Universidad del País Vasco Euskal Herriko Unibertsitatea
VITORIA-GASTEIZKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE VITORIA-GASTEIZ

-

GRADO EN INGENIERÍA INFORMÁTICA EN GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE ALGORITMOS DE NAVEGACIÓN AUTÓNOMA MEDIANTE REDES NEURONALES CONVOLUCIONALES

Alumno/Alumna: Romero Porretti Domenico Michele

Director/Directora (1): Zulueta Guerrero Ekaitz

Curso: 2021-2022

Fecha: ‹30,06,2022›

Resumen

El objetivo de este trabajo es mediante un estudio previo y exhaustivo, diseñar e implementar un sistema que permita la conducción autónoma aplicada a un robot, mediante el uso de redes neuronales convolucionales. Para ello se ha realizado el estudio de todo aquello que engloba las redes neuronales convolucionales, desde la base que es la inteligencia artificial, pasando por los conceptos más básicos de redes neuronales artificiales para por así tener los conocimientos básicos para abordar las redes neuronales convolucionales entendiendo su funcionamiento. Además, se ha desarrollado el código necesario en Matlab que permita tener una conducción autónoma en base a un patrón.

Abstract

The aim of this work is to design and implement a system that allows autonomous driving applied to a robot through the use of convolutional neural networks. To this end, a study of everything that encompasses convolutional neural networks has been carried out, from the basis, which is artificial intelligence, through the most basic concepts of artificial neural networks in order to have the basic knowledge to approach convolutional neural networks and understand how they work. In addition, the necessary code has been developed in Matlab to enable autonomous driving based on a pattern.

AGRADECIMIENTOS

Quisiera agradecer primero a mi tutor, Ekaitz Zulueta por el apoyo mostrado en ciertos momentos para la realización del trabajo. También a los compañeros de departamento, a mi familia, sobre todo a mi padre Miguel por estar siempre detrás de mí, y una triple mención especial a mi madre Giovanna, lamentablemente fallecida y a mis dos abuelas María y Concetta ambas fallecidas mientras estaba en la universidad, va por vosotras.

Muchas gracias a todos.

Indice

Índice de Figuras

CAPITULO 1

INTRODUCCIÓN Y OBJETIVOS

1.1. Introducción

En este capítulo se va a explicar sobre qué objetivos se tiene planeado alcanzar con este trabajo y explicar los capítulos a los que se ha dividido el trabajo para una mejor comprensión.

La motivación principal para realizar este trabajo ha sido el incremento exponencial que ha tenido la importancia de la inteligencia artificial en su conjunto sobre nuestra vida diaria, que van desde conducir un vehículo, hasta la búsqueda de unas vacaciones ideales por el buscador preferido de internet, por lo que además con el paso del tiempo su uso está siendo ya generalizado en todos los campos existentes.

Previamente realicé prácticas de empresa en el sector de la ciberseguridad en vehículos, por lo que tuve la oportunidad de conocer todo lo que engloba a los sistemas de asistencia a la conducción-*ADAS*- como pueden ser el control de mantenimiento del carril, o a las investigaciones sobre el coche autónomo, por lo que generó la duda de cómo funcionan estos vehículos o que tecnología está detrás, de entre las cuales una de ellas son las redes neuronales convolucionales.

Actualmente por ejemplo en el caso de los vehículos autónomos las redes neuronales convolucionales tienen un papel importante a la hora de por ejemplo detectar una persona u objeto que se cruce en medio de la carreta, o que, en función de unas señales de tráfico, el vehículo las respete sin saltárselas son cada vez más una realidad.

Hoy en día, hay una serie de niveles de autonomía en los vehículos [1], como podemos ver en la figura 1 que van desde el nivel 0 con

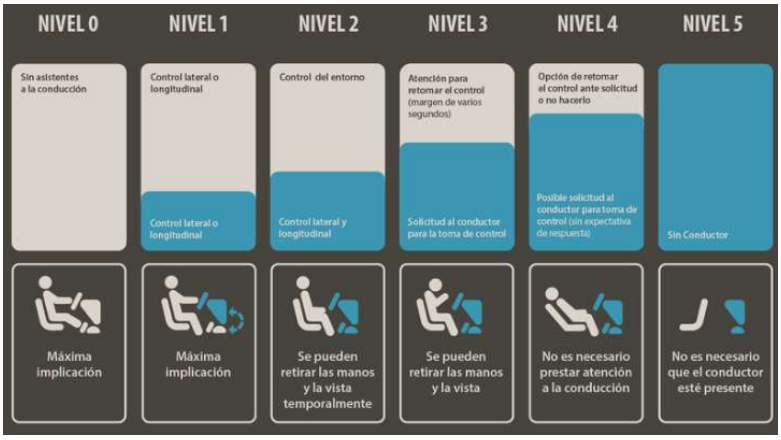


Figura 1.1: Niveles de autonomía de vehículos

menor autonomía hasta el nivel 2, en donde están la mayoría de los vehículos actualmente, el nivel 3 en donde algunas marcas se encuentran ya finalizando sus

investigaciones o bien el nivel 4 donde algunas pocas ya empiezan a dar algunos pasos importantes.

Las redes neuronales convolucionales, por lo tanto, tienen un enorme futuro por delante y sabiendo la enorme complejidad de lo que sería realizar investigaciones a nivel de vehículos, este trabajo se va a centrar en un vehículo autónomo guiado, basándonos en una serie de conceptos teóricos y prácticos que se detallarán en este documento

1.2. Objetivos

Los objetivos que persiguen el trabajo son los siguientes:

- Conocer más sobre las redes neuronales convolucionales, sobre todo a nivel histórico ya que es importante para el desarrollo de las mismas conocer sus orígenes.

- Entender el proceso por el que una imagen es captada y digitalizada, pasando por las diversas capas por las cuales una red neuronal convolucional está formada, hasta la predicción final.

- Documentar y entender las arquitecturas más importantes a nivel de redes neuronales que existen en la actualidad

- Desarrollar mediante una herramienta software un método para poder implementar una red neuronal convolucional que permita tomar decisiones sencillas como pueden ser giros de izquierda a derecha sin que ningún patrón intervenga.

- En caso de no poder desarrollar correctamente la idea descrita anteriormente, buscar una alternativa dentro de las propias redes neuronales para poder alcanzar el objetivo de manera justificada.

1.3. Capítulos

Este trabajo ha sido dividido en diferentes capítulos para tratar de facilitar la comprensión. Dichos capítulos son los siguientes:

1. Introducción y Objetivos

En este apartado se van dar una pequeña introducción relacionada con la motivación para realizar este trabajo. Además, se detallarán una serie de objetivos que se tendrán en cuenta como metas a alcanzar de cara a la finalización del mismo

2. Estado del arte

En este capítulo se va a detallar como se encuentra el mundo de los vehículos autónomos guiados a nivel tecnológico, que es el tipo de vehículo que vamos a basarnos para hacer este trabajo, estudiando los elementos que hay que tener en cuenta en función de su diseño, así como los tipos de vehículos que se encuentran actualmente en servicio.
Será también importante detallar para qué tipo de tareas se les suele encomendar.

3. Inteligencia Artificial

En este capítulo se va a detallar todo lo relacionado con la inteligencia artificial, que van desde los inicios de la computación en el que analizaremos su historia temprana, hasta el análisis de dos términos muy usados en la actualidad que son el *Machine Learning y el Deep Learning,* la base de las redes neuronales artificiales.

4. Redes Neuronales Artificiales

En este capítulo se estudiará, las redes neuronales artificiales, por el cual se revisará sus orígenes, basados en la biología y posteriormente analizando, los tres modelos de redes neuronales más conocidos que son el perceptrón simple, la adalina y el perceptrón multicapa.

También repasaremos los componentes más importantes que forman parte de una red neuronal artificial.

5. Redes Neuronales Convolucionales

En esta sección se hablará sobre las redes neuronales convolucionales, poniendo principal hincapié sobre el concepto y algunas fechas de principal

interés que marcaron el desarrollo de las redes neuronales convolucionales. También se hará un desarrollo de las etapas desde que la imagen es capturada y atraviesa todas las diversas capas tanto de *convolución* como de *pooling*, hasta llegar como entrada de una *fully conected.* También se hará un repaso de las arquitecturas más importantes que hay en la actualidad.

6. **Aplicación Práctica**

   En este capítulo se desarrollará la parte práctica del trabajo mediante un software informático. Se detallará los materiales usados, así como las diversas etapas, desde las pruebas del motor y de la cámara hasta las etapas realizadas con la red neuronal como el entrenamiento y finalmente la implementación el en vehículo

7. **Planificación**

   En este capítulo se va a detallar tanto el tiempo empleado como el presupuesto

8. **Pliego de Condiciones**

   En este capítulo se plasmará el hardware como software usado

9. **Bibliografía**

   En esta sección se detallará todos los libros y enlaces consultados

10. **Anexo**

    Código Utilizado.

CAPITULO 2

ESTADO DEL ARTE

2.1. Introducción

En la actualidad, el sector tecnológico, como el industrial se encuentran en un momento de constantes actualizaciones con la introducción de una manera más exponencial de los sistemas automatizados.

La mayoría de los expertos concuerdan en que nos encontramos en un periodo de tiempo que se denomina Cuarta Revolución Industrial o también llamada Industria 4.0, se corresponde sobre todo al uso de robots y máquinas programables sumado a la inteligencia artificial en la producción, en detrimento del factor trabajo humano [1].

Para entender esto se debe comprender todos los grandes procesos de transformaciones que han tenido lugar a nivel industrial en los años anteriores:

- La primera revolución Industrial entre 1760 y 1830 marcó la transición de la producción manual a la mecanizada

- La segunda durante la segunda mitad del siglo XIX introdujo la electricidad en la industrial pudiendo así comenzar poco a poco a realizar manufactura en masa.

- La tercera revolución se considera que se desarrolla durante sobre todo la segunda mitad del siglo XX con la llamada Revolución Digital.

- Finalmente, la Cuarta Revolución digital como bien se ha definido anteriormente no se define por un conjunto de tecnologías emergentes en sí mismas, sino por la completa digitalización de las cadenas de valor a través de la integración de las nuevas tecnologías

Con esto se quiere llegar a la conclusión que cada vez más, las empresas van a estar aumentando la aplicación de nuevas tecnologías, aumentando también de manera importante la investigación en este sector, sobre todo relacionado con la inteligencia artificial, y por lo tanto no es de extrañar que por ejemplo ya en el año 2018 siete de las diez empresas más cotizadas en la bolsa fueran empresas tecnológicas, todas ellas enormes conocidas por la población tales como Apple, Amazon, Google, Microsoft, entre otras muchas [2].

Las ventajas de la digitalización de las empresas son enormes, enfocados sobre todo a términos de productividad y economía.

No hace falta recordar que la gran ventaja de tener elementos automatizados o robotizados es que no poseen ciertas características humanas tales como el cansancio, enfermedades, etc.

Muchos estudios publicados [3] afirman que la implantación de las ventajas de la industria 4.0 facilitará procesos más rápidos, flexibles y eficientes para producir bienes

de calidad a costes más reducidos incrementando la productividad, tomando como ejemplo la industria alemana que la productividad podría aumentar en un periodo de cinco a diez años entre un 5 y un 8%.

Como se ha comentado antes, ante tales perspectivas muchos países han iniciado ya campañas de digitalización y/o robotización de sus sectores más importantes y el avance que se ha tenido en los últimos años en este aspecto ha sido bastante exponencial.

Hay autores que ya dan datos significativos tales como el avance del sector tecnológico son de tal magnitud que McKinsey [4] estima que el 45% de las tareas existentes en Estados Unidos podrían ser automatizadas hoy mismo, teniendo en cuenta tanto roles como tipos de profesión, como también si es considerada una tarea repetitiva.

Evidentemente no todos los trabajos podrían por lo menos a corto plazo digamos que antes de 2030 poder ser mínimamente parcialmente automatizados, como podemos ver en la siguiente tabla según algunos tipos de empleo teniendo en cuenta el nivel de automatización como su probabilidad [5].

PROFESIÓN	GRUPO DE RIESGO	PROBABILIDAD (%)
Médicos de familia	Bajo	0,42
Compositores, músicos y cantantes	Bajo	4,45
Economistas	Medio	43,00
Analistas Financieros	Medio	46,00
Transportistas	Medio	56,78
Empleados de contabilidad	Alto	97,00
Operadores de telemarketing	Alto	99,00

Profesiones con probabilidad de ser automatizadas [5]

Todo ello hará que a medio y largo plazo muchos tipos de trabajos queden obsoletos por la introducción paulatina de más máquinas, pero a su vez también se va a necesitar nueva mano de obra especializada para poder gestionar y poder desarrollar las nuevas tecnologías que esta cuarta revolución industrial nos pone a nuestra

disposición, como puede ser el caso concreto de los robots AGV (Automated Guided Vehicle), unos modelos de robots que están empleándose ya actualmente en un amplio sector de la industria, sobre todo en el transporte de materiales dentro de una

misma fábrica, algo que McKinsey considera una tarea repetitiva, siendo esta tecnología considerada una las mejores opciones del mercado por ser la más eficiente [6].

2.2. Definición

Un vehículo autónomo guiado -AGV- (***ver Figura 1***) es un sistema de transporte por lo general de mercancías o materiales en la industria, que se mueven de forma autónoma y que poseen una serie de baterías para que puedan trabajar de forma autónoma durante horas.

Figura 2.1: AGV

Estos sistemas se utilizan para aumentar la eficiencia, disminuir los daños a los bienes y reducir los gastos generales. Como bien se sabe su principal característica es la navegación como vehículo autónomo, luego estos vehículos disponen de distintos sistemas de guiado y de sistemas de controles de gestión.

Estos vehículos comenzaron a desarrollarse en los años 50, concretamente en 1954, por parte de la compañía Norteamericana *Barret Electronic*, cuyo propósito era realizar tareas de remolque en una empresa de alimentos, guiado por un cable ubicado en el suelo. Poco a poco y sobre todo en los años 70 gracias a los avances en materia de la electrónica poco a poco han permitido funcionar sin una persona esté a bordo.

2.3. Diseño

Antes de poder desarrollar en términos generales un vehículo guiado automático, algunos expertos señalan que se deben de tener en cuenta una serie de aspectos a la hora de diseñar uno de estos vehículos de acuerdo con el lugar al que van a tener que desarrollar su función destino [7]:

- Trayectoria: Definición de rutas guiadas fijas, ya sean físicas como virtuales

- Gestión del tráfico: Reglas que puedan evitar conflictos tales como colisiones o bloqueos entre vehículos.

- Ubicación de los puntos de recolección y entregas: Zonas de inspección, recogida de carga y lugares de almacenamiento

- Requerimientos del vehículo: Tamaño, forma, sensores a usar, sistemas…etc.

- Gestión: Son por lo general normas internas de la organización a nivel de los gestores de flotas de vehículos, tales como controlar las rutas, programación de los vehículos o gestión de vehículos inactivos

- Aspectos tecnológicos: Aspectos que, al modelar los AGV, se deben tener en cuenta ya que son características específicas de los sistemas automatizados

2.4. Tipos de Vehículos

En la actualidad hay una gran cantidad de tipos en función del sistema utilizado, de entre los cuales los más importantes son [8]:

- Filoguiado: Es un tipo de vehículo guiado por un hilo que está situado bajo el suelo. Es uno de los métodos de guiado más sencillo pero que a su vez no es el menos flexible de todos, porque posee poca libertad de movimiento en el sentido de que su ruta depende del trazado del hilo instalado. Se suele utilizar como opción más cómoda las cintas magnéticas.

- Optoguiado: El vehículo se desplaza con la ayuda de una tira de espejo que se extiende por todo el recorrido del vehículo, colocado en los laterales de los caminos, en el suelo o bien en las esquinas. Por medio de un catadióptrico el vehículo puede detectar la guía.

- Visión Artificial: El AGV reconoce mediante visión artificial una tira de espejo catadióptrico, calculando y corrigiendo en cada instante la desviación existente entre el AGV y la ruta. En función de la ruta que tiene programada y la distancia obtenida mediante la visión artificial, el AGV realiza los movimientos necesarios para continuar con la ruta prefijada.

- Guiado Láser: Los vehículos están equipados con un láser que les permite detectar los reflectores instalados en su perímetro, siendo un tipo de sistema muy fiable.

- Mapeado 2D-3D: Es una de las soluciones más innovadoras en la que gracias a ciertos elementos que ya posee el vehículo tales como cámaras, sensores o ultrasonidos se puede generar la capacidad de formar mapas virtualizados en función de la tecnología empleada ya sean en 2D o 3D. Lo que se realiza es una especie de entrenamiento previo para que se mapee el camino que va a recorrer.

CAPITULO 3

INTELIGENCIA ARTIFICIAL

3.1. Introducción

3.1.1. Historia Inicial de la computación

Para poder comprender un poco donde se encuentra y como se ha desarrollado la inteligencia artificial, se recomienda conocer un poco los orígenes de la computación y para ello, hay que trasladarse a la primera mitad del siglo XX, un poco antes del inicio de la Segunda Guerra Mundial, el matemático británico Alan Turing, desarrolla la llamada Máquina de Turing [9], a la que originalmente llamó como una *"Máquina Automática"* en 1936 en la revista "*Proccedings of the London Mathematical Society*" [10] un simple dispositivo que manipula símbolos sobre una tira de cinta de acuerdo con una tabla de reglas.

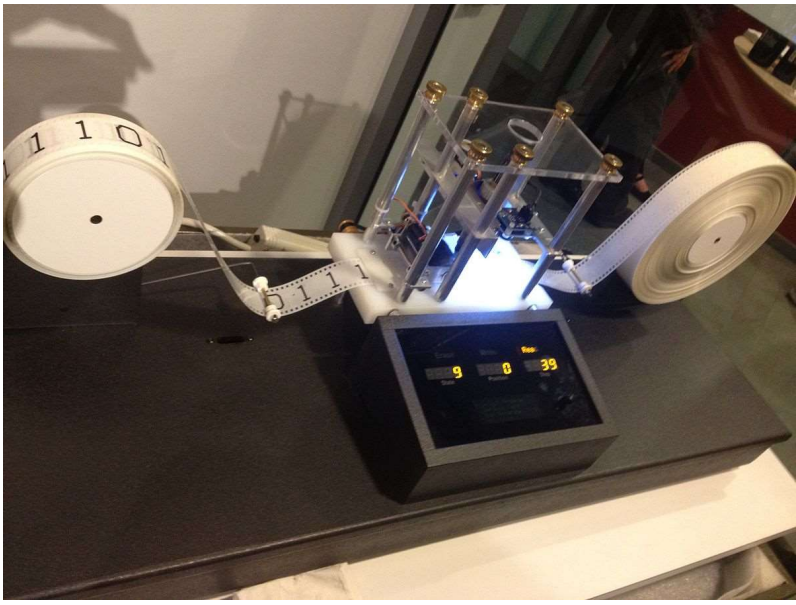


Figura 3.1: Modelo de la Máquina de Turing

Aun siendo un dispositivo bastante simple, el modelo matemático de dicha máquina realizó una tarea inspiradora en la creación de los primeros prototipos de computadoras del siglo XX.

Posteriormente como suele ocurrir en la gran mayoría de los casos, es durante los conflictos bélicos cuando más avances suelen realizarse con el objetivo de tener una superioridad tecnológica frente al

enemigo, que por lo general con el paso del tiempo acaban aplicándose al ámbito civil, como por ejemplo los relojes de pulsera, que se hicieron muy populares, ya que facilitaban a los soldados poder tener un conocimiento y lograr una mayor sincronización para poder tener el conocimiento de cuando realizar un ataque o bien

tener conocimiento de un bombardeo, dejando así las manos libres evitando el uso del reloj de bolsillo que comprometía el uso del fusil.

Durante la Segunda Guerra Mundial [11], es por lo tanto cuando se puede afirmar que se inicia de manera más sería el camino a la construcción de las primeras máquinas inteligentes enfocadas como suele ser siempre en estos casos en el ámbito militar, sobre todo en el sector defensivo con el diseño de ordenadores analógicos ideados para controlar cañones antiaéreos o bien para la navegación.

Algunos investigadores empezaron a observar que había una serie de nexos bastante serios entre el funcionamiento de ciertos dispositivos de control y los sistemas reguladores de los seres vivos.

Por lo tanto, ante la combinación de todos los avances realizados durante la Segunda Guerra Mundial en materia de electrónica, la investigación sobre el sistema nervioso de los seres vivos combinado con las teorías sobre realimentación, hicieron posible desarrollar máquinas que fueran capaces de responder y aprender como los seres vivos, siendo Norbert Wiener[12] la primera persona que utiliza el término cibernética para designar este estudio unificado del control y de la comunicación en los animales y máquinas, con la publicación del su *libro "Cybernetics or Control and Communication in the Animal and the Machine".*

En paralelo al desarrollo de la computación analógica y la cibernética, se inicia también el desarrollo de los ordenadores digitales, como los conocemos conocido también como computación algorítmica, siendo sistemas basados en la separación de la estructura y función (hardware y software).

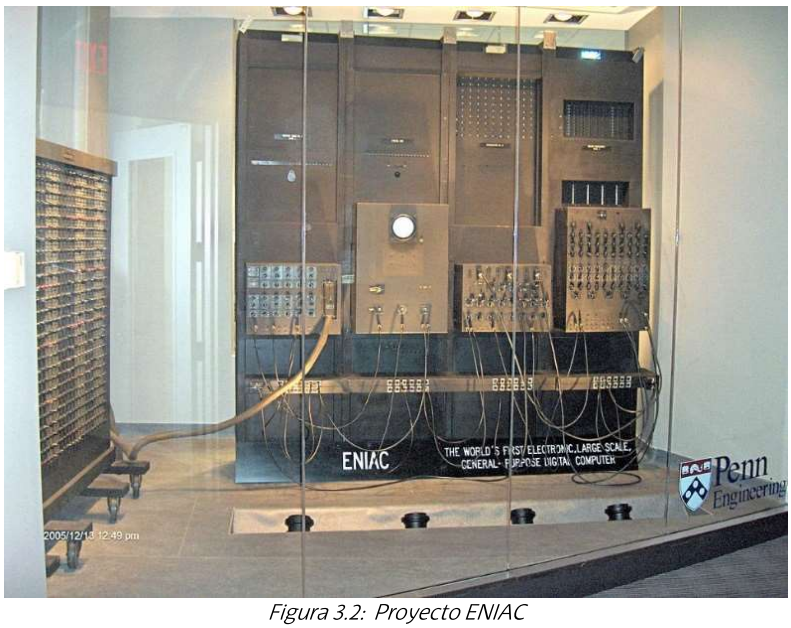


Figura 3.2: Proyecto ENIAC

El primer gran paso en la tendencia de la computación algorítmica lo dio Turing con su máquina descrita al principio de esta introducción, y posteriormente se da el segundo

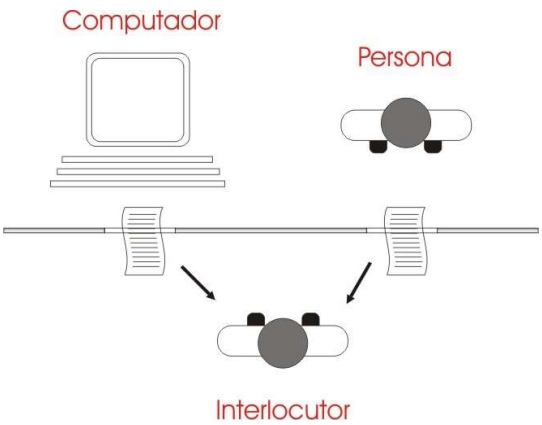
paso importante con el matemático John Von Neumann, por el cual en plena Segunda Guerra Mundial, visita en 1944 la construcción de la ENIAC[13]-"*Electronic Numerical Integrator And Computer*"- un ordenador electrónico diseñado para trabajar en el ámbito militar, por el cual Von Neumann durante su visita comenzó a pensar en cómo proporcionar las instrucciones a seguir, sin necesidad de que el hardware interviniese[14], por lo que Von Neumann, llegó a la conclusión que la computadora tuviera una memoria central, en la cual se pueda almacenar todas las instrucciones necesarias para resolver el problema planteado, reproduciéndose así de forma lógica esas instrucciones naciendo así el concepto de software.

Gracias a su eficacia, versatilidad y eficacia, éste ha sido el enfoque dominante en las últimas décadas de tal forma que el binomio lógico booleana-máquina Von Neumann es la base sobre las que se asientan la mayoría de los computadores digitales actuales.

Por lo tanto, se puede afirmar que la computación algorítmica se centra en la resolución de un problema mediante un algoritmo que se codifica en forma de programa y se almacena en memoria, siendo los propios Turing y Von Neumann los que tenían la esperanza de poder incorporar a alguna de estas máquinas, la capacidad de pensar racionalmente bajo el concepto del software.

Terminada la Segunda Guerra Mundial, adentrándonos en la post guerra es cuando se comienza a definir de manera más clara el concepto de inteligencia artificial siendo en 1950 cuando Alan Turing propone el llamado Test de Turing incluido en su ensayo "Computing Machinery and Intelligence" mientras trabajaba en la universidad de Manchester, con el objetivo de demostrar la existencia de una inteligencia artificial [15] como se puede ver en la ***Figura 3.3***, por el cual con un examen se mide la capacidad de una máquina para exhibir un comportamiento inteligente similar al del ser humano.

Figura 3.3: Test de Turing



Sin embargo, no es hasta 1956 cuando John McCarthy de la universidad de Stanford acuña el término inteligencia artificial o IA, para definir los métodos algorítmicos capaces de hacer pensar a los ordenadores, durante una conferencia [16].

Posteriormente en 1965 Marvin Minsky, Newell y Simon habían desarrollado programas de Inteligencia Artificial que demostraban problemas de geometría.

Aunque habiendo importantes avances desde la post guerra hasta finales de la década de los sesenta, los desarrollos hasta ese momento solo eran capaces de resolver aquellos problemas para los que se habían construido, es decir todavía no eran capaces de desarrollar programas plenamente capaces de resolver problemas de múltiples campos, pero para los investigadores los resultados eran enormemente satisfactorios, llegándose a pensar en poder construir en diez años una máquina realmente inteligente.

En 1969, Marvin Minsky y Seymour Papert publican *"Perceptrons"* [17], generando el también llamado "Invierno de la inteligencia Artificial", en el que mediante una rigurosa investigación lograron demostrar las graves limitaciones de los perceptrones, el modelo neuronal por excelencia de los años sesenta, lo que supuso en una enorme pérdida de confianza en este campo durante alrededor de más de diez años.

3.1.2. Definición de la Inteligencia Artificial

Existen múltiples interpretaciones por parte de diferentes autores que desde su punto de vista interpretaron la Inteligencia Artificial, por lo que no tiene una definición totalmente absoluta, por lo que a continuación vamos a ver una serie de definiciones en función de la percepción de cada autor sobre el tema.

Para John McCarthy "Es la ciencia e ingeniería para construir máquinas inteligentes, especialmente, programadas de computación inteligentes. Así como, lo relativo a la

tarea de usar computadoras para entender la inteligencia humana, pero no limitada a métodos observables biológicamente" [18]

Nils Nilson refiere que "En una definición amplia y tanto circular, que tiene por objeto el estudio del comportamiento inteligente en las máquinas" [19]

Marvin Minsky explica que "Aun cuando todavía no conocemos cómo los cerebros realizan sus habilidades mentales, podemos trabajar hacia el objetivo de hacer máquinas que hagan lo mismo. La Inteligencia Artificial es simplemente el nombre que le dimos a esta investigación" [20].

Shirai & Tsujii la definieron como "El objetivo de la investigación sobre inteligencia artificial es conseguir que un ordenador llegue a realizar importantes funciones de la inteligencia humana" [21].

Russell afirma que “Un sistema inteligente es aquel cuya expectativa de utilidad es la más alta que se puede alcanzar por cualquier otro sistema con las mismas limitaciones computacionales” [22]

En términos generales casi todas las definiciones apuntan a que la Inteligencia Artificial puede dotar a las máquinas de lo que llamamos inteligencia y dependiendo de la naturaleza de cada una de las aproximaciones que se tengan, se podría entender como un proceso centrado en lo que realiza el ser humano o bien en la racionalidad del proceso.

3.2. Conceptos básicos

En la actualidad dentro de lo que conocemos como Inteligencia Artificial tenemos una serie de conceptos bastante importantes que en el siglo XXI están muy de moda que son dos terminologías llamadas *Machine Learning y Deep Learning,* ambos conceptos forman parte como subconjunto del concepto global de inteligencia artificial tal y como podemos ver en la ***Figura 3.4***

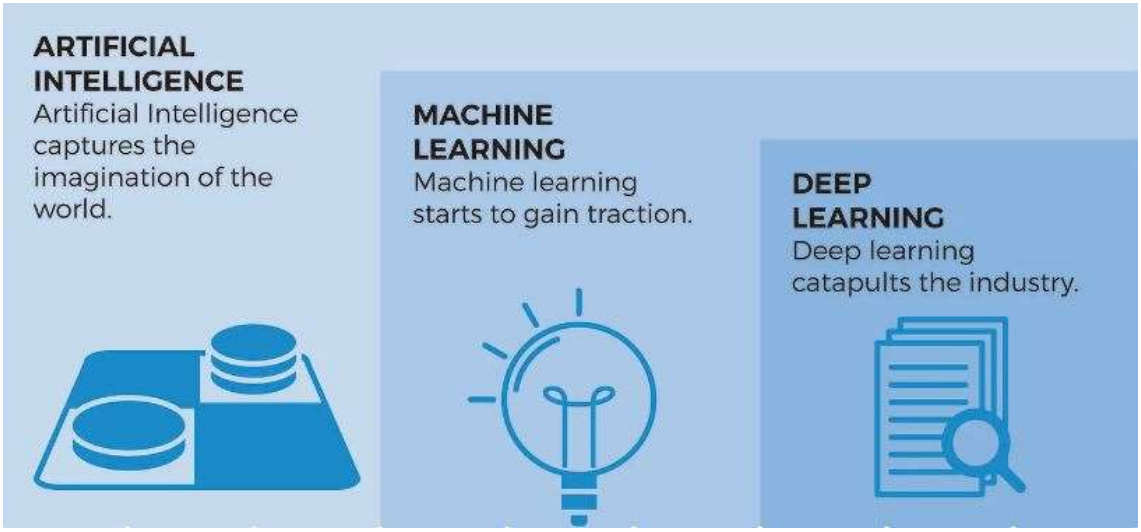


Figura 3.4: Conceptos básicos de la Inteligencia Artificial

Eso no afirma que todos los algoritmos del Machine Learning formen parte del sector de la inteligencia artificial, y el ejemplo más claro es el algoritmo Adaboost

3.2.1. Machine Learning

3.2.1.1. Definición

Llamado también aprendizaje automático, es en términos generales uno de los enfoques principales de la inteligencia artificial, siendo un aspecto de la informática en

el que los ordenadores o las máquinas tienen la capacidad de aprender sin estar programadas para ello [23].

Como ocurre también con la inteligencia artificial, el Machine Learning tiene muchas definiciones según la percepción de una serie de autores.

