



UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA



ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Detección de Áreas de Desastre desde Imágenes Aéreas

Detection of Disaster Areas from Aerial Images

GRADO DE INGENIERÍA ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA

Departamento de Ingeniería de Sistemas y Automática

Realizado por:

Rubén González Navarro

Tutorizado por:

Dr. D. Ricardo Vázquez Martín

Cotutorizado por:

D. Dahui Lin

22 Febrero 2022



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Resumen

En el presente documento se desarrolla un sistema de detección de zonas catastróficas desde un Vehículo Aéreo No Tripulado en entornos de desastre o catástrofes, en concreto, zonas de escombros, incendios e inundadas.

Este objetivo se logra mediante técnicas de aprendizaje profundo, por lo que previo al trabajo se hace un estudio sobre redes neuronales y de la arquitectura YOLOv5, además de un estudio acerca de detección de objetos. Se muestran también los hiperparámetros con los que trabaja esta arquitectura y una breve introducción sobre ella.

Posteriormente, se presentan los dataset originales de los cuales se han obtenido el conjunto de datos para este caso de estudio, las técnicas y herramientas utilizadas para el aumento de datos y el etiquetado de las imágenes.

A continuación, se describen los procesos seguidos a fin de llegar al resultado final. Este proyecto se dividió en dos fases distintas. Una primera fase donde se procede a adaptar la red a la visión aérea y una segunda fase donde habiendo ya adaptado la red a la visión aérea, se volverá a adaptar a los entornos de desastre para nuestro caso de estudio. Dichas fases se basan en la técnica del aprendizaje profundo de la Transferencia de Conocimiento o *Transfer Learning*.

Por último, se estudian todos los resultados de los entrenamientos realizados, analizando todas las métricas e imágenes usadas para poner a prueba la red. Se estudian varios escenarios posibles en el entorno en el que va a trabajar el modelo y se comparan los diferentes resultados.

Palabras Clave

Deep Learning, Red Neuronal Convolucional, Transferencia de Aprendizaje, Detección, YOLOv5, Imágenes Aéreas, Desastre, Dron.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Abstract

This paper develops a system for detection of disaster areas from an Unmanned Aerial Vehicle in disaster environments, specifically, debris, fire and flooded areas.

This objective is achieved through deep learning techniques, so a study on neural networks and YOLOv5 architecture is carried out prior to the work, as well as a study on object detection. It is also shown the hyperparameters with which this architecture works and a short introduction about them.

Subsequently, the original datasets from which the dataset for this case of study was obtained, the techniques and tools used for data augmentation and image labeling are presented.

The processes followed in order to arrive at the final result are described as well. This project was divided into two different phases. A first phase where the network is adapted to the aerial vision and a second phase where the network has already been adapted to the aerial vision, it will be adapted to the disaster environments for our case study. These phases are based on the deep learning technique of Transfer Learning.

Finally, all the training results are studied, analyzing all the metrics and images used to test the network. Several possible scenarios are studied in the environment in which the model will work and the different results are compared.

KeyWords

Deep Learning, Convolutional Neural Network, Transfer Learning, Detection, YOLOv5, Aerial Images, Disaster, Drone.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Declaración de originalidad

Yo, Rubén González Navarro, con DNI 77493894V, 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 “Detección de Áreas de Desastre desde Imágenes Aéreas“ 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 la presente en

Málaga, a 5 de Septiembre de 2022.

Rubén González Navarro



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Índice general

Resumen	3
Abstract	5
Declaración de originalidad	7
1. Antecedentes y Objeto.	17
1.1. Antecedentes	17
1.2. Objeto	17
1.3. Plan de Trabajo	17
1.4. Áreas de Conocimiento	18
1.5. Herramientas del proyecto	18
1.6. Estructura de la Memoria	18
2. Fundamentos Teóricos	21
2.1. Introducción	21
2.1.1. ¿Qué es la Inteligencia Artificial?	21
2.1.2. Analogía de la Inteligencia Artificial con el cerebro humano	21
2.1.3. ¿Qué es el Machine Learning?	22
2.1.4. ¿Qué es el Deep Learning?	23
2.2. Redes Neuronales en Deep Learning	24
2.2.1. Redes Neuronales Artificiales	24
2.2.2. Redes Neuronales Convolucionales	25
2.2.3. Entrenamiento de una RNC	30
2.2.4. Transferencia de Aprendizaje	34
2.2.5. Aumento de Datos	34
2.3. Fudamentos de YOLOv5	34
2.3.1. Introducción	34
2.3.2. Arquitectura YOLOv5	35
2.3.3. Algoritmo YOLO	36
2.3.4. Hiperparámetros YOLOv5	37
2.3.5. Modelos YOLOv5	38
3. Entorno de Trabajo	41
3.1. Google Colab	41
3.1.1. YOLOv5	41
3.2. NVIDIA DGX	42
3.3. PyTorch	42
3.4. Dependencias YOLOv5	43
3.4.1. Matplotlib	43
3.4.2. NumPy	43
3.4.3. OpenCV	43
3.4.4. Pillow	43
3.4.5. PyYAML	43



3.4.6. SciPy	43
3.4.7. Torch	43
3.4.8. Torch Vision	44
3.4.9. Tqdm	44
3.4.10. Tensorboard	44
3.4.11. Seaborn	44
3.4.12. Pandas	44
3.4.13. Thop	44
3.4.14. Request	44
3.4.15. Relación y jerarquización de los módulos	44
3.5. Contenedores	45
4. Obtención del Dataset	49
4.1. Adquisición del Dataset	49
4.2. VisDrone	49
4.2.1. Formato del etiquetado	50
4.3. AIDER	50
4.4. UMA-SAR	51
4.5. Etiquetado de imágenes	52
4.5.1. Herramientas de Etiquetado	52
4.5.2. Aumento del Dataset	54
5. Fases de Entrenamiento	57
5.1. Introducción	57
5.2. Métricas	57
5.2.1. Precisión	57
5.2.2. Sensibilidad	58
5.2.3. Curva PR	58
5.2.4. Mean Average Precision (mAP)	58
5.3. Primera Fase	58
5.3.1. Introducción	58
5.3.2. Primeros Entrenamientos	60
5.3.3. Entrenamiento Final	61
5.4. Segunda Fase	63
5.4.1. Introducción	63
5.4.2. Entrenamientos Sin Aumento de Datos	63
5.4.3. Entrenamientos Con Aumento de Datos	64
6. Resultados	67
6.1. Resultados de la Primera Fase	67
6.2. Resultados de la Segunda Fase	74
7. Conclusiones y Trabajo Futuro	83
7.1. Conclusiones	83
7.2. Trabajo Futuro	84
A. DOCKER	85
A.1. Introducción	85
A.2. Arquitectura DOCKER	85
A.3. Formato de un DockerFile	86
A.4. Comando EXPOSE	87
A.5. Docker Images	87
A.6. Contenedores	88
A.6.1. Mapeo de puertos en los contenedores	89
A.6.2. Borrar contenedores	89
A.7. Networking	90



A.8. Almacenamiento	90
A.8.1. ¿Cómo añadir un volumen a contenedor existente?	91
A.8.2. Aumentar el Tamaño del Contenedor	92
B. Códigos	93
B.1. Convertir tipo de Archivo	93
B.2. Renombrar Archivos	94
B.3. Aumento de Datos	95
C. Herramientas de Etiquetado y Edición	99
C.1. Herramientas de Etiquetado	99
C.2. Herramienta de Edición	100



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Índice de figuras

2.1. Neurona Humana. Fuente: Sebastian Raschka y Vahid Mirjalili. Python Machine Learning.	22
2.2. Perceptrón Simple. Fuente: Ulises Castro Peñaloza. Introducción a los Sistemas Inteligentes.	22
2.3. Deep Learning como subcampo de la Inteligencia Artificial. Fuente: Raul E. Lopez Briega. Introducción al Deep Learning. IAAR Capacitación.	23
2.4. Curva de crecimiento del <i>Deep Learning</i> .	24
2.5. Red Neuronal Artificial de una capa. Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.	25
2.6. Red Neuronal Artificial Multicapa. Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.	25
2.7. Representación gráfica de la Función Sigmóide.	27
2.8. Representación gráfica de la Función Tangente Hiperbólica.	27
2.9. Representación gráfica de la Función Softmax.	28
2.10. Representación gráfica de la Función Softplus y ReLu.	28
2.11. Ejemplo de Max-Pooling y Average-Pooling. Fuente: Dingjun Yu. Mixed Pooling for Convolutional Neural Networks	29
2.12. Capa <i>Flatten</i> . Fuente: Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.	29
2.13. Capa <i>Fully Connected</i> . Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.	30
2.14. Overfitting y Underfitting. Fuente: Ian Goodfellow. Deep Learning.	33
2.15. Dropout. Fuente: Ian Goodfellow. Deep Learning.	33
2.16. Arquitectura YOLOv5. Fuente: Iason Katsamenis. TraCon: A novel dataset for real-time traffic cones detection using deep learning.	35
2.17. Intersección Sobre la Unión. Fuente: Do Thuan. “Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm”.	36
2.18. Algoritmo YOLO. Fuete: Do Thuan. “Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm”.	37
2.19. Modelos YOLOv5. Fuente: Repositorio GitHub ultrlytics.	39
2.20. Gráfica Comparativa Modelos YOLOv5. Fuente: Repositorio GitHub ultrlytics.	39
3.1. Jerarquía de los módulos de YOLOv5.	45
3.2. Creación del contenedor.	45
3.3. Exploración del contenedor.	46
3.4. Acceso al contenedor.	47
3.5. Listar características del contenedor.	47
4.1. Imágenes de Ejemplo del dataset VisDrone. Fuente: PRCV 2022. Aerial-Ground Intelligent Unmanned System Environment Perception Challenge.	49
4.2. Formato etiquetas VisDrone.	50
4.3. Imágenes de Ejemplo del dataset AIDER. Fuente: Christos Kyrkou. AIDER: Aerial Image Database for Emergency Response applications.	51



4.4. Imágenes de Ejemplo del dataset UMA-SAR. Fuente: Jesus Morales, Ricardo Vázquez-Martín, Anthony Mandow, David Morilla-Cabello y Alfonso García Cerezo. "The UMA-SAR Dataset: Multimodal Data Collection from a Ground Vehicle During Outdoor Disaster Response Training Exercises.	51
4.5. Imagen de ejemplo de las clases a detectar.	53
4.6. Organización de los directorios de las imágenes y las etiquetas.	53
4.7. Archivo <i>.txt</i> de las etiquetas.	54
4.8. Fichero <i>customdata.yaml</i>	54
4.9. Organización de los directorios de las imágenes y las etiquetas. Fuente: Do Thuan. "Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm".	55
4.10. Imágenes y sus transformaciones	56
5.1. Ejemplo curvas PR.	58
5.2. Fichero <i>VisDrone.yaml</i>	59
5.3. Listar características del contenedor.	60
5.4. Listar características del contenedor.	62
5.5. Entrenamiento con optimizador SGD sin aumento de datos.	64
5.6. Entrenamiento con optimizador Adam sin aumento de datos.	64
5.7. Entrenamiento con optimizador SGD con aumento de datos.	64
5.8. Entrenamiento con optimizar Adam con aumento de datos.	64
6.1. Gráfica mAP@0.5 de los Primeros Entrenamientos en NVIDIA DGX.	68
6.2. Curva PR del Entrenamiento Final.	69
6.3. Matriz de Confusión del Entrenamiento Final de la Primera Fase.	69
6.4. Gráfica de la Cantidad de Etiquetas de cada Clase del Conjunto de Entrenamiento VisDrone.	70
6.5. Detección de las imágenes de test de VisDrone.	70
6.6. Imágenes Diurnas VisDrone.	71
6.7. Imágenes Nocturnas VisDrone.	72
6.8. Imágenes Distorsionadas VisDrone.	73
6.9. Gráficas Comparativas de Entrenamientos sin Aumento de Datos	75
6.10. Comparativa de Cantidad de Etiquetas	76
6.11. Curva PR de la Red Neuronal Convolutacional Final.	77
6.12. Matriz de Confusión de la Red Neuronal Convolutacional Final.	77
6.13. Comparativa de Detección	78
6.14. Detección de las imágenes de test.	78
6.15. Imágenes de test UMA-SAR.	79
6.16. Imágenes de test UMA-SAR.	80
B.1. Código para Convertir a otros Tipos de Imágenes.	93
B.2. Código para Renombrar Imágenes.	94
B.3. Código para aplicar el Aumento de Datos.	97
C.1. Ejecutar la aplicación LabelImg.	99
C.2. Interfaz Gráfica de LabelImg.	99
C.3. Ejemplo de Etiquetas con LabelImg.	100

Índice de tablas

5.1.	Primer Entrenamiento en Google Colab	60
5.2.	Primer Entrenamiento en NVIDIA DGX	61
5.3.	Tabla Comparativa de Hiperparámetros	62
5.4.	Entrenamiento Final en NVIDIA DGX	63
5.5.	Parámetros de entrenamiento con optimizador SGD sin aumento de datos.	65
5.6.	Parámetros de entrenamiento con optimizador SGD con aumento de datos.	65
6.1.	Primer Resultado del Entrenamiento en Google Colab	67
6.2.	Resultado Final del Entrenamiento	68
6.3.	Comparativa de entrenamientos sin aumento de datos	74
6.4.	Comparativa de entrenamientos con aumento de datos	76



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Capítulo 1

Antecedentes y Objeto.

1.1. Antecedentes

La robótica de rescate lleva utilizándose durante mucho tiempo, pero en las últimas décadas gracias a los avances tecnológicos y al aumento de la fiabilidad de los componentes, esta rama de la robótica ha sufrido un gran auge.

Llevamos conviviendo con los desastres naturales desde el principio de la existencia de los primeros seres humanos. Son procesos que forman parte del ciclo natural de nuestro planeta, tales como pueden ser los terremotos, las avalanchas, los tsunamis, los incendios, ... Los robots de rescate están diseñados especialmente para ayudar en la búsqueda y el rescate de personas tras un desastre.

El uso de los robots de rescate, llega a reducir tanto los tiempos de búsqueda de supervivientes en entornos de desastre, donde cada segundo es crucial, como para reducir la exposición al peligro de las autoridades de rescate. Es por ello que se empiezan a implantar en estos robots mejoras para imitar la percepción humana y que a ser posible supere nuestra sensibilidad [1]. Nos podemos encontrar con entornos donde la vista del ser humano no es capaz de llegar más que a unos escasos metros de distancia. Esto se puede deber a encontrar mucho humo y polvo en suspensión o simplemente porque haya gran cantidad de escombros por toda la superficie.

1.2. Objeto

Es aquí donde entra la motivación de este proyecto. Se empiezan a implementar sistemas de Visión por Computador y técnicas de *Deep Learning* [2] a fin de poder realizar un análisis de las imágenes tomadas por estos robots, incluso a realizar este análisis a tiempo real.

Se propone emplear una red neuronal convolucional para detectar clases de interés tal y como podrían ser diferentes tipos de zonas afectadas por catástrofes naturales grabadas por un dispositivo aéreo no tripulado (comúnmente conocido como UAV por sus siglas en inglés, *Unmanned Aerial Vehicle*). Se utilizará una arquitectura de una red pre-entrenada, ya existente, se adaptará esta arquitectura a nuestro problema de estudio y se aplicaran técnicas de *Deep Learning* a fin de mejorar los resultados y por último se realizará una evaluación de los resultados obtenidos.

1.3. Plan de Trabajo

El desarrollo del trabajo se realizará en 7 partes distintas:

1. Estudio bibliográfico inicial [3] [4].
2. Selección de herramientas: lenguajes, *frameworks* y librerías. Selección de la arquitectura.
3. Búsqueda de *datasets* para el entrenamiento de la red.



4. Etiquetado de imágenes del *dataset*.
5. Entrenamiento de la red neuronal.
6. Validación de Resultados.
7. Documentación.

1.4. Áreas de Conocimiento

Las áreas de conocimiento en las que se desarrolla el trabajo son las siguientes:

- Ingeniería de Sistemas y Automática.
- Ciencias de la Computación e Inteligencia Artificial.
- Arquitectura y Tecnología de Computadores.

1.5. Herramientas del proyecto

Dado el caso en el que se desee replicar este proyecto, a continuación se proporciona un listado con todo el software y el hardware utilizado.

- Hardware
 - Ordenador Portátil con las siguientes características:
 - Sistema operativo Windows 10 64 bits.
 - Procesador Intel(R) Core(TM) i7-7500U CPU 2.70GHz.
 - Memoria RAM de 8,00 GB.
 - Estación de entrenamiento NVIDIA DGX:
 - Sistema operativo Linux dgxstation version 4.15.0-62-generic.
 - Cuatro tarjetas gráficas Tesla V100-DGXS 32GB.
 - Tableta Gráfica Wacom Cintiq 16.
- Software
 - Entorno virtual Anaconda3 versión 4.11.0 (Python 3.9.13).
 - Spyder proporcionado por Anaconda3.
 - Docker v19.3.4.
 - lbleimg.

1.6. Estructura de la Memoria

La memoria se organiza en seis capítulos y cuatro apéndices. A continuación se describe brevemente el contenido de cada uno de ellos:

- **Capítulo 2:** Fundamentos Teóricos. Se comenzará con una breve introducción a la inteligencia artificial, seguida de una introducción al *Deep Learning* y a las Redes Neuronales Convolucionales, donde por último nos centraremos en YOLOv5.
- **Capítulo 3:** Entorno de Trabajo. Descripción de los diferentes entornos de trabajo que se han empleado durante el desarrollo de este TFG.
- **Capítulo 4:** Datasets. Explicación de los distintos conjuntos de imágenes empleados durante los entrenamientos de la red neuronal.



- **Capítulo 5:** Fases de Entrenamiento. Desarrollo de todo el proceso de trabajo y obtención de resultados.
- **Capítulo 6:** Resultados.
- **Capítulo 7:** Conclusiones y Trabajo Futuro.
- **Anexo A:** Docker.
- **Anexo B:** Códigos.
- **Anexo C:** Herramientas de Etiquetado y edición.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Capítulo 2

Fundamentos Teóricos

2.1. Introducción

2.1.1. ¿Qué es la Inteligencia Artificial?

La Real Academia Española define a la inteligencia artificial como una "Disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico".

En los comienzos de la Inteligencia Artificial, esta se usó para resolver problemas que eran prácticamente irresolubles por los seres humanos, pero algo muy sencillo para las máquinas. Estos eran descritos por unas reglas formales y matemáticas. Sin embargo, el verdadero problema de la Inteligencia Artificial viene cuando se le plantean problemas, como los que nos puede parecer algo tan simple, como reconocer la expresión de una persona [2]. Esto es algo que a nosotros nos resulta intuitivo o automático, pero que para una máquina no lo es.

Actualmente la Inteligencia Artificial se aplica en muchos campos, campos como pueden ser el de la medicina para hacer diagnósticos, el de las finanzas para temas de administración e inversiones, los servicios de atención al cliente para resolver consultas rápidas de clientes y especialmente el campo de la robótica en el que iremos profundizando más adelante.

Podemos diferenciar dos enfoques predominantes, el enfoque simbólico que hace uso de la lógica matemática y el enfoque no simbólico que hace uso de algoritmos predictivos. En concreto, será este último en el que nos centraremos, ya que es en este tipo de enfoque donde se encuentra situado el Aprendizaje Profundo o *Deep Learning*.

2.1.2. Analogía de la Inteligencia Artificial con el cerebro humano

El cerebro humano está formado por miles de millones de conexiones entre neuronas. La Inteligencia Artificial lo que trata de hacer es imitar estas conexiones en las conocidas como Redes Neuronales [5].

Las neuronas son células interconectadas en el cerebro donde se producen los procesamientos y transmisiones de señales eléctricas y químicas. Una neurona es descrita como una simple puerta lógica con salidas binarias [6]. No obstante, hay que tener en cuenta que una puerta lógica puede llegar a ser 5 ó 6 órdenes de magnitud más rápida que una neurona. De media, una neurona puede tardar en procesar información en el orden de los milisegundos, mientras que una puerta lógica tarda en el orden de los nanosegundos.

Múltiples señales de entrada que llegan a las dendritas son integradas en el cuerpo de la célula y si la señal acumulada supera un determinado umbral, se genera una señal de salida que se transmite a los axones terminales por el axón [Véase Figura 2.1].

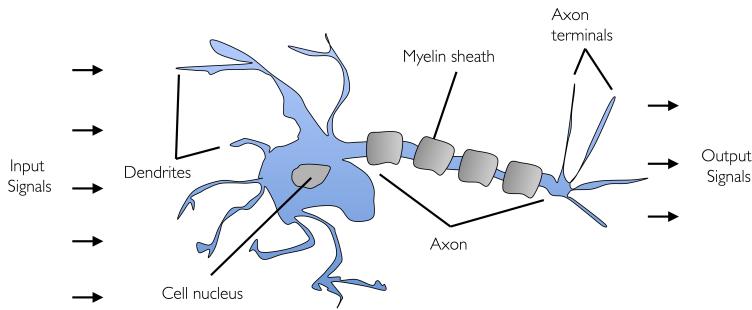


Figura 2.1: Neurona Humana. Fuente: Sebastian Raschka y Vahid Mirjalili. Python Machine Learning.

Más tarde, este modelo da lugar al desarrollo de las neuronas artificiales o también llamado perceptrón. El cual es un algoritmo matemático que aprenderá automáticamente los pesos óptimos por los que será multiplicadas las características de entrada de cara a tomar una decisión [Véase Figura 2.2]. El perceptrón trata de imitar el comportamiento de una neurona, tratando de modelar las características más importantes de ésta [7].

En el contexto del Aprendizaje Supervisado y clasificación se usa para predecir si una muestra pertenece a una clase u otra.

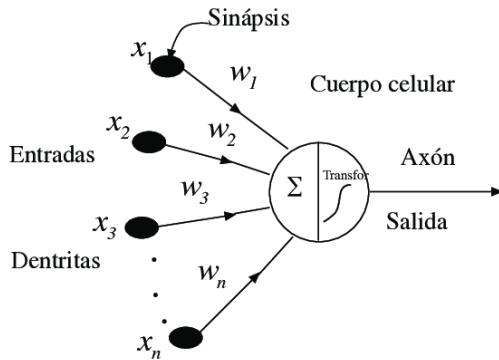


Figura 2.2: Perceptrón Simple. Fuente: Ulises Castro Peñaloza. Introducción a los Sistemas Inteligentes.

Tal y como se puede ver en la Figura 2.2 se pueden destacar cinco partes principales en el modelo del perceptrón:

- Entradas: Representadas con la letra x_i . Denotan los valores principales de las características que se recogen.
- Pesos: Representados con la letra w_j . Son valores adaptativos, que cuantifican la importancia de la señal procedente de la entrada.
- Suma Ponderada: Adición ponderada de todas las entradas $x_i w_j$.
- Función de Activación: Función matemática que devolverá un único valor y que ayudará a predecir el resultado final.
- Salida: Es el valor obtenido después de todos los cálculos.

2.1.3. ¿Qué es el Machine Learning?

Cuando se habla de *Machine Learning* o Aprendizaje Automático, se refieren a aquellos sistemas informáticos que son capaces de aprender sin la intervención del ser humano. Al hablar de que un sistema

aprende, nos referimos a éste es capaz de identificar patrones complejos en los datos analizados mediante algoritmos de aprendizaje. Podemos clasificar estos últimos en tres grandes clases:

- Aprendizaje Supervisado: Son aquellos algoritmos que utilizan unos datos de entrenamiento etiquetados, para procesarlos, realizar predicciones con éstos y corregir aquellas que son erróneas. El entrenamiento terminará cuando se haya alcanzado el nivel de precisión deseado [8].
- Aprendizaje No Supervisado: Cuando el conjunto de datos de entrenamiento no se encuentra etiquetado y no se tiene un resultado conocido, los algoritmos deben deducir las estructuras presentes en los datos de entrada [8].
- Aprendizaje Semi-Supervisado: En estos tipos de algoritmos de aprendizaje, se combinan tanto datos etiquetados como no etiquetados para generar una función deseada o clasificador. Este tipo de modelos deben aprender las estructuras para organizar los datos así como también realizar predicciones [8].
- Aprendizaje por Refuerzo: El sistema debe perseguir un objetivo e irá aprendiendo a medida que se va explorando el entorno con el que interactúa, el cual no se conoce de antemano, evitando las acciones con recompensa negativa e intentando llevar a cabo acciones con recompensa positiva. [9]

Concretamente, en el caso de estudio de este proyecto se va a hacer uso de algoritmos de aprendizaje supervisado. Se tomarán imágenes, donde ya estarán los objetos de interés previamente etiquetados. El sistema irá aprendiendo a reconocer las principales características de estos objetos que se desean detectar. Acabado el proceso de entrenamiento, se procederá a verificar que la precisión obtenida en el entrenamiento es suficiente para nuestro caso de estudio.

2.1.4. ¿Qué es el Deep Learning?

El *Deep Learning* o Aprendizaje Profundo es un campo dentro del *Machine Learning*, que a su vez, éste es otro campo dentro de la Inteligencia Artificial, tal y como se representa en la figura 2.3.

El aprendizaje profundo puede aprender a realizar tareas de clasificación directamente desde imágenes, texto o sonido. Los modelos se entranan utilizando un gran conjunto de datos etiquetados y arquitecturas de redes neuronales que contienen muchas capas.

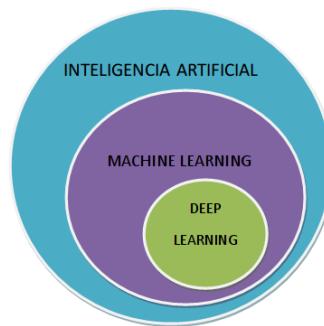


Figura 2.3: Deep Learning como subcampo de la Inteligencia Artificial. Fuente: Raul E. Lopez Briega. Introducción al Deep Learning. IAAR Capacitación.

Los modelos de aprendizaje profundo pueden llegar a alcanzar una precisión mucho más alta que la de los humanos. Aunque el modelo del perceptrón simple se desarrolló en la década de los 60s, no fue hasta la época de los 80s cuando el *Deep Learning* vió la luz. Sin embargo, no ha sido hasta la época moderna cuando el uso del aprendizaje profundo ha crecido exponencialmente, ver Figura 2.4. Esto ha sido gracias a la evolución de la tecnología, en especial al desarrollo de las GPUs (*Graphics Processing Unit*) que ayudan a suplir la demanda de potencia de cálculo de los algoritmos de aprendizaje profundo.

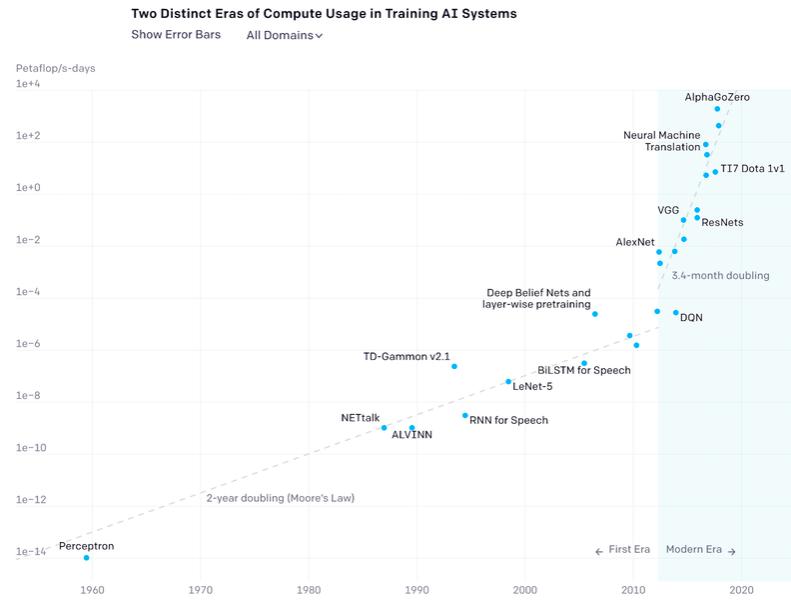


Figura 2.4: Curva de crecimiento del *Deep Learning*.

No obstante, a día de hoy las GPUs siguen evolucionando y están naciendo nuevas técnicas de entrenamiento como puede ser el *Cloud Computing* y nuevos componentes electrónicos como las TPUs (*Tensor Processing Units*) que permite a los equipos de desarrollo reducir el tiempo de entrenamiento para una red de aprendizaje profundo de semanas a horas o menos.

2.2. Redes Neuronales en Deep Learning

Cuando hablamos de las distintas arquitecturas de las redes neuronales de aprendizaje profundo, podemos dividirlas según dos categorías.

