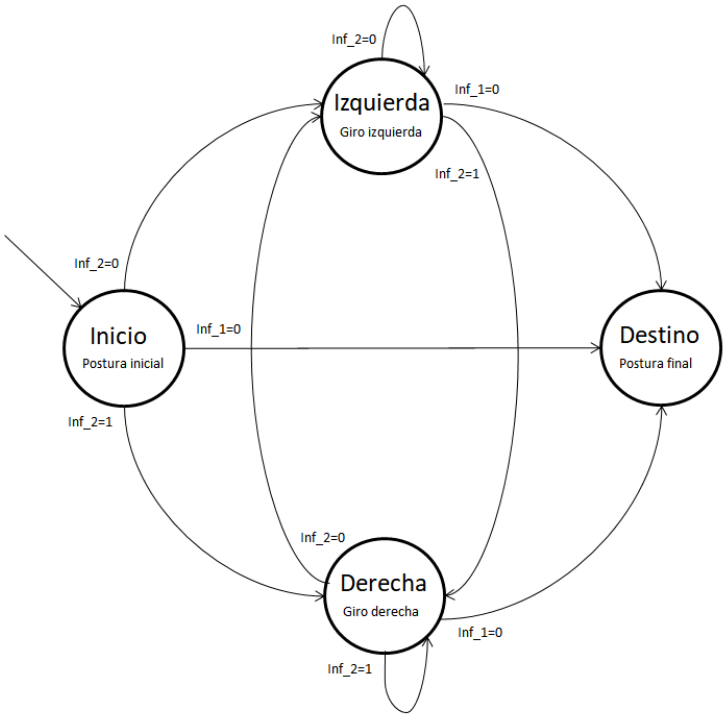


**Figure 3.22.:** Máquina de estados implementada en el desarrollo IV

Como puede apreciarse en la figura 3.22, se ha diseñado una máquina de Moore, la entrada es el resultado de la inferencia realizada por la red neuronal, y la salida es una respuesta motriz del robot. La máquina de estados diseñada está compuesta de los siguientes estados y transiciones:

- Estados:

  - **Inicio**: El estado inicial sirve como punto de partida del control, cada vez que se entra en dicho estado, el robot toma una postura inicial adecuada para capturar la primera imagen correctamente. Se ha elegido la posición de calibración (ver figura 2.8), para ello se envía por el puerto serie la cadena de caracteres «*kcalib*».

  - **Izquierda**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la izquierda. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrL*» y el tiempo de ejecución de dicho giro (configurado a 1 segundo para todas las instrucciones de giro del proyecto).

  - **Derecha**: Cada vez que el sistema entra en este estado, el robot ejecuta un pequeño giro a la derecha. Para ello, se envía por el puerto serie la cadena de caracteres «*kcrR*» y el tiempo de ejecución de dicho giro.

  - **Destino**: Corresponde al estado final del control, cada vez que se entra en dicho estado, el robot toma una postura para indicar el fin. Para ello se envía por el puerto serie la cadena de caracteres «*ksit*».

- Transiciones:

  - **Transición a derecha**: cuando la entrada a la máquina de estados sea una clase 3, el siguiente estado será «derecha».

  - **Transición a izquierda**: cuando la entrada a la máquina de estados sea una clase 2, el siguiente estado será «izquierda».

  - **Transición a destino**: cuando la entrada a la máquina de estados sea una clase 1, el siguiente estado será «destino».

### 3.7.5. Algoritmo de control

El algoritmo de control utilizado en esta sección es idéntico al detallado en el desarrollo III (ver sección 3.6) de este proyecto:

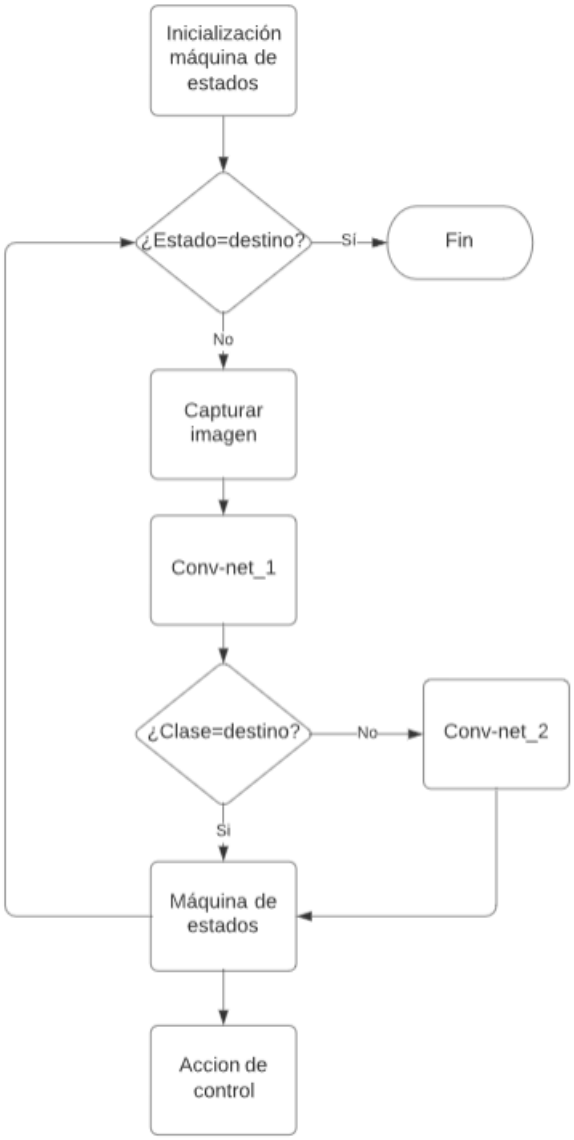


**Figure 3.23.:** Diagrama de flujo del algoritmo de control utilizado en el desarrollo IV

# 4. Resultados

## 4.1. Perspectiva general

En este capítulo se presentan los resultados obtenidos en el desarrollo del proyecto. En la sección 4.2 se muestra la precisión obtenida de los distintos modelos entrenados en cada fase del desarrollo. En las secciones 4.3 y 4.4 se describe y discute el comportamiento observado de cada estrategia de control implementada, así como los parámetros más relevantes de cada lazo de control.

## 4.2. Precisión del modelo

En esta sección se muestra la precisión de cada red neuronal convolucional entrenada. Para ello, el conjunto de datos de test (ver sección 3) se ha utilizado para computar la matriz de confusión.

