



**Universidad
Rey Juan Carlos**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Curso Académico 2022/2023

Trabajo Fin de Grado

**DETECCIÓN AUTOMÁTICA DE
ARMAS DE FUEGO EN IMÁGENES
USANDO REDES NEURONALES
PROFUNDAS**

Autor: Patricia Corral Sanz

Tutor: Ángel Sánchez Calle

Resumen

El aumento de la violencia con armas de fuego es un problema contemporáneo que atenta contra uno de los principales derechos humanos como es el derecho a la vida.

Son muchos los esfuerzos dedicados a la detección de armas con el fin de garantizar la seguridad ciudadana. Una de las soluciones propuestas hoy en día consiste en emplear algoritmos de visión artificial capaces de reconocer su presencia en imágenes o videos en tiempo real.

Este proyecto se ha elaborado con el objetivo de profundizar en la detección de armas de fuego y la clasificación de las mismas en armas largas y cortas con ayuda de redes neuronales profundas.

Para ello se han recopilado imágenes de armas de distintos datasets públicos y completado los datos para garantizar que se ajustan en la medida de lo posible al problema propuesto. El dataset generado contiene una gran variedad de armas largas (fusiles, escopetas...) y armas cortas (pistolas, revólveres).

Se ha seleccionado YOLOv5 como red neuronal convolucional para la detección de las armas de fuego. Por su parte, Roboflow es la herramienta de etiquetado y preprocesado de imágenes que se utiliza en este proyecto. La solución se ha implementado con código en Python y algunas librerías como Pytorch [22] o módulos como Glob.

En este trabajo se presenta una batería de experimentos en los que se estudia el desempeño de YOLOv5 a la hora de detectar armas y clasificarlas, y cómo las características de las imágenes afectan al algoritmo propuesto. Así, se inicia realizando una serie de pruebas que permitan construir un dataset adecuado para el problema presentado y posteriormente, se trabaja con dicho dataset variando parámetros como el ruido u occlusiones presentes en las imágenes y se estudia cómo estos afectan a los resultados.

En base a las conclusiones obtenidas se valorará y expondrá posibles futuros experimentos que puedan aportar resultados útiles y mejoras del trabajo realizado.

Palabras clave: detección de objetos, YOLOv5, red neuronal convolucional, Python, arma corta, arma larga, Deep Learning.

Summary

The increase in firearms violence is a contemporary problem that threatens one of the most important human rights, the right to life.

Many efforts have been devoted to the detection of weapons in order to ensure public safety. One of the solutions proposed today consists of using artificial vision algorithms capable of recognizing their presence in images or videos in real time.

This project has been developed with the aim of gaining more insight into the detection of firearms problem and their classification into long and short weapons with the help of deep neural networks.

For this purpose, gun images have been collected from different public datasets and the data have been completed to ensure that they fit as much as possible to the proposed problem. The generated dataset contains a wide variety of long guns (rifles, shotguns...) and short guns (pistols, revolvers).

YOLOv5 has been selected as the convolutional neural network for the detection of firearms. Roboflow is the image labeling and preprocessing tool used in this project. The solution has been implemented with Python code and some libraries such as Pytorch [22] or modules such as Glob.

In this work we present a battery of experiments in which it is studied the performance of YOLOv5 when detecting weapons and classifying them, and how the characteristics of the images affect the proposed algorithm. Thus, we start by performing a series of tests to build a suitable dataset for the presented problem and then, we work with this dataset by varying parameters such as noise or occlusions present in images and we study how these affect the results.

Based on the conclusions obtained, possible future experiments that can provide useful results and improvements of the work done will be evaluated and presented.

Keywords: object detection, YOLOv5, convolutional neural network, Python, short weapon, long weapon, Deep Learning

Agradecimientos

Ha sido mucho el esfuerzo dedicado y el tiempo consumido en la elaboración de este proyecto. Tiempo en el que he sido acompañada por familia, amigos, compañeros y profesores, que me han brindado su paciencia, ayuda y comprensión.

Quiero agradecerles a mis padres y familia su dedicación y apoyo incondicional, así como los valores inculcados desde que era pequeña. “En el trabajo, el esfuerzo y la preparación está el éxito”

A mis compañeros de clase y amigos, que me han acompañado en este largo viaje: David, Iván, Alicia, Claudia, David, Laura, Cristina, Franco, Adrián, Alba y Julián. Todos habéis tenido un papel importante en estos años de carrera que hemos vivido juntos. Sin vosotros no estaría escribiendo esto ahora mismo.

A Elisa, Olga, Noelia y Begoña. Sabéis lo importantes que habéis sido y que sois.

Muchas gracias a ti, Jesús, qué llegaste en el momento adecuado y has sabido apoyarme, respetarme y ayudarme en todo lo que estaba en tu mano.

Por último, una mención especial para mi tutor, Ángel Sánchez Calle, por haberme regalado su tiempo y conocimientos, siempre amablemente. Ha sido un verdadero placer poder realizar mi TFG contigo.

Índice

1. INTRODUCCIÓN	1
1.1 - MOTIVACIÓN DEL PROYECTO.....	1
1.2 - OBJETIVOS DEL PROYECTO	1
1.3 - ORGANIZACIÓN DE LA MEMORIA	2
2. MARCO TEÓRICO	2
2.1 - APRENDIZAJE PROFUNDO O <i>DEEP LEARNING</i>	2
2.1.1 – <i>Detección de objetos</i>	3
2.1.2 – <i>Redes Neuronales Convolucionales</i>	4
2.1.3 – <i>YOLO</i>	9
2.1.3.1 - <i>YOLOv5</i>	11
3. ESTADO DEL ARTE.....	14
4. SOLUCIÓN INFORMÁTICA	15
4.1 - OBTENCIÓN DE LOS DATOS	15
4.1.1 – <i>Tipos de datos</i>	15
4.1.2 – <i>Datasets</i>	16
4.1.3 – <i>Generación de imágenes sintéticas</i>	18
4.1.4 – <i>Anotación de imágenes. Roboflow</i>	19
4.1.5 – <i>Preprocesado de imágenes. Roboflow</i>	20
4.2 - SOLUCIÓN PROPUESTA	21
4.2.1 – <i>YOLOv5 en la nube. Google Colab</i>	21
4.2.2 – <i>Jerarquía de ficheros</i>	22
4.2.3 – <i>Entrenamiento del modelo</i>	23
4.2.4 – <i>Test del modelo: Detección de objetos en nuevas imágenes</i>	25
4.2.5 – <i>Evaluación del desempeño del modelo</i>	26
5. PRUEBAS	26
5.1 – MÉTRICAS DE EVALUACIÓN	26
5.2 – PRIMERAS PRUEBAS	30
5.3 – EXPERIMENTOS	30
5.3.1 - <i>Experimento 1</i>	30
5.3.2 - <i>Experimento 2</i>	36
5.3.3 - <i>Experimento 3</i>	41
5.3.4 - <i>Experimento 4</i>	46
6. CONCLUSIÓN.....	51
6.1 – CONCLUSIONES GENERALES	51
6.2 – TRABAJOS FUTUROS	52
7. REFERENCIAS	53
8. ANEXOS.....	56
8.1- IMPLEMENTACIÓN <i>INTERSECTION OVER UNION</i> PARA YOLOv5.....	56
8.2- YOLOv5 EN LOCAL.....	57
8.3- EJECUCIÓN DEL PROYECTO Y ACCESO A LOS DATOS EMPLEADOS.....	57
8.4- IMÁGENES DETECCIONES ARMAS CON YOLOv5	57

Índice figuras

FIGURA 1.ESQUEMA SUBCONJUNTOS DE LA INTELIGENCIA ARTIFICIAL.....	3
FIGURA 2. EQUERA MÉTODOS DETECCIÓN OBJETOS.....	4
FIGURA 3.REPRESENTACIÓN DIGITAL DE UNA IMAGEN.	5
FIGURA 4. ARQUITECTURA DE UNA CNN.....	5
FIGURA 5. EQUERA DE UNA CONVOLUCIÓN.	6
FIGURA 6. EJEMPLO FUNCIONAMIENTO MAX-POOLING.	6
FIGURA 7.EQUERA ILUSTRATIVO DETECCIÓN POR REGIONES.....	7
FIGURA 8. EQUERA FAST R-CNN.	8
FIGURA 9. EQUERA FASTER R-CNN.	8
FIGURA 10. COMPARATIVA DE VELOCIDAD. REDES CONVOLUCIONALES.	9
FIGURA 11. DETECCIÓN OBJETOS CON YOLO.....	10
FIGURA 12. COMPARATIVA ENTRE MODELOS DE YOLO.	11
FIGURA 13. APLICACIÓN DE CSPNET A RESNET Y DENSENET.....	12
FIGURA 14. ESTRUCTURA DE UNA RED CON SPATIAL PYRAMID POOLING.....	12
FIGURA 15. ARQUITECTURA DE YOLOv5.....	13
FIGURA 16. ECUACIONES CÁLCULO COORDENADAS BOUNDING BOXES EN YOLOv5.....	13
FIGURA 17. EQUERA GENERACIÓN TENSOR EN YOLOv5.	13
FIGURA 18. EJEMPLOS ARMAS A DETECTAR.	16
FIGURA 19. CLASES,TAMAÑO DE LAS IMÁGENES Y NÚMERO DE OBJETOS DEL DATASET DEFINITIVO.....	17
FIGURA 20. EJEMPLO OPERACIONES REALIZADAS SOBRE LAS IMÁGENES CON LA HERRAMIENTA IMAGE AUGMENTOR.....	18
FIGURA 21. EJEMPLO ANOTACIÓN CON ROBOFLOW.	19
FIGURA 22. EJEMPLO IMPORTACIÓN DRIVE EN GOOGLE COLAB	22
FIGURA 23.EJEMPLO INSTALACIÓN DE YOLOv5 Y DESCARGA DE PESOS PRE-ENTRENADOS EN GOOGLE COLAB.....	22
FIGURA 24. JERARQUÍA DE FICHEROS EN YOLOv5.....	23
FIGURA 25. EQUERA DATA AUGMENTATION EN YOLOv5.....	25
FIGURA 26. CÁLCULO DE LINTERSECTION OVER UNION.	27
FIGURA 27. EJEMPLO APLICACIÓN INTERSECTION OVER UNION.....	27
FIGURA 28. EJEMPLO MATRIZ DE CONFUSIÓN.....	28
FIGURA 29. EJEMPLO GRÁFICA PRECISION-RECALL	29
FIGURA 30. EXPERIMENTO1. YOLOv5s. GRÁFICAS RESULTADO ENTRENAMIENTO.....	31
FIGURA 31. EXPERIMENTO 1. YOLOV5M. GRÁFICAS RESULTADO ENTRENAMIENTO	32
FIGURA 32. EXPERIMENTO 1. YOLOv5s. MATRIZ DE CONFUSIÓN	33
FIGURA 33. EXPERIMENTO 1. YOLOv5s. ETIQUETAS REALES	34
FIGURA 34.EXPERIMENTO 1.YOLOv5s. DETECCIONES.....	34
FIGURA 35. EXPERIMENTO 1. YOLOV5M. MATRIZ DE CONFUSIÓN	35
FIGURA 36.EXPERIMENTO 1. YOLOV5M. DETECCIONES	36
FIGURA 37.EXPERIMENTO 1. YOLOv5M. ETIQUETAS REALES.....	36
FIGURA 38. EXPERIMENTO 2. INSTANCIAS PEQUEÑAS. MATRIZ DE CONFUSIÓN	37
FIGURA 39.EXPERIMENTO 2. INSTANCIAS PEQUEÑAS. ETIQUETAS REALES	38
FIGURA 40.EXPERIMENTO 2. INSTANCIAS PEQUEÑAS. DETECCIONES.....	38
FIGURA 41.EXPERIMENTO 2. INSTANCIAS MEDIANAS. MATRIZ DE CONFUSIÓN	39
FIGURA 42. EXPERIMENTO 2. INSTANCIAS MEDIANAS. DETECCIONES	40
FIGURA 43.EXPERIMENTO 2. INSTANCIAS MEDIANAS. ETIQUETAS REALES	40
FIGURA 44.EXPERIMENTO 2. INSTANCIAS GRANDES. MATRIZ DE CONFUSIÓN	40
FIGURA 45. EXPERIMENTO 2. INSTANCIAS GRANDES. DETECCIONES.....	41
FIGURA 46.EXPERIMENTO 2. INSTANCIAS GRANDES. ETIQUETAS REALES	41
FIGURA 47. EXPERIMENTO 3. RUIDO 0.01. MATRIZ DE CONFUSIÓN.....	42
FIGURA 48. EXPERIMENTO 3. RUIDO 0.01. DETECCIONES	43
FIGURA 49. EXPERIMENTO 3.RUIDO 0.01. ETIQUETAS REALES	43
FIGURA 50. EXPERIMENTO 3.RUIDO 0.02. MATRIZ DE CONFUSIÓN	44

FIGURA 51.EXPERIMENTO 3.RUIDO 0.02. DETECCIONES.....	44
FIGURA 52.EXPERIMENTO 3. RUIDO 0.02. ETIQUETAS REALES	44
FIGURA 53. EXPERIMENTO 3.RUIDO 0.05. MATRIZ DE CONFUSIÓN	45
FIGURA 54.EXPERIMENTO 3. RUIDO 0.05. DETECCIONES	46
FIGURA 55. EXPERIMENTO 3.RUIDO 0.05. ETIQUETAS REALES	46
FIGURA 56. EXPERIMENTO 4. UNA OCLUSIÓN. MATRIZ DE CONFUSIÓN.....	47
FIGURA 57.EXPERIMENTO 4. UNA OCLUSIÓN. DETECCIONES	47
FIGURA 58.EXPERIMENTO 4. UNA OCLUSIÓN. ETIQUETAS REALES	47
FIGURA 59.EXPERIMENTO 4. DOS OCLUSIONES. MATRIZ DE CONFUSIÓN	48
FIGURA 60.EXPERIMENTO 4. DOS OCLUSIONES. ETIQUETAS REALES	49
FIGURA 61.EXPERIMENTO 4. DOS OCLUSIONES. DETECCIONES	49
FIGURA 62.EXPERIMENTO 4. TRES OCLUSIONES. MATRIZ DE CONFUSIÓN.....	49
FIGURA 64.EXPERIMENTO 4. TRES OCLUSIONES. DETECCIONES.	50
FIGURA 63.EXPERIMENTO 4. TRES OCLUSIONES. ETIQUETAS REALES	50

Índice tablas

TABLA 1.RESUMEN DATASET UNIVERSIDAD DE GRANADA.....	16
TABLA 2. RESUMEN DATASET DEFINITIVO CUSTOMIZADO	16
TABLA 3. EJEMPLO FORMATO ANOTACIONES YOLOv5	19
TABLA 4. EXPERIMENTO 1. YOLOv5s. RESUMEN PARÁMETROS EMPLEADOS	31
TABLA 5. EXPERIMENTO 1. YOLOv5m. RESUMEN PARÁMETROS EMPLEADOS	31
TABLA 6. EXPERIMENTO 1. YOLOv5s. RESULTADOS EVALUACIÓN MODELO.....	33
TABLA 7. EXPERIMENTO 1.YOLOv5m. RESULTADOS EVALUACIÓN MODELO	35
TABLA 8. EXPERIMENTO 2. INSTANCIAS PEQUEÑAS. RESULTADOS EVALUACIÓN MODELO	37
TABLA 9. EXPERIMENTO 2. INSTANCIAS MEDIANAS. RESULTADOS EVALUACIÓN MODELO	39
TABLA 10.EXPERIMENTO 2. INSTANCIAS GRANDES. RESULTADOS EVALUACIÓN MODELO.....	40
TABLA 11. EXPERIMENTO 3. RUIDO 0.01. RESULTADOS EVALUACIÓN MODELO	42
TABLA 12. EXPERIMENTO 3.RUIDO 0.02. RESULTADOS EVALUACIÓN MODELO	43
TABLA 13. EXPERIMENTO 3.RUIDO 0.05. RESULTADOS EVALUACIÓN MODELO	45
TABLA 14. EXPERIMENTO 4. UNA OCLUSIÓN. RESULTADOS EVALUACIÓN MODELO.....	46
TABLA 15.EXPERIMENTO 4. DOS OCLUSIONES. RESULTADOS EVALUACIÓN MODELO	48
TABLA 16.EXPERIMENTO 4. TRES OCLUSIONES. RESULTADOS EVALUACIÓN MODELO.....	49

1. Introducción

1.1 - Motivación del proyecto

Se estima un arsenal existente de 639.000.000 armas de fuego en el mundo, de entre las cuales, cerca de la mitad se encontrarían en manos de civiles.

Son muchos los países donde la tenencia de armas es legal. En algunos de ellos está plenamente reconocido como derecho con escasas limitaciones, como es el caso de EE. UU. En otros, al igual que en España, se limita a cuerpos de seguridad o licencias destinadas a actividades como la caza o el tiro deportivo. Sin embargo, las regulaciones existentes en muchos países no evitan la creciente tendencia al tráfico de armas y el papel fundamental de las mismas en homicidios, delincuencia organizada, terrorismo y otros conflictos armados en todo el mundo.

Este problema se encuentra actualmente en el foco de los esfuerzos por mantener el orden y la seguridad ciudadana. [10]

Así, con la aparición de nuevas técnicas y algoritmos de inteligencia artificial y en particular, de visión artificial, son muchos los estudios que se han llevado a cabo en relación con la detección de armas en videovigilancia y su aplicación al mundo real para evitar todo tipo de delitos.

Este proyecto pretende precisamente profundizar más en la detección de armas sobre imágenes y en cómo, las condiciones de la imagen con la que se trabaja pueden afectar al algoritmo de detección empleado, con la esperanza de aportar resultados que resulten útiles para la comunidad científica y ampliar el conocimiento disponible en la materia.

1.2 - Objetivos del proyecto

Este proyecto tiene como objetivo principal la detección de armas de fuego en imágenes y su clasificación en armas cortas y armas largas. También se pretende analizar las características de la imagen que pueden reducir el desempeño del modelo construido y determinar hasta qué punto afectan a la detección. Para lograr dichos objetivos, se deben de completar otra serie de subobjetivos y tareas que se describen a continuación.

- Realizar un estudio de experimentos previos publicados por la comunidad científica e identificar los diferentes métodos de detección de objetos existentes y en qué se basan, así como, entender su arquitectura y funcionamiento básicos.
- Construir el conjunto de datos o *dataset* con el que se trabajará. Este "dataset" deberá contener imágenes con un número objetos por clase equilibrado. El tamaño

del dataset deberá ser lo suficientemente grande para garantizar detecciones fiables.

- Analizar la arquitectura y funcionamiento del algoritmo elegido, YOLOv5 para ser capaces de maximizar los resultados.
- Entrenar el modelo basado en YOLOv5 con el dataset previamente construido.
- Experimentar con el modelo y distintos datos, evaluando los resultados obtenidos en cada experimento.
- Realizar un análisis final de los resultados globales obtenidos.

1.3 - Organización de la memoria

En este documento se detallará como se han ido completando los objetivos mencionados con anterioridad.

Primero se profundizará en el contexto teórico de la detección de objetos y en particular en el *Deep Learning*. Así como en la arquitectura y funcionamiento de YOLO (*You Only Look Once*), el algoritmo elegido en este caso.

Se mencionarán los enfoques de otros estudios en la materia y a continuación se presentará la solución informática que se ha decidido dar al problema de la detección de armas en este trabajo y los distintos experimentos realizados. Tras esto, se analizarán los resultados para extraer una conclusión general de los mismos.

Como añadido, al final del documento podrán encontrarse detalles específicos de la implementación y otra información complementaria en el anexo.

