UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de un sistema de detección de prendas de ropa en imágenes de moda mediante la implementación de redes YOLOv8 personalizadas

Trabajo Fin de Grado

Grado en Tecnología Digital y Multimedia

AUTOR/A: Parés Marín, Guillermo

Tutor/a: Igual García, Jorge

CURSO ACADÉMICO: 2022/2023

Resumen

El sector de la moda es uno donde el uso de la Inteligencia Artificial es más extendido. En este trabajo se propone el uso de redes neuronales para la detección de diferentes prendas en imágenes de moda. La arquitectura que usará será la familia de algoritmos YOLO, en concreto YOLOv8. El trabajo consiste en la búsqueda de conjunto de datos relacionados, estudiar la familia de algoritmos YOLO, sus diferentes implementaciones y aplicarlo al conjunto de datos encontrados.

YOLO es una tecnología de aprendizaje profundo que nos facilita la detección, segmentación y clasificación de objetos en imágenes y videos en tiempo real. En este proyecto nos centraremos en la tarea de detección, adaptando y entrenando una red neuronal YOLO previamente entrenada a las necesidades para la detección de ropa.

Para ello se entrenará nuevamente el modelo de aprendizaje profundo con un conjunto de datos personalizado que contiene las imágenes y anotaciones correspondientes, con la finalidad de que la red neuronal intérprete que prenda está viendo y en qué ubicación.

Una vez entrenado el modelo, será capaz de detectar prendas de ropa automáticamente en imágenes propias. Además incluirá una parte de “front” para poder interactuar con el usuario mucho más fácilmente.

El trabajo final de grado incluirá una revisión bibliográfica sobre el tema, el diseño e implementación del sistema, pruebas y evaluación de su desempeño, y conclusiones sobre los resultados obtenidos y posibles mejoras futuras.

Palabras Clave: imagen, visión artificial, redes neuronales, inteligencia artificial, aprendizaje profundo, detección

Resum

El sector de la moda és un on l'ús de la intel·ligència artificial és més estès. En aquest treball es proposa l'ús de xarxes neuronals per a la detecció de diferents peces en imatges de moda. L'arquitectura que farà servir serà la família d'algorismes YOLO, en concret YOLOv8. El treball consisteix en la cerca de conjunt de dades relacionades, estudiar la família d'algorismes YOLO, les diferents implementacions i aplicar-lo al conjunt de dades trobades.

YOLO és una tecnologia d'aprenentatge profund que ens facilita la detecció, la segmentació i la classificació d'objectes en imatges i vídeos en temps real. En aquest projecte ens centrarem en la tasca de detecció, adaptant i entrenant una xarxa neuronal YOLO prèviament entrenada a les necessitats per a la detecció de roba.

Per fer-ho, s'entrenarà novament el model d'aprenentatge profund amb un conjunt de dades personalitzat que conté les imatges i anotacions corresponents, amb la finalitat que la xarxa neuronal intèrpret que penyi està veient i en quina ubicació.

Un cop entrenat el model, serà capaç de detectar peces de roba automàticament en imatges pròpies. A més, inclourà una part de “front” per poder interactuar amb l'usuari molt més fàcilment.

El treball final de grau inclourà una revisió bibliogràfica sobre el tema, el disseny i la implementació del sistema, proves i avaluació del seu exercici, i conclusions sobre els resultats obtinguts i possibles millores futures.

Paraules Clau: imatge, visió artificial, xarxes neuronals, intel·ligència artificial, aprenentatge profund, detecció

Abstract

The fashion sector is one where the use of Artificial Intelligence is more widespread. This paper proposes the use of neural networks for the detection of different garments in fashion images. The architecture that it will use will be the YOLO family of algorithms, specifically YOLOv8. The work consists of searching for related data sets, studying the YOLO family of algorithms, its different implementations and applying it to the found data set.

YOLO is a deep learning technology that makes it easy for us to detect, segment, and classify objects in images and videos in real time. In this project we will focus on the detection task, adapting and training a previously trained YOLO neural network to the needs for clothing detection.

To do this, the deep learning model will be trained again with a custom data set that contains the images and corresponding annotations, in order to determine which interpreter neural network is seeing and in what location.

Once the model has been trained, it will be able to automatically detect clothing items in its own images. It will also include a "front" part to be able to interact with the user much more easily.

The final degree project will include a bibliographical review on the subject, the design and implementation of the system, tests and evaluation of its performance, and conclusions on the results obtained and possible future improvements.

Keywords: image, computer vision, neural networks, artificial intelligence, deep learning, detection

Agradecimientos

Lo primero quiero agradecer la ayuda a mi familia, mis padres que me han apoyado en todo momento tanto en el desarrollo del trabajo como en mi paso por la universidad, a mí hermano por ayudarme a organizar los primeros pasos del trabajo y a mi pareja por apoyarme todos los días que he dedicado al trabajo y la universidad. Además a todos los grandes amigos que he hecho a lo largo de estos cuatro años que han hecho más amena y divertida la carrera.

Por supuesto agradecer a mi tutor Jorge Igual García, por ayudarme con todo lo necesario, ser tan cercano a nosotros, brindarnos todos sus conocimientos y enseñarme muchas cosas interesantes que no sabíamos de la carrera.

Muchas gracias a todos.

Índice

Índice de figuras

Índice de ecuaciones

Índice de tablas

Lista de siglas y acrónimos

YOLO: You only Look Once (familia de algoritmos)

DL: Deep Learning

IA: Inteligencia Artificial

RESNET: Residual Networks

RNAS: Redes Neuronales Artificiales

RELU: Rectified Linear Unit

TANH: Tangente hiperbólica

CNN: Convolutional Neural Network

SDG: Stochastic Gradient Descent

SSD: Single Shot Multibox Detector

CPU: Central Processing Unit

GPU: Graphics Processing Unit

mAP: Mean Average Precision

FLOP: Floating-point Operations

MSE: Mean Squared Error

ADAM: Adaptive Moment Estimation

RMSprop: Root Mean Square Propagation

Adagrad: Adaptive Gradient Algorithm

Adadelta: Adaptive Delta

CUDA: Compute Unified Device Architecture

VRAM: Video Random Access Memory

RAM: Random Access Memory

VoTT: Visual Object Tagging tool

CLI: Command-Line Interface

Capítulo 1 - Introducción

La visión por computador, o *computer vision* es uno de los campos más importantes de la inteligencia artificial, dentro de este se encuentran tareas complementarias como la clasificación, la segmentación o la detección de objetos. Nos centraremos en la detección de objetos en imágenes ya que es una tarea fundamental que tiene muchas aplicaciones en diferentes campos como la seguridad, la robótica o la medicina entre otras muchas.

El objetivo principal de la detección de objetos es identificar un objeto específico en una ubicación específica, a diferencia de la clasificación, que simplemente se limita a decir que objetos hay en la imagen, sin aportar más información de donde se encuentra.

Sin embargo, la detección de prendas de ropa es un problema difícil que se presenta en el este mundo, ya que hay gran variedad de prendas de ropa, con diferentes colores, texturas, formas, estampados y hasta incluso tamaños.

Esta tecnología puede aplicarse en diferentes prácticas como detectar fallos en la producción de la ropa para optimizar el proceso de optimización, aumentar la seguridad en espacios públicos o por ejemplo, poder identificar a una persona al saber el tipo de ropa que está vistiendo.

Es por eso por lo que el desarrollo de este sistema mediante la implementación de redes YOLO puede tener grandes aplicaciones y se hace con una intención de en un futuro ser capaz de ser implementado en una posible aplicación o web como la que se ha desarrollado para probar e interactuar con el modelo de una manera sencilla y rápida.

1.1 Motivación

Siempre me han llamado la atención las tecnologías y todas sus capacidades, sin embargo, con todo el revuelo y toda la atención que están teniendo hoy las inteligencias artificiales se han dado a conocer y se puede tener mucha más información sobre ellas. Cuando en mi tercer curso del grado cursé la asignatura de Imagen Digital vimos por primera vez temario relacionado con el mundo del Machine Learning, sobre Computer Vision para ser más exactos.

Me resultó bastante interesante, con una proyección de futuro bastante y con unas aplicaciones muy interesantes en el mundo de la actualidad.

Poco después viví la experiencia Erasmus en Siena, Italia, donde escogí unas asignaturas relacionadas con el mundo de la inteligencia artificial y el Deep Learning, donde despertó todo mi interés sobre este campo tan inmenso y con tanto potencial.

Personalmente me gustaría poder trabajar o seguir formándome en este campo, ya que considero que las inteligencias artificiales que están surgiendo hoy en día están siendo una de las mayores revoluciones tecnológicas de la historia.

Además de una motivación personal, el desarrollo de este trabajo motiva a aprender y entender cómo funcionan las computadoras para que puedan identificar y reconocer

objetos en imágenes, ver las distintas aplicaciones prácticas que tiene y contribuir a la comunidad de YOLO.

1.2 Objetivos

Este trabajo consta con varios objetivos principales, el primero de ellos es el entrenamiento de una red neuronal personalizada mediante la familia de algoritmos YOLO, con la finalidad de detectar prendas de ropa específicas en el conjunto de datos.

A la vez se evaluará el rendimiento de la red personalizada mediante métricas de desempeño como la precisión y tiempo de procesado para determinar con mayor claridad su velocidad y sobre todo su eficiencia.

Es importante tener un modelo que sea fiable y pueda identificar características específicas de una prenda de ropa, también nos sirve para analizar la efectividad y rendimiento de la inteligencia artificial y su respectiva detección al ser datos con condiciones lumínicas diferentes.

Además de un estudio sobre las redes YOLO y su familia de algoritmos, como entrenar un modelo de inteligencia artificial con un conjunto de datos personalizado, etiquetado manualmente y con imágenes tanto de consumidores como de proveedores.

Después del estudio y el entrenamiento del modelo personalizado se desarrollará una pequeña parte de *front* donde se implementará el modelo previamente entrenado, con la finalidad de tener una aplicación mucho más intuitiva y fácil de manejar, teniendo simplemente que cargar la imagen que queramos y esperar los resultados.

Finalmente, el objetivo también es probar y validar el modelo personalizado, haciendo una serie de pruebas y predicciones con un subconjunto de datos dedicado específicamente para el apartado del *testing.*

Capítulo 2 – Estudio previo

Lo primero a realizar antes de desarrollar un proyecto o un trabajo de investigación es informarse y tener un estudio previo sobre los conocimientos que son necesarios para comprender el objetivo y el contenido de dicho proyecto.

La inteligencia artificial es un campo inmenso, con muchas aplicaciones y objetivos, que necesita algo de estudio previo para comprenderlo con totalidad.

2.1 Estado de la detección de objetos

La detección de objetos siempre ha estado en constante evolución, sin embargo, con el auge de las inteligencias artificiales se ha comenzado a saber más y a tener un interés más grande. Uno de los mayores ejemplos que podemos conocer a la fecha de hoy es el desarrollo de *ChatGPT*, y aunque no trate de detección de objetos como este trabajo menciona, nos sirve para darnos cuenta lo útiles que pueden llegar a ser, el gran futuro que tienen y el impacto que pueden provocar en nuestra vida cotidiana.

La detección de objetos es uno de los puntos fuertes de la inteligencia artificial, y aunque se pueden desarrollar redes neuronales y redes neuronales convolucionales desde cero, gracias a librerías como *Pytorch* o *Tensorflow*, existe una empresa que se dedica a desarrollar herramientas y soluciones para la detección de objetos.

Esta empresa es *Ultralytics*, principalmente conocida por el desarrollo e implementación de la arquitectura y familia de algoritmos *YOLO (You Only Look Once),* una herramienta que en la actualidad es de la más populares en el mundo de la visión computadora, permitiéndonos detectar objetos con alta precisión en imágenes y videos en tiempo real.

Ultralytics está en constante desarrollo, contando con varias versiones de YOLO. El desarrollo e incremento es tan grande que en apenas un año han actualizado estos algoritmos de la versión cinco a la ocho, siendo esta última la más novedosa y eficaz.

Aunque no solo esta empresa es el centro del mundo, hay muchas más arquitecturas que han recibido grandes avances recientemente, una de las más conocidas en el mundo del *Deep Learning* es *ResNet*. Esta arquitectura está desarrollada por *Microsoft Research* en 2015, ha sido muy utilizada en el mundo del *computer vision* ya que implementó una novedad que permitía a las neuronas de la red aprender a un nivel más profundo y eficaz sin sufrir el problema de desaparición de gradiente..

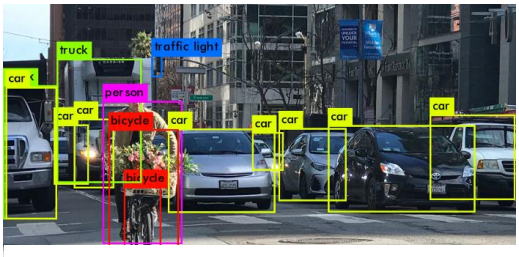

Figura 1. Detección de objetos
https://blog.330ohms.com/2020/11/17/deteccion-de-objetos-con-yolo/

2.2 Marco teórico

El mundo de Deep Learning es un mundo algo complejo sin unos conocimientos previos, en la detección de objetos se utilizan muchos conceptos que pueden sonar extraños. Pero haremos un repaso y un estudio sobre las técnicas que se utilizan para desarrollar estos sistemas con éxito.

2.3 Tipos de aprendizajes

En el campo del Machine Learning existen tres tipos de aprendizaje que se diferencian entre sí por la forma de entrenar al modelo deseado y por el conjunto de datos que se utiliza.

El aprendizaje supervisado generalmente es el más utilizado hoy en día en este campo ya que permite abordar y solucionar gran cantidad de problemas eficazmente gracias a su precisión. Consiste en enseñar al algoritmo como llevar a cabo la tarea de entrenamiento gracias a un conjunto de datos que contiene ejemplos de entrada con sus salidas etiquetadas respectivamente. En este proyecto se lleva a cabo un entrenamiento del algoritmo mediante la técnica de aprendizaje supervisado, ya que contamos con un conjunto de datos que se compone de imágenes y etiquetas, donde indica la prenda de ropa y la ubicación en la imagen que queremos que el modelo aprenda a reconocer y detectar. Esta técnica se lleva a cabo gracias a la extracción de patrones y características indicadas en los datos de entrenamiento.

Por otra parte, también existe el aprendizaje no supervisado, siendo totalmente lo contrario del anterior ejemplo. En esta técnica no se proporciona ningún tipo de datos etiquetados o indicativos que ayuden al algoritmo de entrenamiento a saber la salida deseada. Estos modelos logran su objetivo ya que aprenden a buscar y detectar patrones y estructuras en los datos por sí mismos. Los algoritmos entrenados con este método no suelen utilizarse en la detección de objetos, esta técnica se utiliza principalmente para descubrir información oculta en los datos entre otras muchas aplicaciones.

Por último, se encuentra el aprendizaje por refuerzo. Es un tipo de aprendizaje que no tiene capacitación de datos etiquetados o no etiquetados, en este caso, el algoritmo es capaz de aprender y ser entrenado en un entorno donde no existe ninguna información sobre la salida, esto lo hace gracias a las acciones que se van tomando en mitad del proceso.

2.4 Redes neuronales

Las redes neuronales, también conocidas como redes neuronales artificiales, RNAs, son uno de los métodos más importantes del aprendizaje profundo, consiste en procesar datos de una manera inspirada en el cerebro humano. El principal objetivo de las RNAs es el aprendizaje automático a partir de datos, siendo una herramienta muy importante y útil

en el reconocimiento de imágenes, detección de patrones, predicción de valores numéricos lineales y no lineales y el procesamiento del lenguaje natural.

Gracias a su arquitectura y su capacidad son capaces de procesar grandes cantidades de datos realizando tareas complejas simultáneamente y en tiempos relativamente reducidos. Una red neuronal está formada por neuronas, creando un sistema adaptable que permite a un computador aprender de sus errores y mejorar continuamente. Cada neurona es una unidad independiente que recibe información, esta información es procesada siendo sometida a varias operaciones, produciendo un nuevo valor de salida, que posteriormente será la entrada de una o más neuronas artificiales.

