



UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA



ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Optimización de redes de aprendizaje profundo para
detección de eventos en situaciones de catástrofe en
sistemas embarcados

GRADO DE INGENIERÍA ELECTRÓNICA ROBÓTICA Y MECATRÓNICA

Departamento de Ingeniería de Sistemas y Automática

AUTOR: Pedro Antonio Peña Puente

TUTOR: Dr. D. Ricardo Vázquez Martín

21 de febrero de 2023



UNIVERSIDAD
DE MÁLAGA





Resumen

En el presente documento se desarrolla un sistema de reconocimiento de objetos habituales en situaciones de rescate y emergencias para sistemas embarcados.

Este sistema se ha realizado usando técnicas de aprendizaje profundo con un algoritmo de fusión a las cuales se ha añadido una parte de comunicación para su implementación en tiempo real. En concreto, se ha utilizado la red neuronal YOLOv8 tras su correspondiente estudio debido a su gran rendimiento, explicado y desarrollado más adelante, para la detección de objetos en imágenes.

Se han entrenado dos tipos de redes con YOLOv8, una red enfocada a la detección de objetos en imágenes dentro del espectro termográfico y otra dentro del espectro visible. El conjunto de imágenes utilizado es propiedad de la Universidad de Málaga y procede de las Jornadas sobre Seguridad y Emergencias realizadas en los años 2018 y 2019 en la Escuela de Ingenierías Industriales de esta Universidad. Tras este entrenamiento, se ha creado un algoritmo heurístico de fusión para mejorar el resultado obtenido por ambas redes y poder detectar mejor los objetos. Este algoritmo ha sido comprobado con el análisis de una pequeña muestra para demostrar su buen funcionamiento.

Finalmente, se ha desarrollado un sistema de comunicación mediante ROS2 que permite implementar este proyecto dentro de un robot terrestre y ejecutar y enviar todo en tiempo real. El sistema es capaz de recibir tanto imágenes térmicas como del espectro visible, procesarlas con las redes neuronales, leer los resultados, fusionarlos y crear y enviar un archivo con la información de los objetos dentro de la imagen.

Palabras clave

Deep Learning, YOLOv8, Red neuronal convolucional, Dataset, Entrenamiento, Fusión, Implementación en tiempo real, ROS2, Desastre, Térmico, RGB.



UNIVERSIDAD
DE MÁLAGA





Abstract

This paper develops a system for recognizing common objects in rescue and emergency situations for embedded systems.

This system has been developed using deep learning techniques with a fusion algorithm, to which a communication component has been added for real-time implementation. Specifically, the YOLOv8 neural network has been employed after a thorough study, given its excellent performance in object detection in images.

Two types of networks have been trained with YOLOv8: one focused on object detection in images within the thermal spectrum and another within the visible spectrum. The image dataset used belongs to the University of Malaga and comes from the Security and Emergency Sessions held in the years 2018 and 2019 at the Faculty of Industrial Engineering of this University. After this training, a heuristic fusion algorithm has been created to enhance the results obtained by both networks and improve object detection. This algorithm has been tested with the analysis of a small sample to demonstrate its effectiveness.

Finally, a communication system using ROS2 has been developed, enabling the implementation of this project in a terrestrial robot and the execution and transmission of data in real-time. The system is capable of receiving both, thermal and visible spectrum images, processing them with neural networks, reading the results, merging them, and creating and sending a file with information about the objects within the image.

KeyWords

Deep Learning, YOLOv8, Convolutional neural network, Dataset, Training, Fusion, Real-time implementation, ROS2, Disaster, Thermal, RGB.



UNIVERSIDAD
DE MÁLAGA





Declaración de originalidad

Yo, Pedro Antonio Peña Puente, con DNI, declaro bajo mi responsabilidad que el Trabajo Fin de Grado en Ingeniería Electrónica, Robótica y Mecatrónica, mención en Robótica y Automatización, presentado en la Escuela de Ingenierías Industriales de la Universidad de Málaga bajo el título “Optimización de redes de aprendizaje profundo para detección de eventos en situaciones de catástrofe en sistemas embarcados” es original (no es copia ni adaptación de otra), inédito (no ha sido difundido por ningún medio, incluyendo Internet) y no ha sido presentado con anterioridad por él/ella mismo/a o por otra persona, ni total ni parcialmente.

Además, las fuentes de información consultadas han sido debidamente mencionadas y referenciadas en dicho documento.

Y para que así conste a los efectos oportunos, se firma el presente documento.

En Málaga a 14 de enero de 2023

Fdo.: Pedro Antonio Peña Puente



UNIVERSIDAD
DE MÁLAGA





Índice

Resumen	3
Palabras clave	3
Abstract	5
KeyWords	5
Declaración de originalidad.....	7
Acrónimos	13
Índice de figuras	15
Índice de tablas	17
1. Antecedentes y Objeto	19
1.1. Antecedentes	19
1.2. Objeto	19
1.3. Plan de Trabajo	20
1.4. Estructura de la memoria.....	20
2. Fundamentos Teóricos	23
2.1. Enfoque del proyecto	23
2.2. Introducción.....	25
2.2.1. ¿Qué es la Inteligencia Artificial?	25
2.2.2. Redes neuronales	25
2.2.3. Machine Learning	27
2.2.4. Deep Learning	27
2.3. Redes Neuronales Convolucionales	28
2.3.1. Operación de Convolución	29
2.3.2. Función de activación.....	30
2.3.3. Capa Max Pooling	34
2.3.4. Capa Flatten.....	34
2.3.5. Capa Fully Connected.....	35
2.3.6. Supresión de no máximos	35
2.3.7. Fases de entrenamiento de una RNC.....	36
2.3.8. Hiperparámetros	36
2.3.9. Evaluación de la red.....	37
2.3.10. Análisis de errores.....	38



2.4.	YOLOv8.....	40
2.4.1.	Arquitectura de la red.....	40
2.4.2.	Algoritmo YOLO.....	42
2.4.3.	Novedades YOLOv8.....	43
2.4.4.	Modelos y tamaños de YOLOv8	44
2.5.	Comunicación por ROS	46
3.	Estudios Previos y Entorno de Trabajo	47
3.1.	Proyecto de partida	47
3.2.	YOLOv8.....	47
3.2.1.	Docker	47
3.2.2.	Pytorch	48
3.2.3.	CUDA	48
3.2.4.	ULTRALYTICS	48
3.2.5.	OpenCV.....	48
3.2.6.	Matplotlib.....	48
3.2.7.	Thop.....	49
3.2.8.	Pandas	49
3.2.9.	Seaborn.....	49
3.2.10.	ClearML.....	49
4.	Fases de Entrenamiento.....	51
4.1.	Introducción.....	51
4.2.	Red RGB o del espectro visible	51
4.3.	Red térmica.....	57
4.4.	Mejora de las redes	62
5.	Fusión de las redes	63
5.1.	Introducción.....	63
5.2.	Fusión probabilística.....	63
5.3.	Fusión heurística inicial	65
5.4.	Fusión heurística final.....	65
5.4.1.	Validación	66
5.4.2.	Resultados	69
6.	Implementación de la red	73
6.1.	Introducción.....	73
6.2.	Comunicación mediante ROS2	73



6.3. Análisis de resultados	74
7. Conclusiones y Trabajo Futuro	77
7.1. Conclusiones	77
Entrenamiento de las redes térmica y RGB	77
Sincronización de imágenes térmicas y RGB	77
Fusión de las redes	78
Conclusión final	78
7.2. Trabajo Futuro	78
Anexo A: Docker	81
A.1. Introducción	81
A.2. Dockerfile para el entrenamiento	81
A.3. Creación del contenedor	83
Bibliografía	85



UNIVERSIDAD
DE MÁLAGA



Acrónimos

ANN	Artificial Neural Network (Red Neuronal Artificial)
AP	Average Precision (Precisión Media)
CGM	Confianza Global Muestral
CPU	Central Processing Unit (Unidad Central de Procesamiento)
CNN	Convolutional Neural Network (Red Neuronal Convolucional)
COCO	Common Objects in Context (Objetos Comunes en Contexto)
CUDA	Compute Unified Device Architecture (Arquitectura Unificada de Dispositivos de Cómputo)
cuDNN	CUDA Deep Neural Network library (Librería de CUDA para Red Neuronal Profunda)
FN	Falso Negativo
FP	Falso Positivo
GPU	Graphics Processing Unit (Unidad de Procesamiento Gráfico)
IOU	Intersection Over Union (Intersección sobre la Unión)
mAP	Mean Average Precision (Precisión Media Promedio)
MCI	Media de Confianzas Individuales
P	Precision (Precisión)
PR	Precision-Recall (Precisión-Sensibilidad)
R	Recall (Sensibilidad)
RGB	Red-Green-Blue (Rojo-Verde-Azul, define conjunto de ondas dentro del espectro visible)
ROS	Robot Operating System (Sistema Operativo Robótico)
UMA	Universidad de Málaga
VP	Verdadero Positivo
VN	Verdadero Negativo
YOLO	You Only Look Once (Solo Miras Una Vez)



UNIVERSIDAD
DE MÁLAGA



Índice de figuras

2.1.	Fase inicial de entrenamiento.....	23
2.2.	Fase intermedia de fusión	24
2.3.	Fase final de implementación	24
2.4.	Estructura de una neurona humana	26
2.5.	Modelo de perceptrón multicapa	26
2.6.	Deep Learning frente a Machine Learning e Inteligencia Artificial	28
2.7.	Estructura de una Red Neuronal Convolucional	29
2.8.	Función ReLU.....	31
2.9.	Función sigmoide.....	31
2.10.	Función tangente hiperbólica.....	32
2.11.	Función Leaky ReLU.....	33
2.12.	Función Softmax.....	33
2.13.	Capa Max Pooling	34
2.14.	Capa Flatten	34
2.15.	Capa Fully Connected	35
2.16.	Red neuronal sin dropout vs con dropout	37
2.17.	Definición de IoU	38
2.18.	Subajuste, ajuste apropiado y sobreajuste	40
2.19.	Arquitectura de un detector de objetos moderno	41
2.20.	Arquitectura de la red YOLOv8	42
2.21.	Modelo de detección de YOLO	43
2.22.	Aumento en mosaico	44
2.23.	Rendimiento modelos YOLOv8	45
4.1.	Matriz de confusión de la red RGB.....	53
4.2.	Instancias de cada clase en la red RGB.....	54
4.3.	Curva Precisión-Sensibilidad de la red RGB	55
4.4.	Métricas del entrenamiento de la red RGB	55



4.5.	Detección de personal de rescate junto a un vehículo de rescate con la red RGB	56
4.6.	Detección de distinto personal de rescate con la red RGB	56
4.7.	Matriz de confusión de la red térmica.....	58
4.8.	Instancias de cada clase en la red	59
4.9.	Curva Precisión-Sensibilidad de la red térmica	60
4.10.	Métricas del entrenamiento de la red térmica	60
4.11.	Detección de personal de rescate junto a un vehículo de rescate con la red RGB	61
4.12.	Detección de distinto personal de rescate con la red térmica.....	61
5.1.	Etiquetado con LabelImg	67
5.2.	Comparación de las redes para imagen con dos personas de rescate [Fusión-RGB-Térmica]	70
5.3.	Comparación de las redes para imagen con dos vehículos de emergencia y una persona de rescate [Fusión-RGB-Térmica]	70
5.4.	Comparación de las redes para imagen con una persona de rescate [Fusión-RGB-Térmica]	70
5.5.	Comparación de las redes para imagen con distinto personal de rescate [Fusión-RGB-Térmica]	71
A.1.	Dockerfile utilizado para crear la imagen de Docker.....	82
A.2.	Comando docker build.....	83
A.3.	Comando docker run.....	83



Índice de tablas

2.1.	Especificaciones de los tamaños de YOLOv8	45
4.1.	Hiperparámetros utilizados por las redes RGB y térmica.....	52
4.2.	Resultados del entrenamiento de la red rgb.....	54
4.3.	Resultados del entrenamiento de la red térmica.....	58
5.1.	Tabla de probabilidad	63
5.2.	Extracto del archivo .csv creado	68
5.3.	Resultados de la muestra	69
6.1.	Tiempos de procesamiento de las redes	75
6.2.	Estadísticas del tiempo de ejecución.....	76



UNIVERSIDAD
DE MÁLAGA



Capítulo 1

1. Antecedentes y Objeto

1.1. Antecedentes

La *Inteligencia Artificial* (IA) es una tecnología muy reciente que surgió como idea a mediados del siglo pasado. El concepto máquinas inteligentes fue algo muy revolucionario, pero que en esa época no podía ser completamente desarrollado debido a las carencias tecnológicas de entonces. Conforme la capacidad de computación de las máquinas fue avanzando, problemas más complejos pudieron ser resueltos y esto generó el inicio del aprendizaje automático [1]. El aprendizaje automático empezó con las redes neuronales propias del *Machine Learning*, con una estructura relativamente simple, y culminó con la creación del *Deep Learning*, cuya arquitectura neuronal es bastante más compleja.

Por otro lado, la robótica es otra tecnología disruptiva que ha ido avanzando a gran velocidad durante los últimos años. Esto es debido, en parte, a la gran cantidad de utilidades que puede llegar a tener esta tecnología, lo cual fomenta su constante investigación. Una de esas utilidades es la robótica de rescate, la cual consiste en el diseño, desarrollo y aplicación de robots para operaciones de búsqueda y rescate.

La robótica de rescate surgió para mejorar la eficiencia y evitar riesgos humanos en operaciones de búsqueda y rescate tales como desastres naturales, accidentes industriales u operaciones militares [2]. Este campo, al igual que los dos anteriores, está en constante evolución lo cual nos permitirá observar sus importantes progresos en los años venideros.

1.2. Objeto

Es entre estos tres campos donde estará centrado este proyecto. En el uso del Aprendizaje Profundo para la detección de objetos en tiempo real e implementación en un robot de rescate.

Se propone emplear una red neuronal convolucional (Aprendizaje Profundo) actual para mejorar el resultado obtenido por el compañero Adrián Bañuls Arias en su TFG de 2019. Se entrenarán dos redes neuronales, una para la identificación de objetos en situaciones de rescate a través de una cámara RGB (Red-Green-Blue, visión en el espectro visible) y otra para identificación de objetos en situaciones de rescate a través de una cámara térmica. Una vez se han mejorado los resultados de ambas redes se procederá a intentar fusionarlas y, por último, se procederá a implementar esta “detección fusionada” en un robot terrestre destinado a misiones de rescate mediante el uso del **Sistema Operativo Robótico** (también conocido como **ROS** por sus siglas en inglés, *Robot Operating System*). Este es un sistema operativo de código abierto para robots, muy utilizado para robots de

rescate, que proporciona una amplia variedad de herramientas y bibliotecas para el desarrollo de software de robótica de manera eficiente y modular [3].

1.3. Plan de Trabajo

El desarrollo del trabajo se realizará en 7 partes distintas:

1. Estudio bibliográfico inicial [4]
2. Selección de las herramientas óptimas para este trabajo: lenguajes, arquitectura de las redes a utilizar y librerías
3. Entrenamiento de las redes neuronales
4. Fusión de las redes neuronales
5. Implementación en el robot terrestre
6. Validación de los resultados
7. Documentación

1.4. Estructura de la memoria

La memoria de este proyecto consta de siete capítulos y un anexo:

- **Capítulo 1:** Antecedentes y objeto. Explicación breve del proyecto junto a sus objetivos y pasos a seguir a largo del mismo.
- **Capítulo 2:** Fundamentos Teóricos. Se comienza con un resumido enfoque del proyecto seguido de una breve introducción a la inteligencia artificial para después, explicar el funcionamiento del Aprendizaje profundo y de las redes neuronales convolucionales junto al sistema de comunicación que se va a utilizar, ROS2.
- **Capítulo 3:** Estudios previos y entorno de trabajo de la red neuronal YOLOv8 y del TFG de partida. Explicación del punto desde el que se parte gracias a un TFG anterior junto al desarrollo de las librerías y del entorno donde se ha trabajado.
- **Capítulo 4:** Fases de entrenamiento de las redes y obtención de resultados. Desarrollo de todo el trabajo de entrenamiento de las redes.
- **Capítulo 5:** Fusión de las redes. Desarrollo de los algoritmos creados para hacer la fusión de las redes neuronales y validación.
- **Capítulo 6:** Configuración del ordenador de a bordo Jetson Orin e implementación de la red mediante el sistema de comunicación ROS2.



- **Capítulo 7:** Conclusiones del trabajo y líneas de desarrollo futuro.
- **Anexo A:** Docker.



UNIVERSIDAD
DE MÁLAGA



Capítulo 2

2. Fundamentos Teóricos

2.1. Enfoque del proyecto

Para abordar este proyecto se ha tenido que hacer un estudio en profundidad de todos los campos involucrados en él. Empezando por las redes neuronales, teoría de probabilidad, tratamiento de imágenes o comunicación de sistemas. Posteriormente, se empezó a trabajar en el proyecto.

En primer lugar, se empezó decidiendo qué arquitectura de red neuronal se iba a usar para entrenar las nuestras. Una vez se decidió, se empezó a configurar nuestro contenedor de Docker. Tras ello, se entrenaron las redes neuronales [Véase la Figura 2.1].

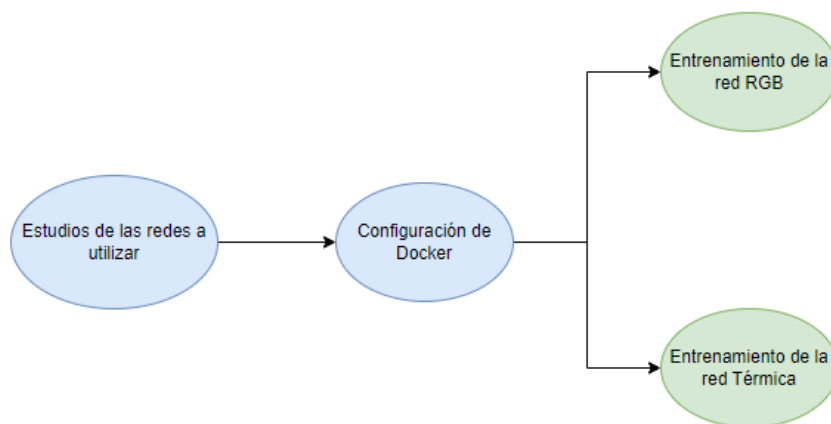


Figura 2.1: Fase inicial de entrenamiento

Tras acabar la fase del entrenamiento de las redes, empezó la fase de la creación del algoritmo de fusión. Esta fase fue una de las más complejas debido a la búsqueda del propio algoritmo. Se empezó estudiando temas de probabilidad para poder implementarlos dentro del algoritmo. También se crearon varios códigos donde se comparaban ambas redes, algo que fue complejo. Una vez se consiguió se fueron creando varias versiones del algoritmo, pero no fue hasta la tercera donde se obtuvo el resultado final. Cabe destacar que todas las versiones difieren en gran medida unas de otras, por lo que no eran simples actualizaciones. En esta última versión se desestimó el uso de leyes probabilísticas debido a fallos explicados más adelante. Para finalizar esta parte, se validó el resultado con una pequeña muestra que se etiquetó a mano y que requirió de varios códigos bastante complejos de crear [Véase la Figura 2.2].

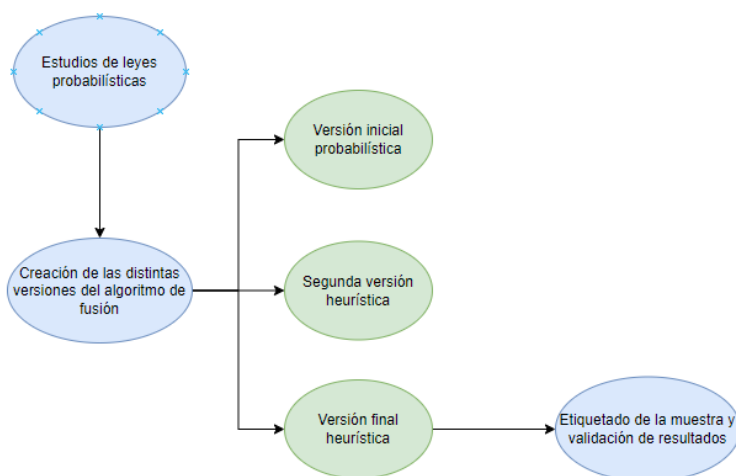


Figura 2.2: Fase intermedia de fusión

La fase final del proyecto trató de la implementación de todo el algoritmo de fusión en ROS2 para su procesamiento en tiempo real. En primer lugar, se creó un código capaz de leer vídeos y transmitir los frames de cada vídeo mediante un topic de ROS2. Tras ello, se creó otro código capaz de leer estas imágenes mediante ROS2, procesarlas con las redes neuronales, fusionar las detecciones mediante el algoritmo previamente creado, y enviar el resultado mediante otro topic de ROS2 para poder ser leído desde otro lugar. Por último, se creó una aplicación capaz de leer las imágenes iniciales junto al resultado anterior y crear una imagen con la detección final superpuesta, enfocado a su uso en un puesto de control remoto (habitual en misiones de rescate). Además, este código muestra tanto la detección final como las detecciones de las redes neuronales entrenadas [Véase la Figura 2.3]. También se realizó un estudio de la velocidad de este sistema dentro de un ordenador de a bordo para su procesamiento en tiempo real.