2. Marco teórico

2.1 - Aprendizaje profundo o *Deep Learning*

El aprendizaje profundo o *Deep Learning* es el subconjunto del aprendizaje automático o *Machine Learning*, a su vez, disciplina de la inteligencia artificial, que permite a través de redes neuronales artificiales aprender de una manera similar al cerebro humano. Así, se extraen características útiles de los datos de manera automática, evitando en todo lo posible la intervención humana. [25]

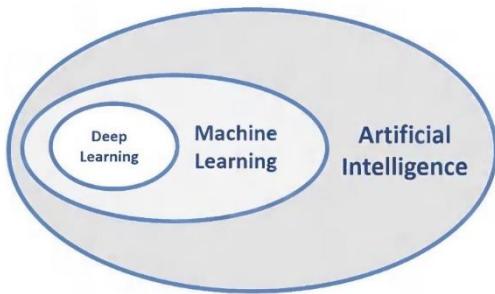


Figura 1. Esquema subconjuntos de la inteligencia artificial.

Fuente: https://deveseed.com/servir-amazonia-ml/docs/Lesson1a_Intro_ML_NN_DL.html

El término *Deep Learning* recibe este nombre debido al número de capas ocultas entre la capa de entrada y la capa de salida de la red neuronal. En el caso del aprendizaje profundo, las redes neuronales cuentan con docenas o centenas de capas intermedias que se encargan de extraer de manera jerárquica las características de los datos. De este modo, lo normal es que cuanto mayor sea el número de capas y nodos en la red, mayor sea la precisión de esta. Sin embargo, ha de tenerse en cuenta que una red de mayor tamaño consume más recursos. [11]

Los entrenamientos de los modelos de *Deep Learning* son costosos en cuanto a tiempo y recursos. Por ello se utilizan GPUs. Estas están diseñadas para realizar cálculos rápidos con matrices de gran escala y dan buenos resultados en la resolución de problemas de *Machine Learning* con datos estructurados y desestructurados, como imágenes o videos.

Hoy en día el aprendizaje profundo está presente en múltiples ámbitos. Desde las redes sociales, hasta los procesadores de lenguaje natural que dan vida a los asistentes digitales o incluso en la medicina, a la hora de predecir enfermedades. En este caso, nos centraremos en otro de sus usos más comunes; la detección de objetos en videos e imágenes.

2.1.1 – Detección de objetos

Cuando hablamos de "detección de objetos" nos referimos a la idea de reconocer y detectar diferentes objetos en una imagen o vídeo, así como etiquetarlos para poder clasificarlos. Mediante la detección de objetos somos capaces de contar el número de instancias o apariciones de dicho objeto, determinando el tipo de objeto y cuál es localización del mismo dentro de la imagen o vídeo. Para ello, se aprovecha el hecho de que cada objeto cuenta con una serie de características que nos permiten diferenciarlo del resto, y que alimentan a los distintos algoritmos de aprendizaje. [11]

Es importante recalcar la diferencia existente entre la clasificación de una imagen y el concepto de detección de un objeto. Podemos entrenar un algoritmo para que nos indique que una imagen contiene un objeto determinado, pero en detección de objetos, nuestro algoritmo debe ser capaz de localizar la posición de dicho objeto en la imagen.

Existen dos enfoques principales: la utilización de técnicas de *Machine Learning* tradicionales y las técnicas de *Deep Learning*. A diferencia de las primeras mencionadas, el *Deep Learning* permite completar el proceso de detección de un objeto, incluyendo la clasificación del mismo, sin necesidad de invocar una etapa de extracción de características previa a la detección.

Estos enfoques, aunque han ido evolucionando en cuanto a velocidad y precisión con el tiempo, presentan algunas dificultades, como la preparación de los datos cuyo volumen debe ser sustancial y que además implica la anotación y el delimitado de los objetos con rectángulos a su alrededor. La diversidad existente entre imágenes y las formas de los objetos y su tamaño, o la velocidad de procesamiento y la memoria necesaria también son delimitantes. Algunos de los modelos de aprendizaje profundo más utilizados actualmente en detección de objetos son la familia de las redes neuronales convolucionales (CNN) y el modelo *You Only Look Once* (YOLO).

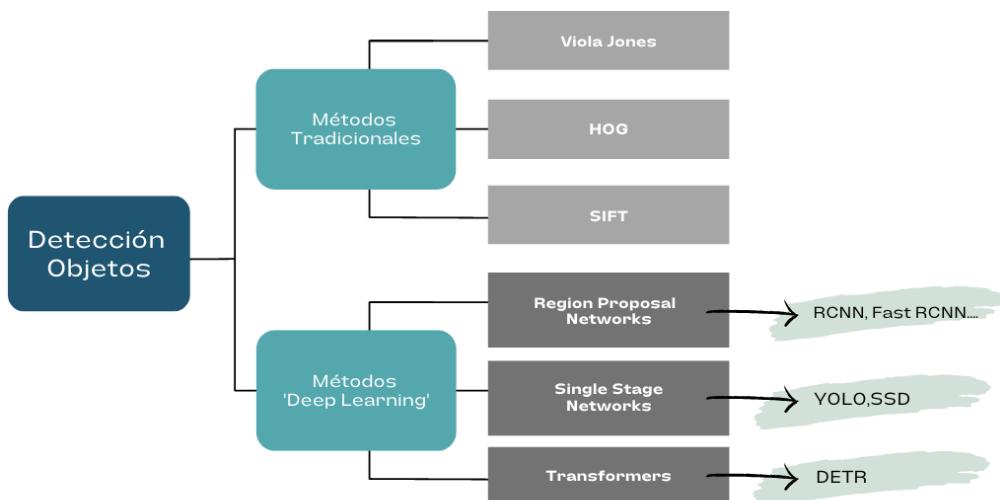


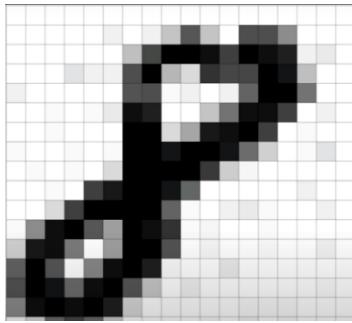
Figura 2. Esquema métodos detección objetos. Figura original

2.1.2 – Redes Neuronales Convolucionales

En este apartado nos centraremos en el uso del *Deep Learning* para detectar objetos, y más en particular, en las redes neuronales convoluciones (CNN) mencionadas previamente.

La idea principal de una red neuronal convolucional es imitar la forma en la que el cerebro humano procesa las imágenes. A grandes rasgos, existen distintas capas de neuronas capaces de reconocer ciertas características simples. Esa información pasa a capas más complejas que interconectan las características mencionadas para, posteriormente, ser capaces de detectar características más complejas que finalmente, en combinación, nos permitan determinar qué es lo que estamos visualizando. Del mismo modo, las redes convolucionales extraen patrones básicos, combinándolos hasta ser capaz de detectar y clasificar objetos.

Antes de profundizar en la arquitectura de las CNN conviene entender cómo se representa una imagen digitalmente. Las imágenes son matrices con valores desde 0 a 255 correspondientes a cada uno de los píxeles que conforman la misma. Normalmente estos valores se normalizan para la red a 0 y 1. [29]



0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
0	0	1	0	0	0	41	160	250	253	235	162	255	238	206	11	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	187	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	3	0	4	15	236	216	0	0	38	109	247	249	169	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	4	0	3	252	250	228	253	255	234	112	28	0	2	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	0	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	122	242	255	144	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 3. Representación digital de una imagen.

Fuente: https://computersciencewiki.org/index.php/Data_representation

La red tomará el número de píxeles como referencia para calcular el número de neuronas. En caso de la imagen sea a color, se necesitarán tres canales por lo que se multiplica por tres el número de neuronas.

Teniendo esto en cuenta podemos describir la estructura básica de una red convolucional.

ARQUITECTURA DE UNA CNN

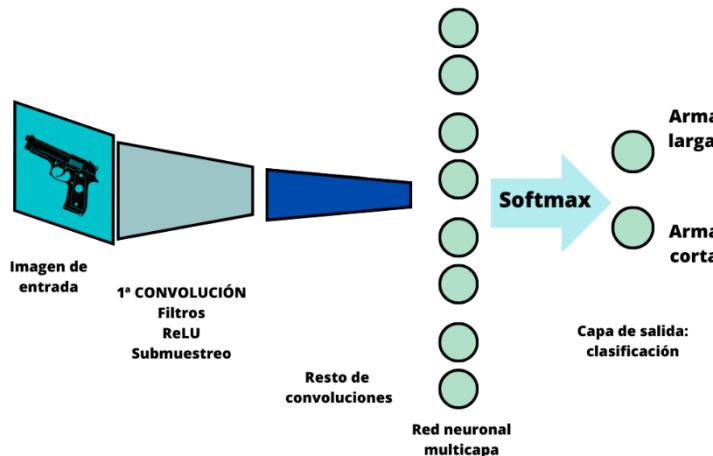


Figura 4. Arquitectura de una CNN. Figura original

Las redes neuronales convolucionales en esencia constan de dos partes: una primera parte convolucional que extrae características y las comprime reduciendo su tamaño original. Las imágenes resultado se denominan tarjetas convolucionales que se concatenan en un vector llamado código CNN. Y una parte de clasificación, donde el código CNN se suministra a lo que se conoce como Perceptrón Multicapa (MLP, Multi Layers Perceptron) donde se combinan estas características para clasificar la imagen.

Las convoluciones consisten en tomar grupos de píxeles cercanos, (el número de píxeles será lo primero que se definirá y lo denominamos ventana de filtro) y con esa ventana,

realizar un producto escalar con otra matriz que denominamos *Kernel*, que irá recorriendo las neuronas de entrada generando una nueva matriz de mayor tamaño y que representará una nueva capa oculta de neuronas. [16]

Lo cierto es que no solo se opera con un *Kernel*, si no con varios cuyo conjunto se denomina "filtros" y que dan lugar a un conjunto de matrices de salida que representan ciertas características de la imagen y que denominamos *Feature Mapping*.

Tras esto, se aplica la función de activación, normalmente ReLu : $f(x) = \max(0, x)$.

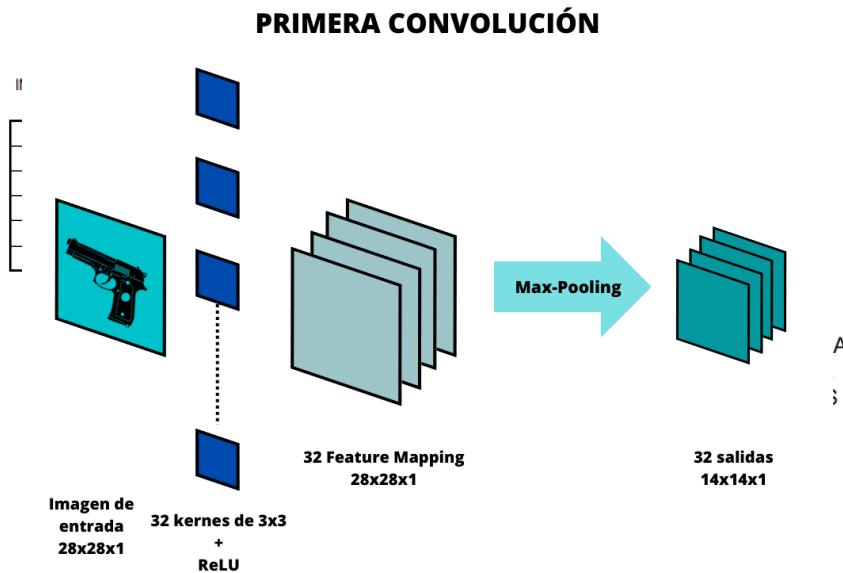


Figura 5. Esquema de una convolución. Figura original

El número de neuronas tras realizar esta primera convolución es muy elevado, por tanto, el siguiente paso es reducirlo aplicando algún método de submuestreo. Es muy común usar el método de Max-Pooling. De esta forma, volveríamos a recorrer las imágenes de salida tomando más de un píxel y solo preservando el valor más alto de entre estos. [30]



SUBSAMPLING:
Aplico Max-Pooling de 2×2
y reduzco mi salida a la mitad

Figura 6. Ejemplo funcionamiento Max-Pooling.

Fuente: <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

Este proceso se repite con el resto de las convoluciones, detectando cada vez características y formas más complejas. Y finalmente, se toma la red obtenida que es "tridimensional" (alto, ancho, mapas) y se aplana convirtiéndola en una red tradicional a la cual le aplicamos la función *Softmax*, que conecta con la capa de salida final. Esta última capa tendrá la cantidad de neuronas correspondientes al número de clases entre las que clasificar.

La red convolucional aprende de una forma similar a las redes tradicionales, mediante el *Backpropagation*, ajustando los pesos. La diferencia radica en que en este tipo de redes debemos ajustar los pesos de los *Kernel*, que, al tener un tamaño reducido, implica muchos menos parámetros.

Dentro de la familia de los llamados *Region Proposal-Based methods* que usan redes convolucionales para detectar objetos en imágenes destacan **R-CNN**, **Fast R-CNN** y **Faster R-CNN**.

Una forma de detectar objetos consiste en dividir en regiones la imagen y clasificar la presencia de un objeto en ellas, pero las posiciones de los distintos objetos, así como los diferentes ratios, obligarían a seleccionar un número de regiones demasiado elevado. Para solucionar este problema surgen las **R-CNN**.

En las **R-CNN** se trabaja solo con 2000 regiones denominadas *Region Proposal* que se extraen a partir de un algoritmo de búsqueda selectiva que segmenta la imagen y va aumentando el tamaño de las regiones mediante un algoritmo *Greedy*. Después estas regiones se deforman en un cuadrado que pasa a la red convolucional, la cual produce un vector de características de 4096 dimensiones como salida.

Estas características alimentan a una *SVM* (*Support Vector Machine*). Además, se definen 4 valores de desplazamiento que permiten aumentar la precisión del *Bounding Box* o cuadro delimitador del objeto detectado, en caso de que este quede cortado.

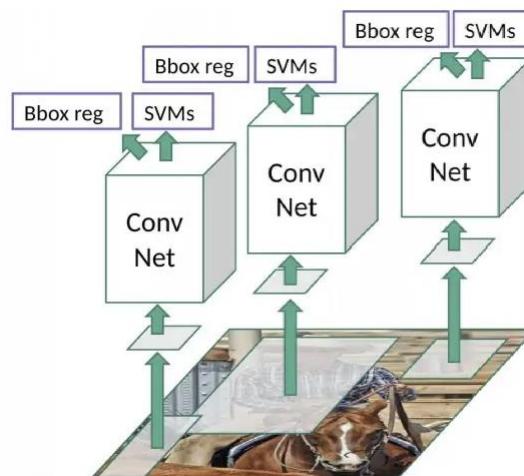


Figura 7. Esquema ilustrativo detección por regiones.

Fuente: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

Este algoritmo toma mucho tiempo y no es implementable en tiempo real, y la búsqueda selectiva no aprende a lo largo del tiempo, lo cual podría conducir a errores a la hora de seleccionar las regiones candidatas. Surgen así, las ***Fast R-CNN***. En este caso, la CNN se alimenta con la imagen de entrada generando un mapa de características. A partir de ese mapa se deducen las regiones que serán transformadas al tamaño adecuado mediante una capa *RoI* (*Region of Interest*) de agrupación. Y a partir del vector generado se pasará a una capa *Softmax* para predecir la clase de la región propuesta y los valores de desplazamiento previamente mencionados [21].

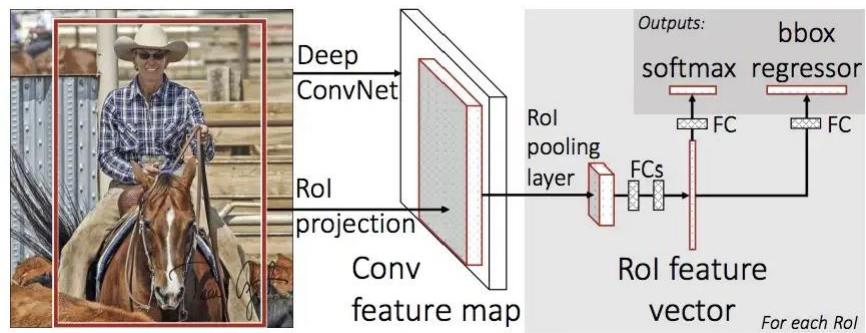


Figura 8. Esquema *Fast R-CNN*.

Fuente: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

Sin embargo, la búsqueda selectiva para las propuestas de región sigue suponiendo un cuello de botella. Esto se soluciona con la llegada de las ***Faster R-CNN***. En este caso, las regiones se deducen usando otra red a partir del mapa de características, acelerando el proceso de detección [15].

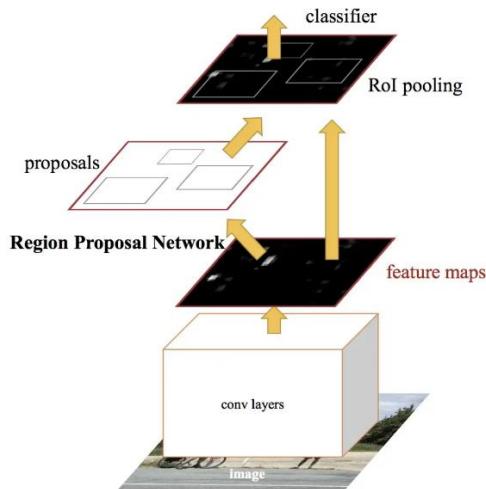


Figura 9. Esquema *Faster R-CNN*.

Fuente: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

En la imagen se aprecian la diferencia de velocidad de los tres tipos de redes comentadas.

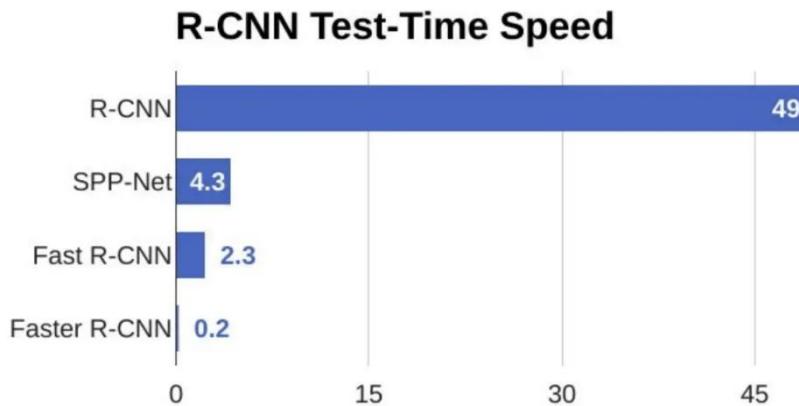


Figura 10. Comparativa de velocidad. Redes convolucionales.

Fuente: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

2.1.3 – YOLO

En el año 2015, Joseph Redmon crea **YOLO** (*You Only Look Once*), un algoritmo de aprendizaje profundo especializado en la detección de objetos basado en redes neuronales convolucionales. Desde entonces, distintas versiones del mismo han ido siendo lanzadas.

El algoritmo destaca frente al resto por su rapidez a la hora de detectar objetos en tiempo real. La entrada es la imagen completa, que solo pasa una vez por la CNN, por eso es lo que se denomina una solución *one-stage*.

El objetivo es maximizar el valor mAP (*mean average precision*) que se calcula a partir del *recall* y la precisión, dos valores muy importantes para cualquier modelo basado en *Machine Learning*.

Conviene conocer también algunos conceptos muy relacionados con la arquitectura de los modelos YOLO para entender su funcionamiento:

- **Backbone:** es una CNN que acumula y produce características con diferentes formas y tamaños usando modelos de clasificación como ResNet, VGG o EfficientNet.

- **Neck:** conjunto de capas que integran y combinan características antes de pasar a la capa de predicción.

- **Head:** se encarga de tomar las características del *neck* junto con las predicciones y clasificar mediante regresión. Su salida consiste en 4 valores. Generalmente, son las coordenadas X, Y, así como la altura y anchura de las *Bounding Box*.

Este algoritmo divide la imagen en una cuadrícula con celdas calculando la probabilidad de que el objeto resida en cada una de ellas. Aquellas cuadriculas cercanas con alta probabilidad de contener un objeto se agrupan como un solo objeto. El resto son descartadas mediante la técnica de *Non-Max Supression (NMS)*.