**Desarrollo I** En este primer paso del proyecto, se ha entrenado una red neuronal VGG-16 con imágenes correspondientes a pelotas situadas en distintas posiciones relativas: derecha, izquierda y centro. Una vez el modelo ha sido entrenado, se ha utilizado para clasificar el siguiente conjunto de imágenes de test:

- 14 imágenes de pelotas centradas
- 21 imágenes de pelotas situadas a la izquierda del robot
- 21 imágenes de pelotas situadas a la derecha del robot

La matriz de confusión computada se presenta a continuación:

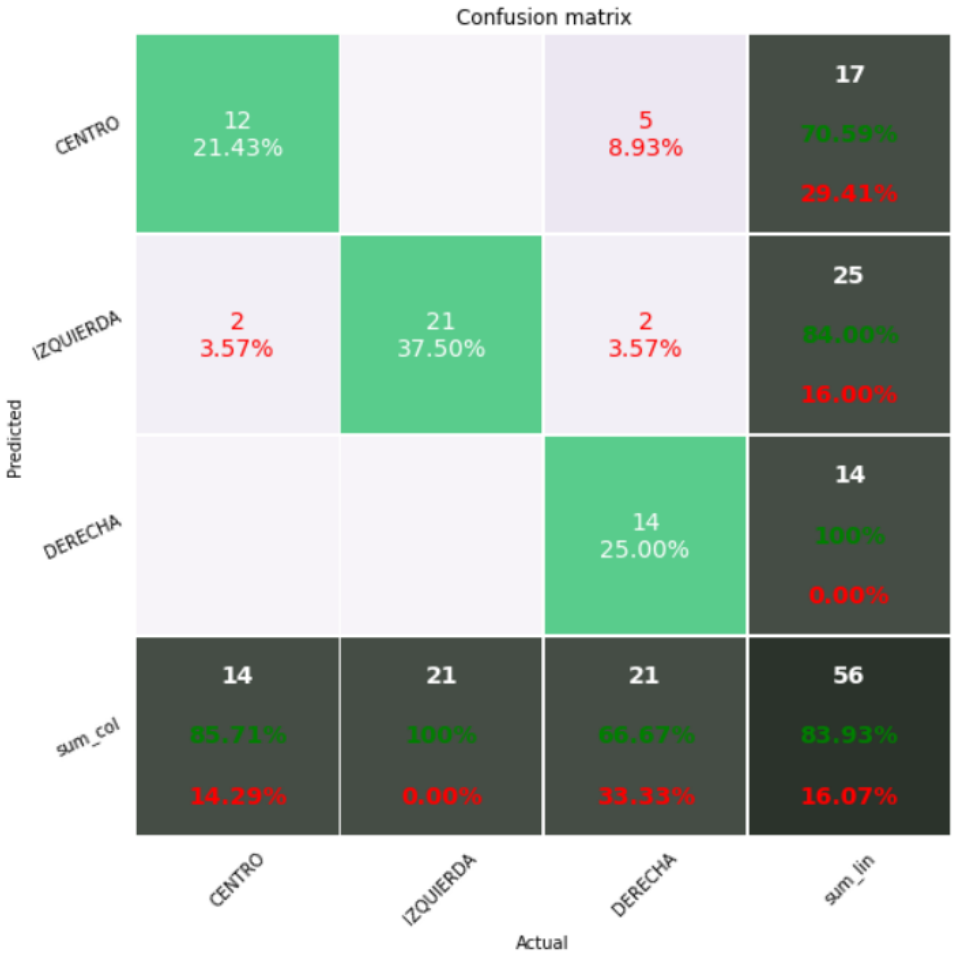


**Figure 4.1.:** Matriz de confusión del modelo entrenado en el desarrollo 1

Como se muestra en la figura 4.1, el modelo es capaz de clasificar satisfactoriamente la gran mayoría de imágenes del conjunto. Se ha obtenido una precisión del 83'9%. También puede observarse como las clases «centro» e «izquierda» contiene un número de falsos negativos reducido o nulo, y como las predicciones realizadas sobre el conjunto de imágenes de clase derecha sufren una desviación más importante en su clasificación.

**Desarrollo II** En el segundo desarrollo del proyecto, se ha entrenado una red neuronal VGG-16 con imágenes correspondientes a pelotas situadas en distintas posiciones relativas: derecha, izquierda, centro y destino. Una vez el modelo ha sido entrenado, se ha utilizado para clasificar el siguiente conjunto de imágenes de test:

- 20 imágenes de pelotas centradas
- 20 imágenes de pelotas situadas a la izquierda del robot
- 20 imágenes de pelotas situadas a la derecha del robot

- 20 imágenes de pelotas «destino»

La matriz de confusión computada se presenta a continuación:

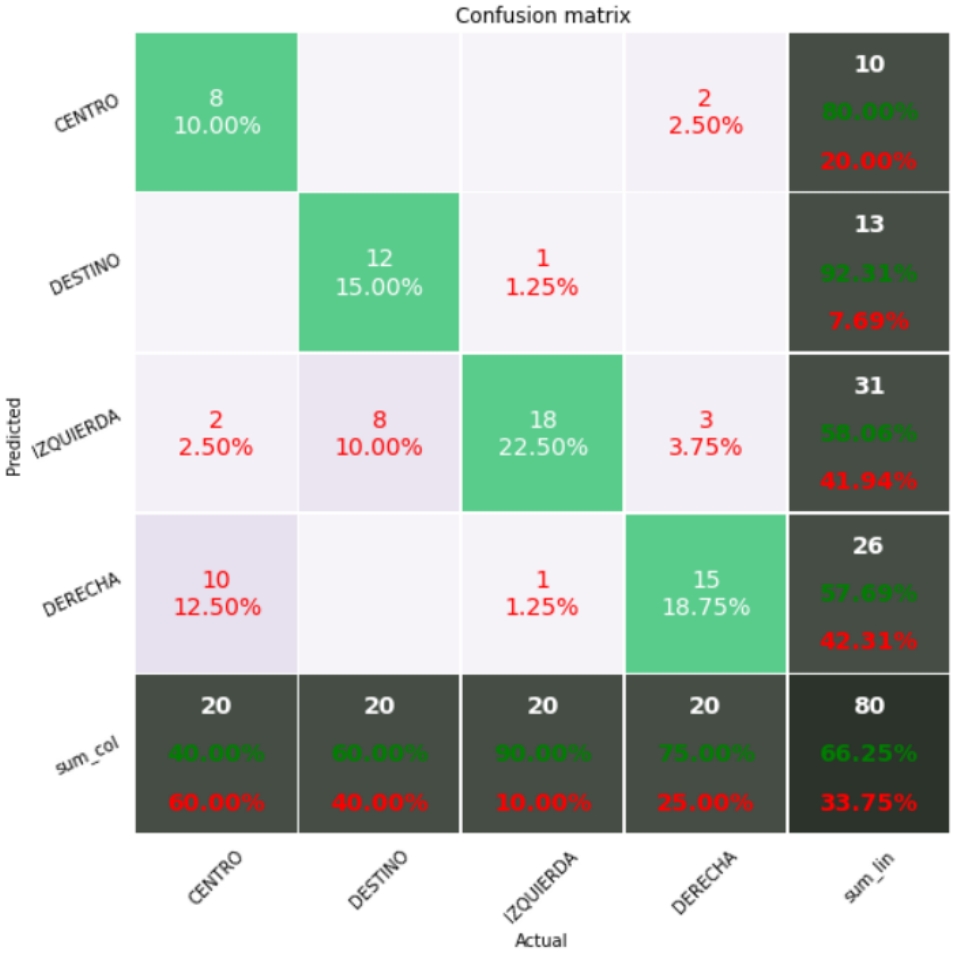


**Figure 4.2.:** Matriz de confusión del modelo entrenado en el desarrollo 2

Como se muestra en la figura 4.2, el modelo es capaz de clasificar satisfactoriamente la mayoría de imágenes del conjunto. Se ha obtenido una precisión del 66%. También puede observarse como las predicciones realizadas sobre el conjunto de imágenes de clase «centro» son las más erróneas del conjunto, seguidas de las clasificaciones realizadas sobre la clase «destino».