Las redes neuronales están formadas por una serie de unidades procesadoras elementales, lo que conocemos como neurona al igual que en el sistema nervioso humano. Una red neuronal está compuesta por tres capas, la capa de entrada, la capa oculta y la capa de salida. Como bien dice, dentro de estas capas lo que encontraremos serán unidades neuronales que sirven para el procesado de información.

Una vez la información ha pasado por las dos primeras capas y ya ha sido procesada, lo siguiente es mandarla a la capa de salida donde junto con los pesos, el sesgo y la información previamente obtenida se obtienen los resultados.

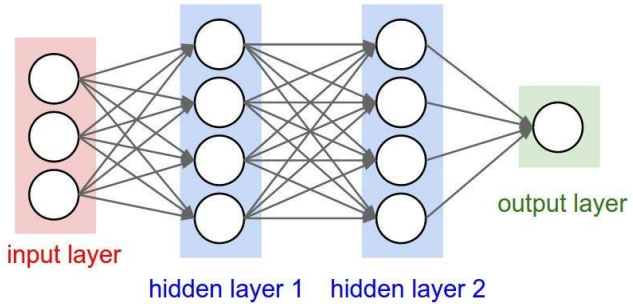

Figura 2. Red neuronal y sus capas
https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

2.4.1 Arquitectura de las RNAs

La arquitectura de una red neuronal depende de su estructura, teniendo en cuenta diferentes factores como el número de capas, el número de neuronas y el tipo, y por último el tipo de conexiones entre ellas.

La estructura siempre suele ser similar, las redes neuronales artificiales constan de tres capas muy importantes, explicadas previamente por cada neurona:

- Capa de entrada: en esta capa se reciben los datos de entrada y suele estar formada por pocas neuronas, dependiendo de la complejidad del problema y el tamaño de los datos de entrada. Las neuronas de esta capa tienen un peso asignado que se utiliza para determinar la fuerza y dirección de la señal que se transmite de una neurona a otra.

- Capas ocultas: estas capas son esenciales, ya que procesan la información mediante pesos, sesgos y funciones de activación. El número de capas y neuronas suele ser bastante grande, dependiendo de la complejidad del problema.

- Capa de salida: es la última capa de la red y proporciona finalmente el resultado o salida de la red, el número de neuronas depende del tipo de problema y el número de tipos, clases o salidas que tenga, Por ejemplo, en este proyecto tratamos con trece diferentes clases, por lo tanto la red neuronal consta de trece neuronas de salida distintas, donde cada una tendrá un valor de predicción previamente calculado en las neuronas ocultas En esta capa, la mayoría de las neuronas suelen utilizar una función de activación llamada softmax, que consiste en producir resultados que representan distribuciones de probabilidad.

Las RNAs pueden ser de distintos tipos siendo:

- Redes Neuronales Feedforward: es la red más básica y consta de una serie de capas que solo trabaja en una dirección. Está formado por la capa de entrada, una o más capas ocultas y por último la capa de salida. Se suele utilizar en problemas de clasificación simples, regresión y predicción.

- Redes Neuronales Recurrentes: estas redes tienen una peculiaridad y es que su estructura está compuesta por ciclos que permite mantener información de estados anteriores y utilizarla para procesar información futura y actual. Se utilizan mucho en problemas de procesamiento de lenguaje natural.

- Redes Neuronales Convolucionales: están diseñadas específicamente para trabajar con señales de video e imagen. Son las redes que utilizaremos para el desarrollo de este proyecto. Contienen capas convolucionales que son capaces de reconocer patrones en imágenes y videos. Es la principal herramienta de la visión artificial, detección, clasificación y segmentación de objetos.

2.4.2 Funciones de activación

La detección de objetos implica identificar y localizar la presencia de objetos en imágenes o videos, para lograr esto utilizamos, como bien hemos visto previamente, redes neuronales convolucionales. En esta arquitectura y en las otras se encuentra una técnica fundamental, las funciones de activación.

Las funciones de activación son una función matemática aplicada a la salida de una neurona introduciendo no linealidad en la red, lo que permite solucionar problemas mucho más complejos. En el mundo de la visión artificial y la detección de objetos son muy importantes ya que pueden determinar y establecer el umbral de activación para detectar un objeto.

Entre muchas funciones, la función *ReLu (Rectified Linear Unit)* es una función no lineal que se define como:

Ecuación 1 ReLU

$$f(x) = max(0, x)\ .$$

De esta manera lo que se consigue es producir una activación positiva cuando el valor de entrada de la neurona correspondiente sea mayor que cero, o cero cuando sea menor.

Esta función destaca más que las demás ya que es realmente útil a la hora de detectar objetos, al eliminar las activaciones negativas conseguimos una reducción de ruido y resaltar características relevantes en los objetos detectados.

Sin embargo, existen múltiples funciones de activaciones, aunque las más comunes son:

1. **Tangente hiperbólica (Tanh):** es no lineal y toma un valor de entrada y produce otro de salida en un rango (-1, 1). Es utilizada en diferentes partes de la red neuronal como por ejemplo en las capas ocultas ya que aporta no linealidad y capacidad de capturar patrones complejos en los datos.

Ecuación 2 Tanh

$$tanh\,(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$$

2. **Sigmoide:** Es similar a la tangente hiperbólica, pero esta toma valores en un rango de (0,1). Suele ser utilizada en redes neuronales para problemas de clasificación binaria ya que puede ser interpretada como una probabilidad, por ello, se puede asignar una probabilidad a la clase positiva (clase 1) u otra a la clase negativa (clase 0).

Ecuación 3. Sigmoide

$$sigmoid(x) = \frac{1}{(1 + e^{-x})}$$

3. **Softmax:** La función de activación softmax es una de las funciones más usadas. Se aplica en el final de la red neuronal, la salida de las neuronas, y se utiliza mucho en problemas de clasificación multiclase. Transforma los valores de salida en una distribución de probabilidad donde las salidas de las neuronas de la última capa suman entre todas ellas uno. Dejando, así una neurona ganadora, que será la encargada de dictar el resultado.

Ecuación 4. Softmax

$$softmax(x_i) = \frac{e^{x_i}}{sum(e^{x_j})}$$

2.4.3 Redes Neuronales Convolucionales

La evolución y progreso del Deep Learning ha conllevado múltiples avances en el mundo de la inteligencia artificial, entre ellos la capacidad de procesar modelos computacionales compuestos por capas convolucionales. Modelos que han sido diseñados especialmente para el procesado de imágenes, videos y audios. Las CNN han demostrado un gran rendimiento en las tareas de detección de objetos gracias a las características de sus filtros convolucionales capaces de capturar diferentes patrones y características visuales.

Las capas convolucionales están formadas por filtros convolucionales que a su vez se componen de pesos que se van actualizando durante el entrenamiento de la red.

Por otro lado, en las redes neuronales convolucionales existen las capas de *pooling*, son unas capas que aplican una operación común en las CNN para reducir la información y los mapas de características de una imagen. Al reducir la dimensionalidad de los mapas de características, el pooling permite que la red neuronal sea más eficiente y robusta a la hora de la detección.

Una de las principales ventajas de las redes neuronales convolucionales es su capacidad para capturar características invariantes de traducción en los datos de entrada. Esto significa que la misma característica se puede detectar en diferentes regiones de la imagen, lo que permite una fácil generalización a nuevas instancias. Además, las CNN tienen la capacidad de aprender automáticamente jerarquías de características, donde las capas anteriores aprenden características simples (como bordes y texturas) y las capas posteriores aprenden características más complejas (como formas y objetos).

Las redes neuronales convolucionales se entrenan mediante la optimización de una función de pérdida utilizando algoritmos de aprendizaje como el descenso de gradiente estocástico (SGD) y la retropropagación de errores. Las CNN aprenden en función de la actualización de pesos y filtros convolucionales para minimizar la diferencia entre los resultados previstos y esperados.

La aparición y el desarrollo de las redes neuronales convolucionales han llevado un gran desarrollo en el campo de la detección de objetos, como el desarrollo de redes neuronales como *Faster R-CNN, Single Shot Multibox Detector (SSD)* y una de las más populares y usadas, *You Only Look Once (YOLO).*

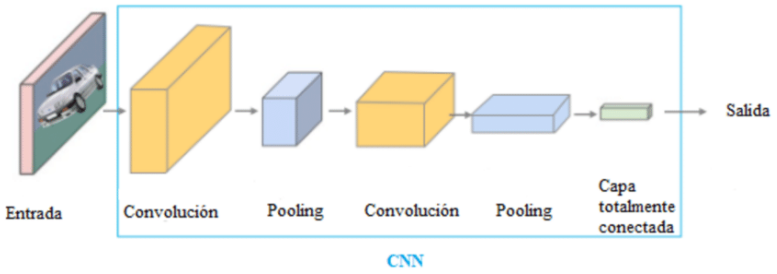

Figura 3. CNN https://www.researchgate.net/figure/Figura-1-Descripcion-del-funcionamiento-de-una-red-neuronal-convolucional-CNN-10_fig1_348825166

2.4.4 Visión por computadora

El concepto de visión por computadora, más famosamente conocido como *computer vision*, se centra en el aprendizaje e interpretación de las máquinas sobre las imágenes para poder procesar, analizar y comprender imágenes o videos para realizar ciertas tareas.

A lo largo de los años este campo ha ido progresando significativamente a lo largo de los años gracias a avances en el hardware y software, además de la creación y evolución de muchos algoritmos de Deep Learning y un gran abanico de conjuntos de datos, dando múltiples posibilidades. El computer visión es un campo que va de la mano del aprendizaje automático, ya que juntos se complementan idealmente para adquirir conocimientos a partir de datos y ser capaces de realizar ciertas tareas con un gran éxito. El procesamiento digital de imágenes consiste en producir una versión modificada de una imagen escogida para poder trabajar sobre ella. Implica la corrección de distorsiones, iluminación y ruido, gracias a esta técnica poder obtener una mejor calidad en las imágenes que queramos extraer las características importantes necesarias para el entrenamiento y del algoritmo de aprendizaje profundo.

A pesar de que el objetivo de este proyecto es aplicar la visión por computadora al mundo de la detección de objetos, también existen varias aplicaciones como: la vigilancia, la realidad aumentada, el seguimiento y segmentación de objetos, la clasificación de imágenes y el reconocimiento facial.

2.5 YOLO (You Only Look Once)

You Only Look Once es un algoritmo muy conocido y usado en el campo de la detección de objetos en vídeos y sobre todo en imágenes. Joseph Redmon y su equipo lo desarrollaron con un enfoque de visión por computadora que hoy en día se considera uno de los más importantes gracias a su efectividad y rapidez a la hora del procesado de información.

El funcionamiento de YOLO no es muy complejo a simple vista, ya que lo que el usuario ve es una interfaz en la que el modelo de aprendizaje profundo predice un resultado y genera unas cajas conocidas como *"bounding boxes"* que identifican la posición y el tamaño del objeto detectado, además de indicar las probabilidades predichas del objeto. Sin embargo, lo que el usuario no ve es la arquitectura y el trabajo del modelo. YOLO utiliza una red neuronal convolucional profunda con la que extrae las características de la imagen, gracias a sus procesos de convolución que se encuentran en sus múltiples capas, estas características es información útil de una imagen como por ejemplo formas, texturas o bordes.

Esto se lleva a cabo gracias a que cada conjunto de filtros se encarga de aprender a detectar alguna característica en especial, y mediante un proceso de múltiples multiplicaciones y sumas ponderadas en ciertas regiones conocidas como ventanas locales, que una vez estos conjuntos de filtros han acabado su trabajo se envía información a las capas de *pooling* donde se reduce la dimensionalidad espacial de las características extraídas y a la vez se mantiene la información más relevante, convirtiendo el modelo en una tecnología más veloz y robusta antes variaciones.

Por último, las capas finales son las encargadas de realizar la clasificación basada en las características que han ido recibiendo a lo largo del funcionamiento.

A lo largo de los años esta tecnología ha ido avanzando y mejorando su funcionamiento, desde 2016 que salió la primera versión mejorada, YOLOv2, hasta principios de 2022 con su última y mejor versión YOLOv8. YOLOv5 fue la versión más popular y utilizada hasta la fecha, ya que aportó grandes mejoras como el rendimiento y la optimización y aportó nuevas funciones como el seguimiento de experimentos integrados y exportación automática a formatos de exportación popular, además de su gran efectividad en el campo del *computer vision*. Sin embargo, a pesar de este proyecto será desarrollado con su última versión hasta la fecha para aprovechar toda la documentación importante y mejoras que ha ido recibiendo a lo largo de la historia.

2.5.1 Modelos YOLO

Una de las cosas que hacen que YOLO sea tan exitoso es la variedad de modelos de *Deep Learning* que nos ofrece, siendo más efectivo y adecuado dependiendo de la dificultad del problema presentado.

Los modelos junto a sus características previamente entrenados y validados en el conjunto de datos de COCOval2017, que nos ofrece este algoritmo son:

MODEL	SIZE (pixels)	mAP	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.52	68.2	257.8

Tabla 1. Características de los modelos YOLO

Aunque todos trabajan con imágenes de tamaño 640 píxeles, como podemos ver en la tabla, los diferentes modelos tienen múltiples características mejor y peor dependiendo el modelo. Como por ejemplo el mAP que se refiere al *mean average precision*, que conocemos como precisión media promedio calculada en el conjunto de datos de validación. También tienen en cuenta la velocidad con la que tarda en llevar a cabo la detección de un objeto utilizando dos factores.

El primer dato hace referencia al tiempo en milisegundos para la configuración que cuenta con el uso del CPU y el formato ONNX cuyas siglas hacen referencia a *Open Neural Network Exchange*, un ecosistema abierto que permite a desarrolladores de IA elegir herramientas adecuadas para su proyecto, proporcionando la facilidad de integrar y transferir los modelos entrenados entre diferentes frameworks del mundo del aprendizaje automático, como por ejemplo TensorFlow, MXNet, Caffe o la utilizada en este proyecto, PyTorch.

La segunda métrica es el tiempo, también en milisegundos, en que YOLO utilizando una GPU A100 y el framework de optimización TesnorRT tarda en procesar una detección. La GPU procesa imágenes mucho más rápido y mejor que la CPU ya que tiene múltiples características que optimizan el modelo y ofrecen una ejecución más rápida y eficaz ante el trabajo de la CPU.

La columna *params (M)* es el número de parámetros que contiene cada modelo, medido en millones. Cada parámetro es un valor ajustable en la red que se aprende durante el entrenamiento, esto varía dependiendo el modelo y por lo tanto su arquitectura. Aunque conforme aumentan los parámetros más recursos requiere el proceso de entrenamiento, se espera que a mayor número de parámetros que contenga un modelo, mayor sea su efectividad resolviendo un problema.

Por último, los *FLOP (floating-point operations)* es la cantidad de operación que realiza el modelo durante el trabajo expresados en billones. Un gran número de FLOPs conlleva un mayor tiempo de ejecución y mayor carga computacional ya que requiere más recursos.

Habiendo visto ya las características de cada modelo, podemos deducir la funcionalidad y efectividad de cada modelo para nuestro problema y así encontrar cual es el más adecuado. En base a su arquitectura desde el más pequeño al más grande:

1. **YOLOv8n (nano):** Es el modelo que consta con la estructura más pequeña, al estar formado por menos capas y parámetros está enfocado a problemas donde se busca obtener una mayor velocidad de detección ante la precisión.

2. **YOLOv8s (small):** Es una versión algo más grande, suele ser el modelo base ya que ofrece un equilibrio entre precisión y velocidad.

3. **YOLOv8m (medium):** Con su estructura y número de capas más grande que los anteriores proporciona mayor precisión aunque sea un poco más lento, posicionando esta versión en el punto intermedio del algoritmo YOLO.

4. **YOLOv8l (large):** Es una de las dos versiones más grandes de YOLO, constando con un gran número de capas y parámetros, aunque comienza a ser algo más lento entrenando y detectando objetos es bastante preciso.

5. **YOLOv8x (extra large)**: Es el modelo más grande con el que consta YOLO, aunque sea pesado y tenga una velocidad bastante reducida otorga la mayor precisión posible a la hora de detectar objetos.

Figura 4. Modelos YOLO

2.5.2 ¿Cómo funciona YOLO?