La técnica de *Non-Max Supression* consiste en seleccionar la mejor *Bounding Box* de entre aquellas que se superponen. Aquí aparece el concepto de *Intersection Over Union* (IoU).

La idea es definir un umbral de confianza y un umbral de IoU, ordenar las *Bounding Boxes* en orden descendiente según el valor de confianza de cada una y eliminar aquellas que no superen el umbral de confianza establecido. Una vez hecho esto, se recorren las cajas restantes empezando por aquella con mayor valor de confianza, calculando el valor IoU de la caja actual con el resto de cajas restantes pertenecientes a la misma clase. Se eliminará la caja con menor valor de confianza si el valor IoU de ambas cajas comparadas es mayor que el umbral IoU propuesto. Este último paso se repetirá hasta terminar con la lista de cajas de la misma clase superpuestas.[23][26]

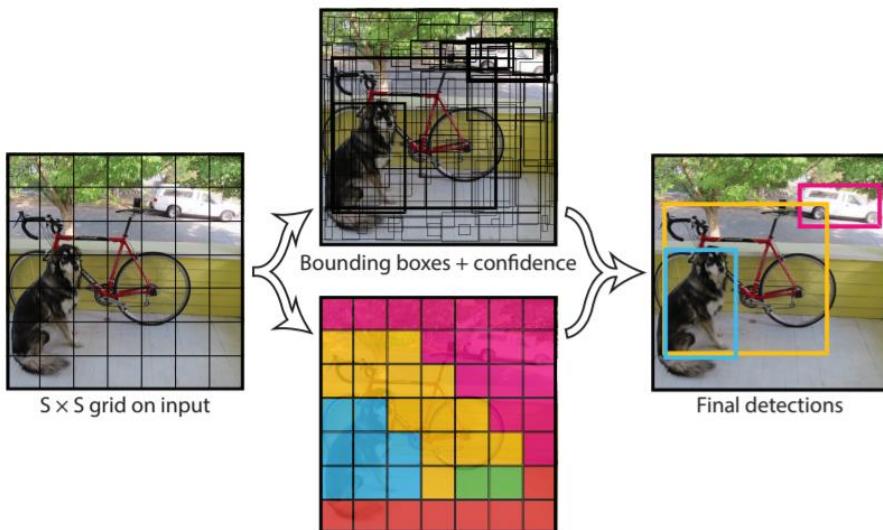


Figura 11. Detección objetos con YOLO.

Fuente: <https://medium.com/analytics-vidhya/understanding-yolo-and-implementing-yolov3-for-object-detection-5f1f748cc63a>

De vuelta al YOLO en sí, a la hora de entrenar, los pesos se ajustan dependiendo de cómo las predicciones se adecuan a la localización real del objeto o *Ground Truth*.

Tras la primera versión de YOLO con un mAP de 63,4, en 2016 de nuevo Joseph Redmon y Ali Farhadi lanzan YOLOv2 o YOLO9000 capaz de detectar más de 90000 categorías de objetos, esta vez con un mAP de 78,6. Esta versión introducía el concepto de *Anchor Boxes*, que no son más que áreas en la imagen que ilustran las posiciones ideales de los objetos a detectar y que se calculan examinando los datos de entrenamiento y haciendo agrupaciones sobre los mismos.

Más tarde, en 2018 aparece YOLOv3 a manos de los mismos creadores que la versión anterior. Consiste en 75 capas convolucionales sin usar capas de *pooling* o completamente conectadas, y mantiene el tiempo mínimo de inferencia usando modelos residuales con *Feature Pyramid Network* (FPN), un extracto de características. Como *backbone*, usa la arquitectura Darknet53. Aunque similar en cuanto a precisión, esta versión de YOLO es tres veces más rápida que la técnica SSD de detección de objetos.

Con la llegada de YOLOv4, creado por Alexey Bochkovskiy en el año 2020, se introducen los conceptos de *Bag of freebies* (BOF) y *Bag of Specials* (BOS).

Las primeras se tratan de técnicas que mejoran el rendimiento del modelo sin incrementar el coste de la inferencia (*Data Augmentation*, *Normalization...*), las segundas, son también técnicas que aumentan la precisión y el coste computacional al mismo tiempo (*Spatial attention modules*, Funciones de activación no lineales...)

Aunque hoy en día existen versiones más recientes como YOLOv6, YOLOv7, y YOLOv8 que parecen haber demostrado un mejor rendimiento, en este caso nos centraremos en el estudio de la versión YOLOv5.

2.1.3.1 - YOLOv5

YOLOv5 fue desarrollado por Ultralytics, que ya había desarrollado en su momento la versión de Pytorch de YOLOv3, y lo lanza en junio de 2020. Esta versión escrita en Python usa el *framework* de Pytorch en contraste con el resto de versiones, que usan C y CUDA. Existen varios modelos de esta versión, cada uno con unos resultados distintos en cuanto a precisión y rendimiento como se puede apreciar en la imagen. La diferencia entre los modelos reside, únicamente, en el número de capas y parámetros utilizados. No existe diferencia a nivel operacional.

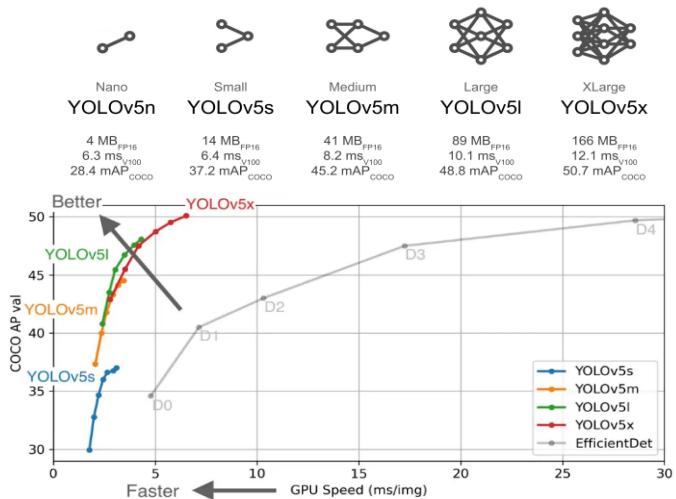


Figura 12. Comparativa entre modelos de YOLO.

Fuente: <https://medium.com/axinc-ai/yolov5-the-latest-model-for-object-detection-b13320ec516b>

En cuanto a la arquitectura de estos modelos, YOLOv5 cuenta con una red CSP-Darknet53 como *backbone*, SPP y PANet como *neck*, y la misma *head* que se utilizaba en YOLOv3 y YOLOv4.

CSP-Darknet53 es una red convolucional a la cual se le aplica la estrategia de *Cross Stage Partial*. Así, YOLOv5 emplea esta estrategia para dividir el mapa de características de la capa base en dos partes y luego las fusiona mediante una jerarquía de etapas cruzadas.

Como resultado la velocidad de inferencia del algoritmo aumenta al reducirse el número de parámetros necesarios.

En el *neck*, YOLOv5 usa una red *Path Aggregation Network* (PANet) y la modifica aplicando la estrategia CSPNet, extrayendo características espaciales de la imagen. Evita que las características de bajo nivel de las primeras capas de la red se pierdan asegurando su conexión con las características de alto nivel posteriores.

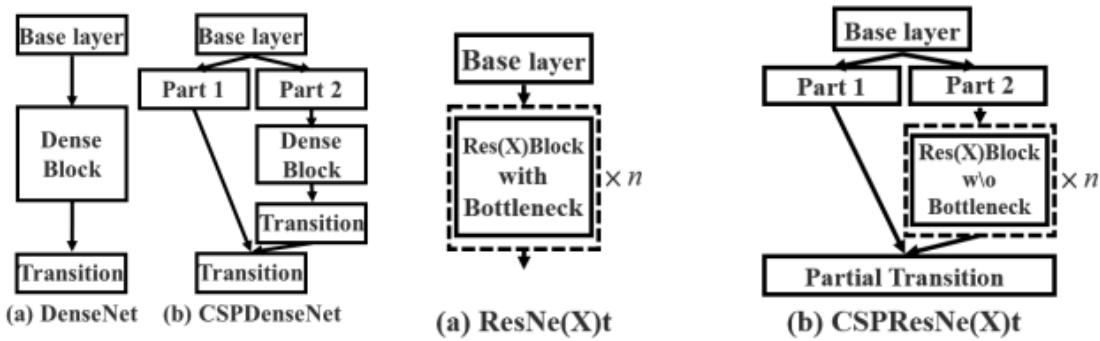


Figura 13. Aplicación de CSPNet a ResNet y DenseNet.

Fuente: <https://iq.opengenus.org/yolov5/>

Por otro lado, mediante la variante más rápida SPPF del *Spatial Pyramid Pooling* (SPP), YOLOv5 mejora la velocidad de la red, eliminando el problema que supone la diferencia de escala y tamaño entre imágenes y generando una imagen de tamaño fijo sin importar el tamaño de la entrada.

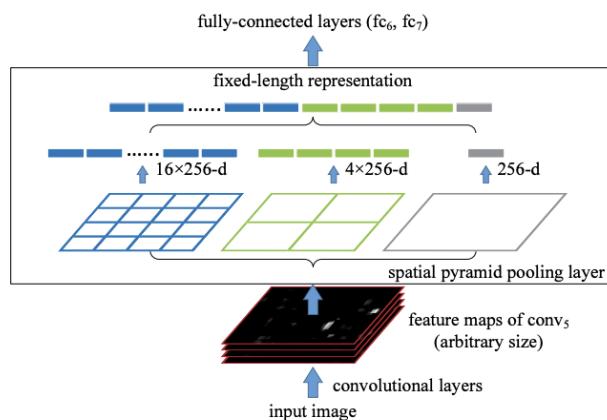


Figura 14. Estructura de una red con Spatial Pyramid Pooling.

Fuente: <https://arxiv.org/pdf/1406.4729v4.pdf>

En cuanto al head de la red, basado en *anchors*, se encarga de la clasificación de los objetos detectados.

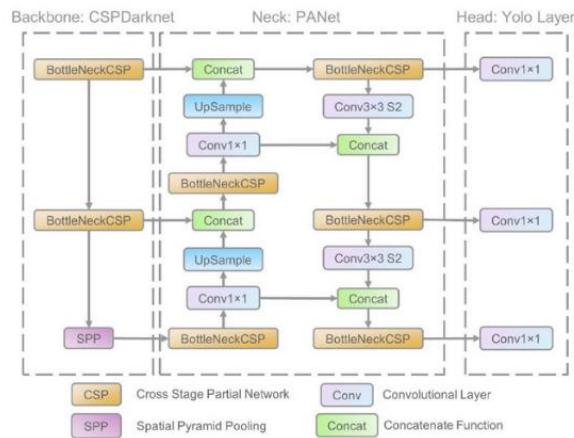


Figura 15. Arquitectura de YOLOv5.

Fuente: https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet_fig1_349299852

El principal cambio con respecto a versiones anteriores son las ecuaciones de cálculo de las coordenadas de las *Bounding Boxes*.

$b_x = \sigma(t_x) + c_x$ $b_y = \sigma(t_y) + c_y$ $b_w = p_w \cdot e^{t_w}$ $b_h = p_h \cdot e^{t_h}$	$b_x = (2 \cdot \sigma(t_x) - 0.5) + c_x$ $b_y = (2 \cdot \sigma(t_y) - 0.5) + c_y$ $b_w = p_w \cdot (2 \cdot \sigma(t_w))^2$ $b_h = p_h \cdot (2 \cdot \sigma(t_h))^2$
(a)	(b)

Figura 16. Ecuaciones cálculo coordenadas Bounding Boxes en YOLOv5.
Fuente: <https://iq.opengenus.org/yolov5/>

Como hemos comentado previamente, el algoritmo divide la imagen en $S \times S$ celdas para que el detector actúe posteriormente sobre cada una de ellas generando un tensor con una serie valores como se observa en la figura.

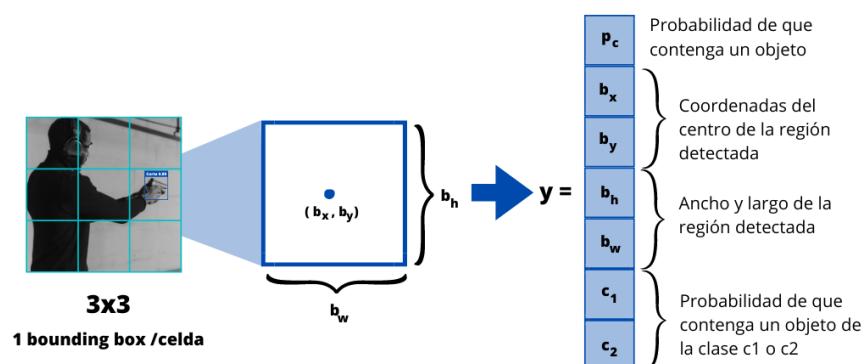


Figura 17. Esquema generación tensor en YOLOv5. Figura original

Como resultado final obtendremos un conjunto de tensores que representen las distintas posibles detecciones de las celdas con un número mayor de dimensiones al original.

3. Estado del arte

En los últimos años, el reconocimiento de armas y más en particular de armas de fuego en imágenes y sobre todo en videovigilancia, se ha convertido en uno de los temas más estudiados en el campo de la detección de objetos.

Son muchos los estudios y documentos publicados en la materia. A continuación, se destacan algunos de ellos:

- *"Automatic handgun detection alarm in videos using deep learning"*

Este estudio realizado por la Universidad de Granada premiado en la categoría de I+D+i del Security Forum 2017, ahonda en el uso de los métodos tradicionales y el *Deep Learning* para la detección de pistolas en videos. Los mejores resultados son obtenidos usando una Faster R-CNN. [2]

- *"Brightness guided preprocessing for automatic cold steel weapon detection in surveillance videos with deep learning"*

Se presenta un detector de armas de metal en videovigilancia basado en *Deep Learning* que mejora sus resultados con un procedimiento de preprocessado guiado por el brillo denominado DaCoLT. [3]

- *"Automatic Handgun Detection with Deep Learning in Video Surveillance Images"*

En este caso se estudia y compara el desempeño de tres modelos basados en redes neuronales convolucionales como son Faster R-CNN, RetinaNet y YOLOv3 en la detección de pistolas en videos. [5]

- *"Automatic Detection of Knives in Complex Scenes"*

Este documento se centra en el estudio de la combinación de técnicas de super-resolución con redes neuronales para la detección de cuchillos en imágenes complejas. La arquitectura de YOLOv4 es la elegida para esta tarea. [8]

- *"Improving handgun detection through a combination of visual features and body pose-based data"*

Aquí se propone la combinación de un clasificador de poses del cuerpo humano con una red que procese imágenes particionadas extrayendo características relevantes para la detección de pistolas en videovigilancia. [7]

- “*Real-time Concealed Weapon Detection on 3D Radar Images for Walk-through Screening System*”

En este caso se hace uso de las llamadas U-Net, otro tipo de redes neuronales convolucionales útiles para la segmentación de imágenes. Así se pretende conseguir detectar armas en tiempo real cuando las personas pasan por un sistema de escáner. Para ello se reformula la red 2D de segmentación. [9]

4. Solución informática

4.1 - Obtención de los datos

Para la realización de los distintos experimentos que se detallan más adelante, se han obtenido datos de distintas fuentes de la comunidad científica usados en otros trabajos similares, así como una recopilación de imágenes de recolección propia. En este apartado se profundizará en la naturaleza de estos datos.

Sobre los datos se aplicarán transformaciones y diferentes pre-procesos para determinar la influencia de los mismos en el rendimiento del modelo a la hora de detectar armas en nuevas imágenes. En cada experimento se detalla con más precisión dichas modificaciones y el uso de los mismos.

4.1.1 – Tipos de datos

Como se ha mencionado previamente, este estudio se centra en la detección de armas de fuego. Se trabaja con dos clases distintas que el algoritmo debe reconocer y clasificar correctamente. Esas clases son:

- Armas cortas --> pistolas y revólveres
- Armas largas --> escopetas, fusiles, ametralladoras y francotiradores



Figura 18. Ejemplos armas a detectar. Figura original

4.1.2 – Datasets

Para la realización de los experimentos ha sido necesario construir un dataset propio que incluyera los tipos de datos indicados en el apartado anterior.

Para agilizar el trabajo partimos de un dataset publicado por la Universidad de Granada que contiene únicamente armas de fuego cortas. A continuación, se presenta una tabla resumen con el contenido de este dataset:

Clases	Tamaño	Train	Valid	Test
1	2971	2080	594	297

Tabla 1. Resumen dataset Universidad de Granada

Como el etiquetado existente no coincide con el formato necesario para usar YOLOv5, se ha hecho uso de Roboflow, que es capaz de transformar las etiquetas XML de las que dispones al nuevo formato que buscamos automáticamente. En apartados posteriores se profundiza en dicho formato y en la herramienta Roboflow.

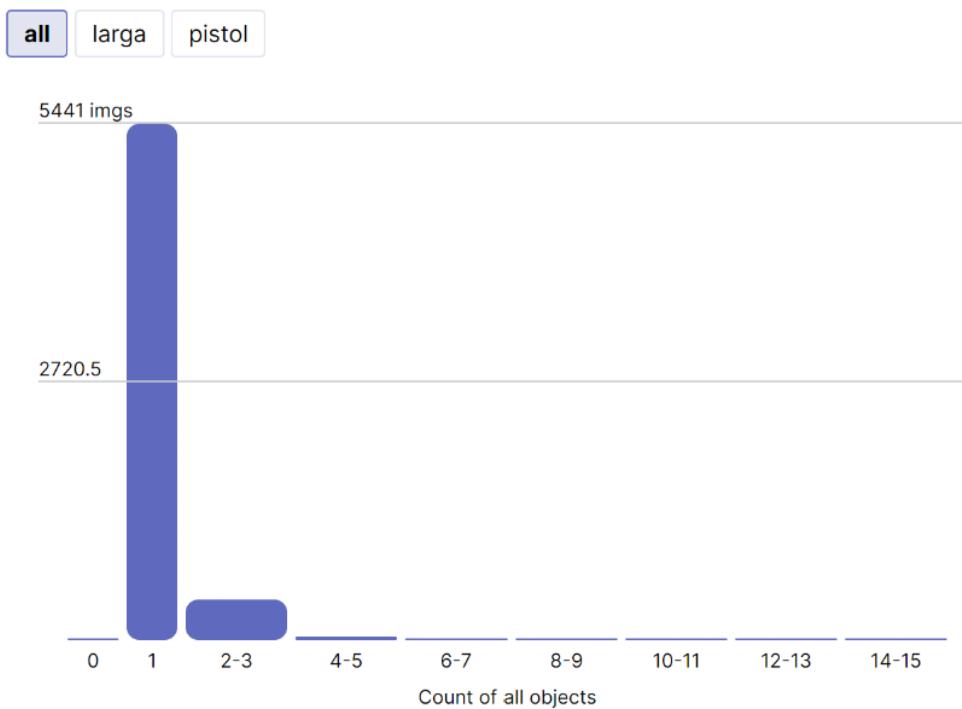
A este dataset se le han añadido imágenes de cosecha propia procedentes de Google Imágenes, y webs como Depositphotos, Shutterstock o Pixabay. De este modo, se han añadido imágenes de armas largas que han sido etiquetadas como tal. Como resultado, se trabajará con un dataset con 2 clases. Esto puede apreciarse en la tabla resumen del dataset final que se ha usado en los experimentos.