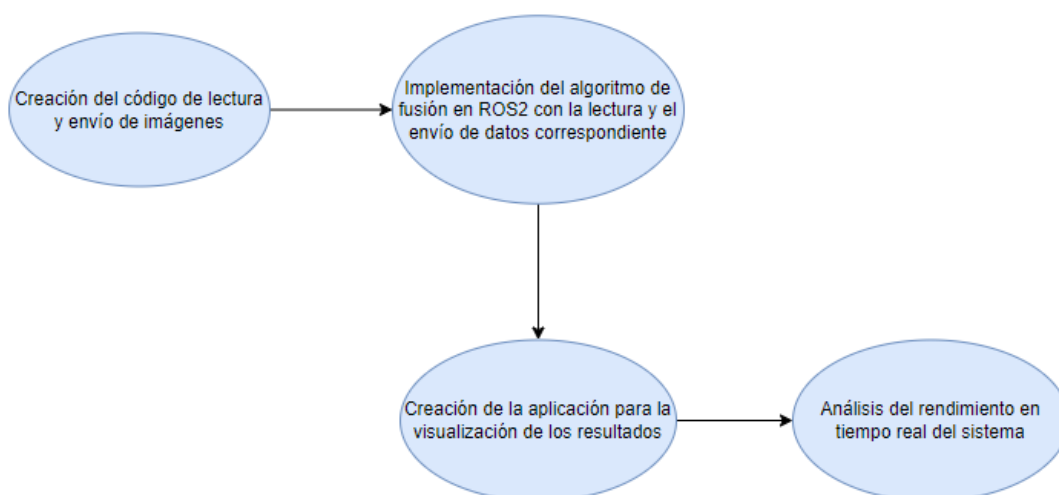


Figura 2.3: Fase final de implementación

2.2. Introducción

2.2.1. ¿Qué es la Inteligencia Artificial?

La Inteligencia Artificial (IA) es un campo de la informática que se enfoca en crear sistemas complejos para los que normalmente se necesita inteligencia humana como el aprendizaje, el razonamiento, la creatividad o la percepción, pero mediante un ordenador [4].

Algo tan simple para los humanos como es identificar a una persona o a un objeto, para los ordenadores es muy complicado, pero a través de la inteligencia artificial esto es posible de hacer. Utilizando métodos matemáticos muy complejos y un conjunto de datos gigantesco, el ordenador es capaz de reconocer patrones tal y como hacemos nosotros, e incluso en muchos casos mejor. Todo este campo ha sido posible gracias al desarrollo tecnológico y a la mejora de la computación.

Hoy en día la Inteligencia Artificial se encuentra en una infinidad de campos como puede ser el sistema de recomendación de películas de una aplicación, el diagnóstico de enfermedades a través de imágenes o incluso la detección automática de las matrículas de los coches en los parkings. En nuestro caso, nos centraremos en la aplicación al campo de la robótica con el *Deep Learning* (Aprendizaje Profundo).

2.2.2. Redes neuronales

El concepto del Aprendizaje Profundo está basado en el concepto original del cerebro humano. Nuestro cerebro funciona mediante la conexión de miles de millones de neuronas que transmiten impulsos eléctricos y químicos con señales que son interpretadas y generan los pensamientos y las emociones [5]. Las neuronas reciben la información mediante unas ramificaciones llamadas dendritas, la cual llega al núcleo de la neurona y cuando supera un cierto umbral transmite esta información por el axón a otras neuronas [Véase Figura 2.4]. Este conjunto de neuronas conectadas es lo que se conoce como **Redes Neuronales**.

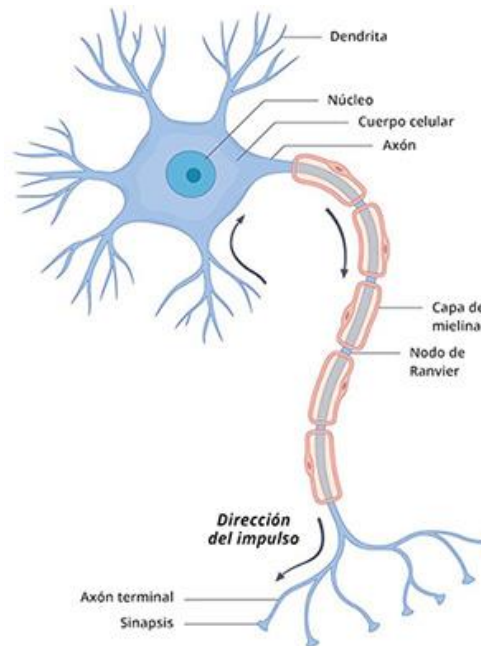


Figura 2.4: Estructura de una neurona humana [6]

Este funcionamiento es replicado mediante un algoritmo matemático en las **redes neuronales artificiales**, también conocido como **perceptrón multicapa o simple**. El algoritmo recibe una serie de entradas o características que se transmiten hacia unos nodos o **neuronas** con unos **pesos**. Estos pesos se utilizan para pasar de una capa a otra y se van modificando automáticamente conforme la red va entrenando hasta que se obtienen los pesos óptimos para cada caso con los que se obtiene la mejor salida posible de las neuronas [7]. La estructura de estas redes neuronales suele estar dividida en 3 partes: una **capa de entrada** que representa las características iniciales de la red, una o varias **capas ocultas** donde se genera la complejidad del algoritmo y una **capa de salida** que expresa la detección final de la red [Véase Figura 2.5].

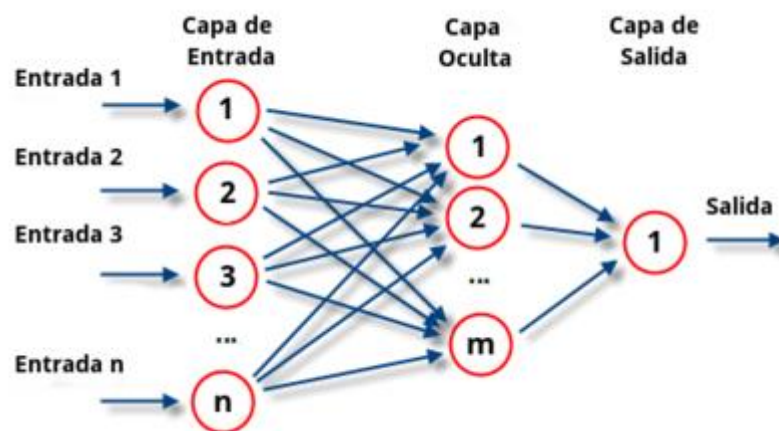


Figura 2.5: Modelo de perceptrón multicapa [8]

2.2.3. Machine Learning

El *Machine Learning* o Aprendizaje Automático es la parte de la inteligencia artificial que es capaz de identificar patrones mediante algoritmos basados en el aprendizaje de la máquina. Se puede clasificar en 4 grupos:

- **Aprendizaje Supervisado:** algoritmos que precisan de datos de entrada y salida etiquetados para el aprendizaje de la red. La parte más tediosa de estos algoritmos es la obtención de estos datos ya que normalmente se consiguen de forma manual (supervisión humana). Se usa para clasificación de imágenes, documentos o palabras escritas, así como para predecir tendencias y resultados futuros a través de patrones en los datos [9].
- **Aprendizaje No Supervisado:** algoritmos que no precisan de preprocesamiento de datos ni de la etiquetación de los mismos. Estos algoritmos requieren mucha menos intervención humana y están enfocados a la comprensión y agrupación de conjuntos de datos, como sistemas de recomendación o reconocimiento de patrones [10].
- **Aprendizaje Semi-Supervisado:** algoritmos que mezclan los dos anteriores. Utilizan una pequeña parte de datos etiquetados y una parte mayor de datos no etiquetados. Estos algoritmos facilitan el entrenamiento para aquellos casos en los que es complicado encontrar datos etiquetados [11].
- **Aprendizaje por Refuerzo:** algoritmos que utilizan la experiencia propia de la máquina para aprender mediante la optimización de los datos obtenidos. A partir de estos datos, emprenderá acciones que repetirá y reforzará dependiendo de las recompensas positivas o negativas que reciba [12].

En este proyecto se utilizará el primer grupo, aprendizaje supervisado, ya que es el óptimo para nuestro caso de detección de objetos en imágenes.

2.2.4. Deep Learning

El *Deep Learning* o Aprendizaje Profundo es una rama del *Machine Learning* en la cual se usan redes neuronales muy complejas o “profundas” que requieren una mayor cantidad de cómputo de datos pero que son capaces de obtener mejores resultados para datos no estructurados como imágenes, vídeos o textos. Para entender mejor la estructura formada por el *Deep Learning*, el *Machine Learning* y la Inteligencia Artificial, se puede observar la Figura 2.6.

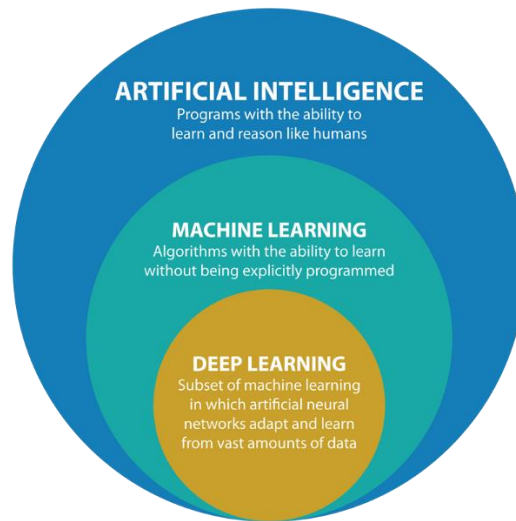


Figura 2.6: *Deep Learning* frente a *Machine Learning* e Inteligencia Artificial [13]

Como se mencionó anteriormente, el *Deep Learning* ha sido posible gracias al gigantesco desarrollo tecnológico que ha permitido crear mejores **GPUs** (*Graphics Processing Units*) con una capacidad de cómputo abismal en relación al siglo pasado, lo cual permite el cálculo de las redes profundas en tiempos asequibles. Sin embargo, la tecnología está en continuo desarrollo por lo que cada vez habrá mejores tarjetas gráficas que permitan el cálculo de redes más y más complejas en un tiempo minúsculo.

El *Deep Learning* surgió a raíz de las **Redes Neuronales Artificiales** (RNA) y se puede clasificar en 5 tipos:

- **Redes Neuronales Convolucionales** (RNC): este tipo de redes usan capas convolucionales capaces de adaptarse mejor a datos no estructurados. Son muy utilizadas para procesar y clasificar imágenes y objetos dentro de ellas.
- **Redes Neuronales Recurrentes** (RNR): son redes capaces de procesar datos secuenciales, es decir, de tener una especie de memoria a corto plazo. Son muy utilizadas para el análisis y traducción de textos [14].
- **Redes Neuronales Generativas Adversarias** (GAN): se entrenan dos redes que se oponen para crear un contenido nuevo que simula ser real. Se utilizan para generar mundos virtuales o imágenes con un resultado muy realista (*deepfakes*) [15].

2.3. Redes Neuronales Convolucionales

En el caso de este proyecto se usarán dos Redes Neuronales Convolucionales, por lo tanto, se explicarán un poco más en profundidad estas en específico.

Las **Redes Neuronales Convolucionales** son una variante de las Redes Neuronales Artificiales, pero con una capacidad adicional, además de las capas aritméticas tradicionales de las RNA (Redes Neuronales Artificiales), estas redes contienen capas convolucionales

que permiten extraer mejor las características de una imagen o vídeo para procesarlas y detectar objetos. Al igual que las RNA, estas redes contienen a su vez **funciones de activación**, que modifican los pesos antes de entrar a cada neurona; capas de *max pooling*, que reduce la complejidad de la imagen para tratarla más fácilmente; una capa *flatten* (o de aplanamiento), que reduce las dimensiones de los datos (3 para una imagen) a un vector unidimensional; una capa *fully connected* [16], que conecta todas las neuronas entre sí y ayuda a mapear la representación entre la entrada y la salida de la red; y una capa *softmax*, que normaliza entre 0 y 1 las probabilidades resultantes de cada salida. Aunque estas partes se explicarán un poco más en profundidad después [Véase la Figura 2.7].

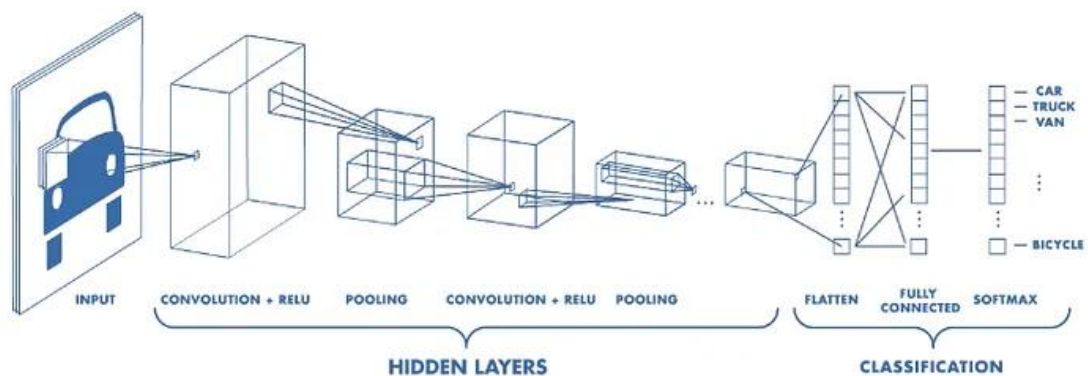


Figura 2.7: Estructura de una Red Neuronal Convolutiva [16]

Pero para entender mejor estas redes, hay que explicar que es la operación de convolución y por qué son tan buenas para el caso de las imágenes.

2.3.1. Operación de Convolución

“La convolución se define como la integral del producto de dos funciones, en el dominio del tiempo, después de desplazar una de ellas una distancia t ” [17]. Esta operación es muy utilizada en el campo del Procesamiento de Señales o de la Electrónica y se puede representar como un asterisco. Para entender su fórmula matemática mostrada en la ecuación 2.1, se parte de dos funciones f y g que se explican a continuación.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\eta)g(t - \eta)d\eta \quad (2.1)$$

En el caso de las imágenes, estas funciones son las matrices que componen cada imagen (f , comúnmente llamada *input*) junto a otra matriz que hace de filtro (g , también llamada *kernel*), lo cual permite cambios en las imágenes tales como aumento de la

nitidez, difuminación, detección de ejes y otros realces basados en el *kernel*, que ayudarán a la detección de rasgos en las imágenes.

Sin embargo, aunque se pueden entender cómo funcionan las modificaciones mediante convolución en las imágenes, en el caso de las redes neuronales, el sistema funciona en parte como caja negra. Tú puedes diseñar una red neuronal por capas como la de convolución o *max pooling* y modificar algunos parámetros de cada capa, pero al final, los filtros internos que usará la red no sabes cuáles serán. Durante el entrenamiento la red irá probando con distintos filtros y parámetros hasta obtener el mejor resultado posible (aunque esto también dependerá del tiempo de entrenamiento y la capacidad de cómputo del dispositivo donde se ejecute).

2.3.2. Función de activación

La función de activación es un parámetro de la red neuronal que actúa como filtro o función limitadora de cada capa, permitiendo pasar solo a los valores que están en un rango específico de una neurona a otra. Dicho de otra forma, es una función que transmite la información generada por la combinación lineal de los pesos y las entradas. Dado que las redes neuronales están diseñadas para resolver sistemas cada vez más complejos, las funciones de activación permiten crear modelos no lineales con mayor complejidad resolutive [18]. Las funciones de activación más famosas e utilizadas son las siguientes:

- **Función ReLU:** Transforma los valores introducidos eliminando los negativos y dejando igual los positivos, por lo tanto, esta función solo se activa si los valores son positivos. Además, también se caracteriza por no estar acotada, puede generar la muerte de demasiadas neuronas, funciona bien con imágenes, y tiene un buen rendimiento con redes convolucionales [19]. Su expresión matemática se muestra en la ecuación 2.2.

$$f(x) = \max(0, x) = \begin{cases} 0 & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases} \quad (2.2)$$

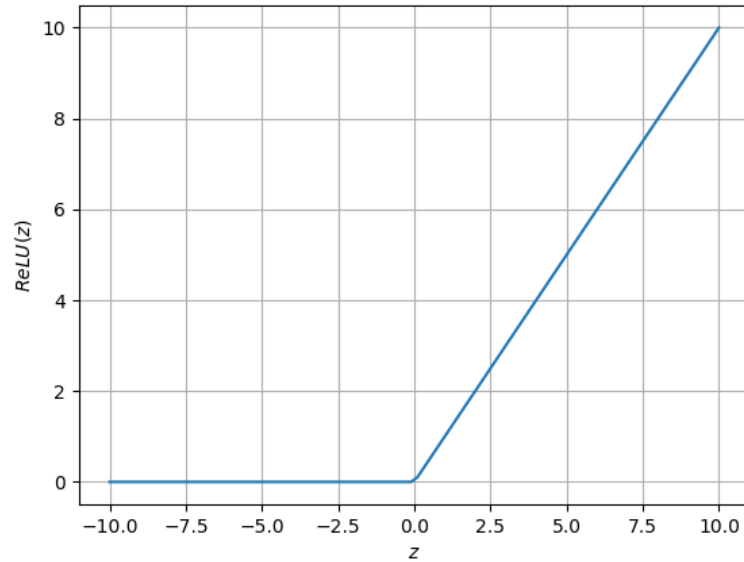


Figura 2.8: Función ReLU

- **Función sigmoide:** Transforma los valores introducidos a una escala (0,1), los valores más altos pasan a valer 1 mientras que los más bajos pasan a valer 0. Esta función satura y mata el gradiente, genera una lenta convergencia, no está centrada en el 0, está acotada entre 0 y 1, y presenta un buen rendimiento en la última capa [19]. Su expresión matemática se muestra en la ecuación 2.3.

$$f(x) = \frac{1}{1+e^{-x}} \quad (2.3)$$

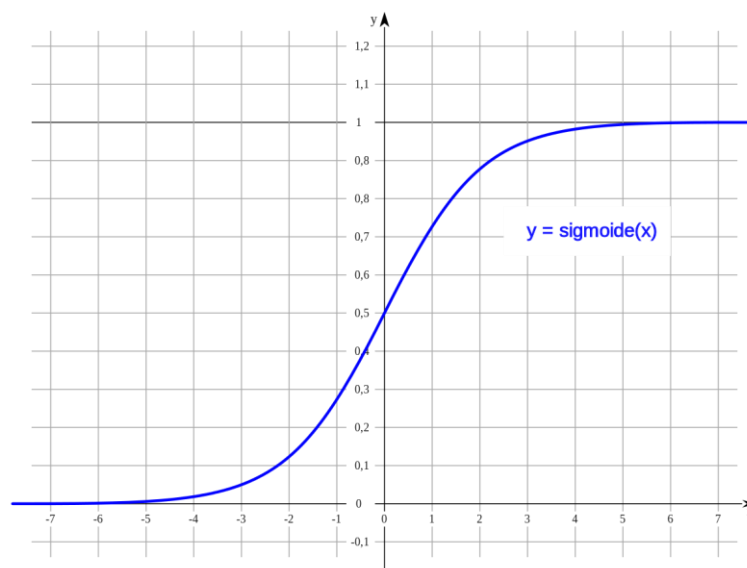


Figura 2.9: Función sigmoide

- **Función tangente hiperbólica:** Transforma los valores introducidos a una escala $(-1,1)$, tendiendo los valores altos a 1 y los valores bajos a -1. Esta función es muy similar a la sigmoide, satura y mata el gradiente, genera una lenta convergencia, está centrada en el 0, está acotada entre -1 y 1, se utiliza para elegir entre una opción y la contraria, y presenta un buen rendimiento en redes recurrentes [19]. Su expresión matemática se muestra en la ecuación 2.4.

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \quad (2.4)$$

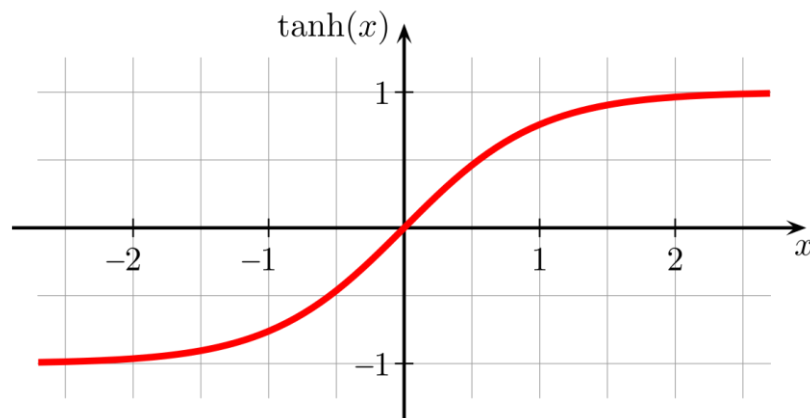


Figura 2.10: Función tangente hiperbólica

- **Función Leaky ReLU:** Transforma los valores introducidos multiplicando los negativos por un coeficiente rectificativo y dejando los positivos inalterados. Esta función es similar a la *ReLU*, penaliza los valores negativos mediante un coeficiente rectificador, no está acotada, funciona bien con imágenes y, por lo tanto, con redes convolucionales. También se usa para entrenar Redes GAN (*Generative Adversarial Networks*) [19]. Su expresión matemática se muestra en la ecuación 2.5.

$$f(x) = \begin{cases} ax & \text{para } x < 0 \\ x & \text{para } x \geq 0 \end{cases} \quad (2.5)$$

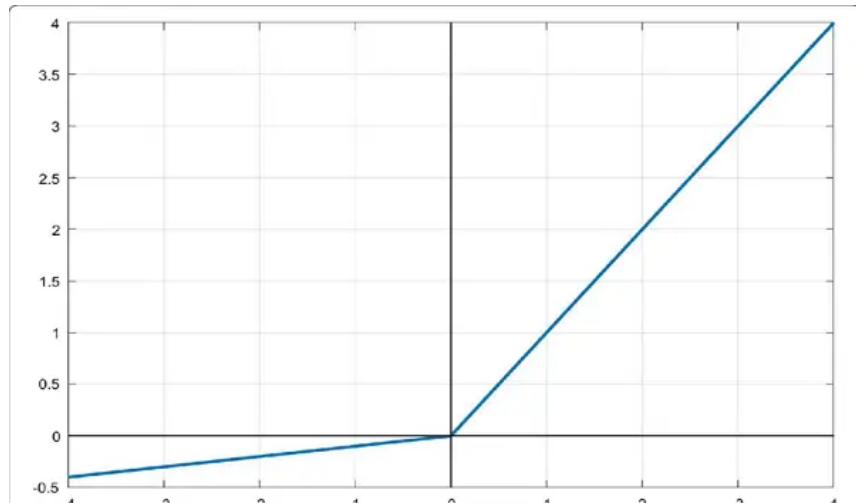


Figura 2.11: Función Leaky ReLU

- **Función Softmax:** Transforma las salidas de las neuronas en forma de probabilidad normalizada entre 0 y 1, de manera que el sumatorio de todas las probabilidades de las salidas dé 1. Esta función se utiliza cuando queremos tener una representación en forma de probabilidades, está acotada entre 0 y 1, es muy diferenciable del resto, se utiliza para normalizar el tipo multi-casos, y tiene un buen rendimiento en las últimas capas [19]. Su expresión matemática se muestra en la ecuación 2.6.