Uno de los pioneros del Machine Learning, Arthur Samuel definió el Machine Learning como “el campo de estudio que da a las computadoras la habilidad de aprender sobre algo para lo que no han sido explícitamente programadas” [24]. Es interesante añadir que Samuel allá por la década de 1950 diseñó un programa informático relacionado con el juego de las damas, cuyo principal interés era el aprender a descubrir cuales eran las posiciones del tablero buenas y cuales las malas. Para ello dicho programa lo que hacía era jugar miles de veces contra sí mismo hasta obtener la experiencia suficiente para determinar las mejores posiciones

Figura 3.5: Arthur Samuel y el juego de las damas

Tom Mitchell proporciono una definición más formal y ampliamente citada de los algoritmos estudiados en el campo del aprendizaje automático allá por el año 1997, basándose en que un problema de aprendizaje bien planteado se define de la siguiente manera: “Un programa aprende de la experiencia E con respecto a una tarea T y a un rendimiento P, si el rendimiento medido por P en la tarea T mejora con la experiencia E”. Si nos centramos a aplicar esta definición al problema de las damas, la tarea T es jugar al juego de las damas, el rendimiento P se mide con el porcentaje de juegos ganados contra oponentes, y la experiencia E es el juego que realiza el programa a método de entrenamiento contra él mismo [25].

Otras decisiones de Machine Learning están más centradas desde un punto de vista matemático. Pero, para poder englobar dichas perspectivas, el Machine Learning se

define como “un conjunto de técnicas dentro del ámbito de la inteligencia artificial que utiliza métodos estadísticos para la búsqueda de patrones a partir de los cuales se

crean máquinas inteligentes capaces de aprender y tomar decisiones a base de datos empíricos obtenidos de diversas fuentes” [26].

3.2.1.2. Aprendizaje

Los diferentes algoritmos de aprendizaje automático se agrupan en función de la salida de los mismos, por lo que podríamos decir de que se dividen en tres grandes grupos, que veremos a continuación, centrándonos sobre todo en aprendizaje supervisado ya que éste será el que se utilice para la realización de este trabajo.

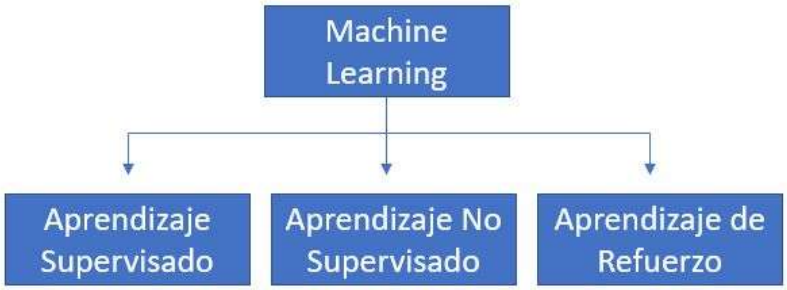


Figura 3.6: Tipos de Aprendizaje

3.2.1.2.1. Aprendizaje Supervisado

En el aprendizaje supervisado, se puede entender como algoritmos que aprenden de los datos introducidos por una persona.

El aprendizaje supervisado resuelve problemas conocidos y utiliza un conjunto de datos etiquetados para entrenar un algoritmo para realizar tareas específicas para

predecir resultados conocidos como por ejemplo hacer un conteo de las personas que hay en una imagen o determinar el color de una imagen.

Un ejemplo de proceso de aprendizaje supervisado podría ser la clasificación de una serie de vehículos a partir de unas imágenes sobre si son aviones de combate o aviones civiles. Para ello, un factor determinante de este tipo de aprendizaje es que debe haber un correcto etiquetado de los datos de entrenamiento para que el algoritmo pueda clasificarlo de manera correcta. El aprendizaje supervisado permite que los algoritmos aprendan de datos históricos de entrenamiento y los apliquen a entradas desconocidas para obtener la salida correcta. Dentro del aprendizaje supervisado, existen dos tipos de aplicaciones:

- Clasificación: La clasificación es el lugar donde se entrena a un algoritmo para clasificar los datos de entrada en variables discretas. Durante el entrenamiento, los algoritmos reciben datos de entrada de entrenamiento con una etiqueta de clasificación. Por ejemplo, determinar si un paciente está enfermo o si un correo electrónico es un spam.

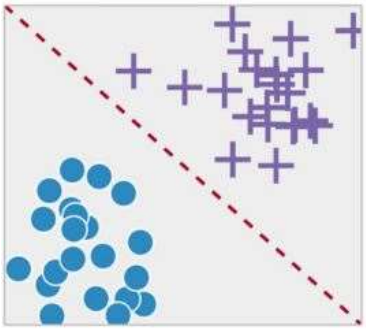


Figura 3.7: Gráfica de clasificación

- Regresión: A diferencia de la clasificación, la regresión es un método de aprendizaje supervisado en el que se entrena a un algoritmo para predecir una salida a partir de un rango continuo de valores posibles. En la regresión, un algoritmo necesita identificar una relación funcional entre los parámetros de entrada y salida. El valor de salida no es discreto como en la clasificación, sino que es una función de los parámetros de entrada. La exactitud de un algoritmo de regresión se calcula en función de la desviación entre la salida prevista y la salida prevista. Un par de ejemplos para el caso de la regresión sería el caso de los precios de una casa al escoger diferentes opciones o la demanda ocupación de un hotel.

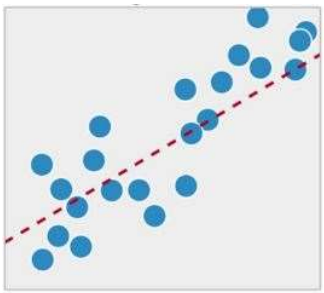


Figura 3.8: Gráfica de Regresión

3.2.1.2.2. Aprendizaje No Supervisado

Para el caso del aprendizaje no supervisado, los algoritmos no utilizan ningún dato etiquetado u organizado previamente para indicar cómo tendría que ser categorizada la nueva información, ya que tienen que encontrar la manera de clasificarlas por ellos

mismos. Esto quiere decir que este en este método, no se requiere intervención humana.

Existen dos tipos de algoritmos no supervisados:

- **Clustering**: Clasifica en grupos los datos de salida.
- **Asociación:** Descubre reglas dentro del conjunto de datos.

3.2.1.2.3. Aprendizaje de Refuerzo

Los algoritmos de este tipo de aprendizaje aprenden de la experiencia. En otras palabras, tenemos que darles un refuerzo

positivo cada vez que aciertan, siendo una comparativa de la vida diaria a la situación en la forma en que se realizan entrenamientos de algunas mascotas como por ejemplo los perros, cuando les damos una recompensa al aprender a sentarse.

3.2.1.3. Aplicaciones del Machine Learning

En los últimos años, se ha convertido en una de las herramientas de vital importancia utilizadas los múltiples campos en los que se requiere extraer información a partir de gran cantidad de datos [27].

Actualmente muchas de las actividades diarias que nos rodean por el cual necesitamos el uso de alguna tecnología están basadas en el Machine Learning, como, por ejemplo:

- **Filtros del Correo Electrónico**: Los filtros de un spam de correo electrónico utilizan este tipo de aprendizaje con el objetivo de poder detectar que mensaje son importantes para el usuario o cuales deben de ser etiquetados como descartables para enviarlos a la papelera marcándolos como correo basura. Estos filtros son utilizados prácticamente por la mayoría de sistemas de correos

  electrónicos tales como por ejemplo Outlook o Gmail, siendo también catalogado como medida de ciberseguridad activa tratando de evitar correos maliciosos.

- **Búsqueda Online:** Los algoritmos usados por los buscadores más usados del mercado para realizar búsquedas online, realizando con esto mejoras en lo que entiende el motor de búsqueda. Cada vez que se ejecuta una búsqueda en cualquier buscador de los que usamos habitualmente, el software observa cómo responde a los resultados, por lo que, si un usuario hace clic en el

resultado superior y permanece en esa página web, se llega a entender que el usuario ha conseguido la información que buscaba por lo que se llega a la conclusión de que la búsqueda fue un éxito. Sin embargo, si se detecta que se hace clic en la segunda página de resultados, o se escribe otra vez en la cadena de búsqueda sin hacer clic en ninguno de los resultados, el motor de búsqueda no ha tenido éxito por lo que el programa puede aprender de ese error para ofrecer un mejor resultado en el futuro .

- **Recomendaciones:** En este caso los algoritmos de Machine Learning de ciertas plataformas de compras online, permiten analizar la actividad de un usuario y compararla con la del resto de usuarios para determinar el tipo de producto que se desearía comprar o que nos gustaría ver, y gracias a estos algoritmos la plataforma puede detectar si un usuario está comprando un producto para él o si estaría comprando un artículo para otra persona

3.2.2. Deep Learning

Llamado también aprendizaje profundo, es un subcampo del “Machine Learning”, siendo este un campo que ha emergido de manera muy importante en los últimos años.

El Aprendizaje Profundo o Deep Learning, está basado en algoritmos de aprendizaje en múltiples niveles de representación y abstracción con el fin de modelar relaciones más complejas entre los datos. Los niveles correspondidos con distintos niveles de conceptos, donde los más altos vienen definidos por los más bajos de forma jerárquica. Esta idea de representaciones sucesivas por capas es lo que da el nombre

de “deep” a este tipo de aprendizaje siendo la profundidad del modelo el número de capas que contiene [28]. En Deep Learning, estas representaciones por capas forman una Red Neuronal, que al fin y al cabo es un conjunto estructurado de neuronas. En el siguiente apartado se entrará en detalle con las redes neuronales y especialmente con un tipo muy específico de ellas que son las Redes Neuronales Convolucionales.

CAPITULO 4

REDES NEURONALES ARTIFICIALES

4.1. Introducción Biológica

Antes de entrar de lleno en el tema de las redes neuronales artificiales, es conveniente hacer un breve repaso para poder saber de dónde proviene el concepto redes neuronales artificiales. Como veremos el embrión de este tema son las redes neuronales biológicas.

La historia de las redes neuronales artificiales comienza a finales del siglo XIX con el científico español Santiago Ramón y Cajal [29], que es el descubridor de la estructura neuronal del sistema nervioso.

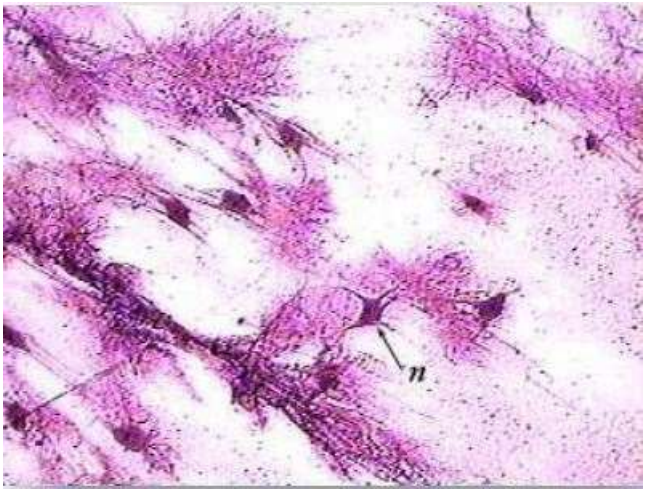


Figura 4.1: Imagen de unas neuronas en el microscopio

A finales del siglo XIX, la teoría reticularista, que sostenía que el sistema nervioso estaba formado por una red continua de fibras nerviosas, era la creencia extendida. Sin embargo, posteriormente Ramón y Cajal demostró que el sistema nervioso en realidad estaba compuesto por una red células individuales, llamadas neuronas ampliamente interconectadas entre sí. Pero no solo observó al microscopio los pequeños espacios vacíos que separaban unas neuronas de otras, sino que también estableció que la información fluye en la neurona desde las dendritas hacia el axón, atravesando el soma que es el punto central de la neuronal.

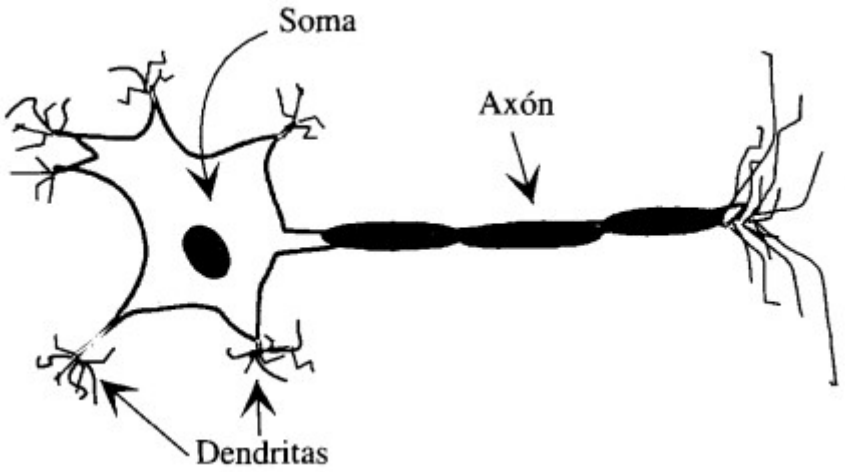


Figura 4.2: Estructura de una neurona biológica

El funcionamiento del sistema es sencillo, las dendrites que tenemos cerca del cuerpo de la neurona son receptores de llamadas, encargados de calcular señales de entrada.

Todas ellas van hasta las neuronas y a partir de ahí se trasmiten a través del axón, justo en ese lugar se lleva a cabo una transmisión química llamada sinapsis que pasa en forma de impulsos eléctrico-químicos con otros dendrites, sin llegar a tocarse físicamente. Actualmente se estima que el sistema nervioso contiene alrededor de cien mil millones de neuronas

Este descubrimiento básico para el desarrollo de las neurociencias en el siglo XX causó una enorme conmoción en su época dada la forma de entender el sistema nervioso, otorgándole así el premio Nobel de medicina a Ramón y Cajal en 1906.

Con el paso del tiempo, gracias al advenimiento de la microscopía electrónica y a la introducción de múltiples nuevas técnicas innovadoras, se ha llegado a profundizar mucho más en el estudio de la neurona [30].

4.2. Estructura de un sistema Neuronal Artificial

Las Redes Neuronales Artificiales imitan la estructura de un hardware del sistema nervioso, con la intención de construir sistemas de procesamiento de la información paralelos, distribuidos y adaptativos, que puedan presentar un cierto comportamiento inteligente.

Los elementos básicos de un sistema neuronal biológico son las neuronas, que se agrupan en conjuntos compuestos por millones de ellas organizadas en capas, constituyendo un sistema con funcionalidad propia. Un conjunto de estos subsistemas da lugar a un sistema global. En la realización de un sistema neuronal artificial puede

establecerse una estructura jerárquica similar. El elemento esencial de partida será la neurona artificial, organizada en capas; varias capas formarán una red neuronal; y por último una red neuronal, junto con interfaces de entrada y salida, más los módulos

convencionales adicionales necesarios, constituirán el sistema global de proceso como podemos observar en la ***Figura 4.3***

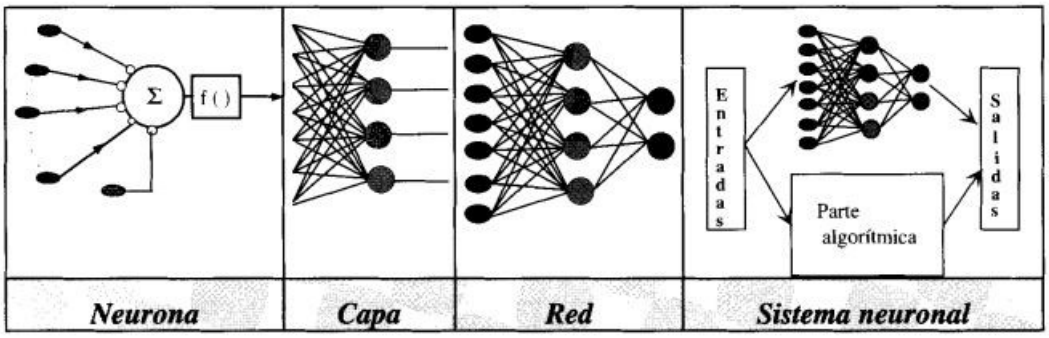


Figura 4.3: Estructura jerárquica de un sistema basado en una Red Neuronal Artificial

Formalmente y desde el punto de vista de una serie investigadores de la Universidad de California llamado grupo PDP (*Parallel Distributed Processing Research Group*), junto con D.E. Rumelhart y J.L McClelland[31][32], un sistema neuronal o conexionista, está compuesto por los siguientes elementos:

- Un grupo de procesadores elementales o neuronas artificiales
- Un patrón de conectividad o arquitectura
- Una dinámica de activaciones
- Una regla o dinámica de aprendizaje
- El entorno donde opera

4.3. Modelo de Neurona Artificial

Se denomina procesador elemental o neurona a un dispositivo simple de cálculo que a partir de un vector de entrada procedente del exterior o de otras neuronas proporciona una única respuesta o salida. Los elementos que forman la neurona de etiqueta *i*, (véase la ***Figura 4.4***) son los siguientes:

- Un conjunto de entradas o valores de entrada, $x_j(t)$

- Los pesos sinápticos de la neurona *i,* w_{ij} , que representan la intensidad de interacción entre cada neurona presináptica *j* y la neurona postsináptica *i*.

- Regla de propagación $\sigma(w_{ij}, x_j(t))$, que proporciona el valor del potencial postsináptico $h_i(t) = \sigma(w_{ij}, x_j(t))$ de la neurona *i* en función de sus pesos y entradas.

- Función de activación $f_i(a_i(t-1), h_i(t))$, que proporciona el estado de la activación actual $a_i(t) = f_i(a_i(t-1), h_i(t))$ de la neurona *i,* en función de su estado anterior $a_i(t-1)$ y de su potencial postsináptico actual. En la *tabla 1* podemos observar una serie de funciones de activaciones más habituales.

- Función de salida $F_i(a_i(t))$, que proporciona la salida actual $y_i(t) = F_i(a_i(t))$ de la neurona *i* en función de su estado de activación

Con todo lo visto anteriormente, la fórmula referente a la neurona *i* puede escribirse de la siguiente manera:

$$y_i(t) = F_i(f_i[a_i(t-1), \sigma(w_{ij}, x_j(t)]) \qquad (1)$$

Este modelo de neurona forma se inspira en la operación de la biológica, en el sentido de integrar una serie de entradas y proporcionar cierta respuesta, que se propaga por el axón.

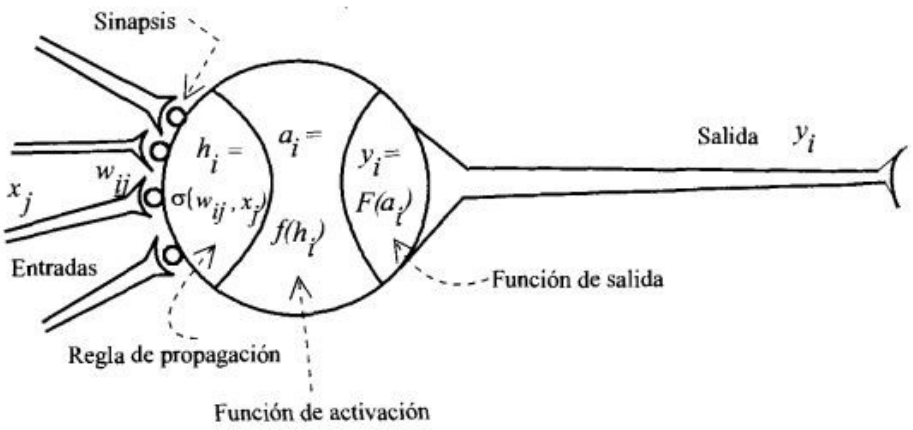


Figura 4.4: Modelo genérico de neurona artificial [31]

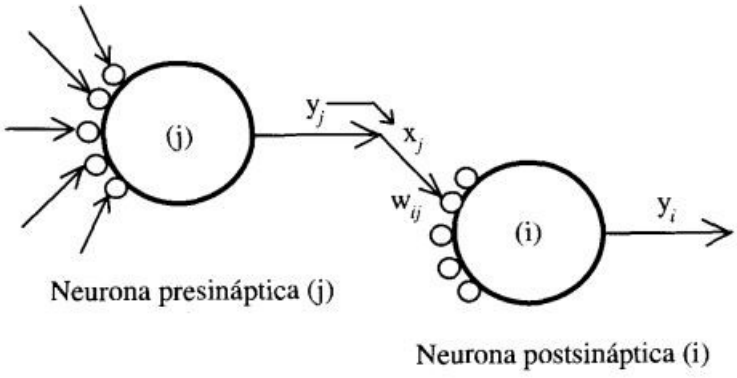


Figura 4.5: Interacción entre una neurona presináptica y otra postsináptica

A continuación, vamos a detallar de manera más clara algunos elementos que forman parte de este modelo, que necesitan una mayor atención.

4.3.1. Regla de Propagación

La regla de propagación permite obtener a partir de las entradas y los pesos, el valor del potencial postsináptico h_i de la neurona

$$h_i(t) = \sigma(w_{ij}, x_j(t)) \qquad (2)$$

La función más habitual es de tipo lineal y se basa en la suma ponderada de las entradas de los pesos sinápticos

$$h_i(t) = \sum_j w_{ij} x_j \qquad (3)$$

que formalmente también puede interpretarse como el producto escalar de los vectores de entrada y los pesos

$$h_i(t) = \sum_j w_{ij} x_j = w_i^T . x \qquad (4)$$

El peso sináptico w_{ij} define en este caso la intensidad de interacción entre la neurona presináptica j y la postsináptica i. Dada una entrada positiva (procedente del sensor o simplemente la salida de otra neurona), si el peso es positivo tenderá a excitar a la neurona postsináptica, si el peso es negativo tenderá a inhibirla. Así se habla de sinapsis excitadoras (de peso positivo) e inhibidoras (de peso negativo).

4.3.2 Función de activación o función de transferencia

La función de activación o de transferencia proporciona el estado de activación actual $a_i(t)$ a partir del potencial postsináptico $h_i(t)$ y del propio estado de activación anterior $a_i(t-1)$

$$a_i(t) = f_i(a_i(t-1), h_i(t)) \qquad (5)$$

Sin embargo, en muchos modelos de ANS se considera que el estado actual de la neurona no depende de su estado anterior, sino únicamente del actual

$$a_i(t) = f_i(h_i(t)) \qquad (6)$$

La función de activación *f(.)* en la mayor parte de los modelos es monótona creciente y continua, como suele ser habitual en las neuronas biológicas. Las funciones de activación más habituales se pueden observar en la *Figura 4.6.*

La más sencilla de todas es la función identidad, empleada por ejemplo en la Adalina. Otro caso también muy simple es la función escalón, empleada en el Perceptrón Simple.

4.3.3 Función de salida

Esta función lo que nos proporciona es la salida global de la neurona $y_i(t)$ en función de su estado de activación actual $a_i(t)$. Muy frecuentemente la función de salida es simplemente la identidad F(x)=x de modo que el estado de activación de la neurona se considera como la propia salida

$$y_i(t) = F_i\big(a_i(t)\big) = a_i(t) \qquad (7)$$

Esto ocurre en los modelos más comunes como el MLP o la Adalina. La función de salida puede ser también de tipo escalón, lo que supone que la neurona no se dispare hasta que la activación supere un cierto umbral.

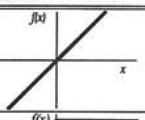
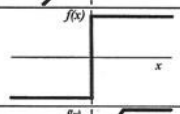
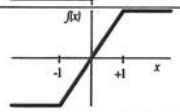
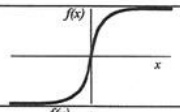
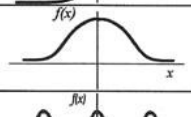
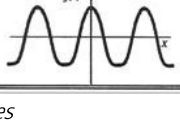
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	{-1, +1} {0, +1}	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	[-1, +1]	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0, +1] [-1, +1]	
Gaussiana	$y = Ae^{-Bx^2}$	[0,+1]	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	[-1,+1]	

Figura 4.6: Funciones de activación más habituales

4.4. Modelo estándar de neurona artificial

En el caso anterior se ha explicado un modelo más general, que en la práctica se puede simplificar de manera más simple que se denomina neurona estándar, que constituye un caso particular del modelo anterior, considerando que la regla de propagación es la suma ponderada y la función de salida es la identidad. Por lo tanto, la forma de este modelo es:

- Un conjunto de entradas $x_j(t)$ y pesos sinápticos w_{ij}.
- Una regla de propagación $h_i(t) = \sigma\left(w_{ij}, x_j(t)\right); h_i(t) = \sum w_{ij}x_j$ es la más común
- Una función de activación $y_i(t) = f_i(h_i(t))$, que representa simultáneamente la salida de la neurona y su estado de activación

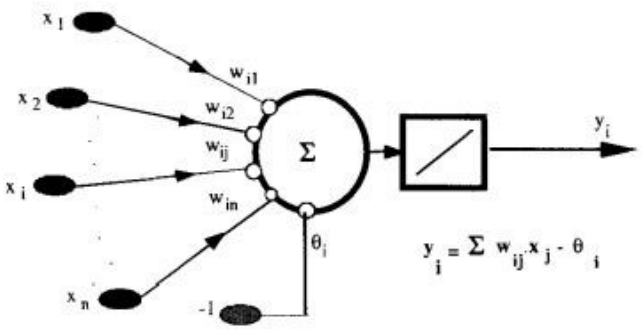


Figura 4.7: Modelo de neurona estándar

Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional ϑ_i, que denominaremos umbral o *thereshold*, que se resta del potencial postsináptico, por lo que el argumento de la función de activación queda

$$\sum_j w_{ij}x_j - \theta_i \qquad (5)$$

Lo que representa añadir un grado de libertad adicional a la neurona. En conclusión, el modelo de neuronas que denominaremos estándar queda:

$$y_i(t) = f_i(\sum_j w_{ij}x_j - \theta_i) \qquad (6)$$

Ahora bien, si hacemos que los índices i y j comiencen en, podemos definir $w_{i0} \equiv \vartheta_i$ y $x_0 \equiv -1$ (constante), con lo que el potencial postsináptico se obtiene realizando la suma desde *j=0*

$$y_i(t) = f_i(\sum_{j=0}^{n} w_{ij}x_j) \qquad (7)$$

4.5. Tipos de arquitecturas

Se denomina arquitectura a la topología, estructura o patrón conexionado de una red neuronal. En una red neuronal artificial los nodos se conectan por medio de la sinapsis, determinado esta conexión el comportamiento de la red.

Las conexiones sinápticas son del tipo direccional, es decir que solo puede propagarse la información en un solo sentido como podemos ver en *la* ***Figura 4.8.***

En general las neuronas se suelen agrupar en unidades estructurales llamadas capas, y a su vez las neuronas de una capa pueden agruparse formando grupos neuronales o *"clusters"*.

Dentro de un grupo de una capa, si no existe este tipo de agrupación, las neuronas suelen ser del mismo tipo. Finalmente, el conjunto de una o más capas constituye una red neuronal.

Se distinguen tres tipos de capas que son las capas de entrada, salida y las capas ocultas como se ve en *la* ***Figura 4.8***, que serán descritas a continuación:

- Una capa de entrada está compuesta por neuronas que reciben datos o señales procedentes del entorno como pueden ser por ejemplo proporcionado por sensores.

- Una capa de salida es aquella cuyas neuronas proporcionan la respuesta de la red neuronal

- Una capa oculta es la que no tiene conexión directa con el entorno.

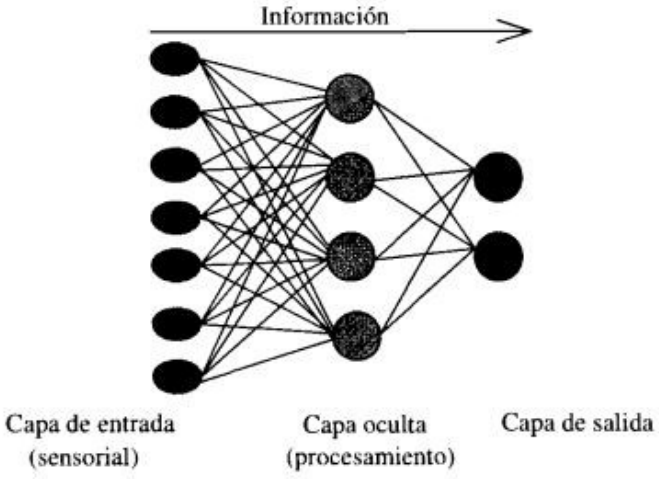


Figura 4.8 : Arquitectura unidireccional de tres capas, de entrada, oculta y de salida

Las conexiones entre las neuronas pueden ser excitadoras o inhibidoras:

- Un peso sináptico negativo define una conexión inhibitoria
- Un peso sináptico positivo determina una conexión excitadora

Habitualmente no se suele definir una conexión como de un tipo o de otro, sino que por medio del aprendizaje se obtiene un valor para el peso que incluye signo y magnitud

Por otra parte, se puede distinguir entre conexiones intra-capa e inter-capa. Las conexiones intra-capa, también denominadas laterales, tienen lugar entre las neuronas pertenecientes a una

misma capa, mientras que las conexiones inter-capa se producen entre las neuronas de diferentes capas. Existen además conexiones realimentadas, que tienen un sentido contrario al de entrada-salida. En algunos casos puede existir realimentaci6n incluso de una neurona consigo misma.

Atendiendo a distintos conceptos, pueden establecerse diferentes tipos de arquitecturas neuronales como podemos ver en la ***Figura 4.9,*** en el primer caso tendríamos una red monocapa y realimentada y en el segundo una multicapa unidireccional.

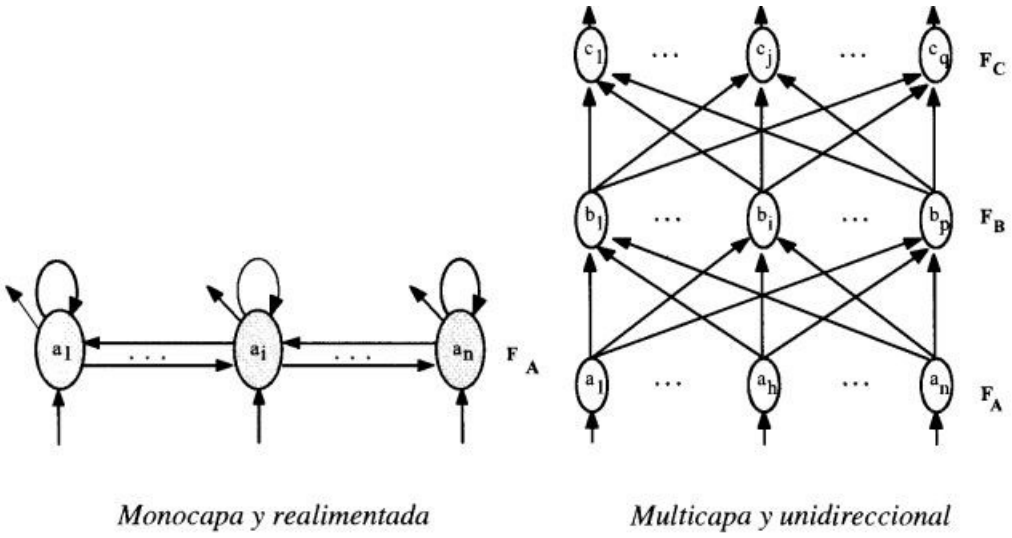


Figura 4.9: Ejemplo de arquitecturas neuronales

Así en relación a su estructura en capas, podemos hablar de redes monocapa y redes multicapa.