Clases	Tamaño	Train	Valid	Test
2	5924	4133	898	893

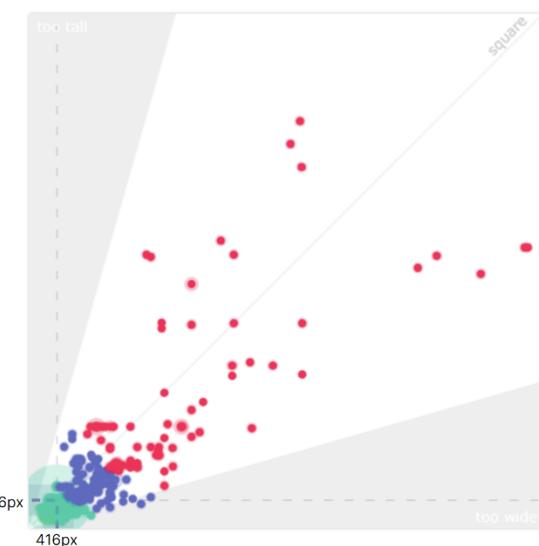
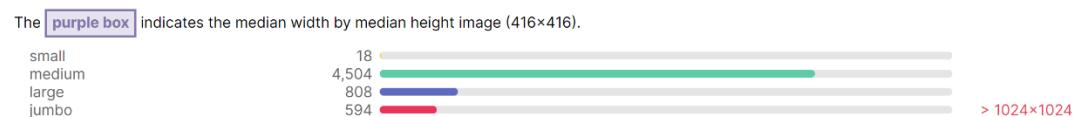
Tabla 2. Resumen dataset definitivo customizado

Se pueden apreciar en los siguientes gráficos otros detalles del dataset construido:

Histogram of Object Count by Image



Size Distribution



Class Balance



Figura 19. Clases, tamaño de las imágenes y número de objetos del dataset definitivo. Fuente: Roboflow

4.1.3 – Generación de imágenes sintéticas

A pesar de los múltiples estudios realizados en materia de detección de armas en los últimos años, los dataset públicos que se han podido encontrar son limitados. Además, los datasets encontrados se enfocaban en otro tipo de armas como cuchillos u otros objetos o solo presentaban armas de fuego cortas.

Como consecuencia los datos obtenidos han sido insuficientes, siendo necesario obtener nuevas imágenes para poder llevar a cabo los experimentos.

La variabilidad en la forma de las armas a detectar supone obstáculo para el detector, que necesita ser entrenado con una cuantiosa cantidad de datos para obtener buenos resultados. También influyen la posición, luminosidad, occlusiones y otras características de la imagen.

Como solución a esta escasez de datos, se ha hecho uso de una herramienta encontrada en GitHub desarrollada en Python, [Image Augmentor](#), que permite aplicar técnicas de *Data Augmentation* a un directorio de imágenes. Roboflow también permite aplicar estas técnicas, pero el número de imágenes generadas queda limitado en la versión gratuita de la herramienta.

En nuestro caso, a las imágenes que componen la clase de arma larga de nuestro dataset customizado, les aplicaremos varias operaciones, generando cinco nuevas imágenes por cada imagen en el dataset origen.

- Rotación de -45 grados + giro horizontal
- Rotación de 45 grados + giro vertical
- Rotación de 25 grados + translación de 20 y 10 pixeles (X,Y)
- Giro vertical + translación de 20 y 10 pixeles (X,Y)
- Rotación de 90 grados y translación de 40 y 20 pixeles (X,Y)



Figura 20. Ejemplo operaciones realizadas sobre las imágenes con la herramienta Image Augmentor. Figura original

De este modo se aumenta considerablemente el número de datos del dataset, aun que se prescindirá de algunas de las imágenes generadas ya que un número excesivamente

elevado de datos ralentizaría demasiado los entrenamientos, en especial teniendo en cuenta que el modelo se entrena desde Google Colab en la nube.

4.1.4 – Anotación de imágenes. Roboflow

Se deben anotar las imágenes con las que vayamos a trabajar para conocer la localización real de los objetos, en este caso, armas, que se vayan a detectar.

Para ello hemos usado Roboflow, un marco de trabajo especializado en la visión artificial, que facilita tanto el trabajo de anotación, como el preprocessamiento de las imágenes y el entrenamiento de modelos. [18]

Roboflow proporciona una interfaz cómoda e intuitiva para la anotación de imágenes, y permite la exportación de las anotaciones en varios formatos, entre ellos el de YOLOv5.

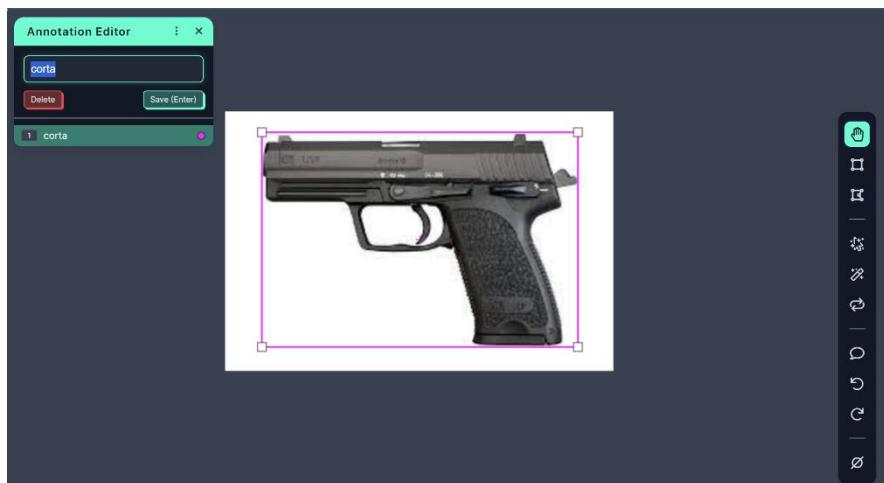


Figura 21. Ejemplo anotación con Roboflow.

Las anotaciones son normalizadas al tamaño de la imagen y se representan en un rango de 0 a 1 de la siguiente forma:

<clase-ID>	<X centro>	<Y centro>	<ancho>	<altura>
0	0.383	0.231	0.173	0.637

Tabla 3. Ejemplo formato anotaciones YOLOv5

Por cada imagen se genera un fichero de texto con el mismo nombre que esta y que contendrá tantas líneas como anotaciones de objetos haya en la imagen siguiendo el formato especificado.

Cabe recalcar que se debe de seguir una cierta coherencia a la hora de anotar. Los objetos deben de quedar encuadrados por la caja de anotación lo más cerca de sus bordes. Además, en caso de que haya objetos con occlusiones, se debe seguir en mismo criterio siempre para anotarlos. Ya sea anotar solo la parte visible o simular que la parte oculta está presente. Dependiendo del problema de detección, esto puede tener un impacto considerable en los resultados.

4.1.5 – Preprocesado de imágenes. Roboflow

El marco de trabajo de Roboflow facilita una serie de operaciones de preprocesado de imágenes que pueden útiles. El preprocesado de las imágenes a utilizar es importante ya que garantiza que el formato sea el mismo para todas ellas en cuanto a características como el tamaño, nivel de contraste u orientación. Las imágenes se transforman para que el modelo pueda usarlas en la fase de entrenamiento e inferencia o, lo que es lo mismo, en la detección de objetos en nuevas imágenes.

Es importante mencionar que el formato que se aplique a las imágenes de test y validación debe ser el mismo que se aplique a las imágenes de entrenamiento, de lo contrario no se obtendrán resultados óptimos en la detección.

Algunas de esas operaciones se usan en este trabajo. A continuación, se describen sus funciones según la propia documentación de Roboflow. [18]

- **Auto-Orient:**

Las imágenes contienen metadata que dicta la orientación según la cual estas deben ser mostradas y lo hace en función de cómo los píxeles se organizan en disco. En algunos casos las aplicaciones no respetan la orientación EXIF lo cual puede causar problemas.

Esta operación se debe aplicar prácticamente, en cualquier caso. Puede prescindirse de ella si se está seguro de que todas las imágenes tienen la misma orientación EXIF para mejorar la velocidad de las operaciones en tiempo de ejecución.

- **Resize:**

Conviene que las imágenes tengan todas el mismo tamaño para que la red aprenda más rápido.

En caso de que las imágenes del dataset varíen en tamaño, apostar por reducir el tamaño de las grandes para ajustarse a las pequeñas suele ser lo recomendable.

Cuando el tamaño de una imagen se modifica respetando el aspect-ratio existen una serie de píxeles de relleno o *padding* que diferencian a la imagen original de la imagen que alimenta a la red neuronal, llamados "píxeles muertos" o *Dead Pixels*. Estos normalmente se llenan con píxeles blancos o negros.

Roboflow ofrece distintos enfoques para abordar este problema, entre ellos:

- Usar píxeles blancos de relleno.
- Usar píxeles negros de relleno.
- Usar el reflejo de la imagen como relleno.

- **Auto-Adjust Contrast:**

El contraste en imágenes puede definirse como la variabilidad entre píxeles. Al aumentar o disminuir el contraste puede aumentar o disminuir la diferencia entre píxeles vecinos.

Roboflow permite modificar el contraste en función del histograma de la imagen para mejorar la normalización y la detección de líneas en condiciones de distinta luminosidad.

Los tipos de contraste que podemos aplicar con esta herramienta son:

- **Contrast Stretching:** La imagen se reescalara para incluir todas las intensidades entre el 2º y 98º percentil.
- **Histogram Equalization:** se distribuyen los valore más frecuentes de la imagen quedando representados los colores de los píxeles de manera proporcionada.
- **Adaptative Equalization:** el algoritmo usado utiliza histogramas calculados sobre regiones de mosaico de la imagen para mejorar el contraste local.

4.2 - Solución propuesta

4.2.1 – YOLOv5 en la nube. Google Colab

Para realizar los distintos experimentos que se presentarán más adelante, se ha optado por usar la versión gratuita de Google Colab. Este asigna GPUs según los recursos disponibles aleatoriamente, algunas más potentes que otras lo cual puede hacer variar el tiempo de los entrenamientos y, por tanto, no debería ser considerado un valor fiable para hacer comparaciones entre entrenamientos. Aun así, se incluye para tener una referencia de las posibles duraciones de los mismos.

Además, la versión gratuita de Colab, solo permite un tiempo máximo de ejecución de 12 horas, menos incluso si fueran muchos los recursos que se están consumiendo, interrumpiendo la ejecución del entrenamiento en algunos casos o haciendo reasignaciones de entorno y por tanto de GPU.

Las GPUs estándar en Colab suelen ser GPU Nvidia T4 Tensor Core. Las premium, normalmente son Nvidia V100 o A100 Tensor Core. Pero esto no está garantizado.

En caso de disponer de una tarjeta gráfica dedicada Nvidia con capacidades de cómputo CUDA y memoria de video de más de 2 GB, se recomienda el uso de los recursos locales. Las características de los mismos tienen un impacto importante en los tiempos de entrenamiento y han de tenerse muy en cuenta.

Para poder trabajar con Google Colab y YOLOv5 debemos preparar primero los datos. Se accede a las imágenes de nuestros dataset almacenadas en Google Drive a través del siguiente fragmento de código:

```
▶ #conectar con google drive  
from google.colab import drive  
  
▶ #clonar YOLO v5  
%cd content  
!git clone https://github.com/ultralytics/yolov5.git  
  
↳ [Errno 21] No such file or directory: 'content'
```

Figura 22. Ejemplo importación Drive en Google Colab

```
remote: Enumerating objects: 14245, done.  
remote: Total 14245 (delta 0), reused 0 (delta 0), pack-reused 14245  
Receiving objects: 100% (14245/14245), 13.56 MiB | 9.48 MiB/s, done.  
Resolving deltas: 100% (9803/9803), done.
```

```
▶ #Instalar YOLOv5  
%cd /content/yolov5  
%pip install -qr requirements.txt
```

```
/content/yolov5  
|██████████| 182 kB 14.7 MB/s  
|██████████| 62 kB 1.3 MB/s  
|██████████| 1.6 MB 66.7 MB/s
```

```
[ ] # Descargar modelo con pesos preentrenado  
...  
Modelos a probar:  
- yolov5s.pt  
- yolov5m.pt  
- yolov5l.pt  
- yolov5x.pt  
...  
  
!wget https://github.com/ultralytics/yolov5/releases/download/v6.1/yolov5s.pt
```

Figura 23. Ejemplo instalación de YOLOv5 y descarga de pesos pre-entrenados en Google Colab

Una vez tenemos acceso desde Colab a nuestro Drive, debemos clonar el repositorio de YOLOv5 de Ultralytics e instalar los requisitos del mismo. En caso de no querer trabajar con YOLOv5 desde *scratch*, es posible descargar los modelos con pesos pre-entrenados en el Dataset de COCO. Desde Colab podemos ejecutar:

4.2.2 – Jerarquía de ficheros

YOLOv5 sigue una jerarquía de ficheros estricta que debe cumplirse para que se encuentren las imágenes de forma adecuada.

Debe existir una carpeta que contenga otras tres subcarpetas; una para las imágenes de entrenamiento, otro para las imágenes de validación y una última carpeta para las imágenes de test. Cada una de estas subcarpetas contiene a su vez, por un lado, las

imágenes y por otro, las etiquetas con las coordenadas de las *bounding boxes* del *ground truth* para cada una de estas imágenes.

Además, en esta versión de YOLO se incluye un archivo yaml, con las rutas necesarias para encontrar las imágenes.

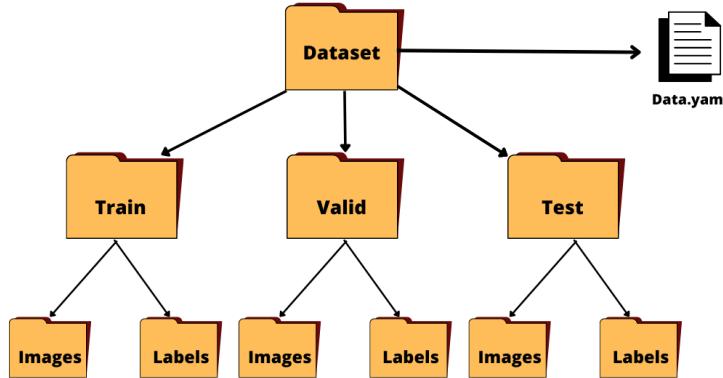


Figura 24. Jerarquía de ficheros en YOLOv5. Figura original

4.2.3 – Entrenamiento del modelo

Una vez se dispone de las imágenes etiquetadas, y se ha configurado YOLOv5 en Google Colab, se puede entrenar la red para detectar armas [13].

Para ello basta con ejecutar el fichero train.py proporcionado en el repositorio de Ultralytics. La instrucción a ejecutar será de una forma similar a la siguiente:

```
!python train.py --img 416 --batch 32 --epochs 100 --data ruta/data.yaml
--weights " --cfg models/yolov5m.yaml --name "nombre_fichero"
```

Como se puede apreciar, la ejecución de fichero de entrenamiento es configurable y se le pasan una serie de argumentos para customizarlo. A continuación, se describen algunos de los más relevantes. [14][17]

- **Image size:** hace referencia al tamaño de las imágenes (416x416). Cuanto más pequeños sean los objetos a detectar, es posible que el modelo se beneficie al usar una resolución mayor para las imágenes.
- **Batch:** el "Batch" o "Lotes", se trata del número de muestras procesadas en cada iteración del entrenamiento. En cada iteración se actualizan los pesos. Se recomienda entrenar con el mayor tamaño de batch que permita el hardware utilizado.

- **Epochs:** las "épocas" son el número de iteraciones que constituyen el entrenamiento. Define el número de veces que todo el conjunto de datos pasa a través del algoritmo de aprendizaje. Se recomienda empezar a entrenar con 300 épocas. Si se detecta overfitting, conviene reducir el número de épocas, en caso contrario, puede aumentarse.
- **Data:** con este valor se le indica al modelo la ruta al fichero de configuración yaml mencionado en apartados anteriores.
- **Weights:** son los pesos de partida del modelo. No son necesarios si el modelo se entrena desde scratch, en cuyo caso se puede definir el tamaño de la red utilizado con el argumento "--cfg". Sin embargo, podrían aplicarse técnicas de *transfer learning* y usar pesos ya entrenados en otro conjunto de datos que puedan acelerar el aprendizaje de la red.
- **Hyperparameters:** se trata de una serie de parámetros configurables que se encuentran en el fichero "hyp.scratch-low.yaml" proporcionado en el repositorio. No se recomienda modificarlo. Aun así, sus valores pueden contribuir a la aparición tardía de overfitting si fuese necesario.

Se destacan algunas de ellos:

- **Momentum:** ajuste del algoritmo de Descenso de Gradiente.
- **Scaling:** parámetro para escalar imágenes.
- **Weight-decay:** técnica de regularización que comprime los pesos en el proceso de backpropagation. Se añade un nuevo valor de penalización en el cálculo de la función de pérdida para limitar la complejidad del modelo.
- **Técnicas de Data Augmentation:** generación de nuevas imágenes a partir de las originales.
 - **Flipud:** volteo aleatorio de imágenes arriba o abajo
 - **Mosaic:** concatenación de cuatro cortes de imágenes

- **Degree:** se especifica los grados de rotación de las imágenes

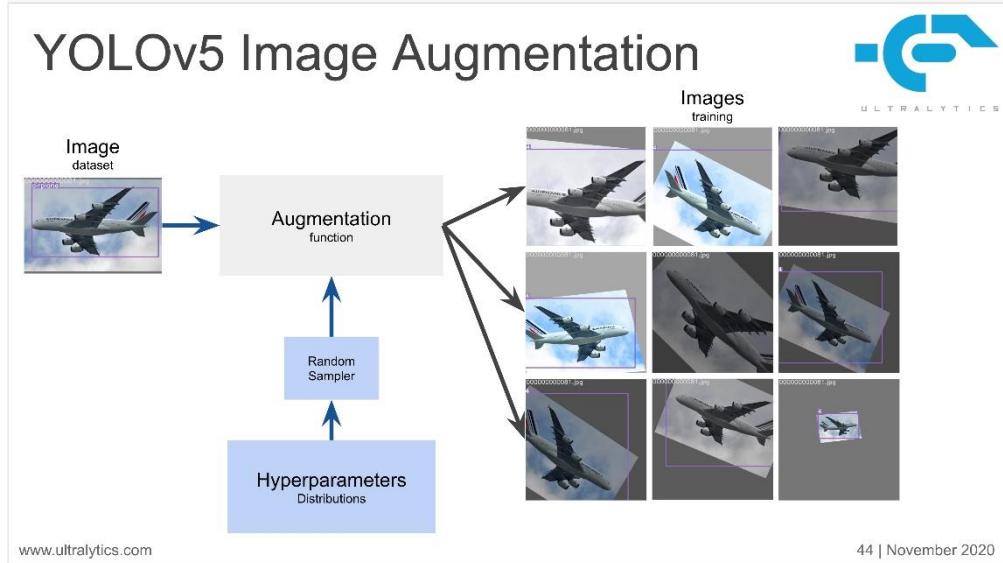


Figura 25. Esquema data augmentation en YOLOv5.
Fuente: <https://github.com/ultralytics/yolov5/issues/8021>

Como resultado de la ejecución se genera una carpeta que contiene una serie de archivos para evaluar el rendimiento del modelo. Además, se obtienen otros ficheros con los mejores pesos del entrenamiento y los últimos. A partir de esos pesos obtenidos se podrán realizar detecciones sobre nuevas imágenes.

4.2.4 – Test del modelo: Detección de objetos en nuevas imágenes

Tras entrenar la red neuronal y haber obtenido los mejores pesos en el archivo "best.pt" o en su caso los últimos pesos "last.pt", es posible realizar lo que se denomina inferencia del modelo para detectar objetos en nuevas imágenes.

Desde Google Colab basta con ejecutar el comando que utiliza los pesos mencionados:

```
!python detect.py --source ruta/test/images --weights ruta/best.pt --img 416  
--save-txt
```

Algunos de los parámetros que se usan son:

- **Source:** ruta a las imágenes sobre las que realizar la detección. Puede también indicarse una ruta a un video o a la web cam mediante el valor 0.

- **Conf-thres**: umbral de confianza que valida una detección. Su valor por defecto es de 0,25.
- **iou-thres**: umbral de la operación IoU que valida una detección. Su valor por defecto es 0,45.

De nuevo el fichero detect.py lo proporciona YOLOv5 y conviene destacar que para obtener resultados óptimos debe usarse el mismo tamaño de resolución de las imágenes de test sobre las que realizamos las detecciones, que en las imágenes de entrenamiento.

YOLOv5 genera una nueva carpeta donde guarda los resultados de las detecciones.