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}} \quad (2.6)$$

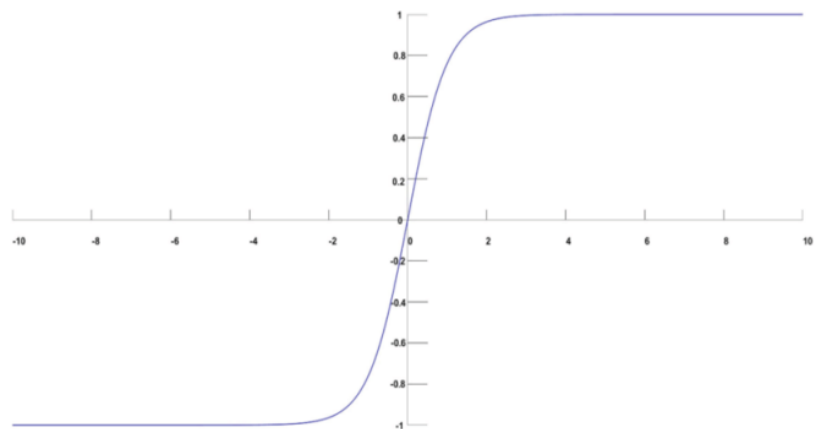


Figura 2.12: Función *Softmax*

2.3.3. Capa Max Pooling

La capa de *Max Pooling* ejecuta una operación de agrupación que calcula el valor máximo para conjuntos dentro de un mapa de características, es decir, si se aplica un filtro 3x3, en cada dimensión de la imagen (RGB) se agruparán conjuntos de 3x3 y en esos conjuntos se buscará el valor máximo el cuál será el que sustituya a esa matriz en “la nueva imagen”. Esta capa suele ser utilizada tras la capa convolucional, lo que simplifica la imagen para la red neuronal teniendo un menor coste computacional y extrayendo características más genéricas de la imagen. Se puede observar su funcionamiento con el ejemplo de la Figura 2.13.

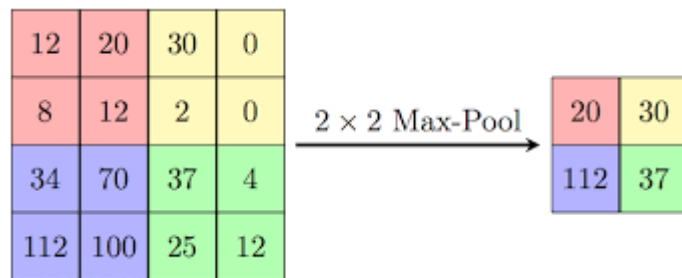


Figura 2.13: Capa *Max Pooling* [20]

2.3.4. Capa Flatten

La capa de *Flatten* (o adelgazamiento) se encarga de adaptar el mapa de características que es recibido de la capa *Max Pooling* para que la última capa *Fully Connected* sea capaz de interpretarla [21]. Esto lo hace reduciendo la dimensionalidad de las características extraídas para hacerla unidimensional, es decir, por ejemplo, con un conjunto de dos matrices 2x2 las modifica y las convierte en un vector de una dimensión con 8 posiciones como se muestra en la Figura 2.14.

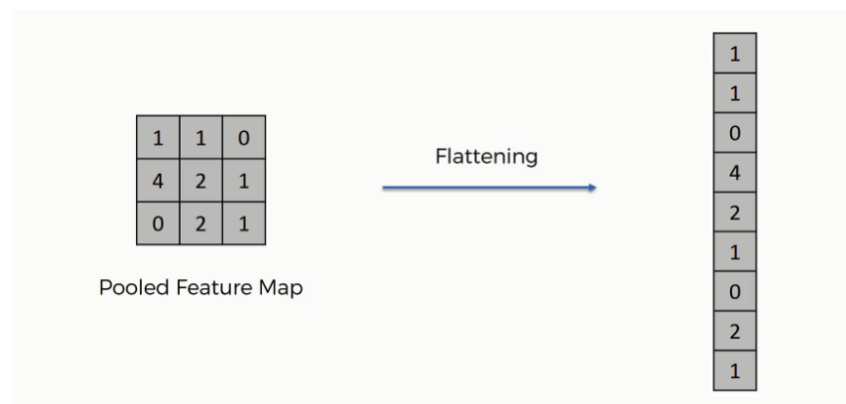


Figura 2.14: Capa *Flatten* [22]

2.3.5. Capa Fully Connected

La capa *Fully Connected* o Completamente Conectado se encarga de recoger los datos adaptados por la capa anterior de adelgazamiento y aplicar una serie de pesos para cada neurona hasta obtener la salida adecuada. Se pueden utilizar varias de estas capas dependiendo de la complejidad y profundidad a la que se quiera llegar en la red neuronal. Esta es una capa formada por una combinación de funciones afines y funciones no lineales que añaden complejidad a la red mejorando el resultado [23]. Se puede observar su estructura con el ejemplo de la Figura 2.15.

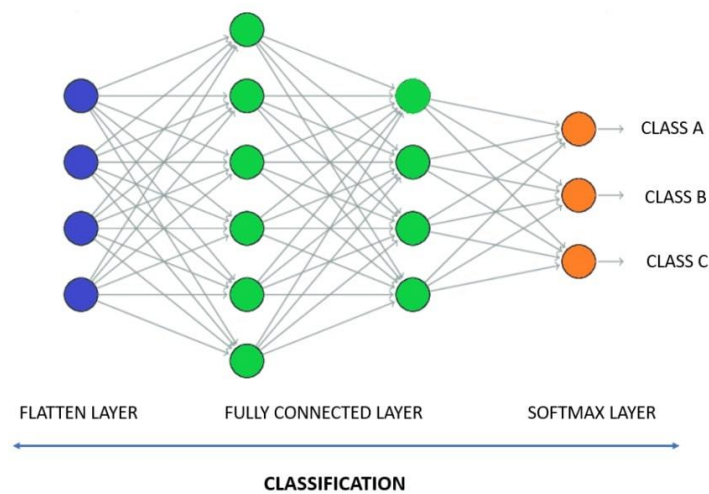


Figura 2.15: Capa *Fully Connected* [23]

2.3.6. Supresión de no máximos

A la hora de hacer las detecciones finales, es necesario filtrar de alguna forma todas las detecciones hechas para quedarse con las óptimas. Ahí entra el algoritmo de **supresión de no máximos** o *Non-maximum Suppression* (NMS).

Este algoritmo recibe como entrada una lista de propuestas de cajas delimitadoras o *bounding boxes* **B**, sus correspondientes confianzas **S** y el umbral de superposición máximo **N**. Su salida es la lista **D** con las mejores propuestas filtradas [24]. A continuación, se explicará paso por paso el algoritmo:

1. Se selecciona la caja delimitadora con mayor confianza, se elimina de **B** y se añade a la lista final.
2. Ahora se compara esta caja con todas las demás, calculando el IoU (*Intersection over Union*, explicado más adelante en el **IoU** más adelante) de esta caja con relación a cada una de las otras. Si el IoU es mayor que el umbral **N**, se elimina esa otra caja de la lista **B**.

3. De nuevo se elige la caja con mayor confianza de B, se añade a D y se elimina de B.
4. Se vuelve a calcular el IoU de esta caja con respecto al resto de cajas de B y se eliminan de B las que superen el umbral N.
5. Se repite este proceso hasta que no quedan más cajas en B

2.3.7. Fases de entrenamiento de una RNC

El rendimiento de una red neuronal dependerá en gran medida de la calidad y cantidad de datos disponibles para el entrenamiento, así como de la estructura que se ha utilizado para crear la red y su posterior optimización. Pero los pasos a seguir son los siguientes:

1. **Selección y procesamiento del conjunto de entrenamiento.** Cuanto más grande mejor. La parte más laboriosa es el etiquetado de imágenes que suele ser prácticamente a mano.
2. **Elección de la estructura de la red,** puede ser una estructura predefinida sin necesidad de crearla de cero.
3. **Elección inicial de las métricas de la red.**
4. **Evaluación del rendimiento de la red.**
5. **Reajuste de las métricas de la red.**

2.3.8. Hiperparámetros

A continuación, se explicarán un poco más en profundidad las métricas o hiperparámetros que se suelen modificar.

- **Learning rate:** Indica la velocidad de aprendizaje de nuestra red neuronal. Este es un valor comprendido entre 0 y 1, donde valores cercanos a 1 implican que la red tenderá a converger más rápido, pero generando más errores, mientras que valores cercanos a 0 implican un entrenamiento mucho más lento, pero con mayor probabilidad de generar resultados más exactos.
- **Weight decay:** Implementa en la red una forma de reducir los problemas de sobreajuste. Se encarga de decidir cuánto penaliza a los pesos de la red en la retropropagación o “*backpropagation*” evitando así en cierta medida el **sobreentrenamiento** (*overfitting*).
- **Epochs:** Representa el número de veces o **épocas** que se va a entrenar el conjunto de datos. En principio, cuantas más épocas mejores resultados se obtienen. Sin embargo, si se excede un cierto número de épocas (que varía dependiendo de cada red), la red puede llegar a memorizar las salidas generando sobreentrenamiento.
- **Batch size:** Es el tamaño de los lotes (conjunto de muestras) con los que se procesan todas las muestras en cada época. Es decir, con lotes de 5 para 100 muestras, en cada época se procesarán las muestras de 5 en 5.

- **Dropout rate:** Expresa la probabilidad que tiene cada neurona de desactivarse en cada iteración. Este parámetro, al igual que el “*Weight decay*”, ayuda a la red a evitar el sobreentrenamiento. Se puede observar gráficamente en la figura 2.16 el funcionamiento del *dropout*.

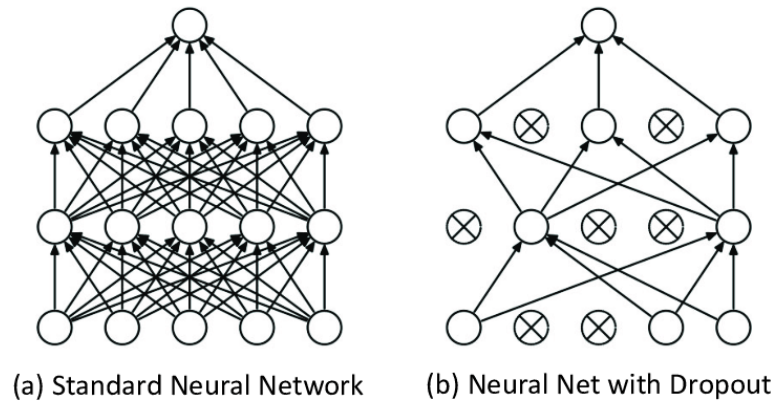


Figura 2.16: Red neuronal sin *dropout* vs con *dropout* [25]

2.3.9. Evaluación de la red

Este tipo de redes suelen evaluarse a raíz del resultado proporcionado por ciertas métricas:

- **Precisión:** Esta métrica mide la capacidad del modelo para hacer predicciones precisas y acertadas. En la ecuación 2.7 se muestra cómo se puede calcular.

$$Precision = \frac{VP}{VP+FP} \quad (2.7)$$

Donde **VP** son los verdaderos positivos (objetos detectados por la red que están realmente ahí, buena detección) y **FP** son los falsos positivos (objetos detectados por la red que no están realmente ahí, mala detección).

- **Recuperación o “Recall”:** Esta métrica mide la capacidad del modelo para detectar los objetos reales que aparecen en la imagen. En la ecuación 2.8 se muestra cómo se puede calcular.

$$Recall = \frac{VP}{VP+FN} \quad (2.8)$$

Donde **VP** son los verdaderos positivos y **FN** son los falsos negativos (objetos reales que no han sido detectados por la red, mala detección).

- **mAP o Mean Average Precision:** Esta métrica evalúa la precisión media de la detección entre todas las clases, proporcionando un valor único para comparar distintos modelos [26]. Se puede obtener calculando el área que hay bajo la **curva Precisión-Sensibilidad** o mediante la ecuación 2.9.

$$mAP = \frac{1}{n^{\circ}clases} \sum_{c \in clases} \frac{|VP_c|}{|FP_c| + |VP_c|} \quad (2.9)$$

Donde VP_c son los verdaderos positivos de cada clase y FP_c son los falsos positivos de cada clase.

- **IoU o Intersection over Union:** Esta métrica mide la cualidad de las *bounding boxes* o **cajas delimitadoras** comparando la intersección y la unión de áreas del objeto predicho y el objeto real. En la figura 2.17 se muestra un ejemplo junto a cómo se calcula este parámetro donde el objeto real sería el “*ground truth*” y el objeto predicho sería “*prediction*”.

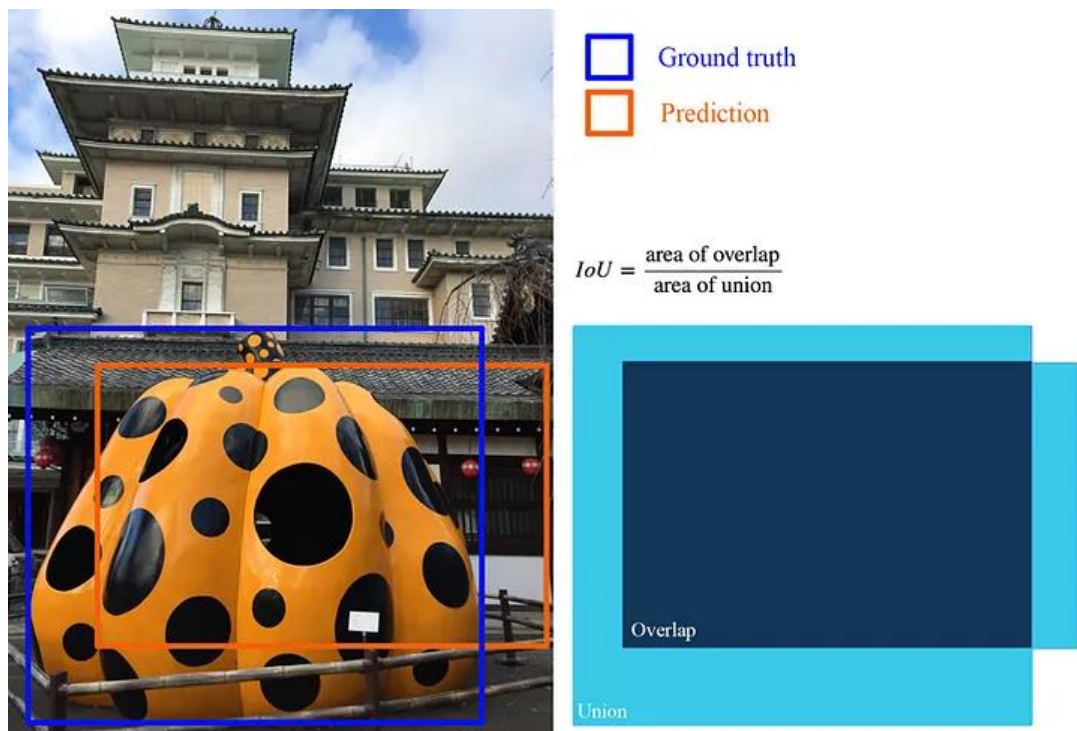


Figura 2.17: Definición de IoU [26]

2.3.10. Análisis de errores

Un factor primordial en el entrenamiento de redes neuronales es el análisis de los resultados para la detección de errores propios del entrenamiento. Existen dos tipos de errores principales que se pueden observar en los resultados:

- **Subajuste o “Underfitting”:** Este problema se puede generar cuando:
 - Un modelo de red neuronal es demasiado simple para obtener características complejas de los datos.
 - El conjunto de datos de entrenamiento no es suficiente.
 - Se hace un uso excesivo de métodos para prevenir el sobreentrenamiento, lo cual reduce demasiado las muestras de entrenamiento o la complejidad de la red.

En rasgos generales, este problema indica que la red no ha sido capaz de detectar características significativas, generando unos resultados finales muy inexactos. Para arreglar este problema se pueden realizar las siguientes modificaciones:

- Incrementar la complejidad del modelo de la red neuronal.
 - Eliminar el ruido del conjunto de datos de entrenamiento.
 - Reducir el uso de técnicas anti sobreentrenamiento.
 - Incrementar el número de épocas o la duración del entrenamiento para obtener mejores resultados [27].
- **Sobreentrenamiento, sobreajuste o Overfitting:** Este problema se da cuando la red neuronal memoriza los datos de entrenamiento (proporcionando muy buenos resultados en el entrenamiento) y deja de extraer características, generando así un mal resultado en el conjunto de test. Este fallo puede ser causado por:
 - Utilización de un modelo de red neuronal demasiado complejo.
 - Conjunto de datos de entrenamiento muy reducido.
 - Uso de un excesivo del número de épocas o tiempo de entrenamiento demasiado largo.

Existen diversos métodos muy utilizados para prevenir el sobreentrenamiento, algunos de ellos, ya mencionados anteriormente. Estos métodos son:

- Incremento de los datos de entrenamiento.
 - Reducción de la complejidad del modelo.
 - **Parada temprana** o *Early stopping*. Consiste en observar los resultados generados durante la fase de entrenamiento, y parar el entrenamiento en cuanto las pérdidas de la red empiezan a aumentar [27].
 - Modificación de los hiperparámetros mencionados anteriormente, el *weight decay* y el *dropout*.

Para entender mejor estos problemas se puede observar como en la figura 2.18, se muestra gráficamente el **subajuste**, **sobreajuste** y un **ajuste apropiado** para la clasificación de dos tipos de datos en una muestra.

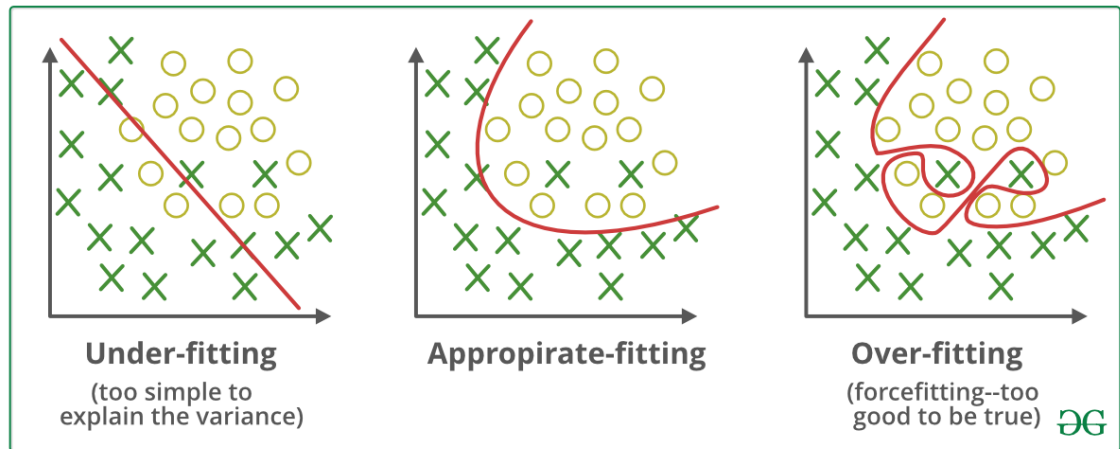


Figura 2.18: Subajuste, ajuste apropiado y sobreajuste [27]

2.4. YOLOv8

“*You Only Look Once*” o más conocido como **YOLO**, es un sistema de código abierto enfocado en la detección de objetos en tiempo real. Este algoritmo está basado en el uso de **Redes Neuronales Convolucionales**, las cuales como se ha explicado anteriormente, son idóneas para la detección de objetos en imágenes. Esto añadido a su gran rendimiento y su fácil utilización hacen de la red YOLO la primera opción a la hora de buscar la detección de objetos en imágenes en tiempo real [28].

Además, el algoritmo **YOLO** se ha ido desarrollando mucho a lo largo de los años con sus distintas versiones, todas ellas de código abierto. En el momento de la creación de este proyecto se compararon varias redes y la que se eligió por su mejor rendimiento y fácil utilización fue la red **YOLOv8** llevada a cabo por la compañía **ULTRALYTICS**. La versión utilizada en el proyecto anterior del compañero Adrián Bañuls Arias fue **YOLOv3**, la cual fue creada por la misma compañía, pero esta actualización conlleva una mejora significativa.