**Desarrollo III** En este desarrollo, dos redes neuronales convolucionales VGG-16 han sido entrenadas (ver sección 3.6). El primer modelo ha sido entrenado para discernir si el objetivo ha sido encontrado (clase «destino») o no. El segundo modelo ha sido entrenado para clasificar la posición discreta del objetivo (izquierda, derecha o centro). Los conjuntos de test están formados por:

- Modelo de clasificación binaria:
  - 29 imágenes de pelotas «destino»
  - 280 imágenes de pelotas «no destino»
- Modelo para clasificar la posición:
  - 60 imágenes de pelotas centradas
  - 60 imágenes de pelotas situadas a la izquierda del robot
  - 60 imágenes de pelotas situadas a la derecha del robot

La matriz de confusión del primer modelo se presenta a continuación:

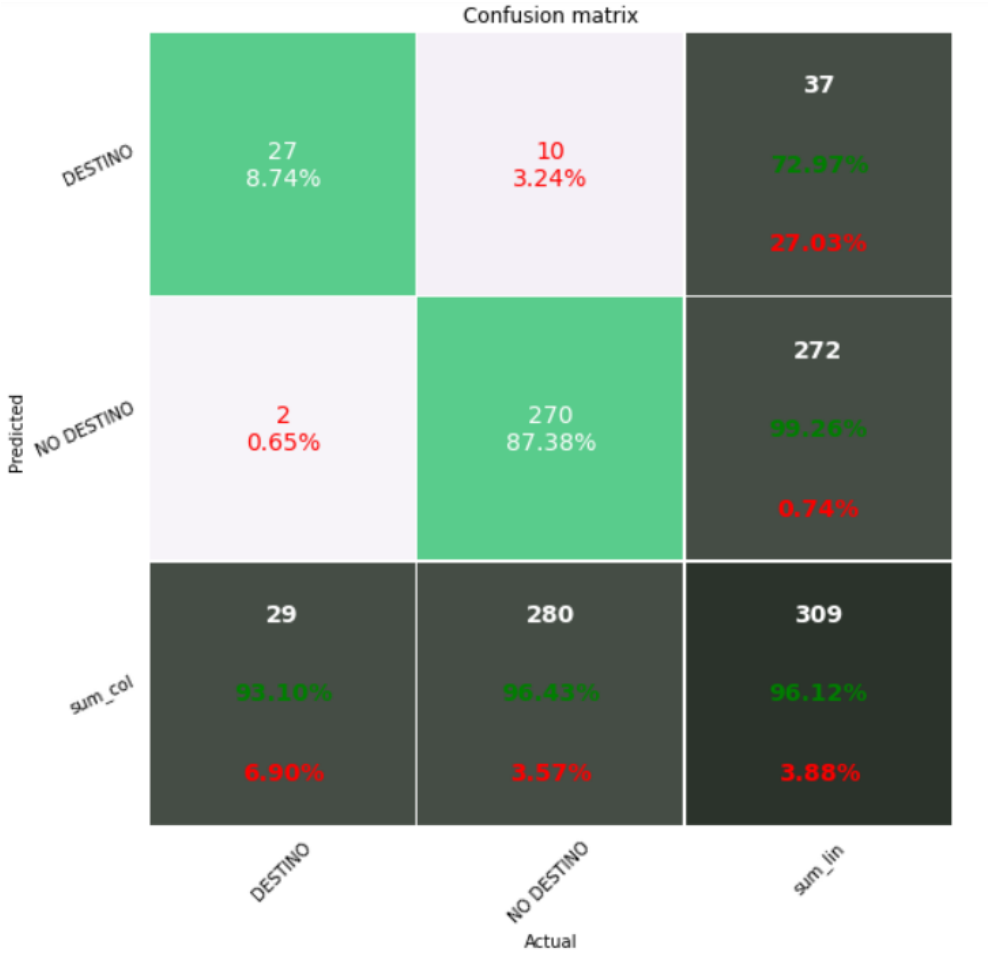


**Figure 4.3.:** Matriz de confusión del modelo entrenado en el desarrollo 3

Como muestra la figura 4.3, el modelo es capaz de clasificar satisfactoriamente la gran mayoría de imágenes del conjunto. Se ha obtenido una precisión del 96%, siendo un modelo de alta fiabilidad para detectar si el robot ha alcanzado el objetivo.

La matriz de confusión del segundo modelo se presenta a continuación:

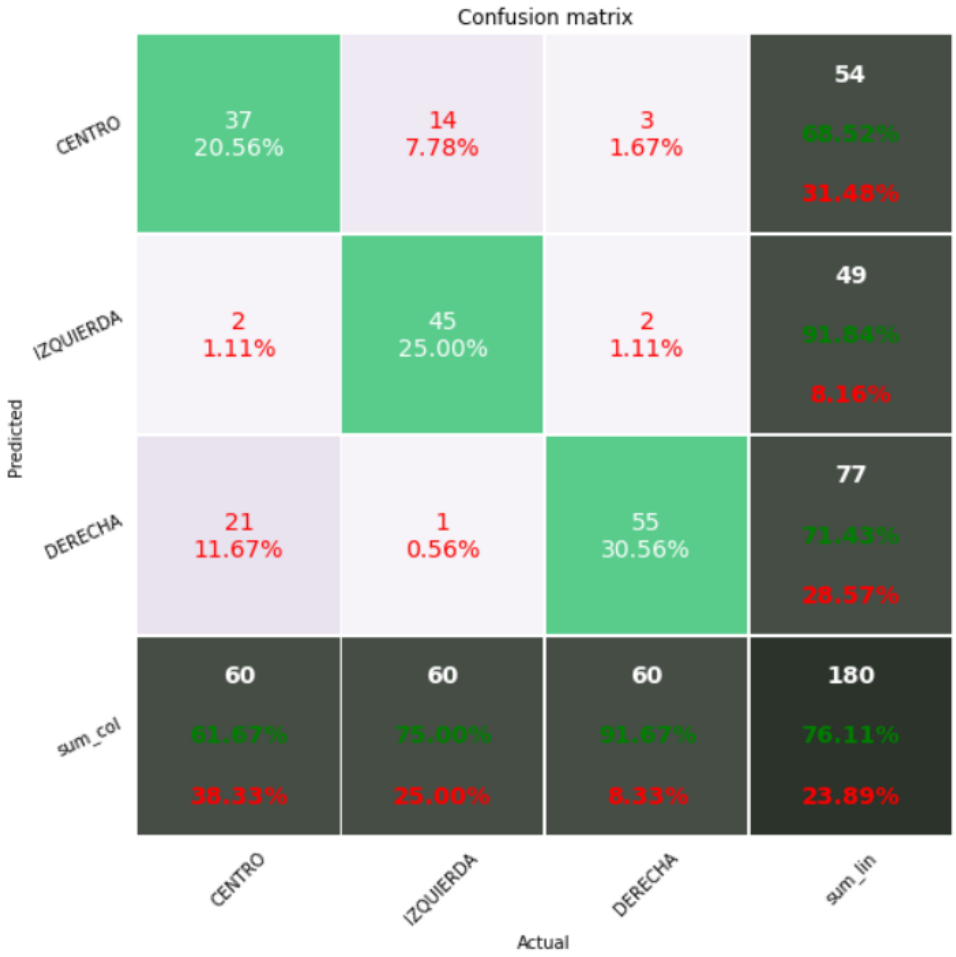


**Figure 4.4.:** Matriz de confusión del modelo entrenado en el desarrollo 3

Como se muestra en la figura 4.4, el modelo es capaz de clasificar satisfactoriamente la mayoría de imágenes del conjunto. Se ha obtenido una precisión del 76%. También puede observarse como las predicciones realizadas sobre el conjunto de imágenes de clase «centro» sufren una desviación más importante en su clasificación.

**Desarrollo IV** La única diferencia con el desarrollo anterior reside en la red neuronal convolucional utilizada para determinar la posición del objetivo. En el desarrollo IV se han utilizado dos modelos, el primer modelo clasifica si el objetvio se ha alacanzado o no (idéntico al desarrollo anterior), y el segundo modelo clasifica si el objetivo se encuentra a la izquierda o a la derecha del robot. El conjunto de datos de test se ha reciclado del desarrollo anterior, y la matriz de confusión computada se muestra a continuación:

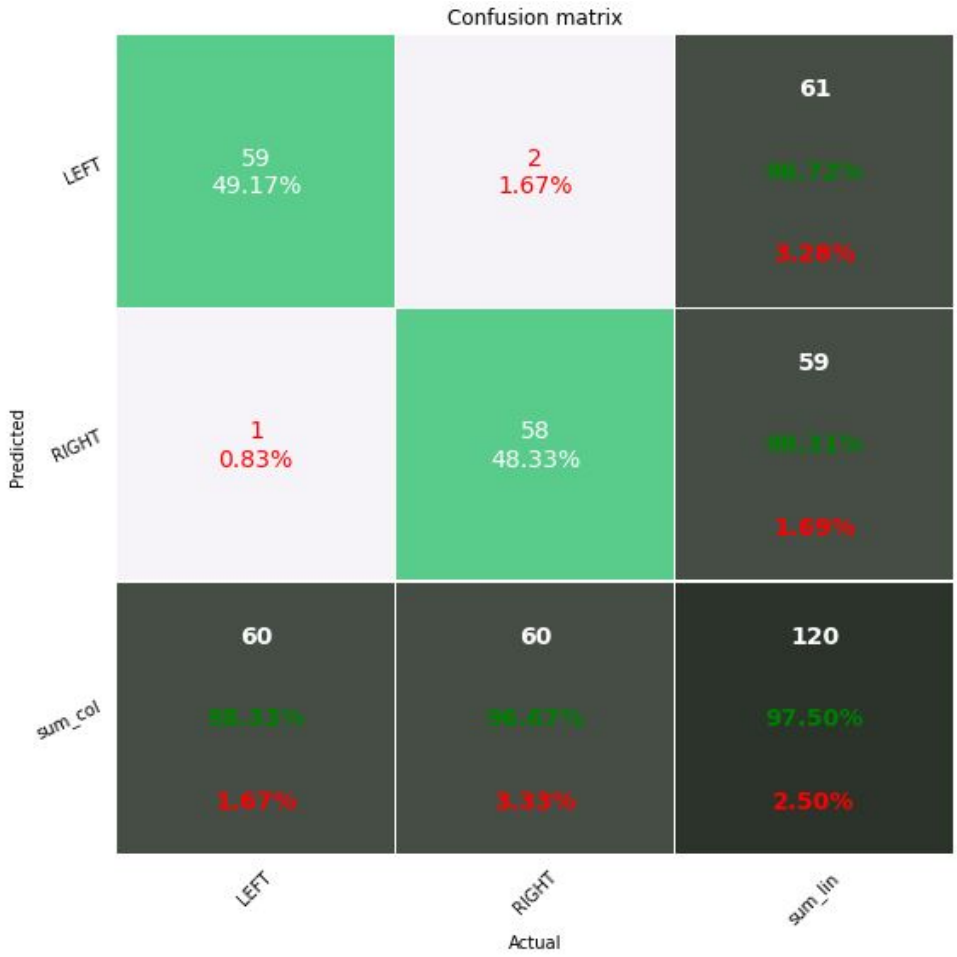


**Figure 4.5.:** Matriz de confusión del modelo entrenado en el desarrollo 3

Como muestra la figura 4.5, el modelo es capaz de clasificar satisfactoriamente la gran mayoría de imágenes del conjunto. Se ha obtenido una precisión del 97%, siendo un modelo de alta fiabilidad para determinar la posición del objetivo.

## 4.3. Resultados prácticos

### 4.3.1. Lazo de control

De manera general, el lazo de control consiste en capturar una imagen, clasificarla y en base al resultado realizar un movimiento u otro. De esta manera, se pretende imitar el mecanismo de planificación de ruta usado por los seres humanos. A medida que el ser humano se mueve, analiza constantemente información del entorno mediante los ojos, y realiza movimientos en base a ello [101]. En el sistema seleccionado se dispone de una Raspberry Pi de 2 GB de RAM como unidad de control.

Así pues, clasificar una imagen utilizando una red neuronal convolucional toma más tiempo de lo deseado. El tiempo de cómputo de cada inferencia oscila entre 2 y 2'1 segundos independientemente del modelo.

Debido a las limitaciones computacionales de la unidad de control, clasificar la posición del objetivo durante el movimiento resulta ser contraproducente. La posición del robot en el instante que captura una imagen y en el instante que clasifica la posición, son totalmente distintas, esta incongruencia provoca que el robot se pierda fácilmente. En la figura 4.6 se representa con más claridad esta limitación:

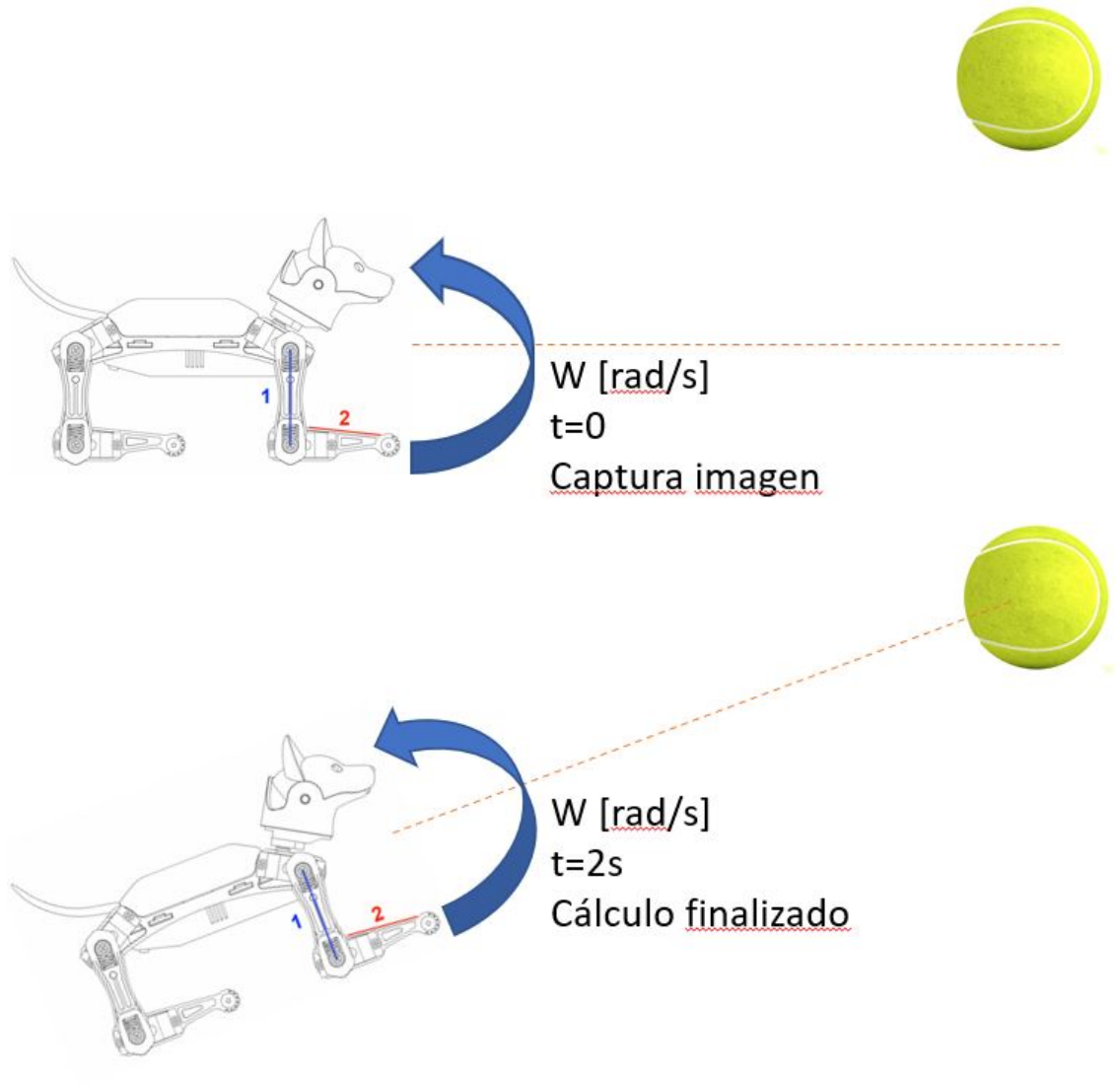


**Figure 4.6.:** Representación del comportamiento del sistema cuando movimiento y clasificación se ejecutan en paralelo

En un instante inicial, el robot se encuentra en una maniobra de giro y captura una imagen del objetivo. Como puede verse en la parte superior, en dicho instante

el objetivo se encuentra a la izquierda del robot. Dos segundos después (figura inferior), la clasificación se ha realizado y el robot se encuentra en una posición distinta. El resultado de la inferencia determina que el objetivo está a la izquierda (imagen capturada en t=0) pero realmente se encuentra en frente. Esto provoca que en la siguiente iteración, el robot siga girando a la izquierda en cambio de caminar a delante.

Para solucionar este problema, se ha incorporado una instrucción de pausa al finalizar cada cadena de movimientos. De esta manera se aporta consistencia al lazo de control, la clasificación de la imagen se obtiene en la misma posición en que la imagen fue capturada.

## 4.4. Observaciones y discusión

### 4.4.1. Desarrollo I

El objetivo del primer desarrollo consiste únicamente en orientar el robot al objetivo, como se ha visto en la sección 3.4. Para ello, se ha entrenado la red neuronal con imágenes de pelotas en distintas posiciones, tomadas desde la misma distancia inicial.

En la figura 4.1 se puede observar cómo el modelo entrenado es capaz de clasificar con una precisión considerable la posición relativa de la pelota, siendo la clase "centro" la clasificación con más falsos negativos, es decir, el modelo asigna otras clases a imágenes con pelotas centradas. En la práctica, este fenómeno se traduce un una dificultad para obtener la clase "centro", es decir, una dificultad para determinar el fin de la rutina (u objetivo alcanzado, ver figura 3.6).

Por otro lado, las instrucciones de giro disponibles en la librería OpenCat no ofrecen rotaciones puras, son movimientos de rototranslación. En la práctica, esto ha provocado que en cada maniobra de movimiento, el robot no solo gire, sino que también avance hacia la dirección clasificada. Puesto que la red neuronal convolucional ha sido entrenada con imágenes tomadas desde la posición inicial, en el momento que el robot avanza en el espacio (debido al fenómeno de rototranslación de cada maniobra) y captura imágenes desde otra distancia o punto de vista, aparecen nuevos patrones que el modelo no ha aprendido, resultando en clasificaciones incorrectas y posteriormente, en trayectorias de movimiento divergentes.

Aunque el comportamiento observado en las primeras iteraciones del control es correcto, ya que el robot clasifica correctamente las imágenes tomadas desde una distancia inicial, a medida que el robot avanza en el espacio, la clasificación del modelo pierde efectividad y provoca el fracaso en la búsqueda del objetivo.

### 4.4.2. Desarrollo II

El objetivo del segundo desarrollo consiste en desplazarse hasta el objetivo partiendo de una posición inicial. Los resultados prácticos del desarrollo anterior han servido para evidenciar la necesidad de entrenar la red neuronal convolucional con imágenes del objetivo tomadas desde distintas distancias y puntos de vista. También, a modo de mecanismo de detección del fin de la rutina (objetivo alcanzado), la red neuronal se ha configurado para detectar cuándo el objetivo se encuentra muy cerca del robot (clase "destino"), a parte de la detección de la posición del objetivo (izquierda, derecha o centro).

En la figura 4.2, de nuevo, la clase "centro" sigue siendo la menos precisa, seguida de la clase "destino". Dando lugar a un modelo con una precisión relativamente baja (66%). En las pruebas de funcionamiento, en cambio, se han observado resultados satisfactorios, el robot ha sido capaz de buscar y alcanzar el objetivo en algunas ocasiones. Aunque la clase "centro" no se clasifique correctamente, las clases "izquierda" y "derecha" se clasifican con precisión, y debido al fenómeno de rototranslación comentado en la sección anterior, el robot consigue avanzar correctamente hasta el destino.

Uno de los problemas observados en el funcionamiento de este desarrollo es la aparición de falsos positivos en la clase "destino", en la práctica, esto se traduce en finalizar la rutina de movimiento cuando el robot no ha alcanzado al objetivo, siendo uno de los fallos más graves que pueden ocurrir. En los siguientes desarrollos esta problemática se solventa.