El popular algoritmo de detección de objetos es algo diferente a los tradicionales algoritmos que estamos acostumbrados a ver, YOLO en lugar de dividir esta tarea en múltiples partes o etapas, lo hace en una sola pasada. Haciendo honor a su nombre You Only Look Once, ya que solo le hace falta ver los datos en una simple pasada porque canaliza todos los componentes de detección en una sola red neuronal.

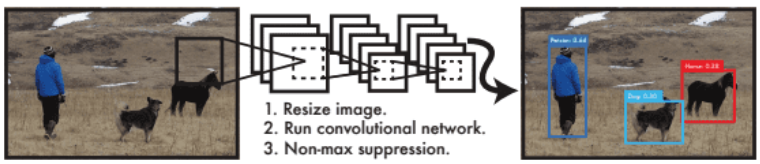

Figura 5. Proceso de detección
https://pyimagesearch.com/2022/04/11/understanding-a-real-time-object-detection-network-you-only-look-once-yolov1/

La figura 5 representa de una manera breve y sencilla el proceso que sigue YOLO para detectar objetos, que consiste en coger y redimensionar la imagen al tamaño solicitado, ejecutar la única red neuronal convolucional en toda la imagen y establecer un umbral de detección para eliminar detecciones no deseadas. Es por este proceso tan unificado que YOLO entrena y detecta a una velocidad bastante superior que los demás algoritmos de aprendizaje profundo.

También en el proceso de detección se divide la imagen en una cuadrícula de celdas, cada cual es encargada de predecir ciertos objetos que se encuentran dentro de esta.

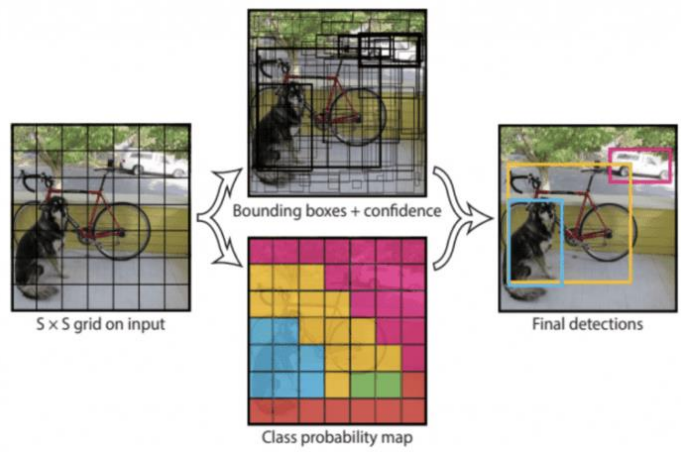

Figura 6. Proceso de detección unificado https://pyimagesearch.com/2022/04/11/understanding-a-real-time-object-detection-network-you-only-look-once-yolov1/

En esta etapa conocida como proceso de detección unificado cada celda predice un número de bounding boxes que tengan posibilidad de contener algún objeto detectado. Posteriormente se calcula las coordenadas de cada caja delimitadora y las probabilidades que tiene cada una para representar alguna clase de objeto. Posteriormente se aplica una técnica que se conoce como supresión de no máximos, que se encarga de eliminar bounding boxes duplicadas y mantener solo las más confiables y que mejor probabilidad y precisión tengan.

El conjunto de todas estas técnicas como dividir la imagen en celdas, predecir las bounding boxes junto a sus probabilidades y suprimir máximos no deseados permiten a YOLO obtener sus detecciones finales de una manera rápida y precisa.

2.5.3 Arquitectura de YOLO

La arquitectura de la red de YOLO es similar a una red neuronal de clasificación como las demás, pero en este caso está inspirada en el modelo GoogLeNet, una CNN desarrollada por Google en 2014, pero utilizando la tarea de clasificación y detección de imágenes.

Está compuesta principalmente por tres tipos de capas diferentes, las capas convolucionales, las capas de *maxpool* y las capas totalmente conectadas.

La arquitectura principal de YOLO contiene 24 capas convolucionales encargadas de extraer características de la imagen, seguidas de dos capas totalmente conectadas para predecir las coordenadas de las bounding boxes y las puntuaciones de detección.

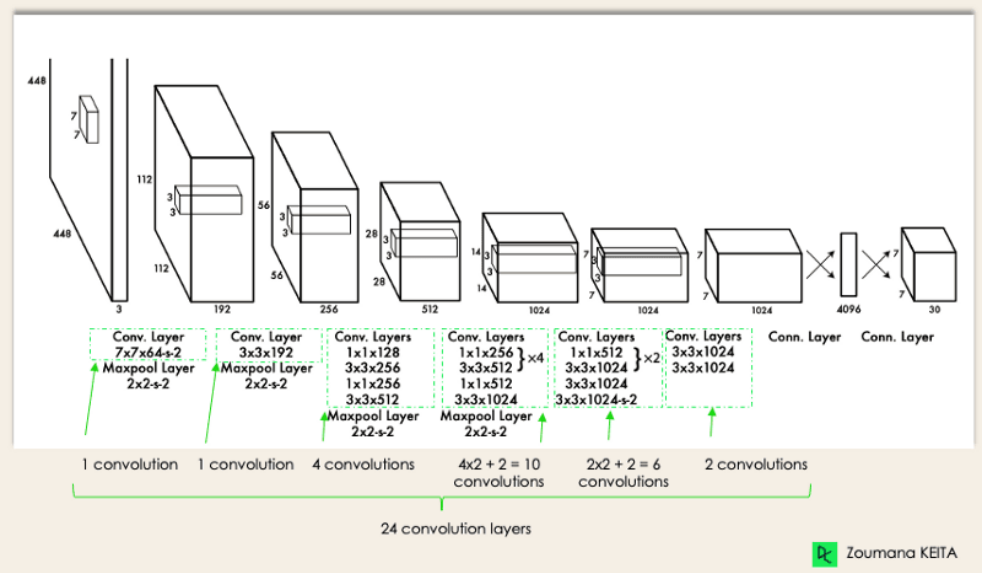

Figura 7. Arquitectura YOLO del proyecto original
https://www.datacamp.com/blog/yolo-object-detection-explained

Esta red es una modificación de la GoogLeNet, primeramente, cambia el tamaño de la imagen de entrada a 448x448 antes de pasar los datos por la red convolucional. Posteriormente utiliza las capas convolucionales de 1 x 1 en lugar de los módulos de inicio para reducir el espacio de las características y seguidamente una capa convolucional de 3 x 3 encargada de generar la salida. Todo bajo la función de activación ReLu a excepción de la última capa que utiliza una función de activación lineal.

2.6 Función de pérdida

En el contexto de la inteligencia artificial y concretamente en las redes neuronales, existen unas funciones matemáticas conocidas como funciones de pérdida, o popularmente en inglés como: *"loss function"*. Estas funciones se utilizan para evaluar el error entre las predicciones realizadas por el modelo y los valores reales o esperados del problema que se está abordando. Son muy importantes en los modelos de aprendizaje y concretamente el proceso de entrenamiento de una red neuronal ya que al medir dicho error son capaces de ajustar durante el mismo proceso los pesos de la red para obtener los resultados esperados.

Dependiendo del tipo de problema que afrontemos podemos escoger una u otra función, entre muchas existen algunas que son las más usadas y populares.

1. **Error cuadrático medio (MSE):** Esta función es comúnmente utilizada en problemas de regresión donde el objetivo es buscar y predecir un valor numérico entero. El objetivo es minimizar esta función para obtener una varianza mínima entre el valor real y el predicho.

2. **Binary cross entropy:** esta función también es muy popular en el deep learning, pero en este caso, esta versión de la entropía cruzada es muy utilizada para los problemas de clasificación binarios, es decir, problemas donde hay 2 clases. Compara las probabilidades predichas por el modelo con la probabilidad real.

3. **Categorical Cross-Entropy:** es muy similar a la anterior, tienen un funcionamiento semejante, pero se aplica en problemas de clasificación donde hay más de dos clases.

4. **Softmax Cross-Entropy Loss:** es una función variante de la mencionada anteriormente, sin embargo, esta función aplica una suavización a las etiquetas reales para evitar que predominen las clases dominantes.

2.7 Back-propagation

Uno de los algoritmos clave en el deep learning es el backpropagation, conocido como retropropagación en español que permite entrenar las redes de manera iterativa ajustando sus pesos y sesgos. El backpropagation fue una revolución en el mundo de la inteligencia artificial ya que aportó una técnica esencial para el entrenamiento de las redes neuronales.

Este algoritmo se compone de dos etapas, la primera que se conoce como *forward pass* propagación hacia delante y la segunda que es el contrario, *backward pass*, propagación hacia atrás.

En la primera fase los datos atraviesan la red neuronal capa por capa donde en cada capa se realiza una operación matemática aplicando pesos y sesgos. Una vez que se ha completado la propagación hacia adelante y se ha obtenido una salida de la red, se utiliza una función de pérdida para calcular la diferencia entre la salida predicha y el valor objetivo. Esta función mide el error entre el valor real deseado y la salida predicha.

En la segunda fase, la propagación hacia atrás, se busca calcular el gradiente de la función de pérdida con los pesos y sesgos que se han ido asignando a la red. El objetivo es propagar el error desde la última capa, la de salida, hacia la primera capa pasando por todas y cada una de las capas de la red, calculando continuamente el gradiente en estas capas. Para este proceso se utiliza la regla de la cadena.

Los gradientes se calculan y luego se utilizan para actualizar los pesos y sesgos de la red con un algoritmo de optimización como el descenso del gradiente estocástico (SGD). Para

reducir la función de pérdida, el algoritmo de optimización ajusta los parámetros en la dirección opuesta al gradiente.

En cada iteración de entrenamiento, se repite el proceso de propagación hacia adelante y hacia atrás, donde los datos de entrenamiento se pasan a través de la red y los gradientes se calculan y utilizan para actualizar los pesos y sesgos. Hasta que la red haya alcanzado una etapa de convergencia y haya reducido la función de pérdida lo suficiente, este proceso continúa.

Este algoritmo ha demostrado ser fundamental en el aprendizaje profundo, sobre todo en problemas y tareas de clasificación de imágenes.

2.8 Funciones de optimización

Los algoritmos de optimización en el mundo de la inteligencia artificial se utilizan para ajustar los parámetros de un modelo para minimizar una función de pérdida o maximizar una función de utilidad. Dado que determinan cómo se actualizan los parámetros en cada iteración para mejorar el rendimiento del modelo, estas funciones de optimización son esenciales para el proceso de entrenamiento de un modelo.

El aprendizaje automático utiliza una variedad de funciones de optimización, cada una de las cuales tiene sus propias características y métodos de actualización de parámetros. Algunas de las características de optimización más comunes son:

1. **Descenso de gradiente:** es la función de optimización más popular en el aprendizaje automático, su objetivo es minimizar la función de pérdida ajustando los parámetros del modelo en la dirección opuesta del gradiente de dicha función. El descenso de gradiente puede tener variantes como *Momentum* y *SGD (descenso de gradiente estocástico)*.

2. **Descenso de gradiente estocástico:** El descenso de gradiente estocástico (SGD) es un algoritmo de optimización que se utiliza para entrenar redes neuronales. El SGD utiliza minilotes aleatorios de ejemplos en lugar de todo el conjunto de datos, a diferencia del descenso de gradiente tradicional. Esto aumenta la productividad y ayuda a evitar los mínimos locales. Sin embargo, debido a la variabilidad de los gradientes estimados, puede tener una convergencia más ruidosa.

3. **ADAM (Adaptive Moment Estimation):** Es un algoritmo de optimización que combina el descenso de gradiente con momento y el descenso de gradiente estocástico. Adam adapta la tasa de aprendizaje adaptativa en función del momento y las estimaciones del segundo momento de los gradientes anteriores.

4. **RMSprop (Root Mean Square Propagation):** Es un algoritmo de optimización que utiliza una tasa de aprendizaje adaptativa que se basa en la magnitud de los gradientes más recientes. RMSprop ajusta la tasa de aprendizaje para cada parámetro.

5. **Adagrad (Adaptive Gradient Algorithm):** Adagrad es un algoritmo de optimización adaptable que adapta la tasa de aprendizaje en función de las frecuencias de actualización de los parámetros. Adagrad disminuye la tasa de aprendizaje para parámetros que se actualizan con frecuencia y viceversa.

6. **Adadelta (Adaptive Delta):** Adadelta, también conocido como Adaptive Delta, es un algoritmo de optimización que se asemeja an Adagrad, pero aborda su limitación de reducir continuamente la tasa de aprendizaje. Adadelta utiliza estimaciones de primer y segundo momento para ajustar de manera más efectiva la tasa de aprendizaje.

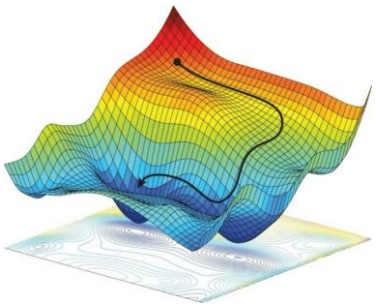

Figura 8. SDG
https://www.researchgate.net/figure/Figura-3511-Descenso-de-gradiente-estocastico-SGD-Arizan-R-y-Hassibi-B-2019_fig9_344388136

2.9 Overfitting

Uno de los problemas más comunes en el Deep Learning y otras técnicas de aprendizaje automático es el overfitting. Esto ocurre cuando el algoritmo de machine learning se entrena demasiado con los datos de entrenamiento, y pierde la capacidad de, en nuestro caso, detectar datos nuevos. El modelo pasa de aprender a detectar objetos a memorizar que objetos pertenecen al conjunto de datos.

Cuando un algoritmo de DL esta sobre ajustado, es muy sensible a los detalles de las imágenes que encontramos en nuestro conjunto de datos y memoriza ciertos patrones específicos.

Este problema puede suceder por causas como: utilizar un conjunto de datos demasiado pequeño o que insuficiente para que el modelo sea capaz de aprender los patrones y detectar objetos en el caso de la detección de objetos, una mala elección de parámetros como el tamaño del batch, el número de capas o el learning rate.

Sin embargo, aunque es un problema bastante común, existen varias maneras de evitarlo como las técnicas de regularización, validación cruzada, eliminación de características irrelevantes, uso de arquitecturas de modelos más simples y búsqueda sistemática de hiperparámetros óptimos para reducir el overfitting en el aprendizaje profundo.

En el proceso de entrenamiento de un modelo de aprendizaje profundo es importante crear un modelo que generalice bien a nuevos datos y crucial encontrar un equilibrio entre la capacidad del modelo y la cantidad de datos de entrenamiento disponibles.

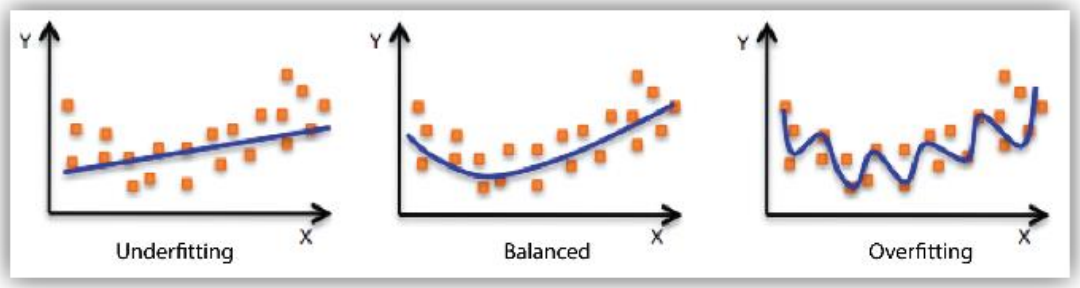

Figura 9. Comparación de ajuste del modelo. https://docs.aws.amazon.com/es_es/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

2.10 Transfer Learning

El transfer learning es una técnica muy popular y usada en el aprendizaje profundo y sobre todo en el contexto de la detección de objetos con la tecnología YOLO (You Only Look Once). Esta técnica nos permite aprovechar todo el conocimiento de un modelo entrenado previamente para acelerar y mejorar el desempeño de un nuevo modelo, gracias a que se utilizan los pesos ya ajustados y calculados del anterior modelo.

En el caso de la familia de algoritmos YOLO, esta técnica conlleva utilizar un conjunto de datos grande y general para transferir los pesos, en este caso se utiliza el popular dataset COCO (Common Objects in Context) para después ajustarlo al nuevo conjunto de datos personalizado.