Las redes monocapa son aquellas compuestas por una única capa de neuronas. Las redes multicapa son aquellas en cuyas neuronas se organizan en varias capas.

Además, en función del flujo de datos de la red neuronal, se pueden describir dos tipos que son las redes unidireccionales o también llamadas *"feedforward"* y las recurrentes o *"feedback"*. En las redes unidireccionales, toda la información circula en un único sentido, desde las neuronas de entrada hacia la salida. En las redes recurrentes o también llamadas realimentadas la información puede circular entre las capas en cualquier sentido, incluido el de salida-entrada.

4.6. Aprendizaje

En general centrándonos en el tema de las redes neuronales, podemos decir que el aprendizaje es el proceso por el que se produce un ajuste de parámetros libres de la red a partir de un proceso de estimulación por el entorno que rodea a la red. Por lo general el aprendizaje consiste en determinar un conjunto de pesos sinápticos que permita a la red realizar de manera satisfactoria el procesamiento deseado.

Cuando se crea un sistema neuronal, se debe partir de un tipo determinado de neurona, una arquitectura, estableciéndose los pesos sinápticos como nulos o aleatorios.

Ahora bien, para que la red pueda funcionar es necesario entrenarla, lo que forma parte del modo de aprendizaje. El entrenamiento o aprendizaje se puede llevar a cabo a dos niveles. El más convencional es el de modelado de las sinapsis, que consiste en modificar 10s pesos sinápticos siguiendo una cierta regla de aprendizaje, construida

normalmente a partir de la optimización de una función de error o coste, que mide la eficacia actual de la operación la red.

Si se denomina $w_{ij}(t)$ al peso que conecta la neurona presináptica *j* con la postsináptica *i* en la iteración *t*, el algoritmo de aprendizaje, en función de las señales que en el instante *t* llegan, el algoritmo de aprendizaje, en función de las señales que en el instante *t* llegan procedentes del entorno, proporcionará el valor $\Delta w_{ij}(t)$ que da la modificación que se debe incorporar en dicho peso, el cual quedará actualizado de la forma

$$\Delta w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \qquad (8)$$

El proceso de aprendizaje es generalmente iterativo, actualizándose los pesos de la manera anterior, una y otra vez, hasta que la red neuronal alcanza el rendimiento deseado.

En cualquier caso, en un proceso de aprendizaje la información contenida en los datos de entrada queda incorporada en la propia estructura de la red neuronal, la cual almacena la representación de una cierta imagen de su entorno.

4.7. Tipos de Redes Neuronales

Dentro de las redes neuronales, existe una gran variedad de modelos. En el caso de este trabajo, se va a centrar en las redes neuronales supervisadas y unidireccionales.

Dentro de este grupo se explicará el perceptrón simple, la adalina y el perceptrón multicapa.

El popular algoritmo de aprendizaje denominado *"back propagation"* o BP se aplica especialmente al perceptrón multicapa. El perceptrón multicapa con aprendizaje BP es el modelo neuronal más utilizado en las aplicaciones prácticas con alrededor de un 70% de los desarrollos [33].

4.7.1. El Perceptrón Simple

Este modelo de red neuronal fue introducido por Frank Rosenblatt en 1962[34]. La estructura del perceptrón está inspirada en las primeras etapas del procesamiento de los sistemas sensoriales de los animales como la visión, por ejemplo, en los cuales la información va atravesando sucesivas capas de neuronas, que realizan un procesamiento progresivamente de más nivel

El perceptrón simple es un modelo unidireccional, compuesto por dos capas de neuronas, una sensorial o de entradas, y otra de salida (ver ***Figura 4.10***) y la operación de una red de esta clase con n neuronas de entrada y m de salida puede expresarse como:

$$y_i(t) = f\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right), \forall i, 1 \leq i \leq m \qquad (9)$$

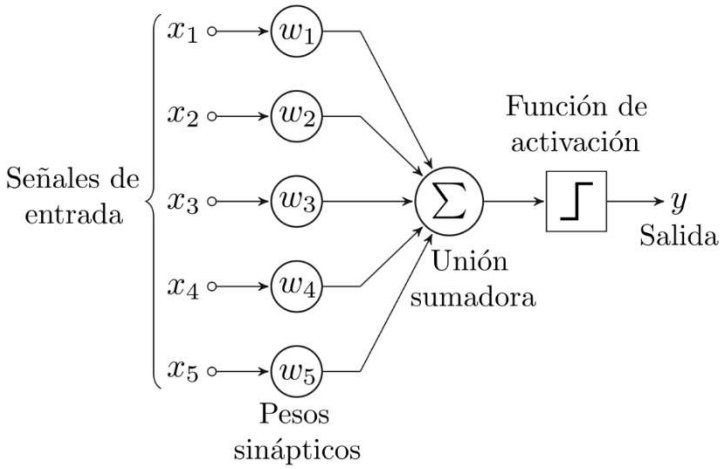


Figura 4.10: Perceptrón simple y función de transferencia

Las neuronas de entrada no realizan ningún cómputo, únicamente envían la información a las neuronas de salida (en el modelo original

estas neuronas de entrada representaban información ya procesada, no datos directamente procedentes del exterior). La función de

activación de las neuronas de la capa de salida es de tipo escalón. Así, la operación de un perceptrón simple puede escribirse

$$y_i(t) = H\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right), \forall i, 1 \leq i \leq m \qquad (10)$$

con H(.) la función de Heaviside o escalón. El perceptrón puede usarse tanto como clasificador, como para la representación de funciones booleanas.

La importancia histórica del perceptrón radica en su carácter de dispositivo entrenable, pues el algoritmo de aprendizaje del modelo introducido por Rosenblatt, y que describiremos más adelante, permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de patrones a partir de un conjunto de ejemplos etiquetados.

Aun así, pese a su gran interés e importancia, el perceptrón presenta serias limitaciones, pues solamente puede representar funciones linealmente separables. Así, aunque pueda aprender automáticamente a representar funciones booleanas o resolver con éxito muchos problemas de clasificación, en otras fallará estrepitosamente.

Cabe recordar que justo aquí es cuando recordamos lo que se menciona en el *apartado 3.1.1,* por el cual Minsky y Papert [35], estudiaron en profundidad el perceptrón, y en 1969 publicaron un exhaustivo trabajo en el que se subrayaba sus limitaciones, lo que resultó decisivo para que muchos de los recursos que se estaban invirtiendo en redes neuronales se desviasen hacia otros campos más prometedores entonces, como era en la época el de la inteligencia artificial.

4.7.2. La Adalina

Otro de los modelos de redes neuronales más conocidos es la Adalina o *"Adaline"* que fue introducida por Bernard Widrow en 1959[36][37], y cuyo nombre proviene de "*ADAptive Linear Neuron".*

Este modelo utiliza una neurona similar a la del perceptrón, pero de respuesta lineal (*ver* ***Figura 4.11***) cuyas entradas pueden ser continuas. Por otra parte, a diferencia del nodo del asociador lineal, el

de la adalina incorpora un parámetro adicional denominado umbral, aunque debe tenerse en cuenta que no se trata de un umbral de disparo como el del perceptrón, sino de un parámetro que proporciona un grado de libertad adicional. De este modo, la ecuación de la adalina queda:

$$y_i(t) = f\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right), \forall i, 1 \leq i \leq m \qquad (11)$$

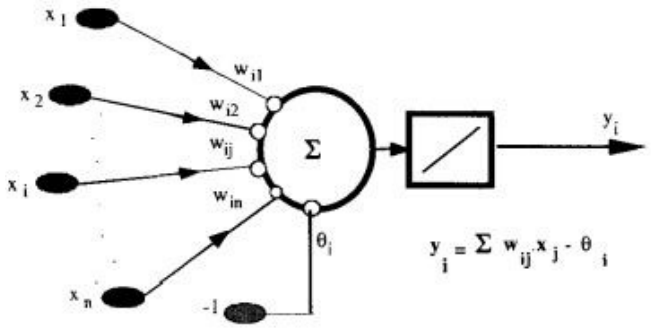


Figura 4.11: La Adalina

No obstante, la diferencia más importante con el perceptrón y con el asociador lineal reside en la regla de aprendizaje que implementa. En la adalina se utiliza la regla de Widrow-Hoff, también conocida como regla LMS (Least Mean Squares, mínimos

cuadrados), que conduce a actualizaciones de tipo continuo, siendo la actualización de los pesos proporcional al error que la neurona comete.

Adaline es un modelo muy conocido y ampliamente utilizado, aunque en ocasiones se hace más referencia a su carácter de dispositivo adaptativo lineal que a su naturaleza neuronal. La adalina se viene

utilizando con asiduidad desde los años sesenta como filtro adaptativo, por ejemplo, para cancelar el ruido en la transmisión de señales, como por ejemplo en el uso de las comunicaciones telefónicas por satélite [38]. De este modo, y desde hace años, millones de módems en todo el mundo incluyen una adalina.

4.7.3. Perceptrón Multicapa

El perceptrón multicapa es simplemente añadir al perceptrón simple unas capas intermedias u ocultas, obteniendo así un MLP (Multi-Layer Perceptron).

Una característica interesante de esta red neuronal es que se suele entrenar con un algoritmo llamado retropropagación de errores o BP (*Back Propagation*).

La estructura del MLP se presenta se presenta como podemos apreciar en **las *Figuras 19 y 20***. Denominamos x_i a las entradas de la red y como suele ser habitual y_j a las salidas de la capa oculta y finalmente z_k a las de la capa final; t_k serán las salidas del objetivo o *"target"*. Por otra parte, w_{ij} son los pesos de la capa oculta y ϑ_j sus umbrales, w'_{kj} los pesos de la capa de salida y ϑ'_k sus umbrales. La operación de un

MLP con una capa oculta y neuronas de salida lineal se expresa matemáticamente de la siguiente manera:

$$z_k = \sum_j w_{kj} y_j - \theta'_i = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i - \theta_j\right) - \theta'_i \qquad (12)$$

siendo *f(.)* de tipo sigmoidea como aparece en la ***Figura 19*** como por ejemplo la siguiente fórmula

$$f(x) \frac{1}{1+e^{-x}} \qquad (13)$$

Esta es la arquitectura más común de MLP, aunque existen numerosas variantes, como incluir neuronas no lineales en la capa de salida, introducir más capas ocultas, emplear otras funciones de activación, limitar el número de conexiones entre una neurona y las

de capa siguiente, introducir dependencias temporales o arquitecturas recurrentes [39].

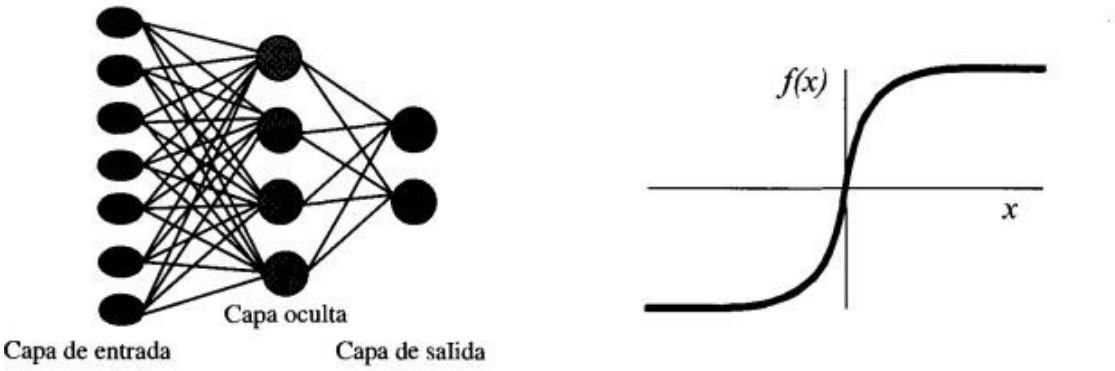


Figura 4.12: Perceptrón Multicapa y función de transferencia de la neurona

Como hemos comentado antes uno de los algoritmos que se utilizan para entrenar una MLP es utilizar el algoritmo de retropropagación de errores o BP, algo que veremos a continuación.

Sea un MLP de tres capas, cuya arquitectura se presenta en la ***Figura 20*** con las entradas, salidas, pesos y umbrales de las neuronas definidas en la sección anterior.

Dado un patrón de entrada x^{μ}, *(μ=1,...,p)*, y la operación global de esta arquitectura se realiza de la siguiente manera :

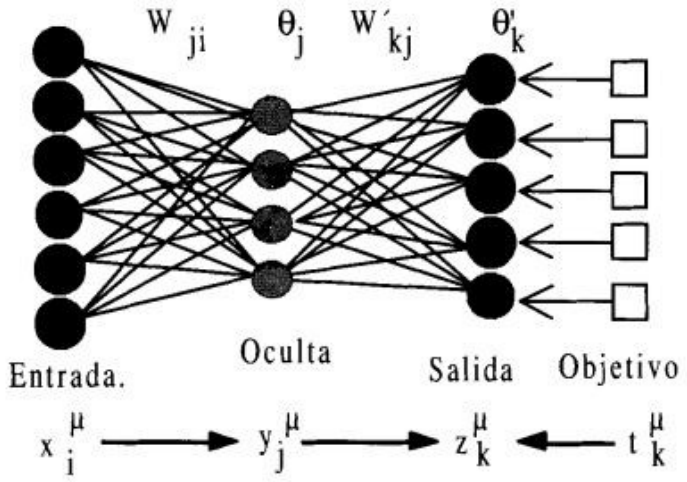


Figura 4.13: Arquitectura del MLP

$$z_k^\mu = \sum_j w_{kj} y_j^\mu - \theta'_k = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i^\mu - \theta_j\right) - \theta'_k \qquad (14)$$

Las funciones de activación de las neuronas ocultas *f(h)* son de tipo sigmoidea, con h el potencial postsináptico o local, siendo la *fórmula 13.* La función de coste de la que se parte es la del error cuadrático medio

$$E\left(w_{ji}, \theta_j, w'_{kj}, \theta'_k\right) = \left(\frac{1}{2}\right) \sum_\mu \sum_k [t_k^\mu - f(\sum_j w'_{kj} y_j^\mu - \theta'_k)]^2 \qquad (15)$$

La minimización se lleva a cabo mediante descenso por el gradiente, pero en esta ocasi6n habrá un gradiente respecto de los pesos de la capa de salida y otro respecto de los de la oculta

$$\delta w_{kj} = -\varepsilon \frac{\partial E}{\partial w_{kj}} \qquad \delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \qquad (16)$$

Las expresiones teniendo en cuenta las de la cadena de actualizaci6n de los pesos se obtienen solo con derivar, dependencias funcionales y aplicando adecuadamente la regla de la cadena

$$\delta w_{kj} = \varepsilon \sum_{\mu} \Delta'^{\mu}_{k}\, y^{\mu}_{j}, con\ \Delta'^{\mu}_{k} = [t^{\mu}_{k} - f(v'^{\mu}_{k})] \frac{\partial f(v^{\mu}_{k})}{\partial v'^{\mu}_{k}} \qquad (17)$$

$$\delta w_{ij} = \varepsilon \sum_{\mu} \Delta'^{\mu}_{j}\, y^{\mu}_{i}, con\ \Delta'^{\mu}_{j} = (\sum_{k} \Delta'^{\mu}_{k} w_{kj}) \frac{\partial f(v^{\mu}_{j})}{\partial v^{\mu}_{j}} \qquad (18)$$

La actualización de los umbrales se realiza haciendo uso de estas mismas expresiones, considerando que el umbral es un caso particular de peso sináptico, cuya entrada es una constante igual a -1

En primer lugar, lo que hacemos es calcular la expresión Δ'^{μ}_{k} que vemos en la fórmula 17, a la que vamos a llamar señal de error, por ser proporcional al error de la salida actual de la red, con el que calculamos la actualización $\delta w'_{kj}$ de los pesos de la capa de salida.

A continuación, se propagan hacia atrás los errores Δ'^{μ}_{k} a través de las sinapsis, proporcionando así las señales de error Δ^{μ}_{j} que tenemos en la fórmula número 18 correspondientes a la sinapsis de la capa oculta; con estás se calcula la actualización

δw_{ji} de las sinapsis ocultas. El algoritmo puede extenderse fácilmente a arquitecturas con más de una capa oculta siguiendo el mismo esquema

En forma resumida se puede decir que el procedimiento a seguir para entrenar mediante BP una arquitectura MLP es:

- Establecer aleatoriamente los pesos y umbrales iniciales (t:=0)
- Para cada patrón μ del conjunto de aprendizaje
    - Llevar a cabo una fase de ejecución para obtener la respuesta de la red ante el patrón *μ-ésimo* (fórmula 14)
    - Calcular las señales de error asociadas Δ'^{μ}_{k} y Δ^{μ}_{j} (fórmulas 17 y 18)
    - Calcular el incremento parcial de los pesos y umbrales debidos a cada patrón μ (elementos de los sumatorios de las fórmulas 17 y 18)

- Calcular el incremento total (para todos los patrones) actual de los pesos $\delta w'_{kj}$ y δw_{ij} según (las fórmulas 17 y 18). Hacer lo mismo para los umbrales.

- Actualizar pesos y umbrales.

- Calcular el error actual (fórmula 15); t:=t+1, y volver al segundo punto si todavía no es satisfactorio.

Se debe comenzar siempre con pesos iniciales aleatorios (normalmente números pequeños, positivos y negativos), ya que si se parte de pesos y umbrales iniciales nulos el aprendizaje no progresará.

En el esquema presentado, que surge de forma natural del proceso de descenso por el gradiente, se lleva a cabo una fase de ejecución para todos y cada uno de los patrones del conjunto de entrenamiento, se calcula la variación en los pesos debida a cada patrón, se acumulan, y solamente entonces se procede a la actualización de los pesos.

CAPITULO 5

REDES NEURONALES CONVOLUCIONALES

5.1. Introducción

Una vez introducido los conceptos iniciales sobre inteligencia artificial, y sobre todo incidir en la parte del deep learning y posteriormente en las redes neuronales artificiales. Ahora el trabajo se centra en un tipo de red neuronal que en los últimos años han tenido un enorme crecimiento a nivel aplicativo como de investigación, que son las redes neuronales convolucionales que es, uno de los principales ejes de este trabajo.

Una red neuronal convolucional o también llamadas CNN (Convolucional Neural Networks) es una herramienta realmente reciente a nivel tecnológico, si lo comparamos con otras tecnologías que ya están profundamente arraigadas en nuestro día a día, pare ello voy a realizar un breve repaso histórico para conocer un poco más de cerca la evolución de dichas redes y tener una idea de qué es una red neuronal convolucional.

5.1.1. Definición de una CNN

Una red neuronal convolucional es un tipo de red neuronal artificial de aprendizaje supervisado, surge como una variación del perceptrón multicapa, diseñado para trabajar con imágenes, tomando estas imágenes como entrada y posteriormente asignándole pesos a ciertos elementos de las imágenes para poder diferenciar unos de otros.

Las redes neuronales convolucionales procesan sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas que en definitiva hacer que pueda identificar objetos y por lo tanto realizar la acción humana de ver.

Las redes neuronales convolucionales poseen varias capas ocultas de las cuales unas cuantas obtienen información básica acerca de los colores o bien de los bordes, pero a medidas que se va profundizando más en las capas más profundas se extraen cada vez más características más abstractas o complicadas de detectar [40].

5.2.1. Historia

Para entender un poco mejor las redes neuronales convolucionales, es importante ver una serie de referencias históricas que han

marcado el desarrollo de esta importante tecnología aplicada a la inteligencia artificial, y poder así facilitar el entendimiento de dicha tecnología.

El primer punto donde comienza todo este mundo de las redes neuronales convolucionales es en 1959 con la presentación de un trabajo realizado por Hubel y Wiesel [41] como se puede ver en la ***Figura 5.1***, un trabajo centrado más en el aspecto

biológico sobre todo si lo comparamos con las redes neuronales artificiales, por el cual dicho trabajo resulto de capital importancia a la hora de poder entender cómo funciona la corteza visual del ser humano, por lo general las células responsables de la selección de orientación y detección de bordes en los estímulos visuales dentro de la corteza visual primaria v1. Dos tipos de células se identificaron debido a que tenían campos receptivos alargados, con lo cual tienen una mejor respuesta a los estímulos visuales alargados como las líneas y los bordes. Estas dos células se denominan simples y complejas.

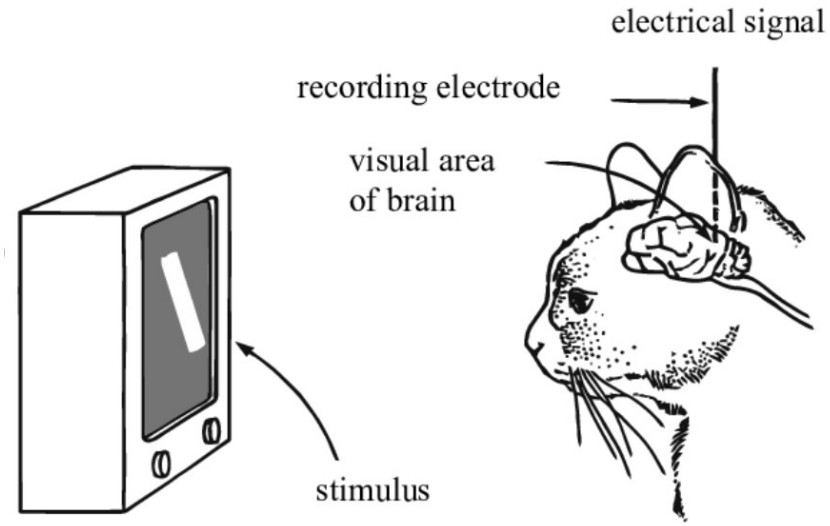


Figura 5.1: Experimento de Hubel y Wiesel, en el que se descubre la neurona de la corteza visual de un gato

Posteriormente en el año 1980 comienza el verdadero inicio de este tipo de redes con el Neocongnitron (*ver* ***Figura 5.2***), introducido por Kunihiko Fukushima [42], desarrolla lo que sería el precursor de las redes convolucionales. Se introducen ciertos conceptos como la extracción de características, agrupación de capas y uso de convolución en una red neuronal. Fukushima se inspiró en algo que casi siempre suele servir de ejemplo en el área de la inteligencia artificial, más concretamente el deep learning que es en la naturaleza, justamente en el sistema nervioso visual de los vertebrados.

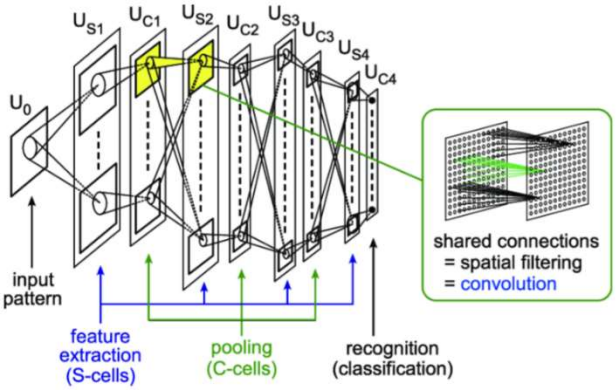


Figura 5.2: Arquitectura del Neocongnitron

Pero uno de los momentos más importantes sobre este campo se desarrolla durante los años 1989 a 1998 cuando Yann LeCun[43] y su equipo desarrollan el LeNet-5 (*ver* ***Figura 5.3***), una red convolucional simple, por el cual era una mejora del Neocongnitron visto antes y desarrollado en los años 80, por el cual LeCun y su equipo aplicaron el aprendizaje basado en el *backpropagation,* con el objetivo de realizar la tarea de reconocimientos de dígitos escritos a mano con una precisión del 99,3%

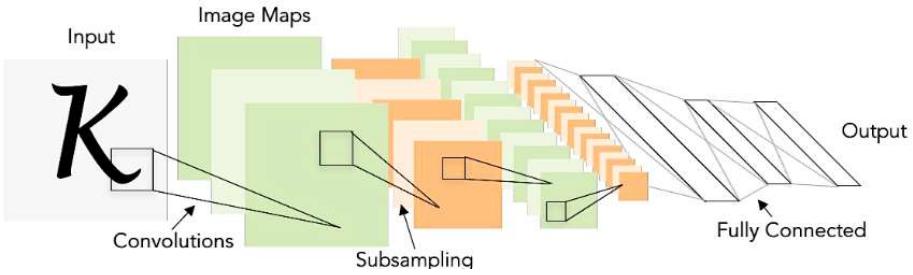


Figura 5.3: Arquitectura de la red neuronal convolucional LeNet-5

Finalmente otra fecha a remarcar en el ámbito de las redes neuronales convolucionales es el año 2012, cuando se crea la red AlexNet por Geoffrey Hinton, Ilya Sutskever y Alex Krizhevsky[36] de cara a un concurso anual llamado ImageNet Large Scale Visual Recognition Challenge (ILSVRC) que es cuando realmente se observan el verdadero potencial de las redes neuronales para la clasificación de imágenes. El resultado principal del documento original fue que la profundidad del modelo era esencial para su alto rendimiento, lo cual era computacionalmente costoso, pero factible debido a la utilización de unidades de procesamiento gráficos (GPU) durante el entrenamiento.

5.2. Funcionamiento de una CNN

Para entender de manera más sencilla el funcionamiento de una red neuronal, se va a repasar en términos generales todas las etapas por las que pasa una imagen desde que es captada hasta que la red finaliza el trabajo como se puede apreciar en a la ***Figura 5.4***

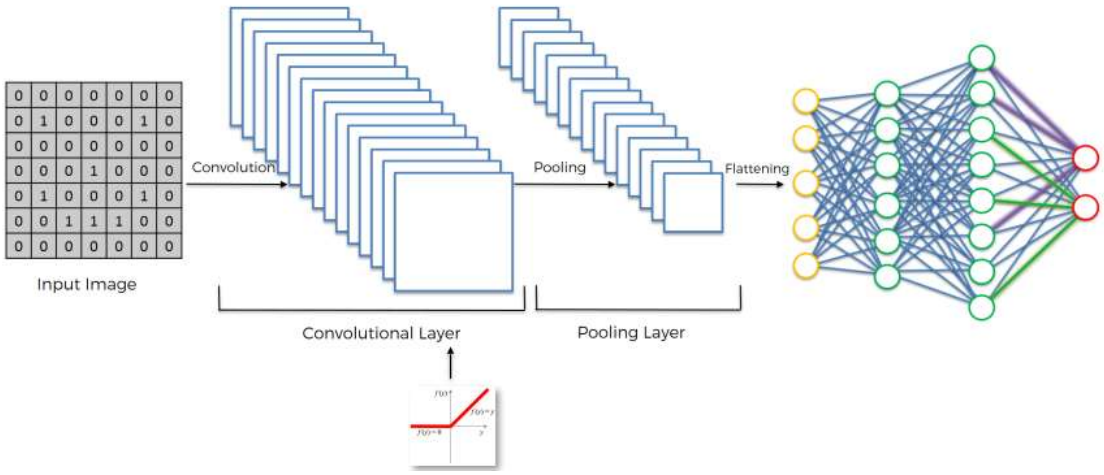


Figura 5.4: Etapas por las que pasa una imagen dentro de una red neuronal convolucional

5.2.1. Imagen digital

Para poder comenzar, lo primero que necesita una red neuronal convolucional es una imagen y que esta sea transformada a un formato en que pueda poder ser analizada de manera digital, por lo

que es importante conocer algunos pequeños detalles sobre la representación de imágenes digitales.

Por lo general existen dos tipos de imágenes que se pueden tratar a la hora de introducirlos como entrada que pueden ser las imágenes en blanco y negro o bien a color.

Una imagen digital monocromática es una función bidimensional de la intensidad de la luz formado por *f(x,y),* donde el valor o amplitud de *f* en las coordenadas espaciales *(x,y)* da la intensidad, iluminación o nivel de gris, de la imagen en aquellas coordenadas.

En definitiva esta definición es aplicable a las imágenes de escala de grises, ya que la luz es una forma de energía f*(x,y)* debe ser positiva y finita[44]

$$0 < f(x,y) < \infty \qquad (19)$$

Una imagen de este estilo puede ser considerada como una matriz de tamaño m x n cuyos índices de fila y columna identifican a un punto

de la imagen, y el valor correspondiente es proporcional a la intensidad [38]

$$\hat{F} = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ f(1,0) & f(1,1) & \dots & f(1,n-1) \\ \vdots & \vdots & & \vdots \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{bmatrix} \qquad (20)$$

A cada elemento f(x,y) de la imagen $\hat{F}$ se de denomina pixel

Para el caso de una imagen de color, se denomina imagen de color *RGB* a un arreglo de tres imágenes monocromáticas independientes de tamaño *m x n* correspondientes a la escala de rojos (R-red-), verdes(G-green-) y azules (B-blue-)

El método de reproducción de color usado en los monitores de color se basa en la primera ley de Gassmann[45], en lo que consiste en partir del negro, e ir añadiendo mayor o menor cantidad de luz de los tres colores básicos que son el *RGB* .

5.2.2. Exploración

Las redes neuronales captan una serie de imágenes de carácter digital, por lo tanto, una vez sabido el concepto básico de una imagen digital, lo que debemos comprender ahora es como reconoce la red una imagen.

Para ello se van a poner dos imágenes (***ver Figura 5.5***) por la cual se procederá, el análisis de las similitudes y diferencias que hay profundizando un poco más la definición o concepto de la imagen digital vista previamente en la que podemos ver dos imágenes una en blanco y negro y la otra a color.

