UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

ESTUDIO DE LA CALIDAD EN LAS DETECCIONES DE LA RED YOLOv5 CON TRANSFORMACIÓN DE IMÁGENES

STUDY OF THE QUALITY OF DETECTIONS IN THE YOLOv5 NETWORK USING DATA AUGMENTATION TECHNIQUES

Realizado por:

Raúl Peinado García

Tutorizado por:

Ezequiel López Rubio

Cotutorizado por:

José David Fernández Rodríguez

Departamento:

Lenguajes y Ciencias de la Computación.

Universidad de Málaga

Málaga, diciembre, 2022

Resumen

Este trabajo de fin de grado tiene como objetivo el estudio del rendimiento del modelo de detección de objetos en imágenes y videos, YOLOv5, al utilizar imágenes con transformaciones. Para ello se empleará la métrica mAP (*mean Average Precision)*.

Estas transformaciones serán modificaciones como: un cambio en el color, brillo, contraste o saturación de la imagen, incluyendo rotaciones de la imagen. Además de las anteriores, se pueden combinar entre ellas para generar nuevas transformaciones.

Se realizará un estudio para obtener las transformaciones que han producido una mayor mejora en el mAP respecto del mAP obtenido del conjunto de imágenes sin transformar.

Palabras clave: Aprendizaje automático, Detección de objetos, Transformaciones, Imágenes, YOLOv5.

Abstract

The aim of this end-of-degree final project is to study the efficiency of the object detection network in images and videos, YOLOv5, when using data augmentation techniques. For this purpose, the mAP (mean Average Precision) meter will be use.

Transformations have been used to apply data augmentation. These transformations are changes in color, brightness, contrast, saturation, or rotations of the image. Moreover, they can be combined with each other to generate new transformations.

A study will be conducted to get which of the applied transformations has improved the value of the mAP meter the most in respect of the value of the mAP meter of the original images set.

Keywords: Machine Learning, Object Detection, Transformations, Images, YOLOv5.

Índice de Contenido

Índice de figuras

Índice de tablas

Capítulo 1.

Introducción a la memoria

1.0 Introducción

La detección de objetos es una aplicación del *machine learning* con utilizad en multitud de campos que van desde la medicina, pasando por la ciencia, hasta la industria. En concreto, con la aparición y perfeccionamiento de nuevas técnicas de visión por computador, se han podido aplicar al mundo actual para realizar tareas de forma automática. Actualmente la complejidad de esas tareas es cada vez mayor, por lo que, obtener modelos con una buena precisión es un requisito fundamental para su aplicación. Tareas como la seguridad en el hogar, automatización de robots, detección de enfermedades, conducción autónoma… han podido ser automatizadas para mejorar nuestra calidad de vida [1].

Estas técnicas consisten en extraer automáticamente información de imágenes o vídeos para ser tratadas por un ordenador. Diariamente aparecen nuevas aplicaciones, una de las más recientes es *Orca IA*, la cual, evita accidentes en embarcaciones utilizando visión por computador o *Conservation Metrics* que se encarga del estudio y seguimiento de especies en peligro de extinción utilizando drones.

Como podemos ver, la detección de objetos en imágenes tiene gran importancia en la actualidad y debido a esto, la mejora en la precisión de los modelos es de gran interés.

1.1 Objetivo

El principal objetivo del proyecto es hacer un estudio de la eficiencia del modelo de detección en imágenes, *YOLO* (*You Only Look Once)* en su versión 5, cuando se comparan los resultados de la métrica mAP obtenidos del conjunto

imágenes originales (sin transformaciones) y del conjunto de imágenes transformado. Estas transformaciones se pueden aplicar de forma independiente o pueden combinarse.

El modelo de detección, YOLOv5, al proporcionarle una imagen como entrada, devuelve un conjunto de cuadros delimitadores o *bounding boxes*. Este conjunto de cuadros delimitadores se corresponde con cada una de las detecciones que se han encontrado en la imagen y serán usados para el cálculo de la métrica mAP.

Al introducir el conjunto de imágenes transformado, se pueden obtener detecciones redundantes sobre una misma imagen. Se desecharán los cuadros que no aportan información hasta obtener el que mejor se adapta a cada objeto detectado. Para realizar este filtro, se utiliza el algoritmo *Non-Maximum Suppression* (NMS) apoyándose en el *Intersection Over Union* (IoU). Tras obtener los cuadros delimitadores simplificados para cada detección, se puede calcular la métrica *mean Average Precision* (*mAP*).

Finalmente, se comparará el valor obtenido de la métrica mAP del conjunto de imágenes originales y del mAP del conjunto de imágenes transformado. Tras obtener los resultados se realizará el estudio para determinar qué transformaciones tienen un mayor impacto en la métrica.

1.2 Ámbito de aplicación

La detección de objetos en imágenes y vídeos se aplica a campos como el de la video vigilancia para detectar personas o cualquier anomalía que se produzca, también tiene utilidad para el seguimiento de objetos, conducción autónoma y análisis de imágenes de tráfico, entre otras. Estas técnicas han empezado a cobrar gran importancia en estos últimos años debido a la llegada del *deep learning* y a la aparición de algunos modelos de detección robustos que obtienen muy buenos resultados. Además de las nombradas anteriormente, se puede destacar la aplicación en muchas de las ramas de la ciencia, desde Astronomía y Medicina hasta la Robótica o la Biología [2]. A continuación, se enumerarán algunas aplicaciones:

- Robótica móvil y vehículos autónomos. Utilizando gran cantidad de sensores para identificar objetos, localizarse en un mapa, esquivar obstáculos, conducción autónoma, entre otras.
- Fabricación y producción. Se utiliza para identificación de piezas, control de calidad y detección de anomalías. Esto agiliza el proceso de producción ya que se pueden analizar más imágenes y en un menor tiempo y de forma automática.
- Análisis de imágenes médicas. Se analizan imágenes tomadas con ultrasonido, tomografía, rayos-X, resonancia magnética, endoscopia, etc. Estas imágenes pueden ser analizadas a gran escala acelerando el proceso de detección de enfermedades. También puede aplicarse en la detección precoz de enfermedades.
- Astronomía. Se analizan las imágenes tomadas con telescopios ayudando a la localización e identificación de objetos en el espacio. La primera imagen que tenemos de un agujero negro pudo ser realizada gracias a la aplicación y combinación de diferentes técnicas de visión por computador.
- Interpretación de imágenes microscópicas en campos de la ciencia como la química, la física o la biología [2].

1.3 Estructuración del Documento

En este apartado se presentará un resumen por capítulos del contenido de la memoria.

Capítulo 1. Introducción. En este capítulo se introduce a la memoria poniendo en contexto al lector y presentando los objetivos y el ámbito de aplicación del proyecto.

Capítulo 2. Estudio del Arte. En este apartado se proporcionarán los aspectos teóricos necesarios para la total compresión de la memoria.

Capítulo 3. Metodología y desarrollo. En este capítulo se hablará tanto del modelo de detección empleado, como de los algoritmos necesarios para la ejecución.

Capítulo 4. Implementación y ejecución. Se exponen las funciones, las transformaciones implementadas y las tecnologías utilizadas. Se incluyen

ejecuciones de ejemplo para obtener una visión más profunda de la aplicación de las transformaciones y cómo afectan estas en las detecciones.

Capítulo 5. Análisis de los resultados. Se analizan los resultados obtenidos para cada uno de los *datasets* empleados y para cada valor del *threshold* del *Intersection Over Union*.

Capítulo 6. Conclusiones y líneas futuras. Se extraen las conclusiones del apartado anterior, se comenta si se han alcanzado los objetivos propuestos al inicio del proyecto y para finalizar se realizan propuestas para continuar la línea de estudio seguida.

1.4 Glosario de Términos

Español	Inglés	Siglas en inglés
Solo miras una vez	*You Only Look Once*	*YOLO*
Objetos comunes en contexto	*Microsoft Common Objects in Context*	*MS COCO*
Media de las precisiones medias	*mean Average Precision*	*mAP*
Redes Neuronales Convolucionales	*Convolutional Neural Networks*	*CNN's*
Intersección sobre la unión	*Intersection Over Union*	*IoU*
Supresión no máxima	*Non-maximum Suppression*	*NMS*
Rectificador lineal	*Rectifier Linear Unit*	*ReLu*
Inteligencia artificial	*Artificial Intelligence*	*AI*
Aprendizaje automático	*Machine Learning*	*ML*

Aprendizaje profundo	*Deep Learning*	*DL*
Teoría de la resonancia adaptativa	*Adaptive Resonance Theory*	*ART*
Valor correcto esperado	*Ground truth*	*GT*
Red de clasificación por regiones	*Region Proposal Network*	*RPN*
Red piramidal de características	*Feature Pyramid Network*	*FPN*
Unidad de Procesamiento Gráfica	*Graphics processing unit*	*GPU*
Unidad Central de Procesamiento	*Central Processing Unit*	*CPU*
Detector de vistazo único	*Single Shot Detector*	*SSD*
Verdadero Positivo	*True Positive*	*TP*
Falso Positivo	*False Positive*	*FP*
Verdadero Negativo	*True Negative*	*TN*
Falso Negativo	*False Negative*	*FN*
Área Bajo la Curva	*Area Under Curve*	*AUC*
Librería para imágenes de python	*Python Imaging Library*	*PIL*
Conjunto de datos	*Dataset*	-
Salida de la red	*Output*	-
Entrada a la red	*Input*	-

Retropropagación	*Back-propagation*	-
Cuadro delimitador	*Bounding box*	-
Umbral	*Threshold*	-
Agrupaciones similares	*Clustering*	-
Redes totalmente conectadas	*Fully-connected networks*	-
Desvanecimiento del gradiente	*Vanishing gradient*	-
Sobreajuste	*Overfitting*	-

Tabla 1.1: Glosario de términos.

Capítulo 2.

Estado del arte

2.0 Introducción

En este capítulo se proporcionan los fundamentos teóricos necesarios para la comprensión de esta memoria, comenzando con una contextualización histórica y pasando por las técnicas de aprendizaje automático más empleadas. También se explicarán los principales conceptos del *deep learning* y la visión por computador y finalmente se nombran algunos de los modelos más importantes en detección de objetos y clasificación de imágenes.

2.1 Introducción a la Inteligencia Artificial

La neuro-computación nació en 1943 gracias al trabajo de Warren McCulloch y Walter Pitts, pero no fue hasta 1958 con Frank Rosenblatt cuando se introdujo la idea de perceptrón. Un perceptrón es una neurona artificial que tiene un conjunto de entradas, cada una de estas con un peso asociado y genera una única salida según la función de activación utilizada. Cada uno de los pesos se van modificando según una regla de aprendizaje cuando se obtiene la salida. Si la salida es correcta, los pesos aumentan su valor para dar más importancia a esa conexión neuronal y en caso contrario, los pesos disminuyen para intentar no producir de nuevo la misma salida errónea [3].

En 1975 en Fukushima, Japón se desarrolló el Cognitrón. Este fue un modelo de red neuronal autoorganizada muy primitiva para el reconocimiento de patrones visuales. Posteriormente en 1980 apareció Neocognitrón que fue una versión mejorada del anterior que superaba algunos inconvenientes que el Cognitrón no pudo, como son: deformaciones, cambios de tamaño y translaciones. Debido a los escasos recursos computacionales, la inteligencia artificial experimentó un estancamiento hasta los 90’s. A partir de 1993 aparece la primera red convolucional aplicada a imágenes, *LeNet*. Esta red fue una

versión mucho más simplificada de la conocida *AlexNet*, la cual trataremos en los puntos finales de este capítulo [4].

2.2 Machine Learning

El *machine learning* es una rama científica de la inteligencia artificial que su propósito consiste en que los algoritmos busquen patrones en datos. Los algoritmos de *machine learning* aprenden automáticamente y mejoran su rendimiento para realizar una tarea o realizar predicciones.

Existen varios tipos de aprendizaje automático, entre ellos se encuentran el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo. A modo de introducción, los algoritmos no supervisados extraen inferencias de conjuntos de datos. Los supervisados, aplican lo aprendido en el pasado a los nuevos datos. Los algoritmos de aprendizaje por refuerzo van aprendiendo iterativamente recibiendo recompensas según la dirección que tome en sus decisiones [5].

Para construir un modelo de *machine learning* tienen que realizarse las siguientes etapas. En la primera etapa, se tiene que explorar los datos disponibles para entender el problema que queremos resolver. En la segunda, definimos un criterio de evaluación en el cual elegimos cuál va a ser nuestra corrección de la medida de error. Tras definir el criterio de evaluación, pasamos a la tercera etapa dónde tendremos que segmentar y preparar los datos ya que nos podemos encontrar con dificultades. Los problemas más frecuentes que podemos encontrar pueden ser: datos incompletos, datos erróneos, datos desbalanceados, entre otros. Los datos se suelen dividir en tres subconjuntos, cada uno de ellos tiene una función concreta. El conjunto de entrenamiento siempre es el conjunto más grande y como su nombre indica, se utiliza para transformar los pesos del modelo. El segundo conjunto de datos que nos encontramos es el conjunto de validación. Este conjunto, no modifica los pesos simplemente se utiliza para monitorizar el entrenamiento y saber si la red está generalizando o está memorizando los datos. Se utiliza para modificar los parámetros o seleccionar el mejor de los modelos entrenados. Por último, encontramos el conjunto de prueba, el cual, nos muestra el error real cometido por el modelo. Una vez se han tratado y segmentado los datos, llegamos a la cuarta etapa. Esta es una de las más críticas del proceso, dónde tenemos que

seleccionar el algoritmo que mejor se adapte al tipo de problema que vamos a resolver. Por último, una vez hemos realizado todo lo anterior, llegamos a la etapa en la cual se entrena el modelo y se va modificando para adaptarse a los datos en función del criterio de evaluación elegido [6].