Una vez tenemos el modelo pre-entrenado el algoritmo se encarga de congelar las capas iniciales del dicho modelo que cubren características generales de bajo nivel, como bordes y texturas. Estas capas se guardan durante el entrenamiento posterior para evitar que cambien y conservar las funciones aprendidas previamente. Las últimas capas que están más cerca de la salida se encargan de capturar características de alto nivel y por ello se modifican y ajustan para entrenar el nuevo conjunto de datos, con el objetivo de mejorar las predicciones esperadas para, en nuestro caso, las prendas de ropa de interés.

Es interesante tener en cuenta que el aprendizaje de transferencia en YOLO requiere una selección adecuada de modelos previamente entrenados y una selección adecuada de capas para congelar y ajustar. Además, se recomienda un mayor ajuste y experimentación para ajustar los hiperparámetros de acuerdo con las características del conjunto de datos adaptado y los requisitos específicos del problema.

Capítulo 3 – Metodología

3.1 Hardware

Aunque en este proyecto predomine el uso y el trabajo software, en el aprendizaje automático también es muy importante la parte de hardware que interviene en diferentes factores. Cuanto más potente sea el hardware que utilizamos, más rápido será el proceso de entrenamiento.

En nuestro caso, mediante Google Colab, contamos con una GPU Tesla T4, con la versión 12 de CUDA, que nos ofrece un alto rendimiento en la velocidad de entrenamiento. Esta GPU acelera significativamente el tiempo de entrenamiento de nuestro modelo y la capacidad de cálculo. Sin embargo, esta tarjeta gráfica está limitada por unos 15 GB de VRAM.

Con los siguientes comandos podemos ver los recursos que nos ofrece esta tecnología:

```
!cat /proc/cpuinfo

!nvidia-smi
```

Como podemos ver en la salida de la celda de Google Colab, observamos que estamos trabajando con la GPU mencionada anteriormente y un CPU Intel(R) Xeon(R) @ 2.20 GHz con 6 núcleos además de 12.7 GB de RAM.

Por otra parte, al trabajar en la nube, estamos limitados en cuanto a la memoria del sistema, como bien hemos dicho, la GPU cuenta con 15 GB de VRAM mientras que nuestro almacenamiento en la nube para el proyecto es de 78.2 GB de espacio en un disco virtual.

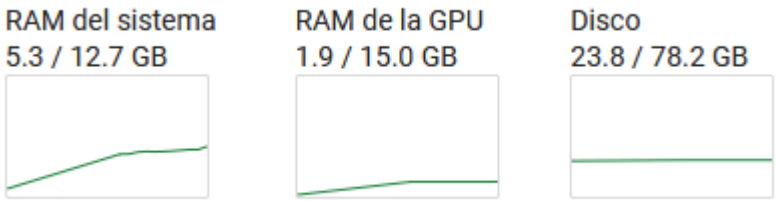

Figura 10. Recursos Google Colab

Además, la mayoría de hardwares que se utilizan en el mundo del aprendizaje automático y profundo, son hardwares diseñados específicamente para tareas de inteligencia artificial, optimizando el rendimiento de las tareas IA.

3.2 Software

Se presentan de forma breve las aplicaciones que se han utilizado para el desarrollo de este proyecto:

3.2.1 YOLO

Principalmente, aunque el hardware es importante, el software juega el papel principal en el mundo del aprendizaje automático. En este caso, enfocándonos en el algoritmo YOLO, su implementación en el contexto del entrenamiento permite obtener grandes resultados gracias a las redes neuronales. Además, existen múltiples bibliotecas que ofrecen implementaciones y herramientas para YOLO que mejoran el desarrollo y entrenamiento del modelo como Darknet, TensorFlow y PyTorch, donde el último es el utilizado para realizar este proyecto.

También este software nos aporta un procesamiento de datos que resulta muy útil. Esto implica preparar conjuntos de datos etiquetados, donde cada imagen se anota con las coordenadas y etiquetas de los objetos existentes. El software utilizado debe proporcionar herramientas para cargar, convertir, normalizar y descomponer correctamente los datos.

El software también juega un papel igual de esencial en la optimización del rendimiento cuando se entrena bajo el modelo YOLO. Los marcos de aprendizaje profundo a menudo proporcionan técnicas avanzadas, como paralelización de cómputo, optimización de memoria y aceleración de GPU, para maximizar la eficiencia y la velocidad del entrenamiento. Además, también nos permite ajustar los hiperparámetros del modelo YOLO durante el entrenamiento. Los hiperparámetros son parámetros que determinan cómo se entrena y ajusta el modelo. El software brinda la flexibilidad para explorar y ajustar estos hiperparámetros, como la tasa de aprendizaje, el tamaño del lote y la función de activación, con el objetivo de mejorar el rendimiento y la precisión del modelo.

3.2.2 Google Colab

Esta herramienta es una plataforma en la nube gratis que ofrece Google. Consisten en un entorno de desarrollo basado en lo que se conoce como “notebooks” del proyecto de desarrollo software Jupyter. Google Colab es altamente conocido en el mundo del aprendizaje profundo debido a la accesibilidad, facilidad y disponibilidad de grandes recursos que nos ofrece de manera gratuita, aunque limitada.

Como bien hemos dicho, el entorno está basado en notebooks, que son documentos interactivos que combinan código, texto explicativo y visualización en un entorno de programación. Permiten la ejecución de código paso a paso y la visualización inmediata de resultados. Los cuadernos se han vuelto muy populares en los campos de la ciencia de datos y el aprendizaje automático por su capacidad para integrar código y narrativa en un solo documento.

Esto facilita mucho el desarrollo del código enfocado al aprendizaje automático y profundo. Además, como bien hemos indicado en el anterior punto de hardware, Google Colab es un software que nos ofrece acceso a recursos (GPUs, CPU, almacenamiento) mucho más potentes y enfocados en el desarrollo de proyectos de inteligencia artificial.

Figura 11. Logo Google Colab
https://www.marketing-branding.com/google-colaboratory-colab-guia-completa-espanol/

3.2.3 Python

Un factor fundamentral, por no decir el más importante, es Python. La base de este proyecto y el lenguaje de programación por excelencia para el desarrollo de proyecto, modelos y herramientas de inteligencia artificial. En este caso utilizaremos Python-3.10.11, la versión que nos viene por defecto en Google Colab, además del IDE Spyder para desarrollar la parte de *front-end* que ejecuta un servidor Flask donde podemos interactuar directamente con nuestro modelo YOLO y ver en tiempo real los resultados.

Resumidamente Python es un lenguaje de programación de propósito general, con una sintaxis muy clara, programación orientada a objetos, un amplio abanico de bibliotecas que aportan grandes posibilidades, una comunidad activa y una gran compatibilidad y portabilidad.

Python se usa en una amplia variedad de aplicaciones, desde desarrollo web y automatización de tareas hasta inteligencia artificial y análisis de datos. Su combinación de legibilidad, flexibilidad y una sólida comunidad de desarrolladores lo convierte en una opción popular tanto para principiantes como para expertos en programación.

Figura 12. Logo Python
https://es.wikipedia.org/wiki/Historia_de_Python

3.2.4 HTML5

También se ha utilizado la última versión del popular lenguaje de marcado HTML junto a flask para desarrollar una pequeña parte de front donde simplemente se espera que el usuario cargue una imagen para ser procesada con YOLO y posteriormente ver el resultado de la predicción, con los objetos detectados.

3.3 Dataset

Para poder llevar a cabo el proceso de detección de objetos con el algoritmo YOLO es necesario unos pasos previos en el mundo del aprendizaje profundo y uno de ellos es el dataset. El conjunto de datos desempeña un papel fundamental en las investigaciones de aprendizaje profundo ya que se podría decir que es la base de información sobre la que el modelo va a aprender a resolver el problema.

En este apartado se describirá el conjunto de datos utilizados en el trabajo, así como sus características, anotaciones y consideraciones de porqué he elegido este dataset.

3.3.1 Fuente

El conjunto de datos escogido para realizar este proyecto se conoce como: *Colorful Fashion Dataset For Object Detection*. Este dataset proviene originariamente de un trabajo que aborda el problema de detectar y analizar imágenes de moda con una supervisión de etiquetas catalogadas por color, por lo tanto, este conjunto de datos cuenta con la ausencia de etiquetas a nivel de píxel.

Sin embargo, este conjunto de datos ha sido modificado y adaptado para la detección de objetos y se han añadido etiquetas en distintos formatos (XML y TXT) para poder realizar un aprendizaje mucho más eficaz.

3.3.2 Características del dataset

Es fundamental contar con un conjunto de datos adecuado para el desarrollo de un proyecto de computer vision, que sea adecuado y representativo para los objetos que queremos detectar. El principal contenido del *dataset* serán imágenes con sus correspondientes etiquetas que contendrán las características necesarias para la detección de objetos.

En total nuestro conjunto de datos está formado por 2682 imágenes en color compuestas por modelos femeninas con diferentes prendas de ropa y accesorios de moda en cada figura. Sin embargo, este dataset no está segmentado y por lo tanto no podemos distinguir las imágenes de entrenamiento con las imágenes de validación. Para ello, ha sido desarrollado un script muy simple en Python que junto a sus librerías nos ha permitido crear un subconjunto que almacene las imágenes correspondientes a la validación.

Gracias a este pequeño script podemos segmentar el dataset en dos subconjuntos de datos, el subconjunto de entrenamiento, formado por el 80% de las imágenes y el subconjunto de validación, formado por el 20% restante. En cuanto a número de imágenes, tendremos unas 2145 imágenes para entrenar y 537 imágenes para validar. Todo esto acompañado, por supuesto, de su correspondiente etiqueta en formato TXT.

La extensión de los datos es relevante ya que este formato nos permitirá tener nuestras imágenes sin pérdidas de color e información a la hora de comprimir y escalar los datos.

Esto último nos permitirá trabajar con nuestros datos de una manera más rápida, ya que la compresión de imágenes nos ayuda a optimizar tanto el almacenamiento como la transmisión de datos. Sin embargo, aunque cuanto más comprimido esté la información más rápido será el algoritmo trabajando, esto no es del todo eficaz ya que afecta a la calidad de la imagen y en algunos casos puede conllevar una pérdida de información importante. En este trabajo las imágenes que nos ofrece el conjunto de datos están escaladas a una resolución de 400 píxeles de ancho y 600 de alto. Siendo está una resolución óptima para no perder demasiada información, pero ideal para trabajar con los datos de una manera rápida.

Además, es interesante tener un conjunto de datos que cuente con bastantes recursos y que estos sean variados, con diferentes características lumínicas, fondos diferentes, objetos más grandes o pequeños etc. El conjunto de estos factores conlleva un algoritmo más robusto

La selección de datos puede involucrar distintas fuentes como una base de datos pública, imágenes personales hechas con dispositivos propios u otros tipos de procedencias como fuentes públicas o fuentes en línea.

3.3.3 Otros datasets

En el mundo de la moda y la industria *fashion* el Deep Learning es realmente interesante debido a la capacidad de analizar y comprender imágenes de manera eficiente. Nos permite detectar y clasificar productos de moda, lo que tiene una gran aplicación en el comercio electrónico, la generación de contenido creativo en el ámbito de la moda o por ejemplo mejorar la calidad de la producción, detectando defectos en prendas de manera efectiva y rápida.

Es por eso por lo que existen múltiples datasets para la industria de la moda, entre ellos los más populares e interesantes son:

- ModaNet: Es un conjunto de datos que combina imágenes de moda de sitios web de comercio electrónico con segmentaciones finas de prendas de vestir. ModaNet permite el entrenamiento y evaluación de modelos de segmentación de moda porque contiene alrededor de 55,000 imágenes con segmentaciones precisas de prendas.

- iMaterialist Fashion Dataset: Es un conjunto de datos cuyo objetivo es clasificar prendas de vestir y deetctar objetos en imágenes relacionadas con la moda. Contiene más de un millón de fotos anotadas de una amplia gama de categorías

de moda. El conjunto de datos también contiene descripciones de texto y características visuales.

- DeepFashion2: Es un conjunto de datos que contiene alrededor de 500.000 imágenes repartidas en 13 diferentes categorías. En su totalidad este dataset cuenta con más de 800.000 prendas de ropa etiquetadas para entrenar modelos de aprendizaje profundo.

- FashionMNIST: Es un conjunto de datos que contiene imágenes de ropa en escala de grises tomadas de Zalando, la plataforma de comercio electrónico. FashionMNIST está compuesto por diez mil imágenes de prueba y sesenta mil imágenes de entrenamiento, divididas en diez categorías de ropa distintas, como camisetas, vestidos y zapatos, entre otras. Cada foto tiene una dimensión de 28 por 28 píxeles.

Aunque existen diferentes conjuntos de datos muy populares y con una gran cantidad de datos, no siempre significa que sean más adecuados para nuestro problema. Para elegir un dataset para un proyecto hay que considerar diferentes factores como el tamaño y la diversidad, teniendo en cuenta que cuanto más grande sea el conjunto de datos, más se demorará nuestro algoritmo en entrenar. La calidad y precisión de etiquetas, ya que no siempre las imágenes que contienen todos los conjuntos de datos ofrecen una vista clara del objeto que queremos detectar. La disponibilidad del dataset, debido a que algunos que podemos encontrar no son accesibles de manera gratuita o para descargar fácilmente.

Junto a varios factores como estos, uno de los más decisivos que han influido en la selección de nuestro dataset, mencionado anteriormente, es el factor del tamaño. Estos conjuntos de datos contienen tantas imágenes que es difícil desarrollar un proceso de entrenamiento con herramientas limitadas y gratuitas como es Google Colab.

A pesar de que con un conjunto de datos podríamos haber obtenido mejores resultados, el proyecto se ha llevado a cabo con un dataset con suficientes imágenes y una buena claridad que permite su desarrollo y la obtención de resultados exitosos.

3.4 Etiquetado del conjunto de datos

Posteriormente, a la hora de preparar el dataset y un paso obligatorio para el proceso de aprendizaje con algoritmos YOLO es el etiquetado de información. Este proceso consiste en indicar manualmente las coordenadas de los objetos que queremos detectar, delimitando el objeto dentro de una caja. Es necesario realizar con precisión el etiquetado para marcar bien los objetos para garantizar la calidad y efectividad del entrenamiento.

Para llevar a cabo el etiquetado del conjunto de datos existen múltiples herramientas que facilitan el proceso, consistiendo en dibujar *bounding boxes* alrededor de los objetos de interés. Algunas de esas herramientas que admiten guardar las anotaciones en formato

YOLO son: LabelImg, RectLabel, Labelbox , VoTT (Visual Object Tagging tool) y una de las más populares, Roboflow.

Posteriormente de etiquetar los datos, obtendremos un archivo TXT por cada imagen que contendrá la información necesaria para el entrenamiento del modelo. Este formato que acepta YOLO contiene una línea por cada objeto etiquetado dentro de imagen, siguiendo el siguiente formato:

<object-class> <x> <y> <width> <height>

Donde el primero elemento sería el número de la clase del objeto detectado, de 0 a 9 ya que hay 10 clases diferentes. La X e Y serían las coordenadas del centro del objeto, haciendo referencia a la relación ancho y alto de la imagen, además, estas coordenadas deberán de estar normalizadas entre 0 y 1. Este proceso lo hace automáticamente el programa con el que etiquetemos los datos, normalizando las coordenadas con las junto a las dimensiones de las imágenes en este caso.

Posteriormente, los dos últimos parámetros indican el ancho y objeto la bounding box que dibujará el objeto detectado. Estos valores también deberán estar normalizados entre 0 y 1 en relación con el ancho y la altura de la imagen.

```
8 0.208750 0.511667 0.087500 0.103333
9 0.363750 0.911667 0.212500 0.070000
6 0.611250 0.548333 0.772500 0.416667
2 0.401250 0.283333 0.502500 0.223333
```

Figura 13. Ejemplo etiqueta TXT

En esta imagen podemos ver que las etiquetas pueden llegar a contener múltiples objetos, tantos como haya que detectar en la imagen anotada. En este ejemplo hay 4 objetos diferentes etiquetados, con su número de clase correspondiente, sus coordenadas y valores de ancho y alto de la caja delimitadora.