2.4.1. Arquitectura de la red

La anatomía de un modelo de detección de objetos en imágenes consta de tres partes [Véase la figura 2.19]:

- **Backbone** o **Columna vertebral**: se encarga de extraer características de la imagen.
- **Head** o **Cabeza**: se encarga de generar las predicciones finales.

- **Neck** o **Cuello**: componente intermedio que trata de conectar la columna vertebral a la cabeza [29].

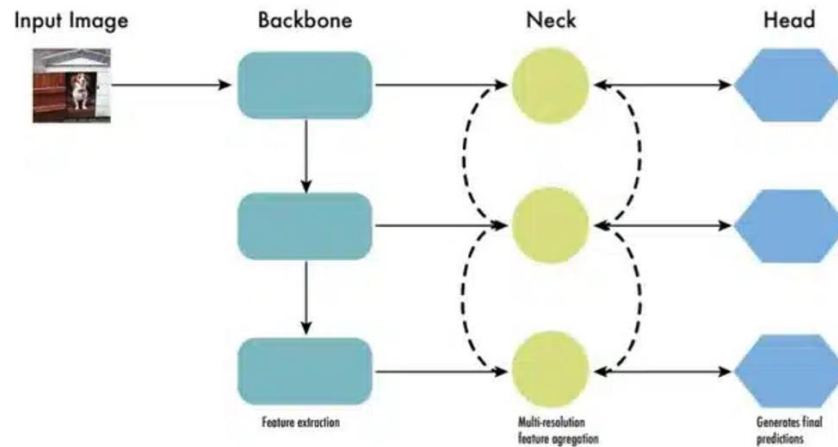


Figura 2.19: Arquitectura de un detector de objetos moderno [29]

En el caso de YOLOv8 la arquitectura de la red es bastante compleja en cada parte. Para una mejor comprensión, se muestra más desarrollada en la figura 2.20.

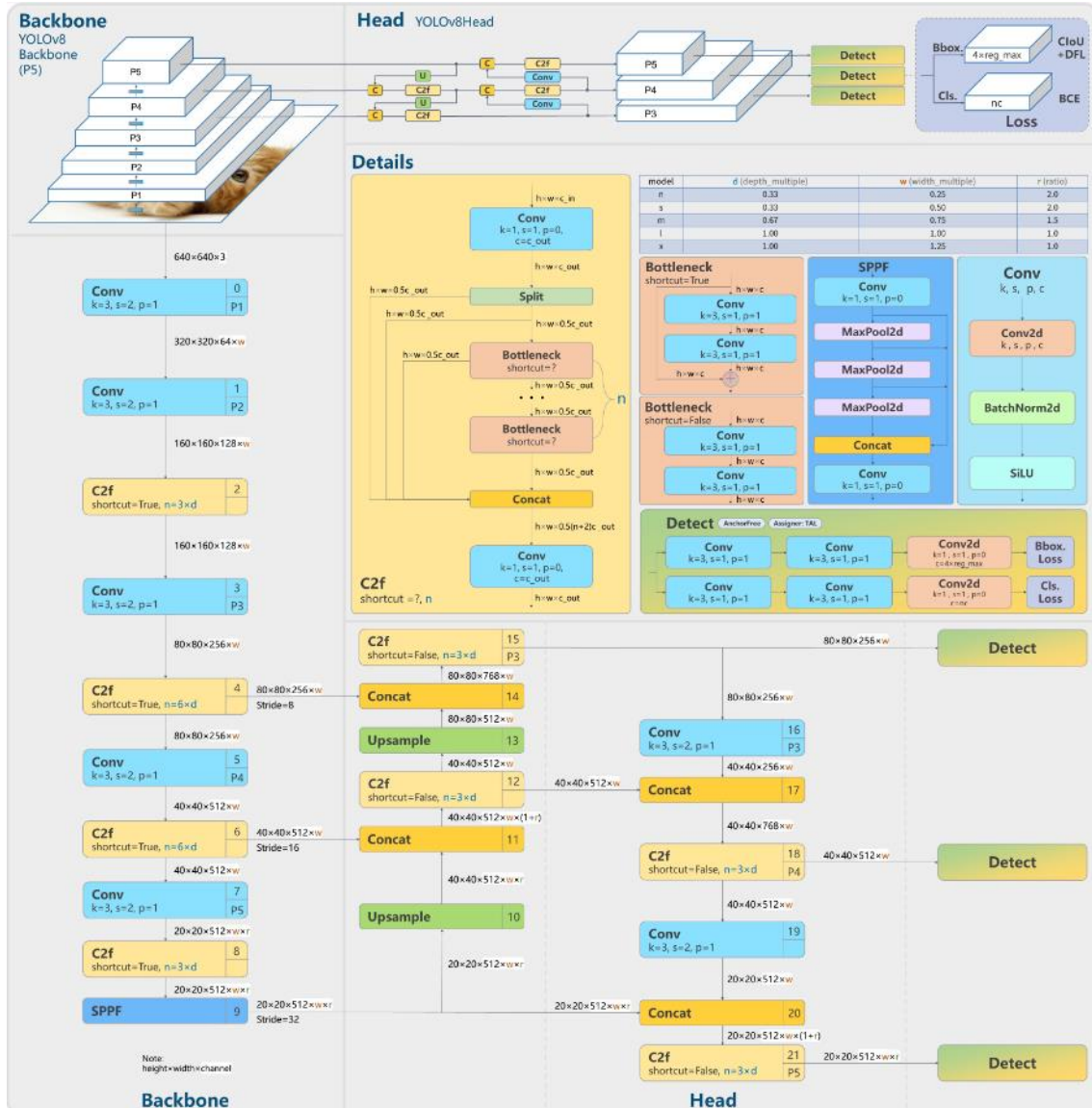


Figura 2.20: Arquitectura de la red YOLOv8 [30]

2.4.2. Algoritmo YOLO

El método de detección de objetos de la red inicial de YOLO detecta todas las cajas delimitadoras en un paso, dividiendo la imagen de entrada en una cuadrícula de tamaño $S \times S$ y prediciendo B cajas delimitadoras con valores de confianza para C clases por elemento encuadrado.

Cada caja delimitadora viene descrita mediante las coordenadas de su centro, su altura y su amplitud respecto a la imagen completa. La salida de la red es un tensor de tamaño $S \times S \times (B \times 5 + C)$, la cual puede estar seguida de un algoritmo de supresión de no máximos para eliminar detecciones duplicadas [28]. En la figura 2.21 se puede observar este proceso gráficamente.

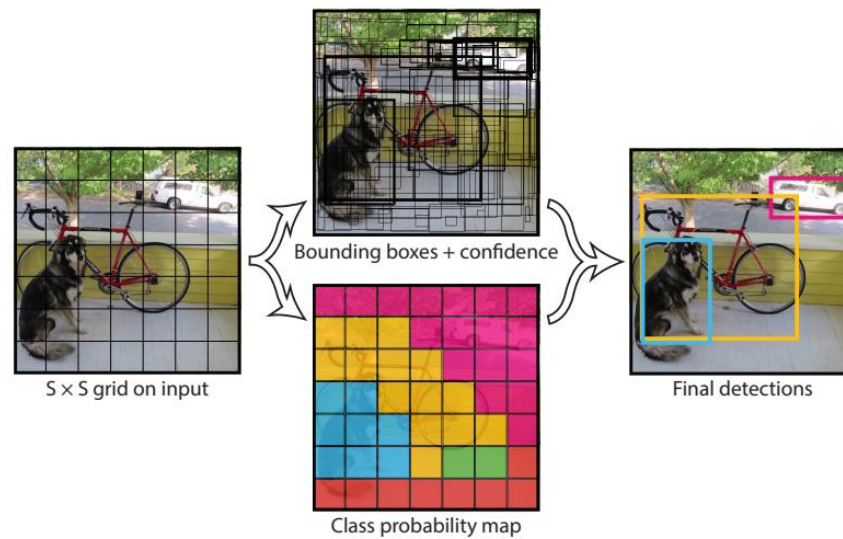


Figura 2.21: Modelo de detección de YOLO [28]

2.4.3. Novedades YOLOv8

En concreto, la versión YOLOv8 proporciona ciertas ventajas respecto a sus versiones anteriores. A pesar de que todavía no hay publicaciones técnicas explicando la arquitectura de la misma, su pueden encontrar estas ventajas investigando su interior.

- Esta red es **anchor-free**, es decir, ya no genera una serie de cajas delimitadoras predefinidas por toda la imagen para después elegir los más convenientes, sino que genera una cantidad mucho menor de los mismos, provocando una respuesta más rápida en el proceso de **Supresión de No Máximos** o *Non-maximum Suppression*.
- Esta red usa un método de aumento de datos predefinido llamado *Mosaic augmentation* o **Aumento en Mosaico**, que funciona generando imágenes compuestas en el entrenamiento de forma que una imagen de entrenamiento puede contener varias imágenes distintas en la misma imagen. Se puede entender mejor observando la figura 2.22. Sin embargo, esta técnica se deshabilita en las últimas 10 épocas porque puede ser dañina si se usa durante todo el proceso de entrenamiento [30].

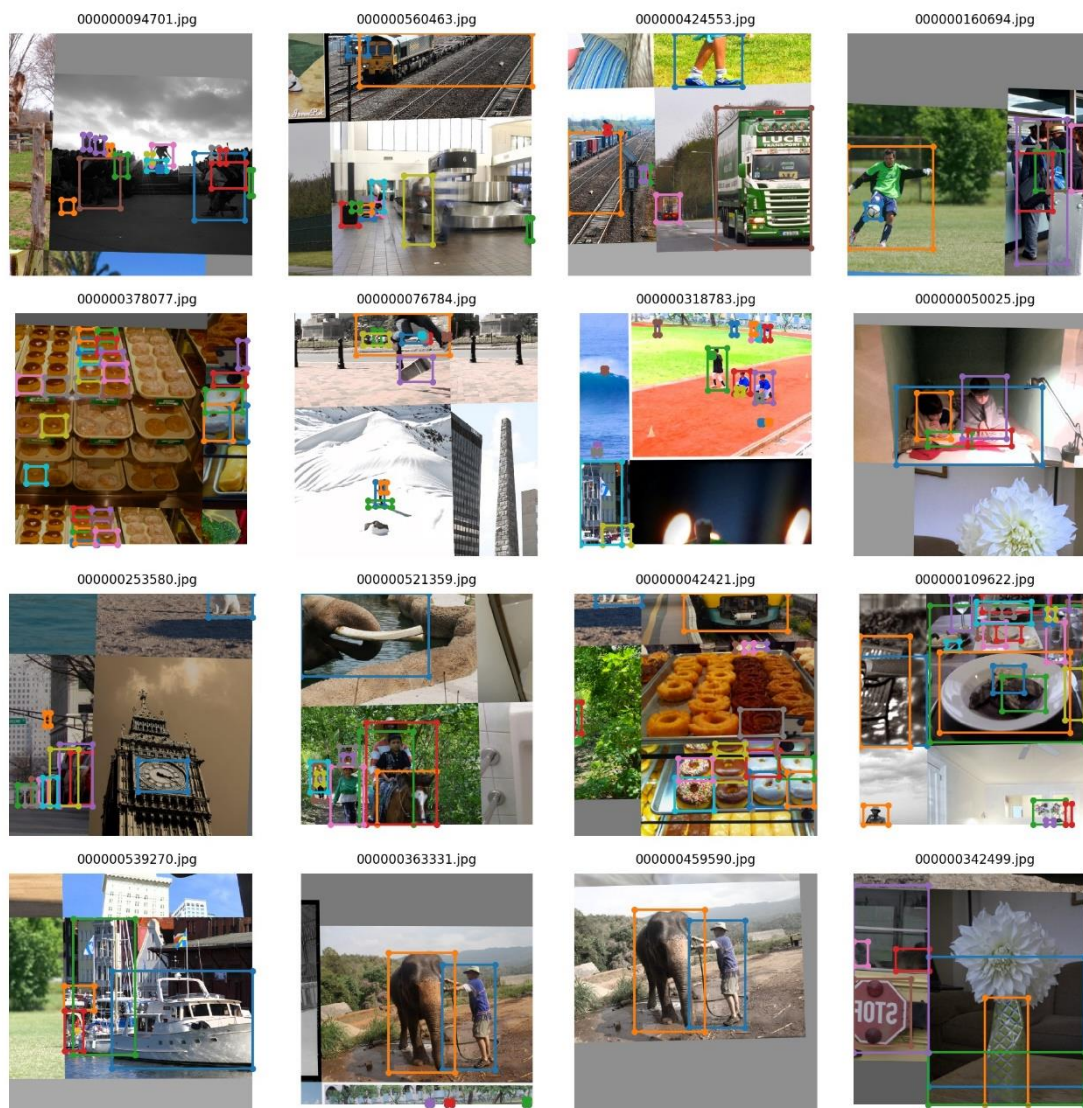


Figura 2.22: Aumento en mosaico [30]

2.4.4. Modelos y tamaños de YOLOv8

YOLOv8 consta de distintos modelos, destinados al objetivo de cada usuario. Tienen un modelo la detección básica llamado **YOLOv8** que es el que se usará en este proyecto, otro para la segmentación llamado **YOLOv8-seg**, otro para la detección de poses y puntos clave llamado **YOLOv8-pose**, y un último específico para la clasificación de objetos llamado **YOLOv8-cls**. A su vez, cada modelo tiene distintos tamaños para la red dependiendo de las especificaciones y rapidez demandados por cada usuario. En concreto, los tamaños disponibles son: YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l y YOLOv8x. Haciendo referencia respectivamente a muy pequeño (*nano*), pequeño (*small*), mediano (*medium*), grande (*large*) y extra-grande (*xl*).

Para elegir el tamaño se tiende a buscar el punto medio entre rapidez y precisión, ya que para tamaños más pequeños la red será más rápida pero menos precisa mientras que para tamaños más grandes la red será cada vez más lenta pero más exacta. En la figura 2.23 se observa la comparación de los distintos tamaños de YOLOv8 entre sí y entre otras versiones de YOLO. Además, en la tabla 2.1 se pueden observar las diferencias específicas entre cada tamaño.

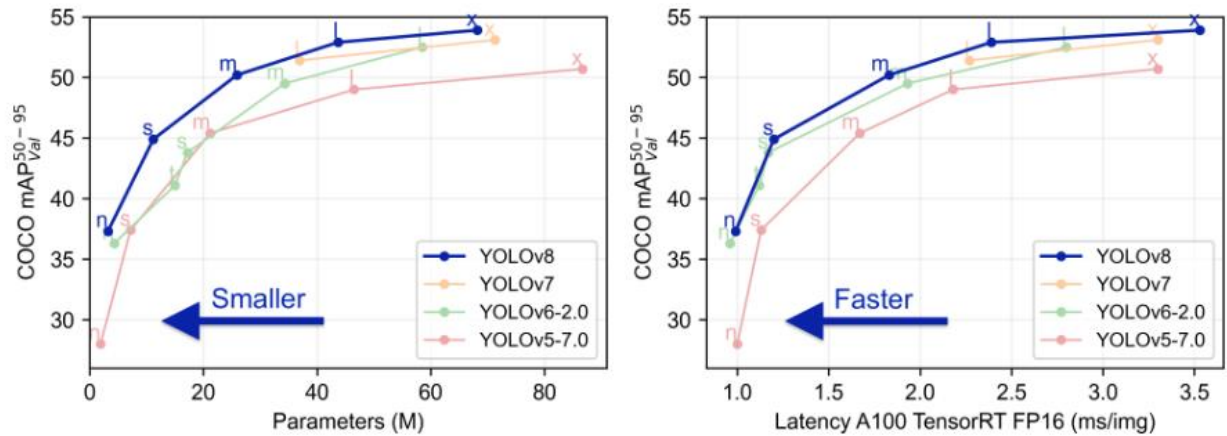


Figura 2.23: Rendimiento modelos YOLOv8 [31]

Model	Size (pixels)	mAP^{val}_{50-95}	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	Params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	1652
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Tabla 2.1: Especificaciones de los tamaños de YOLOv8 [31]

Cabe destacar que todos estos modelos han sido preentrenados con el dataset de **COCO** y que podemos acceder a todos ellos a través de su repositorio de Github [32]. Además, en este proyecto se quiere diseñar un sistema de **fusión de redes neuronales en tiempo real para un UGV** (*Unmanned Ground Vehicle* o Vehículo Terrestre no tripulado), por lo que la **rapidez** de la red es muy importante, pero tampoco se puede descuidar su **rendimiento** que va ligado en parte a la cantidad de parámetros utilizados en la red.

Tras comparar los distintos tamaños de la red con la tabla 2.1 y la figura 2.23, se optó por utilizar **YOLOv8m** ya que la versión superior YOLOv8l no genera una mejora muy significativa respecto a esta y produce una disminución de la rapidez del modelo del 60%, mientras que el salto de precisión de YOLOv8s a YOLOv8m sí es mayor y compensa su aumento de tiempo de procesamiento.

2.5. Comunicación por ROS

Como se explicó en el apartado 1.2, **ROS** son las siglas de *Robot Operating System*, un sistema de comunicación *open source* o de *código abierto*, enfocado a la robótica que funciona mediante publicación y subscripción de *topics*. Los **topics** son como cajas estáticas que se crean cuando se envían mensajes (publicación de topics) y esperan a que otros se suscriban para *escuchar* estos mensajes (subscripción de topics). Sin embargo, esta forma de comunicación **no es Peer to Peer**, es decir, no es una comunicación directa entre dos entidades, sino que conforme los mensajes se van publicando, estos pueden ser escuchados o no por quien se suscriba. El protocolo de comunicación que utiliza se llama **TCPROS**, que es un protocolo customizado basado en TCP/IP [3].

Por otro lado, la mayor parte del código de comunicación de este proyecto ha sido creado con la segunda versión de ROS, es decir, **ROS 2**. Esta nueva versión de ROS cambia multitud de factores respecto a su versión inicial. Por ejemplo, el método de comunicación que se usa para la publicación y subscripción de mensajes pasa a ser la utilización de **DDS**. Han decidido utilizar este middleware para partir de él e ir mejorándolo hasta adaptarse a sus necesidades [33].

Una ventaja de esta nueva versión es la eliminación del Master propia de ROS1. Para poder comunicar dispositivos en ROS1 era necesario mantener un nodo Master activo (también llamado roscore) que controlaba todos los envíos y recepciones de mensajes. En ROS2 esto no es necesario, lo cual mejora la escalabilidad y la robustez del sistema, permitiendo la descentralización de las comunicaciones. Esto permite, por ejemplo, que un dispositivo pueda escuchar un mensaje de otro dispositivo y que a su vez pueda enviar otro mensaje a otro dispositivo externo.

En este proyecto se utilizará ROS2 para la comunicación y recepción de datos, es decir, una vez las redes RGB y térmica estén entrenadas, mediante unos topics de ROS2 se recibirán las imágenes, se procesarán las redes junto al algoritmo de fusión, y se publicará la anotación de la detección fusionada final en otro topic de ROS2 para que se pueda leer desde otro dispositivo.

Capítulo 3

3. Estudios Previos y Entorno de Trabajo

3.1. Proyecto de partida

Este proyecto continúa otro proyecto previo de Adrián Bañuls Arias. En él, mi compañero etiquetó y entrenó dos redes neuronales, una térmica y otra en el espectro visible, con una versión previa de YOLO, YOLOv5. A raíz de aquí, se ha intentado mejorar su entrenamiento usando sus imágenes etiquetadas con la red YOLOv8, para después desarrollar un algoritmo que fusione ambas redes y obtenga una detección mejorada. Finalmente, se ha creado un *topic* de ROS2 para enviar mensajes con la información del resultado de la fusión en cada instante de tiempo.

3.2. YOLOv8

Antes de poder empezar a entrenar las redes, se ha tenido que estudiar el funcionamiento de la red que se va a usar, es decir, de YOLOv8. A pesar de que esta red está diseñada de tal forma que es fácil de usar a partir de su repositorio de Github [32], este ámbito del Aprendizaje Profundo y la programación en general requiere unos conocimientos iniciales considerables para realizar la configuración.

A continuación, se explicarán las dependencias necesarias para ejecutar el entrenamiento de la red junto a una breve explicación del uso de Docker que haremos.

3.2.1. Docker

Para poder entrenar las redes neuronales en un tiempo razonable, se precisa de una **GPU** muy potente. Por lo que se usará la **estación DGX de Nvidia** a la que tiene acceso la universidad. Este ordenador posee 4 GPUs de las más potentes del mercado, enfocadas específicamente para este tipo de entrenamientos. Sin embargo, este ordenador se usa de forma compartida, por lo que siempre que se utilice, se deberán ejecutar todos los procesos en un contenedor de Docker, evitando así también la posible introducción de *malware* o programas maliciosos en este ordenador.

Docker [34] es una pseudo máquina virtual donde puedes ejecutar cualquier programa o sistema operativo sin influir en tu propio sistema operativo y bajo un coste de memoria muy reducido. En el **Anexo A** se pueden observar los códigos y pasos utilizados para ejecutar el contenedor de Docker y todo lo relacionado con él.

3.2.2. Pytorch

Pytorch es una librería de aprendizaje automático de código abierto que fue desarrollada por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (FAIR). Está programada en Python, C++ y CUDA y su meta principal es mejorar la eficiencia de la GPU en ámbitos del aprendizaje automático sin dejar de lado la flexibilidad y velocidad deseada por el usuario.

3.2.3. CUDA

CUDA son las siglas de Compute Unified Device Architecture y es un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten al usuario crear algoritmos específicos para las GPUs de esta marca aumentando la eficiencia de las mismas. CUDA es primordial a la hora de entrenar redes neuronales porque la gran mayoría de las librerías creadas para estas la han usado como base.