4.2.5 – Evaluación del desempeño del modelo

Para poder evaluar el rendimiento del modelo con las imágenes de test sobre las que previamente se han realizado detecciones, YOLOv5 proporciona un fichero ejecutable que genera automáticamente una serie de métricas, imágenes y valores que nos facilitan este trabajo.

La ejecución del mismo, es simple. Se proporciona un ejemplo:

```
!python val.py --weights ruta/last.pt --data ruta/data.yaml --workers 0
--img 416 --save-txt --conf-thres 0.5 --task test
```

Como se puede observar el nombre del fichero es val.py y basta con pasar como parámetros los pesos utilizados para la detección, el fichero de configuración yaml. Además, pueden agregarse otras configuraciones, como definir nuevos valores para los umbrales mencionados en el apartado anterior.

Una vez más, YOLOv5 genera una nueva carpeta con los resultados que podemos interpretar para determinar cómo de bueno es el modelo usado.

5. Pruebas

En este apartado nos centraremos en los experimentos realizados y la evaluación de los resultados obtenidos en cada uno de ellos.

5.1 – Métricas de evaluación

A la hora de determinar cuán bueno es el desempeño de un detector se hace uso de una serie de métricas y valores. A continuación, definiremos las más relevantes para el caso estudiado.

- **Intersection Over Union (IoU)**:

Es una de las métricas de más usuales y útiles en la detección de objetos cuando tenemos algoritmos con *Bounding Boxes* como salida.

Para poder aplicarlos necesitamos conocer las coordenadas de las *Bounding Boxes* del *Ground Truth* y las coordenadas de las *Bounding Boxes* que se han predicho. La operación a realizar se aprecia en la imagen.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{Intersection}}{\text{Union}}$$

Figura 26. Cálculo de Intersection Over Union.

Fuente: <https://iq.opengenus.org/evaluation-metrics-for-object-detection-and-segmentation/>

Normalmente se estable un umbral de IoU y si el resultado de la operación de *Intersection Over Union* del etiquetado real con la detección supera dicho valor, esa detección puede darse por válida. [27]

Se ha implementado una posible solución de esta métrica para YOLOv5. Puede consultarse en el anexo.

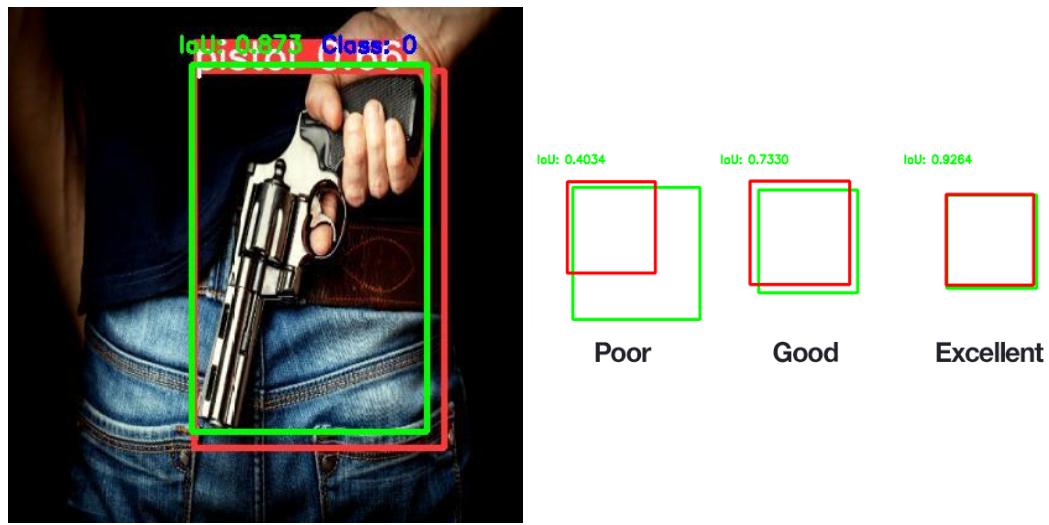


Figura 27. Ejemplo aplicación Intersection Over Union.

- **Confusion Matrix:**

Otra forma rápida y visual de valorar los resultados del detector es la matriz de confusión, que permite determinar a simple vista si la detección y clasificación de los objetos ha sido buena. [24]

Los valores devueltos en la matriz de confusión dependerán de los umbrales de IoU y de confianza establecidos, indicando este último el grado de confianza de la predicción.

- **True Positive (TP):** detección que supera el umbral IoU y de confianza. Además, la clasificación es buena.
- **True Negative (TN):** detección que no supera ninguno de los umbrales de IoU y confianza establecidos.
- **False Positive (FP):** detección que supera el umbral de confianza de la red, pero no supera el umbral de IoU o la clasificación no es buena
- **False Negative (FN):** detección que no supera el umbral de confianza, pero si el de IoU.

		VALORES REALES	
		POSITIVO	NEGATIVO
VALORES PREDICHIOS	POSITIVO	TP	FP
	NEGATIVO	FN	TN

Annotations:

- TP: El valor predicho y el real son positivos
- FP: El valor predicho es positivo y el real negativo
- TN: El valor predicho y el real son negativos
- FN: El valor predicho es negativo y el real positivo

Figura 28. Ejemplo matriz de confusión. Figura original

- **Precision:**

La precisión se entiende como el número de detecciones correctas de entre todas las detecciones calculadas por el modelo [19]. Matemáticamente puede calcularse con la fórmula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall:**

El *recall* indica si el modelo calcula correctamente las detecciones. Es decir, si ha sido capaz de detectar correctamente los objetos que debería haber detectado [19]. Matemáticamente puede calcularse con la fórmula:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Precision-Recall curve:**

Muestra cómo cambia el *recall* en función de la precisión y viceversa. Normalmente se representa gráficamente.

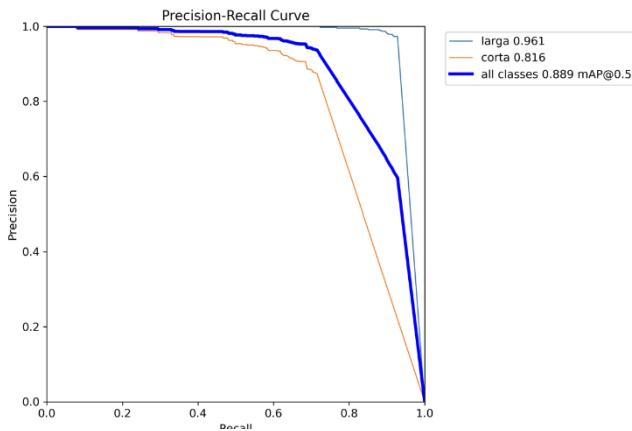


Figura 29. Ejemplo gráfica precision-recall

- **Average Precision (AP):**

Es la media ponderada de las precisiones de cada umbral, usando como peso el aumento del *recall* del umbral anterior. Se puede definir como el área interpolada por debajo de la curva de *precision-recall*. [12]

- **F1-Score:**

La métrica F1-score se trata de una media armónica entre la precisión y el *recall*. [24]

Cuando la precisión incrementa, el valor de *recall* disminuye y viceversa. Esta métrica permite tener una idea combinada de ambos valores y ayuda a encontrar el umbral de confianza óptimo.

$$F1 - score = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

- **Mean Average Precision (mAP):**

Esta métrica es el promedio de las medias de las precisiones de todas las clases de un modelo. Su valor se encuentra entre 0 y 1. Podemos usarla para comparar rendimiento entre distintos modelos o versiones de un modelo.

Para calcular el valor mAP de un modelo, primero se debe definir un umbral IoU. Ya sea un único valor (mAP@0,5) o un rango de valores con incrementos (mAP@0,5:0,95) calculando así el promedio de los valores mAP de cada rango.

Se dividen las detecciones en grupos según la clase detectada y se calcula el valor AP de cada grupo. Luego se hace la media de dichos valores resultando en un mAP para un cierto umbral IoU especificado. [12]

5.2 – Primeras pruebas

Antes de exponer los principales experimentos realizados en este trabajo, conviene comentar una serie de pruebas preliminares realizadas con el fin último de mejorar la calidad y cantidad de los datos con los que posteriormente trabajaremos.

Cabe mencionar que en estas pruebas trabajamos con versiones reducidas del dataset que no utilizaremos en los experimentos, pero que nos permiten, precisamente, determinar cómo puede afectar la escasez de datos a los resultados del detector.

De este modo, a continuación, se exponen las conclusiones obtenidas tras esta primera toma de contacto con el algoritmo.

- Los datos deben etiquetarse seleccionando solo las zonas visibles del arma.
- Debido a la variabilidad de las condiciones de la imagen, así como el tamaño, formas o posiciones de las armas presentes, se necesita un número alto de imágenes para lograr buenos resultados.
- Conviene mantener el *aspect-ratio* de la imagen, este caso con un relleno negro para los llamados *Dead Pixels*. Las imágenes de entrada tendrán un tamaño de 416x416.
- Normalizando las imágenes aplicando el preprocessado de *Contrast-Stretch* que proporciona Roboflow se observa una mejoría en la detección.
- Los resultados mejoran al partir de un modelo pre-entrenado con el dataset de COCO. En este caso la diferencia entre entrenar con la red YOLOv5s y YOLOv5m no es determinante a nivel resultados, por ello se trabajará con la versión *small* de la red, ya que reduce los tiempos de entrenamiento.

5.3 – Experimentos

Para los siguientes experimentos se han tenido en cuenta las conclusiones obtenidas en el apartado anterior que se aplican a los datos con los que se trabaja.

En cada experimento, se expondrán los resultados de las métricas y se hará un breve análisis de los resultados obtenidos.

5.3.1 - Experimento 1

- Descripción del experimento

El objetivo de este primer experimento es entrenar un modelo capaz de detectar de manera efectiva armas de fuego y clasificarlas en “armas largas” y “armas cortas”.

Se entrenará el modelo partiendo de los pesos pre-entrenados con el dataset de COCO de dos formas diferentes:

- Partiendo de una red de tamaño pequeño, YOLOv5s.
- Partiendo de una red de tamaño mediano, YOLOv5m.

Se usarán los mismos hiperparámetros para ambos modelos de YOLOv5.

Se comparará el desempeño de ambas redes y el tiempo de entrenamiento de cada una de ellas.

- Entrenamiento del modelo

YOLOv5s			
Tamaño Batch	Épocas	Tamaño Imágenes	Tiempo (h)
32	100	416	1.220

Tabla 4. Experimento 1. YOLOv5s. Resumen parámetros empleados

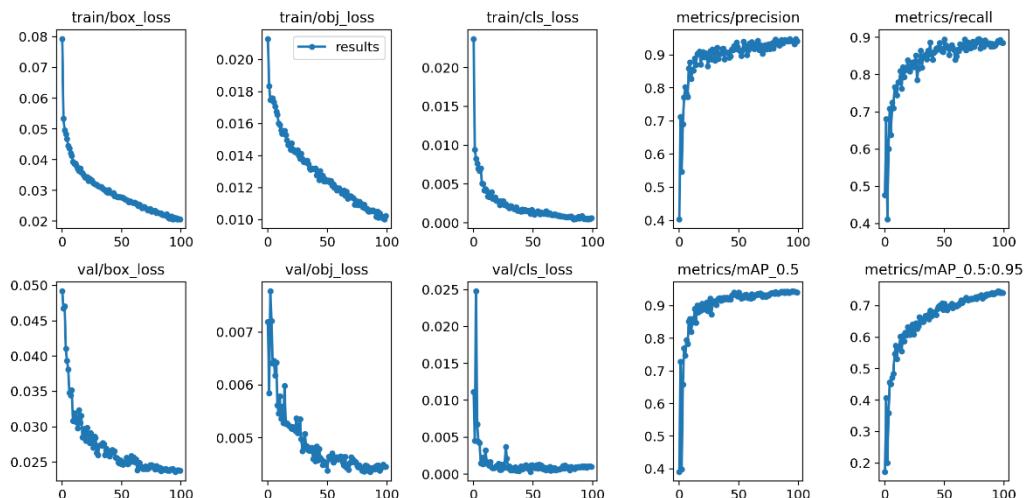


Figura 30. Experimento 1. YOLOv5s. Gráficas resultado entrenamiento

YOLOv5m			
Tamaño Batch	Épocas	Tamaño Imágenes	Tiempo (h)
32	100	416	2.893

Tabla 5. Experimento 1. YOLOv5m. Resumen parámetros empleados

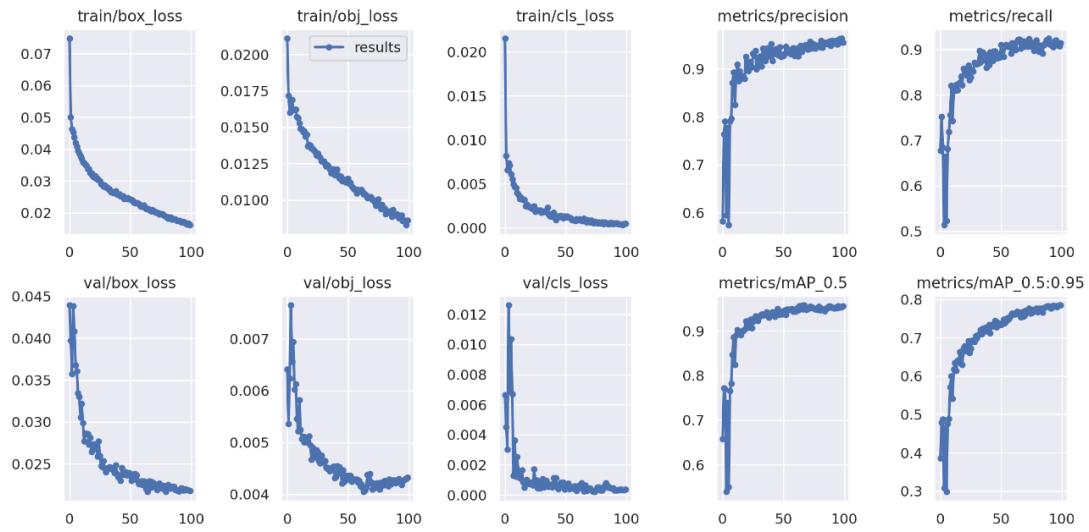


Figura 31. Experimento 1. YOLOV5m. Gráficas resultado entrenamiento

Tanto en la figura A como en la figura B se observa como los resultados son bastante aceptables y mejoran con el incremento de las iteraciones. Los valores de los hiperparámetros usados dan buenos resultados.

No parece que exista *overfitting*. Para afirmar esto último, se deben observar las gráficas “val/boc_loss”, “val/obj_loss” y “val/cls_loss”. A lo largo de las iteraciones, su valor debe decrecer. Si, por el contrario, aumentase, sería un indicativo de *overfitting*. [20]

Como solución, se podría proponer una reducción en el número de épocas de entrenamiento del modelo o incluso aumentar el volumen de los datos con los que estamos trabajando. En este caso, no será necesario aplicar ninguna de estas técnicas.

- Evaluación del desempeño del modelo

YOLOv5s							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	1024	0.921	0.822	0.87	0.889	0.695
larga	892	502	0.973	0.928	-	0.961	0.839
corta	892	522	0.87	0.716	-	0.816	0.552

Tabla 6. Experimento 1. YOLOv5s. Resultados evaluación modelo

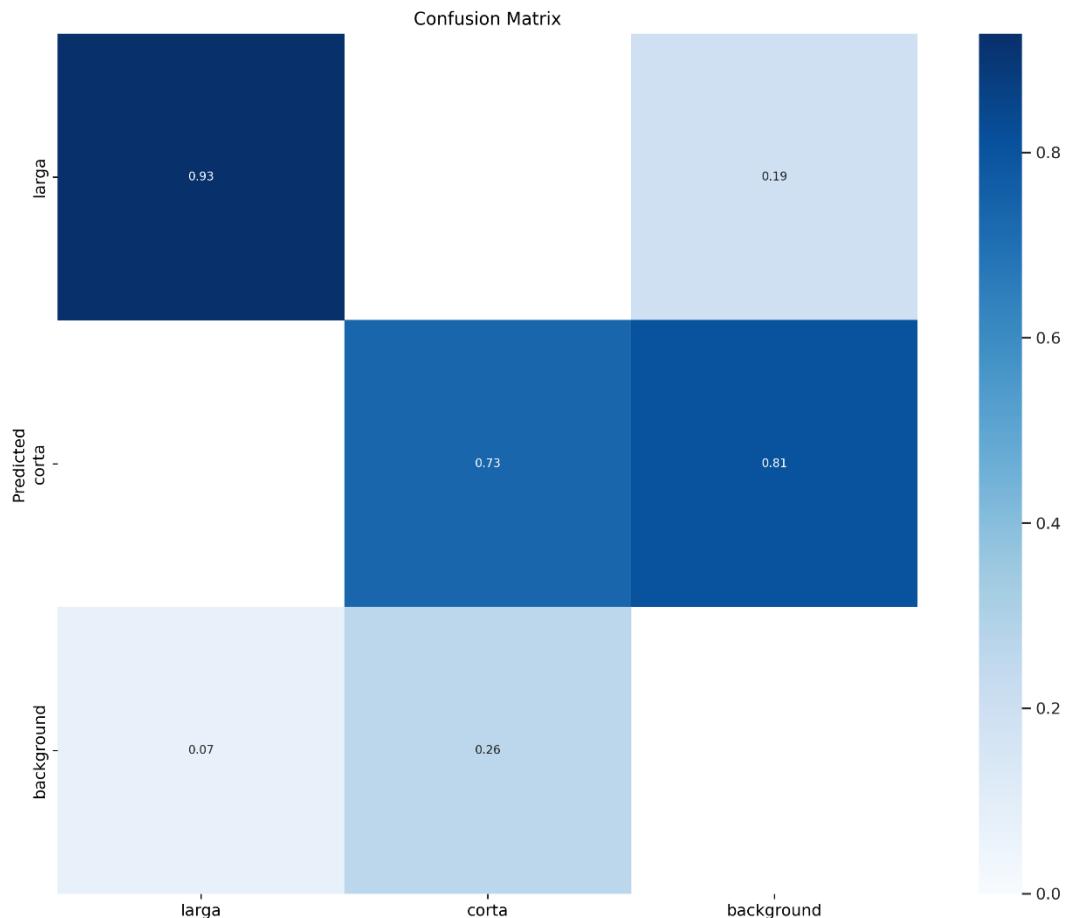


Figura 32. Experimento 1. YOLOv5s. Matriz de confusión

Como se puede observar en la matriz de confusión, se detectan el 93% de las armas largas y el 73% de las armas cortas correctamente. Por tanto, el 7% de las armas largas y el 26% de las cortas no son detectadas siendo clasificadas como *background*.

Debemos considerar también los falsos positivos. En este caso, dado que la precisión y el recall son muy altos, podemos afirmar que las detecciones falsas son mínimas. Como conocemos el número de instancias totales y de cada clase, así como la precisión y el *recall*. Observando los datos de la matriz de confusión, calcular el número de falsos positivos es sencillo. Basta con despejar su valor de cualquiera de las fórmulas del *recall* o la precisión.

Así, se detectan 73 falsos positivos en un total 892 imágenes con 1024 instancias. De estas detecciones, el 81% son falsamente clasificadas como armas cortas, y el resto como largas.

En las figuras 33 y 34 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.

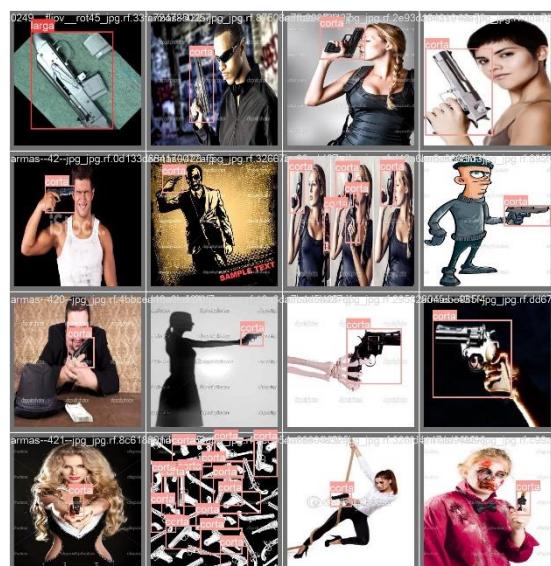


Figura 33. Experimento 1. YOLOv5s. Etiquetas reales.