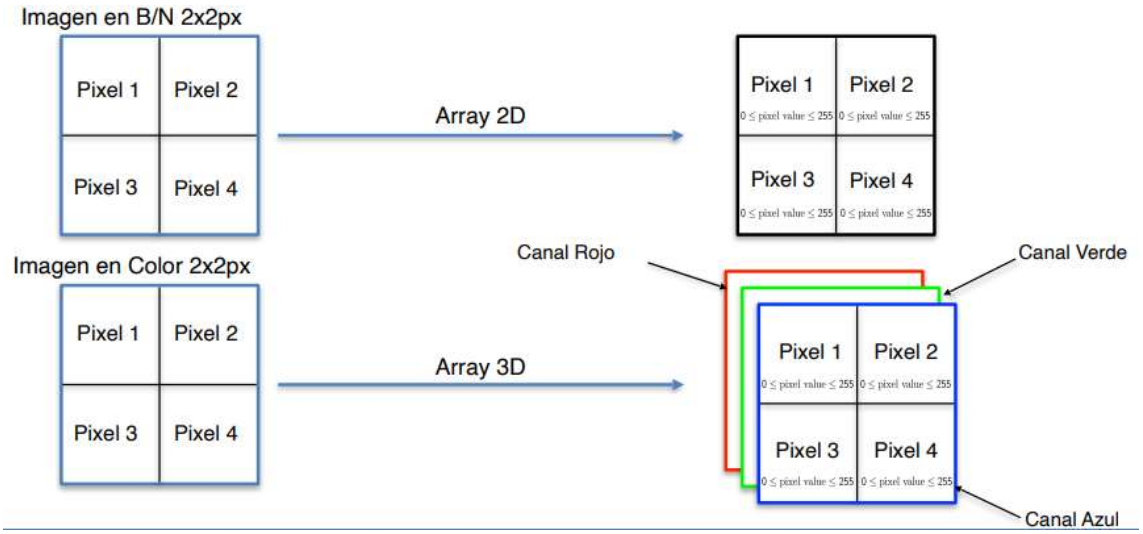


Figura 5.5: Imagen en escala de grises y a color

Al observar estas dos imágenes, se podría decir que se pueden extraer una serie de conclusiones por el cual podemos sacar una serie de similitudes y diferencias

Las principales similitudes que tenemos en ambas imágenes son:

- Cada pixel contiene 8 bits de información con la cantidad equivalente a 1 byte
- Los colores se representan a una escala de 0 a 255, ya que las unidades binarias por la que los bits forman parte sabemos que 8 bits son 1 byte, por lo que podemos tener el valor de 2^8 valores posibles que serían 256. Sabemos que 2^0 tiene el valor de 1 pero al contar desde 255 tomamos el 0 como valor inicial
- En este modelo la 0 toma como referencia el color negro y el 255 es el blanco
- Debemos recordar que los ordenadores no entienden colores tal y como lo conocemos, sino que trabajan con ceros y unos

También tenemos ciertas diferencias que son:

- Las imágenes en blanco y negro son bidimensionales, mientras que las imágenes a color son tridimensionales
- Lo dicho anteriormente conlleva a que, en el caso de las imágenes bidimensionales, a cada pixel se le asigna un número entre 0 y 255 para representar su sombra, pero en una imagen a color esto es representado a tres

niveles, al ser estos una combinación de rojo, verde y azul a diferentes niveles de concentración, a un solo pixel en una imagen coloreada se le asigna un valor separado para cada una de estas capas. Esto significa que una capa roja es representada con un número entre 0 y 255, y lo mismo con las capas azules y verdes. Por ejemplo, un pixel "amarillo" estaría presentado a una red neuronal como (255,255,0).

Por lo tanto, hay cuenta que la red toma como entrada los píxeles de una imagen, luego si tenemos una imagen de 28x28 equivaldría a 784 neuronas. Si fuera una imagen a color necesitaríamos 3 canales y entonces usaríamos 28x28x3 que serían 2352 neuronas de entrada.

5.2.3. Detección de imágenes y preprocesamiento

Para entender un poco más el concepto por el cual una red neuronal capta una imagen en su entrada veamos el siguiente ejemplo como podemos ver en la siguiente figura (***ver Figura 5.6***):

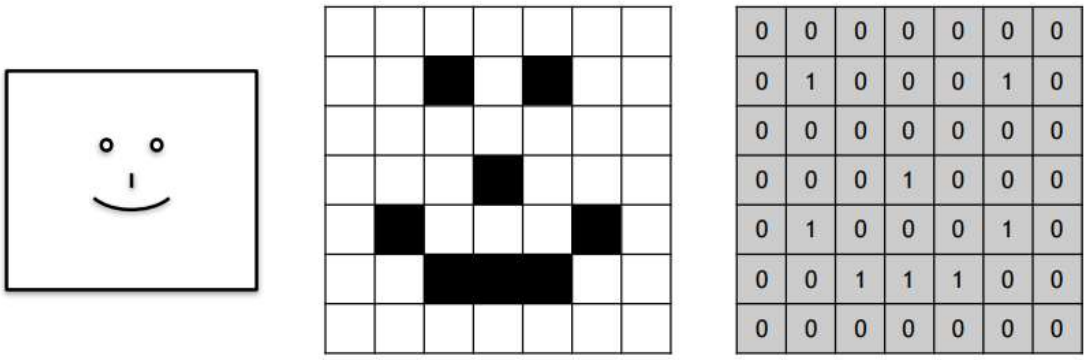


Figura 5.6: Imágenes en varios formatos

Podemos ver como tenemos una imagen de una cara sonriente, una figura muy básica y sencilla.

Como se puede ver, la tabla de la cuadricula central muestra todos los píxeles valorados en 0, mientras que solo las partes donde aparece la cara sonriente están valoradas en 1, algo que difiere con el modelo de 8 bits.

Lo que se hace cuando entrenamos una red neuronal convolucional para detectar en este caso sonrisas es normalizar los valores, es decir el valor de los píxeles sabemos que va de 0 a 255 pero se normaliza para la red de 0 a 1, luego debemos enseñarle los

patrones de 0 y 1 que normalmente se asocian con la forma de una sonrisa, para luego pasarlo a un formato matricial.

Para realizar la normalización simplemente hay que hacer una pequeña transformación de cada pixel:

$$normalización = \frac{valor\ (0-255)}{255} \tag{21}$$

5.2.4. Convolución

La convolución es uno de los primeros pasos importantes que se deben analizar para poder comprender las redes neuronales artificiales.

Antes de nada, se debe tener en cuenta que la convolución tiene una fórmula y es la siguiente:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t-\tau) d\tau \tag{22}$$

En términos meramente matemáticos, convolución es una función derivada de dos funciones dadas por integración, que expresa como la forma de una es modificada por la otra, ahora bien, para la explicación de la convolución, este trabajo va a abandonar el camino matemático para centrarse exclusivamente en cómo funciona este paso, de cara a comprender mejor el desarrollo de la parte práctica.

Lo que realiza la red como se ha dicho antes, es tomar como entrada una imagen y la trata como una matriz de dimensiones número de píxeles x número de píxeles x número de canales de color. Si por lo que fuera la imagen es de color el número de canales será de 3, en referencia al RGB.

Lo primero que se necesitará para realizar la operación de convolución es tener un filtro que será el que se encargue de extraer todas las características de la imagen, y dicho filtro se denomina kernel o también en algunos libros máscara o detector de características.

La convolución consiste en un paso muy sencillo, por el cual se desplaza el kernel por cada uno de los elementos de la imagen original, realizando el producto escalar entre la matriz y el filtro obteniendo así el mapa de características tal y como vemos en las

Figuras 5.7 y 5.8 en las que vemos el inicio y el resultado de la operación de convolución.

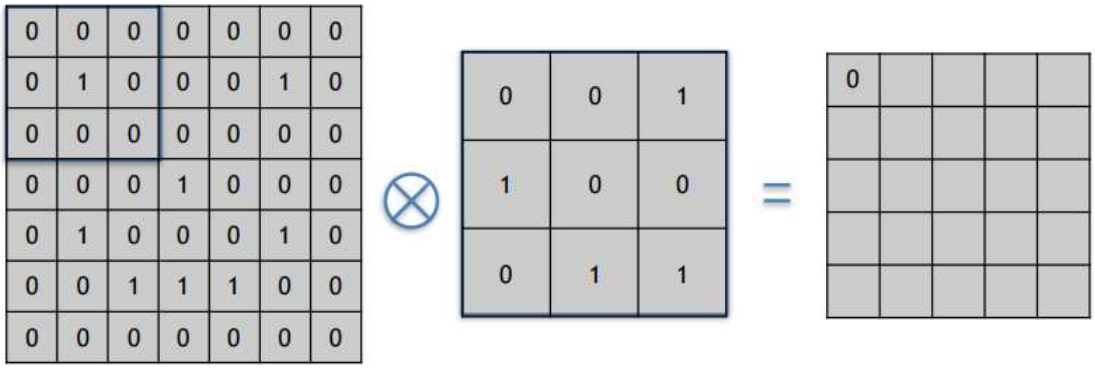


Figura 5.7: Operación de convolución inicial

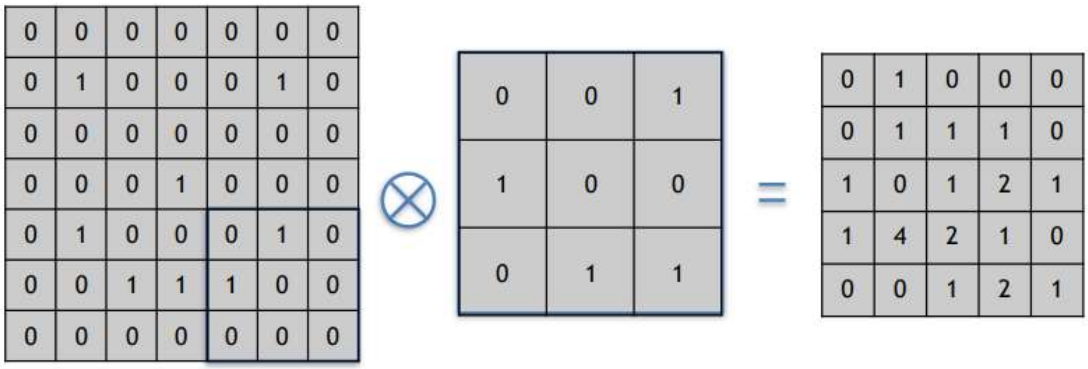


Figura 5.8: Finalización de la operación de convolución

Existen varios parámetros que permiten controlar las dimensiones del mapa de características de salida:

- El tamaño del kernel, así como el número de ellos, este parámetro indicará la profundidad de salida. Por ejemplo, si tenemos 6 filtros, obtendremos 6 mapas de activación separados. Muchas veces se suele utilizar matrices de 5x5, pero lo más común es trabajar con matrices 3x3

- Stride: Cuando la convolución del kernel con la imagen no se realiza en todos los pixeles, sino que se hace cada cierto número de ellos. Este paso se denomina stride y cuanto mayor sea más pequeña es la salida.

- Zero padding: Significa que se agregan píxeles de valor cero alrededores de la imagen original, por lo que, de esta forma, el mapa de características podrá tener el mismo tamaño que la imagen de entrada [46].

A medida que se entrena la red, los valores de los kernel o lo que es lo mismo el peso de las neuronas se van actualizando para disminuir el error en la predicción.

Hay que apuntar que el ejemplo visto en las figuras anteriores es para un caso sencillo, como podemos apreciar en la ***Figura 5.9***

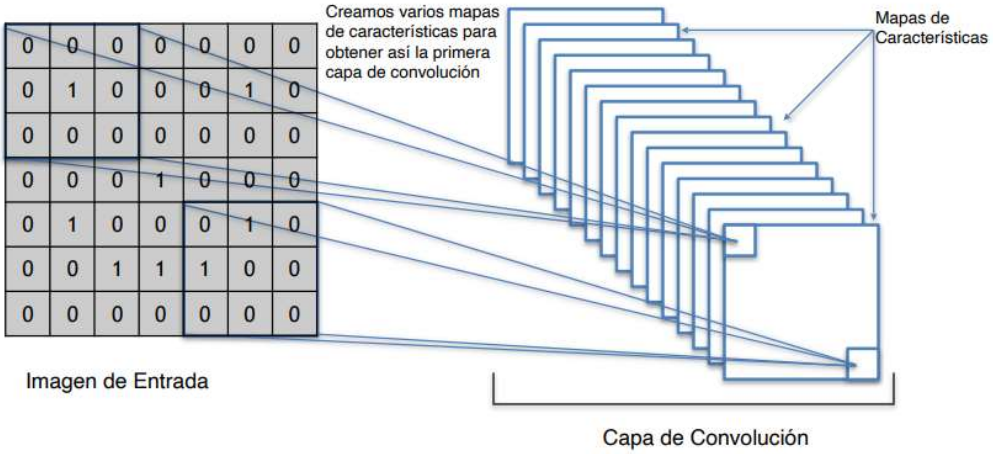


Figura 5.9 Capa de convolución

Existen muchos mapas de características y muchos filtros en lo que formarían parte de la capa de convolución.

5.2.5. Unidad Lineal Rectificadora (ReLU)

La siguiente etapa que ocurre durante la operación es el uso de la Unidad Lineal Rectificadora (ReLU), la cual se puede ver en la ***Figura 5.10*****,** que realmente es una función de activación que también pueden ser utilizadas en las redes neuronales artificiales.

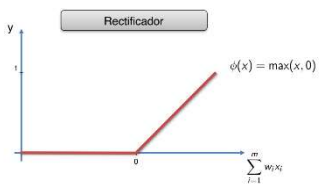


Figura 5.10: Función Unidad Lineal Rectificadora

Función muy popular entre las redes neuronales artificiales, por el hecho de que transforma todo lo negativo directamente en 0.

Elimina todo lo que sea negativo, todo aquello que una vez ponderado por los pesos de la red neuronal da negativo, no interesa desaparece, y a partir de ahí todo lo positivo se conserva como tal.

La principal razón de aplicar esta función de rectificación es incrementar la no linealidad en las imágenes, ya que las imágenes no son lineales.

5.2.6. Pooling

Tras terminar toda la etapa de convolución, es interesante realizar una reducción del volumen de datos, realizando una tarea muy parecida a la que se realiza en la capa de convolución.

En este caso tenemos dos tipos de pooling que son:

- Max Pooling: Dada una matriz A_{axa} podemos definir el proceso de Max Pooling con una amplitud *k* y un stride *p* como la matriz *P(i,j)* tal que :

$$P(i,j) = \max_{n,m=1,\dots k} A[(i-1)p+m,(j-1)p+n] \qquad (23)$$

  En el caso de necesidad de incluir más filas o columnas, se pueden añadir mediante el método de zero-padding.
  Por lo general se suele usar un algoritmo de Max Pooling con una amplitud y stride de 2
- Average Pooling: Es similar al método de Max Pooling, salvo que se trata de una media aritmética

$$P(i,j) = \frac{1}{k^2} \sum_{n,m=1\dots k} A[(i-1)p+m,(j-1)p+n] \qquad (24)$$

Se expondrá un ejemplo de cómo se realiza el proceso, en este caso de Max Pooling como podemos ver en las ***Figuras 5.11 y 5.12***

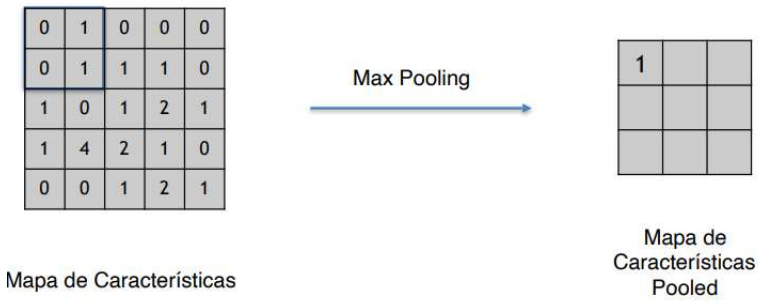


Figura 5.11: Operación de Max Pooling

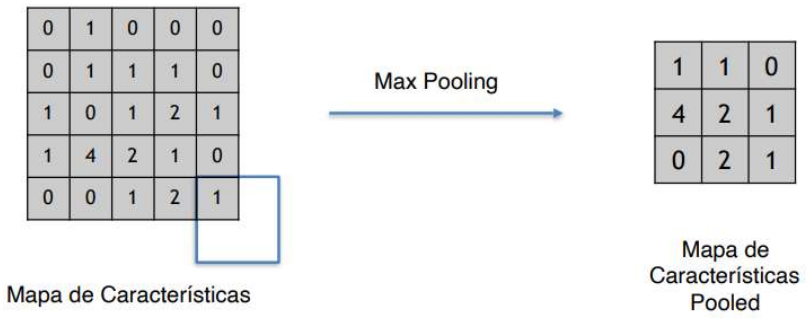


Figura 5.12: Finalización de la operación

Lo que sucede en este caso es con un stride 2 que es el caso más común, va recorriendo la matriz y de entre todos los valores que están dentro de la matriz, se elige el mayor, quedando un mapa de características pooled con una dimensión menor. Es evidente además que con esta reducción se

pierde información, pero es por lo general información innecesaria, ya que reduciendo el tamaño de la imagen a su vez hace que la red pueda realizar su trabajo de manera más eficiente.

Hay que remarcar que tras hacer el paso de pooling se vuelve a realizar el mismo proceso de capa convolutiva varias veces, es decir, volveríamos al punto 5.2.4 , ya que estas redes suelen tener muchas más capas convolutivas.

5.2.7 Flattening

Este paso es bastante sencillo, lo único que se hace es convertir el mapa de características en un array como podemos ver en la ***Figura 5.13*** que servirá de entrada a la red neuronal artificial.

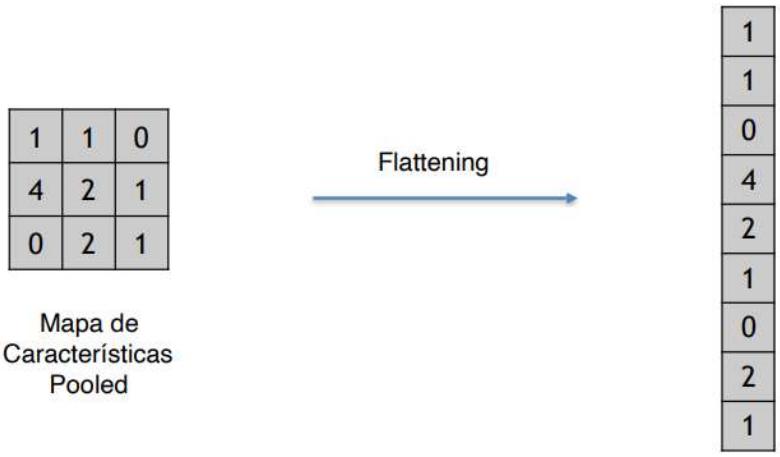


Figura 5.13: Operación de Flattening

5.2.8. Fully Connected

Es una de las últimas capas que, aunque no forman parte del proceso de tratamiento de la imagen en sí mismo, se utiliza para realizar procesos de detección y de clasificación.

Como se ve en la ***Figura 5.4*** vista anteriormente, se coloca al final de la arquitectura de la red, por el cual recibe de entrada todas las imágenes que han sido previamente trabajadas en el proceso de convolución y pooling a las que finalmente tras realizar el proceso de flattening llegan a las entradas de la misma, para después asignar un valor numérico por imagen que representa la probabilidad de pertenecer a una determinada clase.

5.3 Arquitecturas de una CCN

Existen muchas arquitecturas conocidas de redes neuronales convolucionales aparte de la muy conocida LeNet, a continuación, se nombrarán las más importantes:

- AlexNet : Tiene 8 capas en total de las cuales 5 son capas convolutivas y 3 completamente conectadas. Se introdujo al principio una capa de normalización llamada capa de normalización opuesta. Normalizó todos los valores en una ubicación particular a través de los canales en una capa determinada. Además, introdujo la ReLU como función de activación. Posee alrededor de 60 M de parámetros [47].

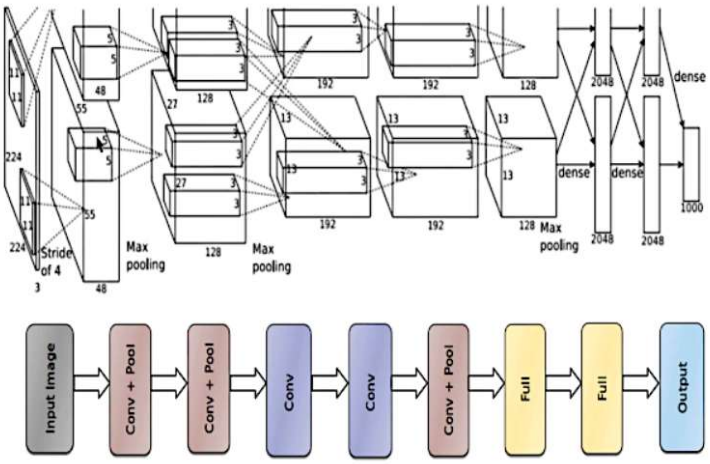


Figura 5.14: Arquitectura AlexNet

- VGGNet: Inventada por Visual Geometry Group(Universidad de Oxford). Se argumentó que al profundizar la red en se pueden resolver mejor los problemas y obtener una tasa de error más baja. En particular VGG-16 consta de 13 capas convolutivas más 3 capas de fully conected, constando de 138 millones de parámetros y una sobrecarga de memoria significativa de 48,6 MB comparados con los 1,9 MB de AlexNet . Hay que destacar que además se añadió una variante más profunda llamada VGG-19 , pero no fue tan popular como al primera [48].

Figura 5.15: Arquitectura de la VGG16 Y 19

- GoogleLeNet : Creada por Google, esta red se centra más en el objetivo de obtener una mayor eficiencia para reducir el conteo de parámetros, memoria y el cálculo. Fue más profundo de hasta 22 capas, pero sin ninguna capa completamente conectada (FC). Al deshacerse de las capas fully conected, la cantidad total de parámetros se redujo a 5 millones. Se introdujo también el módulo de inicio, o también llamado Unidad local con sucursales paralelas fue una innovación clave en el enfoque, por lo que dicho módulo se apiló (en lugar de capas convolucionales) muchas veces a lo largo de la red [49].

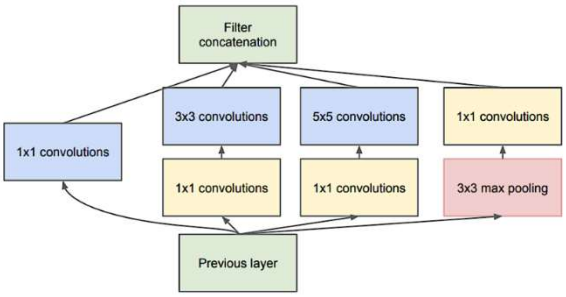


Figura 5.16 Arquitectura GoogleNet

- ResNet: Introducida por Microsoft, la idea son los llamados bloques residuales, que están conectados entre si a través de conexiones de identidad en la arquitectura ResNet.
  Una idea residual es una pila de muchos bloques residuales. Cada bloque residual tiene dos capas convolutivas 3x3 tal y como vemos en la ***Figura 5.17*** de la arquitectura de dicha red

El equipo popularizó la conexión de salto y profundizó aún más hasta 152 capas sin el poder de generalización del modelo. Hay que señalar que ResNet ha ganado varias competiciones tales como ILSVRC y COCO 2015[50].

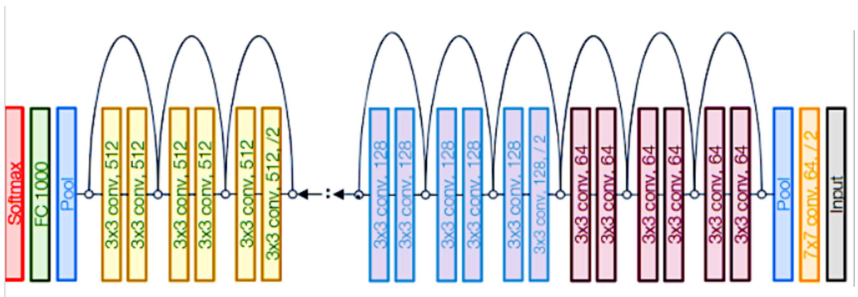


Figura 5.17 Arquitectura ResNet

CAPITULO 6

APLICACIÓN PRÁCTICA

6.1. Materiales necesarios

De cara a la implementación del trabajo, se va a tener que requerir de una serie de elementos que por lo general son bastante fáciles de utilizar e implementar, por lo que a continuación serán descritos los materiales usados:

6.1.1. Arduino Mega 2560

Arduino fue creado en el año 2005 en el Instituto de Diseño Interativo de Ivrea en Italia. Arduino nació como un proyecto de cara a su utilización en el ámbito académico y que sea de un coste bastante reducido para que de esta manera fuera accesible para todos los públicos, pero ante una perspectiva de cierre y pérdida del proyecto se decidió liberarlo y hacerlo público con el objetivo de que todo el mundo pudiera participar de él proponiendo mejoras y sugerencias.

Arduino tiene múltiples ventajas respecto a otras placas, partiendo de su costo que es bastante accesible, posee además una enorme comunidad a nivel mundial que día a día mejora el proyecto Arduino, así como una enorme documentación, la enorme versatilidad y reusabilidad ya que se puede montar y desmontar fácilmente para implantar otros proyectos, así como un lenguaje de programación bastante sencillo basado en C++.

De cara a la realización del proyecto se va a usar una Arduino Mega 2560. Como características principales, posee 54 entradas y salidas digitales de las cuales 15 pueden usarse para salidas PWM, 16 entradas analógicas, 4 UARTs o puertas seriales, conexión USB, un conector de alimentación de corriente y un botón de reset.

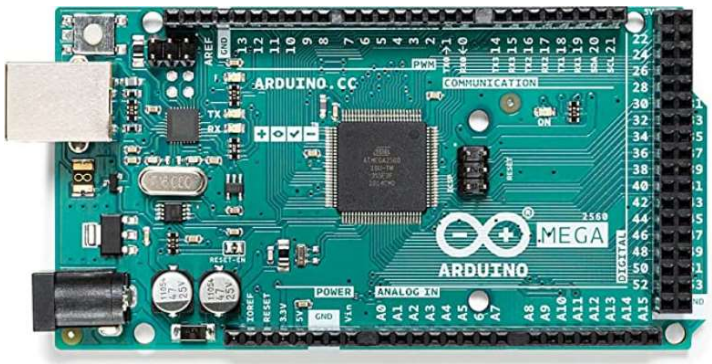


Figura 6.1: Placa Arduino Mega 2650

6.1.2 Módulo H-Bridge o L298N

El módulo controlador de motores L298N H-Bridge, es un elemento fundamental para todos los proyectos de robótica y automatización que requiere el control de motores de continua y paso a paso.

El L298N es la denominación del circuito integrado principal del módulo y se encuentra montado de forma vertical sobre la placa de circuito impreso y tiene adosado un disipador pasivo de aluminio pintado de color negro. Este circuito integrado incluye transistores de potencia para el manejo de hasta dos motores de corriente continua en variedad de tensiones y se le denomina doble puente H.

El rango por el cual trabaja este módulo está comprendido entre los 3V hasta los 35V, con una intensidad de corriente de hasta 2A. En este trabajo usaremos dos módulos por cada motor es decir uno controlará el motor A y el otro se centrará en el motor B

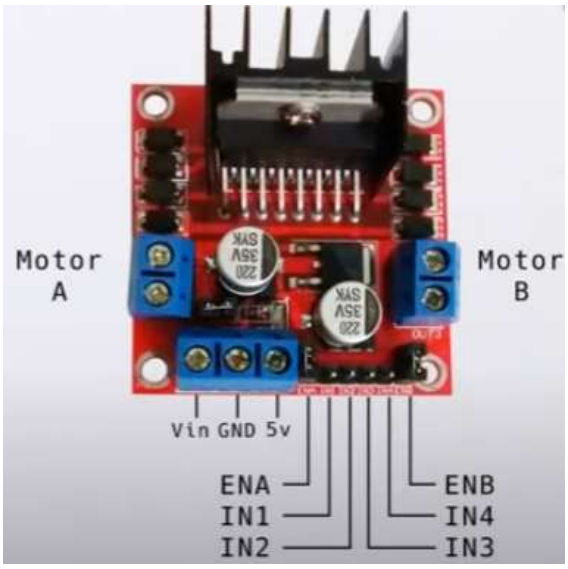


Figura 6.2: Módulo H-Bridge – L298N

6.1.3 Motores Andy Mark Neverest 60

Por lo general este tipo de vehículos es generalmente usado a nivel industrial, como se ha comentado en el estado del arte, utilizados para transportar pesos, por lo que los motores son uno de los elementos más críticos a la hora de un correcto funcionamiento del vehículo. Es importante seleccionar motores potentes que puedan soportar tanto el peso de los componentes del propio vehículo como también en función de la tarea que deba llevar a cabo.

Para este cometido se van a utilizar motores AndyMark Neverest 60 - am3103- usando como conector un JST-VH-2 Anderson, alimentados con una tensión de corriente continua de 12V , llegando a tener una potencia máxima de hasta 14W .

Figura 6.3 Motor Andy Mark Neverest 60

6.1.4. Batería y Cargador

Para poder hacer funcionar los motores, se utiliza una batería de marca DYNO de Plomo-Ácido de 12V-5Ah 90x70x107mm y para poder cargar la batería, se dispone de un cargador de baterías de plomo de 6V y 12V de formato mural con un conector Jack hueco de 5.5x2.1x10mm y pinzas de cocodrilo con una intensidad de carga de 500mA.

Figura 6.4: Batería de Plomo-Ácido

Figura 6.5 :Cargador

6.1.5. Elementos de Movilidad y Base

Para el caso de la base, se reutiliza una base cuadricular para poder colocar todos los elementos necesarios para el funcionamiento del

vehículo. De cara a la movilidad, se en la parte delantera se requerirá de dos ruedas de 65mm con acoplador de acoplamiento flexible.

De cara a la parte trasera, se requerirá de una rueda con pivote de 50mm de diámetro con su correspondiente placa de montaje y rodamiento, Ambos neumáticos son de goma

Figura 6.6 Rueda de 65mm

Figura 6.7 Rueda con pivote

6.1.6. Webcam HD Logitech C270 720p/30fps

La cámara utilizada para el trabajo es una Logitech C270 con una resolución máxima de 720p/30fps con un campo visual diagonal de 55º, de carácter bastante sencillo y ligero ideal para este tipo de trabajos ya que posee un peso de 75 g y un cable USB bastante aceptable de largo de 1,5 m pudiéndolo colocar de una forma bastante cómoda para poder realizar las pruebas necesarias sin necesidad de tener limitaciones.

Figura 6.8 Webcam Logitech

6.1.7. Software

Tras un análisis exhaustivo de todas las opciones disponibles para la realización del trabajo, se ha tomado la decisión de utilizar Matlab para programar los códigos necesarios para

Las razones más importantes de su elección en detrimento de otros lenguajes de programación son sobre todo por su gran versatilidad a la hora de poder aplicarse a múltiples disciplinas de la ingeniería como la aportación de los toolbox.

Matlab en este caso nos va a ser muy útil a la hora de poder utilizar las redes neuronales y poder ejecutar la placa de Arduino a la vez.

6.2. Puesta a punto, pruebas iniciales

Antes de proseguir con la práctica, se deberá comprobar que todos los componentes funcionan adecuadamente, por lo que se va a desarrollar una serie de códigos que servirán de comprobación de que el material está a punto.

Lo primero se va a comprobar es funcionamiento de dos componentes esenciales para la realización de la práctica que son los motores y la cámara, por lo que se va a dividir en dos partes para que

sea más sencillo la comprensión de dichas pruebas por parte del lector, haciendo primero un análisis de la cámara y después un análisis de los motores como también realizando una explicación del funcionamiento de los mismos.

6.2.1. Cámara

El primer paso que se debe realizar es instalar en Matlab, el paquete de soporte de Matlab para cámaras USB, para poder utilizar la webcam c270 para el trabajo.

Para ello se debe ir en la barra de herramientas de Matlab a la sección *"Add-Ons"*, que nos llevará a una ventana donde se encuentran todas las herramientas disponibles que podemos instalar en Matlab que se necesitarán.