Por un lado tenemos las arquitecturas de Aprendizaje Supervisado, donde nos encontramos las Redes Neuronales Convolucionales (RNC), usadas con imágenes y las Redes Neuronales Recurrentes (RNR), utilizadas para el reconocimiento de audio, del habla y de la escritura.

2.2.1. Redes Neuronales Artificiales

Se basan en la analogía que existe en el comportamiento y funcionamiento del cerebro humano, tal y como se comentó en la Sección 2.1.2. Su arquitectura está basada en la interconexión de numerosos perceptrones tratando de imitar el sistema nervioso humano. Dicha arquitectura tendrá una propiedad adaptativa capaz de ajustar los pesos de todas las neuronas.

La característica más importante de las redes neuronales artificiales es su capacidad para aprender de un gran volumen de conjuntos de datos, es decir, pueden encontrar un modelo que se ajuste a dichos datos. El propósito del algoritmo de aprendizaje es ajustar los pesos w_j de la red para que la salida producida por la RNA sea lo más cercana posible a la salida real de una entrada dada.

Podemos encontrar dos tipos de arquitecturas, de una capa o multicapa. Las RNAs de una capa tendremos que entre las entradas y la salida tendremos una capa llamada la capa oculta compuesta por neuronas que serán las encargadas de conectar los valores de entrada con los de salida [Véase Figura 2.5].

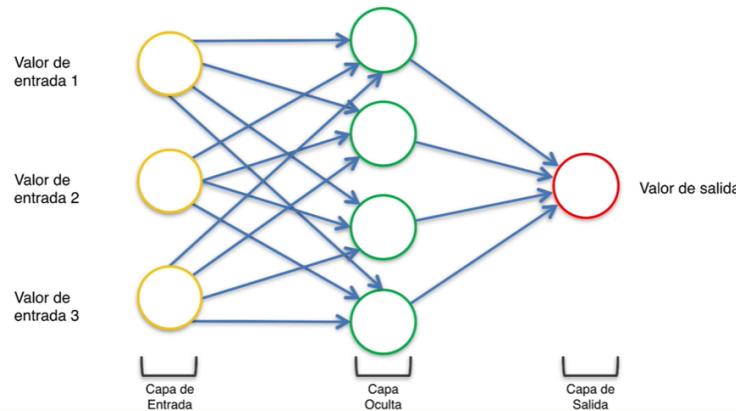


Figura 2.5: Red Neuronal Artificial de una capa. Fuente: Ligenicy I Team. Deep Learning de A a Z:redes neuronales en Python desde cero.

Sin embargo, por lo general, la arquitectura típica que nos vamos a encontrar es la multicapa. Aunque tienen una estructura más compleja, al ser más grandes ofrecen mejores prestaciones en el cálculo computacional que las redes de una sola capa, las cuales son más simples. Las redes multicapa se forman con un grupo de capas ocultas en cascada. La salida de una capa es la entrada de la siguiente capa. [Véase Figura 2.6].

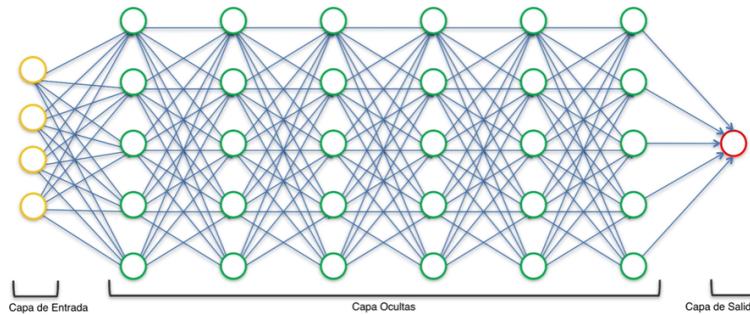


Figura 2.6: Red Neuronal Artificial Multicapa. Fuente: Ligenicy I Team. Deep Learning de A a Z:redes neuronales en Python desde cero.

2.2.2. Redes Neuronales Convolucionales

Introducción a las RNCs

Este tipo de redes neuronales empiezan a ser necesarias cuando surge la necesidad de realizar una análisis de imágenes o vídeos que con algoritmos convencionales de visión por computador pueden llegar a ser muy complejos. El propósito de las Redes Neuronales Convolucionales es extraer todas las características de la imagen y luego usar tales funciones para detectar o clasificar objetos en imágenes.

Al igual que en las RNAs se trataba de imitar las conexiones en el cerebro humano, las RNCs van a intentar es reproducir el conexionado de las neuronas en la corteza visual primaria del cerebro.

Internamente están formadas por una sucesión de capas convolucionales, funciones de activación, capas de *pooling*, capas ocultas o densas, etc. Todo esto en su conjunto, dan lugar a una arquitectura que permite extraer las características deseadas de la imagen de entrada y llevar a cabo a partir de éstas, diferentes tareas como clasificación, detección, regresión o segmentación.

En el contexto de este Trabajo de Fin de Grado, tendremos una tarea de detección, ya que lo que se



va a tratar de hacer es obtener como salida de la red una parte dentro de una imagen donde haya una zona catastrófica.

Capas Convolucionales

Antes de empezar a definir lo que es una capa convolucional dentro de una RNC, se va a definir lo que es la operación de convolución.

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 . Esta es muy común en campos de Procesamiento de Señales o Electrónica. Esta operación se puede denotar también con un asterisco. En la terminología de las redes convolucionales, la primera función de la convolución suele denominarse entrada o *input*, y la segunda función, *kernel*. La salida se denomina a veces mapa de características o *feature map* [2]. Por lo tanto, dada dos funciones llamadas f y g , la expresión vendrá dada por la Ecuación 2.1.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau)g(t - \tau) dt \quad (2.1)$$

En el ámbito de este proyecto, del *machine learning*, se trabaja con arrays multidimensionales como entrada, al igual que el *kernel*. Normalmente, utilizamos convoluciones sobre más de un eje a la vez. Por ejemplo, con una imagen I , que se define como un array bidimensional como entrada, probablemente también querremos utilizar un *kernel* bidimensional K , esto viene dado por la Ecuación 2.2.

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n) \quad (2.2)$$

Las capas de las redes neuronales tradicionales utilizan la multiplicación matricial por una matriz de parámetros con un parámetro separado que describe la interacción entre cada unidad de entrada y de salida. Esto significa que cada unidad de salida interactúa con cada unidad de entrada.

Se tendrán detectores de rasgos, también llamados núcleos de convolución o filtros. Que se irán pasando por la imagen para detectar ciertas características. Cuando se junta la imagen de entrada con el operador \otimes lo que sucede a nivel intuitivo es que se promedian los valores del filtro y de la imagen y en el mapa de características se pone dicho resultado. Se irá moviendo de forma inteligente el filtro por toda la imagen elemento a elemento. El mapa de características lo que nos permitirá será obtener una versión simplificada de la imagen tras pasar el filtro. Al haberse aplicado esto, se ha reducido el tamaño de la imagen. Esto se conoce como *stride*. Cuando se aplica un filtro se pierde información de la imagen, porque se tiene menos valores y menos píxeles, pero el objetivo del detector de características es obtener ciertas características, ciertos rasgos de la imagen global. El filtro puede detectar características pequeñas y significativas, como los bordes, con núcleos que ocupan sólo decenas o cientos de píxeles. A pesar de que la imagen ha sido reducida se ha filtrado o reducido para ver el valor ponderado de todos los vecinos de un determinado pixel.

Si se aplican muchos mapas de características diferentes podemos tener versiones convolucionadas de la imagen original a través de usar diferentes filtros y tener una colección de mapas de características. Esto es lo que se conoce como capa de convolución, donde cada filtro aplicado a la imagen buscará ciertas características diferentes y la red neuronal se encargará de utilizarlas para poder hacer una operación que las junte todas en la siguiente capa.

Función de Activación

Se define como un elemento propio de las redes neuronales que suele situarse a continuación de cada capa convolucional. La finalidad de añadir esta función es introducir una no-linealidad al sistema para poder tener una mayor versatilidad a nuestra arquitectura. Se usan muchos tipos de funciones y cada una de ellas proporcionan activaciones de diferentes características y se utilizan en unos casos determinados.

Sigmoide Es una función de activación no lineal que se utiliza sobre todo en las redes neuronales *feedforward* [10] cuya expresión viene dada por la Ecuación 2.3 y representada en la Figura 2.7. Se puede asimilar a un función escalón, satura en 1 para valores positivos altos y en 0 para valores negativos bajos,

de forma que "comprime" los valores de la entrada en un intervalo, no permitiendo activaciones negativas ni mayores a 1.

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

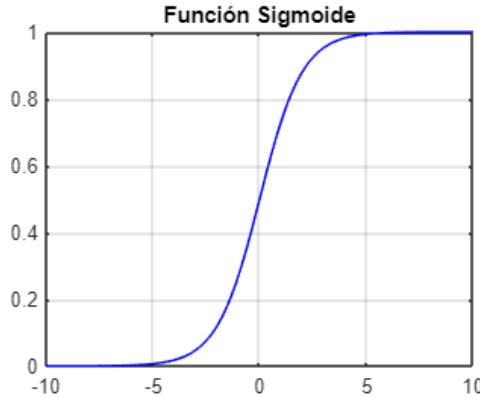


Figura 2.7: Representación gráfica de la Función Sísmoide.

Tangente Hiperbólica Es una función más suave de centro cero cuyo rango está entre -1 y 1, por lo que la salida de la función tangente hiperbólica viene dada por la Ecuación 2.4 y representada en la Figura 2.8. Esta función se suele preferir a la anterior, ya que ofrece un mejor rendimiento de entrenamiento o de las redes neuronales multicapa. La principal ventaja de esta función es que produce una salida centrada en cero, lo que ayuda a la *backpropagation* [10].

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

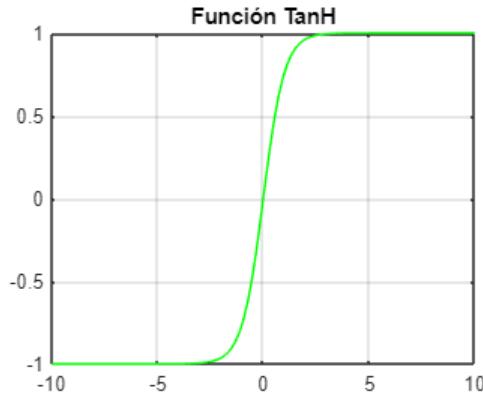


Figura 2.8: Representación gráfica de la Función Tangente Hiperbólica.

Softmax Se utiliza para calcular la distribución de probabilidad de un vector de números reales [10]. Se considera como una generalización de la función Sísmoide, que se usa como una distribución de probabilidad sobre una variable binaria [2]. Esta función se usa a la salida de un clasificador para representar la probabilidad sobre n clases diferentes. Formalmente esta función viene dada por la Ecuación 2.5 y representada en la Figura 2.9 para un vector de valores $\vec{v} = [-8, -1, 0, 2, 6, 9]$.

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (2.5)$$

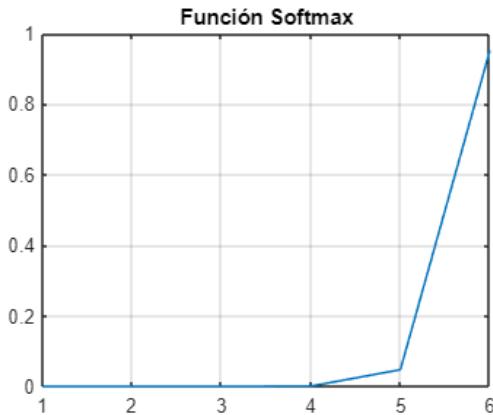


Figura 2.9: Representación gráfica de la Función Softmax.

Además de estas funciones nos encontramos con otro grupo de funciones conocidas como rectificadas, esto implica una transformación de una curva en un segmento de igual longitud. Nos podemos encontrar con *ReLU*, *LReLU*, *PReLU*, *RReLU*, *SReLU*, ... [10] En este caso, nos centramos en la primera que es la más usada.

ReLU Es la función de activación más rápida y permite ser optimizada muy fácilmente ya que se parece mucho a una unidad lineal. Esta función realiza una operación de umbral a cada elemento de entrada en la que los valores inferiores a cero se ponen a cero y que viene dada por la Ecuación 2.6.

$$f(x) = \max(0, x) = \begin{cases} x_i & \text{si } x_i \geq 0 \\ 0 & \text{si } x_i < 0 \end{cases} \quad (2.6)$$

A raíz de esta función nos encontramos con otra llamada *Softplus*, representada en la Figura 2.10 y que viene dada por la Ecuación 2.7. Esta función tiene propiedades de suavización y gradiente no nulo, lo cual mejora la estabilización y el rendimiento de la red neuronal profunda.

$$f(x) = \log(1 + e^x) \quad (2.7)$$

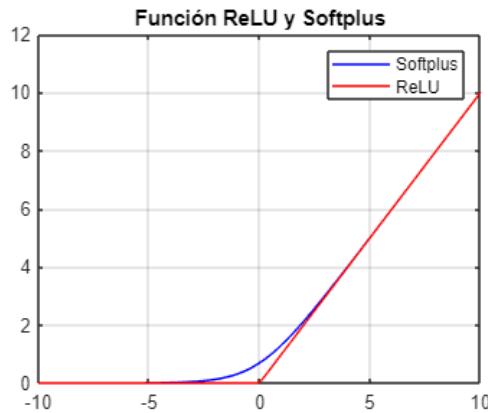


Figura 2.10: Representación gráfica de la Función Softplus y ReLU.

Cuando se trabaja con imágenes, ayuda a corregir el exceso de valores negativos. Las líneas de color negro, pasarán a ser de color gris neutro y por tanto, pasará a tener valores positivos en la imagen. Esto hace que se rompa la linealidad de la imagen, quedando las formas de la imagen al introducir la no linealidad dentro de ésta. Esto ayuda a la convergencia y a la detección de rasgos dentro de las RNCs.

Max-Pooling

Esta capa lo que va a hacer es reducir todavía más la dimensión del mapa de características, proporcionándonos un mapa de características *pooled*. Esto nos agrupará la información de los píxeles cercanos. Se aplicará un filtro más pequeño y lo que se buscará con este filtro es el valor máximo, buscándose así el píxel que resume la información.

Definiremos el paso con el que vamos moviendo el filtro de Max-Pooling, asegurándonos de esta forma que no se repite la información resumida. Se intenta agrupar la información lo máximo posible, preservando las características y teniendo en cuenta distorsiones espaciales, rotaciones u otras transformaciones con respecto a la imagen original.

Como hemos reducido el tamaño, esto ayudará en términos de procesamiento. Al tener un resumen realmente no interesa saber la posición exacta de la característica extraída, sino ayudar a corregir el sobreajuste. Las características se extraen a rasgos generales.

Esta capa lo que trata de hacer, se resume en descartar la información que no es relevante. Además, otra técnica que se utiliza, es el submuestreo que hace lo mismo que el Max-Pooling, pero en lugar del máximo proporciona el promedio. Esta técnica es conocida como *Average-Pooling*. Se puede ver un ejemplo de estas técnicas en la Figura 2.11. Esto permite un enfoque más generalizado para diferentes técnicas [11].

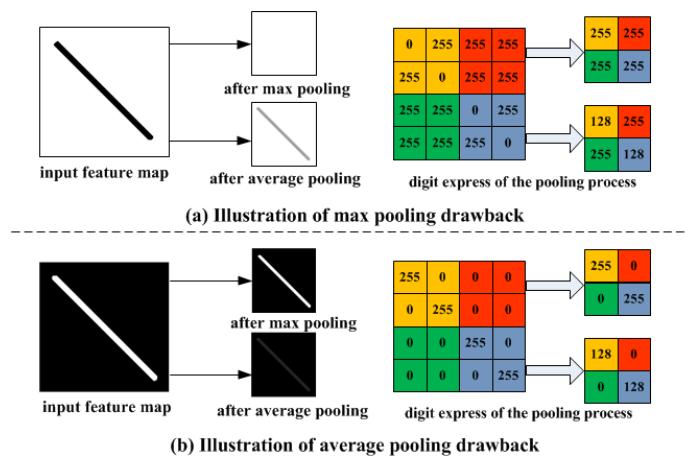


Figura 2.11: Ejemplo de Max-Pooling y Average-Pooling. Fuente: Dingjun Yu. Mixed Pooling for Convolutional Neural Networks

Flattening

Consiste en aplanar una matriz en un vector unidimensional. Dejándolo aplanado como la futura entrada de una RNN.

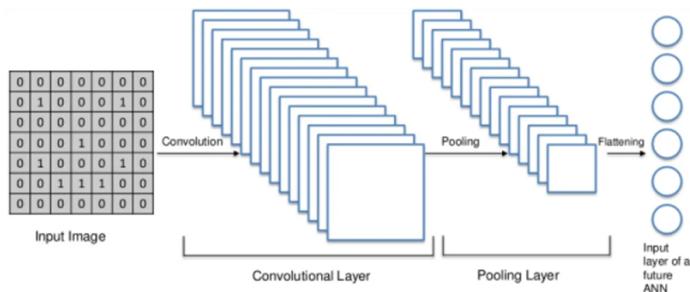


Figura 2.12: Capa *Flatten*. Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.

Fully Connected

Con esta capa, se trata de conectar una RNA con capas totalmente conectadas al resultado del *Flattening*. Es por esto por lo que se ha buscado obtener un vector unidimensional a partir de la matriz, para tener un vector como capa de entrada a una RNA. Se conectan todos los nodos de la RNA y se colocan tantas capas ocultas como queramos. Estos datos se introducirán en la RNA para proporcionarnos la información final.

Dado que se está realizando un proceso de clasificación, las salidas tienen que ser cada una de las posibles clases. Es más o menos como hacer un *clustering*. Cada neurona es responsable de detectar ciertos tipos de características.

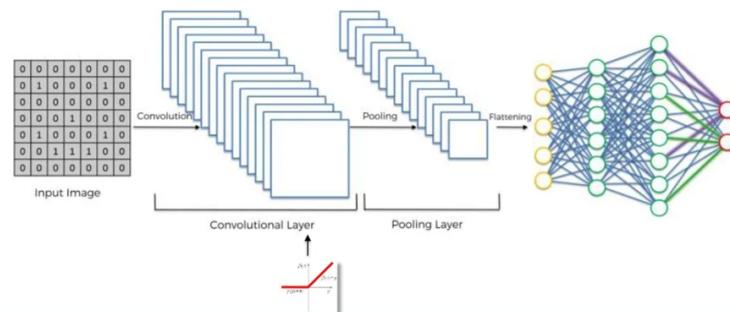


Figura 2.13: Capa *Fully Connected*. Fuente: Ligency I Team. Deep Learning de A a Z: redes neuronales en Python desde cero.

2.2.3. Entrenamiento de una RNC

El desempeño de una red neuronal, tanto a nivel computacional como a nivel predictivo, depende en gran parte de los datos disponibles para el entrenamiento. La arquitectura de la red neuronal necesita ser entrenada y optimizada. Se definen una serie de pasos a seguir para la etapa de entrenamiento:

1. Selección de todas las muestras para el entrenamiento.
2. Elección de las métricas de rendimiento.
3. Elección del clasificador y del algoritmo de clasificación.
4. Evaluación del desempeño del modelo.
5. Ajuste del algoritmo.

A grandes rasgos los parámetros que influyen de forma directa en el proceso de aprendizaje son los que se definen en los siguientes subapartados.

Hiperparámetros

Los hiperparámetros son parámetros ajustables que permiten controlar el proceso de entrenamiento de un modelo. Estos parámetros influyen en la capacidad y características de aprendizaje del modelo y es importante destacar que son ajenos al modelo, ya que no forman parte de este. A continuación se explican los más usados y los que más tarde necesitaremos conocer para el uso del modelo usado en este proyecto y que se explicará en la Sección 2.3.

- **Learning Rate:** Este parámetro nos indicará cómo de rápido aprenderá nuestra red neuronal. La elección de éste valor comprendido entre 0 y 1, es importante ya que valores muy grandes hará que la red pueda converger mucho más rápido, pero esto implicará que tendremos más error y resultados menos precisos y tener valores muy pequeños hará que el entrenamiento deba ser más largo, pero proporcionará, en primera instancia, resultados más exactos.



- **Weight Decay:** Es una técnica de regularización en el aprendizaje profundo. Este parámetro añade un término de penalización a la función de coste del modelo que tiene el efecto de reducir los pesos durante la retropropagación. Esto suele ayudar a corregir problemas de sobreajuste.
- **Batch Size:** Define el número de muestras del dataset que se propagarán por la red en cada época de entrenamiento.
- **Dropout Rate:** Método para la prevención del sobreajuste explicado en la sección 2.2.3.

Optimizadores

Durante el entrenamiento de una red neuronal, la base de datos utilizada se procesa en lotes. Durante este procesado se comparan las salidas obtenidas con las salidas que se quiere obtener, mediante el uso de una función de coste. Cuando se completa el procesamiento de un *batch*, se aplican algoritmos como puede ser el *Stochastic Gradient Descent* (SGD) a los resultados de la función de coste. El fin de estos algoritmos es minimizar la función de coste, intentando encontrar en el mejor caso un mínimo absoluto.

La magnitud de descenso o de actualización de los pesos depende de un parámetro ajustable denominado *learning-rate*. Cuanto mayor sea el valor del *learning-rate*, mayor será el porcentaje de cambio de los pesos tras cada lote.

A raíz del SGD se han desarrollado otros optimizadores que permiten variar de forma dinámica el valor del *learning-rate*. En concreto, nos vamos a centrar en el SGD y en el *Adam*, que son los que se van a utilizar en este trabajo.

SGD Es uno de los algoritmos que más se usa y más tiempo lleva usándose para el entrenamiento de redes neuronales. Matemáticamente hablamos de un vector de derivadas parciales. El SGD tarda en converger porque necesita un *forward* y *backward propagation* para cada registro. Y el camino para alcanzar los mínimos globales se vuelve muy ruidoso [12].

Adam Es una combinación de otros tres optimizadores (*SGD with momentum*, *Adagrad* y *Adadelta*). Este optimizador es uno de los optimizadores preferidos, debido a su fácil implementación, eficiencia y poco requerimiento de memoria. Es muy apropiado cuando hay problemas con gradientes muy ruidosos, ya que resuelve la fijación de la tasa de aprendizaje del SGD. Los hiperparámetros tienen una interpretación intuitiva y suelen requerir muy poco ajuste [12].

Función de Coste

A fin de obtener un buen modelo durante el entrenamiento, se debe buscar una forma de medir el error que va cometiendo el modelo. Para ello se aplican algoritmos de optimización que ayudan a obtener resultados mucho mejores.

Las funciones de coste son las que nos ayudan a interpretar si se está realizando un buen trabajo o no. La elección de esta función depende del modelo que queramos construir, cómo se quiere penalizar los errores y del problema que se quiera resolver.

Error Cuadrático Medio Es la suma de las diferencias cuadráticas de la imagen de referencia con la de salida. Penaliza los valores que son muy grandes, su interpretación no es tan trivial y funciona muy bien para optimizar regresiones en general. Las más comunes son las siguientes.

Error Absoluto Medio Es la suma media de los valores absolutos de la imagen de referencia con la de salida. Su convergencia es más difícil, penaliza menos los valores grandes y es más fácil de interpretar.

Entropía Cruzada Es una medida de precisión. Se explicará esta función más a fondo en el siguiente apartado. Su convergencia es más difícil y es más fácil de interpretar.



Entropía Cruzada

Recordando la función de activación *softmax*, lo que nos hará será normalizar las salidas de las neuronas para que la suma de todas estas sea la unidad. Esto pasa debido a que las neuronas de salida no están conectadas entre sí y no pueden saber los valores que toman las demás neuronas de la salida.

En general, para el típico ejemplo de una RNC que distingue perros y gatos las neuronas de la capa de salida proporcionaron números reales que no tienen por qué estar entre 0 y 1. Si llamamos z_1 al valor que proporciona la neurona correspondiente al perro y z_2 al valor proporcionado por la del gato, en efecto no deben sumar uno como hemos dicho. Al aplicar la Ecuación 2.5, ahora sí z_1 y z_2 son tales que se complementan y suman la unidad. Proporcionando así datos de tal forma que el ser humano lo pueda entender.

La entropía cruzada o *cross-entropy* viene dada por la Expresión 2.8.

$$H(P, Q) = - \sum_x P(x) \log(Q(x)) \quad (2.8)$$

Donde H representa la entropía. Es una mejora de la función de coste del error cuadrático medio, usando esta función de entropía cruzada, se obtiene un resultado bastante mejor. El objetivo será minimizar el error cometido en la clasificación por la red neuronal.

Volviendo al ejemplo anterior y dando por hecho que estamos en la fase de entrenamiento, sabemos que la red nos proporcionará una probabilidad tras aplicar la función *softmax*, pero al fin y al cabo lo que se quiere es una etiqueta de 1 para la etiqueta que se corresponde a la imagen que hemos introducido y de 0 para la que no se corresponde.

Teniendo $H(P, Q)$, P se corresponde a las etiquetas (1 ó 0) y Q se corresponde a las probabilidades proporcionadas por la red.

Se busca saber qué tan disperso es el error, es decir, cómo de probable es que la red catalogue de manera incorrecta. A diferencia que el ECM que proporciona un error entre el 0 y el 1, la entropía cruzada acentúa aún más el error.

Las ventajas de usar la Entropía Cruzada sobre el ECM es que si por ejemplo durante el *backpropagation* al inicio el valor de salida de la red neuronal tiende a ser bastante pequeño, eso significa que la corrección que va a tener que hacer no va a ser tan obvia y al algoritmo le va a costar mucho empezar. Cuando usamos el algoritmo en la entropía cruzada, va a hacer que una pequeña diferencia se note mucho.

El área de movimiento que haría el gradiente con el ECM hace que se dispare y se convierta en un enfoque más óptimo para buscar valores adecuado de los pesos.

En resumen, la entropía cruzada es capaz de mover el gradiente de una forma mucho más rápida de lo que lo haría el ECM. Para el caso de predicción, es mejor usar el ECM, ya que la entropía cruzada mide cuán lejos estoy de la categoría.

Análisis de Errores

El análisis de errores es una de las fases del proceso más importantes del entrenamiento tal y como se comentó en el apartado 2.2.3. Este análisis nos va a permitir saber qué hacer para mejorar el rendimiento de un modelo.

Subajuste Aparece cuando nos encontramos con un error de entrenamiento alto, significa que el modelo tiene problemas para aprender. Por lo tanto, el modelo aún no es capaz de generalizar y sus predicciones son poco fiables. Puede darse cuando el modelo es tan simple, que no es capaz de aprender la relación entre los datos de entrada y los de salida.

Sobreentrenamiento Uno de los puntos clave en este aprendizaje es no llegar al sobre-entrenamiento (*overfitting*), que consiste en que la red no aprenda a realizar la tarea deseada, sino que aprende a asociar las entradas y salidas vistas durante el entrenamiento, pero funciona incorrectamente para otras entradas que no

han sido procesadas. El objetivo final es que la red neuronal aprenda a generalizar, es decir, de proporcionar resultados acertados ante imágenes no vistas nunca antes.

Una de las causas puede ser que la complejidad del modelo. Esto implicaría que la red podría haber aprendido muchos de los datos de memoria y por lo tanto habría que intentar usar un modelo más simple. Otra causa puede ser que haya aprendido con pocos datos, es decir, es más fácil que haya aprendido qué resultado va con cada dato de entrada [13].

Por otro lado lo que se puede hacer es parar el entrenamiento antes de que termine quedándose en la zona de *underfitting* (representado en la figura 2.14) y otra opción, mucho más utilizada, es aplicar *Dropout*. Esto consiste en hacer que en cada época de entrenamiento se "apaguen" aleatoriamente cierta cantidad de neuronas de las capas ocultas de la red 2.15. Lo que se consigue con esto es que ninguna neurona memorice parte de la entrada; que es precisamente lo que sucede cuando tenemos sobreajuste [14].

Por último, nos podemos encontrar con otra técnica conocida como *early stopping*, cuya idea principal es detener el entrenamiento antes de que el modelo comience a sobreajustarse. Nos podemos encontrar con tres métodos para llevarla a cabo. La primera sería entrenando un número preestablecido de épocas e ir viendo donde nos encontramos con el sobreajuste, es un método que lleva mucha prueba y error. El segundo, es hacer que se detenga cuando la actualización de la función de pérdida se haga pequeña, el entrenamiento se detiene normalmente cuando la actualización está en torno a 0.001. El último y el más óptimo es usando la estrategia del dataset de validación. El error de entrenamiento disminuye exponencialmente hasta que el aumento de las épocas deja de tener un efecto tan grande en el error. El error de validación, sin embargo, disminuye inicialmente con el aumento de las épocas, pero después de un cierto punto, comienza a aumentar. Este es el punto en el que un modelo debería detenerse pronto, ya que más allá de esto el modelo empezará a sobreajustarse. Esto puede verse representado en la Figura 2.14, donde la recta roja indicaría el punto del *early stopping*.