### 4.4.3. Desarrollo III

Este desarrollo se ha centrado en mejorar el desempeño del lazo de control presentado en la sección anterior. Principalmente, en este desarrollo se ha mejorado la precisión general del modelo de clasificación, específicamente la precisión en la clase "destino". Para resolver la problemática observada y explicada en la sección anterior, se han implementado dos cambios esenciales:

1. **Aumento del conjunto de datos:** Con la intención de aumentar la precisión general del modelo, aumentar el conjunto de datos de entrenamiento suele ser beneficioso [64]. Con un conjunto de datos mayor, el modelo puede aprender más patrones y características.
2. **División del problema de clasificación:** Intentar resolver el problema de clasificación multivariable con un único modelo ha resultado ser contraproducente en términos de precisión. En este desarrollo se divide el problema de clasificación en dos grupos. Un modelo ha sido entrenado para realizar una clasificación binaria y discernir si el robot ha llegado al objetivo (clase «destino») y otro modelo ha sido entrenado para determinar la posición relativa del objetivo.

Los cambios realizados han sido beneficiosos. Como se muestra en las figuras 4.3 y 4.4, la precisión general en la clasificación ha aumentado, llegando a obtener un modelo de alta fiabilidad en la clasificación de la clase "destino". En la práctica, el robot es capaz de buscar y alcanzar el objetivo en varias ocasiones, también se ha reducido al mínimo la aparición de falsos positivos en la clase "destino", garantizando el fin de la rutina una vez el robot alcanza el objetivo. Por otro lado, el uso de dos redes neuronales convolucionales concatenadas, aumenta considerablemente el tiempo de ejecución de cada iteración, dando lugar a un control extremadamente lento.

### 4.4.4. Desarrollo IV

Las modificaciones realizadas en este desarrollo han surgido de las siguientes observaciones:

1. El excelente desempeño de la red neuronal en un problema de clasificación binaria
2. El hecho de no necesitar la clase «centro» para alcanzar el objetivo

Utilizando el algoritmo de control del desarrollo III, se ha substituido el modelo de clasificación ternaria (izquierda, derecha y centro), por un modelo de clasificación binaria para determinar la posición relativa de la pelota (derecha o izquierda). Como muestra la figura 4.5, se ha obtenido un modelo de alta precisión y fiabilidad.

En la práctica, aunque el control sigue siendo lento debido al cálculo de dos inferencias consecutivas, el desempeño del robot en la búsqueda del objetivo aumenta satisfactoriamente. Debido a la reducción de fallos en la clasificación y a la excelente precisión de los modelos, el objetivo ha sido alcanzado con éxito en todas las pruebas realizadas.

# 5. Conclusiones y futuras líneas de trabajo

## 5.1. Conclusiones

En este trabajo, una estrategia de movimiento, basada en inteligencia artificial, para un robot cuadrúpedo ha sido desarrollada con éxito. El lazo de control diseñado consta de un sensor cámara, una red neuronal convolucional entrenada para clasificar la posición del objetivo y una estructura sencilla de robot cuadrúpedo.

El uso de una red neuronal para resolver una problemática de alto nivel ha mostrado resultados excelentes, obteniendo modelos de alta fiabilidad y precisión en la clasificación. Aportando información de calidad al control, el robot es capaz de alcanzar con éxito el objetivo. Mientras más fiable sea la clasificación del modelo, más probabilidad de éxito tendrá el robot en la búsqueda del objetivo. El algoritmo de control del robot se ha construido sobre una máquina de estados finitos, que determina la siguiente acción de control basándose en la información proporcionada por la red neuronal.

Las tecnologías utilizadas y sus principios teóricos, se han discutido en el Capítulo 2, describiendo el estado actual de los robots cuadrúpedos y profundizando en el área de las redes neuronales, explicando los principios matemáticos, algoritmos de aprendizaje y las distintas tipologías y aplicaciones de estos modelos. El estado del arte presentado condujo a la selección de componentes de hardware y software para el desarrollo del proyecto. El capítulo 3, muestra la metodología seguida en el desarrollo del proyecto, partiendo de un objetivo simple, se ha aumentado la complejidad del control hasta conseguir resultados satisfactorios. En cada desarrollo se muestra el diseño de las entidades que conforman su control, el entrenamiento de cada modelo y las distintas máquinas de estado utilizadas. Por último, en el Capítulo 4 se describe el funcionamiento observado en cada desarrollo del proyecto, mostrando la precisión de cada red neuronal, así como las virtudes y carencias de cada lazo de control.

## 5.2. Futuras líneas de trabajo

En esta sección se presentan nuevas tareas y líneas de trabajo en un futuro previsible. Algunas de las tareas presentadas están enfocadas a mejorar o corregir características

actuales del sistema desarrollado, otras están enfocadas a impulsar y escalar la funcionalidad del sistema para hacerlo operativo en entornos reales.

- **Hardware:** La limitación principal que tiene el sistema es la lentitud del lazo de control. Esto es debido al alto coste computacional asociado a las inferencias realizadas por la red neuronal. Utilizar una unidad de control con más recursos computacionales es un paso necesario para obtener un sistema usable.
- **Expansión de la base de datos:** El proyecto se ha desarrollado en un escenario libre de ruido y obstáculos externos. Para operar en entornos reales, las redes neuronales deben ser entrenadas con conjuntos de datos que representen la realidad, para ello sería necesario ampliar la base de datos e incluir imágenes de objetivos situados en ambientes ruidosos y no ideales.
- **Nuevas técnicas de visión artificial:** el procesado de imagen y la visión por computador es un campo de la inteligencia artificial que está en expansión constante, el uso de nuevos modelos que aborden este tipo de problemas puede aumentar la funcionalidad y usabilidad de este sistema en aplicaciones reales.
- **Fuente de alimentación:** Incluir una fuente de alimentación de más capacidad y potencia es un paso necesario para llevar a cabo las tareas expuestas anteriormente. El uso de modelos más complejos requiere gran capacidad computacional, que a su vez requiere un alto consumo de energía en comparación al sistema expuesto en este proyecto.
- **Rutinas de emergencia:** En todas las pruebas realizadas, el objetivo se encontraba en el campo de visión del robot, añadir un mecanismo de detección de pérdida del rumbo (detectar la ausencia del objetivo en la imagen) y su correspondiente acción de control, reduciría la posibilidad de fracaso en la búsqueda y dotaría al robot de más autonomía.
- **Nuevos movimientos:** La plataforma de código abierto utilizada para controlar el robot permite crear nuevas cadenas de movimiento. Mediante un estudio de cinemática inversa se podrían diseñar nuevos movimientos que aporten todavía más libertad de movimiento al sistema, permitiendo realizar giros perfectos o controlar la velocidad del movimiento.

# Agradecimientos

Me gustaría agradecer al director de este proyecto, Jose Antonio Soria Perez, por el conocimiento, el tiempo y la ayuda prestada durante todo el proceso.

También me gustaría expresar el más sincero aprecio a mi familia, por el soporte y paciencia que he recibido durante mis estudios.

Y por último, agradecer a mis compañeros y amigos, Aleix Vindel y Alexander Cardenas, por el apoyo y motivación que me habéis dado desde siempre.