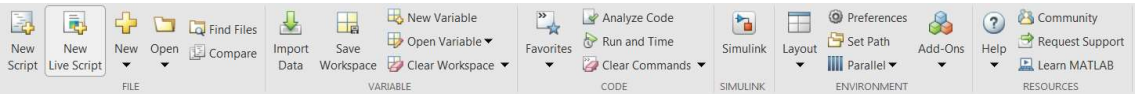


Figura 6.9: Barra de Herramientas de Matlab

Tras acceder al ToolBox de Matlab, se procederá a instalar la herramienta para que se pueda utilizar la webcam, que aparece en la ***Figura 6.10***

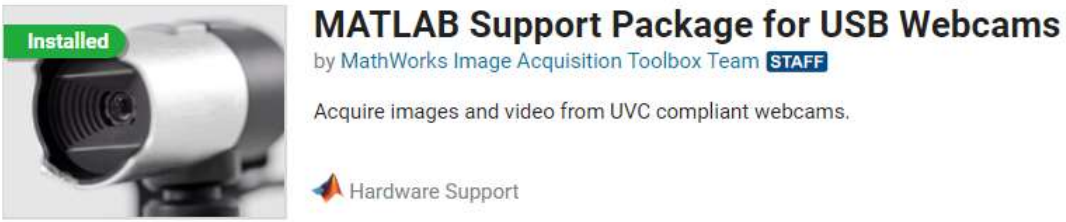


Figura 6.10: Herramienta para las webcams

Finalmente, tras instalar la herramienta, se va a probar la cámara, y lo primero que se debe realizar, teniendo en cuenta que se va a ejecutar la prueba desde un ordenador portátil, es seleccionar la cámara, ya que por defecto los ordenadores portátiles actuales poseen una por defecto. Para ello primero se debe obtener el valor o número identificador ha sido asignado para la webcam, por lo que, en la línea de comandos de Matlab con el comando webcamlist, como se puede ver en la ***Figura 6.11***, y seleccionamos la primera cámara con el numero 1

```
>> webcamlist

ans =

  2×1 cell array

    {'C270 HD WEBCAM'          }
    {'HP Wide Vision HD Camera'}

fx >>
```

Figura 6.11: webcamlist

Posteriormente, ya sabiendo que número tiene asignado la cámara que se va a utilizar para hacer las pruebas, el siguiente paso es realizar un pequeño script llamado *pruebaCamara.m, para* comprobar su funcionamiento como el que se mostrará a continuación:

```
clearvars
camara = webcam(1);
camara.Resolution= '640x480';
preview(camara)
image = snapshot (camara);
closePreview(camara)
imshow(image);
```

Con el código anterior lo que se ha hecho es simplemente sacar una foto como aparece en la ***Figura 6.12*** y mostrarla por pantalla. Simplemente lo que se hace es asignar a una variable que llamamos cámara, la webcam que se va a utilizar para la realización de la prueba, a la que además se añadirá una resolución *640x480,* y en una variable que llamamos *image*, guardamos la captura de una foto con la webcam a la que posteriormente con la instrucción imshow se puede comprobar la foto sacada.

Figura 6.12: Foto tomada con la webcam

Por lo que, tras esta pequeña prueba, se ha podido comprobar el funcionamiento de la webcam. Como elemento adicional podemos ver algunas características como podemos ver en la ***Figura 6.13*** con el comando webcam

```
>> webcam

ans =

  webcam with properties:

                    Name: 'C270 HD WEBCAM'
     AvailableResolutions: {1×19 cell}
               Resolution: '640x480'
                Sharpness: 24
          WhiteBalanceMode: 'auto'
                Saturation: 32
            ExposureMode: 'auto'
                Exposure: -6
            WhiteBalance: 0
                    Gain: 8
                Contrast: 32
              Brightness: 128
    BacklightCompensation: 1

>>
```

Figura 6.13 usando el comando webcam

6.2.2. Motores

Como se recuerda en el apartado 5.1, se va a necesitar de dos componentes esenciales, aparte de los motores y las ruedas, para poder realizar esta prueba, que son la placa Arduino y el Módulo H-Bridge o L298N.

Antes de nada, se pasará a explicar todos los pines el Módulo H-Bridge, y posteriormente se realizará un código de prueba en Matlab para comprobar su funcionamiento. Es posible también realizarlo por por el entorno de programación de Arduino-*Arduino IDE*-, pero como se va a realizar la totalidad del trabajo en Matlab, vamos a realizarlo en este último.

Como se puede apreciar en la ***Figura 6.2*** del apartado 6.1.2, en el Módulo tenemos varios pines que se requieren especial atención.

Lo primero que tenemos en la parte izquierda y derecha son los pines donde vamos a conectar nuestros motores A y B por donde recibirán la energía necesaria para poder girar. Estos pines también son llamados salidas de motor A y B.

En la parte inferior izquierda del módulo, tenemos 3 pines que son V_{in}, GND y 5V. Esos pines van a estar relacionados con la alimentación. El primero debe estar conectado directamente a la batería de 12V, el segundo es la entrada de tierra que debe tener dos conexiones, la primera al polo negativo de la batería de 12V y la segunda conexión irá directamente al pin de tierra de la placa Arduino, como también el

pin de 5V que irá directamente al pin de tensión de entrada Vin de Arduino.

Los pines que se encuentran a la derecha de los pines nombrados anteriormente son los encargados para gestionar los motores, que a su vez se dividen en dos tipos, empezando el recorrido de izquierda a derecha.

Los pines EN tanto A como B sirven para habilitar o deshabilitar los respectivos motores, y generalmente se utilizan para controlar la velocidad, ingresando una señal de PWM por esos pines.

Los pines IN, tanto 1,2,3,4 sirven para controlar la rotación de los motores, por lo que, si se analiza el motor A, si tenemos el pin IN1 en alto y el IN2 en bajo el motor A va a girar en un sentido y lo mismo ocurre con el motor B. Si tenemos el pin IN3 en alto y el pin IN4 en bajo, el motor B girará en una dirección, y si queremos que lo hagan en una dirección opuesta, simplemente cambiamos el estado de los pines.

Lo próximo será realizar las conexiones necesarias entre el módulo, la batería y la placa de Arduino. Dichas conexiones serán de la siguiente forma:

- Control del Motor izquierdo
    - Pin ENA L298N →Pin 7 de Arduino
    - Pin IN1 L298N →Pin 24 de Arduino
    - Pin IN2 L298N →Pin 26 de Arduino

- Control del Motor derecho
    - Pin ENB L298N →Pin 11 de Arduino
    - Pin IN3 L298N →Pin 12 de Arduino
    - Pin IN4 L298N →Pin 13 de Arduino
- Energía
    - Pin V_{in} L298N → Polo positivo de la batería de 12V
    - Pin V_{log} 5V L298N → Pin V_{in} de Arduino
    - Pin GND L298N → Pin GND de Arduino y también al polo negativo de la batería

El esquema de montaje final quedaría como se puede observar en la ***Figura 6.14*** , para que se pueda entender de manera más clara.

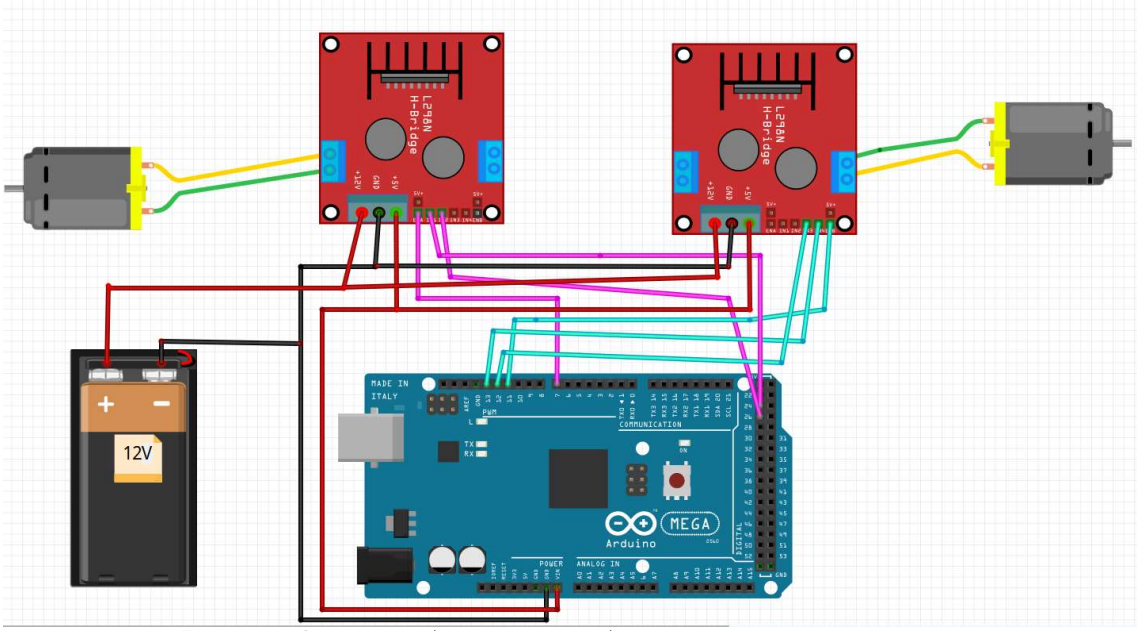


Figura 6.14 : Circuito completamente montado

El siguiente paso para poder realizar esta prueba es configurar Matlab, de tal forma que se puedan realizar trabajos con la placa Arduino, por lo que para ello se debe recurrir una vez más a la toolbox de Matlab e instalamos el soporte de Matlab para Arduino descrito en la ***Figura 6.15***.

Figura 6.15: Paquete de soporte Arduino

Con este paquete de soporte, se puede lograr el objetivo para que Matlab se comunique con Arduino de forma interactiva, pudiéndose leer utilizar muchas funciones de Matlab para analizar y visualizar datos.

Una vez instalado el componente, se dispone ahora a probarlo por lo que para ello creamos un script en Matlab llamado *pruebaMotor.m,* se probarán los componentes.

Lo primero que se debe hacer es declarar una variable donde se carga el objeto Arduino dentro de la variable *ar*

```
ar =
arduino('com6','Mega2560'); %Objeto Arduino
```

El objeto Arduino creado viene indicado por dos componentes que son el puerto serie utilizado que es en este caso el COM6 y el modelo de la placa que como se ha descrito anteriormente es el modelo Mega2560.
Una vez creado el objeto Arduino, el siguiente paso será ir desarrollando el rol que van a tener los pines descritos anteriormente con las conexiones establecidas, por lo que se debe tener en cuenta dos expresiones que son las siguientes:

- *writeDigitalPin(ar,pin,value) :* Esta instrucción posee tres parámetros, el primero es una llamada la placa arduino declarada en una variable *ar* , al que se debe especificar el pin y un valor concreto entre 0 y 1. Será aquí donde se controle la rotación de los motores por lo que los pines para saber hacia dónde se desea desplazar.

- *writePWMDutyCycle(ar,pin,dutyCycle):* Esta instrucción como el caso anterior posee 3 parámetros , por lo que el funcionamiento será parecido al anterior, salvo que en el
  parámetro dutyCycle que adquiere también un valor entre 0 y 1 será el encargado de controlar la velocidad del motor

A continuación, escribimos el siguiente script de prueba de nombre *PruebaMotor.m*:

```
% Motor A
writeDigitalPin(ar,'D12',1);
writeDigitalPin(ar,'D13',0);
writePWMDutyCycle(ar,'D11',0.6)%Velocidad del motor
% Motor B
writeDigitalPin(ar,'D26',1);
writeDigitalPin(ar,'D24',0);
writePWMDutyCycle(ar,'D7',0.6)%Velocidad del motor
```

También se ha probado en el caso de asignar una velocidad paulatina en ascenso mediante un bucle for :

```
% Motor A
writeDigitalPin(ar,'D12',1);
writeDigitalPin(ar,'D13',0);
% Motor B
writeDigitalPin(ar,'D26',1);
writeDigitalPin(ar,'D24',0);

for d=0:0.05:1
writePWMDutyCycle(ar,'D11',d) %Velocidad del motor A
writePWMDutyCycle(ar,'D7',d) %Velocidad del motor B
```

```
end
```

Tras realizar estas pruebas, se ha comprobado que los elementos necesarios para comenzar el trabajo funcionan correctamente, por lo que se pasará ahora al siguiente apartado, relacionado con la implementación de las redes neuronales convolucionales

6.3. Redes neuronales Convolucionales

En este apartado se va a explicar todo lo relacionado con el trabajo realizado. Para empezar, tenemos se debe decidir qué es lo que se desea realizar. Se va a plantear dos modelos de soluciones que son o bien de clasificación o bien por regresión. El caso es obtener una red neuronal que permita tomar una decisión muy sencilla para que el robot pueda ir a izquierda, derecha o seguir recto, y para ello primeramente debemos generar un almacén de imágenes y para ello se debe generar las mismas, para posteriormente etiquetarlas y realizar el entrenamiento de la red

Por lo que lo primero que se debe hacer es obtener un video o un conjunto de imágenes por lo que se deberá generar un script que realice dicha actividad.

De inicio se prueba a desarrollar el siguiente código dentro de un script llamado *CapturaImagnes.m*, por el cual lo que hacemos es capturar una serie de imágenes dentro de un bucle cada x segundos para luego guardarlo en un directorio en formato

.jpg. Este código es bastante sencillo de programar, pero como se verá después también se pueden grabar videos utilizando simulink como podemos ver en la ***Figura 6.17.***

```
%CapturaImagnes.m
clc;clear;close all;
cam = webcam(1); % Selección de la cámara y asignar a
una variable
k=1;
while 1
    im = snapshot (cam); % captura de imagen y
guardarla en al variable im
    imshow(im) % ver la imagen capturada

Fichero=strcat('.\imagenes\Imagen',num2str(k),'.jpg');
% guardar la ruta de la imagen
    imwrite(im,Fichero) %guardar la imagen en un
fichero
    pause(1)
    k=k+1;
end
```

Otra opción es generar un programa en simulink que permita grabar un video, para ello abrimos el simulink, del cual hay que fijarse primeramente en el siguiente trozo en el cual se puede apreciar tres cajas que se va a describir a continuación:

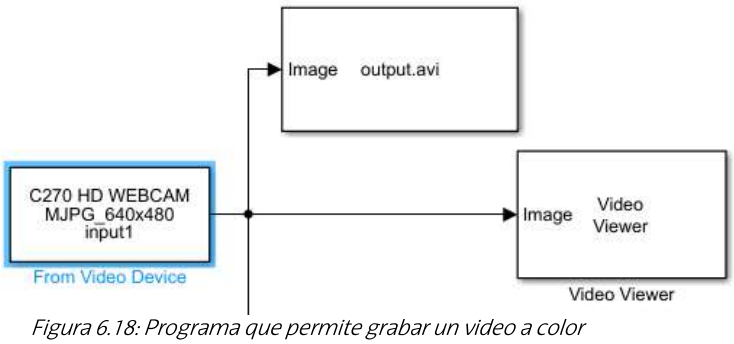


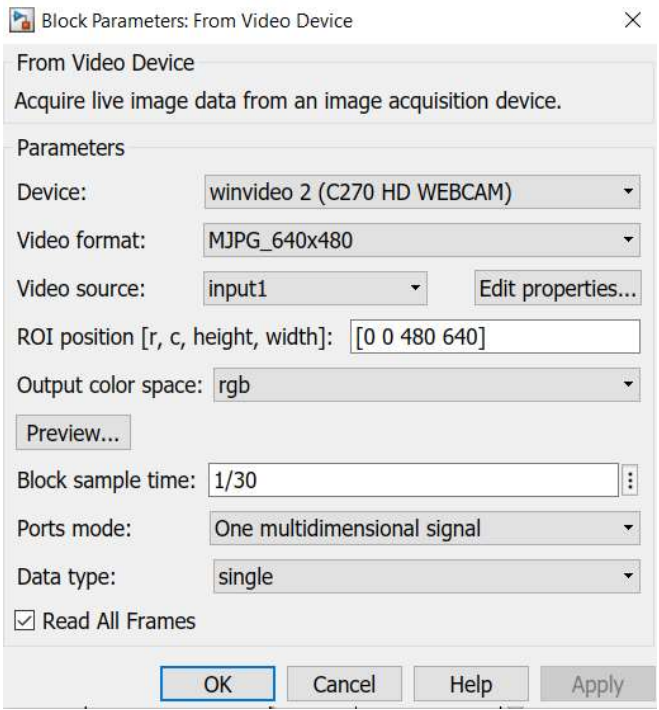
Figura 6.18: Programa que permite grabar un video a color

- Primero se tiene un Imput que es la entrada por la cual vamos a grabar el video. Como se puede apreciar en la entrada lo que se hace es cargar la cámara con al cual se
  quiere grabar, que en este caso es la C270 de Logitech, se introduce el formato al que se desea grabar y finalmente la resolución que en este caso es 640x480. Hay que decir también que se puede grabar en escala de grises por si es necesario hacerlo. El menú de configuración se puede ver en la ***Figura 6.19***

Figura 6.19: Parámetros de entrada

- En la siguiente caja tenemos el Video Viewer que permite mostrar imágenes o cuadros de videos

- Y por último se tiene la salida en donde guardamos el video y en el



formato en el que se desee.

Y es a partir de aquí cuando se va a ir por dos caminos diferentes para facilitar la comprensión del trabajo. Primero se va a realizar un entrenamiento de la red, que en este caso es una VGG-16 utilizando el método de transferencia de aprendizaje, por el cual el objetivo es introducir algunos datos y tiempo de computación, pero mucho

menos que hacerlos desde 0 que conllevaría mucho tiempo de cómputo y una enorme cantidad de datos de entrenamiento por lo que se adaptará dicha red a las necesidades necesarias.

6.3.1. Primera Caso: Sin referencia por Clasificación

Una vez obtenido el video por el cual se va a entrenar una red neuronal, lo primero que se debe realizar es el etiquetado. En esta primera prueba se va realizar un etiquetado de cara a realizar una operación de clasificación.

Para ello se va a desarrollar un script que realice un etiquetado por el cual en función de la dirección que se vaya a realizar, según donde apuntemos o queramos ir, dichas imágenes sean almacenadas en unas determinadas carpetas.

Lo primero que se va a desarrollar es el script *Etiketado.m*

```
%Etiketado.m
clc
close all
clear all
%Etiquetado manual
v =
VideoReader('D:\PruebasVGG16\PrimeraPrueba\output.avi'
);
xtotal=[];
ytotal=[];
refImagen=[];
for i=1:v.NumFrames
   imagenframe = read(v,i);
   imshow(imagenframe);
    [x,y,button] = ginput(1)
    refImagen=[refImagen,i];

xtotal=[xtotal,x];

 ytotal=[ytotal,y];
end
save('Datos_etiquetado')
```

En este script lo que se hace es que por medio del video grabado previamente se va a transformar el video en múltiples imágenes por

el cual se va a guardar tanto dichas imágenes obtenidas, pero a su vez el programa va mostrando una imagen como la que se puede ver en la ***Figura 6.20***, esto ayuda para saber la dirección a la que se quiere ir con el robot. Para ello es de importancia la función *ginput* para trabajar con un identificador de coordenadas, con el objetivo de facilitar la marcación de las coordenadas como se puede ver en la misma ***Figura 6.20*** y además se puede controlar las coordenadas marcadas en el entorno de comandos de Matlab como se puede apreciar en la ***Figura 6.21***

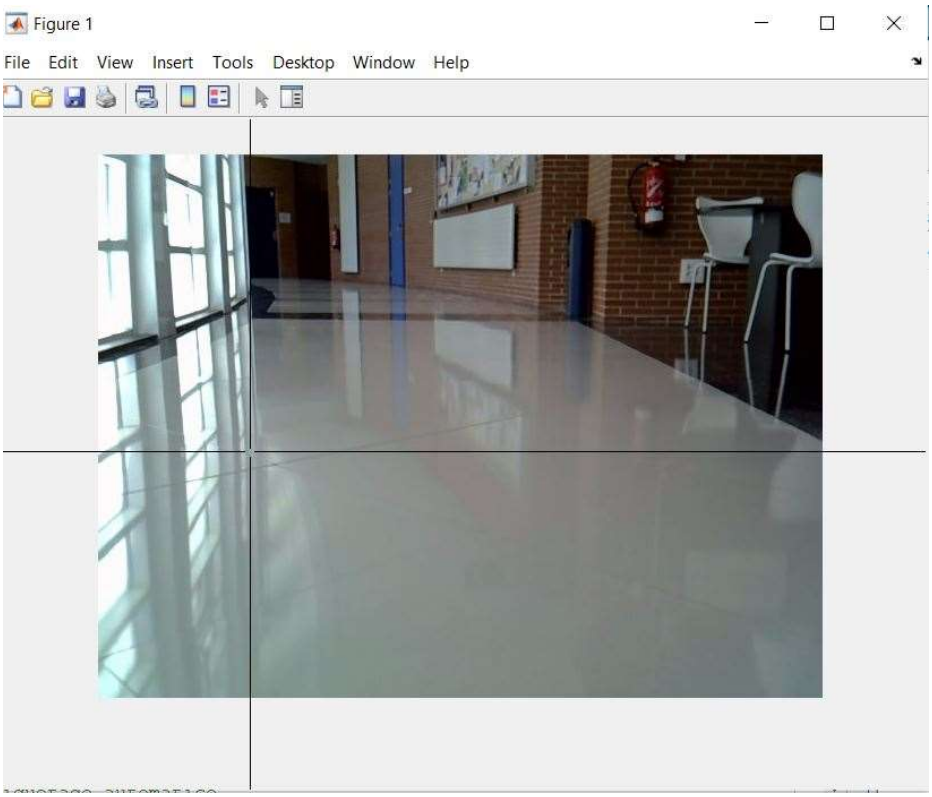


Figura 6.20: Imagen a Etiquetar

Figura 6.21: Coordenadas

Entonces a medida que se va marcando las coordenadas, estas se van almacenando en dos variables X, Y, para luego poder realiza el etiquetado. Hay que remarcar que este tipo de etiquetado es un etiquetado manual, es decir es la propia persona la que va realizando la acción.

El siguiente descomponer cada imagen etiquetada, mediante el siguiente script llamado *Descomponer.m :*

```
%Descomponer
load('Datos_etiquetado.mat')

v = VideoReader('output.avi');
valid_frames=(ytotal>0) | (xtotal>0);
recto = ytotal>100 ;
derecha = xtotal>500 ;
izquierda= xtotal<250;


pathR='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\re
cto';
pathD='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\de
recha';
pathI='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\iz
quierda';



for i=1:v.NumFrames
  imagenframe = read(v,i);
   if valid_frames(i)
       if recto (i)
        img_name=sprintf('img_%04d.jpg',i);
        img_path=fullfile(pathR,img_name);
        imwrite(imagenframe,img_path);

 rawlabel(i)=xtotal(i);
```

```
        elseif derecha (i)


        img_name=sprintf('img_%04d.jpg',i);
        img_path=fullfile(pathD,img_name);
       imwrite(imagenframe,img_path);
       rawlabel(i)=xtotal(i);

       elseif izquierda (i)

       img_name=sprintf('img_%04d.jpg',i);
       img_path=fullfile(pathI,img_name);
        imwrite(imagenframe,img_path);
         rawlabel(i)=xtotal(i);


        end
    end

end

beep

save('rawlabel','rawlabel')
```

En el siguiente script lo que hace es primeramente cargar los datos etiquetados que se han almacenado en las variables descritas en el script anterior tales como las coordenadas

Con las coordenadas guardadas ahora se va a recorrer el video para tratar de identificar cuando se realiza un giro a la izquierda o cuando se hace un giro a la derecha, y para ello se hace referencia a los ejes de coordenadas x e y. Si se observa en la ***Figura 6.20 y 6.21***.

Se define que cuanto más arriba se seleccione la coordenada en el eje *y* ,se tomará en cuenta que se va en dirección recta.

Para el caso de los giros, se tomará como referencia el eje de las x , por lo que cuanto más pequeño sea el valor de x , se tomará en cuenta que el robot se dirige hacia la izquierda, mientras que cuanto más alto sea el valor de x , se tomará en cuenta que el robot se dirige hacia la derecha, como se puede ver en la ***Figura 6.22.***

Para que se pueda por lo tanto separar en diversos directorios independientes de cara al entrenamiento, se toma en cuenta tres variables que serán las tres direcciones planteadas como se puede apreciar en el código *Descomponer.m*, y plasmándose en la imagen como se puede observar en la ***Figura 6.22***.

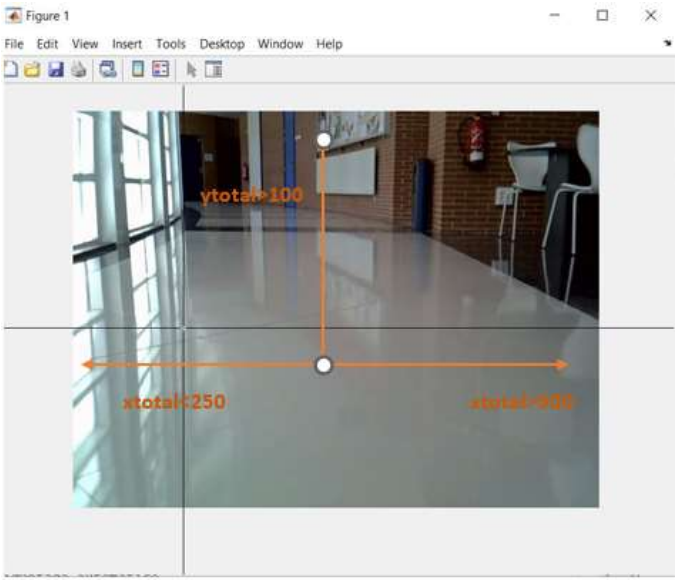


Figura 6.22: Limite de direcciones de los ejes (x, y)

También es importante considerar que algunas imágenes no hayan podido salir según la calidad que se hubiera deseado porque bien queda en un estado muy borroso o inteligible, por lo que se debería generar un mecanismo para desechar esas imágenes. Para ello en el código se ha puesto la instrucción de que se desecharan las imágenes que tengan valores negativos, y para ello en se deberá apuntar en la esquina superior izquierda para marcar dicha imagen con coordenadas negativas.

Es importante también guardar la ruta en tres variables para poder generar los directorios posteriormente, por lo que se definen las variables *path,* adaptadas para cada dirección del robot.

En el código finalmente mediante condicionales se consideran solo las imágenes con coordenadas positivas, y además mediante condicionales también, se decide que imagen en función de las variables previamente preestablecidas serán tanto de izquierda, derecha o recto, generando los directorios que se pueden ver en la ***Figura 6.23***

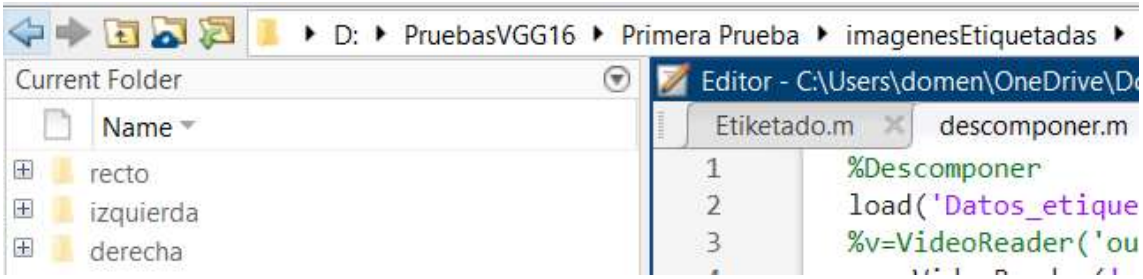


Figura 6.23: Directorios Generados

Teniendo ya las imágenes en diversos directorios, el siguiente paso es comenzar el entrenamiento. Para ello se va a definir que red neuronal se va a utilizar, que es el caso de este trabajo.

Para el siguiente caso se va a hacer uso de la herramienta *Deep Network Designer* se selecciona la estructura de la red neuronal convolucional que se va usar, y si en caso no se encontrarse la red que se desea, simplemente lo que se debería hacer es instalarlo a través del Toolbox de Matlab como se puede apreciar en la *Figura 6.24*.

Figura 6.24: Deep Learning Toolbox Model for VGG-16

Tras tener lista la red a disposición, lo siguiente que se va a realizar es la transferencia de aprendizaje por lo que se adaptará una red neuronal convolucional al problema a resolver, modificando algunas capas sobre todo las últimas.

Antes de realizar el entrenamiento, es importante crear un *datastore o* un almacén de datos para poder enviarlas a la red para realizar el entrenamiento, por lo que se debe crear un pequeño script llamado *crearDatastore.m*

```
clc
clear all;

dsC=imageDatastore('imagenesEtiquetadas\','IncludeSubfolders',true,'LabelSour
ce','foldernames');
audsC = augmentedImageDatastore([56 56],dsC);
```

En el código lo que se hace es crear un almacén de imágenes que para ser entrenadas. Dentro de la función *imageDatastore*, se incluyen las opciones de seleccionar las subcarpetas dentro de la carpeta donde tenemos las imágenes etiquetadas, y tomar como etiquetas el nombre de los directorios, para finalmente con la función

augmentedImageDatastore hacer un reajuste de las dimensiones para que tengan la misma que la entrada de la red neuronal convolucional.

El paso que se va a llevar a cabo es el siguiente:

Primero como lo que se hace es seleccionar una red vgg-16 y se modifican las últimas capas, como se puede ver en la ***Figura 6.24***

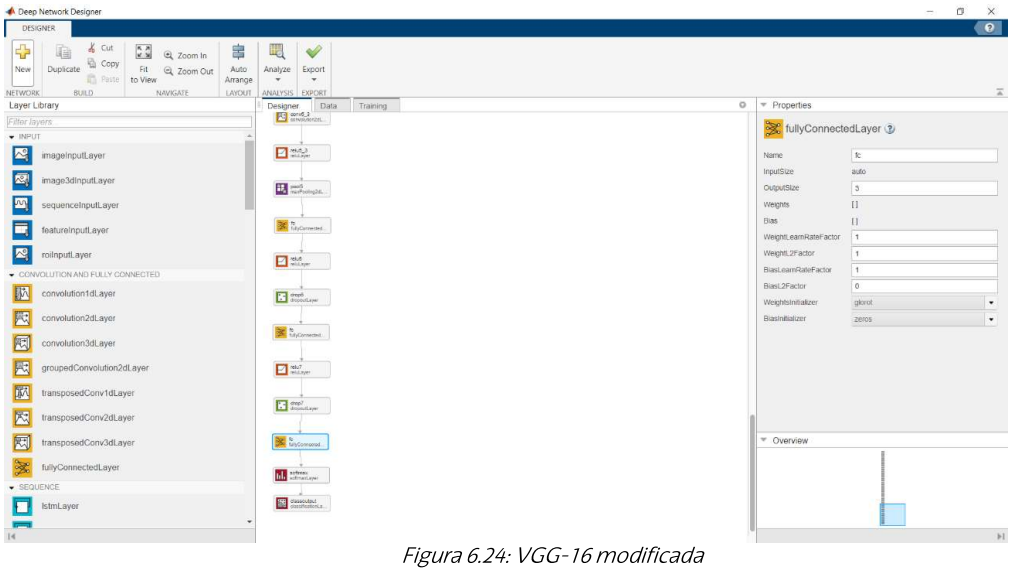


Figura 6.24: VGG-16 modificada

En este caso, como se tiene 3 etiquetas de izquierda, derecha y recto, la salida de la red *fully connected* será de 3. Es importante ir a la opción *Analyze* para ver que la modificación hecha es correcta antes de intentar introducir los datos.