3.2.4. ULTRALYTICS

Ultralytics es la compañía que ha creado la red YOLOv8. Para ejecutar esta red es necesario acceder su repositorio de Github [32] donde se encuentran los pasos a seguir para utilizar la red, así como las configuraciones de la misma. Otra opción, sería descargarse directamente la librería ultralytics, sin embargo, descargándose el repositorio es más fácil de configurar y es lo que se ha hecho en un principio para el entrenamiento.

3.2.5. OpenCV

OpenCV es una librería enfocada en el procesamiento de imágenes y en la visión por computador. Esta librería permite un amplio rango de modificaciones para imágenes lo cual es primordial a la hora de pasar por muchas capas de la red, así como para realizar data augmentation o aumento de datos.

3.2.6. Matplotlib

Matplotlib es una librería enfocada en la generación de gráficos en dos dimensiones. Esta librería no es directamente requerida por la red, pero sí es necesaria para mostrar los resultados gráficamente mediante ClearML.

3.2.7. Thop

Thop, también conocida como PyTorch-OpCounter, permite conocer la cantidad de operaciones en coma flotante (FLOPS) ejecutadas a lo largo del entrenamiento.

3.2.8. Pandas

Pandas es una librería de Python enfocada a la manipulación y al análisis de datos. Es muy utilizada en el ámbito de la Inteligencia Artificial debido a su rapidez, potencia, flexibilidad y fácil uso.

3.2.9. Seaborn

Seaborn es otra librería usada para la visualización de datos basada en Matplotlib que ofrece una interfaz de alto-nivel para dibujar atractivos e informativos gráficos estadísticos. Al igual que Matplotlib, esta librería no es directamente necesaria para el entrenamiento, sino para la monitorización de los resultados con ClearML.

3.2.10. ClearML

ClearML es un programa utilizado para monitorizar los resultados del entrenamiento de redes neuronales. Antes de empezar a entrenar las redes se introducen tus credenciales y puedes observar el avance del entrenamiento desde la web o esperar a que acabe para observar los gráficos creados. Además, te permite comparar distintos entrenamientos para elegir el mejor.



UNIVERSIDAD
DE MÁLAGA



Capítulo 4

4. Fases de Entrenamiento

4.1. Introducción

Como se ha comentado en capítulos anteriores, las imágenes y anotaciones usadas para estas redes venían ya hechas, por lo que no fue necesario rehacerlas manualmente de nuevo. Estas etiquetas identificaban 4 tipos de clase distintas, pero se pueden agrupar también como dos clases raíces, vehículos y personas.

- Emergency vehicle o Vehículo de emergencia
- Non-emergency vehicle o Vehículo civil
- First responder o Personal de emergencias
- Non-first responder o Persona civil

La estructura de las anotaciones que admite esta red, es decir, la manera en que están identificados los objetos dentro de los archivos de texto asociados a cada imagen, es el estándar de YOLO. Gracias a que los datos estaban ya en este estándar no se precisó de un código de reconversión de los datos, sin embargo, en la mayoría de proyectos como este, sí hay que ajustarlos.

En este proyecto se usó el método *fine tuning* que consiste en entrenar una red sobre otra red preentrenada. Se usaron los pesos preentrenados con el conjunto de datos de COCO con la red de YOLOv8 y con el tamaño mediano, obteniendo muy buenos resultados.

Una vez se tienen controlados todos los datos, se introducen en la red, se configura el código de ejecución y comienza el entrenamiento.

4.2. Red RGB o del espectro visible

El etiquetado de imágenes suele ser la parte más larga y tediosa del entrenamiento. Sin embargo, para este proyecto se usaron imágenes ya etiquetadas, lo cual agilizó bastante el proceso. En concreto, el conjunto de datos que se usó para esta red constó de 847 imágenes para el **conjunto de entrenamiento** (imágenes que se usaron para entrenar la red) junto a 149 imágenes para el **conjunto de test** o prueba (imágenes que se usaron para comprobar el buen rendimiento de la red tras el entrenamiento inicial). A pesar de que no son muchas imágenes, (para un entrenamiento desde cero se recomienda disponer del orden de 10000 imágenes por clase), al tratarse de un reentrenamiento se han conseguido buenos resultados ya que este requiere menos imágenes para funcionar correctamente.

El primer entrenamiento RGB se hizo con las métricas predeterminadas para comprobar los resultados iniciales y el buen funcionamiento de la red. Sin embargo, el primer resultado superó cualquier expectativa. A pesar de haber usado un conjunto de datos pequeño, se obtuvo un **mAP 50** de **91.77%** de media entre las 4 clases utilizando **80 epochs** o iteraciones. Todos los datos obtenidos del entrenamiento se obtuvieron gracias a la aplicación ClearML que permitía monitorearlos y visualizarlos. En la Tabla 4.1 se pueden observar las métricas utilizadas para el entrenamiento de la red RGB y para el de la red térmica.

Hiperparámetros	Entrenamiento red RGB	Entrenamiento red Térmica
Batch_size	16	16
Dropout	0	0
Epochs	80	80
Weight_decay	0.0005	0.0005
Learning_rate	0.01	0.01
Shear	0	0
Translate	0.1	0.1
Scale	0.5	0.5
Perspective	0	0
Flipud	0	0
Fliplr	0.5	0.5
Mosaic	1	1
Mixup	0	0

Tabla 4.1: Hiperparámetros utilizados por las redes RGB y térmica

Los 5 primeros hiperparámetros de la Tabla 4.1 vienen explicados en el apartado 2.2.8. y corresponden a hiperparámetros básicos de las redes neuronales. Mientras que el resto, son hiperparámetros relacionados con el aumento de datos, los cuales se explican más adelante en el apartado 4.4.

En la figura 4.1 se puede observar la matriz de confusión generada tras el entrenamiento de la red RGB, la cual muestra gráficamente la precisión de la red para cada clase a lo largo del entrenamiento. La matriz de confusión es una matriz cuya información principal se muestra en su diagonal. Allí, la matriz de confusión muestra el porcentaje de acierto que ha tenido la red sobre cada clase a lo largo del entrenamiento. El resto de los valores de la matriz representan el porcentaje de confusiones (detectar alguna clase como otra que no corresponde) que ha tenido la red con cada clase.

Se considera que se ha obtenido un buen resultado cuando la media de los valores de la diagonal es alta (relativamente cercana a 1), siempre y cuando no haya sobreentrenamiento.

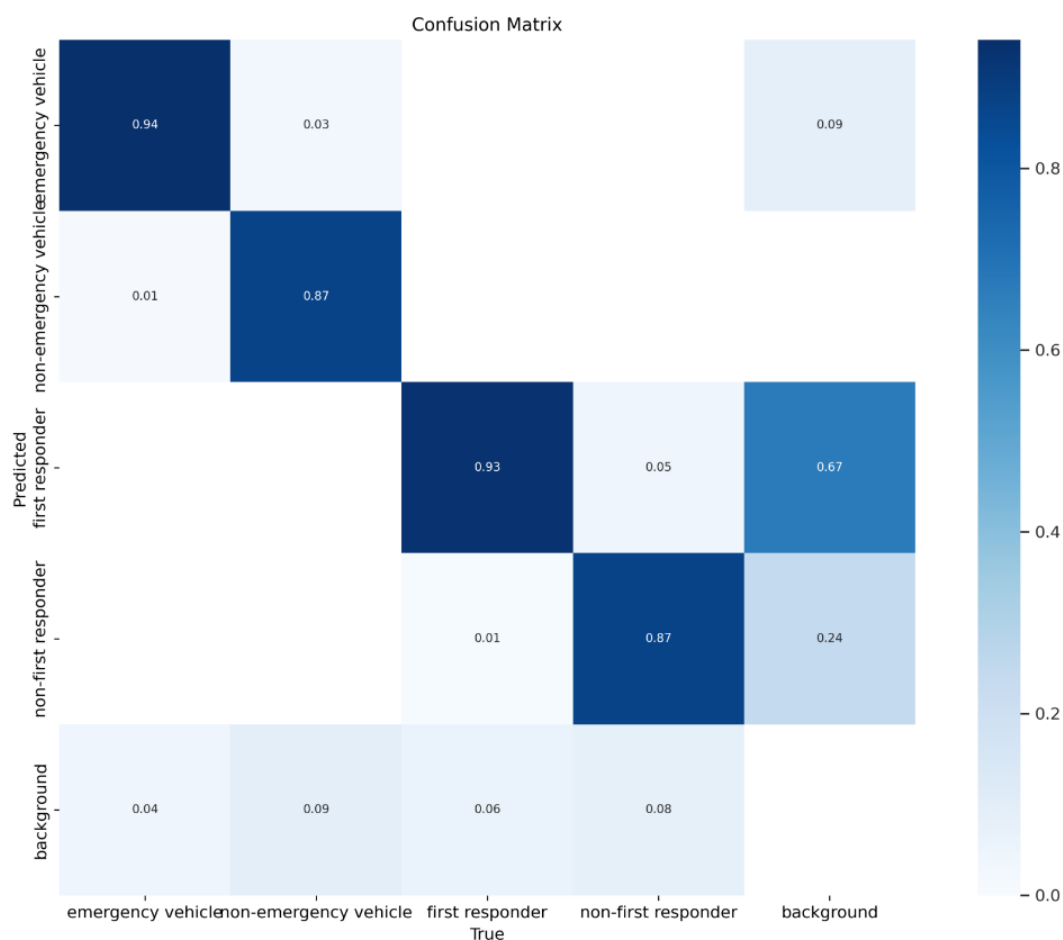


Figura 4.1: Matriz de confusión de la red RGB

Cabe destacar, que las clases con peor rendimiento son aquellas que contaban con menos instancias, lo cual hace destacar incluso más su precisión media del 91.77%. Donde se observa el mayor defecto de la red es a la hora de confundir los falsos positivos, ya que a veces tiende a confundir personas civiles y de rescate con el entorno. En la Tabla 4.2 se pueden observar los resultados de la matriz de confusión de forma más clara.

Clases	Precisión (%)
Vehículo de emergencia	94
Vehículo de civil	87
Personal de rescate	93
Persona civil	87
Media total	91.77

Tabla 4.2: Resultados del entrenamiento de la red RGB

En la figura 4.2, se pueden observar las pocas instancias por clase que hay en este conjunto de imágenes, sobre todo para las clases de vehículos y personal civil que aparecen relativamente poco.

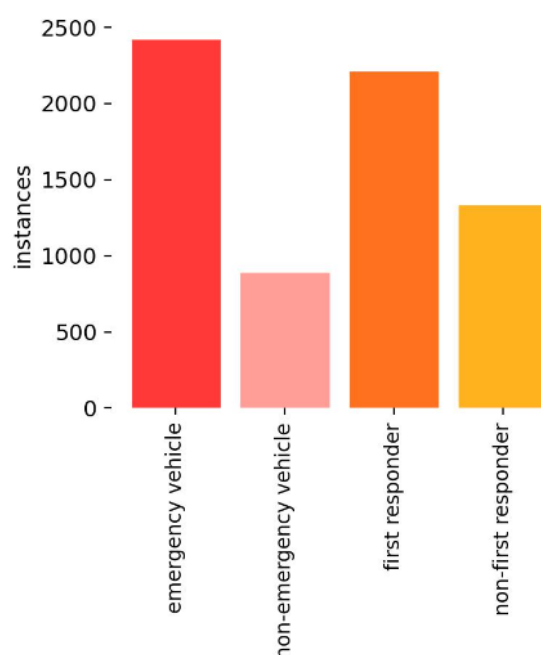


Figura 4.2: Instancias de cada clase en la red RGB

Por otro lado, en la figura 4.3 se puede apreciar el gran rendimiento de la red mediante la curva *Precision-Recall* o Precisión-Sensibilidad. Cuanto mayor es el área bajo la curva, mejor es el resultado (tal y como aparece ahí, la media del [mAP@0.5](#) es de 0.917, bastante buen resultado).

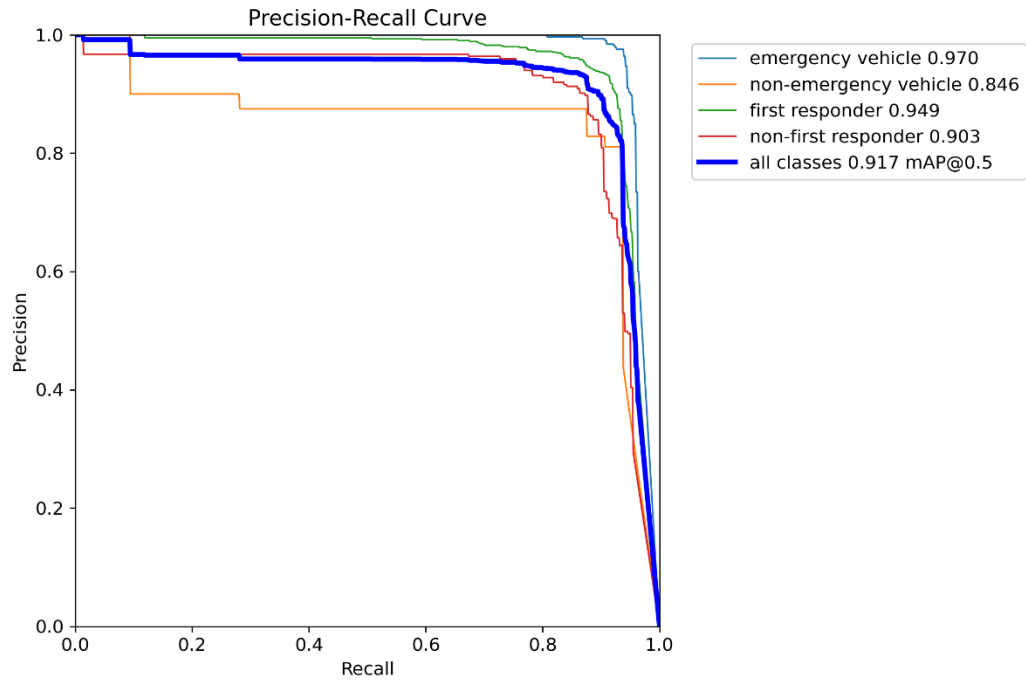


Figura 4.3: Curva Precisión-Sensibilidad de la red RGB

Por último, la figura 4.4 muestra un resumen de gráficas de las métricas y pérdidas del entrenamiento y la validación. Se observa cómo las pérdidas van disminuyendo a lo largo de la validación sin llegar a repuntar en ninguna (cambio de tendencia), confirmando así que no ha habido **sobreentrenamiento** o *overfitting*.

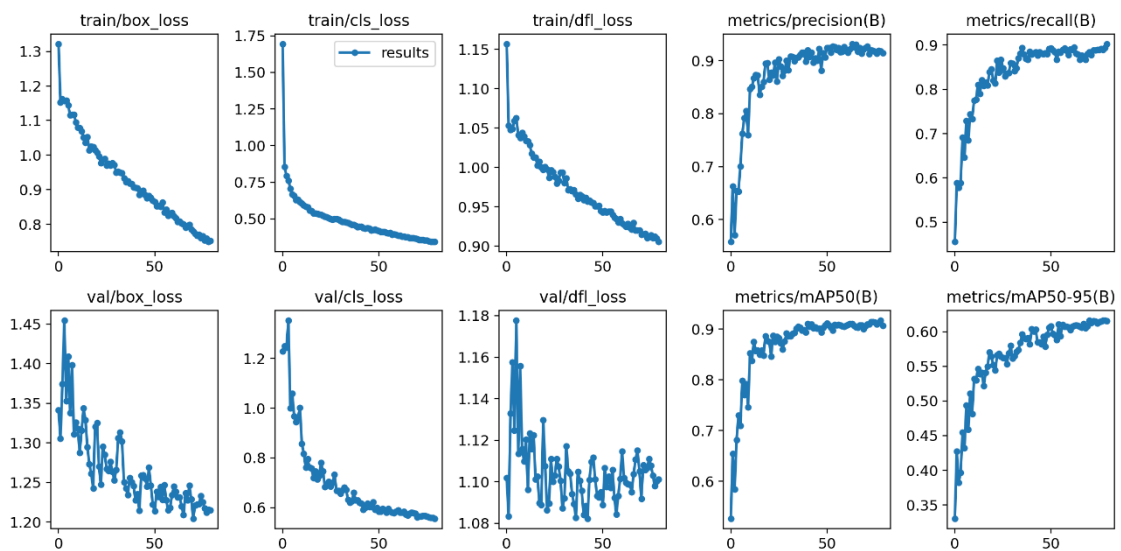


Figura 4.4: Métricas del entrenamiento de la red RGB

En las figuras 4.5 y 4.6 se puede observar el desempeño de la red en dos situaciones distintas.



Figura 4.5: Detección de personal de rescate junto a un vehículo de rescate con la red RGB



Figura 4.6: Detección de distinto personal de rescate con la red RGB

Ambas detecciones son bastante buenas, sin embargo, en la primera se observa cómo se duplica la detección del coche.

4.3. Red térmica

El conjunto de datos que se usó para esta red constó de 1098 imágenes para el **conjunto de entrenamiento** junto a 194 imágenes para el **conjunto de test** o prueba. A pesar de que no son muchas imágenes, (para un entrenamiento desde cero se recomienda disponer del orden de 10000 imágenes por clase), al tratarse de un reentrenamiento de se han conseguido buenos resultados también, ya que este requiere menos imágenes para funcionar correctamente.

El primer entrenamiento térmico se hizo con las métricas predeterminadas para comprobar los resultados iniciales y el buen funcionamiento de la red. Sin embargo, al igual que con la red RGB, se obtuvieron unos resultados muy buenos con un **mAP 50 de 90.11%** de media entre las 4 clases utilizando **80 epochs** o iteraciones. Estos resultados, aun siendo muy buenos son ligeramente inferiores a los 91.77% de la red RGB, y eso es debido a que la cantidad de información que ofrecen las imágenes térmicas es mucho menor en comparación a la ofrecida por una imagen rgb. Por lo tanto, para la red es más complicado extraer información relevante de la misma. En la Tabla 4.1 mostrada anteriormente, aparecen las métricas utilizadas en el entrenamiento de la red térmica junto a las de la red RGB.

En la figura 4.7 se observa la matriz de confusión generada tras el entrenamiento de la red térmica, la cual muestra gráficamente la precisión de la red para cada clase a lo largo del entrenamiento.

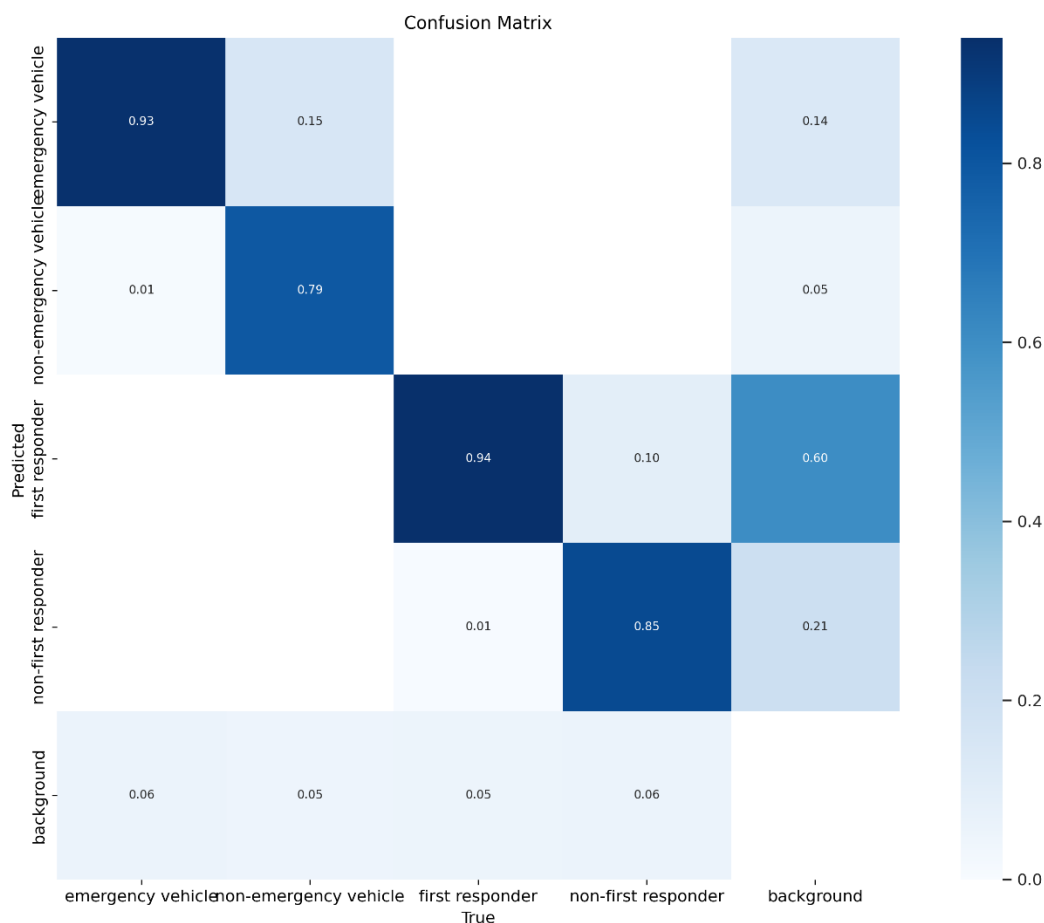


Figura 4.7: Matriz de confusión de la red térmica

Como pasaba con la otra red, las clases con peor rendimiento son aquellas que contaban con menos instancias, haciendo destacar incluso más su precisión media del 90.11%. Sin embargo, como se ha mencionado previamente, el rendimiento de esta red para cada clase es ligeramente menor a la RGB. En la Tabla 4.3 se pueden observar los resultados de la matriz de confusión de forma más clara.