# Bibliography


[1] S. M. Neuman, B. Plancher, B. P. Duisterhof, S. Krishnan, C. Banbury, M. Mazumder, S. Prakash, J. Jabbour, A. Faust, G. C. H. E. de Croon, and V. J. Reddi, "Tiny robot learning: Challenges and directions for machine learning in resource-constrained robots," 5 2022. [Online]. Available: http://arxiv.org/abs/2205.05748

[2] "Aplicaciones de los robots cuadrupedos: optimizacion del mantenimiento de redes de alcantarillado," https://www.universal-robots.com/es/blog/robots-para-agricultura/, accessed: 03-01-2023.

[3] Y. Zhao, L. Gong, Y. Huang, and C. Liu, "A review of key techniques of vision-based control for harvesting robot," pp. 311–323, 9 2016.

[4] F. Sistler, "Robotics and intelligent machines in agriculture," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 3–6, 1987.

[5] Y. Edan and G. Miles, "Systems engineering of agricultural robot design," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 8, pp. 1259–1265, 1994.

[6] "Robots para agricultura: hacia un mayor aprovechamiento de los recursos," https://alisysrobotics.com, accessed: 03-01-2023.

[7] "Construccion de un robot cuadrupedo para tareas de agricultura," http://cio.repositorioinstitucional.mx/jspui/handle/1002/1197, accessed: 03-01-2023.

[8] P. Saraf, A. Sarkar, and A. Javed, "Terrain adaptive gait transitioning for a quadruped robot using model predictive control." Institute of Electrical and Electronics Engineers Inc., 2021.

[9] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[10] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4730–4737.

[11] M. Khorram and S. A. A. Moosavian, "Optimal and stable gait planning of a quadruped robot for trotting over uneven terrains," in *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, 2015, pp. 121–126.

[12] "Open source, programmable robot dog bittle," https://www.petoi.com/pages/bittle-open-source-bionic-robot-dog, accessed: 03-01-2023.

[13] "Camara raspberry pi v2 oficial - sensor sony imx219 de 8 megapixeles," https://solectroshop.com/es/camaras-raspberry-pi/1791-camara-raspberry-pi-v2-oficial-sensor-sony-imx219-de-8-megapixeles.html#description, accessed: 03-01-2023.

[14] "Raspberry pi 4 tech specs," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, accessed: 03-01-2023.

[15] "Nyboard v1 - a customized arduino board with rich peripherals," https://www.petoi.com/products/nyboard-customized-arduino-board, accessed: 03-01-2023.

[16] "python-statemachine 0.9.0," https://pypi.org/project/python-statemachine/, accessed: 03-01-2023.

[17] "Robotica: que es y como ayuda a los seres humanos hoy," https://ilab.net/robotica/, accessed: 09-12-2022.

[18] "Hombre o robot," https://www.madrid.org/, accessed: 09-12-2022.

[19] S. Kajita and B. Espiau, *Legged Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 361–389. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_17

[20] M. VUKOBRATOVIC and B. BOROVAC, "Zero-moment point, thirty five years of its life," *International Journal of Humanoid Robotics*, vol. 01, no. 01, pp. 157–173, 2004. [Online]. Available: https://doi.org/10.1142/S0219843604000083

[21] K. Waldron and R. McGhee, "The adaptive suspension vehicle," *IEEE Control Systems Magazine*, vol. 6, no. 6, pp. 7–12, 1986.

[22] C. Semini and P.-B. Wieber, *Legged Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2020, pp. 1–8. [Online]. Available: https://doi.org/10.1007/978-3-642-41610-1_59-1

[23] M. H. Raibert and E. R. Tello, "Legged robots that balance," *IEEE Expert*, vol. 1, no. 4, pp. 89–89, 1986.

[24] S. Hirose, Y. Fukuda, K. Yoneda, A. Nagakubo, H. Tsukagoshi, K. Arikawa, G. Endo, T. Doi, and R. Hodoshima, "Quadruped walking robots at tokyo institute of technology," *IEEE Robotics AND Automation Magazine*, vol. 16, no. 2, pp. 104–114, 2009.

[25] K. Fujita M, "Development of an autonomous quadruped robot for robot entertainment."

[26] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, F. Kanehiro, K. Fujiwara, S. Nakaoka, and H. Hirukawa, "A biped pattern generation allowing immediate modification of foot placement in real-time," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 581–586.

[27] N. G. Tsagarakis, G. Metta, G. Sandini, D. Vernon, R. Beira, F. Becchi, L. Righetti, J. Santos-Victor, A. J. Ijspeert, M. C. Carrozza, and D. G. Caldwell, "icub: the design and realization of an open humanoid platform for cognitive and neuroscience research," *Advanced Robotics*, vol. 21, no. 10, pp. 1151–1175, 2007. [Online]. Available: https://doi.org/10.1163/156855307781389419

[28] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quadruped robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 822–10 825, 2008, 17th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016407020

[29] T. Takenaka, T. Matsumoto, and T. Yoshiike, "Real time motion generation and control for biped robot -1st report: Walking gait pattern generation-," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 1084–1091.

[30] D. Gouaillier, C. Collette, and C. Kilner, "Omni-directional closed-loop walk for nao," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 448–454.

[31] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq, a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 225, no. 6, pp. 831–849, 2011. [Online]. Available: https://doi.org/10.1177/0959651811402275

[32] M. Hutter, C. Gehring, A. Lauber, F. Gunther, C. D. Bellicoso, V. Tsounis, P. Fankhauser, R. Diethelm, S. Bachmann, M. Bloesch, H. Kolvenbach, M. Bjelonic, L. Isler, and K. Meyer, "Anymal - toward legged robots for harsh environments," *Advanced Robotics*, vol. 31, no. 17, pp. 918–931, 2017. [Online]. Available: https://doi.org/10.1080/01691864.2017.1378591

[33] "Cassie by agility robotics," http://www.agilityrobotics.com/robots/, accessed: 08-12-2022.

[34] "Atlas by boston dynamics inc," https://www.bostondynamics.com/atlas, accessed: 08-12-2022.

[35] T. Gurriet, S. Finet, G. Boeris, A. Duburcq, A. Hereid, O. Harib, M. Masselin, J. Grizzle, and A. D. Ames, "Towards restoring locomotion for paraplegics: Realizing dynamically stable walking on exoskeletons," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2804–2811.

[36] D. Singh and A. Verma, "Replicating the natural limb structure of a mammal-type quadruped on a highly compact 12-dof robot to deduce a relation between limb structure and canine gait efficiency," in *Recent Trends in Design, Materials and Manufacturing*, M. K. Singh and R. K. Gautam, Eds. Singapore: Springer Nature Singapore, 2022, pp. 409–421.

[37] P. Biswal and P. K. Mohanty, "Development of quadruped walking robots: A review," *Ain Shams Engineering Journal*, vol. 12, no. 2, pp. 2017–2031, 2021.

[38] T. Bretl and S. Lall, "Testing static equilibrium for legged robots," *IEEE Transactions on Robotics*, vol. 24, no. 4, pp. 794–807, 2008.