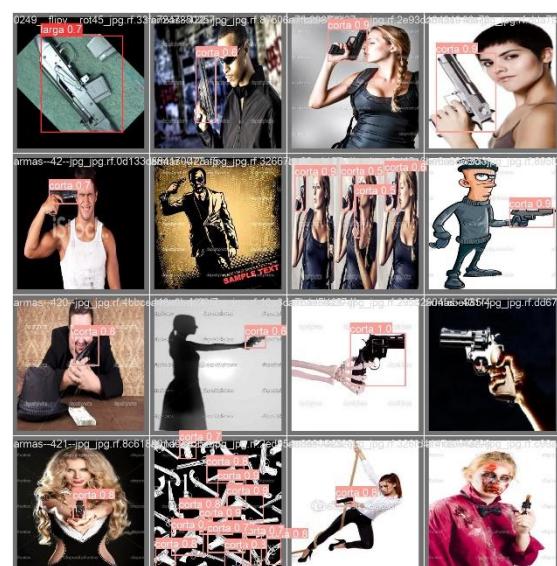


Figura 34. Experimento 1. YOLOv5s. Detecciones.

YOLOv5m							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	1024	0.955	0.832	0.89	0.9	0.73
larga	892	502	0.994	0.922	-	0.959	0.883
corta	892	522	0.917	0.741	-	0.842	0.577

Tabla 7. Experimento 1.YOLOv5m. Resultados evaluación modelo

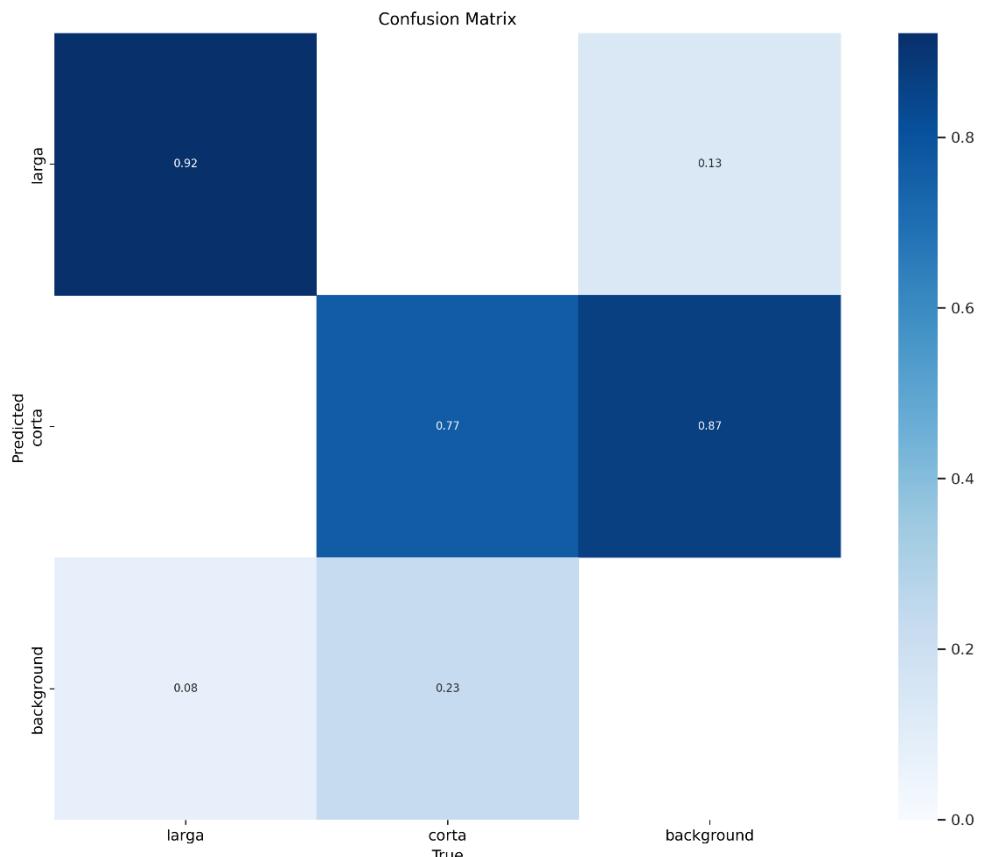


Figura 35. Experimento 1. YOLOv5m. Matriz de confusión

Como se puede observar en la matriz de confusión, se detectan el 92% de las armas largas y el 77% de las armas cortas correctamente. Por tanto, el 8% de las armas largas y el 23% de las cortas no son detectadas siendo clasificadas como *background*.

En este caso se detectan 41 falsos positivos en un total 892 imágenes con 1024 instancias. De estas detecciones, el 87% son falsamente clasificadas como armas cortas, y el 13% restante como largas.

En las figuras 36 y 37 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.



Figura 37.Experimento 1. YOLOv5m. Etiquetas reales

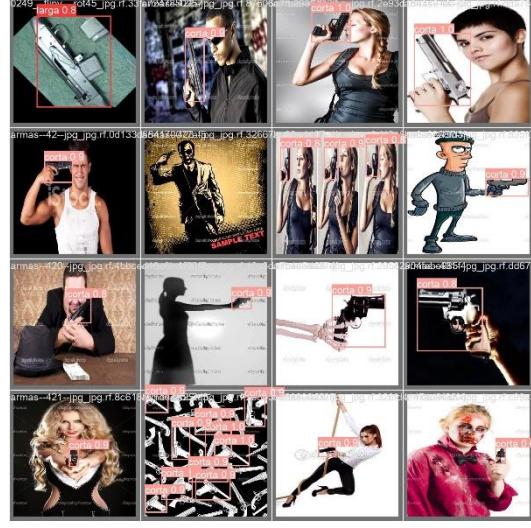


Figura 36.Experimento 1. YOLOv5m. Detecciones

Como se puede ver la diferencia a nivel de detecciones no es grande. Si bien es cierto que con el modelo YOLOv5m se obtienen resultados ligeramente mejores, los resultados del modelo YOLOv5s no se alejan mucho de estos últimos.

Sin embargo, se aprecia un aumento considerable en el tiempo de entrenamiento con el modelo YOLOv5m en la nube. En caso de disponer de una GPU local adecuada, probablemente los tiempos de entrenamiento se verían reducidos y convendría usar una red mayor para maximizar los resultados.

Para los siguientes experimentos, usaremos los pesos obtenidos tras el entrenamiento con YOLOv5s.

5.3.2 - Experimento 2

- Descripción del experimento

En este experimento se estudia como el tamaño de las instancias afecta al modelo construido.

Para ello se han dividido las imágenes de test del dataset que contienen una única instancia, en 3 conjuntos, procurando que el número de imágenes entre conjuntos sea proporcionado:

- Conjunto de **instancias pequeñas** →
Ocupan entre el 0% y el 15% de la imagen.
- Conjunto de **instancias medianas** →

Ocupan entre el 016% y el 35% de la imagen.

- Conjunto de **instancias grandes** →

Ocupan entre el 36% y el 100% de la imagen.

La división se realiza teniendo en cuenta el tamaño de las imágenes (416x416) y calculando el tamaño de la *Bounding Box* de la instancia de la imagen.

Lo siguiente es evaluar las detecciones con el modelo ya entrenado previamente sobre cada uno de estos conjuntos.

- Evaluación del desempeño del modelo

Instancias pequeñas							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	279	279	0.906	0.8	0.85	0.869	0.629
larga	279	74	0.972	0.932	-	0.962	0.804
corta	279	205	0.84	0.668	-	0.776	0.454

Tabla 8. Experimento 2. Instancias pequeñas. Resultados evaluación modelo

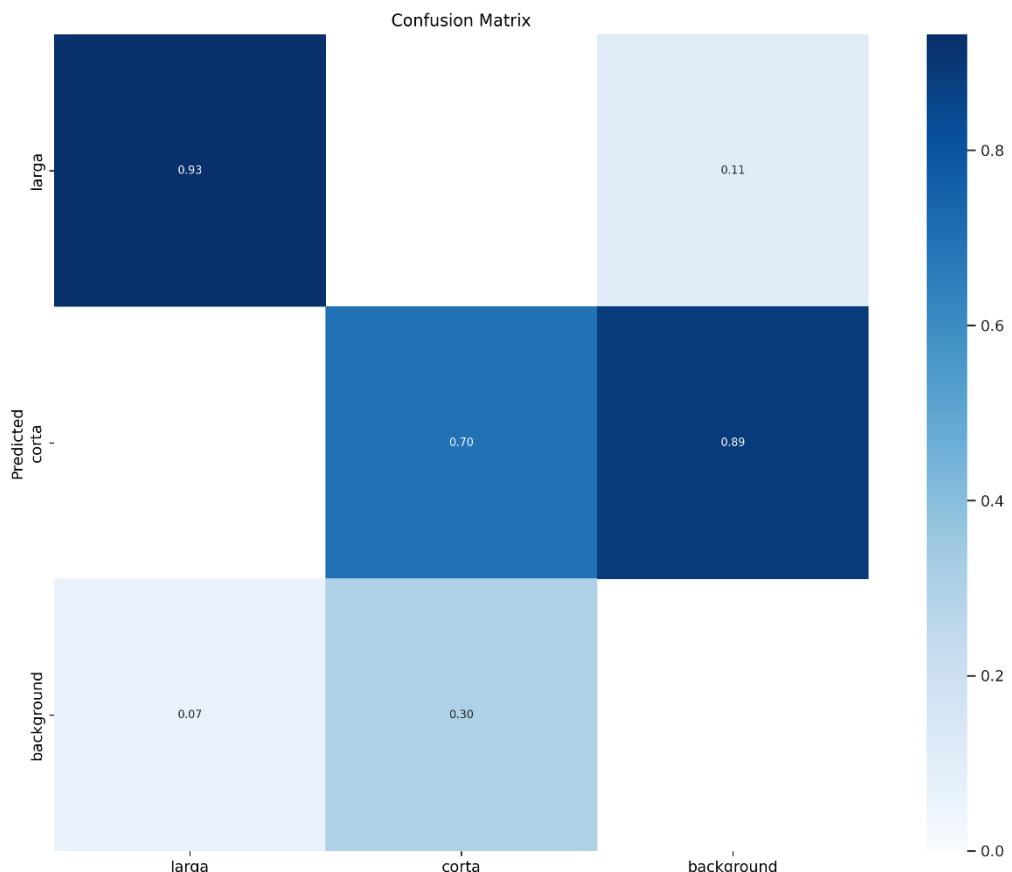


Figura 38. Experimento 2. Instancias pequeñas. Matriz de confusión

Según la matriz de confusión, se detectan el 93% de las armas largas y el 70% de las armas cortas correctamente para instancias pequeñas. Por tanto, el 7% de las armas largas y el 30% de las cortas no son detectadas siendo clasificadas como *background*.

En este caso se detectan 22 falsos positivos en un total de 279 imágenes. De estas detecciones, el 89% son falsamente clasificadas como armas cortas, y el 11% restante como largas.

En las figuras 39 e 40 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.



Figura 39.Experimento 2. Instancias pequeñas.
Etiquetas reales



Figura 40.Experimento 2. Instancias pequeñas. Detecciones

Instancias medianas							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	266	266	0.938	0.876		0.919	0.7
larga	266	204	0.986	0.971		0.991	0.874
corta	266	62	0.89	0.781		0.847	0.526

Tabla 9. Experimento 2. Instancias medianas. Resultados evaluación modelo

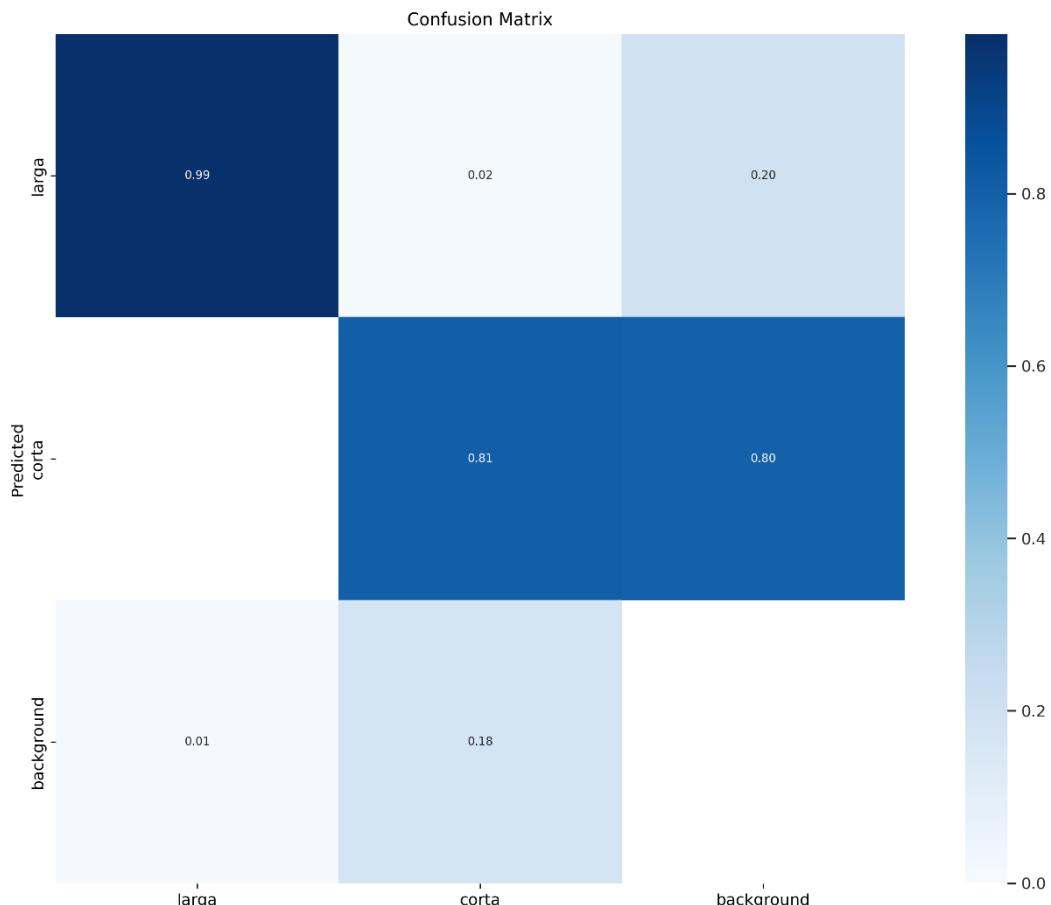


Figura 41. Experimento 2. Instancias medianas. Matriz de confusión

Según la matriz de confusión, se detectan el 99% de las armas largas y el 81% de las armas cortas correctamente para instancias pequeñas. Por tanto, el 1% de las armas largas y el 18% de las cortas no son detectadas siendo clasificadas como *background*.

En este caso se detectan 17 falsos positivos en un total de 266 imágenes. De estas detecciones, el 80% son falsamente clasificadas como armas cortas, y el 20% restante como largas.

En las figuras 42 y 43 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.



Figura 43.Experimento 2. Instancias medianas.
Etiquetas reales



Figura 42. Experimento 2. Instancias medianas. Detecciones

Instancias grandes							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	260	260	0.973	0.938		0.962	0.815
larga	260	114	0.991	1		0.995	0.888
corta	260	146	0.955	0.877		0.93	0.742

Tabla 10.Experimento 2. Instancias grandes. Resultados evaluación modelo

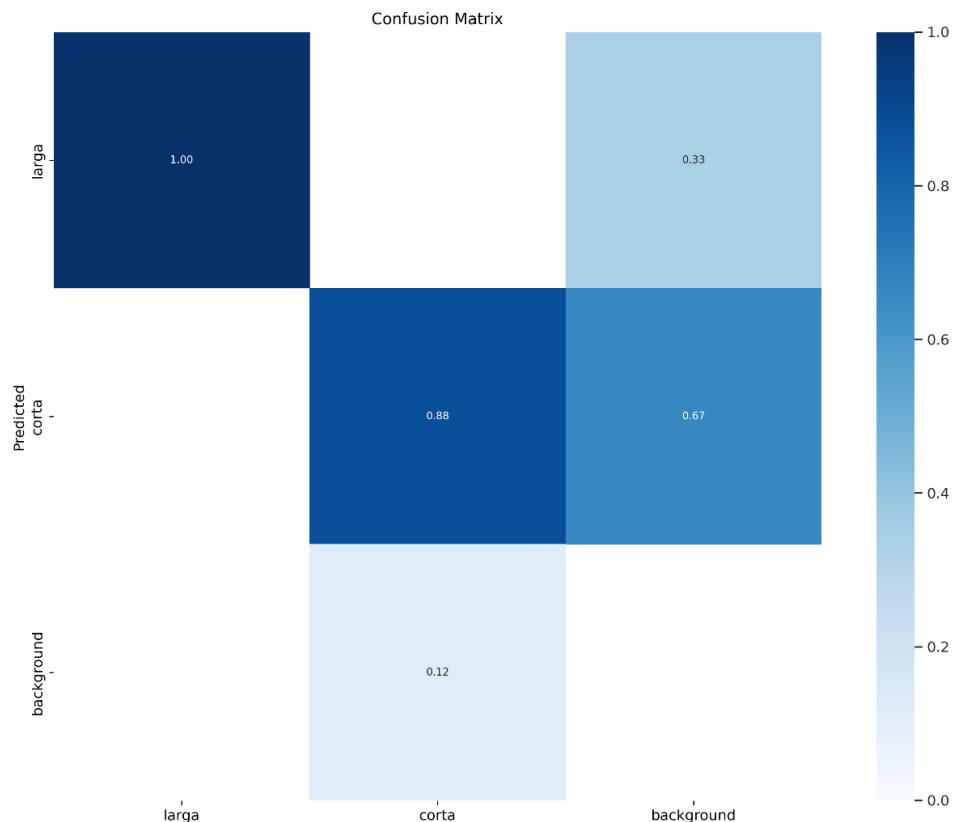


Figura 44.Experimento 2. Instancias grandes. Matriz de confusión

Según la matriz de confusión, se detectan el 100% de las armas largas y el 88% de las armas cortas correctamente para instancias pequeñas. Por tanto, solo el 12% de las armas cortas no se detectan.

En este caso se detectan tan solo 7 falsos positivos en un total de 260 imágenes. De estas detecciones, el 67% son falsamente clasificadas como armas cortas, y el 33% restante como largas.

En las figuras 45 y 46 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.



Figura 46.Experimento 2. Instancias grandes.
Etiquetas reales

Figura 45. Experimento 2. Instancias grandes. Detecciones

Se observa por tanto que cuanto mayor es el tamaño de las instancias, mejores realiza la detección el modelo construido.

También se puede intuir una mayor dificultad del modelo para clasificar cuando el tamaño del arma corta y arma larga es similar. Esto podría deberse a la falta de armas largas de tamaño pequeño en el dataset que provoque algún tipo de sesgo en la clasificación.

5.3.3 - Experimento 3

- Descripción del experimento

En este caso, se busca determinar el efecto del ruido en las imágenes.

El modelo construido no se ha entrenado para enfrentarse a imágenes que presenten ruido.