Tras ingresar los datos en la red, aparece por pantalla una información que puede ser crucial a la hora de que tenga éxito el entrenamiento como se podrá ver en la ***Figura 6.25***

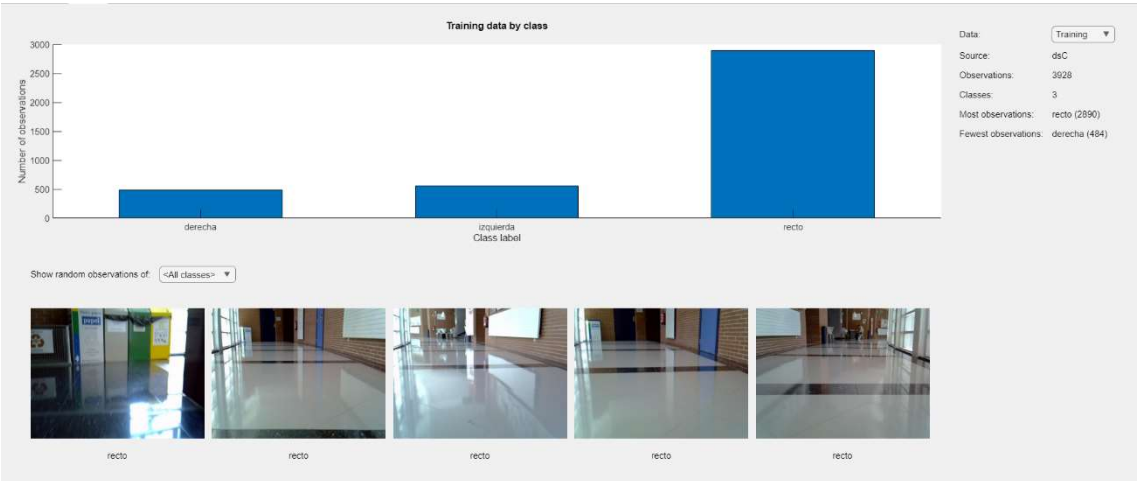


Figura 6.25: Carga de datastore1

Como se puede apreciar, los datos tienen un problema ya que están completamente desbalanceados, ya que con esta proporción de datos es difícil que se logre entrenar una red neuronal. Y como se puede apreciar en la ***Figura 6.25*** se corrobora que no se ha conseguido que la red neuronal aprenda nada, solo viendo la tasa de pérdidas.

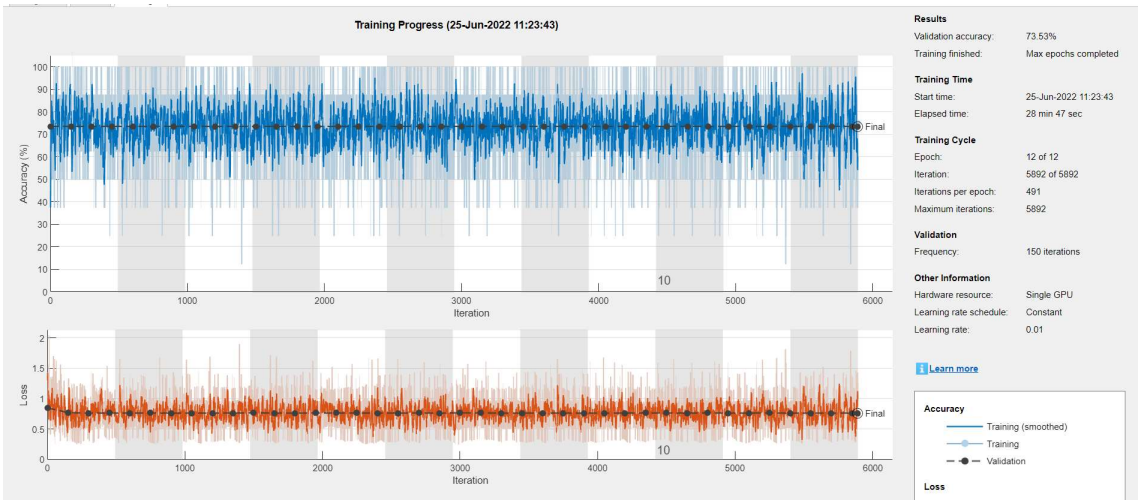


Figura 6.26: Etapa de entrenamiento

Finalmente se puede generar el código con la opción export, para revisar todos los pasos llevados a cabo.

Posteriormente se realiza otra prueba y esta vez con el siguiente resultado como se puede ver en la ***Figura 6.27***

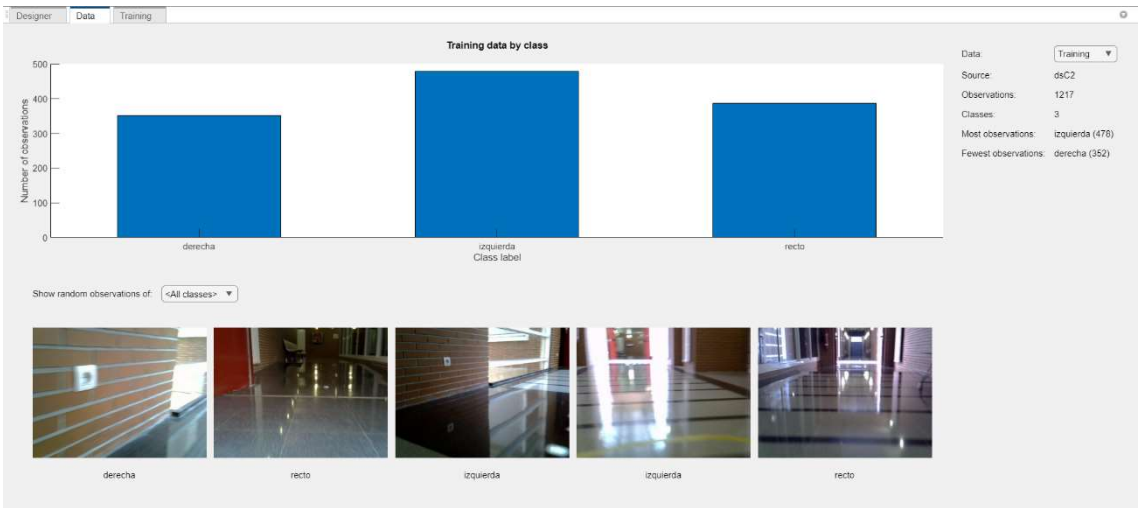


Figura 6.27: Carga Datastore 2

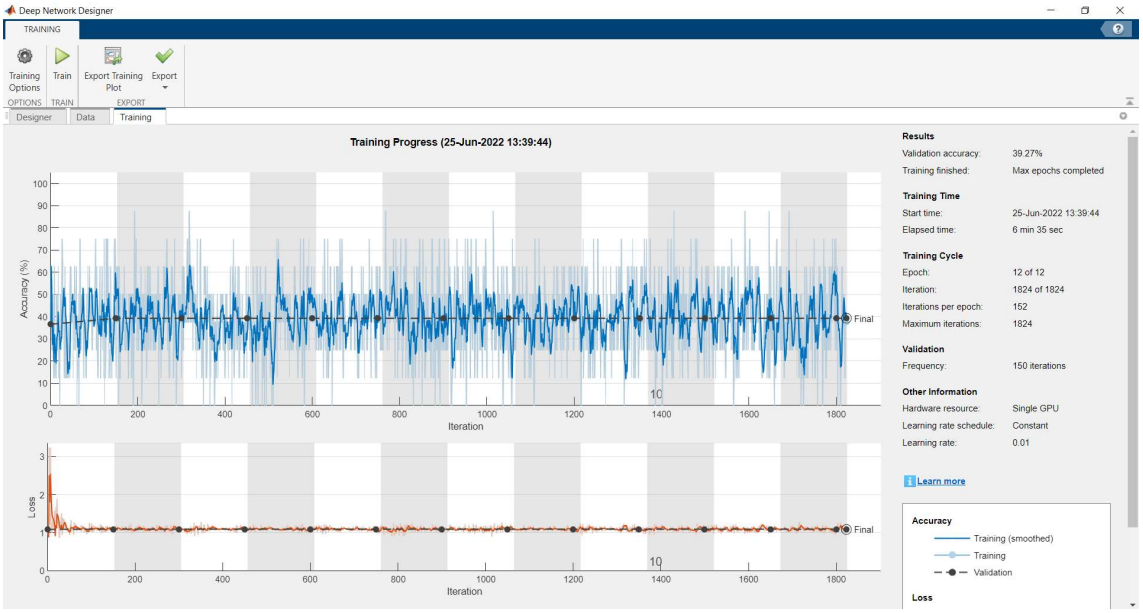


Figura 6.28: Entrenamiento

Se puede observar que los resultados obtenidos no son satisfactorios, ya que es muy difícil que se pueda realizar una clasificación de unos movimientos, es decir que la red pueda entender cuando está girándose hacia la izquierda o hacia la derecha. En el anexo se podrá encontrar el código generado, bajo título *EntrenamientoClasificación.m*

6.3.2 Segundo Caso: Sin referencia por Regresión

En el caso de realizarlo por regresión, la filosofía es totalmente distinta, ya que lo que se quiere no es clasificar direcciones del vehículo, como se ha planteado en el anterior caso mediante directorios, sino que se va a trabajar con coordenadas, por lo que el programa de etiquetado de video de imágenes debe ser modificado.

Hay que tener en cuenta ahora que por cada imagen va a tener que ir acompañado de un punto concreto, y en el caso de los datos a introducir, se pueden hacer de varias maneras.

El código se llama *EtiketadoRegresion.m* y el el siguiente:

```
clc
close all
clear all
%Etiquetado manual

v = VideoReader('D:\PruebasVGG16\Pruebita\video\video.avi');

xtotal=[];
```

```
                                        ytotal=[];
refImagen=[];
%coordenadas
coordenadasL=[];
for i=1:v.NumFrames
   imagenframe = read(v,i);
   imshow(imagenframe);
    [x,y,button] = ginput(1)
    refImagen=[refImagen,i];
    xtotal=[xtotal,x];
    ytotal=[ytotal,y];
    coordenadasL = [xtotal,x;ytotal,y];
end
save('Datos_etiquetado')
```

En este caso a partir de un video, se va capturando cada uno de las coordenadas (x,y) en la cual se desea dirigir. Este es el único cambio que hay respecto a la clasificación. A medida que avanza el script, irán pasando todas las imágenes y en las mismas se va seleccionando hacia donde se desea ir, y todo ello se almacena en la variable coordenadasL.

Tras terminar este paso, el siguiente paso a realizar utilizar el programa ya desarrollado previamente llamado *Descomponer.m* , aplicando una pequeña modificación por la cual ya no necesitamos diferenciar valores de izquierda, derecha o ir recto, sino solamente que se necesita imágenes con coordenadas asociadas.

Para evitar posibles confusiones, llamaremos a este script modificado *DescomponerRegresion.m:*

```
%DescomponerRegresion.m
load('Datos_etiquetado.mat')

v = VideoReader('output1.avi');
valid_frames=(ytotal>0) | (xtotal>0);

path='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas';

for i=1:v.NumFrames
   imagenframe = read(v,i);

 if valid_frames(i)
        img_name=sprintf('img_%04d.jpg',i);
        img_path=fullfile(pathL3,img_name);
        imwrite(imagenframe,img_path);
%        rawlabel(i)=xtotal(i);
   end

end
save('rawlabel','rawlabel')
```

El programa es una versión más sencilla que la vista anteriormente con el formato de clasificación. El programa recibe un video y lo que se hace es convertir en imágenes, descartando siempre las imágenes que no estén bien definidas o salgan borrosas,

descartando las coordenadas negativas. Es posible observar de entrada que solo estamos descargando

las fotos, por lo que posteriormente se necesitaría crear un tipo de dato que enlace las imágenes con las coordenadas.

Para el siguiente paso es conveniente retroceder código *EtiketadoRegresion.m* , en el cual guardamos las coordenadas. El próximo paso va a ser eliminar las coordenadas negativas que han sido descartadas por diversos motivos. Para ello se debe utilizar el siguiente script sencillo llamado *CoordenadasPositivas.m*:

```
%CoordenadasPositivas
f1 = coordenadas(1,:);
f1pos=f1(f1>0);
f2 = coordenadas(2,:);
f2pos=f2(f2>0);

coordenadasP(1,:)=f1pos;
coordenadasP(2,:)=f2pos;
```

Así con esto obtenemos solamente las coordenadas positivas, para después trabajar con ellas. El siguiente paso va a ser crear un datastore como se realizó en el caso anterior, pero en esta ocasión se va realizar de manera diferente, ya que se debe

recordar que se tiene que tener en cuenta los puntos de coordenadas. Primero se va a dar un pequeño salto hacia delante y abriendo el Deep Network Designer, seleccionado la red VGG-16, apreciando la ***Figura 6.29***, se puede ver la entrada de una

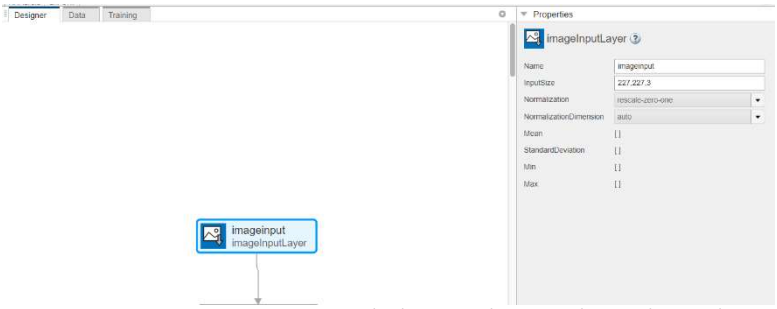


Figura 6.29: Entrada de una red neuronal convolucional

red neuronal convolucional que en este caso es de 227,227,3 es decir indica el tamaño de la imagen aceptada como el tipo de imagen por si es o bien de color o escala de grises. En este caso es de color ya que aparece el número 3. Recordando también la teoría de redes convolucionales, se puede aplicar la normalización de 0 a 1, para evitar que los valores de los pixeles se desborden. Hay que añadir que también se pueden poner otros valores, como por ejemplo 56,56,3.

El siguiente paso va a ser juntar las imágenes con sus respectivas coordenadas, para ello dentro de un mismo código al que se llamará *Ejecución*.m, se tendrán que ejecutar

una serie de instrucciones, en las que en el Anexo se pondrá de manera completa mientras que aquí se irá analizando paso por paso.

Lo primero que se hace es ajustar las coordenadas para que estén en la misma resolución mediante el siguiente código, llamado *ResizeCoordandas.m*:

```
%Resize de coordenadas
 filax = coordenadasP(1,:);
 calculo1 = round(filax * (227/640));
 filay = coordenadasP(2,:);
 calculo2 = round(filay * (227/640));
```

En el caso anterior, el entrenamiento se realizó mediante el uso del Deep Network Designer, que al finalizar el entrenamiento permite exportar el código generado por el mismo.

En este caso se va a realizar dicho entrenamiento a mano, mediante la programación del código, para entender algunos conceptos que aparecen, sobre todo a la hora definir las opciones.

El siguiente paso es crear el datastore, para ello dentro del código *Ejecucion.m* se debe ejecutar lo siguiente:

```
%Datos
imgpath =
'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas';
imDsL = imageDatastore(imgpath);
targetSize=[227,227];
writematrix(coordenadasP','coordenadasP','FileType','spreadsheet');
ssd2rL =  spreadsheetDatastore('coordenadasP.xls');

ssd2rL.ReadSize=1;

Tssds2RL = transform(ssd2rL,@(x) table2array(x));
imdsReSzL = transform(imDsL,@(x) customResize(x,targetSize));
comDS2RL=combine(imdsReSzL,Tssds2RL);


%Datos de Validación
imgpathV = 'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\validacion';

imDsLV = imageDatastore(imgpathV);
writematrix(coordenadasv','coordenadasv','FileType','spreadsheet');
ssd2rLV = spreadsheetDatastore('coordenadasv.xls');ssd2rLV.ReadSize=1;

Tssds2RLV = transform(ssd2rLV,@(x) table2array(x));
imdsReSzLV = transform(imDsLV,@(x) customResize(x,targetSize));
comDS2RLV=combine(imdsReSzLV,Tssds2RLV);
```

Lo que se hace con este código en ambos casos es lo mismo. Lo que se hace primero es crear un almacén de datos de todas las imágenes que se van a utilizar de cara al entrenamiento, para poder luego hacer un resize de las imágenes.

Posteriormente se trabaja con las coordenadas, para ponerlas en formato de tabla para posteriormente combinarlas. Para ver que la ejecución es correcta, se debe ejecutar el comando read sobre la combinación de las imágenes y las coordenadas tal y como se puede ver en la ***Figura 6.30.***

```
>> read(comDS2RL)

ans =

  1×2 cell array

    {227×227×3 uint8}    {[63 191]}
```

Figura 6.30 : Datos concatenados correctamente

El siguiente paso es definir las opciones de entrenamiento, de las cuales se va a explicar que son cada uno de los argumentos seleccionados

```
%Opciones
options = trainingOptions('sgdm', ...
    'MaxEpochs',8,...
    "MiniBatchSize",4,...
    'InitialLearnRate',1e-6, ...
    'Shuffle','every-epoch',...
    'Verbose',false, ...
    'Plots','training-progress',...
    "Verbose",true,...
    "VerboseFrequency",50,...
    "ValidationData",comDS2RLV,...
    "ValidationPatience",200);
```

El primer valor llamado sgdm es *stocastic gradient descend method*, que lo que hace es calcular el gradiente del error y estocástico es que va a ir cogiendo cada uno de los

patrones y va a calcular para cada uno de los patrones el gradiente, y se le denomina estocástico porque si vas evaluando en vez de todos los patrones al mismo tiempo y se calcula su gradiente y los acumulas en uno, se dice que va actualizando cada patrón lo que hace que el gradiente sea muy aleatorio.

El siguiente valor es el número máximo de épocas llamado *MaxEpochs*, el *MiniBatchSize* se refiere al tamaño del lote de imágenes que ingresa en la CPU.

El siguiente valor es el *InitialLearnRate* se refiere al índice de aprendizaje inicial o tasa de aprendizaje inicial. A menor cantidad, más cantidad de iteraciones serán necesarias para modificar los valores del sistema.

El siguiente parámetro es el *Shuffle* o barajeo en el que se ha seleccionado la opción every-epoch en la cual los datos se barajean antes de cada época. También posee las opciones none y once en la cual la primera no se barajea nada y la anterior se barajean los datos antes de entrenar

La *ValidationData y ValidationPatience* se refieren a los datos de validación, seleccionado una pequeña cantidad de datos del general , y la siguiente es referente a la cantidad de veces que la perdida en el conjunto de validación puede ser mayor o

igual a la pérdida pequeña anterior, antes de que se detenga el entrenamiento de la red.

Finalmente se deben de definir las la estructura de red neuronal, que en este caso es la VGG-16, adaptándola para el caso de una regresión lineal.

Lo que se hace es definir la red con el Deep Network Designer, para luego exportarla como código quedando lo siguiente. Para ello se modifica una VGG-16 en las últimas capas, añadiendo una fully connected al final con dos de salida y además una regression layer ya que se va a trabajar con regresión. También se normaliza la entrada de las imágenes

```
%Layers
layers = [

     imageInputLayer([227 227
3],"Name","imageinput","Normalization","rescale-zero-
one","Min",0,"Max",255)
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu1_1")
    convolution2dLayer([3 3],64,"Name","conv1_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu1_2")
    maxPooling2dLayer([2 2],"Name","pool1","Stride",[2 2])




  convolution2dLayer([3 3],128,"Name","conv2_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu2_1")
    convolution2dLayer([3 3],128,"Name","conv2_2","Padding",[1 1 1
1],"WeightL2Factor",0)
```

```
                                    reluLayer("Name","relu2_2")
    maxPooling2dLayer([2 2],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv3_1","Padding",[1 1 1
1],"WeightL2Factor",0)

    reluLayer("Name","relu3_1")
    convolution2dLayer([3 3],256,"Name","conv3_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu3_2")
    convolution2dLayer([3 3],256,"Name","conv3_3","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu3_3")
    maxPooling2dLayer([2 2],"Name","pool3","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv4_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu4_1")
    convolution2dLayer([3 3],512,"Name","conv4_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu4_2")

    convolution2dLayer([3 3],512,"Name","conv4_3","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu4_3")
    maxPooling2dLayer([2 2],"Name","pool4","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv5_1","Padding",[1 1 1
1],"WeightL2Factor",0)

    reluLayer("Name","relu5_1")

 convolution2dLayer([3 3],512,"Name","conv5_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu5_2")


 convolution2dLayer([3 3],512,"Name","conv5_3","Padding",[1 1 1
1],"WeightL2Factor",0)

  reluLayer("Name","relu5_3")
    maxPooling2dLayer([2 2],"Name","pool5","Stride",[2 2])

fullyConnectedLayer(100,"Name","fc_2","BiasLearnRateFactor",1,"WeightLear
nRateFactor",1)
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")

fullyConnectedLayer(10,"Name","fc_3","BiasLearnRateFactor",1,"WeightLearn
RateFactor",1)

reluLayer("Name","relu7")
    dropoutLayer(0.5,"Name","drop7")
```

```
fullyConnectedLayer(2,"Name","fc_1","BiasLearnRateFactor",1,"WeightLearnR
ateFactor",1)
    regressionLayer("Name","regressionoutput")];
```

Para ejecutar el entrenamiento para comenzar el entrenamiento, se ejecuta la siguiente instrucción en la que se tiene los datos, las capas y las opciones:

```
[PruebaRegresion,info] = trainNetwork(comDS2RL, layers, options);
```

Finalmente se ha desarrollado siguiente función que se ejecuta dentro del propio código. Dicha función lo que hace es solucionar un pequeño error a la hora de combinar las imágenes con resize con las coordenadas

```
function out=customResize(x,targetSize)
    imRes=imresize(x,targetSize);
    dummycell=cell(1,1);
    dummycell{1,1}=imRes;
    out=dummycell;
end
```

Tras ejecutarlo, el resultado obtenido es el siguiente, tal y como se puede ver en las ***Figuras 6.31, 6.32, 6.33***

Figura 6.31 : Entrenamiento Regresión

```
Training on single GPU.
Initializing input data normalization.
|=========================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |     RMSE     |     RMSE     |     Loss     |     Loss     |      Rate       |
|=========================================================================================================|
|       1 |           1 |       00:00:07 |        90.11 |       158.57 |    4060.0862 |   12572.1191 |      1.0000e-06 |
|       1 |          50 |       00:00:22 |       107.79 |       158.40 |    5809.3091 |   12544.7588 |      1.0000e-06 |
|       1 |         100 |       00:00:38 |        92.82 |       156.15 |    4308.0825 |   12191.7373 |      1.0000e-06 |
|       1 |         150 |       00:00:53 |       122.21 |       113.88 |    7467.7212 |    6484.4448 |      1.0000e-06 |
|       1 |         200 |       00:01:09 |        38.38 |       143.18 |     736.4238 |   10249.6221 |      1.0000e-06 |
|       1 |         250 |       00:01:32 |        73.85 |       121.92 |    2726.8125 |    7432.2383 |      1.0000e-06 |
|       2 |         300 |       00:01:50 |        45.80 |       145.24 |    1049.0469 |   10547.2900 |      1.0000e-06 |
|       2 |         350 |       00:02:06 |       114.81 |       131.00 |    6590.8623 |    8580.8398 |      1.0000e-06 |
|       2 |         400 |       00:02:22 |        70.27 |       136.86 |    2469.2019 |    9365.6045 |      1.0000e-06 |
|       2 |         450 |       00:02:37 |        27.60 |       128.86 |     380.9177 |    8302.0566 |      1.0000e-06 |
|       2 |         500 |       00:02:55 |        85.03 |       105.50 |    3615.1987 |    5565.5781 |      1.0000e-06 |
|       3 |         550 |       00:03:10 |        49.31 |       147.88 |    1215.7809 |   10934.4453 |      1.0000e-06 |
|       3 |         600 |       00:03:26 |        92.80 |        95.13 |    4305.4707 |    4525.3232 |      1.0000e-06 |
```

Figura 6.32 : Obtención de resultados I

```
|       4 |         800 |       00:04:36 |        39.28 |       133.22 |     771.5164 |    8873.8027 |      1.0000e-06 |
|       4 |         850 |       00:04:51 |       190.57 |       132.00 |   18159.0879 |    8712.3848 |      1.0000e-06 |
|       4 |         900 |       00:05:06 |        34.86 |        98.25 |     607.6185 |    4826.3345 |      1.0000e-06 |
|       4 |         950 |       00:05:21 |        54.17 |       116.65 |    1466.9811 |    6803.8306 |      1.0000e-06 |
|       4 |        1000 |       00:05:36 |        67.97 |       120.15 |    2309.9851 |    7218.5918 |      1.0000e-06 |
|       4 |        1050 |       00:05:53 |        90.61 |       100.13 |    4104.8569 |    5013.4648 |      1.0000e-06 |
|       5 |        1100 |       00:06:08 |        69.59 |       127.25 |    2421.1797 |    8095.9600 |      1.0000e-06 |
|       5 |        1150 |       00:06:23 |        45.04 |        89.81 |    1014.2286 |    4032.9915 |      1.0000e-06 |
|       5 |        1200 |       00:06:38 |       107.54 |       107.95 |    5782.0845 |    5826.5806 |      1.0000e-06 |
|       5 |        1250 |       00:06:53 |        26.94 |       112.00 |     362.7880 |    6272.4707 |      1.0000e-06 |
|       5 |        1300 |       00:07:08 |        60.01 |        94.23 |    1800.7936 |    4439.9751 |      1.0000e-06 |
|       6 |        1350 |       00:07:23 |        60.01 |       114.46 |    1800.6714 |    6550.1812 |      1.0000e-06 |
|       6 |        1400 |       00:07:38 |       167.86 |        98.98 |   14088.9395 |    4898.3208 |      1.0000e-06 |
|       6 |        1450 |       00:07:53 |       110.16 |        95.08 |    6067.9316 |    4519.8154 |      1.0000e-06 |
|       6 |        1500 |       00:08:11 |        37.41 |       108.26 |     699.6364 |    5859.8726 |      1.0000e-06 |
|       6 |        1550 |       00:08:27 |       126.28 |       103.25 |    7973.6704 |    5330.1211 |      1.0000e-06 |
|       7 |        1600 |       00:08:47 |        78.38 |       106.55 |    3071.3301 |    5676.0825 |      1.0000e-06 |
|       7 |        1650 |       00:09:03 |        71.05 |       107.85 |    2524.3926 |    5816.2163 |      1.0000e-06 |
|       7 |        1700 |       00:09:18 |        51.97 |        91.45 |    1350.3883 |    4181.6230 |      1.0000e-06 |
|       7 |        1750 |       00:09:34 |        51.80 |       105.62 |    1341.7173 |    5577.6851 |      1.0000e-06 |
|       7 |        1800 |       00:09:49 |       148.84 |       112.69 |   11077.2451 |    6349.4302 |      1.0000e-06 |
|       8 |        1850 |       00:10:04 |        44.88 |        98.65 |    1007.1101 |    4866.3262 |      1.0000e-06 |
|       8 |        1900 |       00:10:19 |       152.72 |       114.16 |   11662.1729 |    6516.4194 |      1.0000e-06 |
|       8 |        1950 |       00:10:33 |        56.37 |        88.30 |    1588.8365 |    3898.0149 |      1.0000e-06 |
|       8 |        2000 |       00:10:48 |        38.29 |       101.68 |     733.2450 |    5169.3672 |      1.0000e-06 |
|       8 |        2050 |       00:11:03 |        48.76 |       112.94 |    1188.7939 |    6377.9214 |      1.0000e-06 |
|       8 |        2100 |       00:11:20 |       126.39 |        93.16 |    7987.5356 |    4339.3232 |      1.0000e-06 |
|       8 |        2104 |       00:11:26 |        96.57 |        95.12 |    4662.4800 |    4524.0742 |      1.0000e-06 |
|=========================================================================================================|
```

Figura 6.33 : Obtención de resultados II

La conclusión que se puede sacar de ambos casos es que se necesita tener principalmente una referencia para que la red pueda entender. En ambos casos se ha intentado realizar un entrenamiento para que entiendan una serie de direcciones, pero el resultado no ha sido el esperado. En el siguiente apartado lo que se va a hacer es un tercer caso, en este caso siguiendo una línea amarilla.

6.3.3 Tercer Caso: Con referencia por Regresión a través de un conjunto de puntos

La principal diferencia es con los casos anteriores es que se va a trabajar esta vez con una referencia, que será una línea amarilla.

Primero lo que se debe hacer es grabar un video del camino a seguir, para posteriormente comenzar el etiquetado.