[39] P.-B. Wieber, "On the stability of walking systems," in *Proceedings of the international workshop on humanoid and human friendly robotics*, 2002.

[40] P. Zaytsev, S. J. Hasaneini, and A. Ruina, "Two steps is enough: No need to plan far ahead for walking balance," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6295–6300.

[41] "Boston dynamics spot robot dog goes on sale," https://spectrum.ieee.org/boston-dynamics-spot-robot-dog-goes-on-sale, accessed: 09-12-2022.

[42] "Add vision to robots, see the difference," https://www.controldesign.com/sensing/sensors/article/11334756/add-vision-to-robots-see-the-difference, accessed: 10-12-2022.

[43] R. Yang, M. Zhang, N. Hansen, H. Xu, and X. Wang, "Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers," *arXiv preprint arXiv:2107.03996*, 2021.

[44] "Petoi bittle user manuals," https://bittle.petoi.com/, accessed: 04-01-2023.

[45] "Welcome to petoi doc center," https://docs.petoi.com/, accessed: 04-01-2023.

[46] "Esp32," https://www.espressif.com/en/products/socs/esp32, accessed: 15-01-2023.

[47] "Wavego, 12-dof bionic dog-like robot, open source for esp32 and pi4b, facial recognition, color tracking, motion detection," https://www.waveshare.com/product/robotics/dog-like-robots/wavego.htm, accessed: 15-01-2023.

[48] "Freenove robot dog kit for raspberry pi," https://github.com/Freenove/Freenove_Robot_Dog_Kit_for_Raspberry_Pi, accessed: 15-01-2023.

[49] O. Theobald, "Machine learning for absolute beginners," 2017.

[50] "What is computer vision," https://www.ibm.com/topics/computer-vision, accessed: 10-12-2022.

[51] J. F. Peters, *Foundations of Computer Vision*.

[52] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[53] P. Viola and M. Jones, "Robust real-time face detection," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 2, 2001, pp. 747–747.

[54] "The neural approach to pattern recognition," https://ubiquity.acm.org/article.cfm?id=985625, accessed: 12-12-2022.

[55] K.-L. Du and M. N. S. Swamy, *Neural Networks and Statistical Learning*. London: Springer London, 2014.

[56] K. Zilles, *The human nervous system*, New York, 1990.

[57] D. O. Hebb, *The organization of behavior*, New York, 1949.

[58] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65 6, pp. 386–408, 1958.

[59] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer, 2018.

[60] J. F. Wiley, *R deep learning essentials*. Packt Publishing, 2016.

[61] J. Patterson and A. Gibson, *Deep learning: A practitioner's approach*. " O'Reilly Media, Inc.", 2017.

[62] O. A. Montesinos Lopez, A. Montesinos Lopez, and J. Crossa, *Fundamentals of Artificial Neural Networks and Deep Learning*. Cham: Springer International Publishing, 2022, pp. 379–425. [Online]. Available: https://doi.org/10.1007/978-3-030-89010-0_10

[63] F. Chollet and J. Allaire, "Deep learning with r. manning publications, manning early access program," 2017.

[64] A. Amini and A. Soleimany, "Introduction to deep learning," 2022, mIT 6.S191. [Online]. Available: http://introtodeeplearning.com/

[65] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," pp. 436–444, 5 2015.

[66] B. Lantz, *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd, 2019.

[67] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[68] "Understanding confusion matrix," https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62, accessed: 10-01-2023.

[69] "Confusion matrix for your multi-class machine learning model," https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826, accessed: 15-01-2023.

[70] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[71] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[72] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang *et al.*, "Large scale distributed deep networks," *Advances in neural information processing systems*, vol. 25, 2012.

[73] A. Amini and A. Soleimany, "Deep computer vision," 2022, mIT 6.S191. [Online]. Available: http://introtodeeplearning.com/

[74] "Convolutional neural networks (cnns / convnets)," https://cs231n.github.io/convolutional-networks/, accessed: 19-12-2022.

[75] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[76] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2017.

[77] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[78] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[80] "Ai scholar: Researchers find a way to reduce cnn training costs by 31," https://medium.com, accessed: 13-01-2023.

[81] A. B. M. Adamski, *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Gora Press, 2006.

[82] A. N. P. Asahar, S. Devidas, *Sequential Logic Synthesis*. Kluwer Academic Publishers, 1992.

[83] M. Ben-Ari and F. Mondada, *Finite State Machines*. Cham: Springer International Publishing, 2018, pp. 55–61. [Online]. Available: https://doi.org/10.1007/978-3-319-62533-1_4

[84] R. Czerwinski, *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*, 1st ed., ser. Lecture Notes in Electrical Engineering, 231. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[85] A. Barkalov, L. Titarenko, M. Kolopienczyk, K. Mielcarek, and G. Bazydlo, *Background of Finite State Machines and Programmable Logic*. Cham: Springer International Publishing, 2016, pp. 1–31. [Online]. Available: https://doi.org/10.1007/978-3-319-24202-6_1

[86] G. D. Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[87] "Introduccion a la automatizacion industrial," https://bookdown.org/alberto_brunete/intro_automatica/, accessed: 02-01-2023.

[88] "Maquinas de estado finito," https://www.profesionalreview.com/2022/11/26/maquinas-de-estado-finito-que-son-para-que-sirven/, accessed: 02-01-2023.

[89] M. J. E. Leon, "Estrategias de planificacion de movimientos para encontrar un objeto en ambientes tridimensionales con un robot manipulador movil," 2012, benemerita Universidad Autonoma de Puebla.

[90] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[91] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A computationally efficient solution to the simultaneous localisation and map building (slam) problem," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 1009–1014 vol.2.

[92] G. Kahn, P. Abbeel, and S. Levine, "Badgr: An autonomous self-supervised learning-based navigation system," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1312–1319, 2021.

[93] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi, "Visual semantic navigation using scene priors," *arXiv preprint arXiv:1810.06543*, 2018.

[94] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2616–2625.

[95] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[96] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[97] S. Brahmbhatt and J. Hays, "Deepnav: Learning to navigate large cities," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5193–5202.

[98] "Vgg16 and vgg19," https://keras.io/api/applications/vgg/, accessed: 05-01-2023.

[99] "Python opencv cv2 resize image," https://pythonexamples.org/python-opencv-cv2-resize-image/, accessed: 05-01-2023.

[100] "vgg16 weights," https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5, accessed: 05-01-2023.

[101] J. S. Matthis, J. L. Yates, and M. M. Hayhoe, "Gaze and the control of foot placement when walking in natural terrain," *Current Biology*, vol. 28, no. 8, pp. 1224–1233, 2018.

[102] "Como habilitar la interfaz para camara en raspberry pi," https://blog.330ohms.com/2020/07/02/como-habilitar-la-interfaz-para-camara-en-raspberry-pi/, accessed: 04-01-2023.

[103] "picamera," https://picamera.readthedocs.io/en/release-1.13/index.html, accessed: 04-01-2023.

[104] "Raspberry pi serial port as an interface," https://docs.petoi.com/api/raspberry-pi-serial-port-as-an-interface, accessed: 04-01-2023.