2.3 Aprendizaje supervisado

El aprendizaje supervisado es el tipo de aprendizaje automático que utiliza un conjunto de datos etiquetados y su objetivo es predecir tales etiquetas. Este método de aprendizaje requiere la utilización de menos datos en el proceso de entrenamiento, pero tales datos tienen que estar bien etiquetados. Si el conjunto de datos utilizado no es lo suficientemente preciso, el modelo puede estar sesgado y no se obtendrán resultados adecuados al enfrentarse a nuevos datos.

Existen dos tipos de aprendizaje supervisado, la clasificación y la regresión. En clasificación, los algoritmos son entrenados para clasificar los datos en variables discretas. Hay distintos tipos de clasificación, entre los más importantes se encuentran: la clasificación binaria, la clasificación multiclase, la clasificación de etiquetas múltiples y la clasificación con datos desbalanceados.

Respecto a la regresión, se pretenden establecer relaciones entre un número de características y una variable objetivo-continua. El ejemplo más común es el de la regresión lineal que se utiliza para establecer una relación linear entre los datos de entrada y salida. Además de la regresión lineal, existen otros tipos de regresión como pueden ser la regresión logística o la regresión polinomial [7].

2.4 Aprendizaje no supervisado

El aprendizaje no supervisado es el tipo de aprendizaje automático en el cual se aprenden los datos. A diferencia del aprendizaje supervisado, los datos que se les proporcionan a los algoritmos no supervisados, no se encuentran etiquetados. El objetivo de estos algoritmos es extraer información de los datos, encontrar similitudes, estructuras, patrones ocultos y realizar agrupaciones en estos. Este tipo de aprendizaje es útil para explorar datos desconocidos ya que pueden descubrir patrones inexplorados o examinar grandes conjuntos de datos que de manera manual no sería posible. Estos descubrimientos pueden ayudar

al estudio de los datos y sobre todo para crear los vectores de características de los datos [8], [9].

Dentro del aprendizaje no supervisado, encontramos las redes autoorganizadas, la red de *Kohonen*. También encontramos redes competitivas no supervisadas como las redes ART (*Adaptive Resonance Theory*), algoritmos de *clustering*, análisis de componentes principales o descomposición en valores singulares.

2.5 Aprendizaje por refuerzo

En este tipo de aprendizaje automático tampoco disponemos de datos etiquetados y los algoritmos tienen que aprender por sí mismos. A diferencia del aprendizaje no supervisado, no se intentan hacer agrupaciones en los datos. El proceso de aprendizaje constará de un esquema de recompensas y penalizaciones en un entorno cambiante y variable en el tiempo. No se intenta minimizar la función de coste para reducir el error, se intenta maximizar la recompensa. El sistema consta de dos componentes, el agente y el ambiente. El agente es el que toma las decisiones y el ambiente será el entorno dónde se encuentra el agente, el cual, dispondrá de limitaciones y reglas que irán cambiando en el tiempo. Entre estas dos componentes se establecen distintas relaciones, estas serán las acciones que pueda realizar el agente, el estado del ambiente y los elementos que lo componen y, por último, las recompensas o penalizaciones según la decisión tomada [10].

Entre los casos de uso del aprendizaje por refuerzo, encontramos brazos mecanizados, robots, maquinaria industrial, creación de webs personalizadas e incluso en el mundo de los videojuegos. Uno de los algoritmos más utilizados es el *Q-Learning*.

2.6 Redes neuronales artificiales

Una red neuronal es un sistema de procesamiento de información basada en el funcionamiento de las neuronas biológicas. El procesamiento de la información ocurre en los elementos más simples, las neuronas. Estas se organizan en distintos niveles denominados capas. La primera capa es la capa de entrada (*input*) y sirve como entrada de la información a la red neuronal, las

siguientes capas se denominan capas ocultas y realizan el procesamiento de la información. Por último, tenemos la capa de salida (*output*), esta se encarga de tomar las decisiones en base al procesamiento realizado en las capas anteriores. Cada neurona que se encuentra en una capa previa tiene una conexión con alguna neurona de la capa siguiente para transmitir la información. A estas conexiones se le asocia un peso sináptico que será actualizado en el proceso de aprendizaje. Al final del proceso de aprendizaje el valor que almacenan los pesos será el conocimiento adquirido por la red.

Cada una de las capas de la red tiene asocia una función de transferencia dependiendo del problema que queramos resolver. Esta función es la encargada de evaluar la combinación lineal de los pesos con las entradas de la neurona y transmitir esta información en forma de salida. Se pueden utilizar funciones lineales como la función identidad, pero las funciones utilizadas para problemas más complejos como la clasificación y detección serán funciones no lineales. Las más típicamente conocidas son la función escalón, sigmoidal, tangente hiperbólica, *ReLu*, *Softmax*… [11].

Una de las redes neuronales artificiales más frecuentemente empleadas es perceptrón multicapa ya que proporciona salidas no lineales. Esta red consiste en la concatenación de capas formadas por perceptrones simples. Combinando perceptrones, se podían resolver algunos problemas no lineales, pero hasta el momento no existía un mecanismo automático para ajustar los pesos de las capas ocultas de la red de forma general. De esta manera, en 1986, surgió la Regla Delta Generalizada o *back-propagation* que resolvía este problema y además convertía al perceptrón multicapa en un aproximador universal [12], [13].

En este algoritmo se determinan los pesos de tal manera que se minimice el error total. Se utiliza el método de descenso del gradiente comenzando por la capa de salida y propagando el error hacia capas previas, hasta llegar a la capa de entrada [14].

2.7 Visión por computador

Gracias a los avances en la neurociencia y en diagnóstico por imágenes, sabemos que hay redes especializadas en nuestro cerebro que se encargan de detectar diferentes aspectos de la visión. Por ejemplo, unas estructuras detectan la inclinación de las líneas, los bordes, la expresión emocional de los rostros, etc.

Estas detecciones se intentan imitar en la visión por computador al encargar a ciertas unidades de proceso que sean capaces de especializarse en reconocer una determinada característica de los datos [15].

La visión por computador es una de las aplicaciones de las técnicas del *deep learning* para extraer información a partir del análisis de imágenes y videos. Gracias a la aplicación de estas técnicas y métodos podemos procesar, analizar y extraer información numérica que los algoritmos utilizarán para diversos fines. Estas técnicas pueden ser útiles para detectar objetos y clasificar imágenes, reconstruir escenas, estimar movimientos, restaurar imágenes, seguimiento de vídeo, conducción autónoma, entre otras. Dentro de este campo podemos destacar las redes de clasificación de imágenes y las de detección de objetos [16], [17].

2.8 Redes neuronales convolucionales

Para extraer información de imágenes utilizaremos las redes neuronales convolucionales o *Convolutional Neural Networks* (CNN's). Los píxeles se procesan y se genera una matriz que será la representación numérica de cada uno de esos píxeles. Estas redes extraen la información gracias a las llamadas capas convolucionales.

Las neuronas de la primera capa solo se conectarán con una porción de neuronas de la capa siguiente ya que hay grupos de neuronas que se especializan en realizar una tarea concreta. Las primeras capas son capaces de detectar líneas horizontales, verticales y curvas, características básicas de la imagen, mientras que las capas más profundas de la red pueden llegar a reconocer formas complejas como siluetas [18]. Para poder almacenar esa información se aplican las convoluciones, que son operaciones lineales que implica la multiplicación de la matriz de datos de entrada con una matriz de ponderaciones, denominada filtro o *kernel* [19].

La salida obtenida al aplicar este filtro y la función de activación se denomina mapa de características. La función de activación que se utiliza en las capas convolucionales es la función *ReLu* (*Rectifier Linear Unit*) que aplica $f(x) = max(0, x)$, debido a que permite un aprendizaje más rápido y funciona muy bien cuando se quiere reconocer patrones en imágenes. Cuando obtiene

valores negativos el resultado será 0, coincidiendo con el gradiente. Al obtener valores positivos, el valor obtenido será el mismo y el gradiente será 1. Cuando la derivada de la función es 0 se puede generar la muerte de neuronas, esto quiere decir que hay neuronas que nunca intervendrán en la salida de la red. Para evitar esto se utiliza una versión modificada de la *ReLu*, la *Leaky ReLu* cuya función aplicada es $f(x) = max(0.1 * x, x)$ para evitar que todos los valores negativos, tanto grandes como pequeños, sean 0 [20].

Las capas convolucionales se combinan con capas de *max-pooling*. En estas capas se analiza el contenido de la imagen por regiones obteniendo la información más representativa de cada una de estas regiones. Se aplica un *kernel* a la entrada que calcula el máximo de cada región. Cuanto mayor tamaño tenga el *kernel*, menos regiones crearemos por lo que más reduciremos la información que pasa a la siguiente capa [21].

El proceso de convolución combinado con las capas de *max-pooling* se repite varias veces hasta obtener una salida de este conjunto de capas. Esta salida se pasa a una red neuronal tradicional. Esta red suele tener varias capas totalmente conectadas o *fully-connected* activadas con *ReLu* hasta llegar a la capa de salida. Para esta última capa, la función de activación utilizada suele ser la función *Softmax.* La capa de salida tendrá tantas neuronas como clases se quieran clasificar. En el caso del modelo YOLOv5, se utilizan 80 clases distintas por lo que la capa de salida de esta red tendrá 80 neuronas. Se utiliza el método de *back-propagation* como en el perceptrón multicapa para ajustar los pesos de los filtros aplicados en las convoluciones [22].

Normalmente, en las redes neuronales convolucionales se emplea el método *dropout* para las últimas capas. Este método desactiva neuronas de la capa de forma aleatoria. Estas neuronas no intervendrán en el cálculo del error ni en la retropropagación. Se utiliza para evitar el sobreajuste a los datos (*overfitting*) y ayuda a las neuronas a funcionar independientemente y no depender de las relaciones con neuronas próximas. El *dropout* solo se utiliza en la fase de entrenamiento y tiene diferentes valores que van desde 0 hasta 1 según el porcentaje de neuronas que queremos mantener activadas [23].

2.9 Redes de clasificación de imágenes

*AlexN*et es un modelo de *CNN*, cuyo predecesor fue la histórica *LeNet*, capaz de clasificar 1000 clases distintas y mejorando el error cometido en un 9,4% con respecto al mejor modelo que existía hasta el momento. Motivo por el cual ganó en 2012 el concurso *Large-Scale Visual Recognition Challenge* (ILSVRC12) que es el concurso más importante de modelos de reconocimiento y clasificación de imágenes. Fue propuesta en 2012 y su arquitectura de red contiene cinco capas convolucionales seguidas de tres capas *fully-connected*. Se aplica *ReLu* como función de activación en todas las capas menos la de salida ya que descubrieron que con esta función de activación se aceleraba la velocidad de entrenamiento en casi seis veces. En la capa de salida se usa la *Softmax*. Se utiliza *dropout* antes de la primera capa *fully-connected* y después de esta. Esta red no es muy eficiente y las capas de convolución consumen el 95% del cálculo necesario [24].

Dos años después de la aparición de *AlexNet*, *Google* introdujo *GoogLeNet*. Esta red pasa de tener 8 capas a tener 22 pero, a pesar de esto, era mucho más eficiente que su predecesora. También se hizo con la victoria en el *Large-Scale Visual Recognition Challenge* obteniendo una reducción del error cometido de 9.7% con respecto a *AlexNet*. Los investigadores de Google eran conscientes de la cantidad de recursos que consumían las capas convolucionales por lo que crearon una nueva estructura llamada módulo de incepción. Este módulo, se basa en el principio de dispersión ya que se intenta imitar a los sistemas nerviosos biológicos al conectar cada neurona con un número reducido de neuronas de la siguiente capa, no con todas ellas. Con esta mejora, se previene el *overfitting* y se ahorra recursos computacionales. Estos módulos reciben como *input* un conjunto de mapas creados por las capas anteriores y los pasa por distintas capas convolucionales de tamaños variables y combinándose finalmente en una misma capa denominada *filter concatenation*. La arquitectura de red consta de inicialmente de 2 capas convolucionales combinadas con 2 capas de max-pooling. A continuación, se encuentran 9 bloques de incepción separados por capas de *max-pooling*. Todas las capas anteriormente nombradas son activadas con la función *ReLu*. Finalmente hay una última capa de *pooling*, otra de *dropout* al 0.4 para evitar la muerte de neuronas y una última capa de salida activada con *softmax* [25].

Al igual que GoogLeNet, *VGGNet* apareció en 2014. Redujo el error con respecto a AlexNet en un 9.1%. Fue superado por el modelo de Google en un 0.6%. Esta red consta de 13 capas convolucionales combinadas con 5 capas de *max-pooling.* Tras estas capas hay 2 capas *fully-connected* y una última de salida activada por *softmax*. Hay dos versiones de esta red, la VGG-16 y la VGG-19 dependiendo del número de capas convolucionales que contengan [26].