Clases	Precisión (%)
Vehículo de emergencia	93
Vehículo de civil	79
Personal de rescate	94
Persona civil	85
Media total	90.11

Tabla 4.3: Resultados del entrenamiento de la red térmica

En la figura 4.8, se pueden observar las pocas instancias por clase que hay en este conjunto de imágenes, sobre todo para las clases de vehículos y personal civil que aparecen relativamente poco.

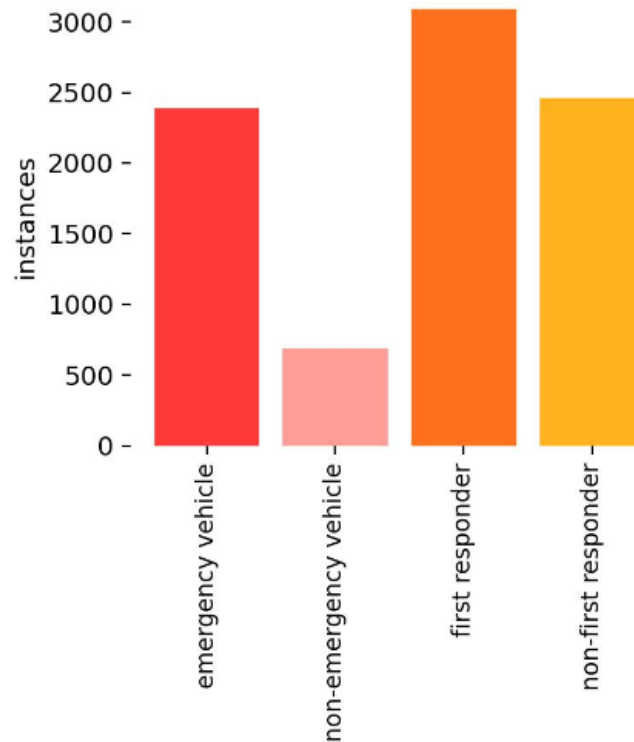


Figura 4.8: Instancias de cada clase en la red térmica

En la figura 4.9 también se puede apreciar el gran rendimiento de la red mediante la curva Precisión-Sensibilidad. Cuanto mayor es el área bajo la curva, mejor es el resultado (tal y como aparece ahí, la media del [mAP@0.5](#) es de 0.901, bastante buen resultado).

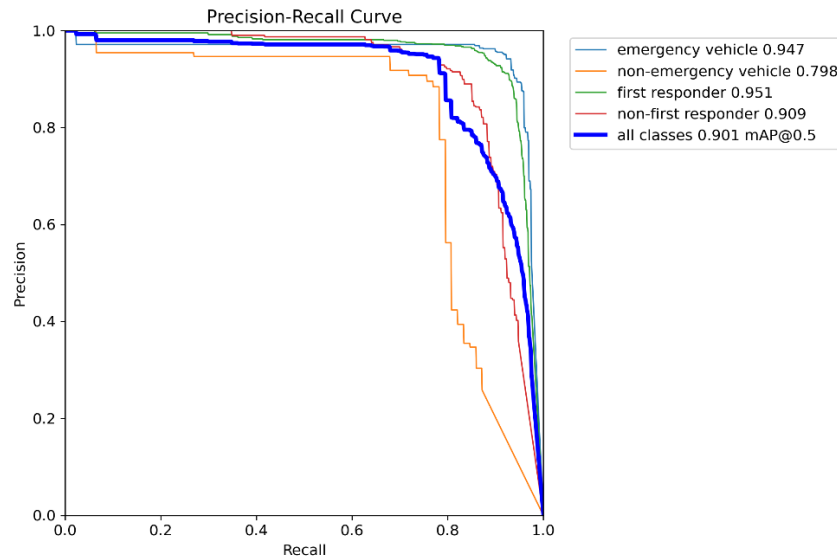


Figura 4.9: Curva Precisión-Sensibilidad de la red térmica

Por último, la figura 4.10 muestra un resumen de gráficas de las métricas y pérdidas del entrenamiento y la validación. Se observa cómo las pérdidas van disminuyendo a lo largo de la validación sin llegar a generar cambio de tendencia claro, solo algunos repuntes excepcionales que no cambian la tendencia, confirmando así que tampoco ha habido **sobreentrenamiento** u *overfitting*.

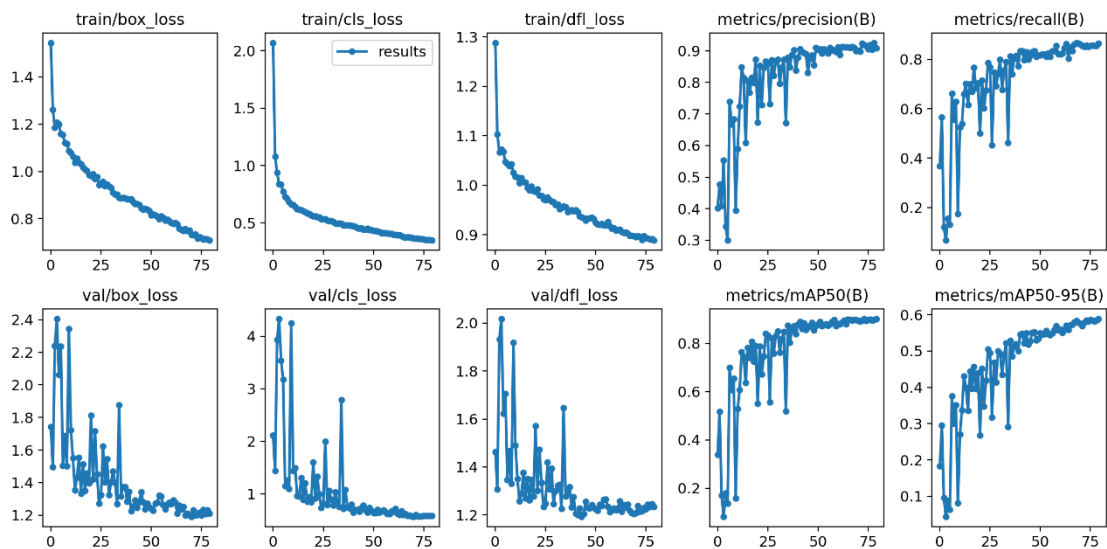


Figura 4.10: Métricas del entrenamiento de la red térmica

En las figuras 4.11 y 4.12 se puede observar el desempeño de la red en dos situaciones distintas. Se han usado los mismos frames utilizados con la red RGB para ver la comparación.

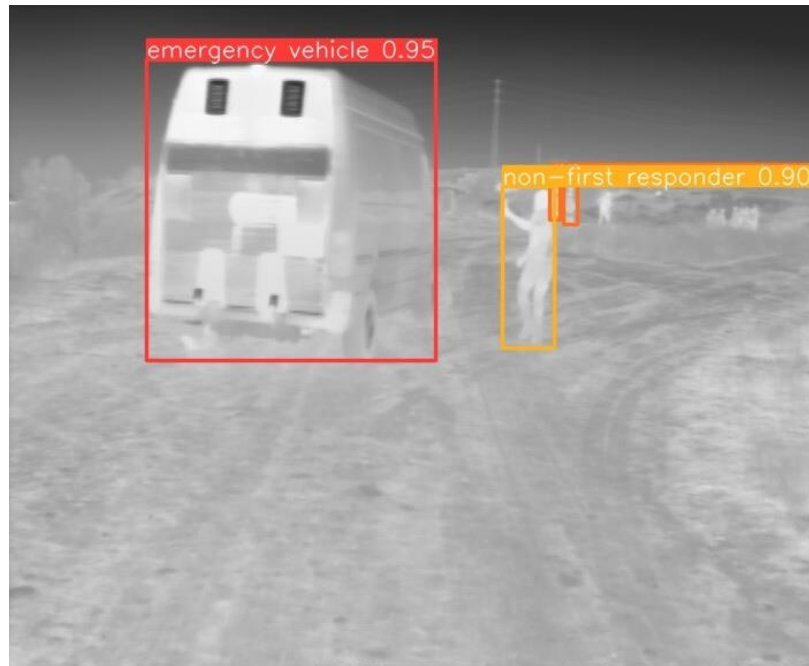


Figura 4.11: Detección de personal de rescate junto a un vehículo de rescate con la red térmica



Figura 4.12: Detección de distinto personal de rescate con la red térmica

Como se puede observar comparando ambas redes, hay veces que una red identifica objetos que la otra no es capaz y viceversa.

4.4. Mejora de las redes

Aunque los resultados mostrados en los apartados anteriores fueron los primeros entrenamientos, estos acabaron siendo los mejores obtenidos con la red. Pero no sin antes intentar una serie de mejoras de los mismos. Se probaron una serie de métodos:

- **Modificación del número de épocas:** Para encontrar el número de épocas óptimo se recomienda utilizar el método de *early stopping* o **parada temprana**. Sin embargo, cuando entrené las redes desconocía la manera automática de llevar a cabo este método. Por lo tanto, lo que hice fue llevar a cabo varios entrenamientos con distintas épocas observando en la gráfica de pérdidas cuándo estas cambiaban de tendencia positiva. Observando los resultados se obtuvo un valor aproximado de 80 épocas para evitar el sobreentrenamiento. Por lo tanto, este fue el número de épocas utilizado para los entrenamientos y, de hecho, el que dio mejores resultados.
- **Aumento de datos o Data augmentation:** El aumento de datos consiste en modificar las imágenes del conjunto de datos del entrenamiento para añadir las nuevamente como imágenes nuevas al conjunto de entrenamiento, produciendo así en ciertos casos una mejora del entrenamiento por el aumento de los datos. En este caso, la red implementa aumento de datos de forma predeterminada y, aunque se probó a modificar los parámetros del aumento de datos tras los primeros entrenamientos, no se observó ninguna mejoría. En concreto, se modificaron los siguientes parámetros:
 - Shear: Controla la deformación de la imagen. Su valor puede ir de -0.5 a 0.5 siendo 0 ninguna deformación aplicada -0.5 deformación aplicada hacia la izquierda y 0.5 deformación del 50% en cualquier dirección.
 - Translate: Traslación de la imagen en dirección aleatoria (de 0 a 1)
 - Scale: Variar la escala de una imagen de forma aleatoria (de 0 a 1)
 - Perspective: Se utiliza para aplicar una transformación de la perspectiva aleatoria (de 0 a 1)
 - Flipud: Probabilidad de girar la imagen verticalmente (de 0 a 1)
 - Fliplr: Probabilidad de girar la imagen horizontalmente (de 0 a 1)
 - Mosaic: Probabilidad de juntar varias imágenes en una, técnica de mosaico (de 0 a 1)
 - Mixup: Probabilidad combinar dos imágenes creando una nueva (de 0 a 1)

Sin embargo, en nuestro caso, la modificación del aumento de datos no mejoró los resultados iniciales tampoco.

Tampoco se siguieron investigando más métodos debido a que los resultados ya obtenidos eran bastante buenos. Por lo tanto, se utilizarán los pesos obtenidos en el primer entrenamiento para ambas redes. Para observar los resultados obtenidos se puede acceder a mi repositorio de Github [36].

Capítulo 5

5. Fusión de las redes

5.1. Introducción

El objetivo de este capítulo es explicar el algoritmo que se ha usado para fusionar tanto la red térmica como la RGB para obtener una detección final conjunta que utilice las detecciones de ambas redes.

La fusión era y ha sido el objetivo más complejo de este proyecto, debido a todo lo que conlleva. En primer lugar, cabe destacar que este es un trabajo fin de grado, el cual tiene un trabajo inmenso detrás, pero para completar este proyecto al 100% es necesario hacer un trabajo de varios años, probablemente como doctorado. De hecho, ya se conocen varios doctorados que han trabajado al mismo tiempo en esta idea de proyecto.

Por la complejidad que conlleva, se han contemplado varias ideas y versiones del proyecto que se explicarán en este capítulo, llegando finalmente a una versión definitiva.

5.2. Fusión probabilística

Esta fue la primera versión del algoritmo de fusión y su idea inicial fue la creación de un algoritmo probabilístico, similar a una red bayesiana, que utilizara tanto la red térmica como la red RGB para obtener una detección final mejorada. Para esta primera versión se tuvo en cuenta la iluminación de la imagen, la cual se obtuvo analizando el histograma normalizado de la imagen rgb. Además, se utilizó una tabla de probabilidad como la mostrada en la tabla 5.1 junto a la regla del producto, aunque **finalmente este concepto fue erróneo y se tuvo que desechar esta versión**. Aunque se trabajó mucho en esta versión, se desarrolló tomando la detección de la red térmica y la detección de la red RGB como sucesos independientes, cuando claramente, están altamente relacionados. Por lo tanto, para usarlos, se precisaba de un estudio estadístico para así obtener la probabilidad condicionada, la cual no se conocía. Este estudio no se llevó a cabo debido al tiempo disponible.

	RGB	NO RGB	
TERM	a	b	term_prob
NO TERM	c	d	no_term_prob
	rgb_prob	no_rgb_prob	

Tabla 5.1: Tabla de probabilidad

En esta tabla de probabilidad la *term_prob* era la probabilidad de que detectara la red térmica, *no_term_prob* era su probabilidad complementaria, y *rgb_prob* junto a *no_rgb_prob* era lo mismo para la red rgb. Tanto *term_prob* como *rgb_prob* eran las probabilidades que nos daba cada red de que fueran ciertas para cada caso. Las probabilidades del interior sumaban juntas las del exterior de la tabla uniendo filas o columnas. Así, las probabilidades *a* y *b* sumaban juntas la probabilidad *term_prob*, o las *a* y *c* sumaban *rgb_prob*. Se le daba un valor predeterminado muy pequeño al caso menos probable (*a*, *b*, *c* o *d*) que solía ser 0.01, y se calculaban el resto de los valores necesarios.

Uno de los primeros problemas que se tuvo que abordar fue el ajuste de las imágenes de la cámara térmica y de la cámara rgb. La cámara térmica tiene un zoom implantado bastante apreciable, por lo que para conseguir que ambas cámaras detectaran “la misma imagen” se tuvo que recortar y redimensionar las imágenes rgb para que se adaptaran al zoom de la térmica. Para ello se creó una simple función que modificaba cada imagen rgb que llegaba desde la cámara.

Este algoritmo empieza con la lectura de los pesos obtenidos del entrenamiento para poder ejecutar la red y hacer detecciones. Tras el ajuste previamente comentado y haber procesado las imágenes con las redes, se leen las anotaciones generadas por las redes y empieza la fusión:

- 1) Se identifican los objetos detectados por ambas redes. Para decidir si un objeto detectado por una red coincide con otro detectado por la otra, se usa el **IoU**, si supera cierto umbral se considera el mismo objeto.
- 2) Se clasifican los objetos por cada tipo de detección:
 - Si ambas redes han detectado el mismo objeto y coinciden en la misma clase específica, se utiliza una tabla de probabilidad para obtener la probabilidad final condicionada.
 - Si ambas redes han detectado un mismo objeto, pero no coinciden en clase específica, se utiliza la iluminación media de la imagen (se le da un valor probabilístico predeterminado a cada red dependiendo de la iluminación que haya en la imagen) y se utiliza la regla del producto de probabilidad mostrada en la ecuación 5.1 para obtener la probabilidad final.
 - Si una red ha detectado un objeto, pero la otra no, se vuelve a usar la tabla 5.1 de probabilidad, pero cambiando el caso desfavorable.

$$P(A|B) = \frac{P(A \wedge B)}{P(B)} \quad (5.1)$$

La ecuación 5.1 es **la regla del producto** de probabilidad y dice que $P(A|B)$, la probabilidad de que ocurra el suceso A dado el suceso B es igual a $P(A \wedge B)$, la probabilidad de que ocurran los sucesos A y B a la vez partido $P(B)$, la probabilidad de B.

Sin embargo, una vez se creó y se probó todo, nos dimos cuenta de una serie de errores conceptuales que estábamos teniendo, por lo que se acabó desechando esta versión y creando otra nueva. Cabe destacar que la búsqueda de información llevada a cabo durante esta versión nos permitió entender mejor el problema y nos ayudó a llevar a cabo el último algoritmo.

Este algoritmo está pasado a código en **Python** y se puede observar en mi repositorio de Github [36] bajo el nombre de “fusion.py”.

5.3. Fusión heurística inicial

Esta fue la segunda versión del algoritmo. Aquí, se siguió intentando encontrar una forma probabilística de abordar el problema (usando leyes de probabilidad). Sin embargo, se llegó a la conclusión mencionada anteriormente, que, para poder crear una red bayesiana o cualquier otro tipo de algoritmo probabilístico, se tendría que hacer un estudio estadístico previo, para lo cual no disponía del tiempo suficiente. Por lo tanto, se optó por desarrollar un algoritmo heurístico donde se calcula la probabilidad final de una forma más imprecisa. **Esta no es la versión final que se utilizará.** El algoritmo utilizado es el siguiente:

- 1) Si ambas redes detectan un mismo objeto y coinciden en la clase específica, se hace una media de la confianza obtenida por ambas redes y se aplica un coeficiente de 1.2 para reforzar esta detección conjunta. Este coeficiente se aplica siempre y cuando el resultado no supere el 99% de confianza.
- 2) Si ambas redes detectan el mismo objeto, pero difieren en la clase específica, en vez de hacer la regla del producto como en la versión anterior junto a los fallos que eso conllevaba, se multiplica la confianza de cada red por un coeficiente asignado que dependerá de la iluminación de la imagen. La red con mayor resultado indicará la clase final. La confianza final se calcula con la media de las confianzas de las redes.
- 3) Si el objeto es detectado solo por una de las redes se elimina la condición de iluminación y se continúa utilizando la tabla 5.1 de probabilidad.

El problema principal de este algoritmo es que utilizaban ponderaciones “a ojo”, es decir, no es muy “científico” un algoritmo donde siempre se multiplicaba por 1.2 la probabilidad para ciertos casos excepto cuando este valor superaba el 100% de probabilidad, y en otros casos siempre se multiplicaba por el valor aleatorio 0.8. Por ello, se intentó buscar otro método menos aleatorio.

Este algoritmo está pasado a código en **Python** y se puede observar en mi repositorio de Github [36] bajo el nombre de “fusion_v2.py”.

5.4. Fusión heurística final

Esta es la **versión definitiva** del algoritmo de fusión. En esta versión se diseñó otro algoritmo heurístico, pero se mejoró significativamente frente al anterior. Se consiguió obtener un algoritmo donde las ponderaciones dependían de la propia probabilidad obtenida, en vez de un valor “a ojo”. Además, en este caso se comprobó el resultado obtenido en una pequeña muestra y se obtuvieron resultados muy próximos a la realidad. En este caso, se desestimó el uso de la iluminación debido a que la totalidad de imágenes de las que se disponía eran imágenes diurnas, con suficiente luz, como para que este parámetro

influyese a la hora de hacer las pruebas. En primer lugar, se explicará el algoritmo utilizado para después explicar el proceso de comprobación que hubo detrás.

Para empezar, se van a definir dos conceptos: P , que es la confianza máxima alcanzada por cualquiera de las redes para un objeto específico, y Q que es la confianza mínima de las dos. A continuación, se relata el algoritmo seguido:

- 1) Si ambas redes detectan un mismo objeto y coinciden en la predicción, la confianza final de la red será calculada mediante la ecuación 5.2.

$$conf_{Final} = P + (1 - P) * Q \quad (5.2)$$

- 2) Si ambas redes detectan un mismo objeto, pero clases detectadas son distintas en la clase específica teniendo la misma clase raíz, la confianza final de la red será calculada mediante la ecuación 5.3.

$$conf_{Final} = \frac{(P + (1 - P) * Q) + (P - P * Q)}{2} \quad (5.3)$$

- 3) Si ambas redes detectan un mismo objeto, pero distintas clases raíces, la confianza final de la red será calculada mediante la ecuación 5.4.

$$conf_{Final} = P - P * Q \quad (5.4)$$

- 4) Si una red detecta un objeto y la otra no, la confianza final es la obtenida por esa red.

Este algoritmo está pasado a código en **Python** y se puede observar en mi repositorio de Github [36] bajo el nombre de “fusion_v3.py”.

Una vez explicado el algoritmo, se desarrollará el proceso de validación que se ha llevado a cabo para comprobar los resultados del mismo. Debido a los procesos erróneos o poco precisos seguidos para obtener los otros algoritmos, además del escaso tiempo disponible para este proyecto, estos se desecharon y solo se comprobó este último.

5.4.1. Validación

Para comprobar el rendimiento de este algoritmo, se tuvo que crear una pequeña muestra de datos con imágenes etiquetadas. Si las imágenes de las que disponía fueran buenas y estuvieran bien coordinadas el trabajo de este proyecto se hubiera reducido a la mitad. Sin embargo, se disponía de imágenes obtenidas de un vídeo térmico y otro RGB. Cada vídeo iba a una velocidad distinta (frames/seg), dependiendo del vídeo se producían cortes en algún punto del vídeo aleatoriamente, lo cual hacía mucho más complejos coordinarlos, y además de la diferencia de zoom entre ambas cámaras, para cada una su centro estaba separado por unos centímetros, lo que generaba en algunos casos diferentes puntos de vista. Esto es un gran problema cuando el proyecto depende de tener dos imágenes iguales con distinta información para realizar la fusión

(información térmica y del espectro visible). Coordinar las imágenes ha sido, sin lugar a dudas, lo más tedioso de todo el proyecto cuando, en principio, no formaba parte de él.

El etiquetado de imágenes suele ser una de las partes más trabajosa y repetitiva a la hora del entrenamiento de redes neuronales. Sin embargo, en este caso, solo se tuvo que etiquetar una pequeña muestra, por lo que no requirió tanto trabajo. Para el etiquetado se utilizó el programa labelImg. Se buscaron 15 imágenes térmicas y RGB que coincidieran en el tiempo, para etiquetarlas a mano con este programa y poder comprobar el algoritmo. Se eligió este programa por su facilidad y rapidez de uso. En la figura 5.1 se puede observar una imagen siendo etiquetada con labelImg.