En este caso se va a utilizar una toolbox de Matlab llamada *Image Labeler* que va a ayudar de esta manera a realizar el etiquetado de manera más clara y sencilla

Lo primero que se hace es definir una etiqueta, y para ello se define una línea, tal y como se puede ver en la *Figura 6.34.*

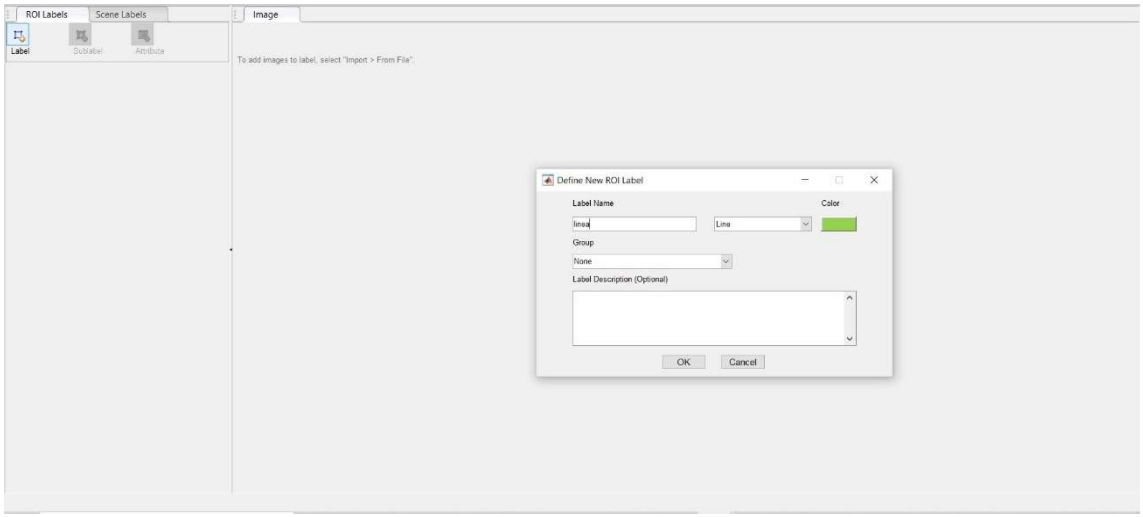


Figura 6.34: Se define la etiqueta como línea

El siguiente paso es cargar las imágenes, y comenzar el etiquetado, tal y como se puede ver en la ***Figura 6.35.***

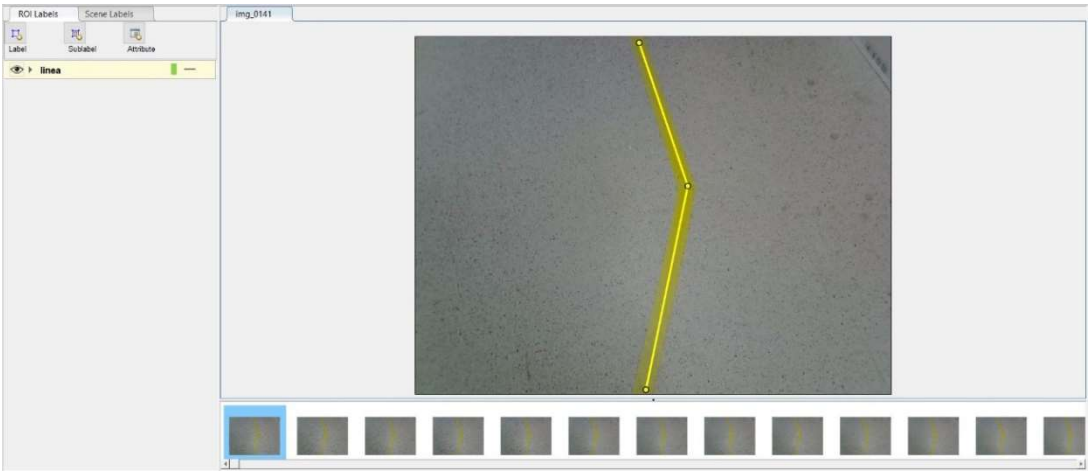


Figura 6.35: Etiquetado de las imágenes

Como suele darse muchas veces, hay imágenes que no están bien definidas o no nos aporta absolutamente nada o se tenga dudas de etiquetarla o no. La ventaja de esta herramienta es que se puede ir recorriendo y volver a revisar una imagen cuantas veces se desee, y además se puede comprobar qué imágenes han sido etiquetadas o no, revisando el *Label Summary* como se puede ver en la *Figura 6.36*.

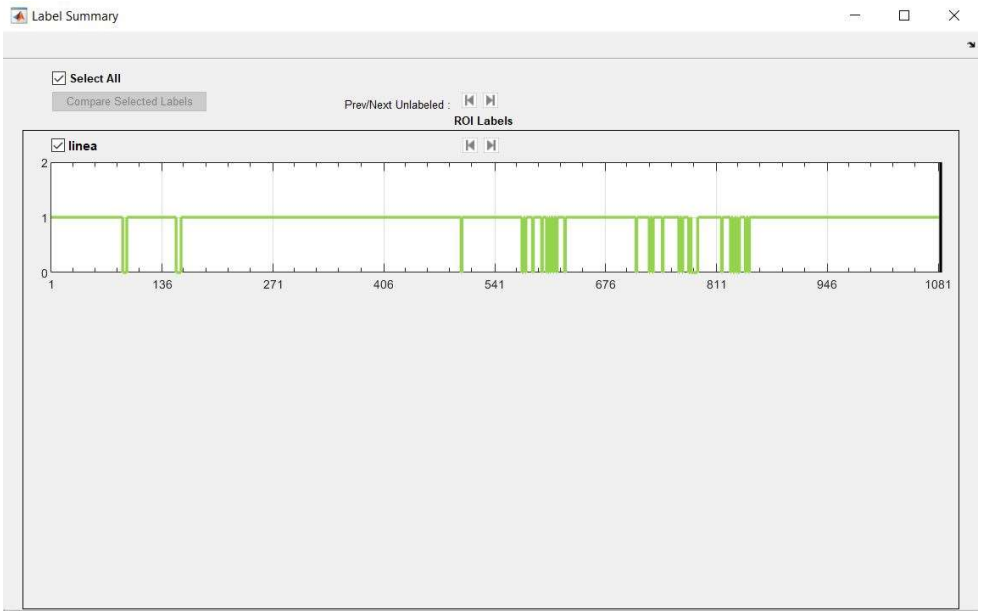


Figura 6.36: Detalle de las imágenes etiquetadas

Tras ello, el siguiente paso es exportar los datos, por el cual se puede hacer en dos formatos, que son como formato de tabla o bien en formato groundTruth seleccionado este último y asignándole un nombre, tal y como se puede ver en la Figura 6.37

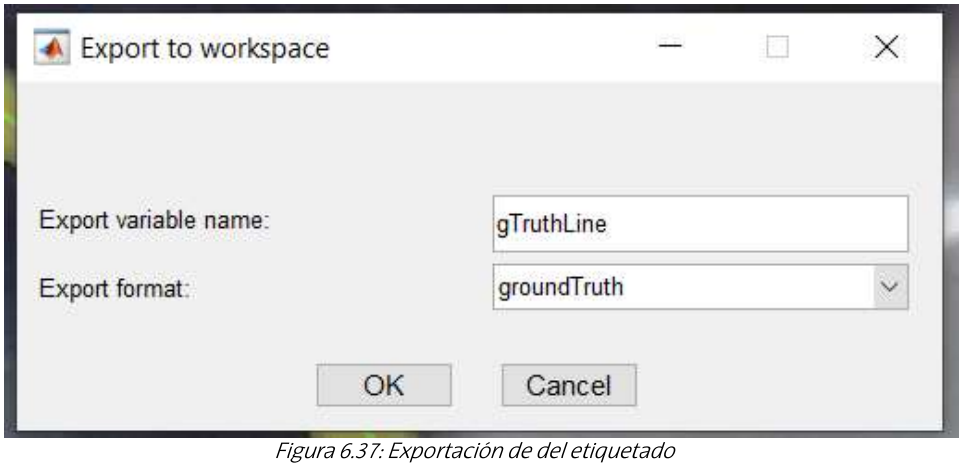


Figura 6.37: Exportación de del etiquetado

Tras exportarse, en el entorno de comandos de Matlab, va a aparecer la siguiente información relevante, dentro de un tipo de dato *groundTruth* en el que aparece la siguiente información, que se puede ver en la *Figura 6.38*

```
gTruthLine =

  groundTruth with properties:

         DataSource: [1×1 groundTruthDataSource]
    LabelDefinitions: [1×5 table]
           LabelData: [1052×1 table]

>>
```

Figura 6.38 : Tipo de dato groundTruth

- DataSource: Especifica el origen de los datos reales y establece la propiedad DataSource

- LabelDefinicions: Especifica las definiciones de etiquetas , subetiquetas y atributos de los datos reales y establece la propiedad LabelDefinitions

- LabelData: Espeficica la información de identificación, la posición y las marcas de tiempo para etiquetas marcadas y establece la propiedad LabelData

Durante el trabajo, como se ha visto antes, ImageLabeler es una herramienta bastante sencilla de utilizar, pero a su vez para el caso del trabajo que se desea realizar, ha generado una serie de dificultades que son las siguientes:

- Es una herramienta perfecta para identificar una serie de objetos, tales como por ejemplo tornillos, coches u otros objetos marcados, para detectar un camino es más complicado ya que Matlab tiene de momento pocas funciones por no decir ni una que ayude para detectar una serie de puntos consecutivos
- Si se centra en las coordenadas, cada punto es una coordenada x,y se debe tener especial cuidado con el número de puntos a la hora de poder tal y como aparece en el histograma siguiente (ver Figura 6.39), a poder ser se debe aplicar un mismo criterio de etiquetado como por ejemplo poner siempre 5 puntos.
- El tipo de dato a tratar es bastante engorroso, ya que por ejemplo las coordenadas están dentro de cellarrays, dentro de otro cellarray

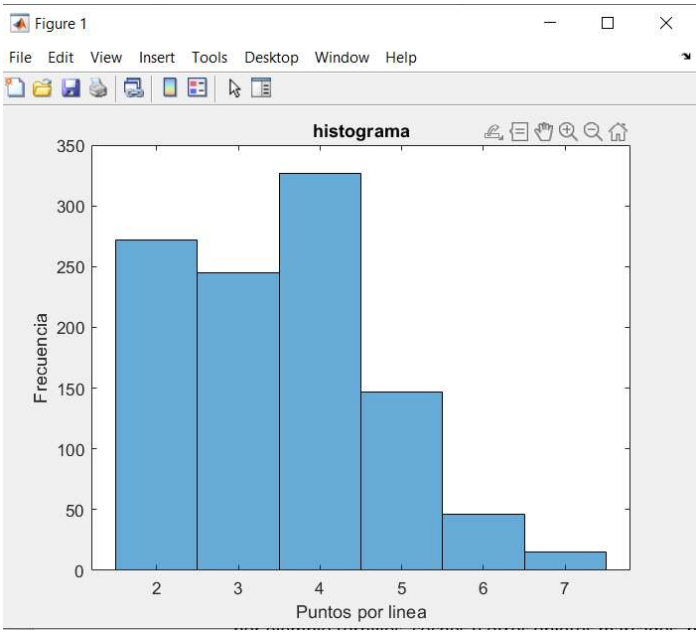


Figura 6.39: Histograma de puntos

Tras finalizar el etiquetado, y viendo las dificultades generadas, se decide realizar este ejercicio mediante otro tipo de solución. De todas formas se desarrolló un script llamado *clean_BBDD.m*, que permite resolver dos problemas importantes, el primero que es la extracción de los datos para enviarlos a un array dummy_Cell, que dado un elemento de tipo *groundTruth* , obtenemos esas coordenadas de las cellarrays. Como se comentó antes, también se aprovecha a normalizar los puntos para que todos tengan 5 puntos, es decir se hace una interpolación y además si en algunos casos hay 6 o 7 puntos lo que se hace es eliminar el primer y el último punto, quedando solo 5.

```
% load('gTruthLineYelow.mat')
%clean_BBDD.m
labels=table2cell(gTruthLineYelow.LabelData);
dummy_Cell=cell(length(labels),1);
num_puntos=zeros(length(labels),1);

for i=1:length(labels)
    clc
    display(i/length(labels)*100)

    puntos=labels{i,1}{1,1};
    if size(puntos,1)<5


    X=puntos(end-1:end,1);
    Y=puntos(end-1:end,2);

    N=2+(5-size(puntos,1));

    addX=linspace(X(1), X(2), N);
    addY=linspace(Y(1), Y(2), N);

    puntosfinal=[puntos(1:end-2,:);[addX',addY']];
```

```
dummy_Cell{i,1}=puntosfinal;

        if size(puntosfinal,1)~=5
            pause()
        end

num_puntos(i)=size(puntosfinal,1);

    elseif size(puntos,1)>5
    puntosfinal=puntos(1:5,:);

    dummy_Cell{i,1}=puntosfinal;
        if size(puntosfinal,1)~=5
            pause()
        end
    num_puntos(i)=size(puntosfinal,1);
    else
    puntosfinal=puntos;
    num_puntos(i)=size(puntosfinal,1);
    end


%     dummy_Cell{i,1}=[];
end
```

6.3.4. Cuarto caso: Con referencia por Regresión a través de un punto.

En este caso, el etiquetado se parece mucho al del caso de regresión. Se debe seleccionar el punto a donde se desea ir, pero con la diferencia de que en este caso se busca un punto sobre la línea de referencia.

En este caso se va a explicar el código por trozos para entender que es lo que se está realizando.

```
imgpath =
'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadasLinea2';
%imDsL = imageDatastore(imgpath);
targetSize=[64,64];
imDsL=imageDatastore(imgpath,'FileExtensions','.jpg','ReadFcn',@(filename)cus
tomResize(filename,targetSize));
load('c.mat')
c=c';
c(:,1)=(c(:,1)-min(c(:,1)))/(max(c(:,1))-min(c(:,1)));
c(:,2)=(c(:,2)-min(c(:,2)))/(max(c(:,2))-min(c(:,2)));
dataoutNormalized = arrayDatastore(c);
i=1;
comDS2RL=combine(imDsL,dataoutNormalized);
```

En esta primera parte del código, lo que se genera es los datos de entrada de la red neuronal convolucional. Para ello también se hace uso de una función llamada

customResize que lo que hace es reajustar las imágenes a un tamaño 64x64, además de normalizarlos. Después lo que se hace es cargar las coordenadas.

Finalmente se combinan los datos normalizados para que puedan ser introducidos sin problemas en la red neuronal.

```
reset(comDS2RL)
while hasdata(comDS2RL)
    data=read(comDS2RL);
    imshow(data{1,1})
    hold on

plot(data{1,2}(1)*targetSize(1),data{1,2}(2)*targetSize(2),'xr','MarkerSize',
5)
    i=i+1;
    hold off
     pause(0.1)
end
```

En la siguiente parte del código lo que se realiza es una ayuda de cara a si los datos se han tomado correctamente, como se puede ver en la ***Figura 6.40***. Para ello lo que se leen son los datos ya concatenados y se van pasando todas las imágenes calculando sus coordenadas marcadas

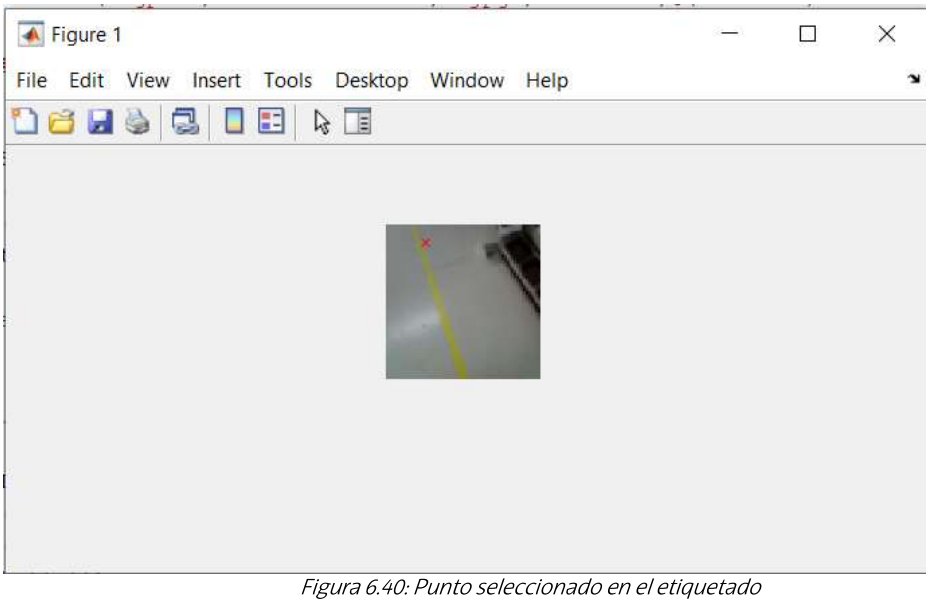


Figura 6.40: Punto seleccionado en el etiquetado

```
%Crear la red neuronal
params =
load("D:\PruebasVGG16\SeptimaPrueba\params_2022_06_23__12_16_07.mat");
layers = [
    imageInputLayer([64 64 3],"Name","imageinput")
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv1_1.Bias,"Weights",params.conv1_1.Wei
ghts)
    reluLayer("Name","relu1_1")
```

```
                                    convolution2dLayer([3
3],64,"Name","conv1_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv1_2.Bias,"Weights",params.conv1_2.Wei
ghts)
    reluLayer("Name","relu1_2")

  maxPooling2dLayer([2 2],"Name","pool1","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv2_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv2_1.Bias,"Weights",params.conv2_1.Wei
ghts)
    reluLayer("Name","relu2_1")
    convolution2dLayer([3 3],128,"Name","conv2_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv2_2.Bias,"Weights",params.conv2_2.Wei
ghts)
    reluLayer("Name","relu2_2")
    maxPooling2dLayer([2 2],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv3_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_1.Bias,"Weights",params.conv3_1.Wei
ghts)
    reluLayer("Name","relu3_1")
    convolution2dLayer([3 3],256,"Name","conv3_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_2.Bias,"Weights",params.conv3_2.Wei
ghts)
    reluLayer("Name","relu3_2")
    convolution2dLayer([3 3],256,"Name","conv3_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_3.Bias,"Weights",params.conv3_3.Wei
ghts)
    reluLayer("Name","relu3_3")
    maxPooling2dLayer([2 2],"Name","pool3","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv4_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_1.Bias,"Weights",params.conv4_1.Wei
ghts)
    reluLayer("Name","relu4_1")
    convolution2dLayer([3 3],512,"Name","conv4_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_2.Bias,"Weights",params.conv4_2.Wei
ghts)
    reluLayer("Name","relu4_2")
    convolution2dLayer([3 3],512,"Name","conv4_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_3.Bias,"Weights",params.conv4_3.Wei
ghts)
    reluLayer("Name","relu4_3")
    maxPooling2dLayer([2 2],"Name","pool4","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv5_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_1.Bias,"Weights",params.conv5_1.Wei
ghts)
    reluLayer("Name","relu5_1")
    convolution2dLayer([3 3],512,"Name","conv5_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_2.Bias,"Weights",params.conv5_2.Wei
ghts)
    reluLayer("Name","relu5_2")
    convolution2dLayer([3 3],512,"Name","conv5_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_3.Bias,"Weights",params.conv5_3.Wei
ghts)
    reluLayer("Name","relu5_3")
    maxPooling2dLayer([2 2],"Name","pool5","Stride",[2 2])
    fullyConnectedLayer(4000,"Name","fc_3")
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")
    fullyConnectedLayer(1000,"Name","fc_4")
```

```
reluLayer("Name","relu7")
    dropoutLayer(0.5,"Name","drop7")
    fullyConnectedLayer(10,"Name","fc_1")
    tanhLayer("Name","tanh_1")
    fullyConnectedLayer(2,"Name","fc_2")

tanhLayer("Name","tanh_2")
    regressionLayer("Name","regressionoutput")];
plot(layerGraph(layers));
RedArquitectura=layerGraph(layers);
```

Como se hacen en los casos anteriores se carga una red neuronal preentrenada a la que se modifica las últimas capas para adaptarla al caso que tratamos re resolver. Como estamos trabajando con regresión, hay que poner siempre en la última capa una fully conencted de 2 salidas.

Ahora el siguiente paso es ajustar las opciones de entrenamiento, para ello se define options de la siguiente manera:

```
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',... %Planifica la evolución del
InitialLearnRate
    'LearnRateDropPeriod',10,...  %periodo de cada cuanto queremos que haya
un drop, para 10 iteraciones
    'LearnRateDropFactor',0.3,... %ratio inicial un 30 % de ellas
    'Momentum',0.9, ... %Suavizar el gradiente , para que no cambie
bruscamente
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'MaxEpochs',30, ...   % cuantas epocas máximas
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ... % Que se barajeen en cada epoch el orden de
los patrones
    'CheckpointPath', tempdir, ... %Si queremos poner un path de chequeo
    'VerboseFrequency',2,...
    'Plots','training-progress');  %Si queremos que nos saque las gráficas
%Con que criterio vamos a parar en caso de que el error de validación suba
```

Finalmente, tras tener los datos, la red neuronal con sus capas y las opciones, el paso final es la realización del entrenamiento. Tras finalizar el entrenamiento, se puede observar la gráfica siguiente en la ***Figura 6.41***

```
[Red, info] = trainNetwork(comDS2RL,RedArquitectura,options);

save all
```

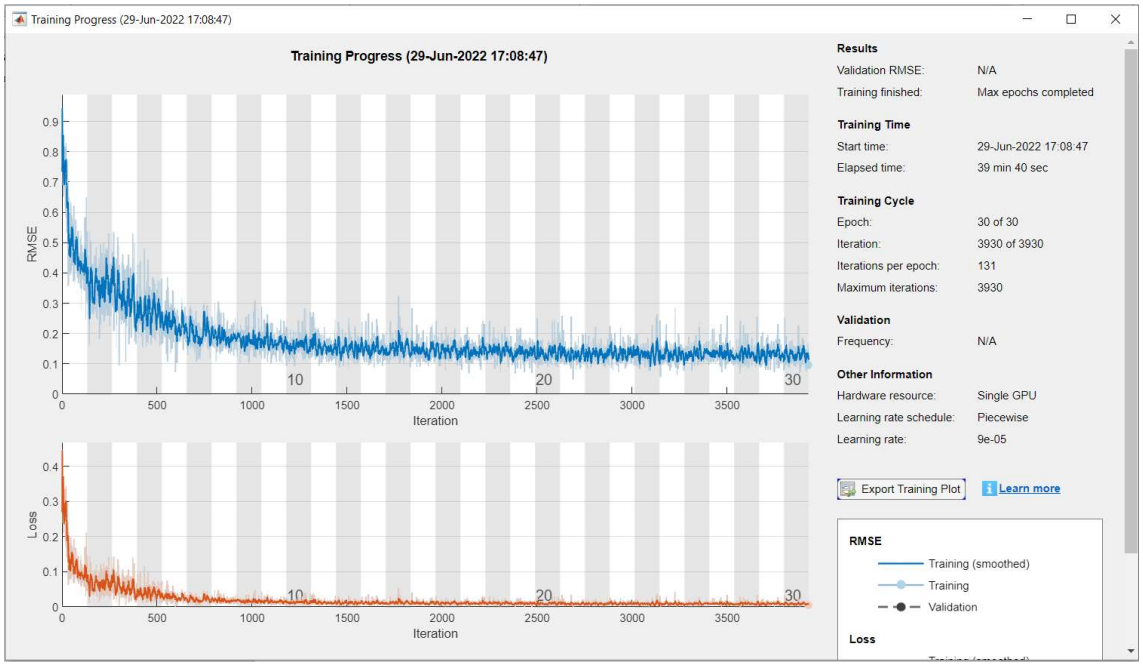


Figura 6.41: Resultado del entrenamiento de la red neuronal

Para saber que se ha tenido éxito en el entrenamiento, simplemente basta comparar esta gráfica, con las anteriores, sobre todo con la que se realizó mediante la regresión sin patrón de seguimiento.

En este caso se recuerda que estamos usando el error cuadrático medio, y por lo que se puede observar, dicho error va disminuyendo a medida que avanza el entrenamiento.

Finalmente, para comprobar que realmente saber cómo ha acertado en las predicciones la red neuronal, simplemente se desarrolla un script que permite saber el nivel de predicción de la red. En este código se compara los valores marcados, junto con los valores predichos por la red neuronal, ver ***Figura 6.42***

```
reset(comDS2RL)
while hasdata(comDS2RL)
    data=read(comDS2RL);
    Prediccion = predict(Red,data{1,1});
    imshow(data{1,1})
    hold on

plot(data{1,2}(1)*targetSize(1),data{1,2}(2)*targetSize(2),'xr','MarkerSize',
5)

plot(Prediccion(1)*targetSize(1),Prediccion(2)*targetSize(2),'ob','MarkerSize
',5)
    i=i+1;
    hold off
    pause(0.1)
end
```

Figura 6.42: Predicción de la red neuronal entrenada

Finalmente se guarda la red con el siguiente comando:

```
save ("RegresionLinea",'Red','-v7.3');
```

CAPITULO 7

PLANIFICACIÓN Y PRESUPUESTO

7.1. Planificación y Diagrama de Gantt

En este apartado se va a exponer la planificación del trabajo realizado, para ello se expondrá en la siguiente tabla las fases de trabajo:

ETAPAS
Planificar el trabajo
Investigación sobre la inteligencia artificial
Investigación sobre redes neuronales
Investigación sobre redes neuronales convolucionales
Deep Learning Matlab
Aplicación Práctica
Documentación del proyecto

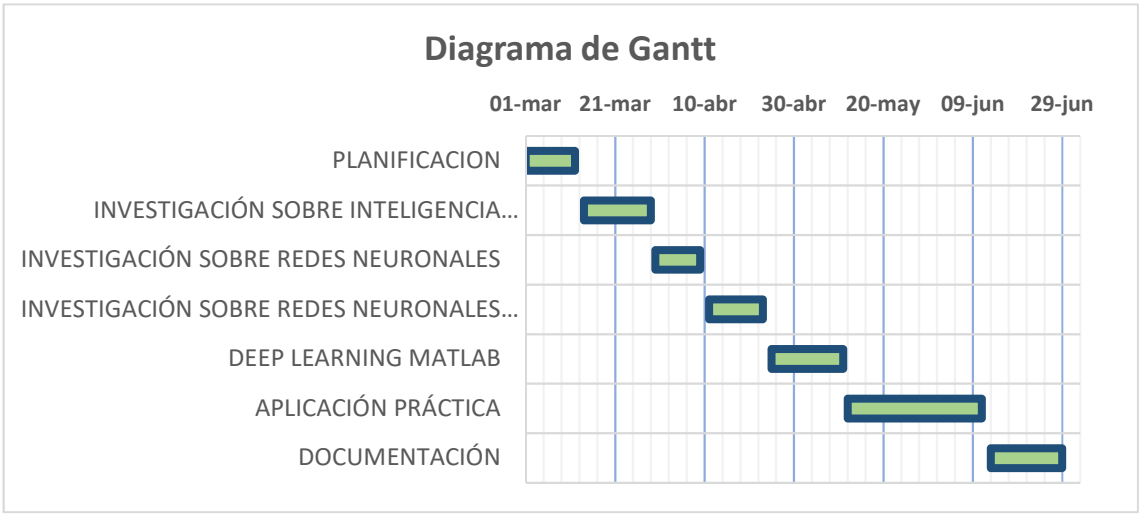
Además, en la siguiente tabla, el autor muestra la información sobre su dedicación y disponibilidad horaria:

FECHA	**INICIO**	**FIN**	**HORAS TOTALES**
Lunes a viernes	08:00	15:00	7 horas
Sábado	10:00	13:00	3 horas

Por la cual se han tenido las siguientes etapas, representadas en la siguiente tabla como con su correspondiente diagrama de Gantt:

Etapa	Descripción	Comienzo	Fin	Horas
Planificación de trabajo	**Planificación de etapas y búsqueda bibliográfica**	**01/03/2022**	**12/03/2022**	**69 horas**
Investigación sobre Inteligencia Artificial	**Con la bibliografía obtenida investigar y redactar**	**14/03/2022**	**29/03/2022**	**90 horas**
Investigación sobre redes neuronales	**Con la bibliografía obtenida investigar y redactar**	**30/03/2022**	**09/04/2022**	**62 horas**
Investigación sobre redes	**Con la bibliografía obtenida**	**11/04/2022**	**23/04/2022**	**76 horas**

neuronales convolucionales	**investigar y redactar**			
Deep Learning Matlab	**Aprendizaje del uso de Matlab enfocado a Redes Neuronales**	**25/04/2022**	**11/05/2022**	**90 horas**
Aplicación Práctica	**Montaje robot y pruebas**	**12/05/2022**	**13/05/2022**	**7 horas**
	Red neuronal convolucional por clasificación sin patrón	**14/05/2022**	**19/05/2022**	**31 horas**
	Red Neuronal convolucional por regresión sin patrón	**20/05/2022**	**03/06/2022**	**83 horas**
	Red Neuronal Convolucional por regresión con patrón	**04/06/2022**	**11/06/2022**	**41 horas**
Documentación	**Redacción de documentación y Revisión**	**13/06/2022**	**29/06/2022**	**97 horas**
Trabajo		**01/03/2022**	**29/06/2022**	**646 horas**



7.2. Presupuesto

De cara al presupuesto para realizar este trabajo, se va a relatar todo lo utilizado en la siguiente tabla:

MATERIAL FÍSICO- HARDWARE			
Nombre	***Cantidad***	***Precio***	***Precio Final***
Arduino Mega2560	1	49,09€	49,09€
Batería de Plomo Ácido 12V 5Ah 90x70x107	1	21,30€	21,30€
Módulo Puente en H L298N	2	3,6€	7,2€
Cargador de Batería	1	14,31€	14,31€
Motores AndyMark Neverest 60	2	73,49€	146,98€
Rueda+soporte	2	5,90€	34,81€
Rueda con Pivote	1	3,28 €	3,28 €
Cámara Logitech	1	27,90€	27,90€
Total, Material Físico			304,87 €
SOFTWARE			
Licencia Matlab Student + Toolboxs	1	6000 €	6000 €

En primer lugar, se va a realizar un cálculo de las horas que se han dedicado al trabajo. Como se puede observar en el apartado anterior, se ha trabajado aproximadamente unas 632 horas y se supone un precio de 40 euros la hora:

$$Coste\ trabajo = 646\ horas\ x\ 40\frac{€}{hora} = 25840\ €$$

A continuación, se verá la amortización del software y de la parte física o hardware, suponiendo que todo ello se amortiza a 4 años y que cada año se trabajan 1700 horas:

<u>Software</u>

$$Licencia\ de\ Matlab = 6000€$$

$$Coste\ de\ Matlab\ x\ hora = \frac{6000€}{\frac{1700\ h}{año} x\ 4\ años} = 0{,}88\frac{€}{h}$$

$$Coste\ de\ Matlab = 0{,}88\ \frac{€}{h}\ x\ 646\ horas = 568{,}48€$$

Hardware

$$Coste\ Hardware = 304{,}87\ €$$

$$Coste\ de\ Hardware\ x\ hora = \frac{304{,}87€}{\frac{1700\ h}{año} x\ 4\ años} = 0{,}045 \frac{€}{h}$$

$$Coste\ de\ Hardware = 0{,}045\ \frac{€}{h}\ x\ 646\ horas = 29{,}07€$$

Luego sumando los costes:

$$Constes\ directos = Coste\ trabajo + Coste\ software + Coste\ hardware$$
$$= 25840\ + 568{,}48 + 29{,}07 = 26437{,}55\ €$$

Suponiendo un 10 % de costes indirectos:

$$Costes\ indirectos = 10\%\ Costes\ directos = 0.1\ x\ 26437{,}55 = 2643{,}76€$$

Sumando los costes directos e indirectos:

$$Costes = Costes\ directos + Costes\ indirectos = 26437{,}55 + 2643{,}76$$
$$= 29081{,}31\ €$$

Suponiendo un 20 % de beneficio industrial:

$$Beneficio\ industrial = 20\ \%\ Costes = 0.2\ x\ 29081{,}31 = 5816{,}27\ €$$

Sumando los costes con el beneficio industrial:

$$Costes\ totales = Costes + Beneficio\ Industrial = 29081{,}31 + 5816{,}27 = 34897{,}57\ €$$

Finalmente aplicando un 21 % de IVA:

$$IVA = 21\ \%\ Costes\ totales = 0{,}21\ x\ 34897{,}57 = 7328.49€$$

El coste final sería:

$$Coste\ final = IVA + Costes\ totales = 7328.49 + 34897{,}57 = 42226{,}06€$$

RESUMEN		COSTE (€)
Costes directos	Mano de obra	25280 €
	Software	556,16€
	Hardware	28,44€
Costes directos totales		26437,55 €
Costes indirectos		2643,76€
Beneficio Industrial		5816,27 €
IVA (21 %)		7328.49€
Coste final		42226,06€

CAPITULO 8

PLIEGO DE CONDICIONES

En este capítulo se va a detallar las especificaciones del software y hardware utilizados durante la realización del trabajo. Todo el código

que se ha realizado en este trabajo se adjuntará con el documento de la memoria técnica.