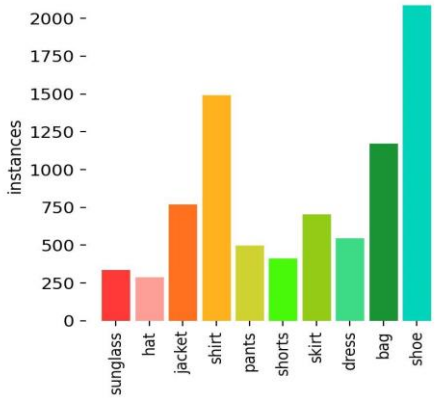

Figura 14. Instancias del conjunto de datos

Con este gráfico podemos observar las instancias de las clases en las imágenes. En este caso lo que más encontramos son zapatos, siendo más de 2000 instancias. Esto quiere decir que en nuestro conjunto de datos encontramos aproximadamente 2100 etiquetas de zapatillas. Seguido de esto encontramos, con 1500 instancias, la clase de las camisetas y en tercer lugar la clase que referencia las bolsas, mochilas o bolsos.

Estas tres clases al ser las más instanciadas quiere decir que aparecen más que las demás, apareciendo siempre en más de la mitad de las imágenes del dataset. Esto conlleva un mejor rendimiento a la hora de detectar estas prendas en las imágenes deseadas. Sin embargo, pasa lo contrario con las clases opuestas, las que menos instancias tienen como las gafas de sol o los gorros, están más condicionadas a un peor rendimiento ya que la red no tendrá tantos ejemplos con los que ser entrenada.

Una vez tenemos todo el conjunto de datos preparado podemos continuar con el proceso de entrenamiento. El siguiente paso será configurar los parámetros necesarios a nuestro gusto para el entrenamiento del modelo YOLO.

3.5 Configuración de parámetros de entrenamiento

Para llevar a cabo nuestro proyecto primero debemos configurar los parámetros del algoritmo de entrenamiento, estos parámetros definen las opciones y los valores que intervendrán en el proceso de entrenamiento, ajustando pesos, para obtener un resultado deseado, efectivo y preciso. Un parámetro muy relevante es el número de clases que queremos detectar con nuestra red YOLO. El algoritmo *You Only Look Once* nos permite modificar hasta cuarenta y cinco parámetros diferentes a la hora de entrenar la red, sin embargo, para la detección de objetos en este proyecto muchos no son relevantes.

3.5.1 Modelo

En este proyecto modificaremos principalmente ciertos parámetros, que se consideran los más importantes; pero para esto, lo primero es indicar el modelo que queremos entrenar y aplicar todos estos parámetros. Simplemente debemos de indicar la ubicación del archivo con extensión ".pt" o descargar uno nuevo gracias a las funciones de YOLO y almacenarlo en una variable, como se refleja en el cuaderno de este proyecto.

El modelo utilizado en este proyecto es el *large*, que corresponde al segundo modelo más grande de los que nos ofrece YOLO. Las características de esta red son:

- 365 capas
- 43637550 parámetros
- 43637534 gradientes

He escogido el modelo *large*, ya que tiene un resultado bastante efectivo en comparación al poco tiempo que nos lleva el entrenamiento y la detección de los objetos.

3.5.2 Datos de entrenamiento

El segundo parámetro se encarga de mostrar los datos que utilizaremos para entrenar como el número de clases, la ubicación de las imágenes de entrenamiento y validación y las anotaciones correspondientes. Esto consiste en indicar donde se encuentra el archivo con extensión 'yaml". En este caso lo hemos nombrado como "data.yaml" y su contenido es el siguiente:

```
train: /content/DATASET/train
val: /content/DATASET/validation

# number of classes
nc: 10

# class names
names: ['sunglass','hat','jacket','shirt','pants','shorts','skirt','dress','bag','shoe']
```

Figura 15. Contenido del data.yaml

Como se puede observar el archivo es muy simple y cuenta con la ubicación (en Google Drive para poder trabajar con Google Colab) de los datos de entrenamiento y validación. Estas carpetas cuentan con dos carpetas más en su interior: *images* y *labels*. A partir de esto, YOLO ya es capaz de interpretar por sí solo dónde se encuentran los datos necesarios. El resto del archivo es sencillo, ya que solo es indicar el número de diferentes clases que se encuentran en nuestras anotaciones e imágenes que queremos detectar.

3.5.3 Epochs

Por otra parte, los *epochs*, que también se conoce como número de épocas, indican el número de veces que el nuestro modelo va a recorrer todo el conjunto de datos durante el proceso de entrenamiento, es decir, en nuestro caso, la red YOLO verá cincuenta épocas (cincuenta veces) todos los datos que se encuentran en nuestro dataset. Este parámetro es muy importante determinarlo adecuadamente, ya que un número excesivo de épocas puede conllevar un sobre entrenamiento del modelo, haciendo que nuestra red sea solo capaz de detectar los objetos que sean prácticamente exactos que los que se encuentran en el dataset. Es decir, en lugar de entrenar el modelo a que aprenda a detectar objetos, entrenas el modelo a que memorice estos objetos exactos.

Para encontrar el número exacto de épocas que entrenar nuestro modelo de aprendizaje profundo es recomendado seguir las estadísticas de entrenamiento y desempeño del modelo mientras su entrenamiento, como la precisión o el valor de la función de pérdidas. Además, también es posible evaluar el modelo de entrenamiento con el subset de validación después de cada época.

En este proyecto, el modelo de inteligencia artificial ha sido entrenado con 50 épocas, esto es debido a que, en los resultados, que se muestran y explican en los siguientes apartados, podemos observar la precisión y el desempeño del modelo, viendo que tras

entrenar durante 50 épocas el modelo tiene un buen rendimiento en un tiempo relativamente corto.

3.5.4 Batch

En el aprendizaje profundo, el parámetro "batch" se refiere al tamaño del lote o al número de imágenes utilizadas en cada paso de entrenamiento. Los datos de entrenamiento, como YOLO, se dividen en lotes para un procesamiento más eficiente.

Cuántas imágenes se incluirán en cada lote durante el entrenamiento se determinará por el parámetro "batch". Por ejemplo, si tiene un conjunto de datos de entrenamiento de 1000 imágenes y establece el parámetro "batch" en 32, se utilizarán 32 imágenes en cada paso de entrenamiento. Esto significa que, antes de pasar al siguiente lote, el modelo actualizará sus pesos y realizará los cálculos necesarios en función de estas 32 imágenes.

El tamaño del lote, también conocido como "batch size", puede variar según el tipo de conjunto de datos y los recursos computacionales disponibles. Se elige generalmente un tamaño de lote lo suficientemente grande para aprovechar la paralelización y mantener un equilibrio entre la estabilidad del gradiente y la eficiencia computacional, sin que sea demasiado grande para agotar la memoria o ralentizar el entrenamiento.

En este caso para el entrenamiento de nuestro modelo de aprendizaje profundo hemos utilizado un *batch size* de 8 imágenes.

3.5.5 Tamaño de imágenes

La resolución de las imágenes es bastante relevante en el proceso de entrenamiento, pues cuantos más píxeles tenga la imagen, mejor calidad tendrá, pero más recursos necesitará nuestro modelo para trabajar con ella. Es por eso por lo que normalmente los conjuntos de datos cuentan con imágenes de “poca calidad” pero que sean claras. Un dataset con un gran número de imágenes de alta resolución podría resultar en un entrenamiento poco efectivo ya que se demoraría demasiado tiempo en cada época.

En este caso, el conjunto de datos seleccionado para el proyecto trabaja con unas 2.682 imágenes con una resolución de 400 píxeles de ancho por 600 píxeles de alto.

Sin embargo, en el proceso de entrenamiento, el modelo nos asignará automáticamente el tamaño de imagen a 608 píxeles. Al usar capas de convolución el tamaño de la imagen debe ser múltiplo de 32, ya que 32 es el número que se establece generalmente para el *max stride*. Esto se realiza para mantener la compatibilidad con todas las capas convolucionales de la red neuronal.

3.5.6 Algoritmo de optimización elegido

YOLO nos permite elegir la función de optimización que queramos utilizar a nuestro gusto y qué más adecuada sea para nuestro tipo de problema.

Nos ofrece una lista de múltiples opciones compuesta por: *SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp,* y una última opción *automática*, que viene por defecto si no se especifica ninguna, que escogerá el algoritmo de optimización por su propio pie detectando cuál es más adecuado para entrenar la red en comparación del modelo y conjunto de datos que estemos utilizando.

En este caso para entrenar nuestra red neuronal utilizamos el popular algoritmo ADAM, ya que es ampliamente usado en el entrenamiento de las redes YOLO, proporcionando grandes resultados gracias a que combina las ventajas del descenso de gradiente estocástico con momentum y el algoritmo de descenso de gradiente RMSprop.

Además, que ADAM adapta automáticamente el *learning rate* en cada parámetro y tiene una gran robustez ante cambios en parámetros e hiper parámetros.

Aunque no haya un algoritmo de optimización perfecto, es posible encontrar el más indicado a nuestro problema haciendo pruebas y comparando resultados.

3.6 Cuaderno interactivo de Google Colab

Todo el proceso de entrenamiento y pruebas del modelo está reflejado en un cuaderno de Google Colab que ha sido desarrollado desde cero para este trabajo.

El cuaderno está formado con un conjunto de celdas que paso a paso consiste en hacer pruebas antes y después de entrenar el modelo con diferentes imágenes, tanto de dataset como fotos propias o extraídas de internet. Además de todo el proceso de instalación del algoritmo YOLO y dependencias necesarias para realizar los entrenamientos, predicciones y pruebas.

3.6.1 Instalación

La primera celda contiene la instalación de YOLO, esto se realiza mediante la clonación del repositorio de github de Ultralytics y la instalación del paquete de este.

Además también contiene la instalación de los requisitos necesarios (más paquetes de python como numpy, matplotlib etc) necesarios para el uso de YOLO. Todas las instalaciones se realizan mediante un proceso de interacción con el CLI virtual que nos ofrece Google Colab además de la herramienta de instalación *pip* de anaconda.

```
[1]   1 # Instalar y descargar ultralytics
      2 !pip install ultralytics
      3
      4 # Clonamos el repositorio para acceder a los archivos
      5 !git clone https://github.com/ultralytics/ultralytics
      6
      7 # Instalamos los requisitos
      8 %cd ultralytics
      9 %pip install -r requirements.txt
     10 %cd ../
```

Figura 16. Celda de instalación del algoritmo ultralytics

Posteriormente habilitaremos la opción de que Google Colab acceda a nuestros archivos subidos en nuestro Google Drive, donde almacenaremos el conjunto de datos y los resultados.

```
[2]   1 # Cargamos los datos de nuestro Google Drive
      2 from google.colab import files
```

Figura 17. Celda de carga de datos en Google Drive

3.6.2 Descarga del Dataset

En esta celda simplemente se accede al conjunto de datos almacenado en Google Drive, en este caso en el mío personal. Una vez tenemos acceso solo hay que descomprimir el dataset para poder trabajar con él mucho más fácilmente, ya que teniendo en la nube directamente nos permite ahorrarnos tiempo.

```
1 # Descomprimimos el Dataset previamente subido en nuestro Google Drive
2 !unzip /content/drive/MyDrive/UNIVERSIDAD/TFG/DATASET.zip
```

Figura 18. Celda de descompresión del conjunto de datos

3.6.3 Librerías necesarias y recursos

Una vez tenemos descargado e instalado el algoritmo YOLO y el conjunto de datos listo para trabajar con él, instalaremos las librerías necesarias para poder llevar a cabo el proyecto.

```
[4]  1 import torch
     2 import numpy as np
     3 import matplotlib.pyplot as plt
     4 import os
     5 import cv2
     6 import ultralytics
     7 import random
```

Figura 19. Celda de importación de paquetes

En este caso, estas son las librerías más utilizadas en el campo del Deep Learning, donde utilizaremos *torch* (PyTorch) para entrenar la red neuronal, *numpy* y *matplotlib* para complementar el algoritmo YOLO, *os* para acceder a carpetas y directorios en Google Colab, *cv2* para representar las imágenes necesarias, *random* para escoger imágenes aleatoriamente de nuestro conjunto de datos y finalmente la librería ultralytics que habíamos descargado e instalado antes.

Para comprobar los recursos que tenemos a nuestra disposición ejecutaremos un par de celdas muy sencillas que nos mostrarán tanto la GPU utilizada, como la memoria, el CPU y versiones CUDA entre otros datos.

```
Sat Jun 10 10:36:19 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   64C    P8    11W /  70W |      3MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figura 20. Recursos a disposición

3.6.4 Descarga y pruebas del modelo YOLO

Una de las grandes ventajas que tenemos al utilizar la familia de algoritmos YOLO es la posibilidad de descargar un modelo de aprendizaje profundo listo para ser entrenado. Para realizar el trabajo como bien se explica en otro apartado, escogemos el modelo *large* y podemos descargarlo con una simple línea de código.

```python
from ultralytics import YOLO

# Desacarga y carga del modelo "Large" de YOLO
modelo = YOLO("yolov8l.pt")
```

Figura 21. Celda de carga del modelo

Antes de comenzar a entrenar el modelo probamos el desempeño del modelo previamente entrenado por YOLO.

Figura 22. Prueba del modelo en dataset COCO128

Como podemos ver, tiene un gran resultado en su objetivo, el modelo directamente descargado está entrenado con un conjunto de datos que permite detectar un grupo de objetos general, como por ejemplo una corbata o una persona entre otras muchas. Ahora vamos a probar el modelo con alguna imagen de nuestro conjunto de datos personalizado, solamente para comprobar el resultado y ver que es capaz de detectar.

Figura 23. Prueba del modelo en dataset personalizado

Aunque el modelo no sea capaz de detectar nada más que una persona y un bolso de mano, la precisión es muy buena. Pero como nuestra intención es detectar prendas de ropa, tendremos que comenzar con el proceso de entrenamiento.

3.6.5 Entrenamiento de la red con Dataset personalizado

En este proceso ejecutaremos la celda que entrena la red neuronal de nuestro modelo YOLO. Aunque en apartados previos ya se ha especificado los parámetros de entrenamiento, la celda se compone por los siguientes:

```
# Entrenaremos el modelo con el dataset personalizado y los siguientes paramétros:
# 1. 50 epochs
# 2. batch de 8 imagenes
# 3. imgsz de 600px
modelo.train(data='/content/DATASET/data.yaml', epochs=50, batch=8, imgsz=600, optimizer="Adam", project="TFG")
```

Figura 24. Celda de entrenamiento

El modelo es el cargado previamente en la celda anterior. De este modo solo nos faltaría indicar la ubicación del archivo con la información necesaria para entrenar y los parámetros de entrada. Escogemos 50 épocas, lotes de 8 imágenes, el tamaño de 600 píxeles y el algoritmo de optimización ADAM. Por último, indicamos el nombre del proyecto para tenerlo mejor ordenado.

El último parámetro indica el nombre del proyecto, creará una carpeta con toda la información y resultados del entrenamiento, y sus próximas predicciones.

Además, mediante el proceso de entrenamiento, YOLO realiza una validación por cada época que entrenamos. En cada iteración utilizamos usa las imágenes de entrenamiento para calcular el gradiente y optimizar los parámetros de entrenamiento, una vez calculado los nuevos parámetros, el algoritmo utiliza el set de imágenes de validación para calcular el error únicamente con esas fotos.

Podemos ver en el resultado de la celda, como este error va bajando poco a poco significando que el algoritmo está aprendiendo y no memorizando, evitando el concepto de *overfitting* o sobre entreno. El objetivo es que el modelo funcione correctamente con datos externos al set de entrenamiento.

3.6.6 Cargar el nuevo modelo entrenado

Esta celda es simple y está enfocada para una vez hayamos entrenado el modelo, poder cargarlo de nuevo. Además de si ejecutamos el cuaderno en otro momento, en lugar de tener que volver a entrenar el modelo, lo que llevaría mucho tiempo de espera, podemos cargar los datos que habremos guardado previamente en nuestro repositorio de Google Drive.

3.6.7 Validación del modelo

Después de entrenar un modelo, nuestro siguiente paso de interés es la validación del modelo. Esta consiste en evaluar el rendimiento y la precisión de un modelo en un conjunto de datos. Para llevar a cabo este proceso necesitaremos un conjunto de imágenes que no hayamos utilizados durante el entrenamiento, en este caso este mismo subconjunto de datos está formado por 537 imágenes de la misma resolución que el conjunto de entrenamiento. Para este proceso utilizaremos la función de validación nos aporta YOLO en su librería. Donde nos mostrará los siguientes datos:

CLASS	IMAGES	INSTANCES	PRECISION
ALL	537	2070	0.796
SUNGLASS	537	93	0.312
HAT	537	66	0.773
JACKET	537	193	0.834
SHIRT	537	365	0.847
PANTS	537	128	0.898
SHORTS	537	94	0.84
SKIRT	537	182	0.94
DRESS	537	136	0.885
BAG	537	294	0.816
SHOE	537	519	0.813

Tabla 2. Validación del modelo

En esta tabla podemos ver los resultados de la validación, en este caso, los datos más importantes. La primera columna indica la clase, en este caso, la primera hace referencia a todas las clases, siendo una media de todos los resultados siguientes. La segunda columna son las imágenes que se han utilizado para validar esa clase, en esta ocasión todas las clases han sido validadas con el mismo número de imágenes. Donde la cosa es diferente es en la tercera y cuarta columna, la tercera son las instancias que ya habíamos visto en apartados anteriores, pero en esta circunstancia se aplica al conjunto de imágenes de validación. Por ejemplo, podemos ver que la clase “hat” en 537 imágenes solo aparece 66 veces.

Donde se encuentra el dato más importante es en la cuarta columna, que es la precisión. Esta es una métrica utilizada para evaluar la calidad de un modelo de detección o clasificación de objetos. Obviamente, cuanto mayor sea la precisión, mejor será el rendimiento de nuestro modelo. Continuando el ejemplo anterior, en la clase que referencia la detección de gorros podemos observar que hay una precisión de 0.773 sobre

1. Este, es un valor decente ya que nos permitirá detectar esta prenda con una buena efectividad.

Sin embargo, no es la mejor ni mucho menos, en otros ejemplos como en la falda, podemos observar que la precisión es de hasta 0.94 puntos sobre 1. Esto quiere decir que el modelo tendrá un desempeño casi perfecto a la hora de detectar faldas en imágenes.

Por otra parte, siempre puede pasar que la precisión de nuestro modelo sea bastante pobre en algunas clases, y aunque no es lo adecuado, puede llegar a pasar. Este es el ejemplo de la clase "sunglass" que hace referencia a las gafas de sol. En este ejemplo lo que ha pasado es que, a la hora de entrenar, había pocas instancias de gafas de sol o es probable que las que hubiese no fueran del todo claras. Aunque en el conjunto de imágenes de validación haya más instancias en de gafas de sol que de otras clases, no quiere decir que el resultado y la precisión sean mejor.

Aunque la precisión es interesante a la hora de la validación, es necesario considerar otras métricas y tener en cuenta el contexto y las características específicas del problema, ya que la precisión puede no ser suficiente para evaluar completamente el rendimiento del modelo.

```
Ultralytics YOLOv8.0.119 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43614318 parameters, 0 gradients
val: Scanning /content/DATASET/validation/labels.cache... 537 images, 0 backgrounds, 0 corrupt: 100%|██████████| 537/537 [00:00<?, ?it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 34/34 [00:19<00:00,  1.74it/s]
                   all        537       2070      0.775      0.796      0.816      0.551
             sunglass        537         93      0.613      0.312      0.411      0.143
                   hat        537         66      0.755      0.773       0.82      0.444
                jacket        537        193      0.712      0.834       0.85      0.657
                 shirt        537        365      0.792      0.847      0.858      0.605
                 pants        537        128      0.924      0.898      0.934       0.72
                shorts        537         94      0.813       0.84       0.88      0.568
                 skirt        537        182       0.78       0.94      0.901      0.717
                 dress        537        136      0.828      0.885      0.881      0.716
                   bag        537        294      0.747      0.816      0.779      0.444
                  shoe        537        519      0.784      0.813      0.843      0.495
Speed: 0.7ms preprocess, 22.4ms inference, 0.0ms loss, 1.7ms postprocess per image
Results saved to runs/detect/val
```

Figura 25. Resultado de la celda de validación del modelo

3.6.8 Predicción de imágenes

Después de validar el modelo y ver que el desempeño es bueno, podemos empezar a detectar prendas de ropa en imágenes. Para ello se hace uso de la función "predict" que nos ofrece la librería YOLO.

```
[ ]  1 img_dir = '/content/DATASET/train/images/181777.jpg'
     2 results = modelo.predict(source=img_dir, conf=0.5, save=True, line_thickness=2, show_labels=True)
     3 img = plt.imread("runs/detect/predict2/181777.jpg")
     4 plt.imshow(img)
```

Figura 26. Celda de predicción

Esta celda del cuaderno interactivo de Google Colab contiene cuatro líneas muy simples, que son las encargadas de llevar a cabo la detección de objetos. Lo primero de todo, el

código leerá la imagen que hayamos seleccionado, en este caso es una imagen que hemos utilizado de ejemplo en previos apartados y la usaremos también en un futuro.

Después ejecutaremos la función "predict" indicándole ciertos parámetros de entrada:

- La imagen que queremos detectar, en nuestro caso la ubicación donde se encuentra dicha imagen.

- El segundo parámetro indica el umbral de detección, un número que si no se supera cuando el modelo indica la puntuación a la hora de detectar un objeto, se considerará erróneo y por ello no se mostrará ni se dibujará la bounding box correspondiente. En este caso, si cada detección no es mayor de 0.50 sobre 1, no se indicará en el resultado.

- El tercer parámetro simplemente indica, con una variable "booleana", si queremos guardar el resultado en nuestro disco, siendo True o Falso, para guardar o no la imagen respectivamente.

- El siguiente parámetro exige un número entero como entrada y nos permitirá modificar el grosor de las líneas de la caja delimitadora que ubica las prendas detectadas.

- Por último, este parámetro como el anterior se indica con True o False y lo usaremos para indicar si queremos ver el nombre de la clase detectada cuando se dibuje la caja delimitadora.

3.6.9 Exportar el modelo y los resultados comprimidos en ZIP

Una vez tenemos el modelo entrenado con nuestro conjunto de datos personalizado, lo comprimimos en ZIP para poder descargarlo más fácilmente. Para ello haremos uso de algunas librerías como zipfile que nos permite comprimir una carpeta, que habremos generado y manejado junto a la librería OS.

Con este código podremos guardar los resultados de todo el entrenamiento y validación, viendo las gráficas, imágenes de batch e imágenes predichas.

```
[ ]
import zipfile
import os

# Función para comprimir resultados
def zip_resultados(carpetas, archivo_zip):
    with zipfile.ZipFile(archivo_zip, 'w') as zipf:
        for carpeta in carpetas:
            for root, _, files in os.walk(carpeta):
                for file in files:
                    file_path = os.path.join(root, file)
                    zipf.write(file_path, os.path.relpath(file_path, carpeta))

# Ejemplo de uso
dir = ['/content/TFG', '/content/runs', '/content/yolov5/runs']
archivo_zip = 'resultados.zip'

zip_resultados(dir, archivo_zip)
```

Figura 27. Celda de exportación del modelo y los resultados

3.6.10 Entrenamiento de modelo YOLOv5

Aunque ya hemos entrenado el modelo de Deep Learning satisfactoriamente, en este trabajo también realizaremos el mismo proceso de entrenamiento para otra versión con la finalidad de comparar resultados y ver que versión es más eficiente con los mismos parámetros y conjunto de datos.

Para ello se han programado un par de celdas simples en el cuaderno de Google Colab que hacen posible este desarrollo.

```
[ ]  1 !git clone https://github.com/ultralytics/yolov5  # clone
     2 %cd yolov5
     3 %pip install -qr requirements.txt comet_ml  # install
     4
     5 import torch
     6 import utils
     7 display = utils.notebook_init()  # checks

YOLOv5 🚀 2023-7-9 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 25.1/78.2 GB disk)