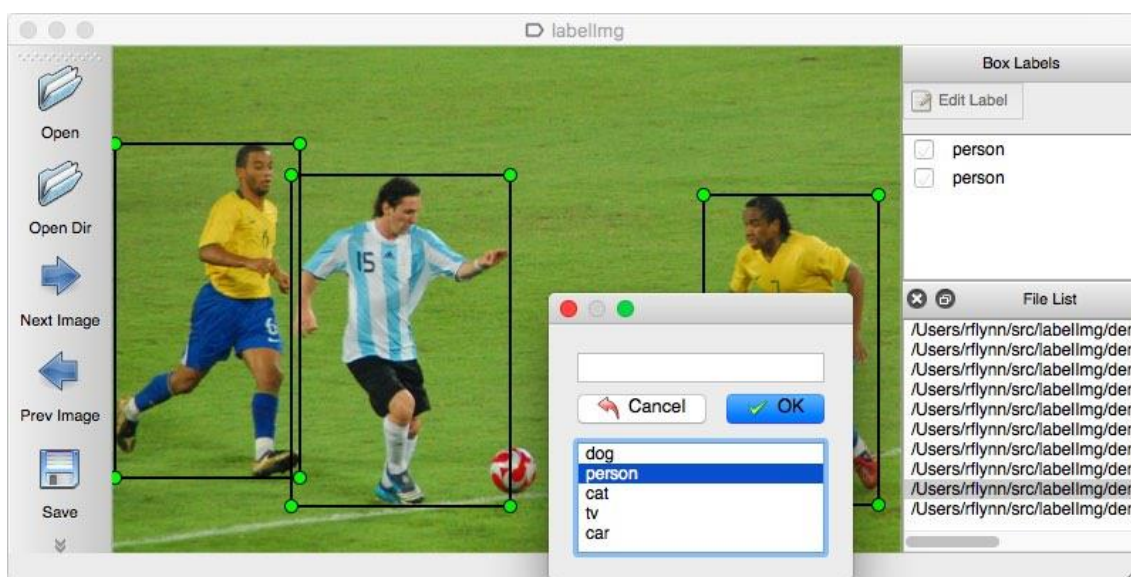


Figura 5.1: Etiquetado con LabelImg [35]

Una vez se etiquetaron las 15 imágenes (solo se etiquetaron las imágenes RGB tras recortarlas para ajustarlas a las imágenes térmicas, ya que la solución vendrá expresada a raíz de la imagen rgb recortada), se desarrollaron dos códigos en Python para crear un archivo .csv donde aparecieran los datos de cada objeto sin llegar a repetirse. El primer código fue “**get_csv_v2.py**” donde juntaba todos los objetos y los escribía en un csv. Mientras que el segundo “**fix_csv.py**”, leía el csv y eliminaba los objetos repetidos. Ambos códigos están presentes en mi repositorio de Github [36].

La creación de ambos códigos fue relativamente compleja, ya que se necesitaba comparar ya no solo la detección térmica con la rgb (que ya se había hecho en otros algoritmos), sino que también entraba la comparación con la detección real realizada, algo para nada simple. Además, otro factor que influyó en la complejidad de estos códigos conseguir agrupar aquellos objetos que fueran iguales, ya que las coordenadas del objeto en la imagen RGB y las coordenadas del objeto en la imagen térmica nunca coincidían exactamente. En primer lugar, para decidir si dos objetos eran en realidad el mismo objeto se utilizó el **IoU**, pero debido a la poca cantidad de objetos que coincidían se añadió otra parte al algoritmo donde si ambas cajas delimitadoras

coincidían mínimo un 30%, se comprobaba si sus centros estaban cercanos entre sí. En caso afirmativo, se concluía que eran el mismo objeto.

Se puede observar un extracto del archivo csv creado en la tabla 5.2, donde la columna llamada `n_image` corresponde al número de la imagen donde se está realizando la detección, las columnas acabadas en `real` corresponden a los datos de los objetos obtenidos a mano, las columnas acabadas en `rgb` corresponden a los datos de los objetos obtenidos por la red RGB, y las acabadas en `tir` corresponden a los datos de los objetos obtenidos por la red térmica. En los casos en los que una red no ha identificado ese objeto o ese objeto no ha sido relacionado con uno real, se clasifica como clase 5 que no existe y se ponen todos los valores de su caja delimitadora a cero.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	n_image	class_rgb	x_rgb	y_rgb	width_rgb	height_rgb	conf_rgb	class_tir	x_tir	y_tir	width_tir	height_tir	conf_tir	class_real	x_real	y_real	width_real	height_real
2	0	1	294	0	276	574	0,51	5	0	0	0	0	0	5	0	0	0	0
3	0	1	259	0	240	570	0,36	1	220	125	161	433	0,89	1	255	33	209	542
4	0	5	0	0	0	0	0	1	457	354	43	113	0,85	1	468	348	54	99
5	0	5	0	0	0	0	0	1	456	354	43	115	0,26	5	0	0	0	0
6	1	1	249	256	272	315	0,59	5	0	0	0	0	0	1	247	251	292	324
7	1	1	358	255	163	315	0,3	5	0	0	0	0	0	5	0	0	0	0
8	2	5	0	0	0	0	0	5	0	0	0	0	0	1	288	133	224	441
9	3	1	406	151	117	362	0,93	5	0	0	0	0	0	1	392	148	129	380
10	3	5	0	0	0	0	0	1	361	201	91	319	0,86	5	0	0	0	0
11	4	1	509	164	23	70	0,82	5	0	0	0	0	0	1	514	164	22	69
12	4	1	0	0	334	571	0,8	5	0	0	0	0	0	1	36	125	308	450
13	4	1	491	170	18	81	0,45	5	0	0	0	0	0	1	498	169	15	84
14	4	5	0	0	0	0	0	5	0	0	0	0	0	1	477	169	21	88
15	5	1	679	117	24	98	0,85	5	0	0	0	0	0	1	673	119	30	97
16	5	5	0	0	0	0	0	5	0	0	0	0	0	1	407	130	263	444
17	6	1	489	50	214	524	0,9	5	0	0	0	0	0	1	488	94	215	480
18	6	1	473	158	47	151	0,89	5	0	0	0	0	0	1	469	153	50	163
19	6	1	583	66	75	133	0,81	5	0	0	0	0	0	5	0	0	0	0
20	6	1	547	37	23	89	0,78	5	0	0	0	0	0	5	0	0	0	0
21	6	1	514	159	27	88	0,69	5	0	0	0	0	0	1	518	151	32	88
22	6	1	564	53	139	522	0,37	5	0	0	0	0	0	5	0	0	0	0
23	6	1	613	71	90	504	0,26	5	0	0	0	0	0	5	0	0	0	0
24	7	1	347	133	226	438	0,94	1	258	224	227	341	0,26	1	334	136	237	440
25	7	1	294	180	61	198	0,85	5	0	0	0	0	0	1	280	179	77	216
26	7	1	280	178	32	199	0,59	5	0	0	0	0	0	5	0	0	0	0
27	7	1	608	198	22	64	0,52	5	0	0	0	0	0	1	608	202	28	59
28	7	1	209	162	79	264	0,51	5	0	0	0	0	0	1	236	157	68	277
29	8	1	407	138	216	433	0,96	5	0	0	0	0	0	1	399	141	226	434
30	8	1	368	196	27	89	0,81	5	0	0	0	0	0	1	363	218	35	68
31	8	0	11	389	163	170	0,69	5	0	0	0	0	0	5	0	0	0	0
32	8	5	0	0	0	0	0	1	286	222	203	331	0,86	5	0	0	0	0
33	8	5	0	0	0	0	0	1	632	212	30	104	0,67	1	639	219	33	98
34	8	5	0	0	0	0	0	1	591	219	40	95	0,31	5	0	0	0	0
35	8	5	0	0	0	0	0	5	0	0	0	0	0	1	673	217	30	99
36	9	1	271	0	352	574	0,72	0	320	9	306	329	0,66	1	272	156	248	420
37	9	1	134	231	30	88	0,65	1	148	225	30	97	0,4	1	132	228	39	108
38	9	1	145	231	23	93	0,26	5	0	0	0	0	0	5	0	0	0	0

Tabla 5.2: Extracto del archivo .csv creado

Con el csv hecho se pasó a realizar la comprobación mediante un código en el lenguaje de programación R. Este código se llama **Validacion.Rmd** y se puede encontrar en mi repositorio de Github [36].

Para comprobar el buen funcionamiento de este algoritmo se calcularon una serie de parámetros:

- **Confianza Global Muestral (CGM):** cálculo de la confianza global de la muestra para cada red, es decir, cuántas detecciones (número de clase) térmicas, rgb y predicha por el algoritmo creado coinciden con la detección real.

- **Media de Confianzas Individuales (MCI):** cálculo de la media de la confianza de cada red. La media de la confianza de los objetos rgb, de los objetos térmicos y de los objetos predichos.

Tras calcular estos dos parámetros para cada red, se obtienen los resultados de la tabla 5.3.

TIPO DE RED	CGM	MCI	DIFERENCIA	DIFERENCIA (%)
TÉRMICA	0.4179104	0.2801493	0.1377611	32.96
RGB	0.5671642	0.4034328	0.1637314	28.87
FUSIÓN	0.4626866	0.5912463	0.1285597	21.74

Tabla 5.3: Resultados de la muestra

El objetivo de esta validación era comprobar con una pequeña muestra, que el resultado obtenido con el algoritmo generaba una respuesta respecto al valor real (imágenes etiquetadas) similar a la generada por las otras redes, es decir, que la diferencia entre el CGM y el MCI de cada red y el del algoritmo de fusión fueran similares.

Analizando la tabla 5.3 se observa, tal y como se esperaba, un valor parecido entre la media de las confianzas individuales de cada red y la confianza global obtenida de la muestra para cada red. Haciendo esta misma comparación entre la media de las confianzas individuales obtenidas por el algoritmo de fusión y la confianza global, se observa también un valor parecido que mejora el de cada red considerada individualmente. Por lo tanto, se puede afirmar que el algoritmo heurístico creado se acerca bastante a la realidad.

5.4.2. Resultados

Finalmente, tras todo el proceso llevado a cabo para crear la fusión, se muestra su correcto funcionamiento en las figuras 5.2, 5.3, 5.4 y 5.6, comparando los resultados de la fusión junto a los resultados de cada red individual. La red fusionada es la imagen de la izquierda, la red RGB es la del centro y la red térmica es la de la derecha.



Figura 5.2: Comparación de las redes para imagen con dos personas de rescate [Fusión-RGB-Térmica]



Figura 5.3: Comparación de las redes para imagen con dos vehículos de emergencia y una persona de rescate [Fusión-RGB-Térmica]

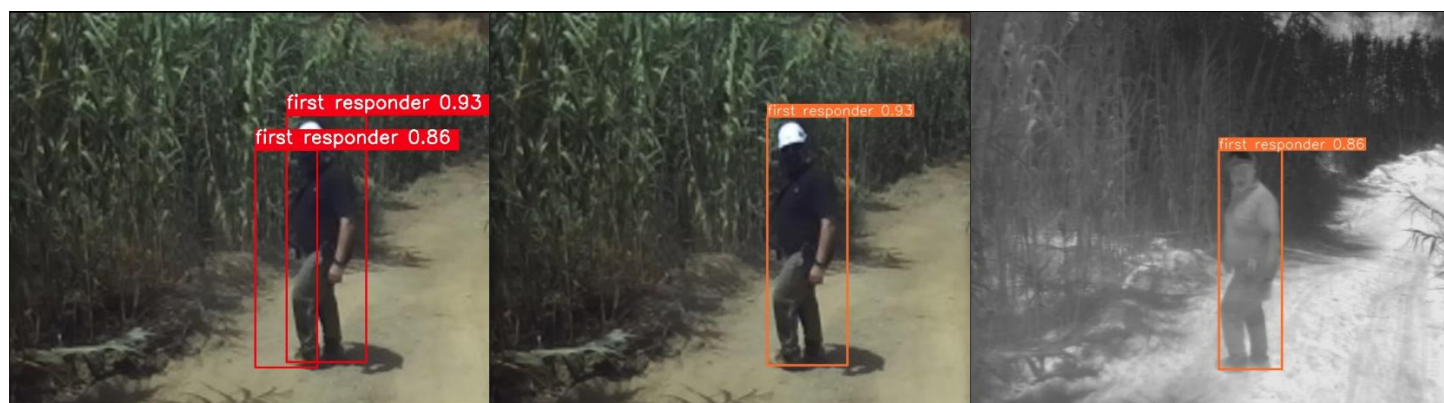


Figura 5.4: Comparación de las redes para imagen con una persona de rescate [Fusión-RGB-Térmica]

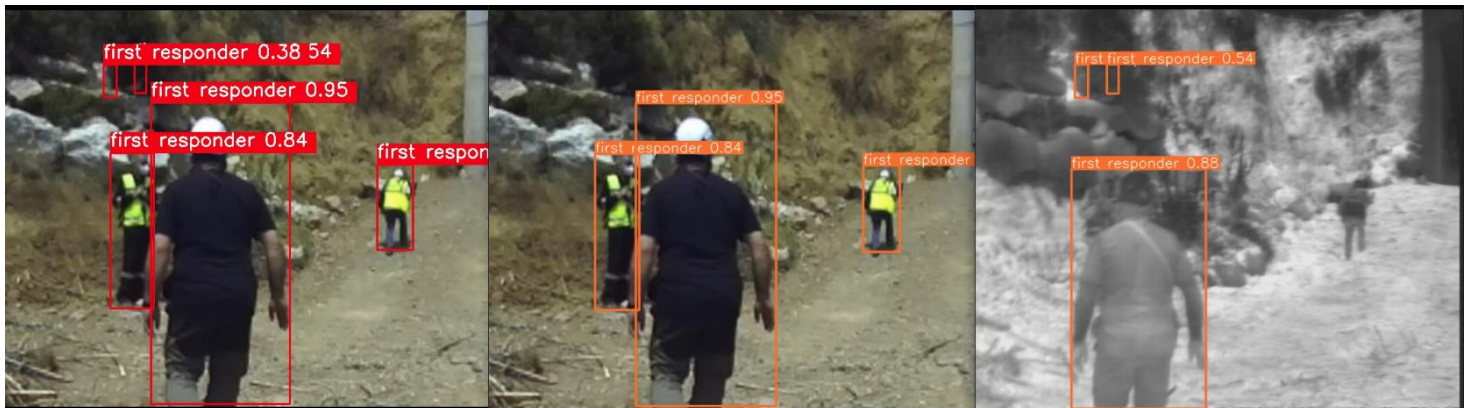


Figura 5.5: Comparación de las redes para imagen con distinto personal de rescate [Fusión-
RGB-Térmica]

Como se puede observar en las imágenes anteriores, las detecciones de las redes individuales no son perfectas. Además, la diferencia de distancia en el foco de cada cámara acaba influyendo levemente, a veces incluso duplicando un mismo objeto. Sin embargo, se ha conseguido mitigar este efecto lo mejor posible, obteniendo los buenos resultados mostrados en las imágenes. Se puede acceder a los resultados y códigos utilizados y descargarlos desde mi repositorio de Github [36].



UNIVERSIDAD
DE MÁLAGA



Capítulo 6

6. Implementación de la red

6.1. Introducción

La implementación de la red se ha hecho en el ordenador de a bordo Jetson Orin. Un pequeño ordenador con una gran potencia gráfica diseñado específicamente para correr este tipo de redes neuronales en robots móviles con un rápido tiempo de respuesta. Para poder utilizar este ordenador se tuvo que investigar cómo configurarlo e instalar el sistema operativo (se optó por Ubuntu 20.04) ya que es un poco distinto a un ordenador convencional y produjo algunos problemas al principio.

Tras configurar el ordenador de a bordo, se procedió a instalar **ROS2** y todas las dependencias necesarias para este proyecto. Una vez todo instalado se comprobó el correcto funcionamiento del algoritmo creado y se empezó a diseñar la parte de comunicación.

6.2. Comunicación mediante ROS2

Este proyecto está enfocado a la detección de objetos en tiempo real mediante dos cámaras implementadas en un robot terrestre. Debido a esto, se ha utilizado tanto el ordenador de a bordo mencionado antes, como el sistema de comunicación ROS2, ya que ambos están diseñados para este ámbito.

Para la comunicación mediante ROS2, se buscaba un sistema capaz de recibir imágenes, tanto RGB como térmicas, para después procesarlas y crear un archivo txt con las anotaciones finales fusionadas, y por último enviar estas anotaciones por ROS para que cualquiera pudiera recibirlo junto a las imágenes.

En un principio se intentó crear el código desde 0. Sin embargo, no fuimos capaces de que ROS2 reconociera la librería de la red neuronal, **Ultralytics**. Por lo tanto, se duplicó un proyecto de github [36] que incluía **YOLOv8** en **ROS2**, lo cual no generó fallos al incluir la librería, y se implementaron los códigos necesarios para que funcionara nuestro proyecto.

Un problema que hubo fue que, para probar el sistema, se tenían que enviar las imágenes mediante otro código para comprobar su funcionamiento y otro código que visualizara las imágenes resultantes junto a sus anotaciones, creando así tres códigos que se pueden encontrar en mi repositorio de Github [36].

El primer código es “img_publisher.py”. Este, abre dos vídeos, uno con imágenes dentro del espectro visible y otro con imágenes térmicas, los lee y envía sus frames por dos topics de ROS2 (“term_topic” y “rgb_topic”) para poder ser leídos por cualquiera que se conecte a esos topics.

El segundo código es “fusion_ros.py”. Este, lee las imágenes de los dos topics anteriores, “rgb_topic” y “term_topic”, procesa las redes neuronales, las fusiona y gracias a un tipo de mensaje customizado que se ha creado con ROS2, se crea la anotación final y se envía por otro topic (“annotations_topic”). Además, de cara a la comprobación de los resultados, este código también envía las imágenes con las detecciones predichas por cada red mediante los topics “rgb_res_img” y “tir_res_img”.

El último código es “visualization.py”. Este es una **aplicación para visualizar los resultados** que se han generado. Lee todos los topics y mediante la librería Opencv de Python, muestra las imágenes resultado de las redes, una en cada ventana diferente, junto a la imagen rgb recortada con las detecciones leídas desde el topic “annotations_topic” implementadas en la imagen.

La ejecución de estos tres códigos a la vez provocó que el ordenador de a bordo fuera un poco más lento de lo que iría cuando funcione solo procesando las redes y enviándolas a otros dispositivos debido a la carga de procesamiento a la que está sometida.

6.3. Análisis de resultados

Tras probar el rendimiento de la fusión en la Jetson Orin, teniendo en cuenta los procesos que hay en segundo plano (lectura y envío de imágenes a raíz de un vídeo, además del procesamiento y muestra de los resultados) que en un caso real no deberían estar ejecutándose, la velocidad de reacción conseguida para el procesamiento de las redes neuronales viene mostrada en la tabla 6.1. Al fin y al cabo, el procesamiento de la red térmica y el de la red RGB son los mayores cuellos de botella del proyecto.

En la tabla 6.1 se recogen los tiempos de preprocesado, inferencia y postprocesado de cada red para una muestra de 50 imágenes junto al tiempo total por imagen. Para observar los valores medios, la varianza y la desviación estándar de cada paso, hay que irse a la tabla 6.2. Los resultados obtenidos indican que la red RGB es ligeramente más lenta que la red térmica (29 ms), esto es comprensible dado que las imágenes RGB contienen más información, por lo que son más lentas de procesar por la red neuronal. A pesar de esto, la diferencia es prácticamente inapreciable. Sin embargo, el dato más importante es el tiempo medio que tardan las dos redes en procesarse, una detrás de la otra. Se observa que el tiempo medio para cada imagen es de 1905.27 ms, es decir, la red enviaría un resultado cada aproximadamente 1.9 segundos si se procesa con la Jetson Orin, o incluso un poco menos en casos donde este ordenador solo se enfoque a esta tarea como se mencionó antes. Analizando la desviación estándar de la tabla 6.2, se observa que no hay una gran dispersión de los datos analizados en la muestra, es decir, la mayoría concuerdan con la media obtenida variando una media de ± 42 ms en relación a los 1905 ms de la media total.