Finalmente, en 2015, apareció *ResNet*. Esta es una red neuronal residual que ganó el *Large-Scale Visual Recognition Challenge* en 2015 tras reducir el error un 3.13% con respecto a GoogLeNet hasta llegar a un 3.57% en error global. Se introdujo un nuevo concepto, la red con bloques residuales, que se basa en el salto de capas para evitar un estancamiento en el aprendizaje al reducirse demasiado el gradiente (*vanishing gradient*). Utiliza una arquitectura de 152 capas con bloques residuales evitando así aumentar la red en profundidad y haciéndolo en amplitud [27].

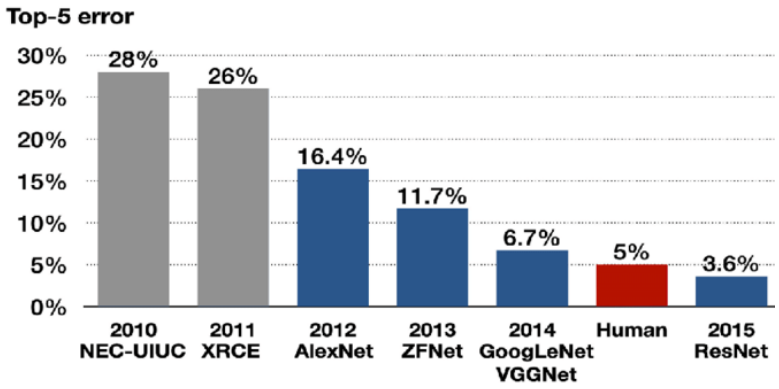

Figura 2.1: Algoritmos que han ganado el concurso ImageNet Large Scale Visual Recognition.

2.10 Redes de detección de objetos

Respecto a las principales redes de detección de objetos, podemos destacar en primer lugar la red R-CNN. Apareció en 2014 y se basa en el concepto de *Region Proposal Network* (RPN). Este concepto realiza una búsqueda selectiva para determinar las regiones de interés de la imagen y posteriormente estas regiones son evaluadas por una red preentrenada que realiza la clasificación [28].

En una primera etapa, se generan los cuadros delimitadores (*bounding boxes*) potenciales de la imagen. A continuación, se utiliza un clasificador sobre los cuadros delimitadores junto con un postprocesado para aumentar la precisión de estos. En esta segunda etapa también se eliminan los cuadros delimitadores duplicados utilizando el algoritmo *Non-maximum Suppression* e *Intersection Over Union* que veremos en el siguiente capítulo. Cada una de estas etapas consume mucho tiempo y este es el principal inconveniente de este modelo.

Posteriormente, llegó el algoritmo *Fast R-CNN* que mejoraba el tiempo de entrenamiento y prueba al reutilizar algunos recursos. Pero no fue hasta la llegada del algoritmo *Faster R-CNN* donde se experimentó un aumento sustancial en la velocidad. En él se integró el algoritmo de propuesta de región anteriormente descrito sobre la red convolucional. De esta manera, se mejoraba bastante el primer proceso de selección de regiones de interés que era el que consumía una mayor cantidad de recursos [29].

La segunda red de detección de objetos que podemos destacar es la *Single Shot Detector* (SSD). Esta red es que posee una red convolucional de tipo piramidal, FNP (*Feature Pyramid Network*). Respecto a la red piramidal de características, resuelve el problema de detectar objetos a diferentes escalas al crear una pirámide con los mapas de características. Esto le permite obtener la detección de objetos grandes y pequeños ya que utiliza múltiples capas de detección que le proporcionan múltiples mapas de características con escalas diferentes [30]. Los mapas con una mayor resolución son utilizados para detectar objetos con mayor escala, mientras que los de menor resolución se utilizan para objetos más pequeños. Puede usar la arquitectura de extracción de características de *VGG*, *GoogLeNet* o *AlexNet*. Como base normalmente se utiliza las capas convolucionales de *VGG-16* que sen ve aumentadas en 6 capas más para la detección de objetos. La ventaja de los mapas de características múltiples es que logran una mejora en la precisión. La gran velocidad en las detecciones de este modelo es favorable para la utilización en la detección de objetos en tiempo real [31].

El siguiente modelo destacable es *RetinaNet*. Utiliza una red piramidal de características al igual que la SSD. Se introduce una mejora en la pérdida focal, la cual, es una mejora sobre la pérdida de entropía cruzada. Esta mejora es posible ya que se centra en las predicciones de una clase incorrecta más que en

las correctas. Utiliza *ResNet* para obtener los mapas de características en distintas escalas, posteriormente, se va recorriendo la pirámide de los mapas de forma inversa, fusionando y tomando muestras de los mapas de características. A continuación, se encuentra la red de clasificación que predice la probabilidad de aparición del objeto y finalmente una red de regresión que calcula el desplazamiento entre los cuadros delimitadores y los esperados (*ground truth*) [32].

Por último, podemos destacar la familia de modelos YOLO. *YOLO* o *You Only Look Once*, es un algoritmo de detección de objetos que puede ser aplicado tanto a imágenes como a video en tiempo real debido a su rendimiento y su gran velocidad de procesamiento. Su rendimiento es tan bueno que podemos ejecutarlo en dispositivos móviles. YOLO es capaz de predecir simultáneamente los cuadros delimitadores asociados a cada detección y las probabilidades de cada uno de estos. Para hacer las predicciones, el sistema consume la imagen completa, esto hace que se limiten los errores a la hora de reconocer las clases de objetos que existe en la imagen. Aprende representaciones generalizables de objetos, lo que hace que se obtenga un menor error cuando introducimos datos de entrada no conocidos. Respecto a su funcionamiento, la imagen es divida en una rejilla dónde cada celda de esta es responsable de detectar un objeto [33]. A diferencia del modelo *R-CNN*, no necesita ningún algoritmo adicional para detectar las regiones de interés de la imagen. La red se basa en la arquitectura de *GoogLeNe*t, se utilizan 24 capas convolucionales intercaladas con capas de *max-pooling*, activadas con *LeakyReLu* y por último dos capas *fully-connected* cuya función de activación es la *ReLu*. Para obtener la confianza de una detección, necesitamos establecer algunos parámetros. En primer lugar, necesitamos establecer la Intersección sobre la unión (*Intersection Over Union, IoU*) del cuadro delimitador y el valor esperado (*ground truth*) que tenemos para esa imagen. Junto con el IoU, para calcular la confianza de la detección, se combina la probabilidad condicionada a cada celda de la cuadrícula que contiene al objeto [34].

2.11 Técnicas de Data Augmentation

Estas técnicas son frecuentemente aplicadas en la visión por computador, sobre todo cuando se dispone de una base de datos limitada. El concepto de

data augmentation no es más que aplicar transformaciones a los datos originales. Tras aplicar estas transformaciones obtendremos nuevos datos, muy parecidos a los originales pero que ya se encuentran clasificados (llevarán la etiqueta del dato original antes de transformarlo).

Otra de las aplicaciones de estas técnicas es cuando, al ejecutar un número de épocas elevado en el proceso de entrenamiento, la red puede llegar a memorizar ciertos datos. Para evitar este inconveniente, se pueden aplicar transformaciones de forma aleatoria cada vez que se introduzca un mismo dato a la red.

Al centrarse el tema principal de este proyecto en imágenes, se van a nombrar algunas de las técnicas más utilizadas para transformar las imágenes. Se puede realizar un volteo de la imagen respecto a los ejes horizontal y vertical, rotaciones, introducción de ruido y defectos, redimensión de la imagen, aplicar deformaciones en la perspectiva, ajustar parámetros como el brillo, la saturación, el color o el contraste. Además de todas estas transformaciones, se pueden combinar entre ellas. De esta manera, dispondremos de más información para entrenar nuestro modelo y se obtendrán mejores resultados cuando se enfrente a imágenes con malas condiciones de luminosidad o tomadas desde ángulos que no se encontraban en la base de datos inicialmente. En resumen, estas técnicas nos ayudan cuando tenemos una base de datos limitada, también evitan que se produzca un sobreajuste en el modelo (*overfitting*) y generalizan mejor la información adquirida [35], [36].

Capítulo 3.

Metodología y desarrollo

3.0 Introducción

La metodología que se ha seguido para el desarrollo de este proyecto ha sido la siguiente:

- Elección de una base de datos. El modelo YOLOv5 utiliza el conjunto de imágenes *COCO* para su entrenamiento [37]. Se han utilizado varios subconjuntos de imágenes del conjunto de validación del *dataset* COCO.
- Desarrollo de las funciones que transforman las imágenes. Las imágenes originales se transforman mediante funciones, dichas transformaciones han consistido en aumentar o disminuir el brillo, el contraste, la saturación y el color de la imagen y, por último, cambios en la orientación de la imagen con rotaciones. Además, estas transformaciones se pueden combinar entre ellas dando como resultado un *dataset* transformado.
- Ajuste de los parámetros en las transformaciones. Se tienen que ajustar los parámetros de intensidad de los filtros que aplicamos a las imágenes en las transformaciones. También hay que ajustar el número de grados que se pueden rotar las imágenes sin que se produzcan detecciones erróneas.
- Evaluación de las imágenes transformadas. Una vez obtenemos el *dataset* de imágenes transformadas, tenemos que evaluarlas en el modelo YOLOv5 y obtendremos una caja delimitadora por cada detección que se realice en cada una de las imágenes. Podemos obtener varias detecciones asociadas a un mismo objeto por lo que se simplificarán con el algoritmo Non-Maximum Suppression.
- Ajuste del *threshold* en el *Intersection over Union*. Este parámetro se utiliza en los modelos de detección para medir la superposición entre una caja obtenida que pertenece a una clase y su caja ideal, la que se ajusta

perfectamente a esa detección. Cuanto mayor área compartan estos dos cuadros delimitadores, mayor será el valor del *Intersection over Union*. En la red YOLOv5 por defecto el *threshold* es del 50%. En este trabajo, además de con el valor por defecto, se han hecho diferentes pruebas con un valor de 75% y 95%. Al ser más alto este valor, el área que han de compartir las dos cajas delimitadoras tiene que ser mayor.

- Utilización de la métrica mAP. Calculamos el mAP de las imágenes originales y de las transformadas. Para poder compararlos y extraer conclusiones.
- Extracción de resultados y evaluación de las mejores transformaciones. Evaluamos todos los resultados obtenidos utilizando la comparación de la métrica y configurando los ajustes de parámetros anteriormente descritos para clasificar con cuáles de las transformaciones se obtienen mejores resultados.
- Redacción del documento. En esta memoria se recoge la documentación de todo el proceso seguido para la realización del proyecto, así como se explican los aspectos teóricos necesarios para su total compresión.

3.1 YOLOv5

La primera versión de YOLO se presentó en 2015 y la última versión es la 5 que fue introducida en 2020. En las versiones sucesoras de la original como lo son la versión 3 y 4, se han introducido mejoras como la utilización de conexiones residuales y la utilización de una red convolucional piramidal (*Feature Pyramid Network)* [38]. La última versión, que es la utilizada en este proyecto, está implementada en el *framework* de *pytorch. Pytorch* es una librería basada en *python* enfocada a la realización de cálculos a través de tensores. Estos tensores se ejecutan en la GPU, con lo cual, los tiempos de entrenamiento y ejecución se reducen considerablemente con respecto a la ejecución en la CPU. Este *framework* se ha convertido en uno de los más utilizados para el desarrollo de aplicaciones de inteligencia artificial, junto con las librerías *Tensorflow* y *Keras* [39].

Para la ejecución de las pruebas se han utilizado los pesos preentrenados de YOLOv5 empleando el *dataset* COCO. En este caso se han empleado los

pesos por defecto que reciben el nombre de YOLOv5s. La elección de los pesos es importante ya que tienen un rendimiento distinto tanto en la métrica mAP como en la utilización de recursos. Con YOLOv5s se consigue unos valores aceptables de mAP al igual que no se tiene un consumo tan excesivo de tiempo y recursos en la ejecución del modelo [40].

3.2 Intersection Over Union

Ahora vamos a definir una de las métricas que se utilizan para comparar la bondad de los resultados obtenidos. La intersección sobre la unión (o *Intersection Over Union, IoU*) evalúa el porcentaje de superposición entre dos cuadros delimitadores (*bounding boxes*). Uno de los dos cuadros delimitadores que se comparan es el que obtenemos de las anotaciones del *dataset* para esa imagen (*ground truth*) y que se usa para poder realizar el entrenamiento y la validación del modelo. Por otro lado, tenemos el segundo cuadro delimitador que es el que hemos obtenido tras la predicción del modelo. Este valor estará más cercano a 1 cuanto mayor porcentaje compartan del área que delimitan ambos cuadros. Podemos ver gráficamente cómo se realiza este cálculo en la figura 3.1.