8.1. Hardware

En este apartado se detallará el equipo físico utilizado, que en caso del hardware es un ordenador portátil con las siguientes características:

8.1.1. Ordenador Portátil

- Modelo: HP OMEN 15-dhOxxx

- CPU: Intel Core i7-9750H CPU @ 2.60 GHz
  - Velocidad de reloj: 1794MHz
  - Descripción: Intel64 Family 6 Model 158 Stepping 10

- GPU: NVIDIA GeForce GTX 1660 Ti
  - Tasa de actualización: 143Hz
  - Versión del controlador: 30.0.14.1179

- RAM: M471A1K43DB1-CTD
  - Capacidad: 8GB
  - Tipo: DDR4
  - Velocidad del reloj predeterminada: 2667MHz

8.1.2. Arduino

- Microcontrolador: ATmega2560
- Tensión de operación: 5V
- Tensión de entrada recomendado:7V-12V
- Pines de E/S digital: 54 de los cuales 15 son de salida PWM
- Pines de entrada analógica: 16
- Memoria Flash : 256 KB, de los cuales 8 KB son usados por el gestor de arranque.

8.2 Software

- Sistema Operativo: Microsoft Windows 10 Home

- Software Programación: Matlab R2020a y Matlab R2022a, para su uso académico, con las aplicaciones correspondientes

    - *Deep Learning Toolbox Model*
    - *Video Labeler*
    - *Image Labeler*
    - *Image Acquisition Toolbox Support Package for OS Generic Video Interface*
    - *Matlab Support Package for Arduino*
    - *Matlab Support Package for USB Webcams*

BIBLIOGRAFÍA

[1] Alexandra Balbás Calvo, Manuel Domínguez, María Del Mar Espinosa. El volante en la conducción autónoma. *Revista Iberoamericana de Ingeniería Mecánica*. 2019;23(2):89-101. https://search.proquest.com/docview/2378093874.

[1] Schwab, K. (2015). The Fourth Industrial Revolution. What It Means and how to Respond. Foreing Affairs, December

[2] Perasso, V. (12, octubre de 2016) "Qué es la cuarta revolución industrial (y por qué debería preocuparnos)" BBC Mundo. Recuperado de http://www.bbc.com/mundo/noticias-37631834.

[3] RUESSMANN, Michael et al. (2015): Industry 4.0. The Future of Productivity and Growth in Manufacturing Industries. The Boston Consulting Group

[4] MCKINSEY & COMPANY (2015): Four Fundamentals of Workplace Automation. McKinsey Quarterly, novembre de 2015.

[5] MORRÓN, Adrià (2016): Arribarà la quarta revolució industrial a Espanya? Informe mensual de febrer de 2016. CaixaBank Research

[6] Mehami, J., Nawi, M., & Y Zhong, R. (2018). Smart automated guided vehicles for manufacturing in the context of Industry 4.0 (p. 10). Elsevier B.V.

[7] Kumar Das S. Design and Methodology of Automated Guided Vehicle-A Review. IOSR J Mech Civ Eng. 2016;03(03):29-35.

doi:10.9790/1684-15010030329-35

[8] Goto, Y., Stentz, A., "Mobile Robot Navigation: The CMU System," IEEE Expert, Vol. 2, No. 4, Winter, 1987.

[9] Hopcroff, J. E. Máquinas de Turing. Investigación y Ciencia, 1984.

[10]Hodges, Andrew(1983). *Alan Turing: The Enigma*. Reino Unido: Burnet Books/Hutchinson. ISBN 0-671-49207-1

[11]Moravec, H. Mind Children. The Future of Robot and Human Intelligence. Harvard University Press, 1988.

[12] RAJSBAUM, Sergio; MORALES, Eduardo. Norbert Wiener y el origen de la cibernética. *Revista de la Academia Mexicana de Ciencias*, 2016, vol. 67, no 1, p. 6-11.

[13] Goldstine, Herman H.; Goldstine, Adele (1946). «The Electronic Numerical Integrator and Computer (ENIAC)». *Mathematical Tables and Other Aids to Computation* **II ,nº 15**

[14] Bromberg S, Saavedra P. Besuch von wichern. *Forschungstraditionen der Sozialen Arbeit*. 2004;40:37. doi: 10.2307/j.ctvddzfzv.5

[15] GONZÁLEZ, Rodrigo. El Test de Turing: Dos mitos, un dogma. *Revista de filosofía*, 2007, vol. 63, p. 37-53.

[16] Leyva-Vázquez M, Smarandache F. *Inteligencia artificial: Retos, perspectivas y papel de la neutrosofía.* Infinite Study; 2018.

[17] ARRESTEGUI, Luis Barrera. Fundamentos históricos y filosóficos de la Inteligencia Artificial. *UCV-HACER. Revista de Investigación y Cultura*, 2012, vol. 1, no 1, p. 87-92.

[18] McCarthy(2007), J. What is Artificial Intelligence? Stanford University, Computer Science Department. EEUU.

[19] Nilsson, N. (2001) Inteligencia Artificial, Una Nueva Síntesis. Primera edición en español. Mc Graw - Hill. España.

[20] Minsky, M. (1990) The Age of Intelligent Machines: Thoughts About Artificial Intelligence. Artículo publicado en KurzweilAI.net. [citado: 27-11-2008] URL: http://www.kurzweilai.net/articles/art010 0.html

[21] Shirai, Y. & Tsujii, J. (1982) Inteligencia Artificial: Conceptos, técnicas y aplicaciones. Primera edición en español de 1987. Editorial Ariel. España.

[22] Russell, S. (2003) Stuart Russell on the Future of Artificial Intelligence. Ubiquity (4-43). Publicación de la ACM. EEUU. [citado: 27-11-2008] URL: http://www.acm.org/ubiquity/interviews/v 4i43_russell.html

[23] ROUHIAINEN, Lasse. Inteligencia artificial. *Madrid: Alienta Editorial*, 2018.

[24] McClendon, L., & Meghanathan, N. (2015). Using Machine Learning Algorithms to analyze crime data. Machine Learning and Applications: an international journal, Vol 2, No 1.

[25] Mitchell, T. (1997). Machine Learning. Portland: McGraw-Hill.

[26] Álogos. (9 de julio de 2018). Introducción a Machine Learning. Obtenido de Álogos: http://alogos.es/introduccion-machine-learning/

[27] Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding machine learning: from theory to algorithms. Nueva York: Cambridge University press.

[28] Deng, L., & Yu, D. (2013). Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing, Vol 7. Pags 197-387.

[29] Gran Enciclopedia Aragonesa. Voz Ram6n y Cajal, Santiago. Tomo X 1982 (http://www.aragob.es/pre/cido/cajal.htm)

[30] Kandel, E. R., Schwartz, T. H., Jessel, T. M. Principles of Neural Science. 4" edicih, McGraw-Hill, 1999.

[31] D. E., McClelland, J.L. (eds.). Parallel Distributed Processing. Vol 1 : Foundations. MIT Press, 1986.

[32] McClelland, J. L., Rumelhart, D.E. (eds.). Parallel Distributed Processing. Vol 2: Psychological and biological models. MIT Press, 1986

[33] Gedeon, T. D., Wong, P. M., Harris, D. Balancing the bias and variance: Network topology and pattern set reduction techniques. Proc. Int. Work. on Artificial Neural Networks, IWANN95, pp. 550-8, Torremolinos (España), junio, 1995.

[34] Rosenblatt, F. Principles of Neurodynamics. Spartan Books, Nueva York, 1962.

[35] Minsky, M., Papert, S. Perceptrons: An Introduction to Computational Geometry. MIT Press, 1969.

[36] Widrow, B., Hoff, M.E. Adaptive switching circuits. 1960 IRE WESCON Convention Record, 4, pp. 96-104, 1960.

[37] Widrow, B., Winter, R. Neural nets for adaptive filtering and adaptive pattern recognition. IEEE Computer, marzo, 25-39, 1988.

[38] Freeman, F. A., Skapura, D.M. Neural Networks: Algorithms, Applications and Programming Techniques. Addison-Wesley, 1992. (Redes Neuronales. Algoritmos, Aplicaciones y Tkcnicas de Programacidn. Addison-WesleyIDiaz de Santos, 1993).

[39] Werbos, P. J. Backpropagation through time: What it does and how to do it. Proc. of the IEEE, oct., pp. 1550- 1560, 1990

[40] Sarahí Silva y Estefanía Freire. Introducción a las redes neuronales convolucionales. https: //medium.com/@bootcampai/redes-neuronales-convolucionales-5e0ce960caf8, 2019. Accedido el 6-7-2020.

[41] HUBEL, David H.; WIESEL, Torsten N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 1959, vol. 148, no 3, p. 574.

[42]Fukushima K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. 1980. Biological Cybernetics, pp. 193-202.

[43] LECUN, Yann, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998, vol. 86, no 11, p. 2278-2324.

[44] D. Cireşan, U. Meier, J. Masci and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification", Neural Networks, vol. 32, pp. 333-338, 2012.

[45] L. M. Hurvich: Color Vision (Sinauer Associates Inc., Massachusetts, 1981) Cap. 20, p. 283.

[46] W. K. Pratt: Digital Image Processing (Jhon Wiley & Sons, Inc., New York, 1991) Cap. 2, p. 93; Cap. 10, p. 310.

[47] Apuntes curso" Deep Learning aplicado al an´alisis de señales e imágenes", CFP Universidad Politécnica de Valencia, 2020.

[48] SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[49] SZEGEDY, Christian, et al. Going deeper with convolutions. En *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015. p. 1-9.

[50] HE, Kaiming, et al. Deep residual learning for image recognition. En *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. p. 770-778.

ANEXO

Prueba de la cámara

```
%pruebaCamara.m
clearvars
camara = webcam(2);
camara.Resolution= '640x480';
preview(camara)
image = snapshot (camara);
closePreview(camara)
imshow(image);
imsz = size(image);
```

Prueba de los motores sin bucle

```
%pruebaMotor.m
ar = arduino('com6','Mega2560'); %Objeto Arduino


% Motor A
   writeDigitalPin(ar,'D12',1);
   writeDigitalPin(ar,'D13',0);
   writePWMDutyCycle(ar,'D11',0.6)%Velocidad del motor


 % Motor B
   writeDigitalPin(ar,'D26',1);
   writeDigitalPin(ar,'D24',0);
   writePWMDutyCycle(ar,'D7',0.6)%Velocidad del motor
```

Prueba de los motores en bucle for

```
%pruebaMotor.m
ar = arduino('com6','Mega2560'); %Objeto Arduino

% Motor A
writeDigitalPin(ar,'D12',1);
writeDigitalPin(ar,'D13',0);
% Motor B
writeDigitalPin(ar,'D26',1);
writeDigitalPin(ar,'D24',0);


for d=0:0.05:1
writePWMDutyCycle(ar,'D11',d) %Velocidad del motor A
```

```
                                writePWMDutyCycle(ar,'D7',d)
%Velocidad del motor B
end
```

Captura de Imágenes

```
%CapturaImagnes.m
clc;clear;close all;
cam = webcam(1); % Selección de la cámara y asignar a
una variable
k=1;
while 1
    im = snapshot (cam); % captura de imagen  y
guardarla en al variable im
    imshow(im) % ver la imagen capturada

Fichero=strcat('.\imagenes\Imagen',num2str(k),'.jpg');
% guardar la ruta de la imagen


 imwrite(im,Fichero) %guardar la imagen en un fichero
    pause(1)
    k=k+1;
end
```

Etiquetado Manual Clasificación

```
%Etiketado.m
clc
close all
clear all
%Etiquetado manual


V=VideoReader('D:\PruebasVGG16\PrimeraPrueba\output.av
i');

xtotal=[];
ytotal=[];
refImagen=[];
for i=1:v.NumFrames
   imagenframe = read(v,i);


     imshow(imagenframe);
    [x,y,button] = ginput(1)
    refImagen=[refImagen,i];
```

```
 xtotal=[xtotal,x];
    ytotal=[ytotal,y];
end
save('Datos_etiquetado')
```

Descomponer para datos de clasificación

```
%Descomponer.m
load('Datos_etiquetado.mat')

v = VideoReader('output.avi');
valid_frames=(ytotal>0) | (xtotal>0);
recto = ytotal>100 ;
derecha = xtotal>500 ;
izquierda= xtotal<250;

pathR='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\recto';
pathD='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\derecha';
pathI='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas\izquierda';


for i=1:v.NumFrames
  imagenframe = read(v,i);
   if valid_frames(i)
       if recto (i)
        img_name=sprintf('img_%04d.jpg',i);
        img_path=fullfile(pathR,img_name);
        imwrite(imagenframe,img_path);
         rawlabel(i)=xtotal(i);

       elseif derecha (i)

 img_name=sprintf('img_%04d.jpg',i);

 img_path=fullfile(pathD,img_name);
      imwrite(imagenframe,img_path);
        rawlabel(i)=xtotal(i);

      elseif izquierda (i)

      img_name=sprintf('img_%04d.jpg',i);
      img_path=fullfile(pathI,img_name);
       imwrite(imagenframe,img_path);
        rawlabel(i)=xtotal(i);


       end
   end

end

beep
```

```
                                    save('rawlabel','rawlabel')
```

Crear datastore para clasficiación

```
%crearDatastore.m

clc
clear all;

%crearDatastore.m
dsC=
imageDatastore('imagenesEtiquetadas\','IncludeSubfolders',true,'LabelSource',
'foldernames');
audsC = augmentedImageDatastore([56 56],dsC);
```

EntrenamientoClasificación

Cargar los datos de entrenamiento

```
trainingSetup =
load("C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\params2_2022_06_25_
_13_48_17.mat");
```

Importar los datos de validación y entrenamiento

```
imdsTrain = trainingSetup.imdsTrain;
[imdsTrain, imdsValidation] = splitEachLabel(imdsTrain,0.7);


% Resize the images to match the network input layer.
augimdsTrain = augmentedImageDatastore([56 56 3],imdsTrain);
augimdsValidation = augmentedImageDatastore([56 56 3],imdsValidation);
```

Cargar la red

```


% Cargar la red.
layers = [
    imageInputLayer([56 56
3],"Name","imageinput","Normalization","rescale-zero-one")
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv1_1.Bias,"Weights",trainin
gSetup.conv1_1.Weights)
    reluLayer("Name","relu1_1")
    convolution2dLayer([3 3],64,"Name","conv1_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv1_2.Bias,"Weights",trainin
gSetup.conv1_2.Weights)
    reluLayer("Name","relu1_2")
    maxPooling2dLayer([2 2],"Name","pool1","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv2_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv2_1.Bias,"Weights",trainin
gSetup.conv2_1.Weights)
    reluLayer("Name","relu2_1")
```

```
    convolution2dLayer([3
3],128,"Name","conv2_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv2_2.Bias,"Weights",trainin
gSetup.conv2_2.Weights)
    reluLayer("Name","relu2_2")
    maxPooling2dLayer([2 2],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv3_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv3_1.Bias,"Weights",trainin
gSetup.conv3_1.Weights)
    reluLayer("Name","relu3_1")
    convolution2dLayer([3 3],256,"Name","conv3_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv3_2.Bias,"Weights",trainin
gSetup.conv3_2.Weights)
    reluLayer("Name","relu3_2")
    convolution2dLayer([3 3],256,"Name","conv3_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv3_3.Bias,"Weights",trainin
gSetup.conv3_3.Weights)
    reluLayer("Name","relu3_3")
    maxPooling2dLayer([2 2],"Name","pool3","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv4_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv4_1.Bias,"Weights",trainin
gSetup.conv4_1.Weights)
    reluLayer("Name","relu4_1")
    convolution2dLayer([3 3],512,"Name","conv4_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv4_2.Bias,"Weights",trainin
gSetup.conv4_2.Weights)
    reluLayer("Name","relu4_2")
    convolution2dLayer([3 3],512,"Name","conv4_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv4_3.Bias,"Weights",trainin
gSetup.conv4_3.Weights)
    reluLayer("Name","relu4_3")
    maxPooling2dLayer([2 2],"Name","pool4","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv5_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv5_1.Bias,"Weights",trainin
gSetup.conv5_1.Weights)
    reluLayer("Name","relu5_1")
    convolution2dLayer([3 3],512,"Name","conv5_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv5_2.Bias,"Weights",trainin
gSetup.conv5_2.Weights)
    reluLayer("Name","relu5_2")



 convolution2dLayer([3 3],512,"Name","conv5_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",trainingSetup.conv5_3.Bias,"Weights",trainin
gSetup.conv5_3.Weights)
    reluLayer("Name","relu5_3")
    maxPooling2dLayer([2 2],"Name","pool5","Stride",[2 2])
    fullyConnectedLayer(100,"Name","fc_2")
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")
```

```
fullyConnectedLayer(100,"Name","fc_3")
    reluLayer("Name","relu7")
    dropoutLayer(0.5,"Name","drop7")
    fullyConnectedLayer(3,"Name","fc_1")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];
```

Opciones

```
opts = trainingOptions("sgdm",...

    "ExecutionEnvironment","auto",...
    "InitialLearnRate",0.01,...
    "MaxEpochs",12,...
    "MiniBatchSize",8,...
    "Shuffle","every-epoch",...
    "ValidationFrequency",150,...


    "Plots","training-progress",...
    "ValidationData",augimdsValidation);
```

Entrenar la red

```
[net, traininfo] = trainNetwork(augimdsTrain,layers,opts);
```

Etiquetado para regresión

```
%EtiketadoRegresion.m
clc
close all
clear all
%Etiquetado manual

v = VideoReader('D:\PruebasVGG16\Pruebita\video\video.avi');

xtotal=[];
ytotal=[];
refImagen=[];
%coordenadas
coordenadasL=[];
for i=1:v.NumFrames
   imagenframe = read(v,i);
   imshow(imagenframe);
    [x,y,button] = ginput(1)
    refImagen=[refImagen,i];
    xtotal=[xtotal,x];
    ytotal=[ytotal,y];
    coordenadasL = [xtotal,x;ytotal,y];
end
save('Datos_etiquetado')
```

Descomponer video para regresión

```
%DescomponerRegresion.m
load('Datos_etiquetado.mat')

v = VideoReader('output1.avi');
valid_frames=(ytotal>0) | (xtotal>0);

path='C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadas';



for i=1:v.NumFrames
   imagenframe = read(v,i);
   if valid_frames(i)
        img_name=sprintf('img_%04d.jpg',i);
        img_path=fullfile(pathL3,img_name);
        imwrite(imagenframe,img_path);
%        rawlabel(i)=xtotal(i);
   end

end
save('rawlabel','rawlabel')
```

Coordenadas Positivas

```
%CoordenadasPositivas
f1 = coordenadas(1,:);
f1pos=f1(f1>0);
f2 = coordenadas(2,:);
f2pos=f2(f2>0);

coordenadasP(1,:)=f1pos;
coordenadasP(2,:)=f2pos;
```

ResizeCoordenadas

```
%Resize de coordenadas
 filax = coordenadasP(1,:);
 calculo1 = round(filax * (227/640));
 filay = coordenadasP(2,:);
 calculo2 = round(filay * (227/640));
```

EjecucionRegresion

Crear un array de capas

```
layers = [
     imageInputLayer([227 227
3],"Name","imageinput","Normalization","rescale-zero-
one","Min",0,"Max",255)
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu1_1")
```

```
                convolution2dLayer([3
3],64,"Name","conv1_2","Padding",[1 1 1 1],"WeightL2Factor",0)
    reluLayer("Name","relu1_2")
    maxPooling2dLayer([2 2],"Name","pool1","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv2_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu2_1")
    convolution2dLayer([3 3],128,"Name","conv2_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu2_2")
    maxPooling2dLayer([2 2],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv3_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu3_1")
    convolution2dLayer([3 3],256,"Name","conv3_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu3_2")
    convolution2dLayer([3 3],256,"Name","conv3_3","Padding",[1 1 1
1],"WeightL2Factor",0)

   reluLayer("Name","relu3_3")
    maxPooling2dLayer([2 2],"Name","pool3","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv4_1","Padding",[1 1 1
1],"WeightL2Factor",0)

    reluLayer("Name","relu4_1")
    convolution2dLayer([3 3],512,"Name","conv4_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu4_2")
    convolution2dLayer([3 3],512,"Name","conv4_3","Padding",[1 1 1
1],"WeightL2Factor",0)

reluLayer("Name","relu4_3")
    maxPooling2dLayer([2 2],"Name","pool4","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv5_1","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu5_1")
    convolution2dLayer([3 3],512,"Name","conv5_2","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu5_2")
    convolution2dLayer([3 3],512,"Name","conv5_3","Padding",[1 1 1
1],"WeightL2Factor",0)
    reluLayer("Name","relu5_3")
    maxPooling2dLayer([2 2],"Name","pool5","Stride",[2 2])

fullyConnectedLayer(100,"Name","fc_2","BiasLearnRateFactor",1,"WeightLear
nRateFactor",1)
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")
```

```
fullyConnectedLayer(10,"Name","fc_3","BiasLearnRateFactor",1,"WeightLearn
RateFactor",1)
    reluLayer("Name","relu7")
    dropoutLayer(0.5,"Name","drop7")

fullyConnectedLayer(2,"Name","fc_1","BiasLearnRateFactor",1,"WeightLearnR
ateFactor",1)
    regressionLayer("Name","regressionoutput")];
```

Dibujar las capas

```
plot(layerGraph(layers));
```

Datos

```
imgpath =
'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadasLinea
8';
imDsL = imageDatastore(imgpath);
targetSize=[227,227];
writematrix(coordenadasP',’coordenadasP','FileType','spreadsheet');


ssd2rL =  spreadsheetDatastore('coordenadasP.xls');ssd2rL.ReadSize=1;

Tssds2RL = transform(ssd2rL,@(x) table2array(x));
imdsReSzL = transform(imDsL,@(x) customResize(x,targetSize));
comDS2RL=combine(imdsReSzL,Tssds2RL);


%Datos de Validación
imgpathV =
'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\validacionLinea8';
imDsLV = imageDatastore(imgpathV);
writematrix(coordenadasv','coordenadasv','FileType','spreadsheet');
ssd2rLV =  spreadsheetDatastore('coordenadasv.xls');ssd2rLV.ReadSize=1;

Tssds2RLV = transform(ssd2rLV,@(x) table2array(x));
imdsReSzLV = transform(imDsLV,@(x) customResize(x,targetSize));
comDS2RLV=combine(imdsReSzLV,Tssds2RLV);




Opciones
options = trainingOptions('sgdm', ...
    'MaxEpochs',8,...
    "MiniBatchSize",4,...
    'InitialLearnRate',1e-6, ...
    'Verbose',false, ...
```

```
                                    'Shuffle','every-epoch',...
    'Plots','training-progress',...
    "Verbose",true,...
    "VerboseFrequency",50,...
    "ValidationData",comDS2RLV,...
    "ValidationPatience",200);
```

Entrenamiento

```
[PruebaRegresion2,info] = trainNetwork(comDS2RL, layers, options);
```

Funcion out

```
function out=customResize(x,targetSize)
    imRes=imresize(x,targetSize);
    dummycell=cell(1,1);
    dummycell{1,1}=imRes;
    out=dummycell;
end
```

clean_BBDD

```
% load('gTruthLineYelow.mat')
%clean_BBDD.m
labels=table2cell(gTruthLineYelow.LabelData);
dummy_Cell=cell(length(labels),1);
num_puntos=zeros(length(labels),1);

for i=1:length(labels)
    clc

    display(i/length(labels)*100)

    puntos=labels{i,1}{1,1};
    if size(puntos,1)<5


    X=puntos(end-1:end,1);
    Y=puntos(end-1:end,2);

    N=2+(5-size(puntos,1));

    addX=linspace(X(1), X(2), N);
    addY=linspace(Y(1), Y(2), N);

    puntosfinal=[puntos(1:end-2,:);[addX',addY']];

 dummy_Cell{i,1}=puntosfinal;

        if size(puntosfinal,1)~=5
            pause()
        end
```

```
                                        num_puntos(i)=size(puntosfinal,1);

    elseif size(puntos,1)>5
    puntosfinal=puntos(1:5,:);

    dummy_Cell{i,1}=puntosfinal;
        if size(puntosfinal,1)~=5
            pause()
        end
    num_puntos(i)=size(puntosfinal,1);
    else
    puntosfinal=puntos;
    num_puntos(i)=size(puntosfinal,1);
    end


%     dummy_Cell{i,1}=[];
end
```

RegresionLinea

```
%RegresionLinea
clc
close all
clear all

imgpath =
'C:\Users\domen\OneDrive\Documentos\MATLAB\VGG16\imagenesetiquetadasLinea2';
%imDsL = imageDatastore(imgpath);
targetSize=[64,64];
imDsL=imageDatastore(imgpath,'FileExtensions','.jpg','ReadFcn',@(filename)cus
tomResize(filename,targetSize));
targetSizeOriginal=[227,227];
load('c.mat')
c=c';
c(:,1)=(c(:,1)-min(c(:,1)))/(max(c(:,1))-min(c(:,1)));
c(:,2)=(c(:,2)-min(c(:,2)))/(max(c(:,2))-min(c(:,2)));
dataoutNormalized = arrayDatastore(c);
i=1;
comDS2RL=combine(imDsL,dataoutNormalized);
reset(comDS2RL)
while hasdata(comDS2RL)
    data=read(comDS2RL);
    imshow(data{1,1})
    hold on

plot(data{1,2}(1)*targetSize(1),data{1,2}(2)*targetSize(2),'xr','MarkerSize',
5)
    i=i+1;
    hold off
     pause(0.1)
end
```

```
%Crear la red neuronal
params =
load("D:\PruebasVGG16\SeptimaPrueba\params_2022_06_23__12_16_07.mat");
layers = [
    imageInputLayer([64 64 3],"Name","imageinput")
    convolution2dLayer([3 3],64,"Name","conv1_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv1_1.Bias,"Weights",params.conv1_1.Wei
ghts)
    reluLayer("Name","relu1_1")
    convolution2dLayer([3 3],64,"Name","conv1_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv1_2.Bias,"Weights",params.conv1_2.Wei
ghts)
    reluLayer("Name","relu1_2")
    maxPooling2dLayer([2 2],"Name","pool1","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv2_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv2_1.Bias,"Weights",params.conv2_1.Wei
ghts)
    reluLayer("Name","relu2_1")
    convolution2dLayer([3 3],128,"Name","conv2_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv2_2.Bias,"Weights",params.conv2_2.Wei
ghts)
    reluLayer("Name","relu2_2")
    maxPooling2dLayer([2 2],"Name","pool2","Stride",[2 2])
    convolution2dLayer([3 3],256,"Name","conv3_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_1.Bias,"Weights",params.conv3_1.Wei
ghts)
    reluLayer("Name","relu3_1")
    convolution2dLayer([3 3],256,"Name","conv3_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_2.Bias,"Weights",params.conv3_2.Wei
ghts)
    reluLayer("Name","relu3_2")
    convolution2dLayer([3 3],256,"Name","conv3_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv3_3.Bias,"Weights",params.conv3_3.Wei
ghts)
    reluLayer("Name","relu3_3")
    maxPooling2dLayer([2 2],"Name","pool3","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv4_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_1.Bias,"Weights",params.conv4_1.Wei
ghts)
    reluLayer("Name","relu4_1")
    convolution2dLayer([3 3],512,"Name","conv4_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_2.Bias,"Weights",params.conv4_2.Wei
ghts)
    reluLayer("Name","relu4_2")
    convolution2dLayer([3 3],512,"Name","conv4_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv4_3.Bias,"Weights",params.conv4_3.Wei
ghts)
    reluLayer("Name","relu4_3")
    maxPooling2dLayer([2 2],"Name","pool4","Stride",[2 2])
    convolution2dLayer([3 3],512,"Name","conv5_1","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_1.Bias,"Weights",params.conv5_1.Wei
ghts)
    reluLayer("Name","relu5_1")
    convolution2dLayer([3 3],512,"Name","conv5_2","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_2.Bias,"Weights",params.conv5_2.Wei
ghts)
    reluLayer("Name","relu5_2")
```

```
                                        convolution2dLayer([3
3],512,"Name","conv5_3","Padding",[1 1 1
1],"WeightL2Factor",0,"Bias",params.conv5_3.Bias,"Weights",params.conv5_3.Wei
ghts)
    reluLayer("Name","relu5_3")
    maxPooling2dLayer([2 2],"Name","pool5","Stride",[2 2])
    fullyConnectedLayer(4000,"Name","fc_3")
    reluLayer("Name","relu6")
    dropoutLayer(0.5,"Name","drop6")
    fullyConnectedLayer(1000,"Name","fc_4")
    reluLayer("Name","relu7")
    dropoutLayer(0.5,"Name","drop7")
    fullyConnectedLayer(10,"Name","fc_1")
    tanhLayer("Name","tanh_1")
    fullyConnectedLayer(2,"Name","fc_2")
    tanhLayer("Name","tanh_2")
    regressionLayer("Name","regressionoutput")];
plot(layerGraph(layers));
RedArquitectura=layerGraph(layers);

options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...  %periodo de cada cuanto queremos que haya
un drop, para 10 iteraciones

 'LearnRateDropFactor',0.3,... %ratio inicial un 30 % de ellas
    'Momentum',0.9, ... %Suavizar el gradiente, para que no cambie
bruscamente
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'MaxEpochs',30, ...   % cuantas épocas máximas
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ... % Que se barajeen en cada epoch el orden de
los patrones
    'CheckpointPath', tempdir, ... %Si queremos poner un path de chequeo
    'VerboseFrequency',2,...
    'Plots','training-progress');  %Si queremos que nos saque las gráficas
%Con que criterio vamos a parar en caso de que el error de validación suba


 [Red, info] = trainNetwork(comDS2RL,RedArquitectura,options);

save all
save ("RegresionLinea",'Red','-v7.3');
reset(comDS2RL)
while hasdata(comDS2RL)
    data=read(comDS2RL);
    Prediccion = predict(Red,data{1,1});
    imshow(data{1,1})
    hold on

plot(data{1,2}(1)*targetSize(1),data{1,2}(2)*targetSize(2),'xr','MarkerSize',
5)

plot(Prediccion(1)*targetSize(1),Prediccion(2)*targetSize(2),'ob','MarkerSize
',5)
    i=i+1;
    hold off
    pause(0.1)
```

```
end
```

CustomResize

```
function out=customResize(filename,targetSize)
    x=imread(filename);
    imRes=imresize(x,targetSize);
    imRes=double(imRes)./255;
    out=imRes;
end
```

DataNormalized

```
function [datan]=DataNormalized(x,TarjetSize)
    datan=x;
    datan(1)=datan(1)/tarjetSize(1);
    datan(2)=datan(2)/tarjetSize(2);
end
```