[ ]  1 !python /content/yolov5/train.py --img 600 --epochs 50 --data /content/DATASET/data.yaml --weights yolov5l.pt --optimizer Adam
```

Figura 28. Celda de instalación y entrenamiento YOLOv5

La primera celda nos permite descargar e instalar el repositorio de la quinta versión de YOLO, directamente clonando el repositorio de ultralytics.

Posteriormente ejecutaremos la línea de código en CLI que nos permite entrenar el modelo, en este caso utilizaremos el archivo Python que nos ofrece YOLO, indicaremos la resolución de las imágenes, de forma similar que en la última versión.

Además, entrenaremos durante 50 épocas, con el optimizador Adam y por supuesto con el conjunto de datos de detección de prendas de ropa que hemos utilizado previamente.

3.7 Desarrollo de Front-End

Como parte final del desarrollo del trabajo, añadimos un front-end que nos permite interactuar directamente con el modelo de aprendizaje profundo. Esta parte consiste en un simple servidor flask que nos brinda la capacidad de subir una imagen a nuestro gusto, hecha en tiempo real con el móvil, con la cámara web o cualquier imagen que tengamos en local.

Este servidor está compuesto por dos partes, la parte servidor desarrollada en Python junto a la librería Flask y la parte de HTML desarrollada con el mismo lenguaje, además de un archivo de estilos en formato CSS.

Una vez entramos a nuestra página web y escogemos la imagen, pulsamos el botón de “Subir imagen” se enviará al servidor Flask, donde se procesará la imagen y se realizarán predicciones por parte del algoritmo YOLO.

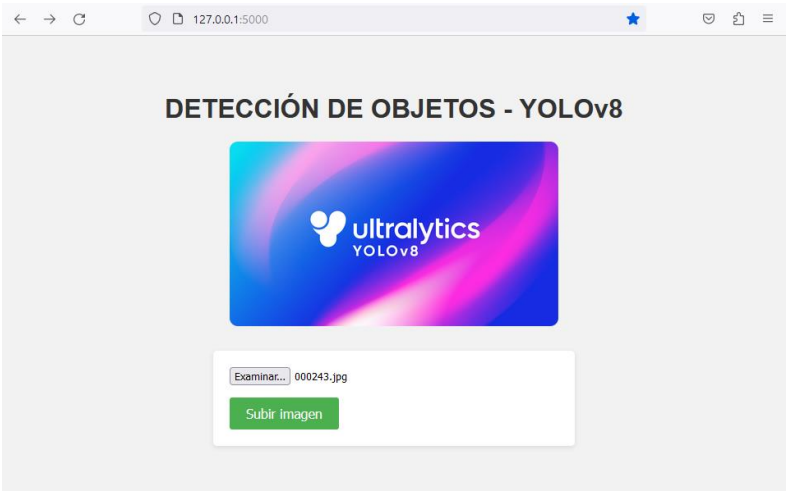

Figura 29. Inicio del front-end

Como podemos observar en la imagen, es un menú simple que nos permite subir una imagen, como bien hemos indicado anteriormente, para mandarla al servidor flask y que se realicen las predicciones obteniendo de esta manera una imagen con los objetos detectados y sus bounding boxes correspondientes.

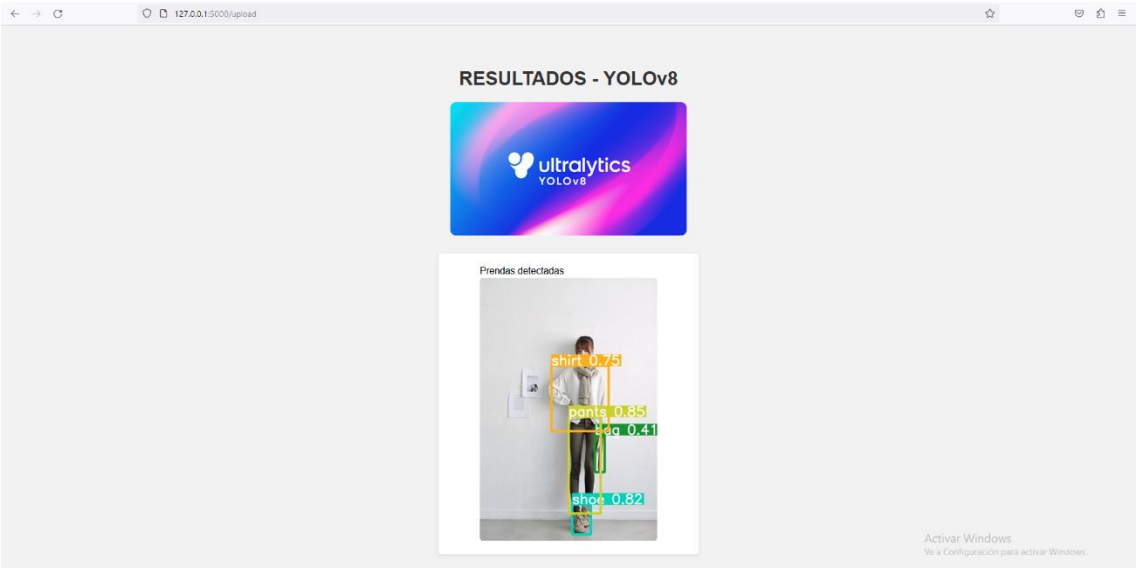

Figura 30. Resultados de la imagen

Una vez subida la imagen y procesada se cargará la siguiente pagina con URL */upload* donde tendremos la imagen con las prendas de ropa detectadas.

Este servicio nos da la flexibilidad de detectar objetos en imágenes a nuestro gusto y mucho más fácil y rápido.

Todo el código correspondiente desarrollado está adjunto en el apartado de anexos.

Capítulo 4 – Resultados

Después de entrenar el modelo y validar su desempeño, es hora de ver los resultados que nos ha proporcionado el entrenamiento y las predicciones.

4.1 Resultados de entrenamiento

Después de entrenar el modelo, es adecuado recurrir a los resultados que nos proporciona YOLO, para poder ver el rendimiento y la calidad del proceso. Esto sirve para ver cómo ha ido avanzando el modelo mientras se entrenaba, permitiéndonos conocer la precisión y otros valores durante cada época, o los batch que ha ido recibiendo el modelo.

Figura 31. Lote de imágenes de entrenamiento

Esta imagen representa lo que conocemos como “batch” o lote imágenes. Este es un lote que el modelo ha utilizado para ser entrenado, en este caso, hemos indicado en los parámetros de entrenamiento el número de imágenes por lote, 8 imágenes. Podemos observar que en cada imagen hay unos cuadrados que se llaman “bounding boxes” o en español, cajas limitadoras. Estas cajas con las encargadas de indicar al modelo la ubicación de cada objeto, en este caso las cajas delimitaran las prendas de ropa junto a un número para indicar a que clase hace referencia y donde se encuentra en la imagen. Como podemos ver en la imagen esto nos sirve para saber que el etiquetado de las imágenes es correcto, es decir, indica con exactitud donde se encuentra el objeto, y que clase de objeto es.

Sin embargo, esto no es todo, además YOLO nos da la opción de ver los resultados del entrenamiento mediante gráficas para ver la calidad de nuestro modelo y su evolución conforme ha sido entrenado.

Aunque YOLO nos exporta automáticamente en la carpeta del proyecto las gráficas donde se plasma el desempeño de nuestro modelo, existe una herramienta llamada TensorBoard que nos permite monitorear los resultados de nuestro modelo de aprendizaje profundo de una manera interactiva y mejor visualmente.

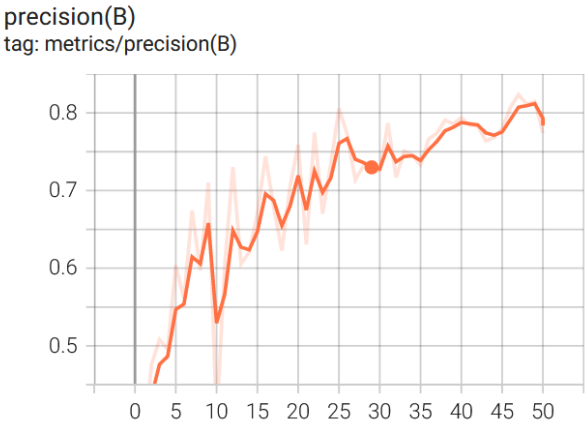

Figura 32. Precisión del modelo

En este gráfico podemos ver la evolución y mejora de la precisión del modelo en función de las épocas de entrenamiento. El modelo comienza con una precisión cercana a cero ya que no ha sido prácticamente entrenado con ningún dato aún.

Poco a poco, la precisión comienza a mejorar conforme el modelo va viendo las imágenes del conjunto de entrenamiento más veces, sin embargo, en la época 10 hay un pico negativo donde la precisión del modelo baja considerablemente, esto puede ser debido a técnicas de sobre ajuste, regularización o haya que ajustar ciertos parámetros para mejorar el rendimiento.

Finalmente, aunque la gráfica va variando en ciertos momentos y el resultado final está suavizado, podemos ver que la precisión se encuentra en su cenit, donde mejor resultado ha obtenido es cerca de la época número 47.

A pesar de que luego la precisión baje algo, el modelo acaba con una precisión de 0.79 puntos sobre 1. Siendo esta una precisión decente a la hora de detectar objetos.

Además, gracias a la familia de algoritmos YOLO tenemos acceso a más gráficas y resultados, en esta ocasión nos interesa observar la popular matriz de confusión. La matriz de confusión es una herramienta muy utilizada en el Deep Learning para validar el desempeño y los resultados de un modelo que muestra como se están clasificando las instancias de los objetos detectados en función de las clases reales.

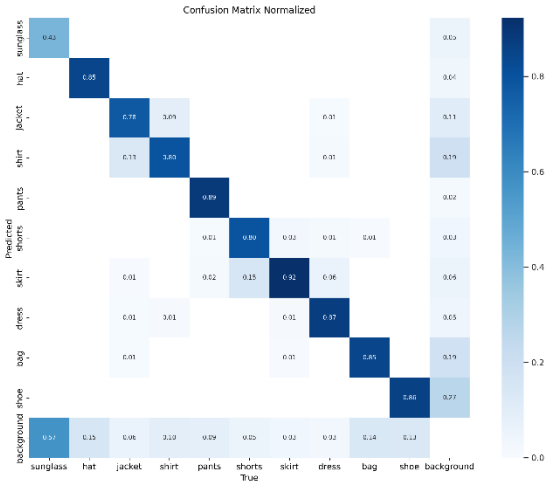

Figura 33. Matriz de confusión

Podemos observar los valores predichos y las confusiones del modelo al tratar de detectar objetos. En este problema multiclase, la matriz de confusión tiene filas y columnas correspondientes a cada clase en el conjunto de datos. Cada celda en la matriz representa el recuento de instancias normalizadas que pertenecen a una clase específica y se predijeron correcta o incorrectamente.

4.2 Predicciones

Una vez el modelo esta entrado, queda la parte más importante y para lo que ha sido desarrollado, para predecir imágenes y detectar objetos en ellas. Para probar el modelo hemos utilizado diferentes datos que pondrán a prueba el modelo y podremos ver el desempeño y eficiencia del algoritmo de Deep Learning. Primero se probará con imágenes del mismo conjunto de datos, y después con imágenes externas, ya sean propias o sacadas de algún conjunto de datos diferente o alguna base de datos.

4.2.1 Predicciones en el propio conjunto de datos de entrenamiento

Cuando se realiza una predicción de imagen con los datos del dataset utilizado para entrenar, lo propio y normal es que el resultado sea totalmente exitoso. Esto sucede ya que se está analizando una imagen que el modelo ha utilizado para ser entrenada, por ello es mucho más sencillo que el modelo encuentre relación y patrones que identifique fácilmente.

Previamente habíamos utilizado una imagen para detectar con el modelo sin entrenar, y como era de esperar el modelo no era capaz de encontrar ninguna clase deseada, sin embargo, ahora con el modelo entrenado el resultado es diferente, y conseguimos una imagen con todos sus objetos detectados perfectamente.

Figura 34. Ejemplo de predicción en dataset de entrenamiento

Como se puede observar, el modelo ha detectado cuatro clases diferentes en esta imagen. Además de la bounding box, YOLO aporta la precisión con la que ha detectado el objeto, en este caso: detecta un bolso con un 0.85 de precisión, los zapatos con un 0.72 de precisión, la chaqueta con un 0.93 de precisión y por último el vestido con un 0.87 de precisión. Cabe destacar que todos estos valores son sobre uno.

Otro ejemplo en el que podemos ver el desempeño del modelo es el siguiente:

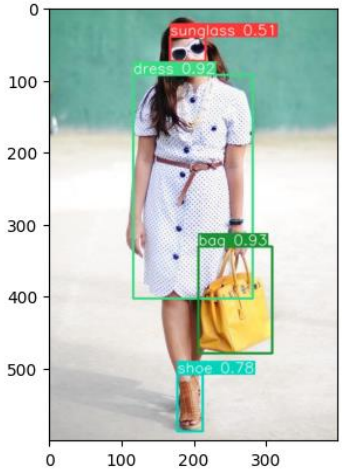

Figura 27. Ejemplo de predicción de gafas de sol

En este ejemplo vemos un resultado bastante similar al anterior, aunque en este caso hay una clase nueva, las gafas de sol. Como hemos visto en apartados anteriores, las gafas de sol son una de las clases más pobres en cuanto eficiencia y precisión a la hora de ser detectadas.

Es por eso por lo que, en este ejemplo, podemos ver como el modelo ha detectado la clase que representa las gafas de sol, pero sin embargo la precisión es bastante justa, siendo esta de 0.51 puntos sobre 1.

Para solucionar este problema deberíamos de tener más instancias de gafas de sol, o aportar imágenes donde la clase sea más clara a la hora de entrenar. Además, también en caso de no ser detectado ciertas prendas podríamos bajar el umbral de detección del modelo, esto aumentará el número de detecciones realizadas en la imagen, incluidos aquellos que son difíciles de detectar o que al modelo le cuesta más. Sin embargo, es importante tener en cuenta que esto conllevará falsos positivos e introducirá detecciones erróneas. Por lo tanto, es necesario equilibrar cuidadosamente el umbral de detección para obtener un resultado óptimo en función de los requisitos y objetivos específicos de la aplicación.

4.2.2 Predicciones fuera del conjunto de datos de entrenamiento

Por otra parte, el modelo también es capaz de detectar objetos en imágenes externas al conjunto de datos utilizado para entrenar, lo cual es el objetivo de este proyecto.

Para realizar estas predicciones se han utilizado imágenes extraídas de internet, relacionadas con la moda donde se encuentran modelos de ropa, gente llevando ropa relacionada, imágenes sacadas de otros conjuntos de imágenes relacionados con el tema e incluso imágenes propias.

Figura 28. Ejemplo de predicción en imagen externa
https://unsplash.com/es/s/fotos/modelo-femenina

En este resultado podemos observar como el modelo funciona con imágenes extraídas de internet, en este caso esta imagen está sacada de la página Unplash, que ofrece una gran variedad de imágenes de uso libre.

Podemos observar que el "score" con el que se detectan las prendas es bastante efectivo.

Además, el modelo, aunque haya sido entrenado con imágenes de modelos femeninas, también es capaz de detectar la ropa que lleva un hombre, siempre y cuando este lleve prendas que estén documentadas en el modelo.

La detección en esta imagen es algo diferente ya que, al ser prendas de hombre, son prendas de ropa que el modelo no ha visto tanto a la hora de entrenar y puede haber confusiones como por ejemplo la sudadera, una prenda que no entra en las clases etiquetadas del conjunto de datos. En esta ocasión sí detecta la chaqueta y las demás prendas correctamente, pero la sudadera no.

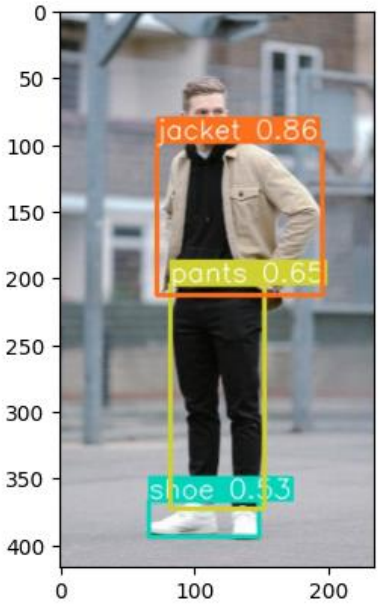

Figura 29. Ejemplo de predicción en modelo masculino https://unsplash.com/es/fotos/kUQhNbKEjVk

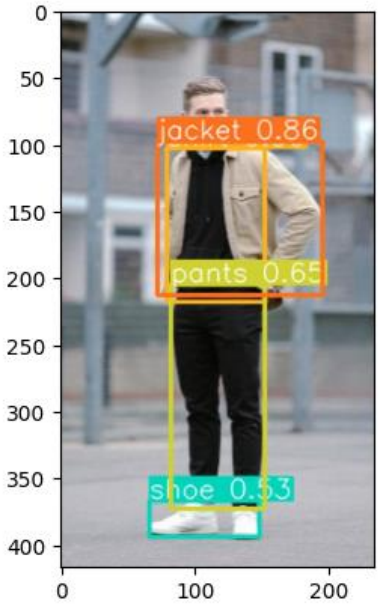

Figura 30. Ejemplo de predicción en modelo masculino con menos umbral https://unsplash.com/es/fotos/kUQhNbKEjVk

Lo más probable que esté sucediendo en este caso es que el modelo esté detectando la sudadera como una camiseta u otra chaqueta, sin embargo, la puntuación que obtiene en la detección no supera el umbral de detección establecido por YOLO y es por ello por lo que no llega a detectar nada ni dibujar una caja delimitadora.

Para solucionar esto simplemente bajaremos el umbral de detección del modelo, en este caso lo pondremos a 0.25 de puntuación, y podremos ver que la sudadera sí que es detectada como camiseta con una puntuación muy cercana a 0.30. Aunque es una

puntuación pobre, nos sirve para hacer las pruebas y ver cómo funciona el umbral de detección.

También existen más conjuntos de datos que recopilar prendas de ropa para el entrenamiento de redes neuronales convolucionales. DeepFashion2 es un datase muy extenso que nos ofrece trece clases diferentes de prendas de ropa, sin embargo, este conjunto de datos al ser tan grande, aunque se obtienen unos mejores resultados, se demora demasiado en entrenar un modelo.

Aquí podemos ver como nuestro modelo entrenado es capaz de detectar prendas de ropa de otros datasets, como el mencionado anteriormente.

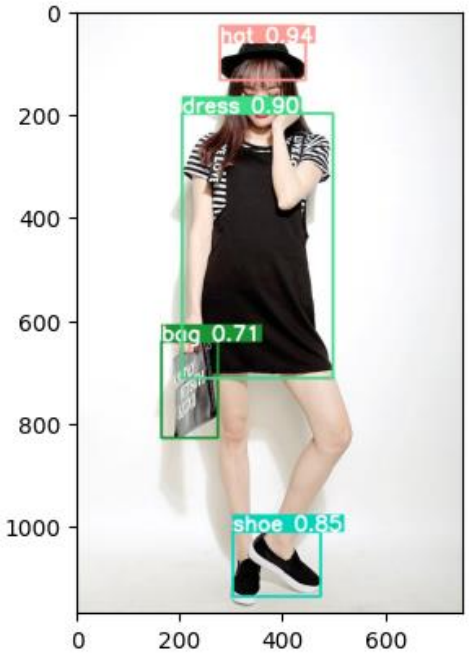

Figura 31. Ejemplo de predicción en imagen de otro dataset

Por último, podemos ver una predicción en una imagen propia, hecha con un teléfono móvil a tiempo real. Esta imagen está escalada para tener unos píxeles similares a las imágenes del conjunto de datos. En esta imagen detecta perfectamente las prendas de ropa con buenas puntuaciones ya que es una imagen con buena iluminación y las piezas de ropa se pueden ver con claridad. También se ha hecho pruebas con más personas donde el resultado ha sido exitoso.

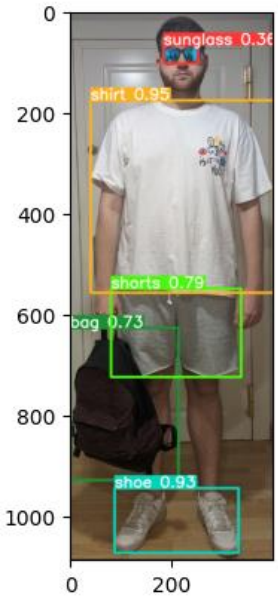

Figura 32. Ejemplo de predicción en imagen propia

4.3 Resultados de YOLOv5

A pesar de ver que el modelo de YOLO y su versión número ocho tienen buenos resultados y hemos podido hacer pruebas exitosas, también es posible utilizar alguna versión inferior que ha sido publicada por la propia empresa de Ultralytics.

En este proyecto utilizaremos los resultados de YOLOv5 para comparar y ver diferencias, utilizados en similares imágenes y de esta manera veremos el rendimiento de los dos modelos.

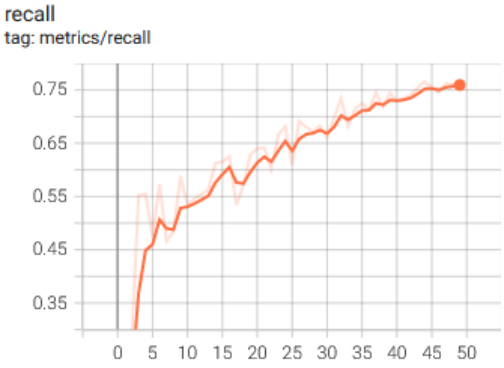

Figura 35. Precisión de YOLOv5

Tras completar el proceso de entrenamiento con los mismos parámetros y conjuntos de datos, podemos ver el desempeño del modelo donde la gráfica de la precisión es bastante similar, pero algo diferente. El punto máximo de la gráfica 0.7593 mientras que, con el modelo de la última versión, YOLOv8, obtenemos 0.7953 con valores suavizados, pero mejorando el entrenamiento se podrían obtener mejores resultados.

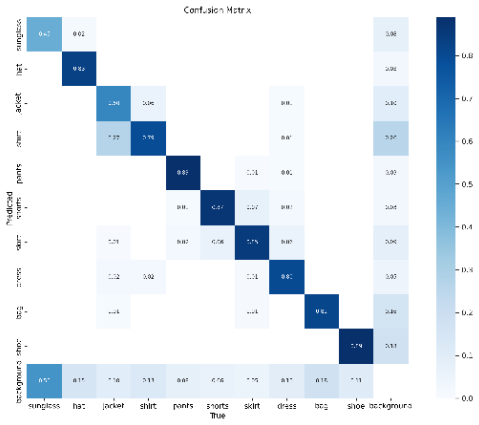

Figura 36. Matriz de confusión YOLOv5

Por otra parte, si vemos la matriz de confusión vemos que, en comparación a la otra versión, tenemos diferentes puntuaciones de predicciones reales, lo que conlleva a unos resultados algo peores que en las versiones superiores.

Ejecutando la primera predicción con YOLOv5 tenemos esta imagen, donde la predicción, aunque sea exitosa no es del todo perfecta y algo peor que con la versión YOLOv8.

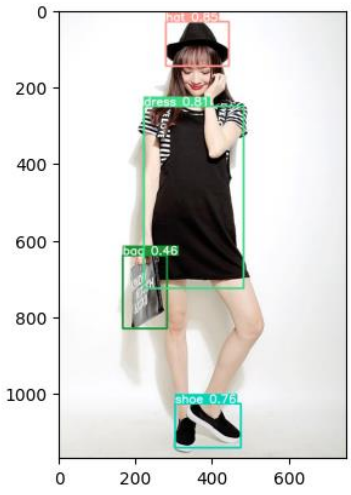

Figura 34. Prueba de dataset con YOLOv5

Mientras que con la versión más nueva detectamos el gorro con un 0.94 de precisión en esta lo hacemos con 0.85. En cuanto al vestido y las zapatillas pasamos de 0.90 a 0.81 y de 0.76 a 0.85 respectivamente. El cambio más grande está en la detección del bolso, donde pasamos de una puntuación de 0.71 a 0.46, este cambio es muy significativo ya

que si limitamos el umbral de detección del modelo es posible que haya objetos como ese que ni siquiera sean detectados.

Aunque las puntuaciones de precisión no son malas pueden llegar a ser determinantes a la hora de resolver el problema y es por ello por lo que, aunque nuestro modelo de Deep Learning no tenga una precisión general muy diferente, hay una diferencia notoria.

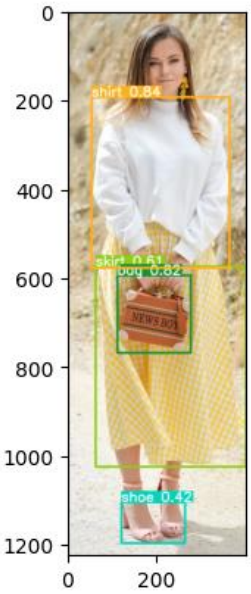

Figura 35. Prueba con imagen externa YOLOv8

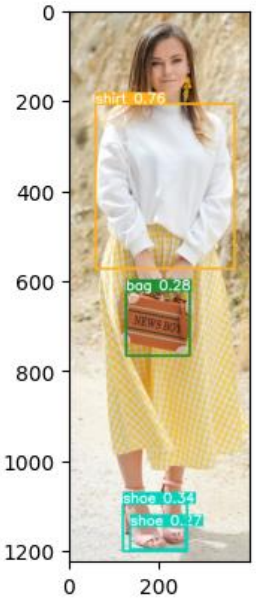

Figura 36. Prueba con imagen externa YOLOv5

Para tener una mejor comparación también hemos realizado dos predicciones de la misma imagen en una imagen totalmente nueva extraída de una página web de uso libre. De esta forma podremos ver como rinden ambos modelos simultáneamente.

Solamente con esta predicción podemos observar que el desempeño es bastante notorio y significante ya que YOLOv8 es capaz de detectar los objetos satisfactoriamente todas las prendas y con una precisión bastante exitosa.

Sin embargo, la quitan versión no es capaz de detectar la falta y se confunde con los zapatos, encontrando dos instancias en una. Viendo estos resultados vemos que, aunque la precisión no sea muy diferente, YOLOv8 cuenta con diferentes mejoras capaces de mejorar bastante su rendimiento y brindar unos resultados bastante exitosos.

Capítulo 5 – Conclusiones

El mundo de la inteligencia artificial está en constante crecimiento y mi interés sobre este aumenta cada vez más con cada cosa que hago relacionada con ello, tanto con la inteligencia artificial en general como el aprendizaje profundo. Todo este trabajo, procesos y resultados me han hecho crecer y mejorar mis habilidades en este campo tecnológico tan interesante y con gran potencial.

En este trabajo, se ha abordado el problema de la detección de prendas de ropa utilizando redes personalizadas YOLO. Este estudio ha permitido comprender y aplicar conceptos fundamentales de aprendizaje profundo y detección de objetos en el contexto particular de la moda y la industria textil.

Se ha entrenado una red neuronal personalizada basada en la arquitectura YOLO durante el desarrollo de este proyecto para detectar prendas de ropa precisas en imágenes. Esto ha requerido la recopilación de un conjunto de datos adecuado que contenga imágenes etiquetadas con las clases de interés, como camisetas, pantalones, faldas, bolsos, zapatos o vestidos. El rendimiento del modelo también se evaluó en términos de velocidad de procesamiento. En comparación con otras técnicas de detección de objetos, la arquitectura YOLO ofrece una gran eficiencia computacional al realizar la detección de objetos en una sola pasada. Esto permite una solución más rápida y en tiempo real para múltiples aplicaciones.

Aunque los hallazgos son positivos, se han identificado algunas áreas en las que se pueden mejorar. Para mejorar aún más la precisión y la capacidad de generalización del modelo, se requiere un conjunto de datos más grande y diverso. La optimización de la arquitectura de la red y los hiperparámetros también puede mejorar el rendimiento.

Por otra parte, se ha podido ver como las versiones de YOLO han ido avanzando y mejorando continuamente, ofreciendo nuevas tecnologías, herramientas y técnicas que permiten obtener un mejor desempeño contando con los mismos parámetros y datos. Además de mejorar la comodidad y flexibilidad del algoritmo, ofreciendo una mayor accesibilidad y posibilidades de entrenar nuestro modelo, realizar predicciones y más.

5.1 Aceptación de la IA en la detección de objetos

A medida que la tecnología se ha vuelto más importante en varios aspectos de nuestras vidas en los últimos años, la aceptación pública de la inteligencia artificial (IA) ha aumentado significativamente. La IA tiene muchos beneficios en áreas como la medicina, la agricultura, la educación y la seguridad. Estos beneficios incluyen la capacidad de facilitar la toma de decisiones, la automatización de tareas repetitivas y la mejora de la eficiencia y precisión de los procesos. Estos avances han aumentado la aceptación de la IA en muchos sectores.

Sin embargo, la aceptación pública de la IA también ha generado preocupaciones y desafíos. Uno de los miedos más comunes es el riesgo de perder empleos como resultado de la automatización. Es preocupante que los avances en la robótica y la IA lleven a la

sustitución de humanos por máquinas, lo que podría generar incertidumbre sobre el futuro laboral. Por otra parte, la aceptación pública de la IA en la detección de objetos ha sido influenciada por su uso exitoso. Las personas han podido beneficiarse directamente de los sistemas de detección basados en IA a medida que se utilizan en aplicaciones del mundo real, como la detección de peatones en vehículos autónomos o la detección de objetos en sistemas de seguridad. El público ha aceptado los sistemas de IA porque pueden mejorar la seguridad, optimizar la eficiencia y proporcionar información importante. Aunque este proceso sea un proceso dinámico y bastante complejo que depende de una gran variedad de factores, también aporta grandes ventajas en muchos sectores.

En resumen, la sociedad acepta la IA para la detección de objetos debido a las mejoras de rendimiento y las aplicaciones exitosas de los sistemas basados en IA. Sin embargo, es necesario abordar las cuestiones éticas y garantizar la transparencia y la regulación adecuada para mantener la confianza pública y garantizar el uso responsable de la inteligencia artificial en la detección de objetos.

5.2 Futuras mejoras

Este proyecto puede contener varias mejoras en el futuro. Principalmente en un proceso de entrenamiento de redes neuronales el conjunto de datos es muy importante y en este proyecto, los datos, aunque han sido suficientes podrían ser mucho mejor. Obtener más datos de entrenamiento etiquetados puede conllevar una gran mejora en el rendimiento del modelo, esto se puede hacer manualmente siempre y cuando se utilicen imágenes de la misma resolución y estén etiquetadas con el mismo número de clases.

La familia de algoritmos YOLO va avanzando y obteniendo mejoras con el paso del tiempo, es por ello por lo que en el futuro esta tecnología recibirá nuevas versiones con novedades y mejoras para el proyecto respecto a la octava versión de YOLO con la cual se ha realizado el trabajo.

También YOLO nos permite la detección, clasificación y segmentación de objetos en video, lo que se podría ampliar el proyecto, utilizando el mismo modelo para detectar prendas de ropa tanto en videos almacenados como en videos en tiempo real a través de la cámara de un dispositivo.

Bibliografía

[1] Martínez, Jesús (2020). ¿Qué es la detección de objeto? https://datasmarts.net/es/que-es-la-deteccion-de-objetos/

[2] Jocher, G., Chaurasia, A., & Qiu, J. (2023). YOLO by Ultralytics (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[3] Data Science Team (2020) Redes Neuronales residuales – Lo que necesitas saber (ResNet) https://datascience.eu/es/aprendizaje-automatico/una-vision-general-de-resnet-y-sus-variantes/

[4] Amazon ¿Qué es una red neuronal? https://aws.amazon.com/es/what-is/neural-network/

[5] López, R. F., & Fernández, J. M. F. (2008). *Las redes neuronales artificiales*. Netbiblo.

[6] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

[7] Artola Moreno, Á. (2019). Clasificación de imágenes usando redes neuronales convolucionales en Python.

[8] Massiris, M., Delrieux, C., & Fernández Muñoz, J. Á. (2018). Detección de equipos de protección personal mediante red neuronal convolucional YOLO. In *XXXIX Jornadas de Automática* (pp. 1022-1029). Área de Ingeniería de Sistemas y Automática, Universidad de Extremadura.

[9] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern recognition*, *77*, 354-377.

[10] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[11] Jiang, P., Ergu, D., Liu, F., Cai, Y., & Ma, B. (2022). A Review of Yolo algorithm developments. *Procedia Computer Science*, *199*, 1066-1073

[12] Jocher, G. (2020). YOLOv5 by Ultralytics (Version 7.0) [Computer software]. https://doi.org/10.5281/zenodo.3908559

[13] Jocher, Glenn, Waxmann, Sergiu (2023) Train Custom Data YOLOv5 https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/

[14] Maji, D., Nagori, S., Mathew, M., & Poddar, D. (2022). Yolo-pose: Enhancing yolo for multi person pose estimation using object keypoint similarity loss. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 2637-2646).

[15] Liu, S., Feng, J., Domokos, C., Xu, H., Huang, J., Hu, Z., & Yan, S. (2013). Fashion parsing with weak color-category labels. *IEEE Transactions on Multimedia*, *16*(1), 253-265.

[16] Nguyễn Gia Bảo Lê (2022) Colorful Fashion Dataset For Object Detection https://www.kaggle.com/datasets/nguyngiabol/colorful-fashion-dataset-for-object-detection

[17] Rojas, E. M. (2020). Machine Learning: análisis de lenguajes de programación y herramientas para desarrollo. *Revista Ibérica de Sistemas e Tecnologias de Informação*, (E28), 586-599.

[18] Mery, D. (2004). Visión por computador. *Santiago de Chile. Universidad Católica de Chile*.

[19] Cheng, W. H., Song, S., Chen, C. Y., Hidayati, S. C., & Liu, J. (2021). Fashion meets computer vision: A survey. *ACM Computing Surveys (CSUR)*, *54*(4), 1-41

[20] Kanani, P., & Padole, M. (2019). Deep learning to detect skin cancer using google colab. *International Journal of Engineering and Advanced Technology Regular Issue*, *8*(6), 2176-2183

[21] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Anexos

A. Creación de subconjunto de validación

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 11 12:53:58 2023

@author: guill
"""

import os
import math
import random
import shutil



# Rutas de las carpetas de imágenes y etiquetas
images_folder = 'C:/Users/guill/Desktop/TFG/DATSETS/DATASET/data/images/'
labels_folder = 'C:/Users/guill/Desktop/TFG/DATSETS/DATASET/data/labels/'

# Obtener la lista de nombres de archivos de imágenes y etiquetas
image_files = os.listdir(images_folder)
label_files = os.listdir(labels_folder)

# Combinar las rutas de las imágenes y etiquetas
image_paths = [os.path.join(images_folder, filename) for filename in image_files]
label_paths = [os.path.join(labels_folder, filename) for filename in label_files]

# Calcular el número de elementos para el 20% del dataset
num_elements = math.ceil(len(image_paths) * 0.2)

# Obtener una muestra aleatoria del 20% de las rutas
selected_image_paths = random.sample(image_paths, num_elements)
```

```
selected_label_paths = [os.path.join(labels_folder, os.path.splitext(os.path.basename(image_path))[0] + ".txt") for image_path in selected_image_paths]

# Rutas de las carpetas de destino para imágenes y etiquetas
subset_images_folder = 'C:/Users/guill/Desktop/TFG/DATSETS/data/val_subset_images/'
subset_labels_folder = 'C:/Users/guill/Desktop/TFG/DATSETS/data/val_subset_labels/'

# Crear las carpetas de destino si no existen
if not os.path.exists(subset_images_folder):
    os.makedirs(subset_images_folder)

if not os.path.exists(subset_labels_folder):
    os.makedirs(subset_labels_folder)

# Copiar las imágenes y etiquetas seleccionadas a las carpetas correspondientes
for image_path, label_path in zip(selected_image_paths, selected_label_paths):
    # Obtener los nombres de archivo de las imágenes y etiquetas
    image_filename = os.path.basename(image_path)
    label_filename = os.path.basename(label_path)

    # Construir las rutas de destino para las imágenes y etiquetas
    subset_image_path = os.path.join(subset_images_folder, image_filename)
    subset_label_path = os.path.join(subset_labels_folder, label_filename)

    # Copiar las imágenes y etiquetas a las carpetas correspondientes
    shutil.copy(image_path, subset_image_path)
    shutil.copy(label_path, subset_label_path)
```