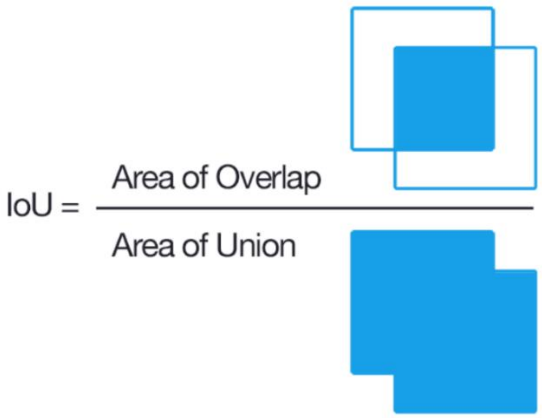

Figura 3.1: Cálculo del Intersection Over Union de forma gráfica.

Esta métrica es uno de los parámetros que podemos modificar para restringir el porcentaje de área que tienen que compartir. Por defecto en el entrenamiento de la red YOLOv5, el umbral o *threshold*, se fija por defecto en 50% pero si queremos ser más restrictivos con los nuevos resultados obtenidos, se puede utilizar 75% y 95% [41].

3.3 Non-maximum Suppression

Este algoritmo es típicamente utilizado en detección de objetos. A este algoritmo le llegan un conjunto de *bounding boxes* que delimitan cada uno de los objetos que se encuentren en la imagen, la confianza de cada una de esas detecciones y el umbral o *threshold* con el que tendrá que filtrar. La salida que proporciona será una única caja para cada una de las detecciones encontradas.

Figura 3.2: Cálculo del algoritmo Non-maximum Suppression de forma gráfica.

Inicialmente, se selecciona el cuadro con mayor confianza y se compara utilizando el *Intersection Over Union*, con todos los cuadros de esa misma clase que se hayan detectado. Si el IoU de todos los cuadros, al compararlo con el que hemos seleccionado, es mayor que el umbral, almacenamos ese valor en el conjunto de salida del algoritmo como el mejor para esa detección. Este algoritmo puede devolver varias detecciones para una misma clase ya que en una imagen se pueden encontrar multitud de objetos pertenecientes a una misma clase. En la figura 3.2 se puede ver gráficamente como selecciona la *bounding box* que mejor representa el objeto, descartando las demás [42], [43].

3.4 Selección del *dataset*

Uno de los principales problemas de la construcción de modelos de inteligencia artificial en general y de la visión por computador en concreto es que se necesitan bases de datos robustas para construir modelos realistas y que puedan realizar predicciones acertadas. En el modelo YOLOv5 se ha usado el conjunto de datos *MS COCO* (*Microsoft Common Objects in Context*) para su

entrenamiento. Este *dataset* es uno de los más utilizados para detección, segmentación, detección de puntos clave, entre otros. La primera versión fue lanzada en 2014 y es la que se ha utilizado para realizar las pruebas en este proyecto. Cuenta con tres conjuntos de datos, el de entrenamiento con 83.000 imágenes, el conjunto de validación con 41.000 imágenes y, por último, el conjunto de prueba con 41.000 imágenes. Contiene 90 clases de objetos diferentes, aunque para entrenar el modelo YOLOv5 únicamente se han utilizado 80 de esas clases. Varios ejemplos de clases de objetos que podemos encontrar en este *dataset* van desde personas, animales, vehículos hasta señales de tráfico, luces de tráfico, hidrantes de incendios, plantas, comida... [44].

3.5 Anotaciones del *dataset*

El *dataset* COCO contiene archivos con información asociada a cada imagen. Estos archivos se llaman anotaciones y en ellos, junto con alguna información extra, se encuentran los cuadros delimitadores dónde existe un objeto en la imagen y la clase de COCO a la que pertenecen. Los cuadros delimitadores son el valor en píxeles de dos de las esquinas que delimitan el rectángulo de la detección. Estas anotaciones nos sirven tanto para entrenar el modelo como para poder realizar comparaciones entre las detecciones obtenidas del modelo y las detecciones ideales [44], [45].

3.6 Funciones de transformación de imágenes

A continuación, se van a mostrar los diferentes ejemplos de cada uno de los 5 ajustes implementados y cómo actúan cada uno de estos en las imágenes.

Ajuste de rotación.

Figura 3.3: Imagen original.

Figura 3.4: Imagen con un ajuste de rotación.

Ajuste de color.

Figura 3.5: Imagen original.

Figura 3.6: Imagen con un ajuste de color.

Ajuste en el contraste.

Figura 3.7: Imagen original.

Figura 3.8: Imagen con un ajuste de contraste.

Ajuste en la saturación.

Figura 3.9: Imagen original

Figura 3.10: Imagen con un ajuste de saturación.

Ajuste en el brillo.

Figura 3.11: Imagen original.

Figura 3.12: Imagen con un ajuste de brillo.

En la tabla 3.1 se muestran los valores que pueden tomar los parámetros en cada uno de los ajustes. Para los ajustes de brillo, contraste, color y saturación se ha empleado una distribución uniforme continua. Se ha seleccionado este tipo de distribución debido a que el paquete de *Python* empleado para las funciones de transformación de las imágenes, necesita de este parámetro para controlar la intensidad del ajuste aplicado [46]. En los ajustes de brillo, contraste y color se ha suprimido el valor 1 ya que con ese valor no se producía ningún cambio en la imagen original.

En los ajustes de rotación, se ha empleado una distribución uniforme discreta con un rango de valores posibles de 10 grados (desde 355 hasta 5 grados). No se han seleccionado un mayor rango de valores posibles debido a que si se rota demasiado la imagen, se producen detecciones erróneas. Al igual que en la generación del parámetro anterior, también se ha suprimido un valor, el de 0 grados de rotación.

Transformación	Rango de valores de los parámetros
Brillo	Distribución uniforme continua entre [0.5, 1.75]
Contraste	Distribución uniforme continua entre [0.5, 2]
Saturación	Distribución uniforme continua entre [0.5, 5]
Color	Distribución uniforme continua entre [0, 2.5]
Rotaciones	Distribución uniforme discreta sobre: [355º, 356º, 357º, 358º, 359º, 1º, 2º, 3º, 4º, 5º]

Tabla 3.1: Valores de los parámetros para las funciones que transforman las imágenes.

Cada uno de estos ajustes se puede combinar para generar nuevas transformaciones. De este modo, con la combinación de estos ajustes, se han aplicado 3 tipos distintos de transformaciones en los conjuntos de imágenes.

Transformación simple con un único ajuste. En este tipo de transformación se aplica un único ajuste, de los anteriormente definidos, a cada imagen del *dataset* utilizado.

Transformación simple con varios ajustes. En este tipo de transformación se aplican 2 o 3 ajustes a cada imagen del *dataset*. Cada uno de estos ajustes se aplica de forma independiente generando una nueva imagen transformada por cada ajuste aplicado. Por ejemplo, si aplicamos dos ajustes, obtendremos dos imágenes nuevas. Estas detecciones finalmente se combinan con el algoritmo *Non-maximum Suppression*, el cual, devuelve la *bounding box* que mejor se adapta a cada objeto detectado.

Transformación compuesta. En este tipo de transformación se aplican 2 de los ajustes anteriormente descritos a cada imagen del *dataset*. Las detecciones se realizan sobre la imagen doblemente transformada y cada uno de estos ajustes se aplican en el orden en el que se encuentran descritos.

3.7 mAP y AP

El AP (*Average Precision*) y mAP (*mean Average Precision)* son las métricas más utilizadas en detección de objetos. Gran cantidad de modelos como Faster R-CNN, SSD, YOLO… utilizan estas métricas para evaluar la calidad de las predicciones.

Para evaluar si una detección es correcta o no, se comparan los cuadros delimitadores de las detecciones del modelo con los cuadros de la *ground truth*. Esto se hace con el algoritmo de *Intersection Over Union* y con respecto al *threshold* que se haya fijado. De esta manera, si el IoU supera el *threshold* será considerado como verdadero positivo (TP) y en caso contrario como falso positivo (FP). También se obtendrá un falso positivo cuando se haga una detección de un objeto inexistente en la imagen. Un falso negativo (FN) se obtendrá cuando no se haya detectado el objeto que sí que se encuentra en la imagen analizada. El concepto de verdadero negativo (TN) no se aplicará en el contexto de la detección de objetos.

La precisión es el número de verdaderos positivos (TP) dividido por el número de falsos positivos (FP) y verdaderos positivos (TP). Con precisión nos referimos a la capacidad del modelo de identificar únicamente objetos relevantes y será el porcentaje de predicciones que se han realizado correctamente respecto a todas las detecciones realizadas.

$$P = \frac{TP}{TP + FP} = \frac{TP}{all\ detections}$$

Por otro lado, también se calcula el *Recall* o exhaustividad. Esta métrica consiste en dividir los verdaderos positivos (TP) entre los verdaderos positivos (TP) junto con los falsos negativos (FN). Con *recall* nos referimos al porcentaje de predicciones correctas respecto a todas las detecciones esperadas para esa imagen.

$$R = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truth}$$

Por último, nos encontramos con la curva de $Precision \cdot Recall$ en la cual se muestra la variación entre la precisión y el *recall* para diferentes valores de la confianza asociada a las *bounding boxes* que son generadas por un modelo de

detección de objetos. Un buen detector debería encontrar todos los objetos que están presentes en la imagen (FN = 0 = *recall* alto) y además identificar solo los objetos relevantes (FP = 0 = precisión alta) con lo cual, un modelo de detección de objetos se considerará bueno si la precisión se mantiene alta mientras el *recall* se incrementa. Es decir, si al variar el valor del *threshold*, la precisión y el *recall* se mantienen altos.

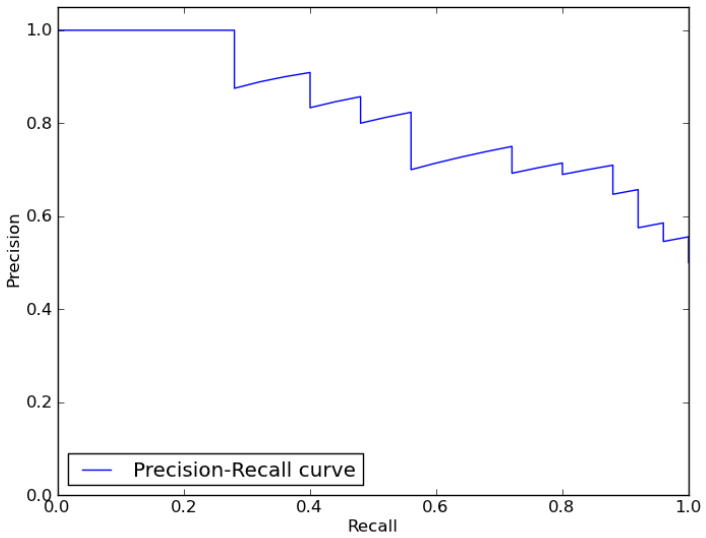

Figura 3.13: Ejemplo de la curva de Precision x Recall.

Para poder medir la relación entre la precisión y el *recall* calculamos el área bajo la curva (AUC). Una mayor área bajo la curva indica que la precisión y el *recall* han tenido valores altos. Como podemos ver en la figura 3.13, la curva que se describe es en zigzag impidiendo calcular el área de forma sencilla por lo que se tiene que procesar con distintos métodos. Los más comunes son: la interpolación de 11 puntos y la interpolación de todos los puntos.

Finalmente, tras procesar la curva de la precisión y el *recall*, se calcula el AP para cada clase midiendo el área bajo la curva. Para el cálculo del mAP, se hace una media del AP de todas las clases [47].

Capítulo 4.

Implementación y ejecución

4.0 Introducción

En este capítulo se explicarán las tecnologías empleadas, el hardware utilizado en las ejecuciones y las funciones implementadas. Por último, se muestra una ejecución por cada tipo de transformación posible para poder entender el proceso descrito en el apartado de metodología de las ejecuciones con una mayor profundidad.

4.1 Tecnologías utilizadas

La principal librería de *python* que se han utilizado para la implementación de este proyecto ha sido *PyTorch*, la cual, ha sido necesaria para la ejecución del modelo ya que como hemos comentado en capítulos previos, el modelo está implementado en ese *framework* para poder ser ejecutado en la GPU.

Se han utilizado otras librerías como *random*, para poder generar parámetros aleatorios en las funciones de transformación de las imágenes. *Pandas* ha sido otra de las librerías empleadas por su sencillez en el manejo de estructuras de datos. Para las rotaciones de las imágenes y las *bounding boxes* se ha empleado la librería *cv2*. La librería empleada tanto para realizar los ajustes de saturación, contraste, color y brillo como para trabajar con imágenes ha sido PIL (*Python Imaging Library*). De esta librería, se ha utilizado concretamente el módulo *ImageEnhance* que tiene implementado los ajustes anteriormente nombrados y que sólo necesita la generación de un parámetro para controlar la intensidad del filtro [46]. A este parámetro ya se le ha hecho referencia con anterioridad en el capítulo 3.6 de la memoria.

El proyecto ha sido implementado utilizando dos *notebooks* de *Jupyter Notebook*. Uno de ellos que contenía todas las funciones implementadas y el

segundo a modo de *main* para realizar las llamadas a estas y ejecutar el proyecto dónde se calcula el mAP.

También se ha implementado utilizando *scripts* de *Python* para poder ejecutarlo utilizando la consola de comandos sin necesidad de instalar *Jupyter Notebook*.