Aplicaremos a las imágenes de test ruido de tipo “sal y pimienta” en mayor y menor proporción para determinar hasta qué punto se ven los resultados afectados por este. Para ello usaremos la herramienta “[Image Augmentor](#)” y generaremos 3 subconjuntos de las imágenes de test:

- Conjunto de **ruido 0.01**
 - Conjunto de **ruido 0.02**
 - Conjunto de **ruido 0.05**
- Evaluación del desempeño del modelo

Ruido 0.01							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	1024	0.912	0.562	0.68	0.752	0.564
larga	892	502	0.932	0.715	-	0.839	0.687
corta	892	522	0.891	0.408	-	0.664	0.44

Tabla 11. Experimento 3. Ruido 0.01. Resultados evaluación modelo

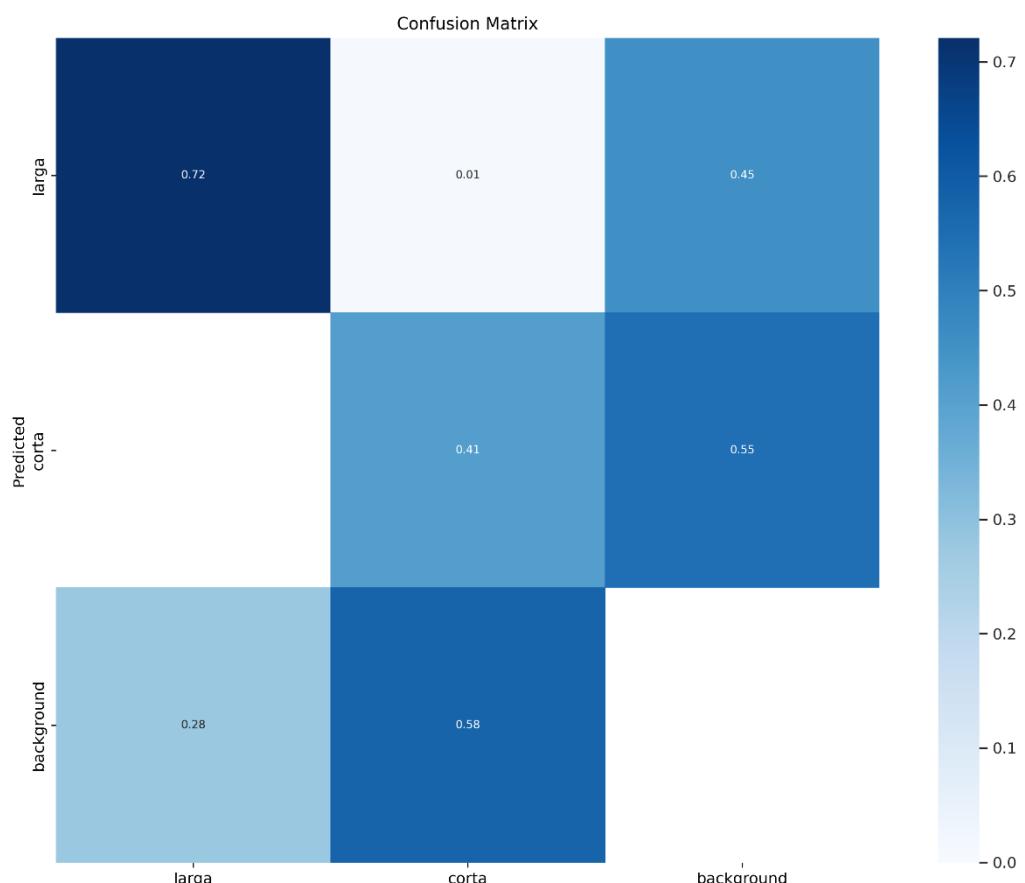


Figura 47. Experimento 3. Ruido 0.01. Matriz de confusión

Según la matriz de confusión, se detectan el 72% de las armas largas y el 41% de las armas cortas correctamente. El 28% de las largas y el 58% de las cortas no se detectan y además, el 1% de las cortas se clasifica incorrectamente como armas largas.

En este caso se detectan tan solo 56 falsos positivos en un total de 1024 instancias en 892 imágenes. De estas detecciones, el 55% son falsamente clasificadas como armas cortas, y el 45% restante como largas.

En las imágenes 48 e 49 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.

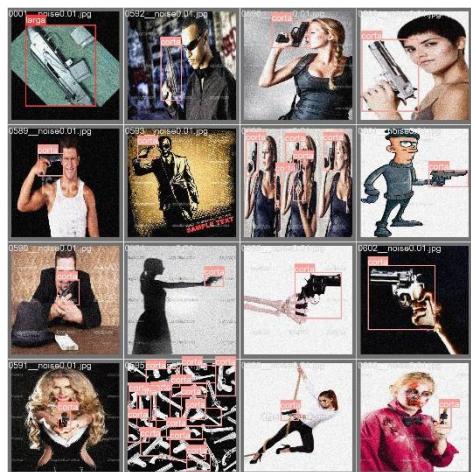


Figura 49. Experimento 3.Ruido 0.01.
Etiquetas reales

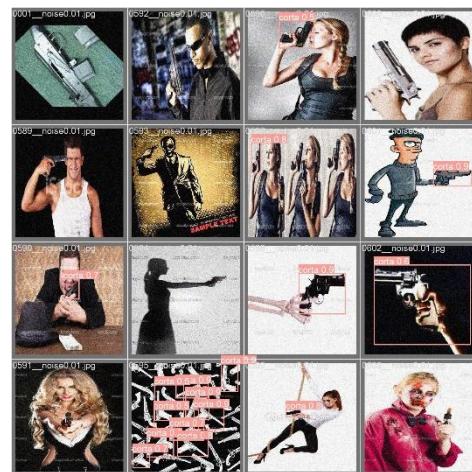


Figura 48. Experimento 3. Ruido 0.01. Detecciones

Ruido 0.02							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	1024	0.905	0.391	0.52	0.659	0.494
larga	892	502	0.908	0.57	-	0.755	0.592
corta	892	522	0.902	0.213	-	0.563	0.397

Tabla 12. Experimento 3.Ruido 0.02. Resultados evaluación modelo

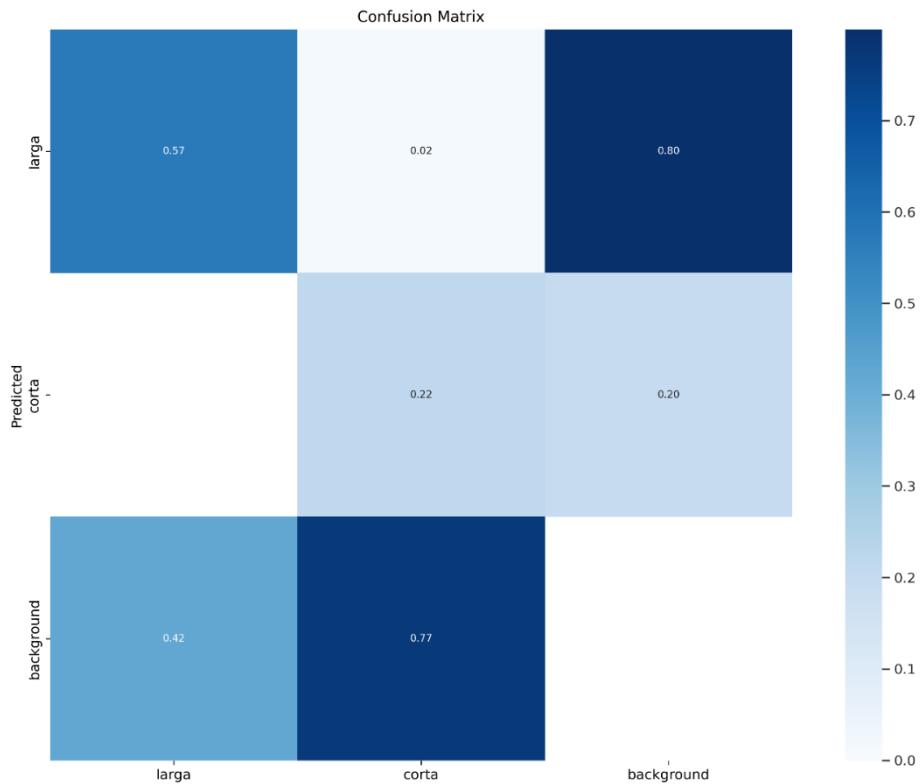


Figura 50. Experimento 3.Ruido 0.02. Matriz de confusión

Según la matriz de confusión, se detectan el 57% de las armas largas y el 22% de las armas cortas correctamente. El 42% de las largas y el 77% de las cortas no se detectan y además, el 2% de las cortas se clasifica incorrectamente como armas largas.

En este caso se detectan 42 falsos positivos en un total de 1024 instancias en 892 imágenes. De estas detecciones, el 20% son falsamente clasificadas como armas cortas, y el 80% como largas.

En las figuras 51 y 52 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza.



Figura 52.Experimento 3. Ruido 0.02. Etiquetas reales



Figura 51.Experimento 3.Ruido 0.02. Detecciones

Ruido 0.05							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	1024	0.877	0.151	0.24	0.52	0.402
larga	892	502	0.888	0.253	-	0.58	0.445
corta	892	522	0.867	0.0498	-	0.46	0.359

Tabla 13. Experimento 3.Ruido 0.05. Resultados evaluación modelo

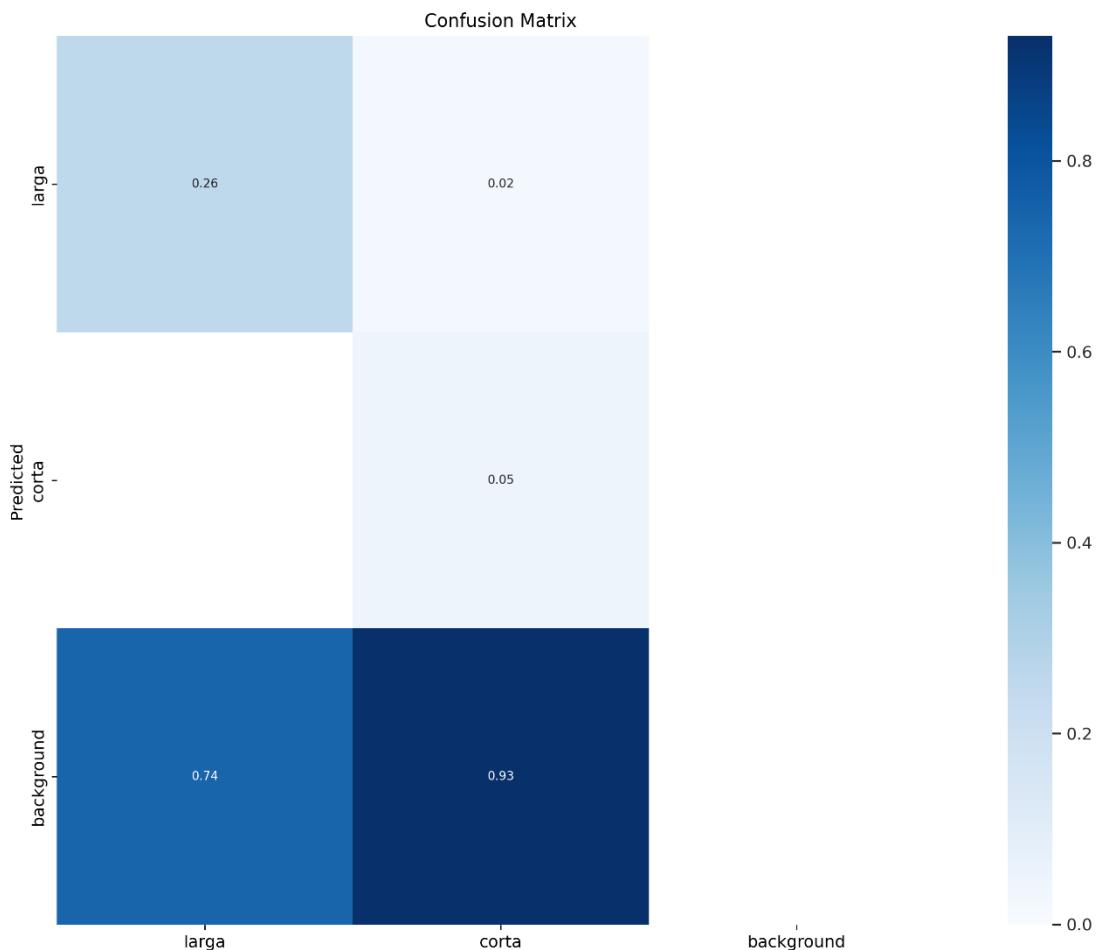


Figura 53. Experimento 3.Ruido 0.05. Matriz de confusión

Según la matriz de confusión, se detectan el 26% de las armas largas y el 5% de las armas cortas correctamente. El 74% de las largas y el 93% de las cortas no se detectan y además, el 2% de las cortas se clasifica incorrectamente como armas largas.

En las figuras 54 y 55 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza



Figura 55. Experimento 3.Ruido 0.05.
Etiquetas reales



Figura 54.Experimento 3. Ruido 0.05. Detecciones

Los resultados de las pruebas de este experimento indican claramente que el ruido en la imagen tiene un gran impacto en el desempeño del modelo. Los resultados empeoran dramáticamente con el aumento del ruido en las imágenes.

5.3.4 - Experimento 4

- Descripción del experimento

El objetivo de este experimento es determinar cómo las occlusiones afectan al modelo construido. Con este fin, se ha decidido incluir occlusiones en las imágenes de test usando las opciones que proporciona Roboflow.

Se oculta el 30% de la imagen de 3 formas diferentes:

- **1** occlusion que ocupa el **30%** de la imagen
- **2** occlusiones que ocupen cada una un **15%** de la imagen
- **3** occlusiones que ocupan cada una un **10%** de la imagen

- Evaluación del desempeño del modelo

1 Occlusion del 30%							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	892	0.76	0.719	0.74	0.728	0.49
larga	892	444	0.788	0.82	-	0.789	0.595
corta	892	448	0.733	0.618	-	0.667	0.385

Tabla 14. Experimento 4. Una occlusion. Resultados evaluación modelo

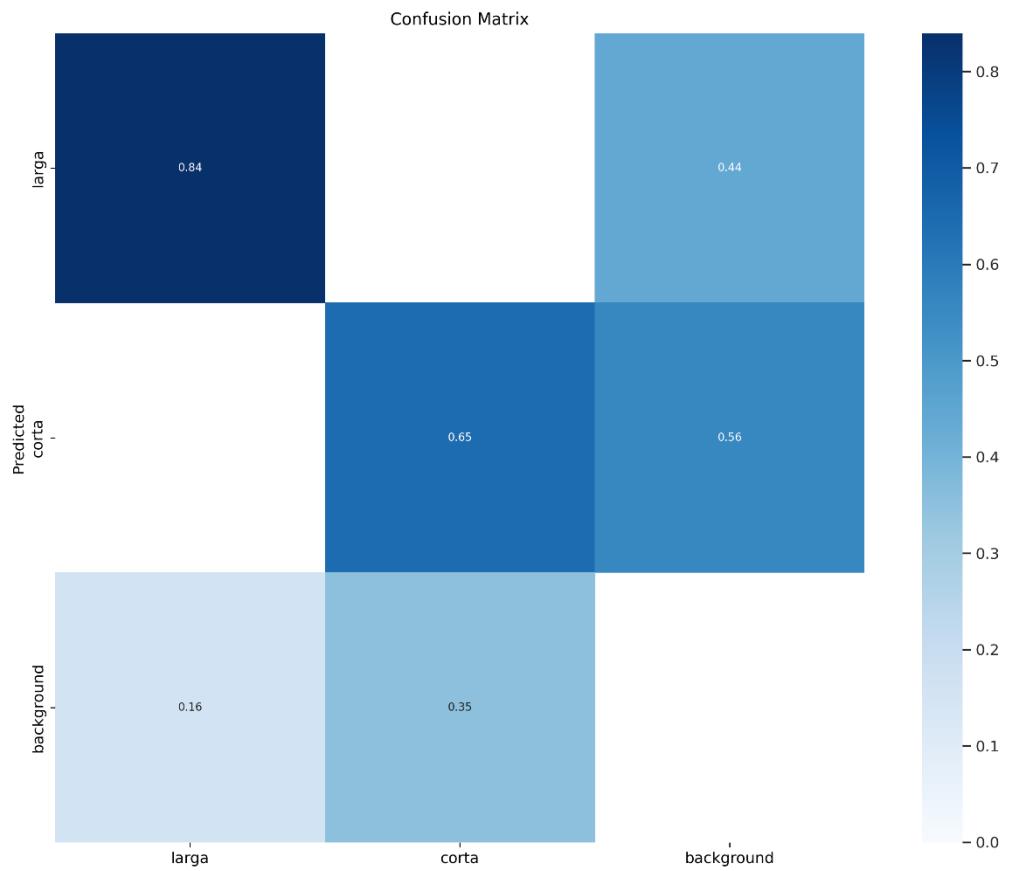


Figura 56. Experimento 4. Una oclusión. Matriz de confusión

La matriz de confusión muestra cómo se detectan el 84% de las armas largas y el 65% de las armas cortas correctamente. El 16% de las largas y el 53% de las cortas no se detectan.

En las imágenes 57 e 58 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza



Figura 58.Experimento 4. Una oclusión. Etiquetas reales



Figura 57.Experimento 4. Una oclusión. Detecciones

2 Oclusiones del 15%							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	892	0.795	0.763	0.78	0.765	0.548
larga	892	444	0.832	0.858	-	0.833	0.658
corta	892	448	0.759	0.667	-	0.697	0.439

Tabla 15.Experimento 4. Dos occlusiones. Resultados evaluación modelo

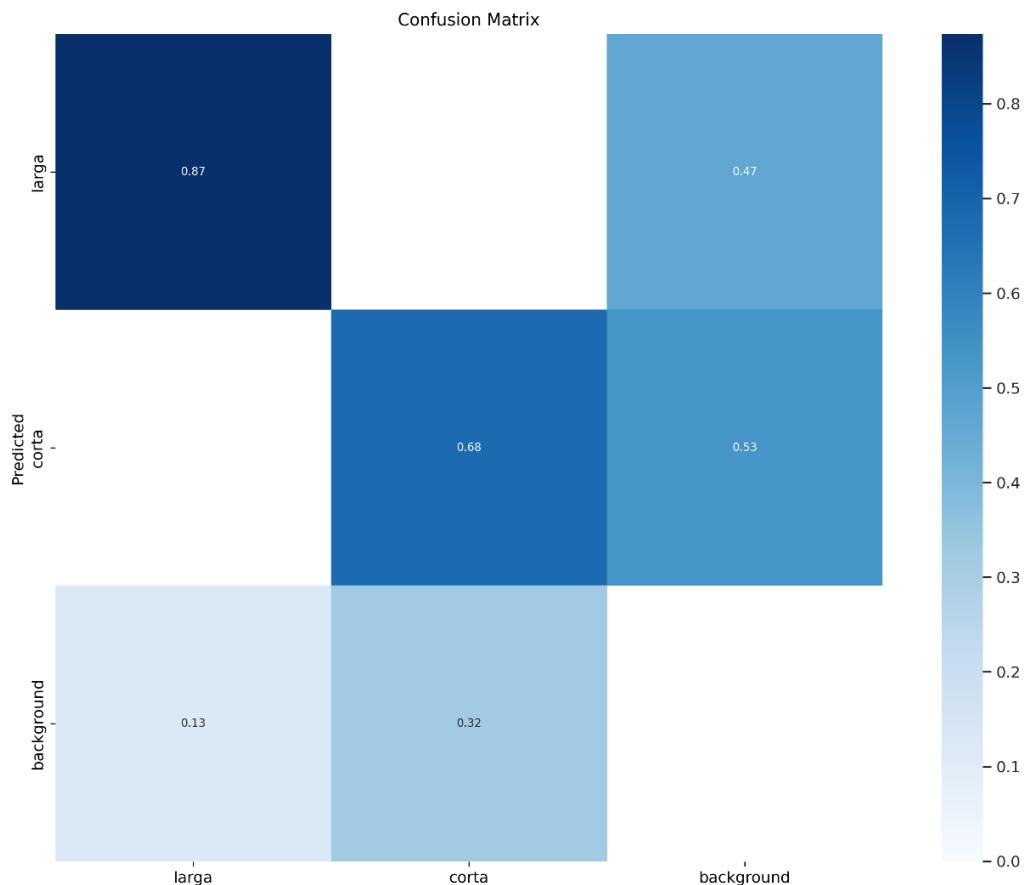


Figura 59.Experimento 4. Dos occlusiones. Matriz de confusión

Según la matriz de confusión, se detectan el 87% de las armas largas y el 68% de las armas cortas correctamente. El 13% de las largas y el 32% de las cortas no se detectan.