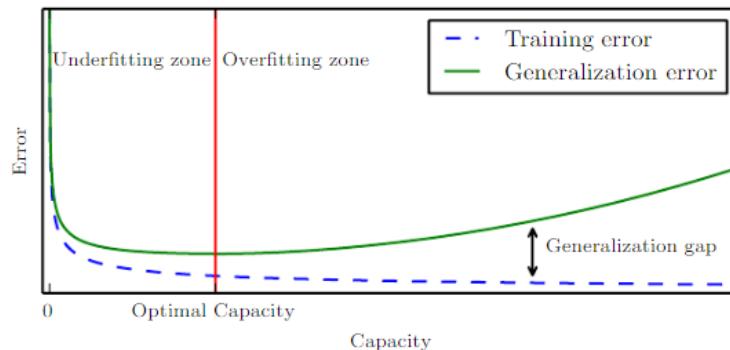


Figura 2.14: Overfitting y Underfitting. Fuente: Ian Goodfellow. Deep Learning.

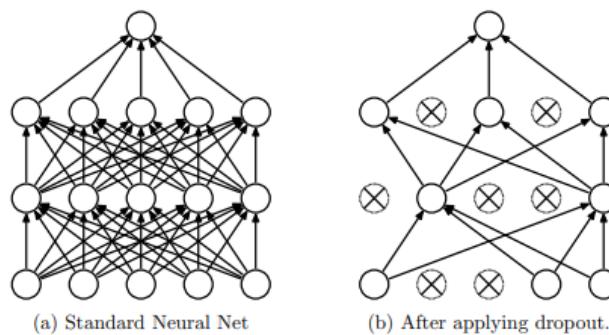


Figura 2.15: Dropout. Fuente: Ian Goodfellow. Deep Learning.



2.2.4. Transferencia de Aprendizaje

Se puede definir como un enfoque de aprendizaje profundo en el que un modelo que ha sido entrenado para una tarea se utiliza como punto de partida para un modelo que realiza una tarea similar. En la práctica, muy poca gente entrena una red convolucional completa desde cero (con inicialización aleatoria), porque es relativamente raro tener un conjunto de datos de tamaño suficiente [15].

Actualizar y reentrenar una red con este enfoque suele ser mucho más rápido y sencillo que entrenar una red desde cero. Este método se utiliza habitualmente en aplicaciones de detección de objetos (como nuestro caso de estudio), reconocimiento de imágenes y reconocimiento del habla, entre otras. Esta técnica se caracteriza por:

- Permitir entrenar modelos con menos datos etiquetados reutilizando modelos populares que ya han sido entrenados en grandes conjuntos de datos.
- Reducir en gran medida el tiempo de entrenamiento y el coste computacional.
- Aprovechar las arquitecturas de modelos desarrolladas por la comunidad de investigación del aprendizaje profundo, incluidas arquitecturas populares como GoogLeNet y ResNet.

2.2.5. Aumento de Datos

El *data augmentation* es un algoritmo que aumenta artificialmente el conjunto de datos mediante el uso de una sintetización específica para producir más datos de entrenamiento. Esto permite aumentar aún más el rendimiento de la CNN y evitar el *overfitting*. Su uso es especialmente adecuado cuando los datos de entrenamiento son limitados o difíciles de recopilar [16].

No obstante, a pesar de ser un poderoso algoritmo, es costoso en términos computacionales y requiere mucho tiempo para su aplicación. Una opción viable es aplicar un algoritmo de este tipo genérico, ya que es computacionalmente barato y fácil de implementar.

Aunque se entrará en detalle más adelante, cuando se explique el algoritmo implementado para este TFG, se han aplicado dos tipos de transformaciones:

- **Transformaciones Geométricas:** Son transformaciones que alteran la geometría de la imagen mediante la asignación de los valores individuales de los píxeles a nuevos destinos. La forma original de la clase representada dentro de la imagen se conserva, pero se altera a una nueva posición y orientación. Nos podemos encontrar con transformaciones tales como volteo, rotación y recorte.
- **Transformaciones Fotométricas:** Son transformaciones que alteran los canales RGB desplazando cada valor de píxel a nuevos valores de píxel según una heurística predefinida. Esto ajusta la iluminación y el color de la imagen y deja la geometría sin cambios. Además de encontrar transformaciones con el sistema de representación del color RGB, hay muchas otras usando el sistema HSV, con el cual podemos modificar características de las imágenes como pueden ser el brillo y la saturación.

2.3. Fudamentos de YOLOv5

2.3.1. Introducción

YOLO es un acrónimo de "You Only Look Once", es considerado la primera opción para la detección de objetos en tiempo real y esto es debido a que es el algoritmo de detección de objetos en tiempo real más avanzado en términos de rendimiento (FPS), facilidad de uso por su instalación y posibilidad de configuraciones y versatilidad, ya que los modelos pueden ser cubiertos para servir a diferentes dispositivos.

YOLOv5 es ligero, extremadamente fácil de usar, se entrena rápidamente, infiere rápidamente y tiene un buen rendimiento. YOLOv5 dispone de una serie de arquitecturas y modelos preentrenados con el dataset COCO.

2.3.2. Arquitectura YOLOv5

Aunque no se haya publicado un artículo científico sobre esta versión de YOLO y únicamente se disponga de un repositorio de GitHub [17], se han realizado varios estudios sobre la arquitectura de la red [18]. Partiendo de la Figura 2.16 podemos definir las tres redes que componen la arquitectura total de YOLOv5 [19]:

- **Primera Red:** CSPDarknet. Es una red de tipo *Backbone* o columna vertebral en español. Esto implica que la red se encarga de la extracción de características que se utiliza dentro de la arquitectura. Esta red divide el mapa de características de la capa base en dos partes y luego las fusiona a través de una jerarquía de etapas cruzadas. Esto permite un mayor flujo de gradiente a través de la red [20]. Termina con una capa de tipo SPP, esto significa que es una capa que elimina la restricción de tamaño fijo de la red, es decir, una RNC no requiere una imagen de entrada de tamaño fijo. Esta capa agrupa las características y genera salidas de longitud fija, que luego se introduce en la siguiente red. Con esta capa se evita la necesidad de recortar o deformar la entrada al principio [21].
- **Segunda Red:** PANet. Es un red de tipo *Neck* o cuello. Utilizando las características extraídas por la red anterior, esta red se encarga de fusionarlas todas a fin de reducir el muestreo. Esto ayuda a acelerar en gran medida el flujo de información en la red. Además esta red hace uso de una capa adaptativa *pooling*, que hace que la información útil de cada nivel de características se propague directamente a las siguientes subredes. Cuando se quiere hacer segmentación en lugar de detección como es nuestro caso de estudio, a fin de mejorar aún más la predicción de la máscara generada, se crea una rama complementaria que captura diferentes puntos de vista para cada propuesta [22].
- **Tercera Red:** YOLO. Es una red de tipo *Head* o cabeza. Usa el *head* de YOLOv3, cuya función es realizar las predicciones densas, es decir, la predicción final compuesta por un vector que contiene las coordenadas del *bounding box* (explicado en la Sección 2.3.3), la puntuación de confianza de la predicción y las clases de probabilidad [18].

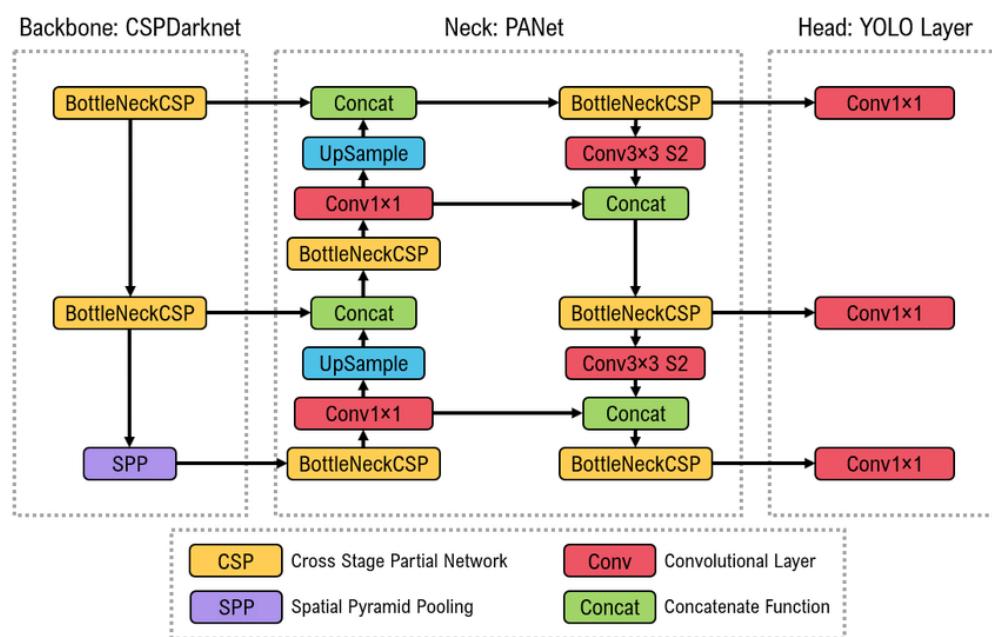


Figura 2.16: Arquitectura YOLOv5. Fuente: Iason Katsamenis. TraCon: A novel dataset for real-time traffic cones detection using deep learning.

2.3.3. Algoritmo YOLO

Para empezar, el algoritmo YOLO toma la imagen de entrada y la divide en $S \times S$ celdas, tal y como se puede ver en la primera imagen de la Figura 2.18 y si un objeto cae en una celda de la cuadrícula, esa celda es responsable de detectar ese objeto. Por otro lado, cada una de estas, producirá un número determinando de *bounding boxes* B, es decir, un rectángulo que intenta predecir la posición de un objeto.

Cada *bounding box* tiene un valor asignado conocido como *box confidence score*, el factor de confianza de la caja y viene dada por la Expresión 2.9. Estos valores de confianza reflejan cómo de fiable es el modelo si esa celda contiene un objeto y también cómo de preciso es al predecirlo. En el caso de que no haya un objeto en una celda, los factores de confianza de dicha celda debe ser cero. De lo contrario, si hay un objeto en esa celda, se busca que la puntuación de confianza sea igual a la Intersección Sobre la Unión o *Intersection Over the Union* (IOU) en inglés.

$$c_s = P(\text{Object})IoU_{pred}^{truth} \quad (2.9)$$

Un vector, de la forma de la ecuación 2.10, será asignado a cada *bounding box*. Donde p_c indica la confianza, c_n son cada una de las clases que se quieren detectar, (x, y) son las coordenadas, cuyos valores irán normalizados en función del número de celdas S y w, h son el ancho y alto de la *bounding box* respectivamente e irán normalizados en función del tamaño de la imagen. Estas cuatro últimas variables irán normalizadas en función del tamaño de la imagen.

$$g_n = \begin{bmatrix} p_c \\ c_1 \\ \vdots \\ c_n \\ x \\ y \\ w \\ h \end{bmatrix} \quad (2.10)$$

Por último, para obtener el factor de confianza de la detección, son necesarios varios parámetros. En primer lugar, se necesita el IOU de la *bounding box* y la anotación realizada sobre la imagen, que llamaremos *ground truth*. Se calculará el IOU como el cociente del área de superposición o intersección entre el área de la unión, tal y como se representa en la Figura 2.17.

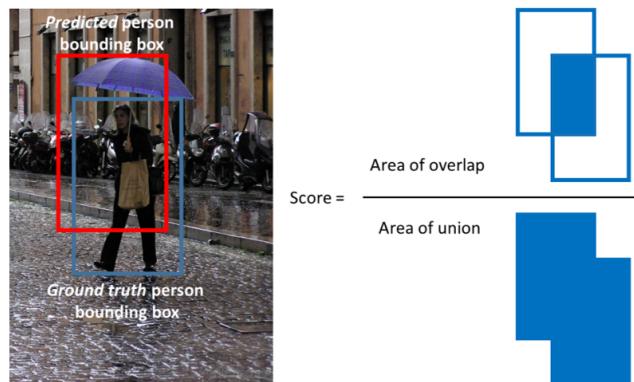


Figura 2.17: Intersección Sobre la Unión. Fuente: Do Thuan. “Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm”.

Cada celda de la cuadrícula también predice C probabilidades de clases condicionadas, $Pr(\text{Class}|\text{Object})$. El cuadro delimitador de B de la misma celda de la cuadrícula comparte un conjunto común de predicciones sobre la clase del objeto, lo que significa que todos los cuadros delimitadores en la misma celda de la cuadrícula tienen la misma clase.

Por último, se aplica el método de Supresión de No Máximos para eliminar todas las *bounding boxes* que no contengan ningún objeto o contengan el mismo objeto que otras cajas delimitadoras (Figura 2.18). Mediante la elección de un umbral, NMS elimina todos los cuadros delimitadores superpuestos que tienen un valor de IOU superior al valor del umbral [18].

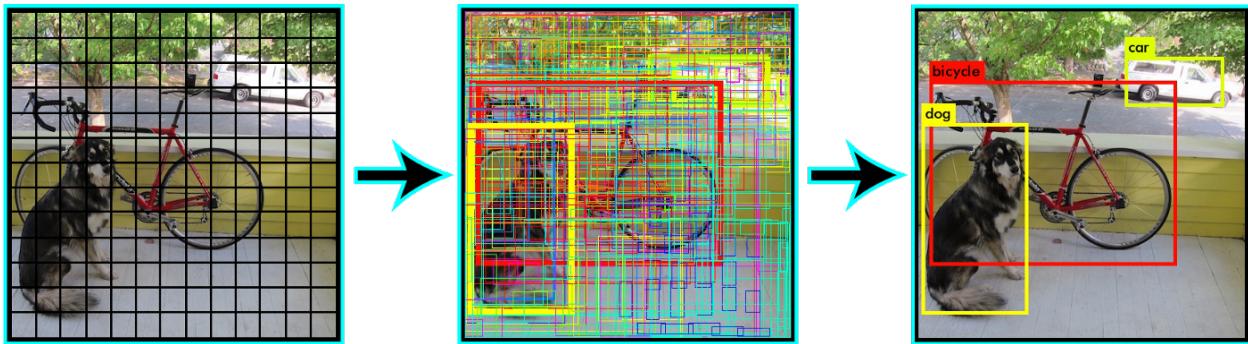


Figura 2.18: Algoritmo YOLO. Fuente: Do Thuan. “Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm”.

2.3.4. Hiperparámetros YOLOv5

YOLOv5 tiene unos 30 hiperparámetros utilizados para varios ajustes de entrenamiento. Estos se definen en archivos *.yaml que se le pasará al algoritmo de entrenamiento. A continuación se van a describir los valores predeterminados, además de los ya explicados en la Sección 2.2.3.

- **Initial Learning Rate:** Valor inicial para la tasa de aprendizaje. Su valor predeterminado es **0.01**.
- **Final Learning Rate:** Valor final para la tasa de aprendizaje. Su valor predeterminado es **0.01**.
- **SGD Momentum:** Indica un coeficiente matemático que se utiliza para mejorar la velocidad y precisión del entrenamiento. Su valor predeterminado es de **0.937**.
- **Weigth Decay:** Su valor predeterminado es de **0.0005**.
- **Warmup Epochs:** Son unas épocas con tasa de aprendizaje baja que sirven para permitir a los optimizadores adaptativos de la red neuronal calcular correctamente los gradientes. Su valor predeterminado es de **3** épocas.
- **Warmup Momentum:** Es el valor del *SGD momentum* durante las *warmup epochs*. Su valor predeterminado es de **0.8**.
- **Warmup Bias Learning Rate:** Todas las redes neuronales necesitan un bias para funcionar correctamente. Este ayuda a solucionar el problema del sobreentrenamiento. Este hiperparámetro indica la tasa de aprendizaje del bias durante las épocas de calentamiento. Su valor predeterminado es de **0.1**.
- **IoU training threshold:** El IOU (Intersection over Union) es un valor utilizado en las métricas. Indica cómo de cercanos estén la *bounding box* del *groundtruth* y la inferida por la red neuronal. Cuanto más cercanos, mayor intersección entre ambas *bounding boxes* y por tanto mayor valor de IOU. Este hiperparámetro indica cuál es el umbral mínimo para que el entrenamiento se dé como correcto. Su valor predeterminado es de **0.2**.

Para el entrenamiento del modelo de este TFG, los hiperparámetros que se han modificado, han sido:

- **Image Size:** Será el tamaño en píxeles de las imágenes de entrada a la red. En concreto, en este TFG se han usado dos tamaños diferentes **1536** y **640**. La elección de estos parámetros se explicarán en las siguientes secciones.



- **Batch Size:** Indica el tamaño de muestras que se van a utilizar en cada época del entrenamiento. La elección del tamaño del batch ha ido variando en función de los distintos datasets, de las pruebas realizadas y de la capacidad del servidor de entrenamiento. En concreto, se han usado tamaños de **4, 8 y 16**.
- **Epochs:** Indica el número de iteraciones máximas de entrenamiento. Al igual que con el tamaño de imágenes y del batch, han variado bastante y en parte también es debido a todo lo explicado en la Sección 2.2.3.
- **Weights:** Indica el valor de los pesos de toda red para realizar un reentrenamiento de ésta. Nos centraremos en este último concepto en la siguiente sección dado que será una parte fundamental de este proyecto.

Además de estos hiperparámetros contenidos en el fichero de configuración, en éste se incluyen también otros parámetros para llevar a cabo *data augmentation*. Cabe destacar que además de estos parámetros se ha desarrollado un código de Python, cuya finalidad será explicada en el apartado ??.

- **Degrees:** Rota la imagen de entrada un ángulo determinado aleatoriamente desde el ángulo indicado, en grados, hasta cero.
- **Translate:** Probabilidad de que la imagen de entrada se vayan a trasladar de forma aleatoria.
- **Scale:** Probabilidad de que se aplique un reescalado a la imagen de entrada.
- **Shear:** Máximo del intervalo, en grados, en el que se van a recortar las imágenes de entrada aleatoriamente.
- **Perspective:** Probabilidad de que se aplique a la imagen de entrada un cambio de perspectiva aleatorio.
- **Flip Up-Down:** Probabilidad de voltear la imagen de forma vertical.
- **Flip Left-Right:** Probabilidad de voltear la imagen de forma horizontal.
- **Mosaic:** Probabilidad de que se realice un mosaico con varias imágenes de entrada.
- **Mixup:** Probabilidad de que se mezclen varias imágenes de entrada y sus *bounding boxes* durante el entrenamiento.
- **Focal Loss Gamma:** Desenfoque máximo a aplicar a las imágenes de entrada aleatoriamente.
- **HSV Image:** Sistema de representación del color. Dicho sistema de representación tiene forma de anillo, donde el matiz (H) es un ángulo, la saturación (S) una distancia y el brillo o valor (V) la posición en el eje central.
 - **Hue:** Matiz máximo a aplicar a las imágenes de entrada aleatoriamente.
 - **Saturation:** Saturación máxima a aplicar a las imágenes de entrada aleatoriamente.
 - **Vue:** Valor máximo a añadir a los píxeles de las imágenes de entrada aleatoriamente.

2.3.5. Modelos YOLOv5

YOLOv5 ofrece diferentes modelos con distintas configuraciones y tamaños de parámetros (véase Figura 2.19). Respectivamente, los modelos YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l y YOLOv5x se refieren a los modelos nano(n), pequeño (s), mediano (m), grande (l) y extra-grande (x). Cada modelo tiene sus ventajas y desventajas, pero al final las diferencias están en sus complejidades, prestaciones y la precisión general.

Todos estos modelos han sido entrenados con el dataset COCO y en el repositorio de GitHub [17] se nos proporcionan sus respectivos pesos. Tras realizar dicho entrenamiento los resultados muestran que el tamaño de los modelos oscila entre los 4MB y los 166MB, los valores de mAP para el dataset COCO completo es de 28.4 a 50,7 y el tiempo de inferencia en la GPU Nvidia V100 varía entre los 6.3 y los 12.1 milisegundos.

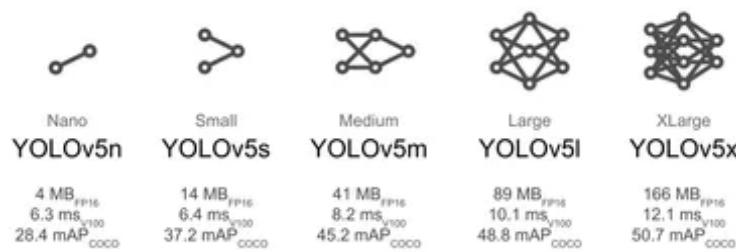


Figura 2.19: Modelos YOLOv5. Fuente: Repositorio GitHub ultralytics.

En el tema de estudio de este TFG, se busca una red neuronal para que funcione en un UAV. Por lo tanto, una de las métricas importantes a considerar es el funcionamiento en tiempo real y que además es un factor primordial en un proyecto cuyo objetivo es la detección de objetos en situaciones de emergencia y rescate. Para la elección del modelo para este proyecto, se ha partido de la gráfica de la Figura 2.20.

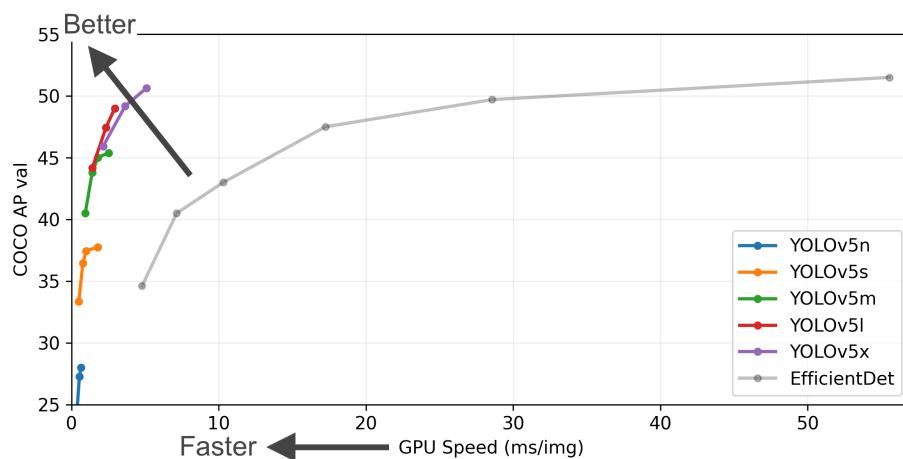


Figura 2.20: Gráfica Comparativa Modelos YOLOv5. Fuente: Repositorio GitHub ultralytics.

Realmente, somos nosotros quienes debemos establecer las condiciones a las que queremos que funcione nuestro sistema trabajo. En este caso, se busca que haya una buena relación de rapidez, precisión y tamaño. En general, los modelos más grandes proporcionarán mejores resultados, pero aunque sean más precisos ocuparán mucha memoria y serán más lentas.

Considerando todos estos factores se ha optado por usar el modelo YOLOv5l. Se ha decidido usar este modelo debido a que creo que hay cierta complejidad en las clases que se van a querer detectar y debido a las circunstancias de emergencia en las que se pretende que se desenvuelva la red, esta podría dar buenos resultados y adaptarse bastante bien.



UNIVERSIDAD
DE MÁLAGA



Capítulo 3

Entorno de Trabajo

3.1. Google Colab

En una primera instancia, se empezó testeando una de las posibles opciones que nos proporciona *ultralytics* desde su repositorio de GitHub, Google Colaboratory. Es una herramienta de desarrollo para aplicaciones de *machine learning* usado normalmente en docencia e investigación. Está basado en *Jupyter Notebooks*, permite la escritura y ejecución de código Python y de comandos de terminal sin necesidad de una configuración previa.

Los *notebooks* se guardan en tu espacio de almacenamiento de Google Drive o como en nuestro caso se pueden acceder a ellos desde GitHub. Estos *notebooks* se basan en el concepto de *Cloud Computing*, el cual consiste en que al ejecutar los códigos de estos *notebooks*, se utilicen recursos proporcionados por Google.

Esta prestación es gratuita hasta cierto punto. El usuario puede elegir entre hacer uso de una GPU o de una TPU, el cual es un circuito integrado especializado desarrollado por Google para acelerar procesos de IA y está enfocado principalmente al aprendizaje automático de redes neuronales. Al conectarse al *notebooks*, se asigna automáticamente un dispositivo. No obstante, esta asignación depende de la demanda y de la prioridad del usuario, dado que su finalidad es, en su mayor parte, la docencia. El tiempo que se puede dejar entrenando la red es limitado y si se sobrepasa se te echa del servidor y al volverte a conectar estarás en una cola de prioridad baja.

Google ofrece una opción premium de pago que proporciona recursos con bastante más prestaciones. Por último, cabe destacar que Google (Google Cloud Services) no es la única empresa que ofrece este tipo de servicios, empresas como Amazon y Microsoft con *Amazon Web Services* y *Microsoft Azure* respectivamente, de estas tres empresas, AWS es la que más tiempo lleva en el mercado y la más importante. Sin embargo, la diferencia está en que Google ha decidido especializarse en aprendizaje automático.

3.1.1. YOLOv5

Tal y como se ha comentado antes, desde el propio repositorio se proporciona un *notebook*, con el cual se empezó a trabajar y a realizar pruebas.

Dicho *notebook* está dividido en varias secciones, las cuales son:

- **Setup:** Se explican los comandos necesarios para clonar el repositorio e instalar la librerías necesarias en el entorno asignado.
- **Inferencia:** Se exponen los comandos y posibles parámetros para hacer inferencia sobre la red. Permite hacer inferencia sobre un directorio, un vídeo de YouTube, una imagen, un vídeo y sobre un Stream HTTP.
- **Validación:** Se muestran los comandos necesarios para validar la precisión del modelo entrenado y visualizar las métricas. En concreto, como ejemplo se usa el modelo preentrenado con la parte de validación del dataset COCO, que es con el dataset que se entrenó esta red.



- **Entrenamiento:** Se enseña lo necesario para poder poner a entrenar uno de los distintos modelos de YOLOv5 con un dataset específico.
- **Visualización:** Aunque se exponen tres métodos para la visualización de resultados, por su gran versatilidad, comprensibilidad y compatibilidad, se ha decidido usar el entorno *Weights & Biases*.
- **Extas:** Por último, nos encontramos con otras tres secciones extras donde se explican otros posibles usos del repositorio y otras posibilidades de uso de transformación y evaluación del modelo.

3.2. NVIDIA DGX

Por otro lado se ha hecho uso de una estación de entrenamiento desarrollado por NVIDIA. la estación de entrenamiento NVIDIA DGX es un ordenador desarrollado para la Computación de Alto Rendimiento o *High Computing Performance* en inglés. Está formado por cuatro tarjetas gráficas Tesla V100-DGXS 32GB y un Sistema Operativo Linux. Es un centro de datos de IA en un ordenador destinado a la ciencia de datos y muy vanguardista. Proporciona un rendimiento muy elevado y tiene soporte para que varios usuarios puedan usarlo al mismo tiempo.

Las operaciones de entrenamiento e inferencia del aprendizaje profundo se benefician enormemente de la aceleración de la GPU tanto en sistemas de una o varias GPUs. Las GPU utilizan ampliamente para acelerar los sistemas de aprendizaje profundo y son mucho más rápidas que las CPU tanto como para el entrenamiento como en la inferencia. Las nuevas características de las GPUs punteras proporcionan un mayor rendimiento de entrenamiento e inferencia de las redes neuronales. Otra de las características principales de este sistema es su soporte para el entrenamiento con multi-GPU que ofrecen una enorme escalabilidad del rendimiento. La programabilidad de la GPU permite desarrollar e implantar nuevos algoritmos rápidamente. Las GPUs NVIDIA proporcionan el alto rendimiento, la escalabilidad y la programabilidad necesarias para satisfacer las continuas demandas de los sistemas y algoritmos de IA y aprendizaje profundo para el entrenamiento y la inferencia.

En el mundo académico y la industria reconocen que las GPUs NVIDIA son los motores más avanzados para el entrenamiento de redes neuronales profundas debido a sus ventajas de velocidad y eficiencia energética. Como las redes neuronales se crean a partir de un gran número de neuronas idénticas, son altamente paralelas por naturaleza. Este paralelismo se traslada de forma natural a las GPU, que proporcionan un aumento significativo de la velocidad con respecto al entrenamiento en la CPU. Las redes neuronales dependen en gran medida de las operaciones matemáticas matriciales y las redes complejas de varias capas requieren enormes cantidades de rendimiento y ancho de banda tanto para como la eficiencia como para la velocidad. Las GPU tienen miles de núcleos de procesamiento optimizados para las operaciones matemáticas matriciales, que proporcionan entre decenas y cientos de TFLOPS de rendimiento. Las GPU son la plataforma de computación obvia para aplicaciones de inteligencia artificial y aprendizaje automático basadas en redes neuronales profundas. Las arquitecturas de estas GPUs especializadas en Deep Learning están altamente especializadas para ejecutar altas cargas de trabajo [23].