4.2 Especificación del hardware

Se han tenido que realizar multitud de pruebas para poder calcular el mAP en cada una de ellas y se ha utilizado la GPU para acelerar el proceso de cómputo. Se ha empleado una *GPU NVIDIA GeForce GTX1050* con *4GB GDDR5* de *VRAM*, un procesador *Intel Core i7-8750H*, una *RAM* de *8GB DDR4* y un *SSD* de *256GB M.2 NVME*. El tiempo total de las pruebas ha sido de 30 horas, pero hubiera sido mucho más elevado si se hubiera ejecutado en la *CPU* en vez de en la *GPU*.

4.3 Descripción de las funciones implementadas

En primer lugar, se ha implementado una función que busca en el archivo de anotaciones los cuadros delimitadores esperados (*ground truth*) de las imágenes que se estén utilizando. Estas imágenes serán las que se encuentren dentro de la carpeta *original_images* en el momento en el que se realice la ejecución.

Se han implementado 5 funciones que realizan ajustes de brillo, color, saturación y contraste. Para su funcionamiento, se genera un parámetro con el que se transformará la imagen. El parámetro generado se encuentra entre un rango de valores que se le tienen que especificar a la función que lo genera y que sigue una distribución uniforme. Estos parámetros han de tener un valor coherente ya que, si tienen un valor muy alto o bajo, pueden distorsionar la imagen y provocar detecciones erróneas.

Para el caso de las rotaciones, se ha tenido que crear otro tipo de función para que genere un ángulo de rotación de la imagen. El rango de ángulos que se rotará una imagen irá desde 355º hasta 5º, en total 10º de rotaciones posibles. Una vez la imagen se ha rotado y se han obtenido las *bounding box*es de las detecciones, se tiene que deshacer esta rotación para que las *bounding box*es

tengan una orientación adecuada y poder compararlas con las *bounding box*es de la *ground truth*. Debido a ello, se han tenido que implementar otras funciones para deshacer las rotaciones de dichas *bounding box*es.

Una vez tenemos las funciones que transforman la imagen, se le proporciona como entrada al modelo este nuevo *dataset* de imágenes transformadas. Para ello se ha creado otra función que obtiene los cuadros delimitadores y la confianza de cada imagen transformada. Esta función también obtendrá las detecciones de las imágenes sin transformar ya que serán necesarias posteriormente para calcular su mAP y poder comparar.

Tras obtener las detecciones, se ha implementado una función para aplicar el algoritmo *Non-maximum Suppression* y suprimir las *bounding boxes* redundantes.

Todas estas detecciones se escriben en archivos de texto con el nombre de la imagen a la que corresponden. Además, también se han de escribir en archivos de texto los valores de las *ground truths* para cada imagen empleada. Debido a esto, se ha implementado una funcione para escribir en las carpetas del proyecto que calcula la métrica.

Para la evaluación de la métrica se ha usado un proyecto de *python* que se publicado en *GitHub* [48] ya que cuenta con algunas funciones muy fáciles de usar para realizar el cálculo del mAP. Se le han hecho algunas modificaciones ya que su implementación no estaba adaptada para objetos con nombres múltiples como por ejemplo osos de peluche (*teddy bears*).

4.4 Metodología de las ejecuciones

Inicialmente se calculan las *ground truths* del conjunto de imágenes que se va a utilizar ya que se necesitarán para calcular el mAP tanto de las imágenes transformadas como de las originales. Se escribe un archivo de texto por cada una de las imágenes con todas las clases de objetos que contienen y la posición de sus *bounding boxes*.

Tras obtener las *ground truths* se calcula el mAP del *dataset* sin transformar, es decir, la métrica que queremos mejorar. Este cálculo se realiza siguiendo el diagrama de flujo de la figura 4.1.

Una vez obtenido el mAP del dataset sin transformar, se calcula el mAP del *dataset* al aplicar transformaciones. Dependiendo de qué tipo de

transformación se haya aplicado, se seguirá una metodología u otra. En la figura 4.2 se muestra el flujo para transformaciones simples de ajuste único. En la figura 4.3 se puede ver el flujo para transformaciones simples con varios ajustes y en ese caso se están aplicando 3 de los ajustes nombrados en el capítulo 3.6. Finalmente, en la figura 4.4 se expone el flujo que se sigue al aplicar transformaciones compuestas con dos ajustes sobre la misma imagen.

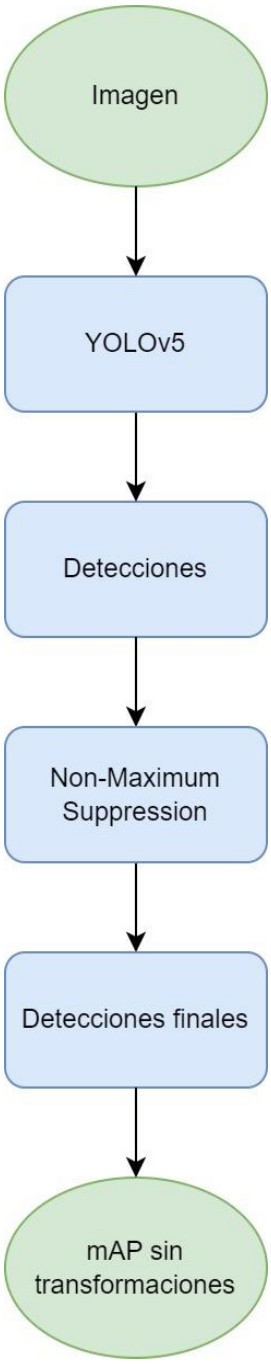

Figura 4.1: Diagrama de flujo para el cálculo del mAP de una imagen sin transformar.

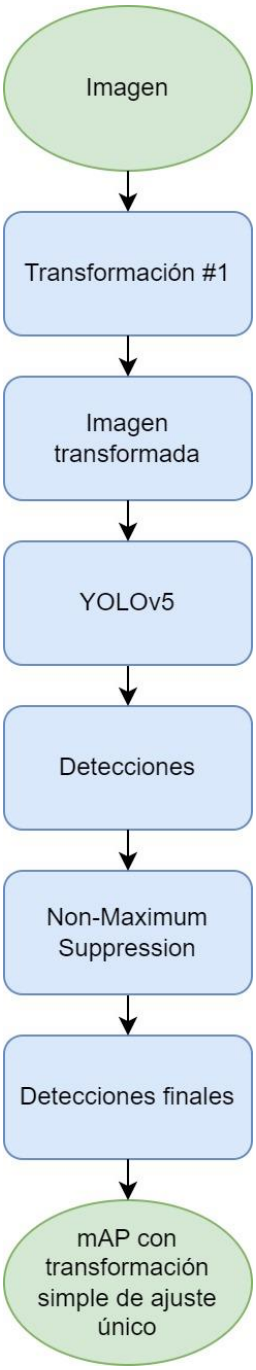

Figura 4.2: Diagrama de flujo para el cálculo del mAP con transformaciones simples de ajuste único

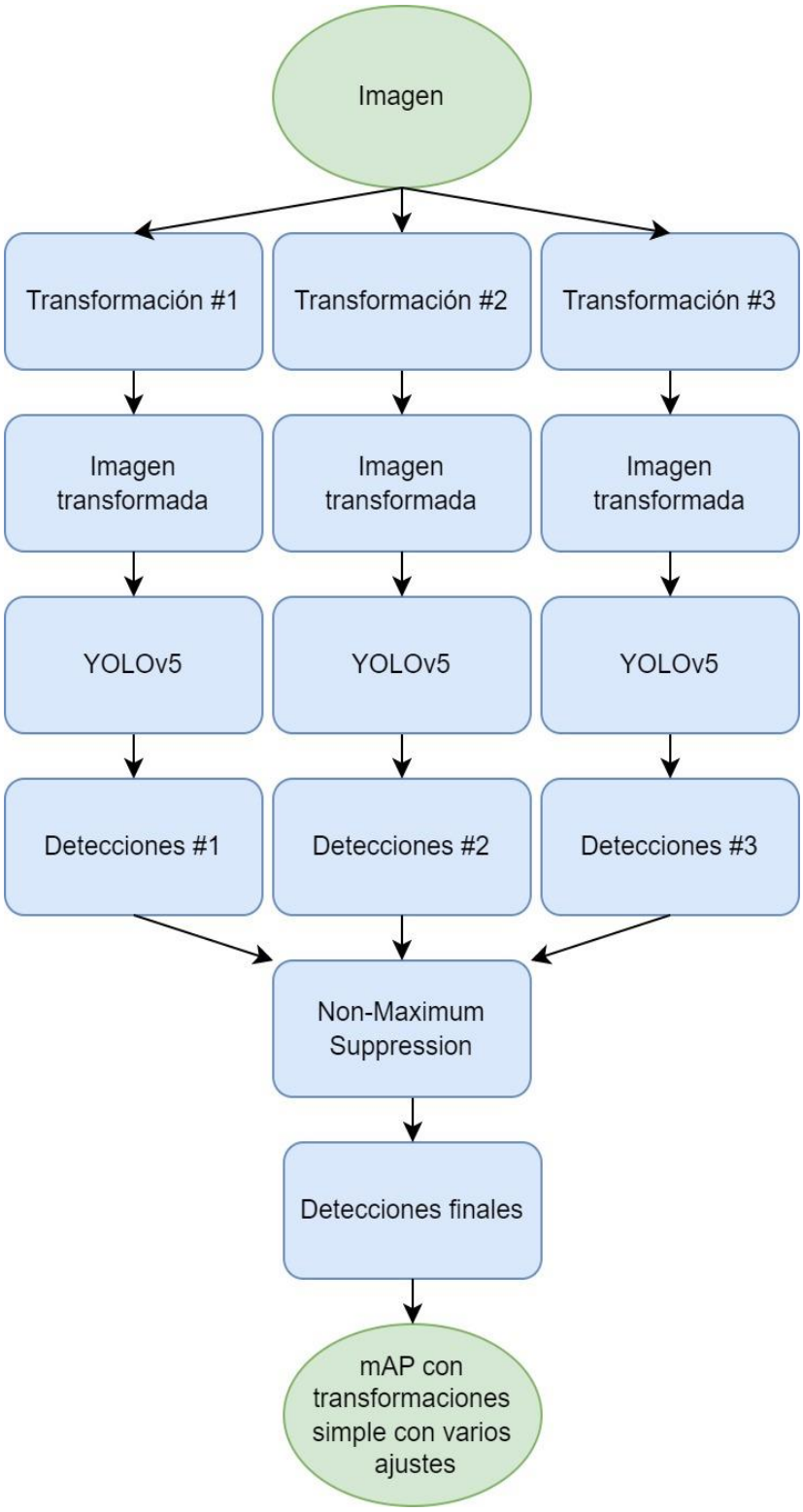

Figura 4.3: Diagrama de flujo para el cálculo del mAP con transformaciones simples con varios ajustes.

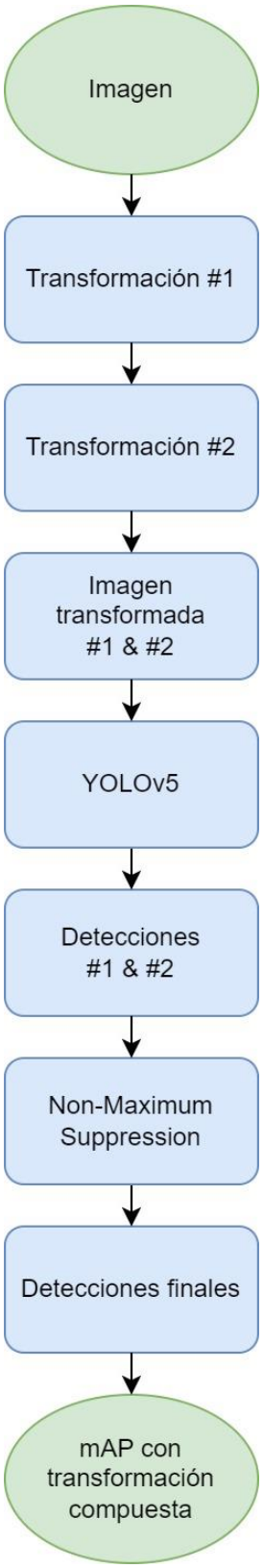

Figura 4.4: Diagrama de flujo para el cálculo del mAP con transformaciones compuestas.

Respecto a las figuras anteriores, cuando el flujo pasa por YOLOv5 se refiere a que se generan todas las detecciones de la imagen. *En el caso de Non-Maximum Suppression*, las detecciones generadas anteriormente se simplifican con este algoritmo para generar las detecciones finales. Estas detecciones se escriben en un archivo de texto del proyecto en el cual se calcula la métrica. Tras escribir un archivo por cada imagen con cada una de las detecciones finales, ya se puede ejecutar el proyecto que calcula el mAP.

4.5 Ejemplo de ejecución de la metodología

En este apartado vamos a ver todos los cambios que experimenta la imagen y como se van obteniendo las detecciones y las *bounding box*es a lo largo de todo el proceso. Se aplicarán los 3 tipos de transformaciones, transformaciones simples con un único ajuste, transformaciones simples con varios ajustes y transformaciones compuestas.

Figura 4.5: Imagen del dataset MS COCO 2014.