RED RGB				RED TÉRMICA				TIEMPO TOTAL
Preprocesado	Inferencia	Postprocesado	Total (ms)	Preprocesado	Inferencia	Postprocesado	Total (ms)	ms
6,3	987	1,1	994,4	6,1	934,6	1,8	942,5	1936,9
6,6	952,7	1,8	961,1	6,1	921,9	1,8	929,8	1890,9
6,5	970,5	1,8	978,8	5,6	914,8	1,7	922,1	1900,9
5,7	961,1	1,8	968,6	5,8	938,8	2	946,6	1915,2
5,2	938,4	1,7	945,3	6	944,5	1,7	952,2	1897,5
5,9	962,5	1,8	970,2	5,7	939	1,8	946,5	1916,7
5,3	934,9	1,8	942	5,3	909	1,8	916,1	1858,1
5,5	955,1	1,8	962,4	5,8	913,9	1,8	921,5	1883,9
5,7	952,2	1,8	959,7	5,5	937,7	2	945,2	1904,9
6,1	989,9	1,7	997,7	6,2	925,3	1,8	933,3	1931
5,9	963,3	1,8	971	6,1	933,6	1,8	941,5	1912,5
5,8	948,9	1,8	956,5	6,1	933,6	2	941,7	1898,2
5,2	963,8	1,8	970,8	5,7	923,6	1,8	931,1	1901,9
5,3	932	1,8	939,1	6,2	921,6	1,8	929,6	1868,7
5,4	952,8	1,8	960	5,8	914,9	1,7	922,4	1882,4
5,4	948,3	1,8	955,5	5,6	930,6	1,7	937,9	1893,4
5,2	942,2	1,8	949,2	5,7	939,6	2	947,3	1896,5
5,9	966	1,8	973,7	5,8	904,1	1,8	911,7	1885,4
5,5	958,9	1,8	966,2	5,5	919,1	1,8	926,4	1892,6
5,9	958,1	1,7	965,7	5,7	916,6	1,8	924,1	1889,8
5,8	938,7	1,7	946,2	6	936,7	1,8	944,5	1890,7
5,2	962,8	1,9	969,9	6,1	936,1	1,8	944	1913,9
5,8	948,2	1,7	955,7	5,4	937,5	1,8	944,7	1900,4
5,6	968,8	1,8	976,2	6,1	931	1,8	938,9	1915,1
5,9	943,9	1,8	951,6	5,8	896,7	1,8	904,3	1855,9
5,4	950,1	1,8	957,3	5,8	935,2	2	943	1900,3
5,6	955	1,8	962,4	6,4	984,6	1,8	992,8	1955,2
5,5	947,8	1,8	955,1	5,7	903,5	1,8	911	1866,1
5,4	1002,8	1,8	1010	6,1	931,2	1,8	939,1	1949,1
5,4	989,9	1,9	997,2	5,9	932,8	1,9	940,6	1937,8
5,2	959,5	1,8	966,5	6,1	917,2	1,8	925,1	1891,6
5,3	942,5	2,1	949,9	5,6	984,3	1,8	991,7	1941,6
5,5	944,4	1,8	951,7	5,7	939,1	1,8	946,6	1898,3
5,5	979,7	1,7	986,9	6	933	1,7	940,7	1927,6
5,6	949,9	1,8	957,3	5,9	919,2	1,8	926,9	1884,2
5,7	949,4	2	957,1	5,7	954,4	1,8	961,9	1919
5,7	945,1	1,8	952,6	6,3	958,5	1,8	966,6	1919,2
5,2	941,5	1,8	948,5	5,7	928,2	1,8	935,7	1884,2
5,4	965,8	1,8	973	6,4	952,4	1,9	960,7	1933,7
5,7	931,2	1,8	938,7	5,6	926,2	1,8	933,6	1872,3
5,9	939,6	1,8	947,3	6	917	1,9	924,9	1872,2
5,3	948,3	1,7	955,3	6	938,6	1,8	946,4	1901,7
5,8	952,7	1,8	960,3	6,2	920,2	1,8	928,2	1888,5
6,2	974,3	1,8	982,3	5,8	912,6	1,8	920,2	1902,5
5,3	936,6	1,9	943,8	6	917,2	2	925,2	1869
5,4	942,9	1,8	950,1	5,9	928,3	1,8	936	1886,1
5,9	999,4	1,8	1007,1	5,9	927,8	1,9	935,6	1942,7
5,5	959,1	1,8	966,4	6	919,3	1,8	927,1	1893,5
5,2	934,7	1,8	941,7	5,5	902,5	2	910	1851,7
6,1	1148,3	1,8	1156,2	7,4	976,8	1,8	986	2142,2

Tabla 6.1: Tiempos de procesamiento de las redes



	RED RGB				RED TÉRMICA				TIEMPO TOTAL
	Preprocesado	Inferencia	Postprocesado	Total (ms)	Preprocesado	Inferencia	Postprocesado	Total (ms)	ms
Tiempo medio (ms)	5,626	959,83	1,788	967,244	5,906	930,298	1,826	938,03	1905,274
Varianza	0,120	1025,891	0,015	1032,543	0,110	346,549	0,007	352,063	1766,630
Desviación estándar	0,346	32,030	0,121	32,133	0,332	18,616	0,083	18,763	42,031

Tabla 6.2: Estadísticas del tiempo de ejecución

Capítulo 7

7. Conclusiones y Trabajo Futuro

7.1. Conclusiones

El objetivo de este proyecto era la creación de un algoritmo de fusión a raíz del entrenamiento de dos redes neuronales, una red térmica y otra red RGB (con imágenes dentro del espectro visible) para detectar personas y vehículos dentro de imágenes en situaciones de catástrofe, añadiendo, en última instancia, una forma de comunicación para implementar el proyecto en un robot terrestre enfocado al procesamiento en tiempo real. Las conclusiones extraídas tras desarrollar este proyecto se presentan a continuación.

Entrenamiento de las redes térmica y RGB

El entrenamiento de cada red ha producido muy buenos resultados por separado a pesar de las “pocas” imágenes con las que se han entrenado. Esto puede ser debido al alto nivel de desarrollo llevado a cabo por la empresa Ultralytics para crear esta nueva versión de YOLO llamada YOLOv8. A pesar de todo, para obtener resultados incluso mejores, capaces de adaptarse a un mayor número de paisajes para hacer buenas detecciones, se debería aumentar el conjunto de entrenamiento, aumentando la versatilidad de las redes.

Sincronización de imágenes térmicas y RGB

Las imágenes obtenidas con la cámara térmica y las obtenidas con la cámara rgb han sido muy difíciles de coordinar, siendo esta la peor parte del proyecto. Por culpa de esto, me he visto obligado de hacer una selección a mano de imágenes similares térmicas y rgb para probar los resultados en numerosas ocasiones.

En primer lugar, la cámara térmica tenía un zoom considerable debido a que esta estaba enfocada a su uso desde las alturas. Para poder sincronizar esta cámara con la rgb se han tenido que preprocesar las imágenes rgb con un recorte más un redimensionamiento de las mismas.

En segundo lugar, ambas cámaras están desplazadas aproximadamente 20cm una de la otra, variando esta distancia dependiendo del año del que se extraiga el vídeo o incluso dependiendo del momento del vídeo, porque a veces se movían las cámaras. Este no sería un problema muy grande si la distancia fuera fija siempre para los casos de objetos lejanos a mínimo 3-4 metros de distancia de la cámara. Sin embargo, el movimiento dentro de los vídeos genera problemas a veces incluso para estos casos de objetos “lejanos”, aunque cuanto mayor es la distancia menor es la diferencia de posición (se acercan al “infinito”). Para arreglar en cierta medida este problema se podría hacer un cambio de coordenadas en el origen de las cámaras, pero esto requiere una distancia fija entre ellas y, además, mi proyecto se ha alargado tanto que no me ha sido posible realizar esta parte.

En tercer lugar, los vídeos generados por la cámara térmica y la cámara rgb iban a distintas velocidades o frames por segundo. Esto producía que a la hora de sincronizar un vídeo con el otro tuviera que desechar muchas imágenes captadas por la cámara rgb para obtener los mismos frames. El problema principal generado por esto era que las velocidades de ambas cámaras no eran exactas, sino que la cámara térmica leía a 2.75 frames por segundo mientras que la rgb leía a 14 frames por segundo, por lo tanto, era difícil de compatibilizar exactamente.

En cuarto y último lugar, los vídeos no empezaban exactamente a la vez, sino que el vídeo rgb tenía 7 segundos de retraso. Por lo tanto, se tuvieron que sincronizar a mano. Pero la peor parte de todo fue que debido a fallos de transmisión, cada vídeo sufría cortes de unos segundos de forma aleatoria y no sincronizada, por lo tanto, si quería sincronizar dos vídeos, era necesario mirar frame a frame dónde había cortes y eliminar esa parte del otro vídeo. Al final se optó por escoger frames a mano que pudiera comprobar que se trataban del mismo momento del vídeo.

Fusión de las redes

El algoritmo de fusión ha sido una de las partes más complejas de este proyecto. Aunque en un principio se quería conseguir un algoritmo probabilístico usando redes bayesianas u otras leyes de probabilidad y de hecho se intentó, se acabó desechando esta idea por la necesidad de datos que solo se podían obtener con un análisis estadístico que no podía implementar en mi proyecto debido al tiempo. Se acabó utilizando un algoritmo heurístico junto a una comprobación asociada a una muestra. Esto demostró el buen rendimiento de este algoritmo y finalizó esta parte en la que tanto aprendí pero que no se llegó a completar con la idea inicial.

Conclusión final

Sopesando todo el trabajo llevado a cabo y analizando los resultados obtenidos, concluimos que, de haber tenido disponibles mejores datos (imágenes térmicas y RGB bien sincronizadas), podría haberse reducido bastante la complejidad de este proyecto. Además, debido a los buenos resultados de las redes entrenadas por separado, no está del todo clara la mejora que conlleva esta fusión en relación con todo el trabajo que tiene detrás. Sin embargo, se ha creado una buena aplicación de comunicación para el procesamiento, la transmisión y la visualización de estas redes mediante ROS2. **Se puede encontrar todo el código desarrollado en mi repositorio de Github [36].**

7.2. Trabajo Futuro

Tras haber finalizado los objetivos propuestos para este proyecto, se han encontrado una serie de líneas de desarrollo que podrían mejorarlo y se proponen para su continuación:

- Aumento del conjunto de imágenes de entrenamiento haciendo un etiquetado a mano de imágenes variadas que mejore la versatilidad de las redes.



- Sincronización de las coordenadas del foco de cada cámara para que visualicen la misma imagen.
- Realización de un estudio estadístico para obtener la probabilidad de dependencia entre la detección de un objeto por una red y la detección de ese mismo objeto por la otra.
- Modificación del algoritmo de fusión utilizando leyes probabilísticas y no heurísticas, tras la realización del estudio estadístico mencionado anteriormente, con el objetivo de publicar un artículo de investigación.



UNIVERSIDAD
DE MÁLAGA



Anexo A: Docker

A.1. Introducción

Tras la breve explicación de Docker llevada a cabo en el capítulo 3, en este anexo se desarrollará un poco más en profundidad y se explicarán los pasos llevados a cabo para poder ejecutar el contenedor de Docker para el entrenamiento de las redes.

Docker funciona con **contenedores**, que son las pseudo máquinas-virtuales mencionadas anteriormente. Cada contenedor puede ser creado con cualquier sistema operativo y cualquier programa, permitiendo así una gran versatilidad para el programador sin tener restricciones por su sistema operativo base. Los contenedores de Docker no poseen interfaz gráfica, lo que los hace muy ligeros y los diferencia principalmente de las máquinas virtuales convencionales. Además, es mucho más rápido crear un contenedor de Docker que una máquina virtual convencional.

La versatilidad de los contenedores permite crear **imágenes de Docker** (estructura o funcionalidad del contenedor) muy específicas para cada programa que se quiera utilizar. Una vez se tiene la imagen de Docker, se puede crear el contenedor, ejecutarlo e incluso entrar dentro de él para modificar parámetros del programa en ejecución.

Para utilizar una imagen de Docker se puede descargar una ya creada directamente de **Docker Hub** [37], una gran librería de imágenes oficiales de Docker, o te puedes crear la tuya propia mediante un archivo llamado **Dockerfile**. Este archivo descarga una imagen de Docker Hub y ejecuta una serie de comandos para configurar tu contenedor con las dependencias y programas necesarios.

A.2. Dockerfile para el entrenamiento

En este caso, se creará una imagen de Docker de Pytorch y CUDA, que son dos librerías muy utilizadas para el ámbito del Aprendizaje Profundo, para después ir añadiendo el resto de las dependencias dentro de este contenedor. En la figura B.1 se puede observar el Dockerfile utilizado para la creación de la imagen.

```
FROM pytorch/pytorch:2.0.0-cuda11.7-cudnn8-runtime

# Downloads to user config dir
ADD https://ultralytics.com/assets/Arial.ttf
https://ultralytics.com/assets/Arial.Unicode.ttf
/root/.config/Ultralytics/

# Actualizo el sistema e instalo dependencias
RUN apt-get update \
    && apt install --no-install-recommends -y gcc git zip curl htop \
    libgl1-mesa-glx libglib2.0-0 libpython3-dev gnupg g++ python3-pip

# Instalo otras dependencias (necesario para la red)
RUN pip install opencv-python matplotlib thop pandas seaborn

# Actualizaciones de seguridad
RUN apt upgrade --no-install-recommends -y openssl tar

# Creo el directorio de trabajo
RUN mkdir -p /usr/src/ultralytics
WORKDIR /root

# Descargo el repositorio de ultralytics (librería de la red neuronal)
RUN git clone https://github.com/ultralytics/ultralytics
/root/ultralytics
# Copio los pesos de la red neuronal a utilizar (como quiero usar la red
s, m o l deberé cambiar el yolov8n.pt)
ADD
https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8m.pt
/root/ultralytics/model/

# Instalo clearml para monitorizar los resultados
RUN pip install clearml

# Creo un volumen con la carpeta para el entrenamiento
VOLUME root/TFG
```

Figura A.1: Dockerfile utilizado para crear la imagen de Docker

A.3. Creación del contenedor

Una vez creado el Dockerfile, se ejecuta el comando “docker build” (figura B.2) en nuestra consola de Ubuntu para crear la imagen de Docker que contendrá nuestro contenedor a raíz de este Dockerfile.

```
pedro@ubuntu:~$ docker build -t pedrop_tfg:v7 .
```

Figura A.2: Comando docker build

Una vez se ha creado la imagen de Docker, se crea un contenedor con esta imagen con el comando de la figura 3.3, que es donde se ejecutará todo el entrenamiento.

```
pedro@ubuntu:~$ docker run -it -v  
/home/pedrop/Desktop/TFG:/root/TFG -gpus "device=1"  
pedrop_tfg:v7 bash
```

Figura A.3: Comando docker run

En este contenedor se utiliza el campo -it para mantener el contenedor activo, aunque no esté ejecutando nada y poder entrar dentro al ejecutar. También implementa un volumen (-v /home/pedrop/Desktop/TFG:/root/TFG) para relacionar una carpeta interna del contenedor con una carpeta propia de mi ordenador, así cuando modifique algo en cualquiera de las carpetas también se modificará en la otra. Además, se selecciona la GPU que se va a utilizar (-gpus “device=1”) dependiendo de cual está disponible. Luego se introduce el nombre de la imagen con la que se quiere ejecutar el contenedor y su versión (pedrop_tfg:v7), y por último, se especifica el tipo de consola que se quiere utilizar, en este caso bash.

Una vez se han ejecutado los comandos de las figuras anteriores en consola, se entra en el contenedor y se puede empezar a entrenar la red neuronal.



UNIVERSIDAD
DE MÁLAGA



Bibliografía


- [1] Arango, D. A. J., Jaramillo Arango, D. A., & Montenegro, D. I. (2019). De la Inteligencia Artificial al juego de los dioses. *ComHumanitas: Revista Científica de Comunicación*, 10(3), 85–106. <https://doi.org/10.31207/rch.v10i3.210>
- [2] *Robots al rescate: Uso de la tecnología para mitigar los efectos de los desastres naturales / Naciones Unidas*. (n.d.). Retrieved June 21, 2023, from <https://www.un.org/es/robots-al-rescate-uso-de-la-tecnolog%C3%ADa-para-mitigar-los-efectos-de-los-desastres-naturales>
- [3] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., & Ng, A. (n.d.). *ROS: an open-source Robot Operating System*. Retrieved January 9, 2024, from <http://stair.stanford.edu>
- [4] Russell, Stuart J. (Stuart Jonathan), 1962-. (2010). *Artificial intelligence : a modern approach*. Upper Saddle River, N.J. :Prentice Hall
- [5] *¿Cómo es y cómo funciona nuestro cerebro?* (n.d.). Retrieved July 12, 2023, from <https://blog.fpmaragall.org/como-es-y-como-funciona-nuestro-cerebro>
- [6] *¿Cuáles son las partes del sistema nervioso? | NICHD Español*. (n.d.). Retrieved June 20, 2023, from <https://espanol.nichd.nih.gov/salud/temas/neuro/informacion/partes>
- [7] *El modelo de redes neuronales - Documentación de IBM*. (n.d.). Retrieved June 20, 2023, from <https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model>
- [8] *Perceptrón multicapa - Wikipedia, la enciclopedia libre*. (n.d.). Retrieved July 12, 2023, from https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa
- [9] *Aprendizaje supervisado y no supervisado | Blog UE*. (n.d.). Retrieved August 15, 2023, from <https://universidadeuropea.com/blog/aprendizaje-supervisado-no-supervisado/>
- [10] *Aprendizaje no supervisado - Wikipedia, la enciclopedia libre*. (n.d.). Retrieved August 19, 2023, from https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado
- [11] *Semi-Supervised Learning...el gran desconocido*. (n.d.). Retrieved August 19, 2023, from <https://telefonicatech.com/blog/semi-supervised-learningel-gran-desconocido>
- [12] *Aprendizaje por refuerzo y optimización - Expertos en IIC*. (n.d.). Retrieved August 19, 2023, from <https://www.iic.uam.es/inteligencia-artificial/aprendizaje-por-refuerzo/>
- [13] *Diferencias entre la Inteligencia Artificial y el Machine Learning | by Experiencia Oracle | Medium*. (n.d.). Retrieved August 19, 2023, from <https://medium.com/@experiencia18/diferencias-entre-la-inteligencia-artificial-y-el-machine-learning-f0448c503cd4>
- [14] *Tipos de Deep Learning: Todo lo que debes saber | Tokio School*. (n.d.). Retrieved August 22, 2023, from <https://www.tokioschool.com/noticias/tipos-deep-learning/>

- [15] *Redes Neuronales Generativas Adversarias (GANs)* - *IArtificial.net*. (n.d.). Retrieved August 22, 2023, from <https://www.iartificial.net/redes-neuronales-generativas-adversarias-gans/>
- [16] *Convolutional Neural Networks, Explained | by Mayank Mishra | Towards Data Science*. (n.d.). Retrieved October 3, 2023, from <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>
- [17] Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press
- [18] *¿Qué es la Función de Activación?* (n.d.). Retrieved November 7, 2023, from <https://aiofthings.telefonicatech.com/recursos/datapedia/funcion-activacion>
- [19] *Función de activación - Redes neuronales - Diego Calvo*. (n.d.). Retrieved November 7, 2023, from <https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>
- [20] *Cómo crear Red Convolutiva en Keras - Ander Fernández*. (n.d.). Retrieved November 9, 2023, from <https://anderfernandez.com/blog/que-es-una-red-neuronal-convolutiva-y-como-crearla-en-keras/>
- [21] *What is a neural network flatten layer?* (n.d.). Retrieved November 9, 2023, from <https://www.educative.io/answers/what-is-a-neural-network-flatten-layer>
- [22] *Convolutional Neural Networks (CNN): Step 3 - Flattening - Blogs - SuperDataScience | Machine Learning | AI | Data Science Career | Analytics | Success*. (n.d.). Retrieved November 9, 2023, from <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening>
- [23] *Fully Connected Layers in Convolutional Neural Networks – IndianTechWarrior*. (n.d.). Retrieved November 15, 2023, from <https://indiantechwarrior.com/fully-connected-layers-in-convolutional-neural-networks/>
- [24] Hosang, J., Benenson, R., & Schiele, B. (2017). Learning non-maximum suppression. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017-January*, 6469–6477. <https://doi.org/10.1109/CVPR.2017.685>
- [25] *An illustration of the dropout mechanism within the proposed CNN. (a)... | Download Scientific Diagram*. (n.d.). Retrieved November 24, 2023, from https://www.researchgate.net/figure/9-An-illustration-of-the-dropout-mechanism-within-the-proposed-CNN-a-Shows-a_fig23_317277576
- [26] *mAP (mean Average Precision) for Object Detection | by Jonathan Hui | Medium*. (n.d.). Retrieved November 30, 2023, from <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>
- [27] *ML | Underfitting and Overfitting - GeeksforGeeks*. (n.d.). Retrieved November 24, 2023, from <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>
- [28] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Computer Society Conference on*



Computer Vision and Pattern Recognition, 2016-December, 779–788.

<https://doi.org/10.1109/CVPR.2016.91>

- [29] *The History of YOLO Object Detection Models from YOLOv1 to YOLOv8* / Deci. (n.d.). Retrieved November 29, 2023, from <https://deci.ai/blog/history-yolo-object-detection-models-from-yolov1-yolov8/>
- [30] Chen, Y., Zhang, P., Li, Z., Li, Y., Zhang, X., Qi, L., Sun, J., & Jia, J. (n.d.). *Dynamic Scale Training for Object Detection*.
- [31] *YOLOv8 - Ultralytics YOLOv8 Docs*. (n.d.). Retrieved December 12, 2023, from <https://docs.ultralytics.com/models/yolov8/>
- [32] *ultralytics/ultralytics: NEW - YOLOv8*  in PyTorch > ONNX > OpenVINO > CoreML > TFLite. (n.d.). Retrieved December 14, 2023, from <https://github.com/ultralytics/ultralytics?tab=readme-ov-file>
- [33] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, Architecture, and Uses In The Wild. *Science Robotics*, 7(66). <https://doi.org/10.1126/scirobotics.abm6074>
- [34] *Docker Docs*. (n.d.). Retrieved January 11, 2024, from <https://docs.docker.com/>
- [35] *labelImg · PyPI*. (n.d.). Retrieved January 3, 2024, from <https://pypi.org/project/labelImg/>
- [36] *mgonzs13/yolov8_ros: Ultralytics YOLOv8 for ROS 2*. (n.d.). Retrieved January 7, 2024, from https://github.com/mgonzs13/yolov8_ros
- [36] *(Peanpepu/Neural-Networks-Fusion, n.d.)*. Retrieved January 12, 2024 from <https://github.com/Peanpepu/Neural-Networks-Fusion>
- [37] *Docker Hub Container Image Library | App Containerization*. (n.d.). Retrieved January 11, 2024, from <https://hub.docker.com/>