3.3. PyTorch

PyTorch es una biblioteca de aprendizaje automático y de código abierto desarrollada por Facebook. Está basada en Python, como su nombre sugiere, y pretende ofrecer una alternativa/sustitución más rápida a NumPy proporcionando un uso fluido de las GPU y una plataforma para el aprendizaje profundo que proporciona la máxima flexibilidad y velocidad.

Proporciona un marco de trabajo extremadamente fácil de usar, extender, desarrollar y depurar. Al estar escrito sobre Python es fácil de adoptar para la comunidad de ingenieros de software, investigadores y desarrolladores.

En el campo en el que a este trabajo se enmarca, PyTorch también facilita la producción de modelos de aprendizaje profundo. Está equipado con un tiempo de ejecución de C++ de alto rendimiento que los desarrolladores pueden aprovechar para los entornos de producción, evitando la inferencia a través de Python. El hecho de que Pytorch haga uso de la librería de Python Numpy, hace que sea más fácil de utilizar. En



general, PyTorch proporciona un marco y una plataforma excelentes en problemas de aprendizaje profundo de vanguardia mientras se centran en las tareas que importan y pueden depurar, experimentar y desplegar fácilmente [24].

3.4. Dependencias YOLOv5

Para poder hacer uso de esta RNC, es necesario importar una serie de librerías. Dichas librerías vienen proporcionadas en un archivo de texto, el cual es proporcionado en el repositorio. Se va a proceder a estudiar cada una de las librerías que contiene este archivo.

3.4.1. Matplotlib

Matplotlib es una librería bastante completa para crear visualizaciones estáticas, animadas e interactivas en Python. Será usada para representar el rendimiento de la red neuronal y la evaluación de las estadísticas con el paso de las épocas de entrenamiento. [25]

3.4.2. NumPy

NumPy es quizás la librería que más se ha mencionado hasta este momento y esto se debe a que es un paquete fundamental para la computación científica en Python. Da soporte para prácticamente todas las herramientas matemáticas, en especial para operaciones matriciales, lo cual es esencial cuando se trabaja con RNCs. [26]

3.4.3. OpenCV

OpenCV es actualmente la biblioteca de visión por computador más grande e importante, no solo porque contenga prácticamente todos los algoritmos de este campo implementados, sino porque está desarrollada tanto en Python como en C++. En este caso YOLOv5 hace uso de la versión de OpenCV escrita en Python. Será la encargada dibujar las *bounding boxes* a todo lo que procese la red. [27]

3.4.4. Pillow

Pillow es una librería multiplataforma que contiene todas las herramientas para la edición y manipulación de imágenes en Python. Se comunica con OpenCV y sirve como driver para grabar a disco las imágenes generadas en la fase de inferencia, ya que da soporte a muchos formatos de imágenes. [28]

3.4.5. PyYAML

PyYAML es un paquete de Python que proporciona funcionalidades para manipular, leer y analizar ficheros YAML. Es de las más importantes, ya que será la encargada de leer nuestro dataset e indicarle a la red donde de encuentra y el nombre de las clases que se va a detectar. Entraremos en más detalle en estos ficheros en la sección X. [29]

3.4.6. SciPy

SciPy es una librería de código abierto que se compone de herramientas y algoritmos matemáticos que usará nuestra red junto a NumPy. [30]

3.4.7. Torch

PyTorch es una de las librerías fundamentales del aprendizaje automático. La arquitectura YOLOv5 ha sido desarrollado sobre este entorno, sustituyendo así a darknet sobre la que se habían construido las anteriores versiones de YOLO. [31]



3.4.8. Torch Vision

Torchvision completa a PyTorch y consta de conjuntos de datos populares, arquitecturas de modelos y transformaciones de imágenes comunes para la visión por computador. [32]

3.4.9. Tqdm

Tqdm es un módulo de apoyo visual y es la encargada de generar las barras de progreso que aparecerán durante los procesos de entrenamiento y validación. Junto a esta barra también aparecerán otros parámetros como el tamaño del batch, el porcentaje de progreso, la memoria interna empleada. [33]

3.4.10. Tensorboard

TensorBoard proporciona la visualización y las herramientas necesarias para experimentar con el aprendizaje automático. Permite visualizar métricas tales como la pérdida y la exactitud y poder seguir la evolución de las mismas. Además nos da la posibilidad de visualizar el grafo del modelo, ver histogramas de pesos, sesgos y otros tensores a medida que cambian con el tiempo. [34]

3.4.11. Seaborn

Seaborn es una librería de visualización de datos en Python basada en *matplotlib*. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos. [35]

3.4.12. Pandas

Pandas es una librería de software escrita como extensión de NumPy para manipulación y análisis de datos. Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. Se usará, en concreto, para la manipulación de ficheros CSV. [36]

3.4.13. Thop

PyTorch-OpCounter proporcionará información sobre el número de operaciones en coma flotante (FLOPs) que empleará nuestro modelo durante su entrenamiento y durante la fase de inferencia. [37]

3.4.14. Request

Request es un modulo de Python que permite hacer peticiones HTTP a cualquier servidor web y permite imprimir por pantalla la respuesta de dicha petición. [38]

3.4.15. Relación y jerarquización de los módulos

Anteriormente, se han definido todos los módulos que son necesarios para hacer funcionar la red neuronal. Se podría decir que todas estas librerías siguen una jerarquía piramidal poseyendo una dependencia relativa entre ellas en cuanto a su uso. Este concepto se representa en la figura 3.1.

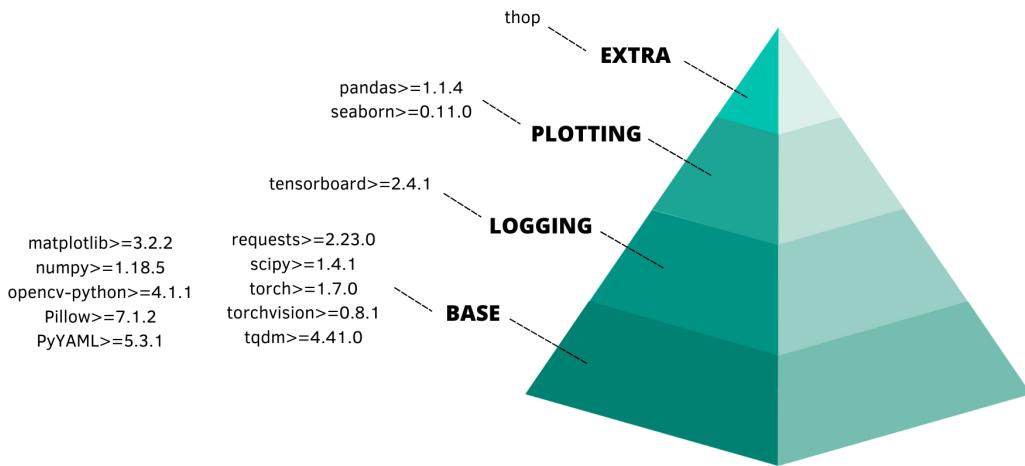


Figura 3.1: Jerarquía de los módulos de YOLOv5.

```
$ docker pull ultralytics/yolov5
Using default tag: latest
latest: Pulling from ultralytics/yolov5
[...]
Digest: sha256:a4f61f983630de1527eb54222c0887c897d3c5273cff59984ba2542899ee3250
Status: Downloaded newer image for ultralytics/yolov5:latest
docker.io/ultralytics/yolov5:latest
$ docker image ls
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
ultralytics/yolov5  latest   faff1a6f87cc  46 hours ago  15.4GB
$ docker run -it --name tfg_yolov5_ruben -v C:\Users\Usuario\Desktop\Volume:/usr/src/Volume ultralytics/yolov5:latest --gpus device=0
```

Figura 3.2: Creación del contenedor.

En la parte más baja de la pirámide, nos encontramos con lo que se define como *"base"* donde se encuentran todos los módulos encargados de las operaciones centrales de todo el proceso de convergencia de RNC. Sobre esta capa se encuentra otra denominada *"logging"* con las que se realiza la conexión con otras aplicaciones de interfaces gráficas. Encima, se sitúa la capa *"plotting"* encargada de la presentación de resultados. La última que se encuentra en la cumbre de la pirámide, está la capa *"extra"*, destinada a la obtención e interpretación de los resultados extraídos durante la ejecución.

3.5. Contenedores

En último lugar y de lo que se ha hecho uso en última instancia ha sido un contenedor. Se describe a fondo de donde viene este concepto, además de algunos comandos para ejecutarlo, en el Anexo A. Algo muy



común en este tipo de proyectos es usar *Docker*. Desde el repositorio de GitHub podemos acceder a *DockerHub* desde donde nos podremos descargar la imagen que nos ofrece el creador con todas las configuraciones.

A continuación, se van a exponer en la Figura 3.2 los comandos necesarios para la configuración del contenedor que se va a usar para realizar tanto el entrenamiento como la validación.

Se han ejecutado tres comandos hasta este punto tal y como se muestra en la Figura 3.2. En primer lugar, lo que se ha realizado es una descarga de la imagen desde el repositorio de DockerHub. Tras finalizar la descarga de la imagen, comprobamos si esta todo en orden listando las imágenes, para ello ejecutamos el segundo comando. Lo siguiente que se debe hacer es construir el contenedor, que se creará usando el tercer comando. Para crear el contenedor dada una imagen, hacemos uso del comando *docker run* y se han usado las siguientes opciones:

- *-it*: Esta etiqueta hará que el contenedor sea interactivo.
- *-name*: Nombre que se le va a asociar al contenedor, en este caso, se creará con el nombre *tfg_yolov5_ruben*.
- *-v*: Haciendo uso de esta etiqueta asociaremos un volumen a nuestro contenedor, gracias a esto podremos ir compartiendo los resultados entre el contenedor y nuestro dispositivo.
- *ultralytics/yolov5:latest*: Con esta acción, indicamos la imagen que queremos que se use para crear el contenedor y la versión.
- *-gpus*: Esta etiqueta, tal y como su nombre indica, es la que se usa para asociar una o varias GPUs al contenedor. En este proyecto solo se va a usar una de las cuatro GPUs que tiene disponible la estación DGX.

Automáticamente, tras la ejecución de este comando estaremos dentro del contenedor y ya estará listo para su uso. En la figura 3.3 se muestra el contenido del contenedor:

```
$ root@b3fc1c5fcbd3:/usr/src/app# ls -l
total 260
-rw-r--r-- 1 root root 4964 Aug 10 22:27 CONTRIBUTING.md
-rw-r--r-- 1 root root 35127 Aug 10 22:27 LICENSE
-rw-r--r-- 1 root root 21798 Aug 10 22:27 README.md
drwxr-xr-x 5 root root 4096 Aug 10 22:27 data
-rw-r--r-- 1 root root 13626 Aug 10 22:27 detect.py
-rw-r--r-- 1 root root 30674 Aug 10 22:27 export.py
-rw-r--r-- 1 root root 6842 Aug 10 22:27 hubconf.py
drwxr-xr-x 3 root root 4096 Aug 10 22:27 models
-rw-r--r-- 1 root root 1138 Aug 10 22:27 requirements.txt
-rw-r--r-- 1 root root 1731 Aug 10 22:27 setup.cfg
-rw-r--r-- 1 root root 33522 Aug 10 22:27 train.py
-rw-r--r-- 1 root root 58784 Aug 10 22:27 tutorial.ipynb
drwxr-xr-x 7 root root 4096 Aug 10 22:27 utils
-rw-r--r-- 1 root root 19656 Aug 10 22:27 val.py
$ root@b3fc1c5fcbd3:/usr/src/app# cd ..
$ root@b3fc1c5fcbd3:/usr/src# ls -l
total 8
drwxrwxrwx 1 root root 512 Aug 13 10:43 Volume
drwxr-xr-x 1 root root 4096 Aug 10 22:27 app
drwxr-xr-x 2 root root 4096 Jun 30 23:19 cudnn_samples_v8
lrwxrwxrwx 1 root root 13 Jul 5 10:15 tensorrt -> /opt/tensorrt
```

Figura 3.3: Exploración del contenedor.

El contenedor está construido sobre un sistema operativo Linux por lo tanto nos moveremos en el usando los comandos propios de este. En primer lugar listamos el directorio en el que nos encontramos, y



vemos que se ha clonado el repositorio de GitHub y ahora tenemos disponible todos los ficheros necesarios para usar la RNC.

Cuando se construyó el contenedor (Figura 3.2), se asoció un volumen a este en un directorio por debajo del que nos encontramos. Si se cambia de directorio usando el segundo comando, se puede comprobar que se ha creado dicha carpeta.

```
$ root@b3fc1c5fcbd3:/usr/src/app# exit
$ docker exec -it b3fc1c5fcbd3 /bin/bash
$ root@b3fc1c5fcbd3:/usr/src/app#
```

Figura 3.4: Acceso al contenedor.

En el caso que se deba salir del contenedor para realizar alguna acción en local aplicaremos los comandos de la Figura 3.4. Para salir del contenedor usaremos el comando *exit*. Luego tendremos que usar el ID que tiene asociado que en el caso de que no se sepa se puede usar uno de los comandos de la Figura 3.5 para averiguarlo. Aplicando el último comando, se volverá a estar dentro del contenedor.

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
b3fc1c5fcbd3      ultralytics/yolov5:latest "/opt/nvidia/nvidia_"
17 minutes ago     Up 9 minutes       6006/tcp, 8888/tcp
tfg_yolov5_ruben
$ docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
b3fc1c5fcbd3      ultralytics/yolov5:latest "/opt/nvidia/nvidia_"
17 minutes ago     Up 9 minutes       6006/tcp, 8888/tcp
tfg_yolov5_ruben
```

Figura 3.5: Listar características del contenedor.



UNIVERSIDAD
DE MÁLAGA



Capítulo 4

Obtención del Dataset

4.1. Adquisición del Dataset

Las imágenes utilizadas para componer el dataset este TFG son una selección de otros tres datasets relacionados con la temática que a nosotros nos concierne:

- **VisDrone:** Utilizado en la primera fase de este proyecto. Se entrará en más detalle en la explicación de estas fases en la Sección 5.3.
- **AIDER:** Utilizado en la segunda fase. Se escogerán ciertas imágenes para la formación del dataset final y del que obtendremos los resultados que evaluaremos.
- **UMA-SAR:** Las imágenes de este dataset componen la otra parte del dataset final.

4.2. VisDrone

VisDrone es un dataset formado por un conjunto de imágenes ya etiquetadas con el fin de que la visión por computador se adapte a los drones. Este conjunto de datos ha sido recogido por el equipo AISKEYEYE del Laboratorio de Aprendizaje Automático y Minería de Datos de la Universidad de Tianjin (China). [39]



Figura 4.1: Imágenes de Ejemplo del dataset VisDrone. Fuente: PRCV 2022. Aerial-Ground Intelligent Unmanned System Environment Perception Challenge.

El conjunto de datos está formado por fotogramas procedentes de clips de vídeos e imágenes estáticas, capturados por varias cámaras montadas en drones. Hay que tener en cuenta que el conjunto de datos se



recopiló en diferentes escenarios y bajo diversas condiciones meteorológicas y de iluminación.

Estos fotogramas están anotados manualmente con más de 2,6 millones de *bounding boxes*. También se proporcionan algunos atributos importantes, como la visibilidad de la escena, la clase de objeto y la oclusión, para una mejor utilización de los datos. [39]

4.2.1. Formato del etiquetado

VisDrone aunque proporciona ya todas las imágenes con sus etiquetas, sigue una especificación de estas distintas a las de YOLO (dicho formato es explicado en la Sección 4.5.1). En este caso, tendremos ocho parámetros que definirán nuestra *bounding box*, tal y como se muestra en la Figura 4.2.

```
<bbox_left>, <bbox_top>, <bbox_width>, <bbox_height>, <score>,  
<object_category>, <truncation>, <occlusion>
```

Figura 4.2: Formato etiquetas VisDrone.

- ***bbox_left***: La coordenada x de la esquina superior izquierda de la *bounding box* del objeto predicho.
- ***bbox_top***: La coordenada y de la esquina superior izquierda de la *bounding box* del objeto predicho.
- ***bbox_width***: La anchura en píxeles de la *bounding box* del objeto predicho.
- ***bbox_height***: La altura en píxeles de la *bounding box* del objeto predicho.
- ***score***: La puntuación en el archivo DETECTION indica la confianza de la *bounding box* predicha que encierra un objeto. La puntuación en el archivo GROUNDTRUTH se establece en 1 o 0. 1 indica que la *bounding box* se tiene en cuenta en la evaluación, mientras que 0 indica que la *bounding box* será ignorada.
- ***object_category***: La categoría del objeto indica la clase a la que pertenece el objeto etiquetado.
- ***truncation***: La puntuación en el archivo de resultados de DETECCIÓN debe fijarse en la constante -1. La puntuación en el archivo GROUNDTRUTH indica el grado en que las partes del objeto aparecen fuera de un cuadro (es decir, sin truncamiento = 0 (proporción de truncamiento 0 %), y truncamiento parcial = 1 (proporción de truncamiento 1 % - 50 %)).
- ***occlusion***: La puntuación en el archivo DETECTION debe establecerse en la constante -1. La puntuación en el archivo GROUNDTRUTH indica la fracción de objetos ocluidos (es decir, sin oclusión = 0 (proporción de oclusión 0 %), oclusión parcial = 1 (proporción de oclusión 1 % - 50 %), y oclusión fuerte = 2 (proporción de oclusión del 50 % al 100 %)).

4.3. AIDER

AIDER es un dataset formado por una recopilación de imágenes de cuatro catástrofes: incendio/humo, inundación, derrumbe de edificios/burbujas y accidentes de tráfico, así como de entornos sin la presencia de situación de desastre. [40]

Estas imágenes aéreas se recogieron de múltiples fuentes, como buscadores de imágenes como por ejemplo, google images, bing images, youtube, sitios web de agencias de noticias, etc.. Otras de las imágenes proceden de otras bases de datos con imágenes aéreas generales y otras son tomadas por el mismo equipo con sus UAVs. Los distintos sucesos de catástrofe del dataset se capturaron con diferentes resoluciones y bajo diversas condiciones de iluminación y perspectiva.



Figura 4.3: Imágenes de Ejemplo del dataset AIDER. Fuente: Christos Kyrkou. AIDER: Aerial Image Database for Emergency Response applications.



Figura 4.4: Imágenes de Ejemplo del dataset UMA-SAR. Fuente: Jesus Morales, Ricardo Vázquez-Martín, Anthony Mandow, David Morilla-Cabello y Alfonso García Cerezo. “The UMA-SAR Dataset: Multimodal Data Collection from a Ground Vehicle During Outdoor Disaster Response Training Exercises.

Como se ha comentado, este dataset esta compuesto de cinco clases, de las cuales sólo tres son de interés para este proyecto. Se han seleccionado las clases de fuego, inundación y escombros de donde se han escogido 100 imágenes de dichas categorías, es decir, un total de 300 imágenes.

4.4. UMA-SAR

Es un dataset propio del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga. Las imágenes de este dataset se han obtenido durante las Jornadas sobre Seguridad y Emergencia de la Escuela de Ingenierías Industriales de los últimos años. [41]

Las imágenes utilizadas en los procesos de entrenamiento, validación y test son procedentes de una selección de fotogramas de tres vídeos de las Jornadas sobre Seguridad y Emergencia de años anteriores. Se hizo una selección de tres vídeos distintos en situaciones bastante distintas. A fin de obtener los fotogramas de estos, se utilizó una herramienta del reproductor de vídeos VLC. Este procedimiento se explica en el Anexo C. La elección de los fotogramas se hizo pensando en evitar que la red neuronal pudiera sobre-entrenar



durante el entrenamiento porque hubieran muchas imágenes repetidas. En este caso, de todos los fotogramas que se generaron, se cogieron un total de 195.

4.5. Etiquetado de imágenes

Se etiquetaron para este TFG un total de 505 imágenes para los procesos de entrenamiento, validación y test. El dataset etiquetado se compone de las imágenes seleccionadas de AIDER y UMA-SAR. La distribución de las imágenes ha sido la siguiente:

- **Entrenamiento:** (70 % del total) Utilizadas para el entrenamiento de la red para entrenar el modelo.
- **Validación:** (15 % del total) Utilizadas durante el entrenamiento de la red para ajustar los parámetros del modelo y evitar el sobre-entrenamiento.
- **Test:** (15 % del total) Utilizadas para testear el modelo, tras la finalización del entrenamiento. Además de estas imágenes se han seleccionados otros vídeos de YouTube para comprobar otros escenarios.

Las clases de interés en este trabajo, es decir, las áreas de desastre que el modelo debe ser capaz de detectar una vez terminado el entrenamiento son:

- **Fuego:** Nombrado como *fire* para la detección.
- **Escombros:** Nombrado como *debris* para la detección.
- **Inundación:** Nombrado como *flood* para la detección.
- **Humo:** Nombrado como *smoke* para la detección.

Cabe destacar que estas clases son de por sí muy difíciles de detectar ya que poseen características muy aleatorias. Debido a esto, a la red le costará más generar los filtros necesarios para aprender a detectar dichas clases.

Para mejorar la detección en las zonas de desastre y que la RNC sea más útil, se han añadido dos clases más:

- **Persona:** Nombrado como *person* para la detección. Aumenta bastante la complejidad de la red dado que una persona vista desde el aire es bastante pequeña, es decir, se encuentra en unos pocos píxeles. Se decidió incluir esta clase dado que normalmente siempre hay personal de los cuerpos de emergencia. Además,
- **Vehículo de Emergencia:** Nombrado como *emergency-vehicle* para la detección. A raíz de la clase anterior, se optó por detectar estos vehículos dado que se suelen encontrar por los alrededores de la zona.

Se muestra en la Figura 4.5 un ejemplo con algunas de las clases descritas.

4.5.1. Herramientas de Etiquetado

Para realizar el etiquetado de imágenes se ha usado la herramienta *labelImg*. Es una herramienta gratuita de código abierto de anotación de imágenes, que se puede conseguir desde el repositorio de GitHub de su creador [42]. Se adjunta un anexo explicativo sobre el uso y procedimiento de etiquetado [Véase Anexo C].

De las varias opciones que hay disponibles para poder ejecutar esta herramienta, para este proyecto se ha optado por usar Anaconda para ejecutarla. El único requisito para la ejecución es tener instalado una versión mayor de Python 3.

La versión de YOLO que usamos, requiere que tanto las imágenes como las etiquetas de dichas imágenes estén organizadas de una determinada forma, tal y como se muestra en la figura 4.6. Para la



Figura 4.5: Imagen de ejemplo de las clases a detectar.

exportación de etiquetas en la barra lateral de opciones, se elegirá la opción de YOLO y de esta forma el fichero `.txt` de salida estará organizado de una forma que la red entenderá. Cada vez que se etiquete algo y guardemos de irá añadiendo una línea con cinco parámetros, tal y como se muestra en la Figura 4.7. Los parámetros de las etiquetas son:

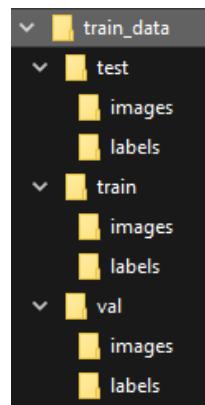


Figura 4.6: Organización de los directorios de las imágenes y las etiquetas.

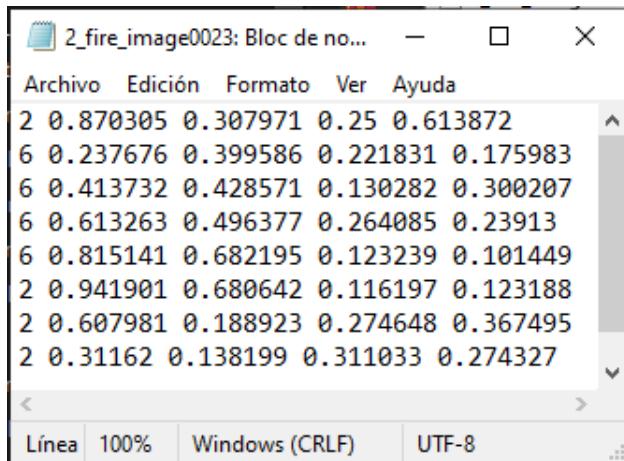


Figura 4.7: Archivo .txt de las etiquetas.

```

1 path: ../train_data # dataset root dir
2 train: ../train_data/images/train/ # train images (relative to 'path') 2938 images
3 val: ../train_data/images/val/ # val images (relative to 'path') 76 images
4 test: ../train_data/images/test/ # test images (relative to 'path') 75 images
5
6 # Classes
7 nc: 10 # number of classes
8 names: [ 'pedestrian', 'people', 'bicycle', 'car', 'van', 'truck', 'tricycle', 'awning↔
     -tricycle', 'bus', 'motor' ] # class names
    
```

Figura 4.8: Fichero customdata.yaml.

- **Primer parámetro:** Indica la clase que representa la *bounding box*. Estas clases y su orden están definidas en el fichero *customdata.yaml*, el cual es necesario para el entrenamiento. La estructura del fichero *customdata.yaml* se puede observar en la figura 4.8. Estas clases son las que se han descrito anteriormente en la sección 4.5.
- **Segundo parámetro:** Parámetro *bx*. Coordenada relativa del eje X en el que se encuentra la esquina superior izquierda de la caja delimitadora.
- **Tercer parámetro:** Parámetro *by*. Coordenada relativa del eje Y en el que se encuentra la esquina superior izquierda de la caja delimitadora.
- **Cuarto parámetro:** Parámetro *bw*. Anchura relativa de la caja delimitadora.
- **Quinto parámetro:** Parámetro *bh*. Altura relativa de la caja delimitadora.

4.5.2. Aumento del Dataset

A fin de mejorar el resultado del entrenamiento siempre es recomendable realizar un aumento del dataset, aplicando diversos tipos de transformaciones a las imágenes de entrenamiento. Cabe destacar que cuando se aplican dichas transformaciones, se aplican tanto a las imágenes como a sus etiquetas. De este modo, se puede multiplicar el tamaño del dataset existente. De cara a nuestro modelo, estas nuevas imágenes

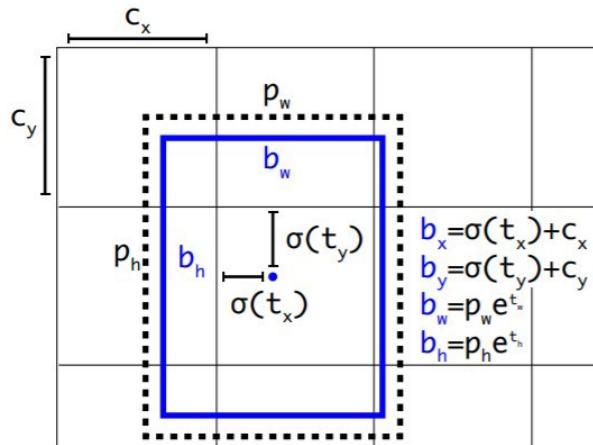


Figura 4.9: Organización de los directorios de las imágenes y las etiquetas. Fuente: Do Thuan. “Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm”.

generadas son nuevas y pueden aportar información extra. Por lo tanto, el entrenamiento se va a realizar con las imágenes de entrenamiento y validación originales, además de las nuevas imágenes de entrenamiento obtenidas con el aumento de datos.

Es bastante importante que para al final del entrenamiento, la red sea capaz de generalizar y proporcionar buenas detecciones, el dataset debe ser muy variado, intentando simular posibles entornos en los que se pueda encontrar funcionando la red como pueden ser condiciones atmosféricas y cambios de punto de vista. En el caso de no hacer esto la red solo estaría preparada para desenvolverse en un tipo de escenario.

Por lo tanto, debido a estas razones, se decidió aplicar esta técnica. Para llevarlo a cabo, se ha optado por desarrollar un script de Python. Se plantearon dos opciones, usar la librería de Visión por Computador, OpenCV o una librería específica de dataaugmentation, albumations. Al final, se optó por la segunda debido a su uso específico, simplicidad y mayor diversidad. Dicho código se adjunta en el Anexo B.

El script que se desarrolló requiere que todas las imágenes tengan un formato *png*, sin embargo al haber recopilado el dataset de diversas fuentes todas tienen un formato diferente. Por lo tanto, se escribió un segundo código que realizaba dicha conversión. Dicho código se adjunta en el Anexo B.

A fin de dar una mayor uniformidad al dataset, se creyó conveniente renombrar todas las imágenes. Para ello, se escribió un tercer *script* que se encargaba de leer todos los ficheros de un directorio que fueran de tipo *png* y darle un nuevo nombre. Dicho código se adjunta en el Anexo B.

El programa desarrollado se encarga de leer los ficheros contenidos en una carpeta. El script leerá la imágenes y las etiquetas siguiendo la jerarquía de la figura 4.6. Una vez leída la imagen y su etiqueta asociada, se procede a aplicar las siguientes transformaciones:

- **Volteo Horizontal:** Es la primera transformación que se aplica. Se le ha asociado una probabilidad del 100 %, es decir se va a aplicar a todas las imágenes que se lean.
- **Cambio de Saturación:** Esta transformación aplicará un cambio de saturación entre el -30 % y el 30 % a todas las imágenes.
- **Cambio de Contraste:** Esta transformación aplicará un cambio de contraste entre el -20 % y el 20 % a todas las imágenes.
- **Rotación:** Con esta transformación se aplica una rotación en un rango entre -98º y 98º a la totalidad de las imágenes.
- **Reescalado:** Se aplicará un reescalado de hasta un 20 % más del tamaño anterior con una probabilidad del 100 %.

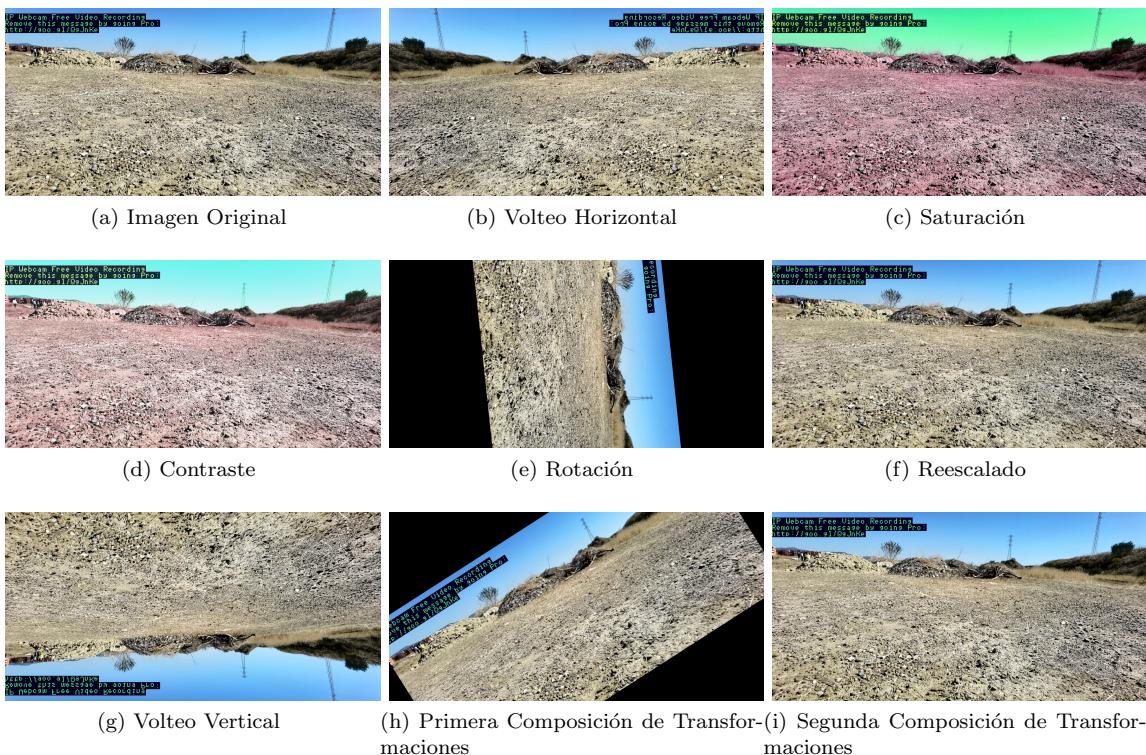


Figura 4.10: Imágenes y sus transformaciones

- **Volteo Vertical:** Al igual que la transformación del Volteo Horizontal, se le ha asociado una probabilidad del 100 %.
- **Primera Composición de Transformaciones:** Se han mezclado las transformaciones de reescalado y de rotación con las mismas características que cuando se aplicaron por separado.
- **Segunda Composición de Transformaciones:** En esta ocasión se han combinado las transformaciones de difuminado tipo cristal y de difuminado gausiano.

Tras la aplicación de estas transformaciones, se obtuvo un dataset aproximadamente ocho veces mayor que el original. Se pueden observar ejemplos de cada transformación en la figura 4.10.

Capítulo 5

Fases de Entrenamiento

5.1. Introducción

Este proyecto se decidió dividirlo en dos fases de trabajo. Una primera fase donde se partió de la red neuronal YOLOv5 preentrenada con el dataset COCO y se trabajó con el dataset VisDrone para hacer una primera adaptación de la RNC para visión aérea.

Posteriormente, una vez se obtuvieron unos resultados aceptables en la primera fase, se procedió a reentrenar la red para el objetivo final de este trabajo, siendo esta la segunda fase de éste.

5.2. Métricas

Antes de explicar el desarrollo de las dos fases de entrenamiento es conveniente explicar varios conceptos claves que se van a usar para explicar los resultados finales.

En los problemas de clasificación, a fin de evaluar el desempeño de la red, existen unas métricas usadas para determinar cuán bien es capaz de detectar las diferentes clases una RNC. Dichos indicadores son los siguientes:

- **Verdadero Positivo o *True Positive*** (TP): Son aquellos valores que el algoritmo clasifica como positivos y efectivamente son positivos. Es decir, la red ha hecho la detección correctamente.
- **Verdadero Negativo o *True Negative*** (TN): Son valores que el algoritmo clasifica como negativos y que realmente son negativos. Es decir, no se ha realizado una detección incorrecta.
- **Falso Positivo o *False Positive*** (FP): Son valores que el algoritmo clasifica como positivo cuando realmente son negativos. En los algoritmos de detección, se puede considerar como FP aquellas detecciones cuyo IOU no supere un determinado umbral, que generalmente es del 50 %.
- **Falso Negativo o *False Negative*** (FN): Son valores que el algoritmo clasifica como negativo cuando realmente son positivos. Esto ocurre cuando el modelo no detecta un elemento que efectivamente se encuentra en la imagen.

Cuando organizamos estos cuatro indicadores en una matriz obtenemos lo que se conoce como **matriz de confusión**. Dicha matriz siempre será de tamaño $n \times n$, siendo n el número de clases que vaya a detectar nuestro modelo.

5.2.1. Precisión

La precisión es utilizada para poder saber qué porcentaje de valores que se han clasificado como positivos son realmente positivos, es decir, la precisión mide la exactitud del modelo a la hora de clasificar una muestra como positiva. Dicha relación viene dada por la ecuación 5.1. Normalmente, se denota con la letra **P** de *precision* en inglés.

$$precision = \frac{TP}{TP + FP} \quad (5.1)$$

5.2.2. Sensibilidad

La sensibilidad es utilizada para determinar el porcentaje de detecciones correctas que ha realizado el modelo entre todas las detecciones reales. Cuanto mayor sea la sensibilidad, más muestras positivas se detectarán. Sólo se preocupa de cómo se clasifican las muestras positivas. Es independiente de cómo se clasifican las negativas. Se calcula siguiendo la expresión 5.2. Normalmente, se denota con la letra **R** de *recall* en inglés.

$$recall = \frac{TP}{TP + FN} \quad (5.2)$$

5.2.3. Curva PR

Esta gráfica se obtiene al enfrentar en una gráfica la precisión contra la sensibilidad [43]. Se emplea para la evaluación del rendimiento de modelos de detección de elementos. Ambas métricas son inversamente proporcionales, es decir, si se entrena el modelo para aumentar la precisión, disminuirá la sensibilidad, y viceversa. En la figura 5.1, la curva morada representaría a un modelo perfecto.

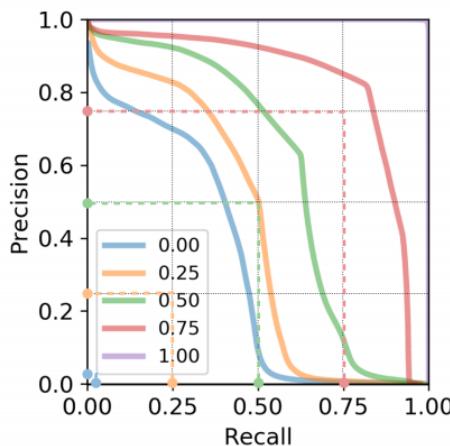


Figura 5.1: Ejemplo curvas PR.

5.2.4. Mean Average Precision (mAP)

Métrica que se utiliza habitualmente para analizar el rendimiento de los modelos de detección y segmentación. Se define como la precisión media de todas las mediciones que superen un determinado umbral de IOU. Matemáticamente, es el área bajo la curva PR, a partir del umbral IOU escogido.

5.3. Primera Fase

5.3.1. Introducción

En primer lugar, fue importante analizar la red YOLO ya entrenada. Detectar objetos desde un vehículo aéreo es mucho más complejo que detectarlos desde uno terrestre, dado que los objetos son mucho más pequeños y es más difícil obtener filtros capaces de identificar las clases deseadas. Además, en los modelos de detección cuyas métricas se basan en el IOU, suele ser muy difícil obtener valores de mAP@0.5 altos. Las *bounding boxes* generadas por la red y las del *ground-truth* muchas veces deberán generar un IOU mayor del 50 % en muy pocos píxeles.

```

1 # YOLOv5 by Ultralytics , GPL-3.0 license
2 # VisDrone2019-DET dataset https://github.com/VisDrone/VisDrone-Dataset by Tianjin ←
3 # University
4 # Example usage: python train.py —data VisDrone.yaml
5 # parent
6 #       yolov5
7 #       datasets
8 #           VisDrone      downloads here (2.3 GB)
9 # Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt , or 3) list: ←
10 [path/to/imgs1, path/to/imgs2, ...]
11 path: ./datasets/VisDrone # dataset root dir
12 train: VisDrone2019-DET-train/images # train images (relative to 'path') 6471 images
13 val: VisDrone2019-DET-val/images # val images (relative to 'path') 548 images
14 test: VisDrone2019-DET-test-dev/images # test images (optional) 1610 images
15 # Classes
16 nc: 10 # number of classes
17 names: ['pedestrian', 'people', 'bicycle', 'car', 'van', 'truck', 'tricycle', 'awning←
18 tricycle', 'bus', 'motor']
19 # Download script/URL (optional) ←
20
21 download: |
22     from utils.general import download, os, Path
23
24     def visdrone2yolo(dir):
25         from PIL import Image
26         from tqdm import tqdm
27
28         def convert_box(size, box):
29             # Convert VisDrone box to YOLO xywh box
30             dw = 1. / size[0]
31             dh = 1. / size[1]
32             return (box[0] + box[2] / 2) * dw, (box[1] + box[3] / 2) * dh, box[2] * dw, ←
33             box[3] * dh
34
35             (dir / 'labels').mkdir(parents=True, exist_ok=True) # make labels directory
36             pbar = tqdm((dir / 'annotations').glob('*.*txt'), desc=f'Converting {dir}')
37             for f in pbar:
38                 img_size = Image.open((dir / 'images' / f.name).with_suffix('.jpg')).size
39                 lines = []
40                 with open(f, 'r') as file: # read annotation.txt
41                     for row in [x.split(',') for x in file.read().strip().splitlines()]:
42                         if row[4] == '0': # VisDrone 'ignored regions' class 0
43                             continue
44                         cls = int(row[5]) - 1
45                         box = convert_box(img_size, tuple(map(int, row[:4])))
46                         lines.append(f'{cls} {'.join(f'{x:.6f}' for x in box)}\n")
47                         with open(str(f).replace(os.sep + 'annotations' + os.sep, os.sep + '←
48                             labels' + os.sep), 'w') as fl:
49                             fl.writelines(lines) # write label.txt
50
51     # Download
52     dir = Path(yaml['path']) # dataset root dir
53     urls = [
54         'https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019←
55         DET-train.zip',
56         'https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019←
57         DET-val.zip',
58         'https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019←
59         DET-test-dev.zip',
60         'https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019←
61         DET-test-challenge.zip'
62     ]
63     download(urls, dir=dir, curl=True, threads=4)
64
65     # Convert
66     for d in 'VisDrone2019-DET-train', 'VisDrone2019-DET-val', 'VisDrone2019-DET-test←
67     dev':
68         visdrone2yolo(dir / d) # convert VisDrone annotations to YOLO labels

```

Figura 5.2: Fichero *VisDrone.yaml*.



Parámetros de Entrenamiento	Valor
Nº de épocas	30
Tamaño de Lote	8
Tamaño de la Imagen	640
Pesos Iniciales	Por defecto
Modelo	YOLOv5s

Tabla 5.1: Primer Entrenamiento en Google Colab

A raíz de este último problema, es de donde sale la idea de hacer uso de un conjunto de datos ya conocido, como es el caso del dataset de VisDrone, el cual es el que se ha usado en esta primera fase para realizar la adaptación del modelo a imágenes aéreas. El equipo que generó este dataset propone cada año un concurso con varias categorías distintas. Entre ellas se encuentra una que intenta hacer que YOLOv5 consiga una alta precisión en la detección de las clases de este dataset.

5.3.2. Primeros Entrenamientos

Los primeros entrenamientos consistieron en realizar un primer análisis de las clases a detectar y modificando los hiperparámetros para ver el efecto que se conseguía con dichas modificaciones.

Antes de explicar como se han realizado los entrenamientos, es necesario comentar varios ficheros necesarios para la ejecución del comando usado para dicho fin. Por un lado, tendríamos el fichero de configuración indicando las clases y la ubicación de las imágenes, análogamente como en la Figura 4.8 en el repositorio se nos proporciona un fichero de configuración, el cual se muestra en la Figura 5.2.

Este fichero realiza bastantes funciones. En primer lugar, empieza definiendo los directorios relativos donde se encontrará el dataset. Posteriormente, define las clases a detectar. A continuación, nos encontramos con una serie de líneas de código que constituyen una función que transformará el formato de las etiquetas de VisDrone al formato YOLO. Por último nos encontramos otra serie de líneas encargadas de descargar el dataset desde el repositorio oficial de VisDrone [44] y un bucle encargado de leer todas las imágenes y llamar a la función de conversión.

Comentado este fichero, se va a proceder a exponer el procedimiento seguido para poner a reentrenar la red neuronal con este dataset. Las primeras pruebas fueron realizadas usando el entorno de Google Colab. Se partió del *notebook* proporcionado en el repositorio. Habiendo ejecutado los primeros comandos para instalar en el entorno de ejecución las librerías necesarias, nos vamos a la sección de Entrenamiento y se ejecutó el comando de la Figura 5.3.

```
$ python train.py --img 640 --batch 8 --epochs 30 --data VisDrone.yaml --weights yolov5s.pt --cache
```

Figura 5.3: Listar características del contenedor.

En resumen, se ejecutó el entrenamiento con los parámetros de la tabla 5.1. No obstante, la duración del entrenamiento fue de cinco horas dado que la GPU que proporciona el entorno es limitada. Además, dado que el fin de Google Colab es educativo principalmente, hay un tiempo de uso limitado y el propio entorno detiene la ejecución pasado un tiempo para liberar la GPU para otros posibles usuarios.

El hecho de poder ejecutar la red durante 30 épocas únicamente hacia que la red no alcanzase un mAP@0.5 óptimo, a pesar de que se estaba usando el modelo pequeño de YOLO. Debido a que no se obtuvieron resultados óptimos con estos recursos, se decidió empezar a usar la estación NVIDIA DGX, la cual proporciona unos recursos más potentes y no tiene un límite temporal de uso.

Tras la decisión de cambiar el entorno de trabajo, se realizaron más pruebas con este dataset. Dichas pruebas consistieron en aumentar el número de épocas y cambiar el modelo YOLOv5s por el modelo



Parámetros de Entrenamiento	Valor
Nº de épocas	2000, 80, 40
Tamaño de Lote	8
Tamaño de la Imagen	960
Pesos Iniciales	Por defecto
Modelo	YOLOv5l

Tabla 5.2: Primer Entrenamiento en NVIDIA DGX

YOLOv5m. No obstante, ninguno de estos entrenamientos proporcionaban buenos resultados y se llegaron a las siguientes conclusiones.

- En primer lugar, el tamaño de las imágenes del dataset VisDrone es de 960x540. Al estar usando un tamaño de 640 puede ocurrir que muchas etiquetas desaparezcan al estar reduciendo el tamaño de la imagen.
- Por otro lado, las primeras pruebas realizadas se realizaron en torno a las 100 épocas. Muchos modelos necesitan muchas más épocas para poder converger y proporcionar buenos resultados. Por lo tanto, se aumentó el entrenamiento a unas 2000 épocas para ver donde se producía *overfitting* y ver en torno a qué época se encontraban los mejores resultados.
- En último lugar, el modelo que se estaba usando puede ser que no fuera lo suficientemente grande para aprender todos los filtros necesarios. Debido a esto se contempló el uso de YOLOv5x, pero viendo el gran tamaño de la red se optó por utilizar un modelo inferior (YOLOv5l). Esto se hizo porque la finalidad de esta red es ser usada en un UAV, por lo tanto no era eficiente tener una red tan grande funcionando en el vehículo y debía encontrarse un equilibrio.

Teniendo en cuenta todas estas conclusiones, se realizaron varios entrenamientos dentro del contenido (cuya puesta en marcha se explicó en la Sección 3.5) con los parámetros de la Tabla 5.2. Tras hacer un entrenamiento de 2000 épocas se vio que a partir de la época 80 se producía *overfitting* por lo tanto, se realizó otro entrenamiento con esa cantidad de épocas y en realidad se veía que se producía en torno a la época 45. Por último, se decidió realizar un entrenamiento con 40 épocas, con el cual fue posible alcanzar unos resultados mejores que los anteriores.

Los resultados obtenidos tras la modificación de los parámetros aumentaron considerablemente respecto a los primeros, el mAP@0.5 ha aumentado de un entrenamiento a otro en aproximadamente un 20 %, es decir, efectivamente las conclusiones expuestas anteriormente eran acertadas, pero entrenar una red neuronal convolucional tan pocas épocas puede derivar en futuros fallos de la red.

5.3.3. Entrenamiento Final

En este punto del trabajo, se optó por realizar una búsqueda de otros proyectos de investigación que hubieran trabajado con este mismo dataset. Dicha búsqueda fue bastante larga, dado que como se comentó previamente, la finalidad de este dataset es realizar una competición para ver quién consigue los mejores resultados y por lo tanto había bastante material. Finalmente, se decidió seguir la investigación de la Beihang University de Beijing, China [45].

Dicha investigación consistió en desarrollar un red variante de YOLOv5. Este equipo mezclando una red neuronal de tipo transformer¹ con la arquitectura YOLO obtuvieron otra red a la que ellos llaman TPH-YOLO. Esta red, fue evaluada con el dataset VisDrone y tras varias pruebas obtuvieron un mAP@0.5 de un 37.32 %.

¹Modelo de Deep Learning compuesto por una arquitectura de tipo codificadora o *encoder* y decodificadora o *decoder*. Siguen un modelo de aprendizaje autosupervisado, el cual es un subtipo de aprendizaje no supervisado. Su uso se debe a que son mucho más resistentes a las distorsiones de las imágenes de entrada. [46]



Hyps	Por defecto	TPH-YOLO
lr0	0.01	0.0032
lrf	0.01	0.12
momentum	0.937	0.843
weight_decay	0.0005	0.00036
warmup_epochs	3.0	2.0
warmup_momentum	0.8	0.5
warmup_bias_lr	0.1	0.05
box	0.05	0.07
cls	0.5	0.18
cls_pw	1.0	0.631
obj	1.0	0.15
obj_pw	1.0	0.911
iou_t	0.2	0.2
anchor_t	4.0	3.0
fl_gamma	0.0	0.0
hsv_h	0.4	0.015
hsv_s	0.7	0.3
hsv_v	0.4	0.5
degrees	0.0	0.2
translate	0.1	0.0
scale	0.5	0.4
shear	0.0	0.0
perspective	0.0	0.0
flipud	0.0	0.0
fliplr	0.5	0.5
mosaic	1.0	1.0
mixup	0.0	0.2
copy_paste	0.0	0.0

Tabla 5.3: Tabla Comparativa de Hiperparámetros

Gracias a que el equipo tiene dicho experimento como código abierto, fue posible investigar los parámetros que ellos usaron para realizar los entrenamientos. Entre otras cosas, destacaba el uso de hiperparámetros. En los entrenamientos previos realizados en este proyecto se usaron los que había por defecto, no obstante había bastante diferencia en muchos de los hiperparámetros que este grupo usaron. Destacaba especialmente la diferencia de *learning rates*. La Tabla 5.3 muestra una comparativa de los hiperparámetros por defecto y los usados en esta investigación.

Dado que dichos cambios en los hiperparámetros parecían funcionar bastante bien, se decidió probar cómo funcionaban con una arquitectura YOLOv5l. Efectivamente, se consiguió un modelo entrenado que proporcionaba unos resultados próximos a los que obtuvo este equipo. Los parámetros usados en este entrenamiento fueron los de la Tabla 5.4 y se usó el comando de la Figura 5.4.

```
$ python train.py --img 1536 --batch 4 --epochs 120 --data VisDrone.yaml --weights yolov5l.pt --hy data/hyps/hyp.VisDrone.yaml --cfg models/yolov5l.yaml --cache --device 0
```

Figura 5.4: Listar características del contenedor.



Parámetros de Entrenamiento	Valor
Nº de épocas	120
Tamaño de Lote	4
Tamaño de la Imagen	1536
Pesos Iniciales	TPH-YOLO
Modelo	YOLOv5l

Tabla 5.4: Entrenamiento Final en NVIDIA DGX

5.4. Segunda Fase

5.4.1. Introducción

En esta segunda fase se pretende entrenar la red en situaciones de catástrofes, con un dataset diferente al anterior y con clases relevantes para una situación de rescate. Para esto último, fue necesario etiquetar más imágenes, siendo así como se consiguió formar el conjunto de datos para esta parte del proyecto. Dicho dataset fue elaborado con imágenes de UMA-SAR y AIDER tal y como se relató en la Sección 4.5.

Una vez es obtenida la red neuronal entrenada con el dataset de VisDrone, ya podemos considerar que hemos adaptado YOLOv5 a una visión aérea. Esto es muy importante para el siguiente paso a realizar, ya que los pesos de las neuronas han sido adaptados para detectar ciertas clases desde el aire y cuando realicemos la siguiente serie de entrenamientos será más fácil que la red aprenda a detectar las clases de interés de este proyecto.

En los apartados siguientes, a fin de ver cómo es posible obtener los mejores resultados, se ha optado por comparar cómo cambian los resultados usando la técnica del aumento de datos. Asimismo, en cada escenario se estudia como evolucionan los resultados usando dos optimizadores distintos. En concreto, dichos optimizadores son el **SGD** y el **Adam**.

Llegados a este punto es importante destacar que lo que se va a volver a aplicar la técnica de transferencia de conocimiento y obtener una RNC adaptada a nuestro campo de estudio.

5.4.2. Entrenamientos Sin Aumento de Datos

Los primeros entrenamientos de esta segunda fase se realizaron con el dataset personalizado que se formó con imágenes del dataset AIDER y el de UMA-SAR, tal y como se explicó en la sección 4.5. Usando una de las herramientas de Docker conocida como **volumen** podemos compartir todas las imágenes con el contenedor. Para ello desde nuestro ordenador local copiamos las imágenes en nuestra carpeta establecida como volumen. Realizado esto, se elabora el fichero de configuración para indicar la ubicación del dataset, el cual es el que se muestra en la Figura 4.8.

Para comenzar, en este entrenamiento con el dataset y las clases de rescate, se va a utilizar como pesos iniciales los mejores obtenidos en el entrenamiento anterior, ya que ya han sido optimizados para imágenes aéreas. Debido a esto, antes de empezar a realizar los entrenamientos, lo que se hizo fue buscar el fichero que contiene los mejores pesos del entrenamiento realizado al final de la primera fase.

Tras finalizar cada entrenamiento se genera un directorio dentro de nuestro entorno de trabajo que contiene las estadísticas y los pesos de nuestra red neuronal. En concreto de guardan dos ficheros distintos, uno con los pesos de la última época de entrenamiento y otro con los de la mejor época de entrenamiento. Generalmente, la última época no tiene porqué tener las mejores estadísticas, por lo tanto para este proyecto se va a elegir el archivo que contiene las mejores estadísticas. Dicho fichero se denomina *best.pt*, el cual moveremos al directorio principal donde se encuentra el código *train.py* que se ejecuta para entrenar. Este fichero será con el que se trabajará a partir de ahora en lugar de con el fichero de pesos *yolov5l.pt* que se usaba en la primera fase. En este caso, se decidió utilizar los hiperparámetros por defecto que usa YOLOv5, los cuales se pueden ver en la Tabla 5.3. Se realizó una primera serie de entrenamientos con el optimizador SGD, para ello se usó el comando de la figura 5.5 que usa los parámetros de entrenamiento de la Tabla 5.5.



```
$ python train.py --batch 8 --epochs 130 --data ../Volume/customdata.yaml --weights best.pt --cache --device 0
```

Figura 5.5: Entrenamiento con optimizador SGD sin aumento de datos.

```
$ python train.py --batch 8 --epochs 130 --data ../Volume/customdata.yaml --weights best.pt --cache --device 0 --optimizer adam
```

Figura 5.6: Entrenamiento con optimizador Adam sin aumento de datos.

```
$ python train.py --batch 16 --epochs 150 --data ../Volume/customdata.yaml --weights best.pt --cache --device 0
```

Figura 5.7: Entrenamiento con optimizador SGD con aumento de datos.

```
$ python train.py --batch 16 --epochs 150 --data ../Volume/customdata.yaml --weights best.pt --cache --device 0 --optimizer Adam
```

Figura 5.8: Entrenamiento con optimizar Adam con aumento de datos.

Como se puede observar en la Figura 5.5 no se ha especificado el tamaño de la imagen de entrada ni el optimizador del que se va a hacer uso, esto se debe a que cuando no se usan las etiquetas *-imgz* ni *-optimizer* los parámetros por defecto son 640 y SGD respectivamente. Se decidió entrenar la red 130 épocas dado que entre las pruebas que se realizaron se vio que en torno a ese número de épocas era cuando la red empezaba a sobreentrenar.

Análogamente, para realizar el entrenamiento con los mismos parámetros pero con el optimizador Adam, se debe ejecutar el comando de la Figura 5.6, el cual ejecutará el entrenamiento con los mismos parámetros de la Tabla 5.5 a excepción del optimizador. No obstante, los resultados obtenidos no son totalmente satisfactorios y es conveniente mejorarlos, para ello, en este punto del trabajo se opta por usar la técnica del aumento de datos.

5.4.3. Entrenamientos Con Aumento de Datos

En la sección 4.5.2 se generó el nuevo dataset que se usará para realizar los entrenamientos a partir de este punto del trabajo. A fin de no tener que modificar el archivo de configuración, lo que se hace es simplemente eliminar el dataset anterior y copiar el nuevo usando los mismos nombres en los directorios que en el anterior. Realizado esto, ya se puede comenzar con los nuevos entrenamientos.

Los comandos utilizados para realizar los entrenamientos en esta ocasión se muestran en las Figuras 5.7 y 5.8. Dichos comandos, realmente son análogos a los de las Figuras 5.5 y 5.6. Únicamente se ha realizado un par de modificaciones en los parámetros respecto a la sección anterior. Dichos cambios se pueden ver en la Tabla 5.6. Se ha aumentado el tamaño del lote y se ha entrenado la red durante 20 épocas más respecto a los realizados en la Sección anterior.



Parámetros de Entrenamiento	Valor
Nº de épocas	130
Tamaño de Lote	8
Tamaño de la Imagen	640
Optimizador	SGD, Adam
Pesos Iniciales	<i>best.pt</i>
Modelo	YOLOv5l

Tabla 5.5: Parámetros de entrenamiento con optimizador SGD sin aumento de datos.

Parámetros de Entrenamiento	Valor
Nº de épocas	130, 150
Tamaño de Lote	16
Tamaño de la Imagen	640
Optimizador	SGD, Adam
Pesos Iniciales	<i>best.pt</i>
Modelo	YOLOv5l

Tabla 5.6: Parámetros de entrenamiento con optimizador SGD con aumento de datos.

La diferencia de imágenes de una dataset a otro es de más de 2600 imágenes. Es decir hay ocho veces más de imágenes en esta ocasión, por lo tanto, tenemos la cantidad de etiquetas de cada clase también han aumentado en consonancia con el número de imágenes. Esto es bastante bueno de cara a la mejora de los resultados, ya que la red tendrá mas instancias para generar una mayor cantidad de filtros y aumentar la calidad de detección de la red.

Efectivamente, tras la finalización de todos los resultados y comparando las métricas obtenidas, se puede observar que el mAP@0.5 medio de todas las clases ha aumentado. Analizando el mAP@0.5 de cada clase y analizando las detecciones con las imágenes de test se puede concluir que llegados a este punto la red es capaz de detectar todas las clases y algunas de ellas con una fiabilidad bastante alta. Por lo tanto, se puede decir que se ha obtenido un modelo YOLOv5l que es capaz de detectar zonas catastróficas.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Capítulo 6

Resultados

6.1. Resultados de la Primera Fase

Los primeros entrenamientos realizados se comenzaron a realizar con la ayuda de Google Colab, tal y como se contó en la Sección 5.3.2. Tras la finalización del primer entrenamiento se obtuvieron las métricas de la Tabla 6.1. El mAP@0.5 medio de todas las clases fue de 0.281, el cual es demasiado bajo. Como se comentó, no es posible obtener buenos resultados debido al limitado tiempo que se nos proporciona en dicho entorno.

Analizando el mAP@0.5 de cada una de las clases, destaca el 0.677 del coche. No obstante, debido a la gran cantidad de instancias de coches que hay cabe de esperar que es posible aumentar dicho valor. Dado que existía la posibilidad de cambiar de entorno y conseguir mejores resultados, esto fue lo que se hizo.

Tras el cambio de entorno de trabajo de Google Colab a la estación NVIDIA DGX, se realizaron una serie de entrenamientos muy parecidos a los que se hicieron anteriormente. Se modificaron parámetros como las épocas de entrenamiento y el modelo de YOLOv5, sin embargo no se conseguían resultados óptimos. Tras esto, se llegaron a ciertas conclusiones, las cuales tras tenerlas en cuenta en los entrenamientos sucesivos, proporcionaron unos resultados mucho más adecuados. En la Figura 6.1, se muestra la gráfica del mAP@0.5, donde se puede ver que se obtiene un valor de 0.4922 en la época 40 de entrenamiento, la cual es la última.

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
all	548	38759	0.39	0.301	0.281	0.146
pedestrian	548	8844	0.41	0.348	0.337	0.135
people	548	5125	0.389	0.325	0.286	0.092
bicycle	548	1287	0.198	0.141	0.0891	0.0262
car	548	14064	0.537	0.689	0.677	0.434
van	548	1975	0.384	0.277	0.278	0.184
truck	548	750	0.398	0.284	0.256	0.148
tricycle	548	1045	0.41	0.132	0.148	0.0777
awnig-tricycle	548	532	0.289	0.0865	0.0891	0.0559
bus	548	251	0.466	0.334	0.312	0.179
motor	548	4886	0.419	0.396	0.338	0.124

Tabla 6.1: Primer Resultado del Entrenamiento en Google Colab

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
all	548	34897	0.463	0.345	0.346	0.21
pedestrian	548	7985	0.547	0.385	0.414	0.194
people	548	4735	0.49	0.328	0.335	0.131
bicycle	548	1064	0.296	0.148	0.141	0.0616
car	548	12500	0.703	0.752	0.765	0.534
van	548	1819	0.413	0.378	0.37	0.267
truck	548	718	0.451	0.35	0.352	0.244
tricycle	548	964	0.461	0.14	0.171	0.111
awnig-tricycle	548	486	0.275	0.121	0.0797	0.0555
bus	548	234	0.511	0.416	0.429	0.325
motor	548	4392	0.488	0.433	0.406	0.175

Tabla 6.2: Resultado Final del Entrenamiento

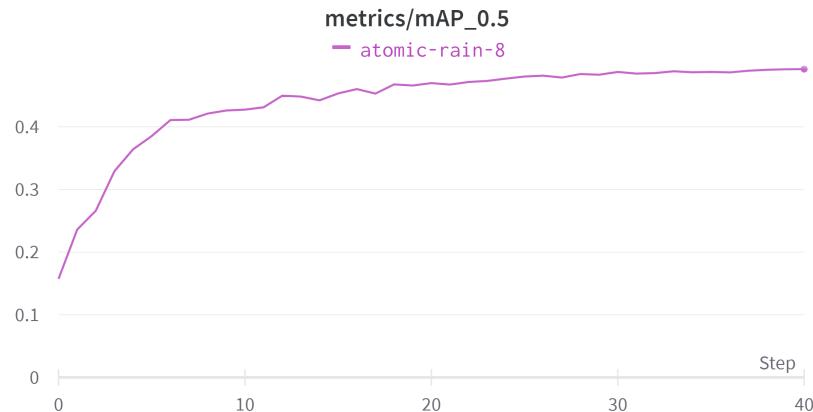


Figura 6.1: Gráfica mAP@0.5 de los Primeros Entrenamientos en NVIDIA DGX.

No obstante, aunque el mAP@0.5 fuera próximo al 50% dicho resultado se obtuvo con muy pocas épocas de entrenamiento para la gran cantidad de imágenes que se debían procesar y esto puede derivar en fallos, ya que puede darse el caso en el que la red no esté generalizando bien. Debido a esta razón se optó por buscar otros posibles cambios en el entrenamiento.

Los últimos entrenamientos realizados fueron en los que se hizo uso de los hiperparámetros del proyecto de investigación TPH-YOLO [45]. Dichos cambios fueron aplicados y fue posible realizar los entrenamientos durante más épocas sin que hubiera un sobreajuste. Pueden observarse dichos resultados en la Tabla 6.2. A primera vista pueden parecer resultados muy bajos, dado que el mAP@0.5 es simplemente de un 34.6 %.

A continuación, en las figuras 6.2 y 6.3 se muestran respectivamente la curva PR del modelo y la matriz de confusión del conjunto de validación usados.

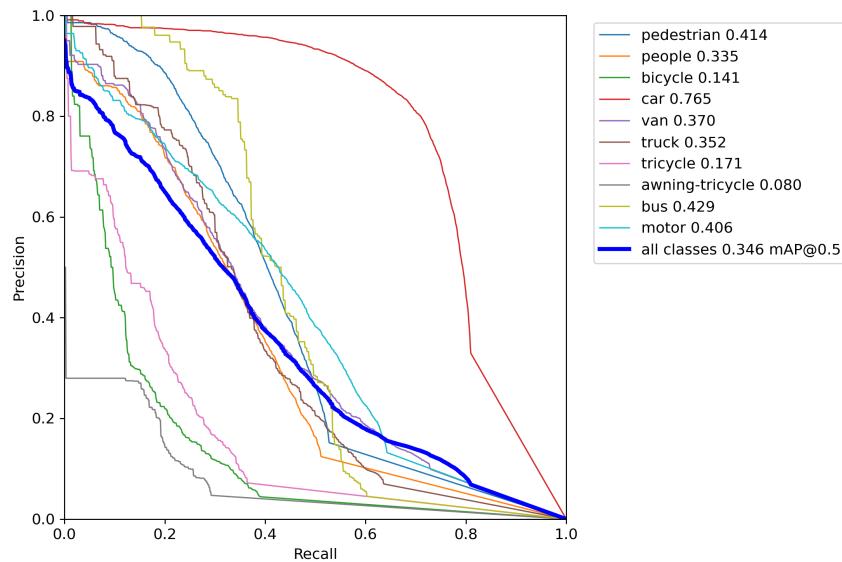


Figura 6.2: Curva PR del Entrenamiento Final.

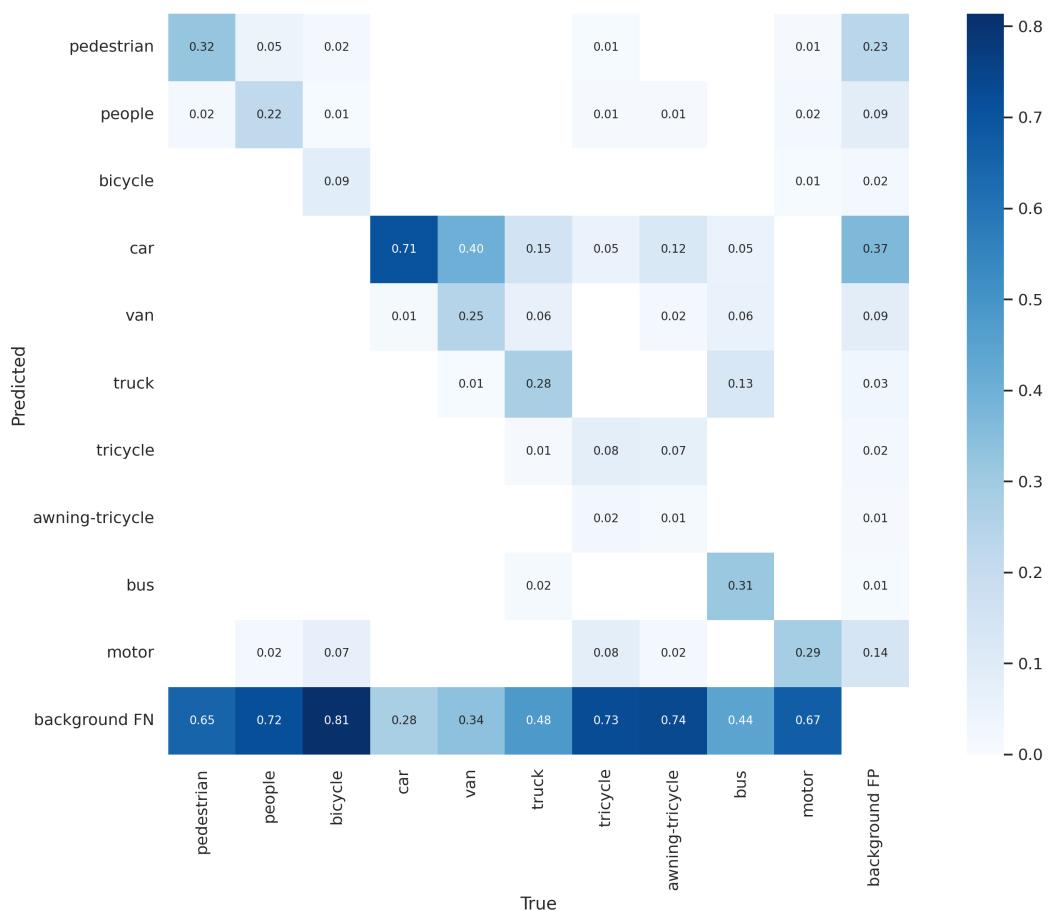


Figura 6.3: Matriz de Confusión del Entrenamiento Final de la Primera Fase.

Mirando en primer lugar la curva PR de la Figura 6.2, se puede ver que la clase que mejor reconoce es la del coche, ya que es la que mejor relación guarda en precisión y sensibilidad. Sin embargo, mientras que las mayoría de las clases guardan una relación precisión-sensibilidad media, hay tres que parecen bastante malas, siendo éstas el triciclo, la bicicleta y el triciclo con toldo. Esto mismo que se ve mirando las curvas PR de cada clase, se refleja en la matriz de confusión del conjunto de validación mostrada en la Figura 6.3, donde efectivamente el coche es la clase que mayor porcentaje de acierto tiene y los porcentajes más bajos corresponden al triciclo, la bicicleta y al triciclo con toldo.

Llegados a este punto, puede surgir la cuestión de cuál es la razón por la que se están obteniendo estos resultados. Para responder a esto simplemente tenemos que mirar la cantidad de etiquetas que hay asociadas a cada clase. En la Figura 6.4 queda representado la cantidad de etiquetas que hay asociadas a cada clase en todas las imágenes usadas durante el entrenamiento. Claramente se puede ver que la categoría que más veces aparece en las imágenes es la del coche. Esto implica que la red ha tenido más ejemplos de esta clase para estudiar y por esa razón tras el entrenamiento pudo detectar especialmente esa clase con un gran porcentaje de aciertos. Del mismo modo, de las clases que menos instancias se tienen, son las que menor porcentaje de acierto se tiene.

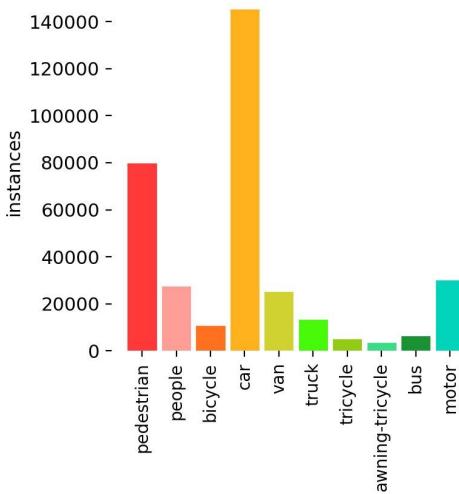


Figura 6.4: Gráfica de la Cantidad de Etiquetas de cada Clase del Conjunto de Entrenamiento VisDrone.

Puede parecer a primera vista que un mAP@0.5 del 34.6 % sea bajo, tal y como se puede ver en la Tabla 6.3, para ello se procede a verificar dichos resultados y poner a prueba la red con el conjunto de imágenes de *test*. Para ello, copiaremos al directorio *./data/images/* las imágenes de este conjunto por mayor comodidad, ya que por defecto al ejecutar el comando de la Figura 6.5 el código lee las imágenes de dicho directorio¹. Al igual que cuando se entrenaba la red, se tiene que pasar los pesos de la red ya entrenada.

```
$ python detect.py --weights best.pt
```

Figura 6.5: Detección de las imágenes de *test* de VisDrone.

Gracias a que el conjunto de *test* tiene gran variedad de imágenes podemos estudiar tres situaciones distintas, las cuales son imágenes aéreas durante el día, imágenes aéreas durante la noche e imágenes aéreas con perturbaciones.

¹Dado el caso en el que se quiera leer de otro directorio hay etiquetas que se pueden usar para que el código cambie el directorio de lectura.



Figura 6.6: Imágenes Diurnas VisDrone.



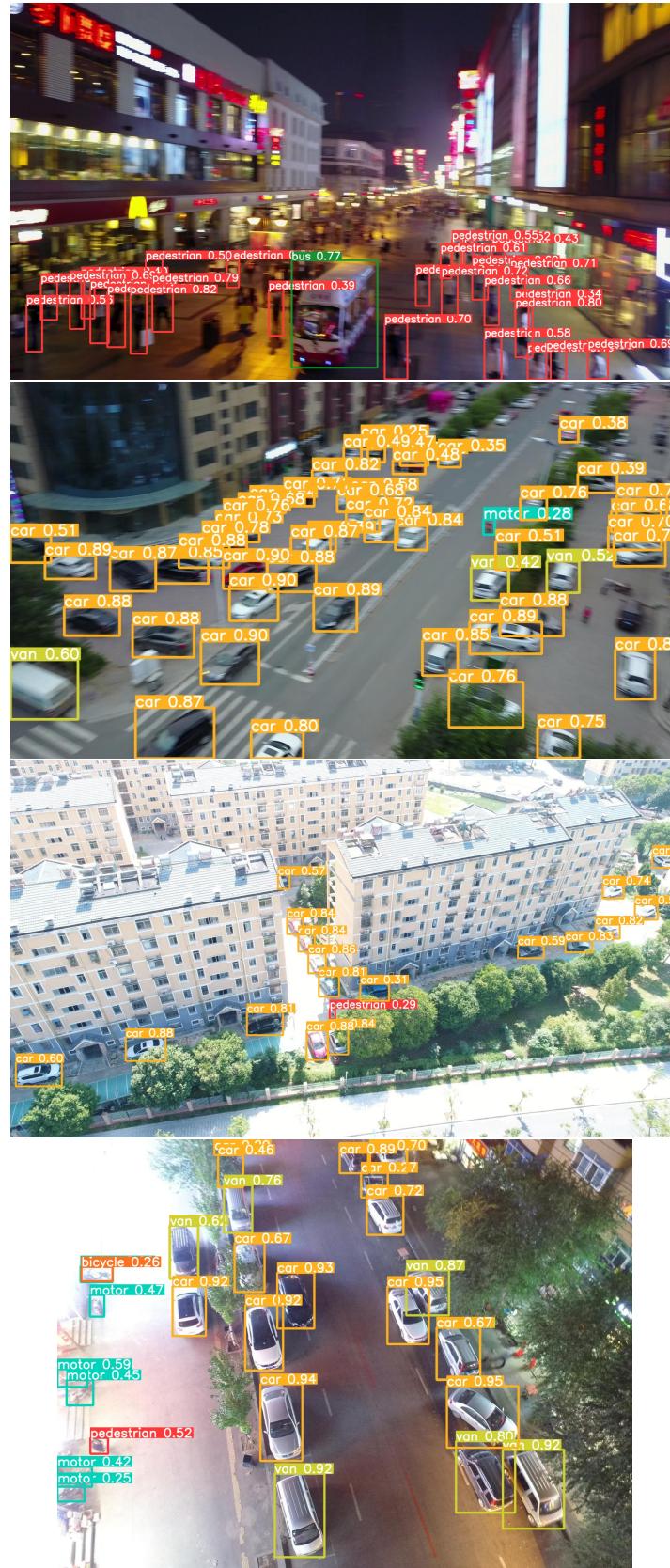


Figura 6.8: Imágenes Distorsionadas VisDrone.



OPTIMIZER	SGD	Adam
ÉPOCAS	130	130
Clase	mAP@0.5	mAP@0.5
all	0.609	0.533
car	0.429	0.269
person	0.796	0.739
smoke	0.596	0.541
debris	0.784	0.769
flood	0.578	0.359
emergency-vehicle	0.573	0.545
fire	0.504	0.506

Tabla 6.3: Comparativa de entrenamientos sin aumento de datos

Las detecciones, por defecto se realizan con un IOU del 45 %. En la Figura 6.6 se pueden ver una serie de imágenes representativas de imágenes tomadas durante el día. Queda bastante bien representado como es capaz de detectar ciertas clases con un gran grado de seguridad. Además, demuestras que en ocasiones reconoce instancias que incluso para el ojo humano son difíciles de percibir.

Por otro lado, en la Figura 6.7 se observan un serie de imágenes nocturnas. El nivel de confianza baja respecto a las imágenes diurnas, sin embargo teniendo en cuenta la poca iluminación de las imágenes, sorprende como es capaz de reconocer todas las instancias. Se demuestra así que la red es bastante robusta antes situaciones donde hay poca iluminación.

En último lugar, en la Figura 6.8 se muestran otra serie de imágenes, las cuales han sufrido ciertas distorsiones tales como pueden ser imágenes desenfocadas, tomadas durante un movimiento y con un alto contraste. A pesar de que las instancias a detectar no están representadas claramente en las imágenes la red sigue detectándolas con gran grado de confianza. De esta forma, queda mostrado otra vez cómo de robusta es la red ante ciertas perturbaciones. Siendo todas estas razones, los motivos por los cuales se decidió partir de esta red entrenada para la siguiente fase.

6.2. Resultados de la Segunda Fase

Esta segunda fase consistía en la adaptación de la red preentrenada con VisDrone a entornos de catástrofe. Para ello, se hizo uso de un dataset personalizado con imágenes del dataset AIDER y UMA-SAR. Esta fase se dividió en dos partes, una primera donde únicamente se hacía uso del dataset sin la aplicación de la técnica de aumento de datos y otra segunda parte dónde sí se hizo uso de ésta.

Durante los entrenamientos sin aumento de dataset se terminaron haciendo dos entrenamiento de 130 época comparando los cambios que producían dos optimizadores distintos en los resultados. En la Figura 6.9 se pueden ver dos gráficas comparando ambos entrenamientos.

La curva verde corresponde al entrenamiento con el optimizador Adam, mientras que la curva roja al realizado con SGD. Se puede observar que en este caso en el que se usa el optimizador SGD se consigue que el error sea mucho más pequeño con la misma cantidad de épocas, es decir llega a converger antes.

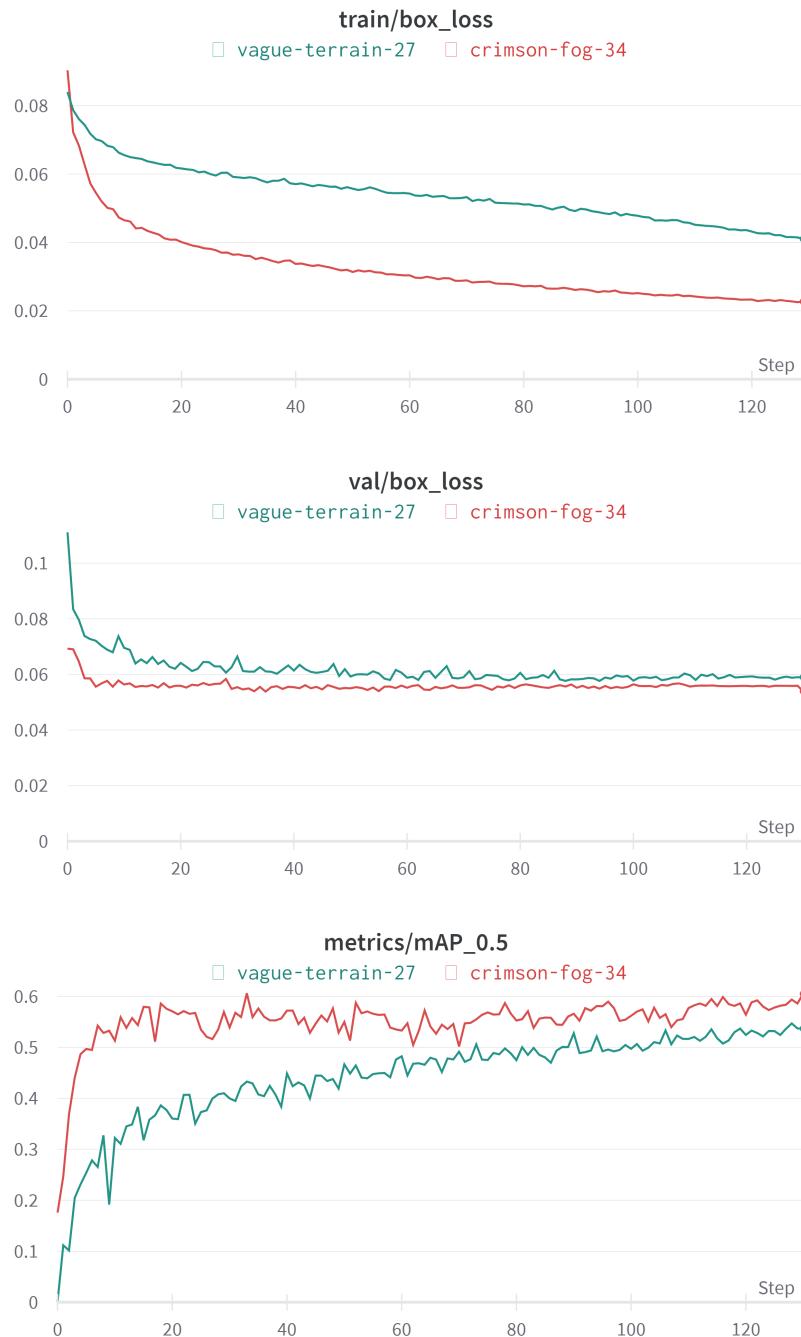


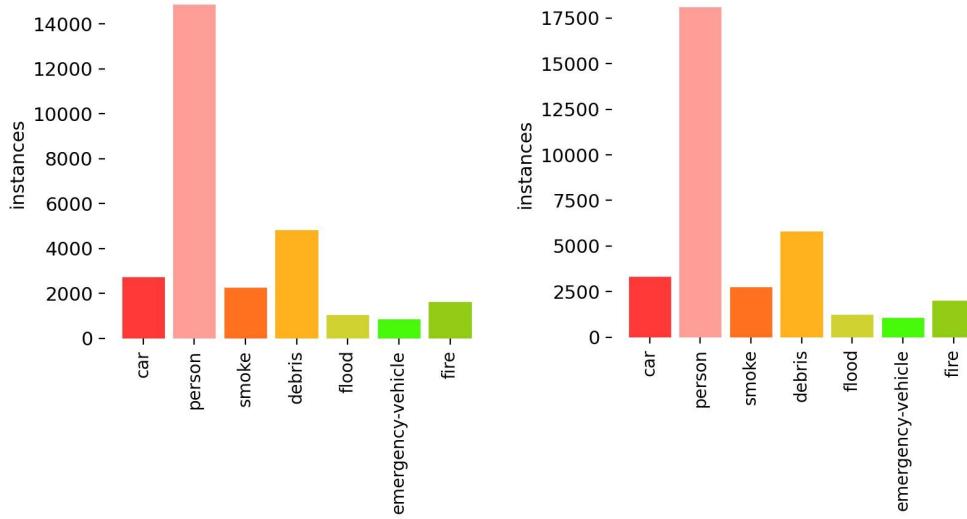
Figura 6.9: Gráficas Comparativas de Entrenamientos sin Aumento de Datos .

Dado que si se seguía entrenando no se obtenían resultados mejores, fue por eso por lo que se decidió aplicar la técnica del aumento de datos que ayuda a aumentar las métricas de las redes durante el entrenamiento.

En esta ocasión de decidió ver qué era lo que pasaba entrenando la misma cantidad de épocas habiendo aumentado el número de imágenes. Efectivamente, como era de esperar las métricas subieron, tal y como se puede ver en la Tabla 6.4. Entonces, se optó por seguir entrenando durante varias épocas más. Al entrenar durante 150 épocas las estadísticas aumentaron ligeramente. En este caso fue también con el optimizador SGD con el que se consiguieron mejores resultados.

OPTIMIZER	SGD		Adam	
ÉPOCAS	130	150	130	150
Clase	mAP@0.5	mAP@0.5	mAP@0.5	mAP@0.5
all	0.6	0.608	0.531	0.545
car	0.412	0.358	0.264	0.27
person	0.855	0.822	0.735	0.709
smoke	0.697	0.55	0.532	0.481
debris	0.802	0.814	0.75	0.76
flood	0.204	0.665	0.374	0.534
emergency-vehicle	0.75	0.543	0.556	0.523
fire	0.482	0.507	0.508	0.538

Tabla 6.4: Comparativa de entrenamientos con aumento de datos



(a) Cantidad de Etiquetas Sin Aumento de Datos (b) Cantidad de Etiquetas Con Aumento de Datos

Figura 6.10: Comparativa de Cantidad de Etiquetas

Se ha conseguido obtener un mAP@0.5 medio superior a 0.6 en el entrenamiento de 150 épocas usando el optimizador SGD. Se muestra en las figuras 6.11 y 6.12 la curva PR del entrenamiento y la matriz de confusión del conjunto de validación.

Analizando la curva PR se ve que la gran mayoría de clases a detectar tienen una buena relación precisión-sensibilidad, lo cual es muy buena señal, ya que aunque se sean clases bastante difíciles de detectar, esta red es capaz de hacerlo con una buena precisión. No obstante, puede llamar la atención que la clase que peor relación precisión-sensibilidad es la del coche. Esto se debe a que vuelve a ocurrir que el coche es una de las clases con menos instancias, tal y como se puede ver en la Figura 6.10.

En la matriz de confusión de la Figura 6.12, la gran parte de los valores están concentrados en la diagonal principal. Esto implica que hay muy pocos falsos positivos, es decir, entre todas las imágenes de validación no han habido gran cantidad de situaciones en la que la red se confunda al detectar la clase. En la Sección 7.1, se entrará en más detalle en las posibles mejoras que se pueden aplicar para obtener resultados aun mejores.

A continuación se van a analizar las imágenes de validación y estudiar las diferencias que hay entre las etiquetas de las imágenes que se han hecho a mano y las generadas por la red.

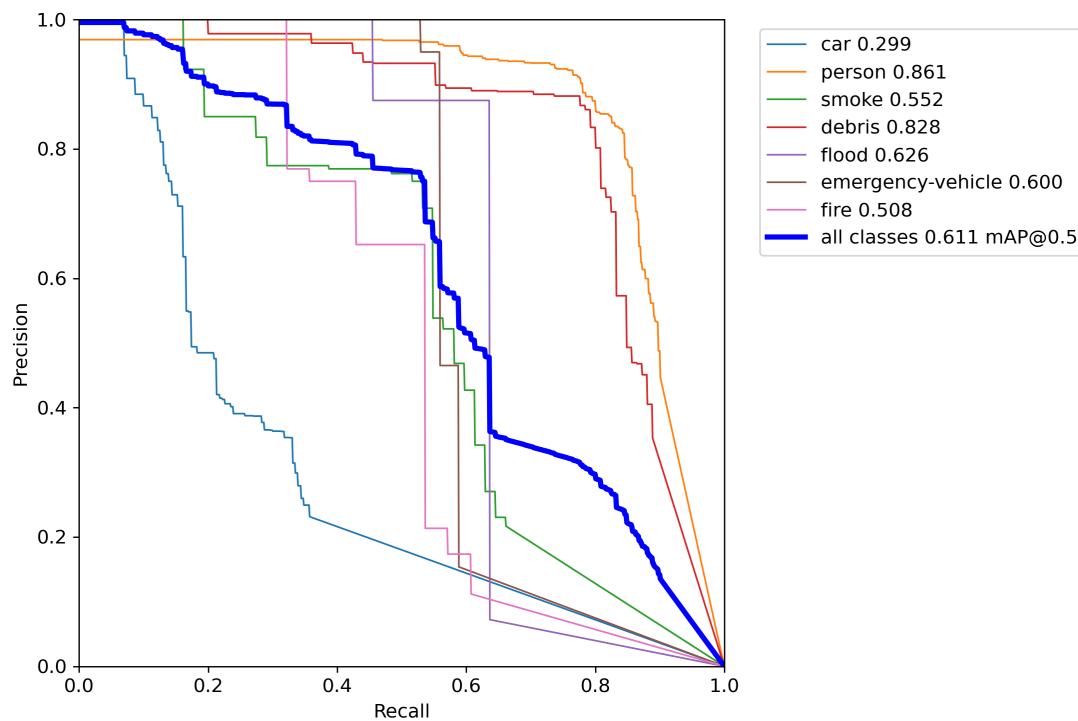


Figura 6.11: Curva PR de la Red Neuronal Convolucional Final.

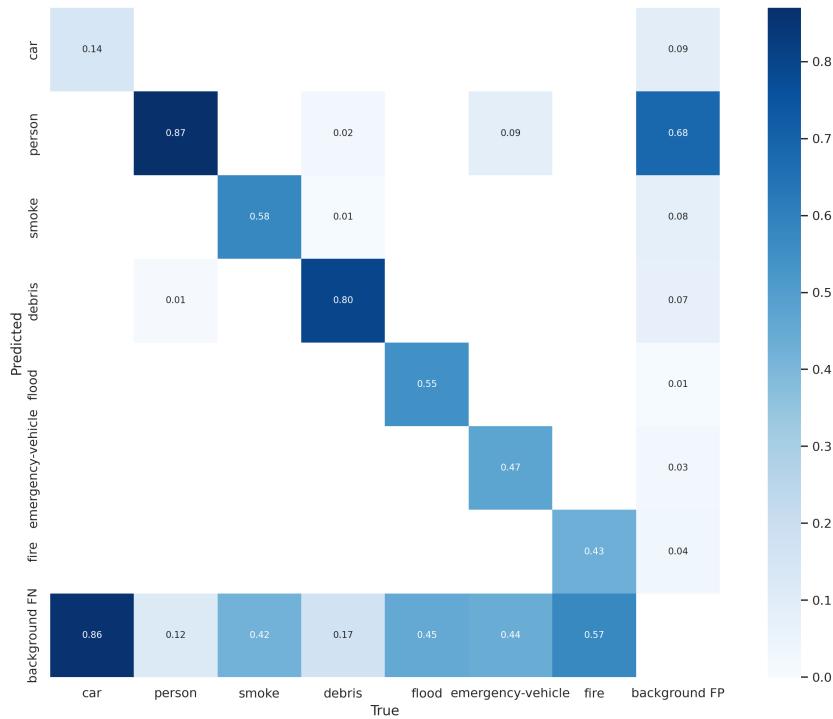
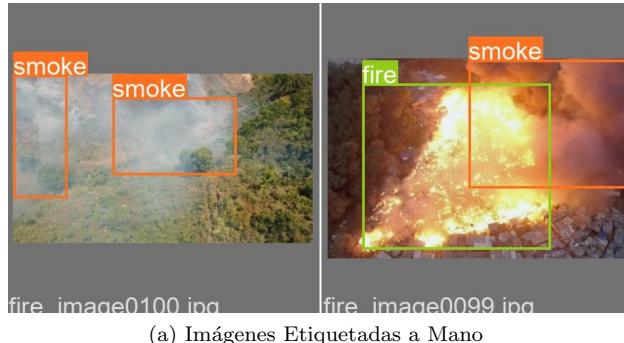


Figura 6.12: Matriz de Confusión de la Red Neuronal Convolucional Final.



(a) Imágenes Etiquetadas a Mano



(b) Imágenes tras la Detección

Figura 6.13: Comparativa de Detección

En la Figura 6.13 se pueden ver dos ejemplos de las imágenes de validación. En concreto, en la Figura 6.13a se muestran las etiquetas puestas a mano en las imágenes y en la Figura 6.13b son las mismas imágenes tras ser procesadas por la red neuronal que se ha entrenado. Estas dos son casos claros de porqué se están obteniendo mAP@0.5 bajos. En efecto se están detectando el fuego y el humo, pero las *bounding boxes* generadas son bastante diferentes a las que se marcaron a mano. Por lo tanto, eso hace que el IOU no sea tan alto haciendo que el mAP@0.5 baje. Esto desemboca también en que los porcentajes de fiabilidad puedan ser bajos. Si se bajara el IOU requerido durante el entrenamiento el mAP subiría.

Ahora se va a proceder a verificar las imágenes del conjunto de *test*. Para ello sólo hay que repetir el mismo procedimiento que en la primera fase, es decir, usar el archivo de pesos de la red entrenada en esta fase y poner las imágenes a procesar en el directorio `./data/images/`. Aunque en esta ocasión, a fin de hacer uso de más funciones que nos ofrece el código para hacer la detección, se van a usar las etiquetas `-iou-thres` y `-conf-thres` que nos permiten modificar el umbral de IOU y fiabilidad respectivamente. Se ha usado un valor de 0.25 para el umbral de IOU y 0.15 para el de fiabilidad, respecto al 0.45 y 0.25 respectivamente que se usan por defecto.

Para aplicar la detección se usará el comando que se muestra en la Figura 6.14, donde efectivamente se le pasan los pesos y se le indican los niveles de IOU y fiabilidad que se quiere usar.

```
$ python detect.py --weights best.pt --iou-thres 0.25
--conf-thres 0.15
```

Figura 6.14: Detección de las imágenes de test.

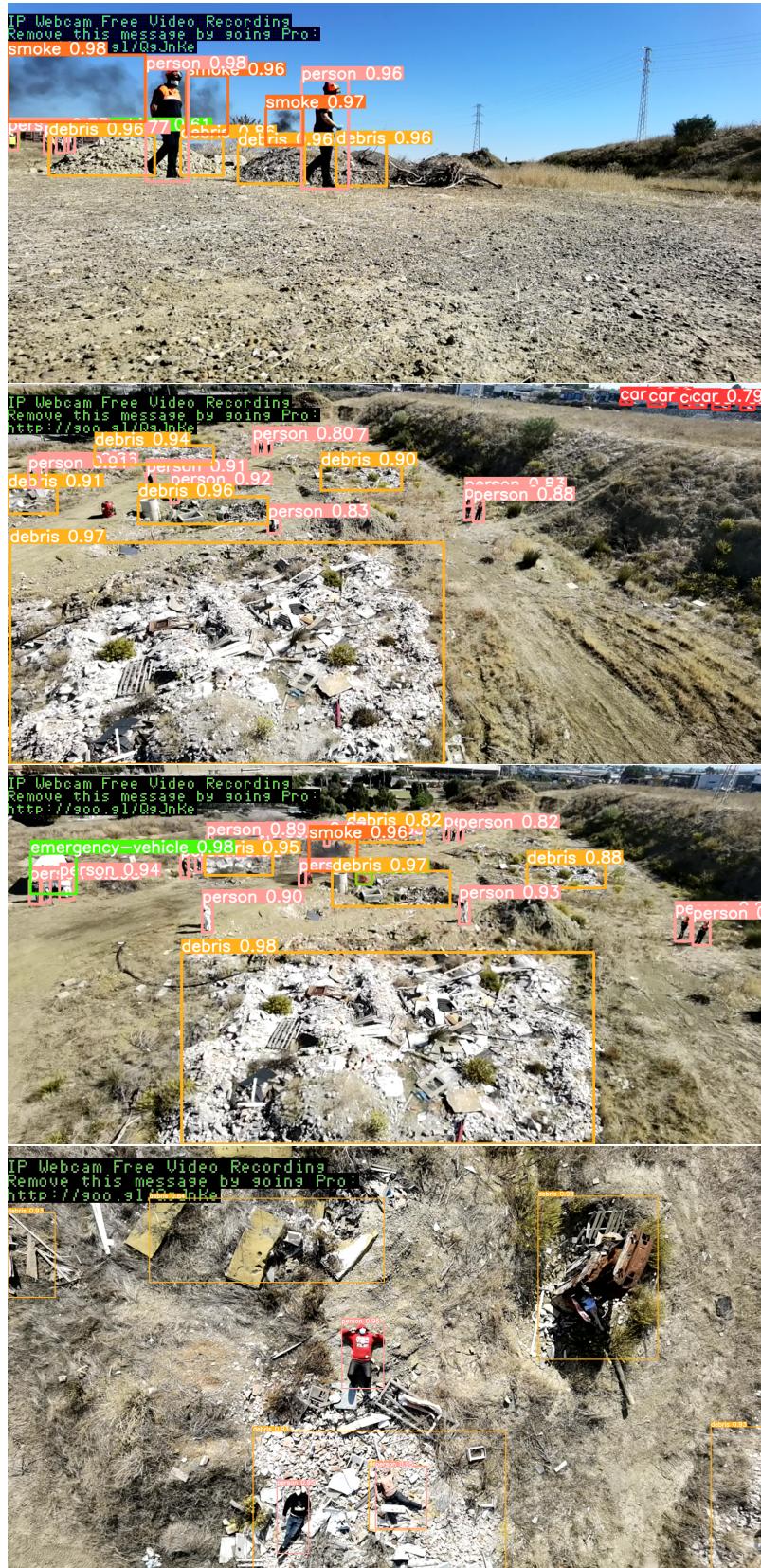


Figura 6.15: Imágenes de test UMA-SAR.



Figura 6.16: Imágenes de test UMA-SAR.



La Figura 6.15 muestra unos ejemplos de las imágenes del dataset UMA-SAR usadas para el conjunto de *test*. En ellas se pueden ver como la red es capaz de detectar todas las clases sin problema, a excepción de las inundaciones, ya que no hay instancias de ésta en este dataset. Es destacable también el hecho de que la red haya sido capaz de detectar una serie de coches que pasan por la autovía del fondo del entorno, los cuales son casi imperceptibles para el ojo humano. La red es capaz de reconocer las zonas con escombros con bastante certeza, incluso es capaz de reconocer víctimas sobre éstos. Mirando todas las imágenes podemos decir que es capaz de detectar todas clases con gran nivel de fiabilidad.

Por otro lado, en la Figura 6.16 se pueden observar otra serie de imágenes, pero en esta ocasión pertenecientes al conjunto extraído del dataset AIDER para las imágenes de *test*. Con este conjunto de imágenes se puede volver a observar la capacidad de red detectando zonas catastróficas muy diferentes unas de otras, lo cual indica que ha sido capaz de generalizar lo suficiente para adaptarse a una gran cantidad de situaciones. Quizás la clase más difícil de detectar es la de la inundación, ya que suelen ser zonas con muy pocas características y pueden parecer que detectar agua es difícil. Sin embargo, ha sido capaz de detectar este tipo de zonas.

Habiendo analizado los conjuntos de *test* y las métricas obtenidas, se puede decir que se ha conseguido entrenar una red que es capaz de desenvolverse en entornos de desastre reconociendo el tipo de entorno catastrófico en el que se encuentra, siendo este el objetivo de este Trabajo de Fin de Estudios.

Por último, cabe destacar que este Trabajo Fin de Grado está disponible en un repositorio de GitHub de libre acceso. [47]



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

El objetivo de este Trabajo Fin de Estudios era el empleo de una red neuronal convolucional para detectar clases de interés tal y como podrían ser diferentes tipos de zonas afectadas por catástrofes naturales grabadas por un vehículo aéreo no tripulado, haciendo uso de una arquitectura de una red pre-entrenada, ya existente y adaptarla a nuestro problema de estudio. Las conclusiones que se han extraído al analizar este TFG se presentan a continuación.

Redes Neuronales Convolucionales para Imágenes Aéreas

Desde el comienzo del proyecto, se viene comentando que detectar cualquier tipo de objeto desde el aire es mucho más difícil de hacerlo desde tierra. Dichos objetos disminuyen el tamaño cuánto más alto se encuentre nuestro vehículo aéreo esto hace que para la red neuronal convolucional que se esté usando le resulte mucho más difícil desarrollar los filtros necesarios para llevar a cabo la detección.

Además de los filtros, a lo hora de la detección interviene también el IOU. Poniendo como ejemplo la clase de persona de este proyecto, en la gran mayoría de casos, se podían encontrar en cuestión de 10 píxeles de la imagen. Cuando la red iba a realizar la detección debía generar la *bounding box* con un margen de error muy pequeño.

Detección de Zonas de Catástrofe

Entrenar una red para llevar a cabo la detección de zonas catastróficas es realmente complejo, dado que los desastres naturales son fenómenos aleatorios y nunca poseen una estructura determinada. En el caso de la detección de los escombros, éstos pueden encontrarse en infinitas posiciones y en gran cantidad de entornos. Debido a esto, fue muy importante buscar una gran cantidad de entornos distintos, para poder hacer que la red fuera capaz de generalizar y desenvolverse en cualquier tipo de entorno.

Influencia de los Parámetros de Entrenamiento

Durante ambas fases de entrenamiento se llegó a los resultados finales a base de ir modificando las épocas de entrenamiento el número del lote y los hiperparámetros. Es muy importante guardar un equilibrio entre el número de épocas de entrenamiento y las métricas finales.

En la primera fase de entrenamiento, durante los primeros entrenamientos se consiguió un mAP@0.5 mayor que al que se consiguió en el entrenamiento final. Sin embargo, aunque la red tuviera un mAP@0.5 más alto, éste se obtuvo entrenando durante muy pocas épocas de entrenamiento. Esto puede derivar en que la red no sea capaz de generalizar lo suficientemente bien.

El uso de hiperparámetros fue clave para llegar al resultado final. Al finalizar los primeros entrenamientos no era posible de conseguir mejores resultados, hasta que se decidió modificar los hiperparámetros. Haciendo un buen uso de éstos es posible mejorar los resultados finales y conseguir retrasar el sobreajuste de la red pudiendo entrenar la red durante más épocas.



Influencia de los Optimizadores durante el Entrenamiento

Durante la segunda fase de entrenamiento, los entrenamientos se basaron en la comparación de los resultados entre el optimizador SGD y Adam. Por lo general siempre se recomienda usar Adam, ya que éste está construido en base a muchos otros optimizadores que le precede, por lo tanto cabe esperar que su rendimiento sea mayor. Todos los optimizadores en mayor o menor medida están basados en SGD y en ocasiones no es el más eficiente. No obstante, no hay una regla fija que diga el optimizador que haya que usar en cada situación.

Puede ser que el optimizador SGD no sea el más eficiente, sin embargo se ha visto en los resultados de la segunda fase que es el que más rápido converge para nuestro conjunto de datos y el que mejor resultados nos proporciona. Esto se debe a que el conjunto de datos de esta fase no muy grande y generalmente éste optimizador no funciona bien cuando se trabaja con grandes conjuntos de datos e imágenes con una gran cantidad de píxeles. Es debido a esto, por lo que no siempre se debe usar Adam, es importante comparar resultados ya que no siempre se van a obtener los mejores resultados con Adam.

7.2. Trabajo Futuro

Habiendo cumplido todos los objetivos propuestos en el proyecto, se proponen ciertas líneas de desarrollo para la continuación de este trabajo:

- Obtención de un mayor número de imágenes en otros tipos de entornos y el etiquetado de las mismas, para aumentar el tamaño del dataset actual. Por ejemplo, se pueden buscar otros datasets relacionados con este caso de estudio, como puede ser el de ISBDA¹ y hacer una selección de imágenes para enriquecer el dataset personalizado de este TFG.
- Implementación de este desarrollo en un vehículo aéreo no tripulado para su uso en ambientes de rescate reales en tiempo real.
- Extender y profundizar en los resultados obtenidos en este TFG con el objetivo de publicar un artículo de investigación.

¹Consiste en vídeos aéreos generados por usuarios de redes sociales daños en edificios. Este dataset ayuda a hacer un estudio sobre el nivel de daño que han sufrido los edificios tras algún fenómeno natural.[48]

Apéndice A

DOCKER

A.1. Introducción

Antes de empezar a introducirnos en el mundo de los contenedores debemos quedarnos con tres frases importantes sobre diferentes elementos de DOCKER:

- *Dockerfiles get built.*
- *Docker Images get shipped.*
- *Containers are run.*

A.2. Arquitectura DOCKER

DOCKER y todo el proceso de contenerización gira en torno a tres componentes principales:

Docker Client: Es la máquina o medio a través del cual nosotros como usuarios interactuamos con Docker.

Encontramos dos formas de comunicación:

- Docker CLI: Command Line Interface (Interfaz de línea de comandos).
- Docker API: Application Program Interface (Interfaz de programa de aplicación)

Los comandos se pueden utilizar directamente desde el terminal del cliente mientras que las API se pueden utilizar para que algunas aplicaciones hablen con Docker.

Docker Host: Es la máquina que realmente realiza la tarea de contenerización. Ejecuta un programa o trozo de software llamado "Docker Daemon" que escucha y realiza las acciones solicitadas por el cliente Docker. El Docker Daemon construye un archivo Docker y lo convierte en una imagen Docker. Las imágenes Docker también pueden ejecutarse como contenedores. Los contenedores pueden comunicarse con el Docker Daemon a través de las imágenes Docker. En otras palabras, cualquier cambio realizado en el contenedor también se refleja en la imagen Docker temporalmente.

Docker Registry: El componente más sencillo. Lugar donde se almacenan las imágenes Docker y se ponen a disposición de los demás.

El Docker Client es el que pasa la solicitud a través de Docker CLI y APIs y recibe los resultados que se muestran. Luego, tenemos Docker Host que también ejecuta el Docker Daemon y trabaja con imágenes y contenedores Docker. Por último, Docker Registry actúa como un lugar universal para acceder a las imágenes Docker disponibles.



A.3. Formato de un DockerFile

Es una secuencia de instrucciones destinadas a ser procesadas por el Docker Daemon. Es la forma principal de interactuar con Docker. El orden de las secuencias en el fichero es importante. Cada instrucción crea una capa. Las capas pueden ser mandadas a caché y reutilizadas por Docker. Si dos DockerFiles van a utilizar la misma capa en algún momento, el demonio Docker puede simplemente reutilizar la capa creada para tales fines. Es un archivo sin extensión alguna, simplemente se crea este archivo usando cualquier editor de texto y lo llamamos "Dockerfile". Dentro del DockerFile podemos encontrar instrucciones para ser pasadas. Las instrucciones se pueden dividir generalmente en tres categorías: instrucciones fundamentales, de configuración y de ejecución.

Algunas instrucciones fundamentales que se pueden utilizar en los DockerFiles son:

- **ARG**: Utilizado para definir los argumentos utilizados por las instrucciones FROM. Aunque, no es necesario usar el comando ARG y no usarlo no causa ningún daño a la imagen resultante directamente a veces ayuda a omitir parámetros como las versiones bajo control.
- **FROM**: Utilizado para especificar la imagen base para la imagen docker resultante que pretendemos crear. DEBE estar ahí en cualquier DockerFile y la única instrucción que se puede escribir antes de ella es ARG. normalmente este argumento va seguido de una imagen del sistema operativo o de una imagen de aplicación que está disponible públicamente en DockerHub.

Para construir el DockerFile:

```
1 docker build -t nombre-imagen .
```

Donde "-t" se utiliza para que el Dockerfile sea fácilmente reconocible. En este caso lo vamos a etiquetar como nombre-imagen y el punto al final dirige a Docker al Dockerfile almacenado en el directorio actual.

Para verificar si la imagen está construida o no podemos ejecutar el comando:

```
1 docker images
```

Algunas instrucciones de configuración son:

- RUN pide a Docker que ejecute el comando mencionado encima de la imagen base y los resultados se consignan como una capa separada encima de la capa de la imagen base.
- ENV configura las variables de entorno podemos tener como variables de entorno USER, SHELL y LOGNAME

Si queremos ejecutar esta imagen como contenedor:

```
1 docker run -itd --name nombre-contenedor nombre-imgen:version
```

Donde -itd significa interactive, teletype y detached respectivamente.

Le pasamos al Docker Daemon el nombre que le queremos asignar al contenedor nombre-contenedor y la imagen que queremos ejecutar.

Después de ejecutar este comando Docker nos proporcionará un id de contenedor único.

Usando el siguiente comando, podemos ver los contenedores en ejecución:

```
1 docker ps -a
```

Para ejecutar el comando "bash":

```
1 docker exec -it cont_run-env bash
```



Gracias a este *flag*, le estamos diciendo a Docker que ejecute el contenedor en segundo plano. Al utilizar este comando, estamos trayendo el contenedor en ejecución al primer plano.

Una vez dentro podemos ver las variables de entorno utilizando:

```
1 echo $USER / $SHELL / $LOGNAME$
```

Cuando queramos salir del entorno Docker, podemos simplemente ejecutar el comando

```
1 exit
```

A.4. Comando EXPOSE

&& significa que para ejecutar el siguiente comando el primero debe ser un éxito.

EXPOSE es un tipo de información que se le pasará a Docker sobre el puerto en el que el contenedor está escuchando. No publica el puerto, pero llena el vacío entre el constructor de la imagen de Docker y la persona que ejecuta el contenedor. Para ejecutar contenedores con una instrucción expose debemos utilizar el comando:

```
1 docker run -itd -rm --name nombre_contenedor -p 8080:80 ←  
      nombre_imagen
```

El *flag* '-rm' eliminará automáticamente el contenedor una vez que se haya detenido.

'-p 8080:80' significa mapear el puerto 80 del contenedor con el puerto 8080 del host.

Si queremos que otro contenedor sirva en el puerto 8080 del host debemos liberar el puerto usando el comando:

```
1 docker container stop nombre_contenedor
```

A.5. Docker Images

Una imagen es una pila de múltiples capas creadas a partir de instrucciones de Dockerfile. Cada capa, a excepción de la superior, es de tipo *Read-Only*. La capa superior es de tipo *Read-Write*. Pueden ser reconocidas por su nombre o por su ID de imagen.

Se pueden hacer *push* o *pull* desde el Docker Hub.

Este comando nos permite buscar imágenes docker en el Docker hub:

```
1 docker search nombre_imagen:version
```

Se pueden establecer filtros:

```
1 docker search nombre_imagen -f "is-official=true"
```

Se puede formatear el resultado de la búsqueda:

```
1 docker search --format "table {{.Name}} {{.Description}} {{.←  
    IsOfficial}}" nombre_imagen
```

Se puede listar todas las imágenes que se tiene del mismo repositorio:



```
1 docker images nombre_imagen
```

Incluso podemos especificar la versión que buscamos:

```
1 docker images nombre_imagen:version
```

Se puede extraer una imagen especificada desde el Docker Hub a nuestro Docker Host:

```
1 docker image pull nombre_imagen:version
2 docker image pull --all-tags nombre_imagen
```

En el Docker Hub se puede crear un repositorio y empujar las imágenes a él:

```
1 docker pull usuario/nombre_repositorio
```

Para entrar en el Docker Hub se usa el comando, después del cual se pedirá el nombre de usuario y la contraseña:

```
1 docker login
```

Otros comandos interesantes pueden ser:

```
1 docker tag nginx:latest nombre-de-usuario:custom-tag
2 docker image push usuario/nombre-repo:custom-tag
3 docker image inspect ubuntu:latest
4 docker image inspect --format "{{.RepoTags}} : {{.RepoDigests}}" ←
    ubuntu:latest
5 docker image inspect --format "{{json .Config}}" ubuntu > inspect←
    -report-ubuntu.txt
```

Este comando nos mostrará todas las capas intermedias de una imagen:

```
1 docker history
2 docker image history ubuntu
```

Para liberar espacio en la máquina anfitriona:

```
1 docker image rm nginx:1-alpine-perl
2 docker rmi image_id
```

Dado el caso en el que se tenga algunos repositorios con el mismo id:

```
1 docker rmi image_id -force
```

A.6. Contenedores

Instancia de ejecución de una imagen Docker. Proporciona un aislamiento similar al de las Máquinas Virtuales, pero más ligero... MUCHO MÁS LIGERO. Añade una capa de escritura encima de las capas de la imagen y trabaja sobre ella. Puede hablar con otros contenedores como procesos en Linux. Utiliza Copy-on-Write

Al ejecutar un contenedor, nos referimos a que estamos proporcionando recursos como la computación, la memoria y el almacenamiento.



```
1 docker container create -it --name cc_busybox_A busybox:latest
2 docker container run -itd --rm --name cc_busybox_B busybox:latest
```

Para ejecutar el contenedor cc_busybox_A y parar el contenedor cc_busybox_B por ejemplo:

```
1 docker container start cc_busybox_A
2 docker stop cc_busybox_B
```

Para reiniciar y renombrar un contenedor:

```
1 docker container restart --time 5 cc_busybox_A
2 \textbf{ }
3 docker container rename cc_busybox_A my-busybox
```

Se puede acceder a un contenedor utilizando el comando:

```
1 docker attach mi_busybox
```

De este modo se accederá a la terminal de este contenedor. Sin embargo, si se ejecuta el comando *exit*, se detendrá nuestro contenedor.

El comando *docker exec* nos permite utilizar cualquier comando que queramos, y lo ejecuta en el contenedor.