Inicialmente, obtenemos el valor de la *ground truth* de la imagen anterior que se encuentra en el archivo de anotaciones que tiene el *dataset MS COCO*.

```
dog       106.54    35.31    361.7    307.58
frisbee   143.93    100.46   115.15   115.15
```

Figura 4.6: Ground truth extraída de las anotaciones de MS COCO para la imagen anterior.

Según la información extraída de las anotaciones (figura 4.6), se deben realizar dos detecciones, una de ellas es un perro y la otra un disco volador. Respecto a los parámetros extraídos, obtenemos los primeros dos valores que son las posiciones *x* e *y* de la esquina inferior izquierda del cuadro delimitador y los otros dos valores serán el ancho y el alto de tal cuadro. La información extraída se guarda en un archivo de texto en la carpeta *ground truth* del proyecto que calcula el mAP ya que será necesario para calcular el mAP.

4.5.1 mAP con las imágenes originales

Tras obtener los valores de la *ground truth*, se calcula el mAP de las imágenes originales. Este valor será el que se intentará mejorar al aplicar las transformaciones.

Se introduce el conjunto de imágenes originales como entrada del modelo y obtenemos las detecciones asociadas a cada una de estas. Tras obtener las detecciones aplicamos el algoritmo *Non-maximum Suppression* para simplificar las *bounding box*es redundantes.

En el caso de la imagen usada como ejemplo (figura 4.5), se han realizado las siguientes detecciones.

	xmin	ymin	xmax	ymax	confidence	class	name	Deteccion
0	141.768784	99.592819	259.092468	223.860825	0.940400	29	frisbee	0
1	118.649261	34.501984	468.763153	331.917389	0.704025	16	dog	0

Figura 4.7: Detecciones que realiza YOLOv5 con la figura 4.5 como entrada.

En primer lugar, YOLOv5 devuelve las posiciones de las esquinas inferior izquierda y superior derecha de las *bounding boxes*. En segundo lugar, se devuelve la confianza, la clase y el nombre de la clase a la que pertenecen.

Las *bounding boxes* de la *ground truth* se adaptan perfectamente al contorno del objeto (imagen izquierda) mientras que las *bounding boxes* extraídas por YOLOv5 se adaptan levemente peor (imagen derecha).

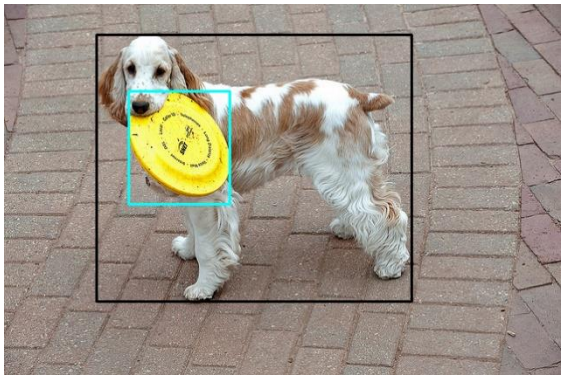

Figura 4.8: Representación de las bounding boxes extraídas de la ground truth sobre la imagen original.

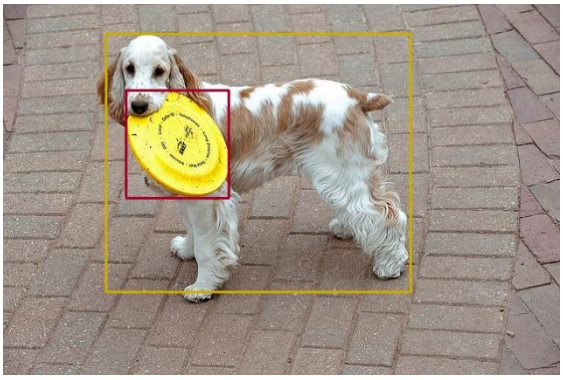

Figura 4.9: Representación de las bounding boxes obtenidas por YOLOv5 sobre la imagen original.

Una vez obtenemos las detecciones de las imágenes originales, las escribimos en un archivo de texto. Este archivo se guarda en la carpeta *detections* del proyecto que calcula el mAP. Una vez hemos generado todos los archivos con las detecciones sobre el *dataset* de imágenes originales, se puede calcular el mAP de dicho *dataset* que será el mAP que queremos mejorar.

4.5.2 mAP usando transformaciones simples con ajuste único

En las transformaciones simples con un único ajuste, se aplicará el ajuste en la saturación para transformar la imagen original. Se realizará la detección en la imagen transformada y se simplificarán las *bounding boxes* redundantes con *Non-maximum Suppression*. La imagen empleada sigue siendo la de la figura 4.5.

	xmin	ymin	xmax	ymax	confidence	class	name	Deteccion
0	141.408356	101.784042	259.638519	221.642929	0.899590	29	frisbee	0
1	119.690811	73.915527	464.798157	331.729889	0.380759	22	zebra	0
2	107.073288	66.527130	465.267456	344.069305	0.335693	16	dog	0

Figura 4.10: Detecciones de YOLOv5 al realizar un ajuste en la saturación.

En este caso, se puede ver en la figura 4.10 cómo se ha producido una detección errónea ya que se ha detectado una cebra en la imagen. En la figura 4.11, se puede ver como el ajuste al perro no es demasiado bueno.

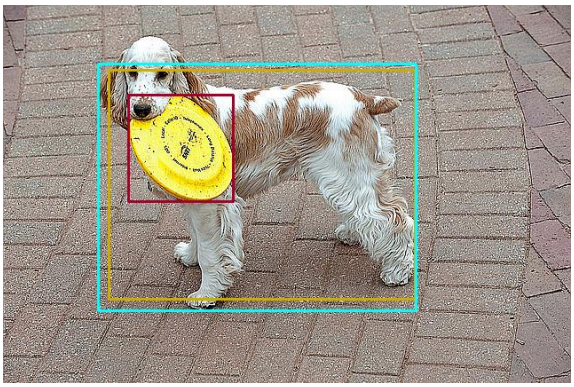

Figura 4.11: Representación de las bounding boxes detectadas por YOLOv5 con una imagen con un ajuste en la saturación.

Tras obtener esta información del modelo, la escribimos en la carpeta de las métricas y ya podemos calcular el mAP para transformaciones simples con una única imagen.

4.5.3 mAP usando transformaciones simples con varios ajustes

Para las transformaciones simples con varios ajustes, utilizaremos los ajustes de rotación, contraste y saturación que aplicaremos a la figura 4.5.

Figura 4.12: Ajuste en la rotación.

Figura 4.13: Ajuste en el contraste.

Figura 4.14: Ajuste en la saturación.

Tras realizar cada uno de estos ajustes, obtendremos las detecciones del modelo. Cada una de las imágenes produce una detección como podemos ver en la columna detección de la figura 4.15. La detección 0 está asociada a la figura 4.12, la detección 1 con la figura 4.13 y la detección 2 con la figura 4.1.

	xmin	ymin	xmax	ymax	confidence	class	name	Deteccion
0	148.255160	105.179218	267.498909	232.246190	0.944880	29	frisbee	0
1	117.887974	39.256737	479.448009	342.413004	0.746058	16	dog	0
0	141.854721	100.505997	259.160217	223.373489	0.932929	29	frisbee	1
1	118.193741	46.762115	468.758667	331.888489	0.751263	16	dog	1
0	142.400726	100.911316	259.781799	223.672668	0.925251	29	frisbee	2
1	122.938873	62.199524	466.218170	334.394775	0.750703	16	dog	2

Figura 4.15: Detecciones de YOLOv5 para las tres imágenes transformadas anteriormente.

Estas detecciones se simplifican, como se puede observar en la figura 4.12, con el algoritmo *Non-maximum Suppression* para obtener una detección final que es la que se escribirá en un archivo de texto en la carpeta *detections* del proyecto en el que se calcula el mAP.

	xmin	ymin	xmax	ymax	confidence	class	name	Deteccion
0	148.255160	105.179218	267.498909	232.246190	0.944880	29	frisbee	0
1	118.193741	46.762115	468.758667	331.888489	0.751263	16	dog	0

Figura 4.16 : Detecciones simplificadas tras aplicar Non-maximum Suppression.

4.5.4 mAP usando transformaciones compuestas

En este tipo de transformaciones se aplicará un ajuste en el contraste y posteriormente un ajuste en la saturación. Estos ajustes se aplicarán siguiendo el orden en el que están definidos. En la figura 4.17 se puede observar cómo afectan las transformaciones sobre la figura 4.18.

Figura 4.17: Imagen original.

Figura 4.18: Imagen con un ajuste en el contraste y la saturación.

Tras generar la imagen con ambas transformaciones, obtenemos las detecciones de YOLO y finalmente aplicamos el *Non-maximum Suppression*.

	xmin	ymin	xmax	ymax	confidence	class	name	Deteccion
0	143.541397	98.189087	261.177826	223.934479	0.791782	29	frisbee	0
1	244.712494	86.290161	466.479523	317.886505	0.665622	22	zebra	0

Figura 4.19: Detecciones de YOLOv5 de la imagen con transformación en contraste y saturación.

Como podemos ver, al combinar ajustes en una misma imagen, podemos obtener transformaciones erróneas. En el siguiente capítulo estudiaremos los resultados obtenidos y veremos cómo este tipo de transformaciones no han obtenido buenos resultados debido a que distorsionan demasiado la imagen.

Capítulo 5.

Análisis de los resultados

5.0 Introducción

Se han utilizado 3 conjuntos de imágenes cuyos tamaños son de 60, 120 y 240. Se han elegido esos conjuntos para poder estudiar la evolución de la métrica con conjuntos proporcionales entre sí. Estos *datasets* se han extraído del conjunto de validación del *dataset* de imágenes COCO, como ya se expuso en el capítulo tercero.

Estos conjuntos contienen imágenes de todo tipo desde animales, vehículos, señales de tráfico, hasta utensilios de cocina. El número medio de objetos detectables en los *datasets* es de aproximadamente 2 objetos por imagen. El filtro para construir los *datasets* ha consistido en que las imágenes no podían contener más de 6 objetos por imagen. Esto se debe a que con pruebas anteriores se había detectado que, al realizar transformaciones con gran cantidad de objetos solapados, las detecciones fallaban en multitud de casos.

Para la obtención de la métrica mAP sobre cada uno de los conjuntos anteriores, se ha variado el valor del parámetro *threshold* del *Intersection Over Union* tomando los valores de 50%, 75% y 95%.

No se han realizado más pruebas con *datasets* de mayor tamaño debido al tiempo de ejecución. El tiempo total de ejecución de los tres *datasets,* utilizando en cada uno de ellos los tres valores distintos del *threshold*, ha sido de 30 horas aproximadamente. Para cada *dataset* se ha empleado un total de 5, 10 y 15 horas respectivamente.

Para cada valor de la métrica se ha repetido el proceso de cálculo 10 veces, empleando las mismas las transformaciones sobre el mismo dataset y usando el threshold correspondiente. Tras calcular el valor de la métrica se ha hecho una media entre todos los valores obtenidos. Este proceso se ha seguido para suavizar el resultado y evitar valores extremos.

5.1 Explicación del contenido de las tablas

Se han aplicado los tres tipos de transformaciones definidos con anterioridad. Una transformación simple con ajuste único, como su mismo nombre indica, habrá sido aplicada cuando en la columna de transformaciones aplicadas aparece un único ajuste (por ejemplo: brillo).

Las transformaciones simples con varios ajustes se aplican en aquellos casos dónde los ajustes de la columna transformaciones, se encuentran separados con comas (por ejemplo: brillo, saturación, color).

Las transformaciones compuestas vienen en la columna de transformaciones aplicadas como la suma de dos ajustes (por ejemplo: brillo + saturación).

En la primera columna se encuentra el mAP de las imágenes originales que es el resultado que ha proporcionado la métrica mAP con las imágenes sin transformar. Este valor es el que queremos mejorar al aplicar las transformaciones.

En la segunda columna está el mAP de las imágenes transformadas que es el nuevo valor que hemos obtenido tras aplicar las transformaciones.

En la tercera columna se ha incluido el porcentaje de mejora de cada transformación sobre el valor del mAP obtenido con las imágenes originales.

Por último, en la cuarta columna tenemos las transformaciones que se han aplicado en cada caso.

5.2 Resultados utilizando un *threshold* del 50%

En la tabla 5.1 se muestran los resultados de la métrica mAP al utilizar los 3 tipos de transformaciones y empleando un *dataset* de 240 imágenes y un *threshold* del 50%.

Las transformaciones simples con varios ajustes han sido las que mejor resultado han obtenido. Siendo el mejor valor de la métrica 51.65% con ajustes de rotación, brillo, color, frente a 39.53% del mAP original. Se ha mejorado el valor del mAP en un 12.12%. Otras transformaciones como brillo, color y contraste, color han mejorado el valor de la métrica en un 11.81% y un 11.21% respectivamente.

Respecto a las transformaciones simples con ajuste único y a las transformaciones compuestas, en este caso no han obtenido ningún resultado a destacar.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	51.65%	12.12%	Rotación, brillo, color
	51.34%	11.81%	Brillo, color
	50.74%	11.21%	Contraste, color
	50.62%	11.09%	Brillo, contraste
39.53%	50.16%	10.63%	Brillo, saturación, color
	50.13%	10.60%	Brillo, contraste, color
	49.56%	10.03%	Rotación, saturación, color
	49.48%	9.95%	Brillo, saturación
	48.47%	8.94%	Saturación, color
	48.19%	8.66%	Contraste, saturación

Tabla 5.1: Resultados del mAP en un dataset de 240 imágenes con un threshold del 50%.