En las figuras 60 y 61 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza

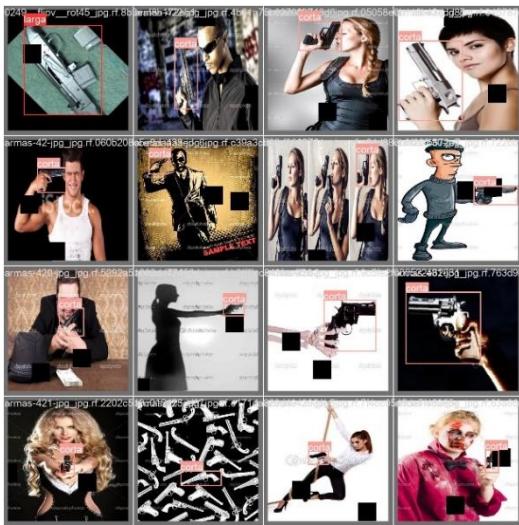


Figura 60.Experimento 4. Dos oclusiones.
Etiquetas reales



Figura 61.Experimento 4. Dos oclusiones. Detecciones

3 Oclusiones del 10%							
Clases	Imágenes	Instancias	Precision	Recall	F1	mAP50	mAP50-95
todas	892	892	0.811	0.8	0.80	0.795	0.595
larga	892	444	0.849	0.901	-	0.863	0.723
corta	892	448	0.773	0.901	-	0.727	0.466

Tabla 16.Experimento 4. Tres oclusiones. Resultados evaluación modelo

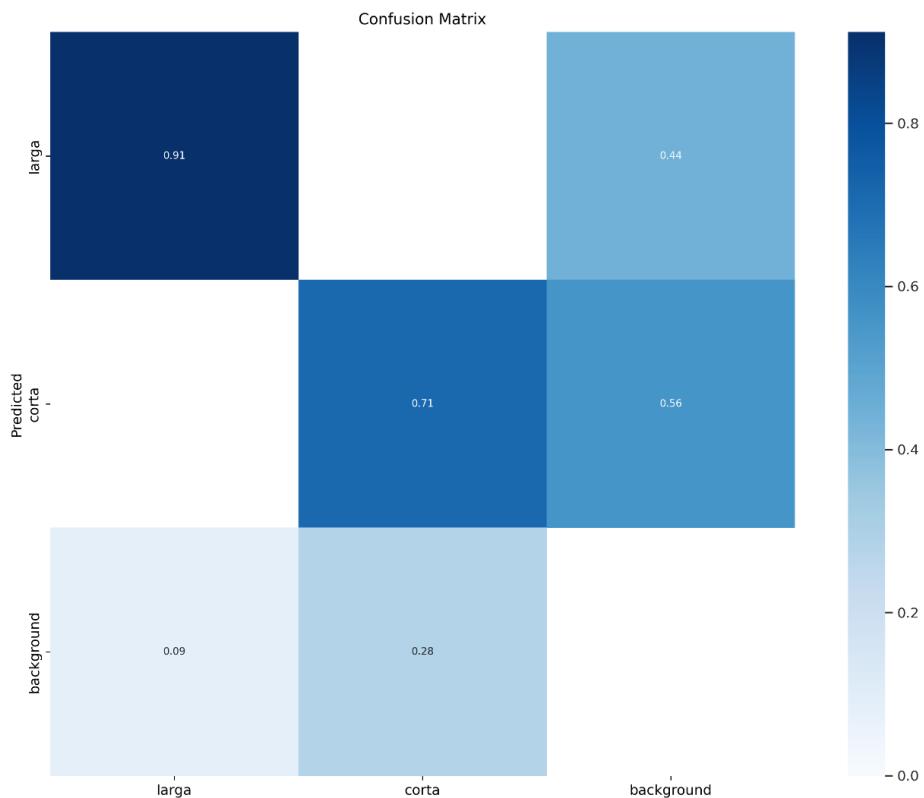
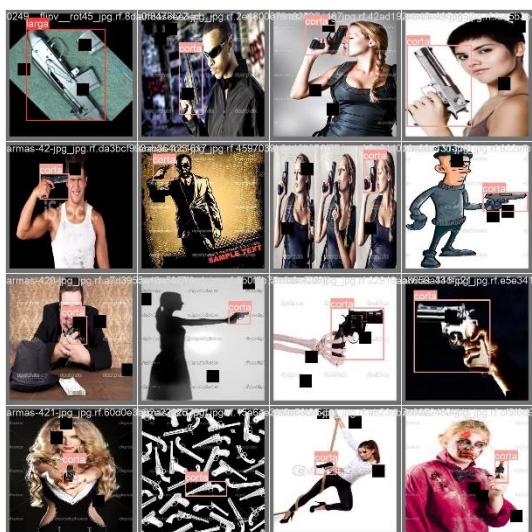


Figura 62.Experimento 4. Tres oclusiones. Matriz de confusión.

Según la matriz de confusión, se detectan el 91% de las armas largas y el 71% de las armas cortas correctamente. El 9% de las largas y el 28% de las cortas no se detectan.

En las figuras 63 y 64 se observan algunas de las etiquetas reales y las detecciones obtenidas para esos mismos objetos con su valor de confianza



*Figura 64.Experimento 4. Tres occlusiones.
Etiquetas reales*



Figura 63.Experimento 4. Tres occlusiones. Detecciones.

Por tanto, se puede afirmar que cuanto mayor es el tamaño de las occlusiones en la imagen, peor realiza las detecciones el modelo construido.

6. Conclusión

Este apartado se centra en el análisis general de los resultados obtenidos en los experimentos realizados, así como en las futuras líneas de trabajo futuro a seguir con el fin de mejorar y completar este estudio.

6.1 – Conclusiones generales

Analizando el trabajo realizado se puede afirmar que los objetivos mencionados previamente en este documento se han cumplido. Se ha elaborado un dataset con el tamaño adecuado de datos como para obtener resultados satisfactorios tras entrenar el modelo y realizar una serie de pruebas con el mismo sobre el conjunto de imágenes de test. Así mismo, se ha aprendido a usar el marco de trabajo de Roboflow para la elaboración del dataset mencionado, tanto para el etiquetado de las imágenes como para el preprocesado de las mismas. Además, se ha profundizado en la historia, arquitectura y uso de YOLOv5 y sus modelos.

Por otro lado, de los resultados de experimentos realizados pueden extraerse algunas conclusiones. Primeramente, cabe mencionar que la variabilidad en los tipos de armas dentro de cada una de las clases elegidas, dificultad el trabajo de la red que necesita aprender de un número elevado de ejemplos. Se concluye también que las redes de mayor tamaño aumentan la calidad de los resultados disponiendo de los mismos datos, sin embargo, el entrenamiento se alarga considerablemente en tiempo.

Otro dato a destacar es como el tamaño de la instancia a detectar dificulta su detección. Las imágenes que contienen armas que ocupan en torno al 50% o más de imagen se detectan con mayor seguridad. En relación con esto, los datos arrojan más información. Las armas clasificadas correctamente como largas en general se detectan mejor en el conjunto completo de experimentos. Esto puede deberse, en parte, precisamente a que el tamaño de las instancias de armas largas que componen el dataset construido sea, en proporción, mayor al de las instancias de armas cortas. YOLOv5 parece detectar peor los objetos pequeños.

Se ha observado también como el ruido en la imagen deteriora dramáticamente los resultados de las detecciones, siendo conveniente aplicar algún tipo de preprocesado previo que lo elimine antes de intentar hacer detecciones con el modelo construido. De la misma forma, se identifica un claro detrimento en el desempeño del modelo cuando se encuentra ante occlusiones que ocultan parte de las instancias a reconocer, siendo peores los resultados, conforme aumenta el tamaño de dichas occlusiones. Podría interesar, por tanto, mejorar los datos de entrenamiento para que el modelo identifique las armas que posean occlusiones con mayor facilidad.

En base a los resultados obtenidos en los experimentos, se concluye que YOLOv5, aun que presenta limitaciones y se centra en la detección en tiempo real, es un modelo adecuado para la detección de armas de fuego.

6.2 – Trabajos futuros

Aunque los resultados obtenidos en los experimentos son satisfactorios, son sin duda mejorables.

Como se ha mencionado previamente la variabilidad de los datos es grande y se necesita una colección de imágenes grande como para maximizar los resultados. El dataset construido cuenta con una media de 3300 instancias por clase. Sin embargo, Ultralytics recomienda 10000 objetos etiquetados por clase. Además, las clases seleccionadas abarcan tipos de armas muy distintas. Una forma interesante de completar el experimento sería aumentar tanto el número de imágenes e instancias como el número de clases. Dividiendo así las armas cortas en pistolas y revólveres y las armas largas, en fúsiles, metralletas, escopetas, etc. Además, el dataset debería completarse añadiendo imágenes que presenten armas con más occlusiones para garantizar que el modelo es capaz de detectarlas correctamente.

Otro posible enfoque es el de la detección en tiempo real. Dada la naturaleza de YOLOv5 y del problema en sí. Sería útil determinar cuál es el rendimiento del modelo en videos a la hora de detectar armas de fuego y clasificarlas. Esto ayudaría a decidir si es un modelo adecuado e implementable en sistemas de seguridad en tiempo real con cámaras de vigilancia o escáneres, o, si, por el contrario, conviene buscar otro tipo de solución.

7. Referencias

- [1] Grega, M., Matiolanski, A., Guzik, P., & Leszczuk, M. (2016). *Automated Detection of Firearms and Knives in a CCTV Image*. Sensors 2016, 16, 47. Doi:10.3390/s16010047.
- [2] Olmos, R., Tabik, S., & Herrera, F. (2017). *Automatic Handgun Detection Alarm in Videos Using Deep Learning*. ELSEVIER, Vol.275 (Neurocomputing), páginas 66-72.
- [3] Castillo, A., Tabik, S., Pérez, F., Olmos, R., & Herrera, F. (2018). *Brightness Guided Preprocessing for Automatic Cold Steel Weapon Detection in Surveillance Videos with Deep Learning*. ELSEVIER, Vol.330 (Neurocomputing), páginas 151-161.
- [4] Warsi, A., Abdullah, M., Husen, M. N., & Yahya, M. (2020). *Automatic Handgun and Knife Detection Algorithms: A Review*. IEEE.
- [5] Salido, J., Lomas, V., Ruiz-Santaquiteria, J., & Deniz, O. (2021). *Automatic Handgun Detection with Deep Learning in Video Surveillance Images*. Appl. Sci. 2021, 11, 6085. Doi: <https://doi.org/10.3390/app11136085>.
- [6] Ruiz-Santaquiteria, J., Velasco-Mata, A., Vallez, N., Bueno, G., Alvarez-Garcia, J. A., & Deniz, O. (2021). *Handgun Detection Using Combined Human Pose and Weapon Appearance*.
- [7] Ruiz-Santaquiteria, J., Velasco-Mata, A., Vallez, N., Deniz, O., & Bueno, G. (2022). *Improving Handgun Detection Through a Combination of Visual Features and Body Pose-Based Data*. ELSEVIER, Vol.136 (Pattern Recognition).
- [8] Moran, M., Conci, A., & Sánchez, Á. *Automatic Detection of Knives in Complex Scenes*.
- [9] Khan, N. S., Ogura, K., Cosatto, E., & Ariyoshi, M. (2023). *Real-time Concealed Weapon Detection on 3D Radar Images for Walk-through Screening System*.
- [10] UNODC Research. (2020). *Estudio Mundial sobre el Tráfico de Armas de Fuego: Introducción, Resumen Ejecutivo, Conclusiones y Consecuencias en Materia de Políticas y Resumen por Regiones*.
- [11] Vadapalli, P. *Ultimate Guide to Object Detection Using Deep Learning*. [En línea] Disponible en: <https://www.upgrad.com/blog/ultimate-guide-to-object-detection-using-deep-learning/>. [Último acceso: 25-03-2023]
- [12] Cochard,D. *mAP : Evaluation metric for object detection models*. [En línea] Disponible en: <https://medium.com/axinc-ai/map-evaluation-metric-of-object-detection-model-dd20e2dc2472>. [Último acceso: 11-04-2023]
- [13] Gur Arie,L. *The practical guide for Object Detection with YOLOv5 algorithm*. [En línea] Disponible en: <https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843>. [Último acceso: 11-04-2023]

- [14] Ultralytics. *Tips for Best Training Results*. [En línea] Disponible en: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/. [Último acceso: 12-10-2022]
- [15] Gandhi,R. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. [En línea] Disponible en: <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>. [Último acceso: 10-04-2023]
- [16] Datascientest, *Convolutional Neural Network : definición y funcionamiento*. [En línea] Disponible en: <https://datascientest.com/es/convolutional-neural-network-es>. [Último acceso: 13-04-2023]
- [17] Joecher,G. GitHub. [En línea] Disponible en: <https://github.com/ultralytics/yolov5>. [Último acceso: 12-10-2022]
- [18] Roboflow, Roboflow. [En línea] Disponible en: <https://roboflow.com/>. [Último acceso: 23-01-2023]
- [19] W. contributors. Wikipedia, Wikipedia, The Free Encyclopedia. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1089762876. [Último acceso: 20-03-2023]
- [20] R. Roelofs, V. Shankar, R. Benjamin, J. Miller, S. Fridovich-Keil, M. Hardt y L. Schmidt (2019) *A Meta-Analysis of Overfitting in Machine Learning, Advances in Neural Information Processing Systems*. Vol.32
- [21] Neelam,S. *Introduction to Object Detection with RCNN Family Models*. [En línea] Disponible en: <https://medium.com/analytics-vidhya/introduction-to-object-detection-with-rcnn-family-models-310558ce2033>. [Último acceso: 14-04-2023]
- [22] Pytorch. Pytorch Documentation. [En línea] Disponible en: <https://pytorch.org/docs/stable/index.html>. [Último acceso: 25-01-2022]
- [23] Rosebrock,A. *Intersection over Union (IoU) for object detection*. [En línea] Disponible en: <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>. [Último acceso: 15-03-2023]
- [24] Bajaj,A. *Performance Metrics in Machine Learning (Complete Guide)*.[En línea] Disponible en: <https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide>. [Último acceso: 15-03-2023]
- [25] DevelopmentSEED. *Introduction to machine learning, neural networks and deep learning*. [En línea] Disponible en: https://devseed.com/servir-amazonia-ml/docs/Lesson1a_Intro_ML_NN_DL.html [Último acceso: 01-06-2023]
- [26] Neelam, S .*YOLO for Object Detection, Architecture Explained!* [En línea] Disponible en: <https://medium.com/analytics-vidhya/understanding-yolo-and-implementing-yolov3-for-object-detection-5f1f748cc63a>. [Último acceso: 01-06-2023]

- [27] Mandour, A. *Evaluation metrics for object detection and segmentation*. [En línea] Disponible en: <https://iq.opengenus.org/evaluation-metrics-for-object-detection-and-segmentation/>. [Último acceso: 01-06-2023]
- [28] Imane, C. *YOLO v5 model architecture (Explained)*. [En línea] Disponible en: <https://iq.opengenus.org/yolov5/>. [Último acceso: 01-06-2023]
- [29] Computer Science Wiki. *Data representation*. [En línea] Disponible en: https://computersciencewiki.org/index.php/Data_representation. [Último acceso: 08-01-2023]
- [30] Na8. *¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador*. [En línea] Disponible en: <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>. [Último acceso: 16-01-2023]

8. Anexos

8.1- Implementación *Intersection Over Union* para YOLOv5

En este apartado se proporciona la implementación de un posible código para calcular IoU dadas las coordenadas de dos *Bounding Boxes* en formato x,y,w,h, donde (x,y) representa el centro de la *Bounding Box* y, w y h representan el ancho y la altura respectivamente. YOLOv5 realiza automáticamente estos cálculos.

```
#formato: x,y,w,h ---> formato: x1,y1,x2,y2
def rectangulo(box):
    box_x1 = box[0] - box[2]/2
    box_y1 = box[1] - box[3]/2
    box_x2 = box[0] + box[2]/2
    box_y2 = box[1] + box[3]/2
    return [box_x1,box_y1,box_x2,box_y2]

def intersection_over_union(target_bbox,predicted_bbox):

    #formato: x,y,w,h

    #Calcular esquinas bboxes
    tg = rectangulo(target_bbox)
    pred = rectangulo(predicted_bbox)

    #Calcular esquinas interseccion
    x1 = max(pred[0], tg[0])
    y1 = max(pred[1], tg[1])
    x2 = min(pred[2], tg[2])
    y2 = min(pred[3], tg[3])

    #area = w * h
    intersection = max(0,(x2 - x1)) * max(0,(y2 - y1))

    box1_area = abs((pred[2] - pred[0]) * (pred[3] - pred[1]))
    box2_area = abs((tg[2] - tg[0]) * (tg[3] - tg[1]))

    return intersection / (box1_area + box2_area - intersection + 1e-16)
```

8.2- YOLOv5 en local

En este trabajo se ha utilizado Google Colab, pero si se dispone de una gráfica dedicada NVIDIA con núcleos CUDA, es recomendable entrenar en local. Para ello pueden seguirse los siguientes pasos:

- Comprobar que se dispone de CUDA. En caso de no tenerlo instalado puede descargarse en: <https://developer.nvidia.com/cuda-downloads>
- Para poder clonar el repositorio de YOLOv5 será necesario instalar Git. Puede descargarse en: <https://git-scm.com/>
- Descargar el gestor de paquetes de Python, pip. Más información en:
<https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/>
- Usar el siguiente comando para instalar la versión adecuada de Pytorch con CUDA:
pip3 install torch torchvision torchaudio --extra-index-url https://download.pytorch.org/whl/cu113
- Una vez realizados los pasos anteriores se puede clonar el repositorio de YOLOv5 e instalar los requisitos de este, así como entrenar los modelos de forma similar a como se ha explicado en este proyecto para Google Colab.

8.3- Ejecución del proyecto y acceso a los datos empleados

Para facilitar el acceso al código y los datos empleados en este proyecto se ha creado un repositorio al cuál se puede acceder a través de:

<https://github.com/patriciacs99/WeaponDetectionYOLOv5>

Se proporciona un cuaderno de Google Colab con el código necesario para entrenar, aplicar y testear los modelos con los que se ha trabajado en este proyecto.

Así mismo, los dataset mencionados, los modelos entrenados y los resultados obtenidos están también disponibles en:

https://drive.google.com/drive/folders/15O3lpCT-JYyuhEc5WftPzr0vCELjLPSS?usp=drive_link

Se recomienda leer el fichero README.md para mayor comprensión del usuario.

8.4- Imágenes detecciones armas con YOLOv5

A continuación, se exponen algunos ejemplos de los resultados obtenidos en la detección de armas con el modelo YOLOv5s.

larga 0.96



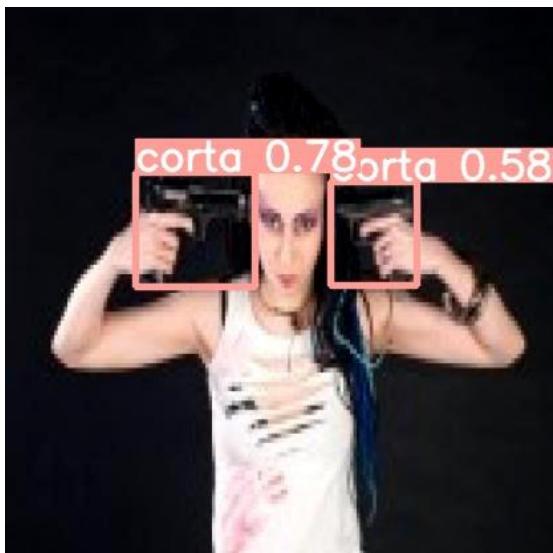
 CENTURY
ARMS
www.centuryarms.com

larga 0.86



alamy - PAC8RM

corta 0.78 **corta 0.58**



corta 0.94



corta 0.64



larga 0.90