```
1 docker container start mi_busybox
2 docker exec -it mi_busybox pwd
```

A.6.1. Mapeo de puertos en los contenedores

Se especifican los puertos que se quieren mapear:

```
1 docker container run -itd --name cont_nginx -p 8080:80/tcp ←
    img_expose
```

Permitiendo a Docker portar los mapas por sí mismo:

```
1 docker container run -itd --name cont_nginx -P img_expose
```

Se puede visualizar el mapeo ejecutando el comando:

```
1 docker container port cont_nginx
```

A.6.2. Borrar contenedores

Para borrar contenedores se usará el comando:

```
1 docker container rm cont_from
```

Se puede borrar el contenedor usando su id en lugar de su nombre:

```
1 docker container rm 33626589e12
```

Para que el contenedor sea borrado con éxito, éste debería estar parado, pero si se da el caso en el que esté en ejecución:



```
1 docker container rm 33626589e12 --force
```

No obstante, si está funcionando pero queremos eliminarlo de forma segura:

```
1 docker container kill --signal=SIGTERM cont_nginx
```

Se pueden eliminar todos los contenedores parados a la vez:

```
1 docker container prune
```

A.7. Networking

Los contenedores pueden comunicarse uno a uno, uno a muchos o muchos a muchos.

En el caso de los contenedores docker, estas comunicaciones son gestionadas por objetos llamados controladores de red. Un controlador de red es una pieza de software que permite la conexión en red de los contenedores. Son responsables de invocar una red dentro del host o dentro del cluster. Proporciona un controlador de red nativo que se envía con el motor Docker.

Docker también soporta controladores de red remotos que son desarrollados por terceros y pueden ser instalados como plugins.

Aparte de los controladores de red, docker también proporciona IPAM o drivers de gestión de direcciones IP que maneja la dirección IP y las distribuciones si no son especificadas por el administrador.

A.8. Almacenamiento

Los datos necesitan ser respaldados en algún lugar como un almacenamiento permanente y una pregunta rápida que nos surgiría sería, qué o cuáles son los datos que deben ser respaldados.

Tenemos dos tipos de capas, las de "sólo lectura", que tienen datos permanentes y nunca se modifican, y las de "lectura-escritura", que contienen datos temporales o volátiles. Si un contenedor se detiene o muere, los datos volátiles desaparecen. Por lo tanto, tenemos que hacer una copia de seguridad de los datos de esta última capa. Podemos guardarla en nuestro Docker Host, en otro servidor o en la Nube.

Existe un tipo de objeto para respaldar estos datos, que se llama *Docker Volume*.

Está completamente aislado del sistema de archivos del host. Aunque los datos del volumen están ordenados en un directorio específico del host es controlado por la línea de comandos de Docker. Los volúmenes son más seguros de enviar y más fiables de operar que cualquier otro tipo de almacenamiento.

Los volúmenes son objetos de almacenamiento de docker que se montan en los contenedores. Son directorios dedicados en el sistema de archivos del host.

Si una aplicación en contenedor se envía junto con el volumen, las personas, aparte del propio desarrollador que utiliza la aplicación, acabará creando dicho directorio en sus propios hosts docker. El contenedor proporciona datos al motor docker y el usuario proporciona comandos para almacenar los datos en el volumen o para gestionar los datos en el mismo. Aunque, lo que el contenedor conoce es sólo el nombre del volumen, no la ruta en el host. La traducción tiene lugar en las máquinas Docker y, por lo tanto, las aplicaciones externas que tengan acceso a los contenedores no tendrán medios para acceder a los volúmenes directamente. Este aislamiento mantiene la integridad y la seguridad de los hosts y los contenedores.

La segunda opción son los *bind-mounts*. El intercambio de información es bastante similar, aparte del hecho de que en lugar de crear un directorio inspirado en el nombre del volumen, los bind-mounts nos permiten utilizar cualquier directorio en el host docker para almacenar los datos. También expone la ubicación de almacenamiento del contenedor, lo que puede hacer mella en la seguridad general de la aplicación o del propio host.



Por último, tenemos los *tmpfs* o sistema de archivos temporales. Los volúmenes y el bind-mount permiten compartir archivos entre la máquina anfitriona y el contenedor para poder hacer que los datos persistan incluso después de que el contenedor se detenga. Si se utiliza Docker en Linux, se puede encontrar una tercera opción, *tmpfs mounts*. El contenedor puede crear archivos fuera de la capa de escritura del contenedor. A diferencia de los volúmenes y los bind-mounts, un *tmpfs mount* es temporal y sólo persiste en la memoria del host, no en el almacenamiento interno. Cuando el contenedor se detiene, el *tmpfs mount* se elimina y los archivos escritos allí no se mantienen. El único caso de uso sensato es el de almacenar archivos sensibles que no quieras que persistan una vez que la aplicación se elimine. Los *tmpfs mounts* pueden ser creados pero no enviados y no funcionarán en entornos no-Linux como Mac o Windows.

Para crear un volumen, se puede utilizar el comando :

```
1 docker volume create vol_busybox
```

El siguiente comando significa que se va a montar el volumen 'vol_ubuntu' en el contenedor ubuntu en el directorio tmp:

```
1 docker run -d --volume vol_ubuntu:/tmp ubuntu
```

Para listar todos los volúmenes:

```
1 docker volume ls
```

El siguiente comando permitirá filtrar todos los contenedores que no tengan asociado ningún volumen:

```
1 docker volume ls --filter "dangling=true"
```

Si se da el caso en el que se quiere saber información relativa al volumen, se puede usar el comando:

```
1 docker volume inspect vol_ubuntu
```

Por último, también se puede eliminar un volumen:

```
1 docker volume rm vol_ubuntu
```

A.8.1. ¿Cómo añadir un volumen a contenedor existente?

Esta fue una cuestión que surgió durante el proyecto. Para resolver este problema se aplicaron los siguientes comandos:

```
1 docker stop old_container_id
2 docker ps -a
3 docker commit old_container_id new_image_name
4 docker run -it -v /some/dir/host/:/some/dir/container/ --name ←
    new_container_name (+ any other flag like --gpu) new_image_name
```

En resumen, lo que hacen estos comandos es crear una imagen a partir del contenedor que tenemos y una vez obtenida dicha imagen podemos crear el contenedor que se quería con los parámetros necesarios.



A.8.2. Aumentar el Tamaño del Contenedor

Durante el trabajo ocurrió que el contenedor que se creaba no tenía mucha capacidad y era necesario aumentar el tamaño de éste. Por lo tanto lo que hubo que hacer fue aplicar el proceso de la Sección A.8.1 y ejecutar el contenedor con la etiqueta *-shm-size*:

```
1 docker run -it --name new_container_name --shm-size=2gb ←  
    new_image_name  
2 docker inspect new_container_name | grep -i shm
```

Apéndice B

Códigos

B.1. Convertir tipo de Archivo

```
1  """
2  Created on Fri Jul  8 20:24:59 2022
3  @author: Rubén González Navarro
4  """
5
6  from PIL import Image
7  from os import listdir
8  import os
9  from os.path import splitext
10
11 target_directory = '.'
12 target = '.png'
13 delete = '.jpg'
14
15 for file in listdir(target_directory):
16     filename, extension = splitext(file)
17
18     try:
19         if extension not in ['.py', target]:
20             im = Image.open(filename + extension)
21             im.save(filename + target)
22
23             rmvFile = filename + delete
24             os.remove(rmvFile)
25     except OSError:
26         print('Cannot convert %s' % file)
```

Figura B.1: Código para Convertir a otros Tipos de Imágenes.



B.2. Renombrar Archivos

```
1      """
2      Created on Mon Jun 27 18:20:26 2022
3      @author: Rubén González Navarro
4      """
5
6      import os
7
8      folder = 'C:/Users/Usuario/Desktop/Facultad/Cuarto GIERM/Trabajo Fin de Grado/←
9          Etiquetado/train_data/train/labels/'
10     # count increase by 1 in each iteration
11     # iterate all files from a directory
12     for file_name in os.listdir(folder):
13         # Build old file name
14         source = folder + file_name
15
16         # Adding the counter to the new file name and extension
17         destination = folder + "datasetDRON_" + str(count) + ".txt"
18
19         # Renaming the file
20         os.rename(source, destination)
21         count += 1
22     print('All Files Renamed')
23
24     print('New Names are')
25     # verify the result
26     res = os.listdir(folder)
27     print(res)
```

Figura B.2: Código para Renombrar Imágenes.



B.3. Aumento de Datos

```
1 """
2 Created on Tue Jul 12 11:58:57 2022
3 @author: Rubén González Navarro
4 """
5
6 import albumentations as A
7 import os
8 import cv2
9 import sys
10 from numpy import genfromtxt
11 import numpy as np
12
13 import time
14
15 def getTransform(loop):
16     if loop == 0:
17         transform = A.Compose([
18             A.HorizontalFlip(p=1),
19             A.BboxParams(format='yolo', min_visibility = 0.1)])
20     elif loop == 1:
21         transform = A.Compose([
22             A.augmentations.transforms.ColorJitter(saturation = 0.3, p=1),
23             A.BboxParams(format='yolo', min_visibility = 0.1)])
24     elif loop == 2:
25         transform = A.Compose([
26             A.augmentations.transforms.ColorJitter(contrast = 0.2, p=1),
27             A.BboxParams(format='yolo', min_visibility = 0.1)])
28     elif loop == 3:
29         transform = A.Compose([
30             A.augmentations.Rotate(limit = 98, interpolation = 1,
31             border_mode=cv2.BORDER_CONSTANT, value = None, mask_value = None,
32             always_apply = False, p=1),
33             A.BboxParams(format='yolo', min_visibility = 0.1)])
34     elif loop == 4:
35         transform = A.Compose([
36             A.augmentations.RandomScale(scale_limit = 0.2,
37             interpolation=1, always_apply=False, p=1),
38             A.BboxParams(format='yolo', min_visibility = 0.1)])
39     elif loop == 5:
40         transform = A.Compose([
41             A.VerticalFlip(p=1),
42             A.BboxParams(format='yolo', min_visibility = 0.1)])
43     elif loop == 6:
44         transform = A.Compose([
45             A.RandomScale(scale_limit = 0.2, interpolation=1, always_apply=False, p=1),
46             A.augmentations.Rotate(limit = 98, interpolation = 1,
47             border_mode=cv2.BORDER_CONSTANT, value = None, mask_value = None,
48             always_apply = False, p=1),
49             A.BboxParams(format='yolo', min_visibility = 0.1)])
50     elif loop == 7:
51         transform = A.Compose([
52             A.augmentations.transforms.GlassBlur(sigma=0.7, max_delta=4,
53             iterations=2, always_apply=False, mode='fast', p=0.5),
54             A.augmentations.transforms.GaussianBlur(blur_limit=(3, 7),
55             sigma_limit=0, always_apply=False, p=0.5),
56             A.BboxParams(format='yolo', min_visibility = 0.1)])
57
58     return transform
59
60 def readFilesinFolder(path):
61     lista=[]
62
63     for filename in os.listdir(path):
64
65         if filename.endswith('.jpg') or filename.endswith('.png'):
66             title, ext = os.path.splitext(os.path.basename(filename))
67             lista.append(title+ext)
```



```
56     return lista
57
58 def start():
59
60     labels = ['car', 'person', 'smoke', 'debris', 'flood', 'emergency-vehicle', 'fire']
61
62     files = readFilesinFolder(originFolder)
63
64     for k in range(8):
65         transform = getTransform(k)
66
67     for j in files:
68         try:
69             image = cv2.imread(j) #image = cv2.imread(j+'.jpg')
70             #image = np.array(image)
71             tmp = j[:-4]
72             bboxes = genfromtxt(originPath+tmp+'.txt', dtype = str, delimiter=' ')
73
74             #bboxes = bboxes.tolist()
75
76             tmp3 = [0.0, 0.0, 0.0, 0.0, 0.0]
77             tmp2 = []
78             for bbox in bboxes:
79                 '''
80                 for x in len(bbox):
81                     tmp3.append(float(bbox[x]))
82                 bboxes.append(tmp3)
83                 tmp3.clear()
84                 '''
85                 #for i in range(bbox):
86                 #    #ESTA CONVERSIÓN NO SE ESTA HACIENDO!!!!!!
87                 tmp3[0] = float(bbox[0])
88                 tmp3[1] = float(bbox[1])
89                 tmp3[2] = float(bbox[2])
90                 tmp3[3] = float(bbox[3])
91                 tmp3[4] = float(bbox[4])
92                 tmp2.append(tmp3)
93                 tmp3 = [0.0, 0.0, 0.0, 0.0, 0.0]
94
95             for bbox in tmp2:
96                 if(bbox[0]==0.0):
97                     bbox.append(labels[0])
98                     bbox.pop(0)
99                 elif(bbox[0]==1.0):
100                     bbox.append(labels[1])
101                     bbox.pop(0)
102                 elif(bbox[0]==2.0):
103                     bbox.append(labels[2])
104                     bbox.pop(0)
105                 elif(bbox[0]==3.0):
106                     bbox.append(labels[3])
107                     bbox.pop(0)
108                 elif(bbox[0]==4.0):
109                     bbox.append(labels[4])
110                     bbox.pop(0)
111                 elif(bbox[0]==5.0):
112                     bbox.append(labels[5])
113                     bbox.pop(0)
114                 elif(bbox[0]==6.0):
115                     bbox.append(labels[6])
116                     bbox.pop(0)
117
118                 transformed = transform(image=image, bboxes=tmp2)
119                 transformed_image = transformed['image']
120                 transformed_bboxes = transformed['bboxes']
121                 with open(destPath+"train/labels/"+str(k)+"_"+tmp.replace("frame", "")+"_"+tmp+".txt", 'w') as f:
122                     for x in transformed_bboxes:
123                         if (x[-1]=="car"):
```

```

124         f.write("0 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
125             round(x[2],6), round(x[3],6)))
126     elif (x[-1]== "person"):
127         f.write("1 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
128             round(x[2],6), round(x[3],6)))
129     elif (x[-1]== "smoke"):
130         f.write("2 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
131             round(x[2],6), round(x[3],6)))
132     elif (x[-1]== "debris"):
133         f.write("3 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
134             round(x[2],6), round(x[3],6)))
135     elif (x[-1]== "flood"):
136         f.write("4 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
137             round(x[2],6), round(x[3],6)))
138     elif (x[-1]== "emergency-vehicle"):
139         f.write("5 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
140             round(x[2],6), round(x[3],6)))
141     elif (x[-1]== "fire"):
142         f.write("6 %s %s %s\n" % (round(x[0],6), round(x[1],6),←
143             round(x[2],6), round(x[3],6)))
144
145     cv2.imwrite(destPath+"train/images/"+str(k)+"_"+tmp+j[-4:], ←
146         transformed_image)
147     except:
148         print("FAILED TO READ IMAGE: "+j)
149         print(sys.exc_info())
150
151     originPath = "C:/Users/Usuario/Desktop/Facultad/Cuarto GIERM/Trabajo Fin de Grado/←
152         Etiquetado/train_data/train/labels/"
153     destPath = "C:/Users/Usuario/Desktop/Facultad/Cuarto GIERM/Trabajo Fin de Grado/←
154         Etiquetado/train_data_augmented/"
155     originFolder= "C:/Users/Usuario/Desktop/Facultad/Cuarto GIERM/Trabajo Fin de Grado/←
156         Etiquetado/train_data/train/images/"
157
158     files=[]
159     if __name__ == "__main__":
160         start()

```

Figura B.3: Código para aplicar el Aumento de Datos.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Apéndice C

Herramientas de Etiquetado y Edición

C.1. Herramientas de Etiquetado

Para llevar a cabo el etiquetado se hizo uso de la herramienta *labelImg*, la cual es una herramienta bastante utilizada y de código abierto que de consiguió desde el repositorio de GitHub de su creador [42]. En el caso de este proyecto, se hizo uso de Anaconda3 para ejecutar la herramienta. Para ello se usaron los comandos de la Figura C.1.

```
$1 git clone https://github.com/heartexlabs/labelImg  
$2 cd labelImg  
$3 python labelImg.py
```

Figura C.1: Ejecutar la aplicación LabelImg.

Al abrir la aplicación nos aparecerá el entorno gráfico donde se va a trabajar, el cual se muestra en la Figura C.2. En el panel de la izquierda se encuentran las opciones para elegir los directorios donde se encuentran las imágenes y para elegir el directorio donde queremos que se guarden las etiquetas. Además, se proporciona al usuario unas opciones para ir cambiando de imágenes. Por último y quizás lo más relevante, la opción del formato en el que queremos que se escriban nuestras etiquetas, que para el caso de este proyecto es formato YOLO.

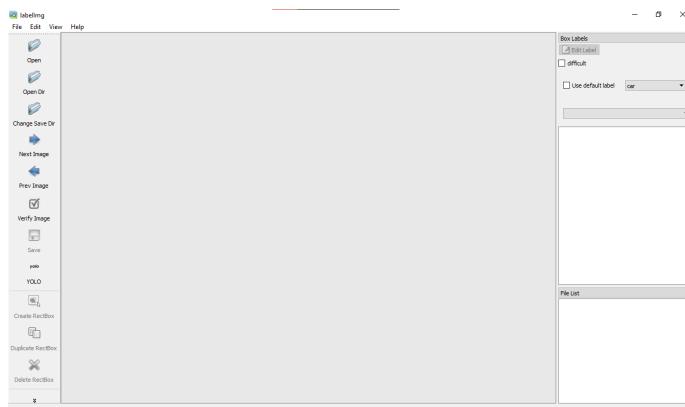


Figura C.2: Interfaz Gráfica de LabelImg.

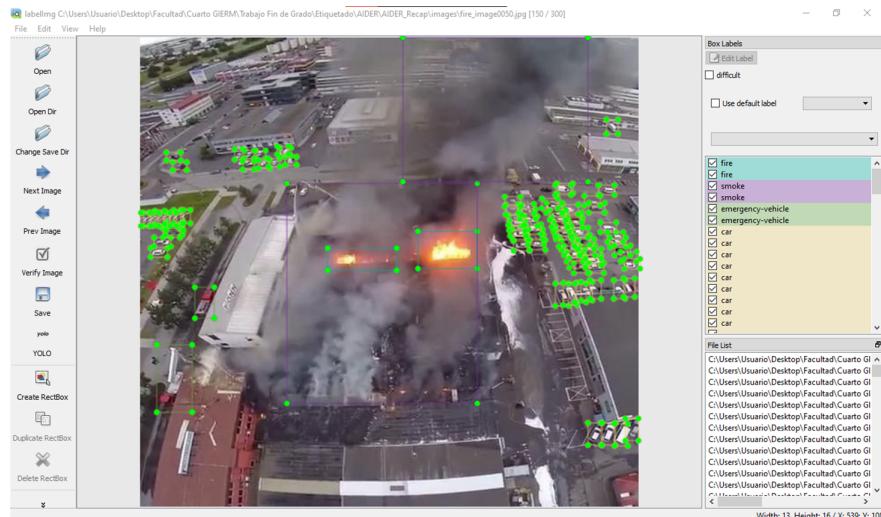


Figura C.3: Ejemplo de Etiquetas con LabelImg.

Por otro lado, en el panel de la derecha nos encontramos con unos paneles de ayuda, los cuales ayudaran a organizar todas las etiquetas que se pongan en las imágenes y a visualizar todas las imágenes de un directorio.

El etiquetado se realizó con una tableta gráfica la cual simplificaba mucho el etiquetado de las imágenes, ya que para poner una etiqueta se debe pulsar y arrastrar en la imagen mientras se dibuja la *bounding box*. Al terminar de dibujarla, aparecerá una ventana emergente indicando la clase que se quiere asignar a dicha *bounding box*. En la Figura C.3.

C.2. Herramienta de Edición

El dataset UMA-SAR que había disponible para este proyecto estaba en formato de vídeo, por ello a fin de poder utilizarlos para extraer imágenes de ellos se decidió separar todos los fotogramas de componían el vídeo, para ello se usó la herramienta de edición *Reproductor VLC*. Para ello se seguirán los siguientes pasos:

- En primer lugar, lo que se debe hacer es crear un directorio donde se quiera guardar todos los fotogramas.
- Una vez creado, se deberá copiar la ruta al directorio.
- A continuación, se abrirá el reproductor VLC y seleccionaremos la opción *Herramientas* → *Preferencias*.
- Aparecerá una ventana, donde se deberá seleccionar la opción *Mostrar Ajustes* → *Todos*. Dicha opción se encuentra en la esquina inferior izquierda da la ventana.
- La ventana cambiará, mostrando más opciones. En el panel izquierdo habrá que dirigirse a la opción *Vídeos* → *Filtros* → *Filtros de Escena*. Al hacer click, aparecerán una serie de opciones. En la opción *Prefijo de la ruta de carpeta*, pondremos la ruta que se copio en el segundo paso.
- El último paso es dirigirse a *Vídeos* → *Filtros* y marcar la opción de Filtro de Escena. Hacer click en Guardar.

Una vez realizados todos estos pasos, lo único que queda por hacer es empezar a reproducir el vídeo y todos los fotogramas se empezarán a guardar en el directorio creado a medida que el vídeo avance.



NOTA: No olvidar desmarcar la opción del último paso una vez finalizada la extracción, ya que si se reproducen otros vídeos seguirán guardándose los fotogramas de éste y puede llegar a llenar la memoria del ordenador.



UNIVERSIDAD
DE MÁLAGA

E•• ESCUELA DE
INGENIERÍAS
INDUSTRIALES

Bibliografía

- [1] Adrián Bañuls, Anthony Madow, Ricardo Vázquez-Martín, Jesús Morales y Alfonso García-Cerezo. “Object detection from thermal infrared and visible light cameras in search and rescue scenes”. En: (2020), págs. 380-386.
- [2] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [3] Sebastian Raschka y Vahid Mirjalili. *Python Machine Learning*. third. Packt Publishing, 2019.
- [4] Benjamin Planche y Eliot Andres. *Hands-On Computer Vision with TensorFlow 2*. first. Packt Publishing, 2019.
- [5] Geoffrey Currie y Eric Rohren. “Intelligent Imaging in Nuclear Medicine: the Principles of Artificial Intelligence, Machine Learning and Deep Learning”. En: *Seminars in Nuclear Medicine* 51 (2 mar. de 2021), págs. 102-111. ISSN: 15584623. DOI: 10.1053/j.semnuclmed.2020.08.002.
- [6] Warren S Mcculloch y Walter Pitts. *A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY** n. 1990, págs. 99-115.
- [7] Ulises Castro Peñaloza, Nun Pitalua-Diaz, Jose Ruz-Hernandez y Ruben Lagunas-Jimenez. *Introducción a los Sistemas Inteligentes*. Dic. de 2009. ISBN: ISBN-UABC: 978-607-7782-17-9. ISBN-UNISON: 978-607-7753-47-6.
- [8] Nicolás Alonso Benjamin Cicerchia Leonardo Esnaola Claudia Russo Hugo Ramón y Juan Pablo Tessore. *Tratamiento Masivo de Datos Utilizando Técnicas de Machine Learning*. <http://repositorio.unnoba.edu.ar:8080/xmlui/handle/23601/107>. 2016.
- [9] A. Núñez Reiz, M. A. Armengol de la Hoz y M. Sánchez García. *Big Data Analysis and Machine Learning in Intensive Care Units*. Oct. de 2019. DOI: 10.1016/j.medin.2018.10.007.
- [10] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan y Stephen Marshall. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. En: (nov. de 2018). URL: <http://arxiv.org/abs/1811.03378>.
- [11] Dingjun Yu, Hanli Wang, Peiqiu Chen y Zhihua Wei. “Mixed Pooling for Convolutional Neural Networks”. En: oct. de 2014, págs. 364-375. ISBN: 978-3-319-11739-3. DOI: 10.1007/978-3-319-11740-9_34.
- [12] Gunand Mayanglambam. “Deep Learning Optimizers”. En: (nov. de 2020). URL: <https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f>.
- [13] Jose Martínez Heras. “Análisis de Errores en Machine Learning”. En: (sep. de 2020). URL: https://www.iartificial.net/analisis-de-errores-en-machine-learning/#Causas_del_overfitting_sobreajuste.
- [14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever y Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. En: *Journal of Machine Learning Research* 15.56 (2014), págs. 1929-1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html>.
- [15] Standford Vision y Learning Lab. “CS231n Convolutional Neural Networks for Visual Recognition”. En: (2022). URL: <https://cs231n.github.io/transfer-learning/>.
- [16] Luke Taylor y Geoff Nitschke. “Improving Deep Learning with Generic Data Augmentation”. En: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2018, págs. 1542-1547. DOI: 10.1109/SSCI.2018.8628742.



- [17] ultralytics. *YOLOv5*. <https://github.com/ultralytics/yolov5.git>. Jun. de 2020.
- [18] Do Thuan. "Evolution of Yolo algorithm and Yolov5: The State-of-the-Art object detection algorithm". En: (2021). URL: <https://www.theseus.fi/handle/10024/452552>.
- [19] Iason Katsamenis, Eleni Karolou, Agapi Davradou, Eftychios Protopapadakis, Anastasios Doulamis, Nikolaos Doulamis y Dimitris Kalogerias. *TraCon: A novel dataset for real-time traffic cones detection using deep learning*. Mayo de 2022. DOI: 10.48550/arXiv.2205.11830.
- [20] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen y Jun-Wei Hsieh. "CSPNet: A New Backbone that can Enhance Learning Capability of CNN". En: *CoRR* abs/1911.11929 (2019). arXiv: 1911.11929. URL: <http://arxiv.org/abs/1911.11929>.
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". En: *CoRR* abs/1406.4729 (2014). arXiv: 1406.4729. URL: <http://arxiv.org/abs/1406.4729>.
- [22] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi y Jiaya Jia. "Path Aggregation Network for Instance Segmentation". En: *CoRR* abs/1803.01534 (2018). arXiv: 1803.01534. URL: <http://arxiv.org/abs/1803.01534>.
- [23] NVIDIA Corporation. *NVIDIA TESLA V100 GPU ARCHITECTURE THE WORLD'S MOST ADVANCED DATA CENTER GPU*. 2017. URL: <https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf>.
- [24] Nikhil Ketkar y Jojo Moolayil. *Deep Learning with Python*. Apress, 2021. DOI: 10.1007/978-1-4842-5364-9.
- [25] *Matplotlib*. 2022. URL: <https://pypi.org/project/matplotlib/>.
- [26] NumPy Developers. *NumPy*. 2022. URL: <https://numpy.org/doc/stable/>.
- [27] doxygen. *OpenCV*. 2021. URL: https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html.
- [28] Fredrik Lundh. *Pillow*. 2022. URL: <https://pillow.readthedocs.io/en/stable/>.
- [29] Nitz Mahone. *PyYAML*. 2022. URL: <https://pyyaml.org/>.
- [30] The SciPy community. *SciPy*. 2022. URL: https://docs.scipy.org/doc/scipy/getting_started.html.
- [31] PyTorch Contributors. *PyTorch*. 2022. URL: <https://pytorch.org/docs/stable/index.html>.
- [32] Torch Contributors. *TorchVision*. 2017. URL: <https://pytorch.org/vision/stable/index.html>.
- [33] Casper Da Costa-Luis, Stephen Karl Larroque, Kyle Altendorf, Hadrien Mary, Richard Sheridan, Mikhail Korobov, Noam Raphael, Ivan Ivanov, Marcel Bargull, Nishant Rodrigues, Guangshuo Chen, Antony Lee, Charles Newey, , James, JC, Martin Zugnoni, Matthew D. Pagel, Mjstevens777, Mikhail Dekt-yarev, Alex Rothberg, , Alexander, Daniel Panteleit, Fabian Dill, FichteFoll, Gregor Sturm, HeoHeo, Hugo Van Kemenade, Jack McCracken y MapleCCC. *tqdm: A fast, Extensible Progress Bar for Python and CLI*. 2022. DOI: 10.5281/ZENODO.595120. URL: <https://zenodo.org/record/595120>.
- [34] Tensorflow Community. *TensorBoard*. 2022. URL: <https://www.tensorflow.org/tensorboard?hl=es-419>.
- [35] Michael Waskom. *Seaborn*. 2021. URL: <https://seaborn.pydata.org/>.
- [36] Jeff Reback, Jbrockmendel, Wes McKinney, Joris Van Den Bossche, Matthew Roeschke, Tom Augspurger, Simon Hawkins, Phillip Cloud, Gfyoung, Sinhrks, Patrick Hoefler, Adam Klein, Terji Petersen, Jeff Tratner, Chang She, William Ayd, Shahar Naveh, JHM Darbyshire, Richard Shadrach, Marc Garcia, Jeremy Schendel, Andy Hayden, Daniel Saxton, Marco Edward Gorelli, Fangchen Li, Torsten Wörtwein, Matthew Zeitlin, Vytautas Jancauskas, Ali McMaster y Thomas Li. *pandas-dev/pandas: Pandas 1.4.3*. 2022. DOI: 10.5281/ZENODO.3509134. URL: <https://zenodo.org/record/3509134>.
- [37] Ligeng Zhu. *THOP: PyTorch-OpCounter*. Jul. de 2022. URL: <https://github.com/Lyken17/pytorch-OpCounter/>.
- [38] Kenneth Reitz. *Request*. Jul. de 2022. URL: <https://github.com/psf/requests>.
- [39] PRCV 2022. *Aerial-Ground Intelligent Unmanned System Environment Perception Challenge*. URL: <http://aiskyeye.com/>.



- [40] Christos Kyrkou. *AIDER: Aerial Image Database for Emergency Response applications*. 2020. URL: <https://github.com/ckyrkou/AIDER>.
- [41] Jesús Morales, Ricardo Vázquez-Martín, Anthony Mandow, David Morilla-Cabello y Alfonso García-Cerezo. “The UMA-SAR Dataset: Multimodal Data Collection from a Ground Vehicle During Outdoor Disaster Response Training Exercises”. En: *The International Journal of Robotics Research* 40.6-7 (2021), págs. 835-847. doi: 10.1177/02783649211004959.
- [42] Tzutalin Heartexlabs. *labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images*. URL: <https://github.com/heartexlabs/labelImg>.
- [43] Loïc Simon, Ryan Webster y Julien Rabin. “Revisiting Precision and Recall Definition for Generative Model Evaluation”. En: (mayo de 2019). URL: <http://arxiv.org/abs/1905.05441>.
- [44] VisDrone. *VisDrone/Visdrone-Dataset: The dataset for drone based detection and tracking is released, including both image/video, and annotations*. URL: <https://github.com/VisDrone/VisDrone-Dataset>.
- [45] Xingkui Zhu, Shuchang Lyu, Xu Wang y Qi Zhao. “TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios”. En: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, págs. 2778-2788.
- [46] Muzammal Naseer, Kanchana Ranasinghe, Salman H. Khan, Munawar Hayat, Fahad Shahbaz Khan y Ming-Hsuan Yang. “Intriguing Properties of Vision Transformers”. En: *CoRR* abs/2105.10497 (2021). arXiv: 2105.10497. URL: <https://arxiv.org/abs/2105.10497>.
- [47] Rubén González Navarro. *TFG Aerial Disaster*. 2022. URL: <https://github.com/ricardovmartin/TFGAerialDisaster>.
- [48] Xiaoyu Zhu, Junwei Liang y Alexander Hauptmann. *MSNet: A Multilevel Instance Segmentation Network for Natural Disaster Damage Assessment in Aerial Videos*. 2020. arXiv: 2006.16479 [cs.CV].