En la tabla 5.2 se exponen los resultados obtenidos al utilizar el *dataset* de 120 imágenes. Se han extraído resultados similares a los anteriores, siendo las transformaciones simples con varios ajustes las que mejor han funcionado. Empleando el ajuste de rotación, brillo, color se ha obtenido un valor en la métrica de 55.45% frente a 42.13% que era el valor del mAP original, mejorando la métrica en un 13.32%. El ajuste de contraste, saturación también ha obtenido un buen resultado, mejorando el valor de la métrica en un 11.01%

Respecto a las transformaciones simples con ajuste único, el ajuste en el color ha sido el más destacable, mejorando el mAP en un 0.41%.

Las transformaciones compuestas no han mejorado el valor del mAP original.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	55.45%	13.32%	Rotación, brillo, color
	53.14%	11.01%	Contraste, saturación
	52.73%	10.60%	Brillo, color
	52.73%	10.60%	Brillo, saturación, color
	52.39%	10.26%	Brillo, saturación
42.13%	52.10%	9.97%	Contraste, color
	51.70%	9.57%	Brillo, contraste
	51.69%	9.56%	Brillo, contraste, color
	51.08%	8.95%	Rotación, saturación, color
	49.37%	7.24%	Saturación, color
	42.54%	0.41%	Color

Tabla 5.2: Resultados del mAP en un dataset de 120 imágenes con un threshold de 50%.

Se han obtenido resultados similares tanto con los *datasets* de mayor tamaño como con el más reducido que se puede ver en la tabla 5.3. De nuevo, las transformaciones simples con varios ajustes son las que más han mejorado el valor de la métrica. La transformación con ajuste en brillo, contraste ha mejorado el valor del mAP de las imágenes sin transformar en un 16.68%, pasando de un 49.21% a un 65.89%. La combinación con tres ajustes: rotación, brillo, color han mejorado el valor de la métrica en un 16.45%.

En las transformaciones simples con un único ajuste el color y el brillo han mejorado el mAP en un 1.02% y un 0.67% respectivamente.

En este caso, la transformación compuesta de brillo + color ha mejorado el valor de la métrica en un 1.31%.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	65.89%	16.68%	Brillo, contraste
	65.66%	16.45%	Rotación, brillo, color
	65.42%	16.21%	Contraste, color
	65.15%	15.94%	Brillo, saturación, color
	64.96%	15.75%	Brillo, saturación
	64.96%	15.75%	Brillo, color
49.21%	64.73%	15.15%	Brillo, contraste, color
	64.55%	15.34%	Contraste, saturación
	64.31%	15.10%	Rotación, saturación, color
	64.25%	15.04%	Saturación, color
	51.43%	2.22%	Rotación, brillo
	50.52%	1.31%	Brillo + color
	50.23%	1.02%	Color
	49.88%	0.67%	Brillo
	49.83%	0.62%	Rotación, contraste

Tabla 5.3: Resultados del mAP en un dataset de 60 imágenes con un threshold de 50%.

5.3 Resultados utilizando un *threshold* de 75%

En las siguientes tablas, se muestran los resultados obtenidos al emplear cada uno de los 3 *datasets* de tamaños diferentes con un *threshold* de 75%.

Como se puede ver en la tabla 5.4, se ha empleado el *dataset* de 240 imágenes. Las transformaciones con varios ajustes son las que mejor resultado han obtenido. En concreto, el ajuste de contraste, color y la transformación de brillo, color han mejorado el valor de la métrica con respecto del original en un 9.04% y 8.27% respectivamente.

Las transformaciones con ajuste único no han mejorado suficientemente el valor del medidor, obteniendo un resultado de mejora con un ajuste de color de un 0.05%.

Respecto a las transformaciones compuestas, en este caso tampoco han obtenido ningún resultado destacable.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	45.27%	9.04%	Contraste, color
	44.50%	8.27%	Brillo, color
	44.31%	8.08%	Contraste, saturación
	44.15%	7.92%	Brillo, contraste, color
	43.46%	7.23%	Brillo, saturación, color
36.23%	43.22%	6.99%	Brillo, contraste
	43.20%	6.97%	Saturación, color
	42.99%	6.76%	Brillo, saturación
	41.18%	4.95%	Rotación, brillo, color
	39.00%	2.77%	Rotación, saturación, color
	36.28%	0.05%	Color

Tabla 5.4: Resultados del mAP en un dataset de 240 imágenes con un threshold de 75%.

Con el *dataset* de 120 imágenes y un *threshold* de 75%, la transformación más destacada han sido las transformaciones simples con varios ajustes. Siendo los ajustes de brillo, saturación, color y brillo, contraste, color los que mejor resultado han obtenido mejorando la métrica en un 9.97% y 8.75%.

Las transformaciones simples con ajuste único más destacadas han sido la transformación en color y en saturación. Estas han obtenido una mejora en la métrica del 0.75% y 0.07% respectivamente.

Las transformaciones compuestas tampoco han obtenido resultados favorables para este *dataset* y esta configuración de *threshold*.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	46.87%	9.97%	Brillo, saturación, color
	45.65%	8.75%	Brillo, contraste, color
	45.62%	8.72%	Contraste, saturación
	45.06%	8.16%	Contraste, color
	44.74%	7.84%	Brillo, saturación
36.90%	44.69%	7.79%	Brillo, contraste
	44.65%	7.75%	Brillo, color
	43.79%	6.89%	Rotación, brillo, color

	43.72%	6.82%	Rotación, saturación, color
	42.91%	6.01%	Saturación, color
	37.65%	0.75%	Color
	36.97%	0.07%	Saturación

Tabla 5.5: Resultados del mAP en un dataset de 120 imágenes con un threshold de 75%.

Por último, se ha empleado el *dataset* de 60 imágenes, como se puede ver en la tabla 5.6. Las transformaciones simples con varios ajustes han sido las más destacables. Las combinaciones de contraste, color y brillo, contraste han mejorado el valor de la métrica en un 14.66% y un 14.28%.

Las transformaciones simples con ajuste único destacadas han sido la de color y la de brillo, aunque no han supuesto una mejora destacable al obtener una mejora de un 0.90% y 0.39%.

En este caso, las transformaciones compuestas no han obtenido ningún resultado resaltable.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	59.22%	14.66%	Contraste, color
	58.84%	14.28%	Brillo, contraste
	58.59%	14.03%	Brillo, contraste, color
	58.14%	13.58%	Brillo, saturación, color
	57.80%	13.24%	Saturación, color
	57.05%	12.49%	Brillo, color
44.56%	56.50%	11.94%	Brillo, saturación
	55.86%	11.30%	Contraste, saturación
	55.02%	10.46%	Rotación, saturación, color
	53.17%	8.61%	Rotación, brillo, color
	45.46%	0.90%	Color
	44.95%	0.39%	Brillo

Tabla 5.6: Resultados del mAP en un dataset de 60 imágenes con un threshold de 75%.

5.4 Resultados utilizando un *threshold* de 95%

En estos últimos resultados se ha aplicado un valor del *threshold* más restrictivo que en los apartados 5.3 y 5.2. Al igual que en estos, se ha realizado el estudio con los tres *dataset* de distintos tamaños.

En la tabla 5.7, se muestran los resultados tras utilizar el *dataset* de 240 imágenes. Las transformaciones simples con varios ajustes han vuelto a ser las más destacables siendo la transformación en contraste, color la que mejor resultado ha obtenido. Esta transformación ha mejorado la métrica original en un 0.91% pasando de 10.75% a 11.66%.

Tanto las trasformaciones compuestas como las transformaciones simples con ajuste único no han obtenido resultados destacables.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
10.75%	11.66%	0.91%	Contraste, color
	11.16%	0.41%	Brillo, color
	11.13%	0.38%	Brillo, contraste, color
	11.12%	0.37%	Brillo, contraste
	11.03%	0.28%	Brillo, saturación, color
	10.99%	0.24%	Contraste, saturación
	10.91%	0.16%	Brillo, saturación

Tabla 5.7: Resultados del mAP en un dataset de 240 imágenes con un threshold de 95%.

Utilizando el *dataset* de 120 imágenes como se puede ver en la tabla 5.8, las transformaciones simples con varios justes con los ajustes de contraste, saturación y contraste, color han sido los que más han mejorado el valor de la métrica, haciéndolo en un 3.00% y un 2.47% respectivamente.

Las transformaciones simples con ajuste único más destacadas han sido la de contraste, la de color y la de saturación con un 1.75%, 0.63% y 0.31% de mejora respectivamente.

Se puede destacar una leve mejora de 0.06% empleando las transformaciones compuestas de contraste + color.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	14.77%	3.00%	Contraste, saturación
	14.24%	2.47%	Contraste, color
	13.52%	1.75%	Contraste
	13.05%	1.28%	Brillo, saturación, color
	12.84%	1.07%	Brillo, color
11.77%	12.66%	0.89%	Saturación, color
	12.40%	0.63%	Color
	12.36%	0.59%	Brillo, contraste, color
	12.26%	0.49%	Brillo, contraste
	12.08%	0.31%	Saturación
	11.83%	0.06%	Contraste + color

Tabla 5.8: Resultados del mAP en un dataset de 120 imágenes con un threshold de 95%.

Por último, se mostrarán los resultados al emplear el *dataset* de menor tamaño como se puede ver en la tabla 5.9.

La transformación simple con varios ajustes más destacada ha sido la de brillo, contraste mejorando el mAP de las imágenes originales en un 3.14% pasando de un 16.96% a un 20.10%.

Respecto a las transformaciones simples de ajuste único, el brillo y el color has sido los ajustes más destacados obteniendo un 1.70% y un 0.68% de mejora respectivamente.

Las transformaciones compuestas de brillo + contraste y contraste + color han mejorado el valor de la métrica en un 1.09% y 0.55%.

mAP Imágenes Originales	*mAP* Imágenes Transformadas	Porcentaje de mejora	Transformaciones Aplicadas
	20.10%	3.14%	Brillo, contraste
	19.37%	2.41%	Brillo, saturación, color
	19.19%	2.23%	Brillo, saturación
	18.79%	1.83%	Brillo, color
	18.76%	1.80%	Brillo, contraste, color
16.96%	18.66%	1.70%	Brillo

18.05%	1.09%	Brillo + contraste
17.64%	0.68%	Color
17.51%	0.55%	Contraste + color
17.39%	0.43%	Contraste, color
17.06%	0.10%	Contraste, saturación

Tabla 5.9: Resultados del mAP en un dataset de 60 imágenes con un threshold de 95%.

Capítulo 6.

Conclusiones y líneas futuras

En este capítulo se analizarán los resultados y se obtendrán conclusiones de estos. Se ha de tener en cuenta que los resultados obtenidos son los extraídos para *datasets* de tamaños 240, 120 y 60 imágenes por lo que habría que estudiar cómo varía el comportamiento de las transformaciones con *datasets* mucho más extensos para poder formalizar conclusiones más vigorosas.

Se pueden encontrar similitudes en el conjunto global de las pruebas realizadas respecto al rendimiento de las transformaciones empleadas.

Las transformaciones simples con varios ajustes son las que mejor resultado han tenido independientemente del tamaño del *dataset* empleado o del valor del *threshold* definido. La máxima mejora obtenida con este tipo de transformaciones es de 16.68% sobre la métrica de las imágenes originales, empleando un *threshold* poco restrictivo (50%) y usando el *dataset* de menor tamaño (60 imágenes). Esta información se encuentra en la tabla 5.3. Las transformaciones más destacadas han sido las de contraste, color y la de brillo, contraste.

Las transformaciones simples con ajuste único no han obtenido resultados aceptables, comparándolos con los anteriores. Los ajustes más destacados en este caso han sido los de color y de brillo. Estas transformaciones han superado a las transformaciones compuestas, pero están lejos de las mejoras conseguidas con las transformaciones simples con varios ajustes. La máxima mejora que han conseguido ha sido de 1.75% con un ajuste en el contraste, empleando el *dataset* de 120 imágenes y un *threshold* de 95% (esta información se encuentra presentada en la tabla 5.8).

Respecto de las transformaciones compuestas, al igual que con las transformaciones anteriores, no han resultado ser transformaciones útiles ya que no han obtenido resultados favorables. Uno de los motivos por los que no han

conseguido mejores resultados es el hecho de que distorsionan demasiado la imagen. Es decir, provocan detecciones erróneas en gran cantidad de casos al modificar la imagen original varias veces. A pesar de esto, la transformación compuesta de brillo + color ha sido la más destacable en las pruebas obteniendo una mejora de 1.31% sobre el mAP original y empleando el *dataset* de 60 imágenes y un *threshold* del 50% (tabla 5.3).

Se pueden encontrar tendencias en los resultados ya que el valor del mAP de las imágenes originales crece cuanto más se reduce el *dataset* de imágenes empleado para la prueba. Esto es coherente ya que, si hay menos imágenes, se cometerán menos errores. Esta tendencia se mantiene independientemente del valor del *threshold* que estemos empleando.