B. Desarrollo de servidor Flask

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 14 08:48:37 2023
```

```
@author: guill
"""

from flask import Flask, render_template, request
from ultralytics import YOLO
import torch
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
import ultralytics
import random
from PIL import Image

app = Flask(__name__, static_folder='runs')

modelo = YOLO("C:/Users/guill/Desktop/TFG/GIT HUB/TFG_FASHION_DETECTION/FlaskServer/models/best.pt")

def predict(model):

    # Leer la imagen
    image = Image.open("runs/imagen.jpg")

    # Nuevo tamaño de la imagen (ancho, alto)
    new_size = (400, 600)

    # Escala del 50% (mitad del tamaño original)
    factor_escala = 0.5
    nuevo_tamano = (
        int(image.width * factor_escala),
        int(image.height * factor_escala)
```

```
    )
    imagen_escalada = image.resize(new_size)

    # Predecir la imagen
    resultado = modelo.predict(source=imagen_escalada, conf=0.35, save=True, line_thickness=3, show_labels=True)
    return resultado



@app.route('/', methods=['GET'])
def index():
    return render_template('index.html')


@app.route('/upload', methods=['GET','POST'])
def upload():
    if request.method == 'POST':

        # Obtenemos la imagen del HTML
        file = request.files['image']

        # Guarda la imagen en el sistema de archivos
        file.save('runs/imagen.jpg')

        # Llamamos a la función encargada de hacer el predict
        resultado = predict(modelo)

    return render_template('results.html')

if __name__ == '__main__':
    app.run()
```

C. Desarrollo de index en HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>TFG</title>
  <link rel="stylesheet" href="../runs/estilos.css">
</head>

<body>
  <h1>DETECCIÓN DE OBJETOS - YOLOv8</h1>

  <img src="../runs/yolo.png" alt="Foto" style="width: 400px; height:
auto; display: block; margin: 0 auto; border-radius: 10px;">

  <div class="upload-form">

    <form action="/upload" method="post" enctype="multipart/form-data">
      <input type="file" name="image" accept="image/*">
      <input type="submit" value="Subir imagen">
    </form>

   </div>

</body>
</html>
```

D. Desarrollo de la página del resultado

```
<!DOCTYPE html>
<html>
<head>

  <title>TFG</title>
  <link rel="stylesheet" href="../runs/estilos.css">
</head>

<body>

  <h1>RESULTADOS - YOLOv8</h1>

  <img src="../runs/yolo.png" alt="Foto" style="width: 400px; height:
  auto; display: block; margin: 0 auto; border-radius: 10px;">

  <div class="upload-form">

    <img src="../runs/detect/predict/imagen.jpg" alt="Prendas
  detectadas" style="width: 300px; height: auto; display: block;
  margin: 0 auto; border-radius: 10px;">
```

```
    <img src="../runs/detect/predict/image0.jpg" alt="Prendas detectadas" style="width: 300px; height: auto; display: block; margin: 0 auto; border-radius: 10px;">

    </div>

</body>
</html>
```

E. Desarrollo de la página de estilos

```
body {
    font-family: Arial, sans-serif;
    background-color: #f2f2f2;
    margin: 0;
    padding: 20px;
}

h1 {
    color: #333333;
    text-align: center;
    margin-top: 50px;
}

.upload-form {
    max-width: 400px;
    margin: 30px auto;
    background-color: #ffffff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

.upload-form input[type="file"] {
    display: block;
    margin-bottom: 15px;
}

.upload-form input[type="submit"] {
    background-color: #4caf50;
    color: #ffffff;
    border: none;
    padding: 10px 20px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 16px;
    cursor: pointer;
```

```
MMM.     border-radius: 3px;
NNN.     transition: background-color 0.3s ease;
OOO.   }
PPP.
QQQ.   .upload-form input[type="submit"]:hover {
RRR.     background-color: #45a049;
SSS.   }
```

F. Enlace al cuaderno interactivo utilizado

https://colab.research.google.com/drive/1X2rbZS0piqhaY3pYxD8vYxfLSKkUy1kh?usp=sharing