Analizando globalmente los resultados, se puede ver cómo los valores de mejora del mAP también siguen la tendencia del mAP de las imágenes originales, obteniendo un valor más bajo con el *dataset* de 240 imágenes y un valor mayor con el de 60 imágenes.

Los mejores resultados se han obtenido al emplear el *dataset* de menor tamaño, el de 60 imágenes, y un *threshold* poco restrictivo, del 50% y los resultados menos favorables se han obtenido al emplear el *threshold* del 95% con el *dataset* de 240 imágenes.

En vista a futuras versiones del proyecto, se pueden emplear transformaciones más sofisticadas para seguir estudiando el comportamiento de estas sobre el mAP y se podrían emplear *datasets* de mayor tamaño para poder llegar a conclusiones más rigorosas.

Se pueden aplicar diferentes filtros de suavizado para las imágenes. Esta operación se usa frecuentemente en el procesamiento de imagen y lo que realmente aplica a la imagen es un desenfoque. La finalidad de estos filtros es reducir el ruido.

Uno de estos filtros de suavizado aplicables es el suavizado Gaussiano de la imagen. En este tipo de transformación, se aplica un kernel de Gauss sobre la matriz asociada a los píxeles de la imagen.

Se podría aplicar también el filtro de mediana, en el cual, se reemplaza cada píxel de la imagen con la mediana de sus píxeles vecinos.

Se pueden aplicar distintas transformaciones geométricas además de la rotación. Se pueden realizar translaciones, cambio de dimensión y escalado, incluso recortar la imagen.

Por último, podemos aplicar transformaciones no lineales. Un ejemplo de estas transformaciones puede ser el filtro de ojo de pez que se aplica tomando un píxel como centro. Otra transformación no lineal es el filtro de onda horizontal o vertical que distorsiona la imagen en base a una amplitud y una frecuencia.

Bibliografía

[1] Wikipedia. *Visión artificial*. Fecha de consulta: 18 de septiembre del 2022. Disponible en <https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial>

[2] Dynatec. *Computer Visión: la visión artificial y sus aplicaciones.* Fecha de consulta 18 de septiembre de 2022. Disponible en <https://dynatec.es/2021/07/24/computer-vision-la-vision-artificial-y-sus-aplicaciones/>

[3] *Introducción a la Neurocomputación*. Fecha de consulta: 18 de septiembre de 2022. Disponible en <https://xdoc.mx/preview/tema-2-5f8528a0c648e>

[4] P. Gupta. *The Cognitron and Neocognitron Notes*. Fecha de consulta: 18 de septiembre de 2022. Disponible en <https://sites.google.com/site/hacksbyredbios/Downhome/Topic5/thecognitronandneocognitronnotes>

[5] M de Lecture. *Machine Learning: definición, funcionamiento, usos*. Fecha de consulta: 18 de septiembre de 2022. Disponible en <https://datascientest.com/es/machine-learning-definicion-funcionamiento-usos>

[6] J. Martinez Heras. *Las 7 Fases del Proceso de Machine Learning*. Fecha de consulta: 18 de septiembre de 2022. Disponible en < https://www.iartificial.net/fases-del-proceso-de-machine-learning/>

[7] Tibco. *¿Qué es el aprendizaje supervisado?* Fecha de consulta: 18 de septiembre de 2022. Disponible en <https://www.tibco.com/es/reference-center/what-is-supervised-learning>

[8] S. Russell, P. Norvig. *Inteligencia Artificial: Un enfoque moderno,* 2008*.* Fecha de consulta: 18 de septiembre de 2022. Disponible en < http://jdelagarza.fime.uanl.mx/IA/Libros/inteligencia-artificial-un-enfoque-moderno-stuart-j-russell.pdf>

[9] Tibco. *¿Qué es el aprendizaje no supervisado?*
Fecha de consulta: 5 de septiembre de 2022.
Disponible en <https://www.tibco.com/es/reference-center/what-is-unsupervised-learning>

[10] Na8. *Aprendizaje por Refuerzo*.
Fecha de consulta: 5 de septiembre de 2022.
Disponible en <https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/>

[11] Bootcamp AI. *Redes neuronales. Programa de Visión Artificial/Computacional*. Fecha de consulta: 22 de septiembre de 2022.
Disponible en <https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb>

[12] Wikipedia. *Perceptrón multicapa.*
Fecha de consulta: 22 de septiembre de 2022.
Disponible en
<https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa>

[13] S. Haykin. *Redes Neurais: Princípios e Práctica*, 2001.

[14] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems*, 2019.

[15] E. L. Otero, A. E. Figueroa, T. Fiszon, G. Segesso, Y. Graña y D. Brasi. *Deep Learning, la tecnología del mañana*, 2019.

[16] D. Ballard y C. Brown. *Computer Vision*, 1982.

[17] D. Mery. *Visión por Computador*, 2004.

[18] D. Calvo. *Clasificación de redes neuronales artificiales*.
Fecha de consulta: 6 de septiembre de 2022.
Disponible en <https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/>

[19] Sitiobigdata. *Redes neuronales de aprendizaje profundo*.
Fecha de consulta: 21 de septiembre de 2022.
Disponible en <https://sitiobigdata.com/2019/07/08/redes-neuronales-aprendizaje-profundo/#>

[20] Bootcamp AI. *Redes neuronales*.
Fecha de consulta: 20 de septiembre de 2022.

Disponible en <https://bootcampai.medium.com/redes-neuronales-13349dd1>

[21] KeepCoding. *Capas de pooling en una red neuronal convolucional*. Fecha de consulta: 24 de septiembre de 2022. Disponible en <https://keepcoding.io/blog/capas-pooling-red-neuronal-convolucional/>

[22] Na8. *Convolutional Neural Networks: La Teoría explicada en Español.* Fecha de consulta: 6 de septiembre de 2022. Disponible en <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

[23] V. Rodríguez. *Dropout y Batch Normalization*. Fecha de consulta: 20 de septiembre de 2022. Disponible en <https://vincentblog.xyz/posts/dropout-y-batch-normalization>

[24] Datapeaker. *Alexnet Architecture: Introducción a la arquitectura de Alexnet*. Fecha de consulta: 24 de septiembre de 2022. Disponible en <https://datapeaker.com/big-data/alexnet-architecture-introduccion-a-la-arquitectura-de-alexnet/

[25] R. Rodríguez Abril. *GoogLeNet.* Fecha de consulta: 24 de septiembre de 2022. Disponible en <https://lamaquinaoraculo.com/computacion/googlenet/>

[26] Datacientest. *VGG: ¿Qué es este modelo?* Fecha de consulta: 24 de septiembre de 2022. Disponible en <https://datascientest.com/es/vgg-que-es-este-modelo-daniel-te-lo-cuenta-todo>

[27] S. Tsang. *Review: ResNet — Winner of ILSVRC 2015 (Image Classification, Localization, Detection*. Fecha de consulta: 24 de septiembre de 2022. Disponible en <https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8>

[28] R. Gandhi. *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

[29] A. Fawzy Gad. *Faster R-CNN Explained for Object Detection Tasks.* Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://blog.paperspace.com/faster-r-cnn-explained-object-detection/>

[30] J. Hui. *Understanding Feature Pyramid Networks for object detection (FPN)*. Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c>

[31] J. Hui. *SSD object detection: Single Shot MultiBox Detector for real-time processing*. Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06>

[32] Papers with code. *RetinaNet Explained.* Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://paperswithcode.com/method/retinanet>

[33] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. Fecha de consulta: 8 de septiembre de 2022. Disponible en <http://pjreddie.com/yolo/>

[34] M. Menegaz. *Understanding YOLO*. Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://hackernoon.com/understanding-yolo-f5a74bbc7967>

[35] J. Durán. *Técnicas de Regularización Básicas para Redes Neuronales*. Fecha de consulta: 7 de septiembre de 2022. Disponible en <https://medium.com/metadatos/t%C3%A9cnicas-de-regularizaci%C3%B3n-b%C3%A1sicas-para-redes-neuronales-b48f396924d4>

[36] A. Gandhi. *Data Augmentation | How to use Deep Learning when you have Limited Data*. Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/>

[37] Papers with code. *COCO Dataset*.

Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://paperswithcode.com/dataset/coco>

[38] R. Rodríguez Abril. *Detección de objetos I: YOLO.* Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://lamaquinaoraculo.com/computacion/deteccion-de-objetos/>

[39] M. Merino. *Aprende PyTorch para crear redes neuronales.* Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://www.genbeta.com/desarrollo/asi-puedes-aprender-a-usar-pytorch-herramienta-accesible-para-crear-redes-neuronales>

[40] G. Jocher. *YOLOv5 in PyTorch*. Fecha de consulta: 21 de noviembre de 2022. Disponible en <https://github.com/ultralytics/yolov5>

[41] A. Rosebrock. *Intersection over Union (IoU) for object detection*. Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>

[42] J. Hosang, R. Benenson, and B. Schiele. *Learning non-maximum suppression*, 2017.

[43] Sambasivarao. *Supresión no máxima (NMS).* Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c>

[44] T.Yi Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár y C. Lawrence. *Microsoft COCO: Common Objects in Context*, 2014.

[45] Papers with code. *COCO Dataset.* Fecha de consulta: 8 de septiembre de 2022. Disponible en <https://paperswithcode.com/paper/microsoft-coco-common-objects-in-context>

[46] GeeksforGeeks. *Image Enhancement in PIL*. Fecha de consulta: 25 de noviembre de 2022. Disponible en <https://www.geeksforgeeks.org/image-enhancement-in-pil/>

[47] S. Yohanandan. *mAP (mean Average Precision) might confuse you!* Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>

[48] R. Padilla. *Most popular metrics used to evaluate object detection algorithms*. Fecha de consulta: 25 de septiembre de 2022. Disponible en <https://github.com/rafaelpadilla/Object-Detection-Metrics>

[49] Nvidia. *Installation Guide Windows: CUDA Toolkit Documentation*. Fecha de consulta: 25 de octubre de 2022. Disponible en <https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html>

Apéndices

A.1 Descripción de los archivos entregables

En los archivos entregables de este proyecto se han adjuntado el directorio *Scripts*. En esta carpeta se encuentran los archivos necesarios para realizar las pruebas tanto en consola de comandos como en *Jupyter Notebook*. Entre estos archivos se encuentran:

- La carpeta *original_images* que será la que almacenará las imágenes que analizaremos en la prueba actual.
- La carpeta *Object-Detection-Metrics-Master* que es el proyecto con el que se calcularán las métricas.
- Un archivo a modo de *main* que será el que tendremos que ejecutar tanto si usamos *Jupyter Notebook* (*main.ipynb*), como si usamos la consola (*main.py*).
- Un segundo archivo *functions* que contiene todas las funciones necesarias para transformar las imágenes, realizar las detecciones etc.
- La carpeta *annotations* en la se encuentra el archivo en formato *JSON* con todas las anotaciones del conjunto de validación del *dataset MS COCO 2014* que es el que se ha empleado para realizar el proyecto.

Además de los archivos anteriormente descritos, se ha adjuntado una carpeta con los *datasets* empleados para las pruebas y los archivos con las ground truths correspondientes a cada una de las imágenes. Se han incluido dos ejemplos de ejecución uno con *Jupyter Notebook* y otro con la consola y, por último, un archivo de *excel* con los resultados obtenidos en las pruebas.

A.2 Manual de uso

Las pruebas pueden realizarse tanto por consola, ejecutando el archivo *main* que se encuentra en la carpeta *Scripts*, como inicializando un *notebook* en la carpeta anteriormente nombrada y ejecutando el *book main*.

También se han incluido dos archivos *readme* con los pasos a seguir para poder ejecutar las pruebas con las dos implementaciones, en consola de comandos y usando *Jupyter Notebook*.

A.3 Configuración de la GPU

Para una mayor velocidad de ejecución de las pruebas, se ha aprovechado que la implementación del modelo YOLOv5 está realizada en *PyTorch* por lo que se puede ejecutar utilizando la *GPU*.

La *GPU* empleada ha sido la *NVIDIA GeForce* 1050 Ti con la versión 3.9.0 de *Python*.

Los pasos que se han de seguir para poder hacer uso de la *GPU* son los siguientes, además, se recomienda instalar las versiones indicadas en cada uno de los pasos para evitar incompatibilidades:

1. Se ha de comprobar la compatibilidad de la *GPU* con *CUDA* ya que, si no es compatible, no se podrá realizar esta configuración.
2. Si la *GPU* es compatible, se deberá instalar una versión de *tensorflow* y *tensorflow-gpu* actualizada. Se recomiendan las 2.8.0.cr1 y la 2.8.0 que han sido las utilizadas para este proyecto.
3. Instalar *CUDA*. Se recomienda una versión actualizada para evitar incompatibilidades. En este caso se ha utilizado la versión 11.6.
4. Instalar los paquetes *Python Wheels* que dan soporte a *CUDA*. Para ello se recomienda leer la guía de *CUDA* y el apartado de *Python Wheels* [49].
5. Por último, se debe de instalar la versión de *PyTorch* adecuada. En la página oficial de pytorch se ha de rellenar una tabla con las prestaciones, sistema operativo y versión de *CUDA* instalado. Para este proyecto se ha usado la versión 1.12.0 de *pytorch* compatible con *CUDA* 11.6.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga