UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Creación de un sistema de reconocimiento de dorsales

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Catalán López, Tomás

Tutor/a: Martínez Hinarejos, Carlos David

CURSO ACADÉMICO: 2022/2023

Resumen

El reconocimiento de dorsales en fotos de carreras se ha convertido en un área de investigación cada vez más relevante debido a la creciente popularidad del *running* como deporte, y a que los corredores y organizadores de carreras desean tener acceso fácil y rápido a las fotos de los participantes en la misma. La tarea de reconocer los números de dorsal de los corredores en fotos de carreras es desafiante debido a las variaciones en la apariencia de los dorsales, el tamaño, la orientación y la iluminación en las fotos. Además, el reconocimiento de dorsales puede tener aplicaciones prácticas en la identificación y seguimiento de atletas, la mejora de la experiencia de los corredores y la seguridad en eventos deportivos. En este Trabajo de Fin de Grado, se propone desarrollar un modelo de reconocimiento de dorsales en fotos de carreras utilizando técnicas de visión por ordenador y aprendizaje automático. Se explorará el uso de diversas arquitecturas de redes neuronales, como *Faster R-CNN* y *YOLO*. Además, se investigará el uso de diferentes técnicas de preprocesamiento de imágenes para mejorar la precisión del modelo.

Palabras clave: Visión por ordenador, redes neuronales, *Deep learning*, OCR, *machine learning*

Abstract

The recognition of race bibs in photos has become an increasingly relevant area of research due to the growing popularity of running as a sport. Runners and race organizers require easy and fast access to participant photos from the race. However, the task of recognizing runner bib numbers in race photos is challenging due to variations in bib appearance, size, orientation, and lighting in photos. Furthermore, the recognition of bibs can have practical applications in athlete identification and tracking, improving the runner experience, and ensuring safety in sporting events. This Bachelor's Thesis proposes the development of a model for recognizing race bibs in photos using computer vision and machine learning techniques. The use of various neural network architectures, such as Faster R-CNN and YOLO, will be explored to achieve accurate detection and recognition of bibs in race images. Additionally, different image preprocessing techniques will be investigated to improve the model's performance.

Key words: Computer vision, Neuronal networks, Deep learning, OCR, Machine learning

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La visión por ordenador es cada vez más importante en la actualidad debido a la enorme cantidad de datos visuales que se generan diariamente en el mundo. Tiene múltiples aplicaciones en el mundo real, desde la detección de objetos y personas en videos de vigilancia hasta la identificación de patologías en imágenes médicas [17]. Además, es una herramienta fundamental para el desarrollo de sistemas de conducción autónoma [18] o robótica [10], por poner un ejemplo. Los avances recientes en el aprendizaje profundo [1] han permitido que la visión por ordenador sea aún más precisa y efectiva en el procesamiento de grandes cantidades de datos visuales.

Este trabajo se centra en la detección de dorsales, pero ajustando el modelo y el entrenamiento realizado, podría llegar a ser utilizado para otros propósitos, como lectura de matrículas de coches, o identificación de logotipos en objetos. A pesar de que este proyecto tiene un objetivo muy concreto, la metodología utilizada puede escalarse y modificarse para propósitos muy diferentes.

1.1 Motivación

Este proyecto es una gran forma de poner a prueba todo lo aprendido durante este curso, y además es un tema en el que tengo un interés personal. A la hora de participar en una carrera, la organización de esta suele subir la imágenes de la carrera en masa, tan solo ordenándolas cronológicamente en la mayor parte de los casos. Esta herramienta podría evitar hacer perder una cantidad considerable de tiempo a las personas que quieren buscarse en estas fotos.

1.2 Objetivos

El objetivo principal del proyecto es el desarrollo de un modelo de reconocimiento de números en dorsales para imágenes de carreras. No obstante, la elaboración de este proyecto es una buena oportunidad para explorar el estado del arte de la visión por ordenador, enfocado a la detección de objetos y lectura de texto, como se desarrollará en el capítulo 2.

Los mayores problemas que tiene el proyecto se deben, fundamentalmente, al entorno utilizado. Muchos pasos se han realizado con la idea de usar Kaggle para entrenar y ejecutar el modelo, como se menciona en la sección 5.8.1. Es posible que con acceso a un equipo más potente y una mayor libertad para usar diferentes herramientas se hubiesen conseguido unos mejores resultados.

1.3 Estructura de la memoria

Este trabajo consta de 7 apartados:

- **Introducción.** En este apartado se presenta de forma general el proyecto, para que se tenga cierto contexto en términos generales.
- **Visión por Ordenador.** Repaso de la visión por ordenador, cómo ha evolucionado los últimos años, y estado del arte.
- **Análisis del problema.** Aquí se elabora el problema del reconocimiento de dorsales, y también se dan detalles de los conjuntos de datos utilizados.
- **Identificación y análisis de soluciones.** Esta sección presenta varias formas de enfocar el problema, así como sus diferentes dificultades y cuál solución se ha decidido utilizar finalmente.
- **Tecnología utilizada.** En este apartado se explica en profundidad las diferentes tecnologías que se han utilizado durante el proyecto.
- **Resultados.** Presentación de los resultados finales.
- **Conclusiones.** Aquí se elabora en más profundidad sobre los resultados obtenidos, principalmente intentando encontrar una explicación para éstos.

Aparte, existen varios anexos con información relevante, aunque no esencial, para este trabajo:

- **Glosario de términos.** Traducciones de anglicismos o terminología técnica que se usa en la memoria.
- **Ejemplos de imágenes de los *datasets* utilizados.** Ejemplos de fotos de los diferentes *datasets* utilizados.
- **Objetivos de Desarrollo Sostenible.**

CAPÍTULO 2
Visión por ordenador

La visión por ordenador es un campo de la informática que se ocupa del estudio de cómo las computadoras pueden ser programadas para interpretar imágenes y videos. Esto incluye el desarrollo de algoritmos y técnicas para el procesamiento de imágenes, el análisis de imágenes y el reconocimiento de objetos.

La visión por ordenador se utiliza en una variedad de campos, como la robótica, la automoción, la seguridad, la medicina y la industria [17]. Uno de los desafíos más importantes en visión por ordenador es desarrollar sistemas que puedan funcionar en entornos cambiantes y con una gran variedad de objetos y escenas.

Los algoritmos utilizados en visión por ordenador incluyen el procesamiento de imágenes, el aprendizaje automático, el análisis estadístico y la optimización [40]. Los sistemas de visión por ordenador modernos combinan estas técnicas para lograr un rendimiento óptimo.

2.1 Detección de objetos

Para este proyecto la detección de objetos es más relevante. Es similar a la localización de objetos, pero, a diferencia de ésta, no solo está pensada para localizar un solo objeto en una imagen, si no que en ésta puede haber uno, varios, o ninguno. Esto plantea una serie de problemas, como que no podemos tener un *output* fijo, ya que la imagen puede contener, por ejemplo, un dorsal o cinco. Aparte de esto, el modelo tiene que devolver no solo la localización del objeto, sino también decir qué tipo de objeto es.

Otro problema es la resolución de las imágenes. En el caso de MNIST [31], las imágenes son de 28x28 píxeles, por lo que un perceptrón de varias capas puede obtener buenos resultados. Sin embargo, para la detección de objetos normalmente se trabaja con imágenes de mayor resolución, lo que aumenta el coste computacional.

Una de las opciones que se plantean es aplicar una ventana deslizante o *Sliding Window*. Esta técnica consiste en ir realizando recortes a la imagen, aplicar clasificación de imágenes al recorte y ver si contiene algún objeto. El problema con este método es el coste computacional, ya que para una imagen de anchura W y altura H existen un total de $\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1)$ opciones. En la anterior fórmula w y h hacen referencia a todas las posibles combinaciones de anchura y altura que puede tener cada región.

2.2 Estado del arte

En los últimos años se han desarrollado varios algoritmos de detección de objetos que han logrado un rendimiento muy elevado en tareas de detección de objetos. Algunos de los algoritmos más destacados incluyen las redes neuronales convolucionales de regiones (RCNN) y *You Only Look Once* (YOLO).

2.2.1. R-CNN

Las R-CNN, redes convolucionales basadas en regiones, parten de unas regiones de interes que pueden ser el *output* de otro modelo, se reescalan, y luego aplican el método usado en AlexNet[2], una red convolucional que clasifica esa imagen. Para obtener mejores resultados, también se predice la localización del objeto, por si la primera región no se ajusta perfectamente al objeto. En la figura 2.1 se puede ver un esquema de su funcionamiento

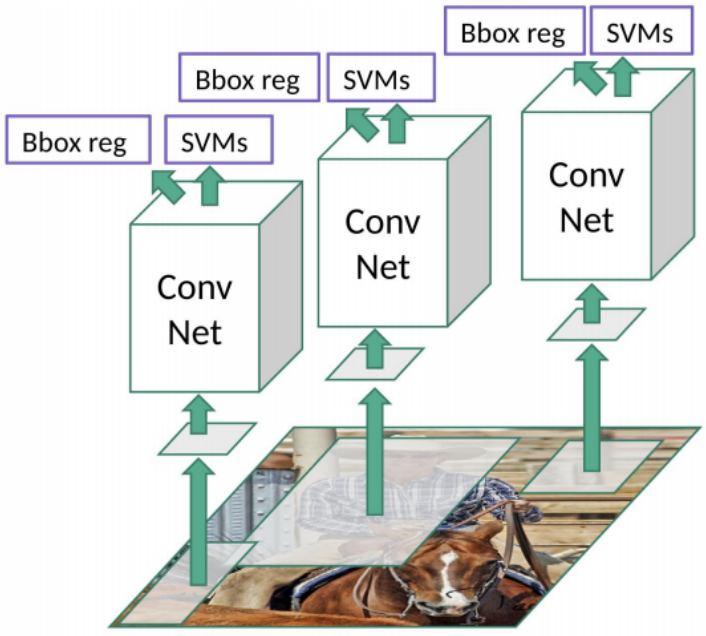


Figura 2.1: Funcionamiento de una red R-CNN.

A lo largo de los años se ha investigado en modelos cada vez más rápidos y precisos, empezando por R-CNN [11], más tarde llegó *Fast-RCNN* [12], y por último *Faster-RCNN* [39].

Como su nombre indica, *Fast-RCNN* fue un salto en cuestión de velocidad respecto a su anterior versión, principalmente al procesar toda la imagen en una sola red neuronal. Aparte combina la clasificacion de objetos y la regresión de las coordenadas en un solo modelo, lo que reduce la complejidad del mismo. *Faster-RCNN* por otro lado implementa un sistema para proponer y clasificar regiones en la imagen, lo cual mejora la velocidad y precisión del modelo.

2.2.2. YOLO (*You Only Look Once*)

YOLO, el acrónimo de "You Only Look Once", es un modelo de reconocimiento de objetos en tiempo real. La versión original del modelo se publicó en el año 2015 [38], y

actualmente van por la séptima versión de éste: YOLOv7 [48]. Todavía no se ha publicado un artículo para esta versión en concreto, pero existe de ella información en el repositorio oficial de Github. Para este trabajo en concreto se ha utilizado la quinta versión de YOLO, YOLOv5 [1].

Funcionamiento

El acercamiento de YOLO al problema de detectar una cantidad indeterminada de objetos en una imagen es el siguiente: primero divide la imagen en una cuadrícula, y procesa cada sección por separado, como en el diagrama de la figura 2.2. Esto genera un conjunto de *bounding boxes*, con una clase y una confianza asociada.

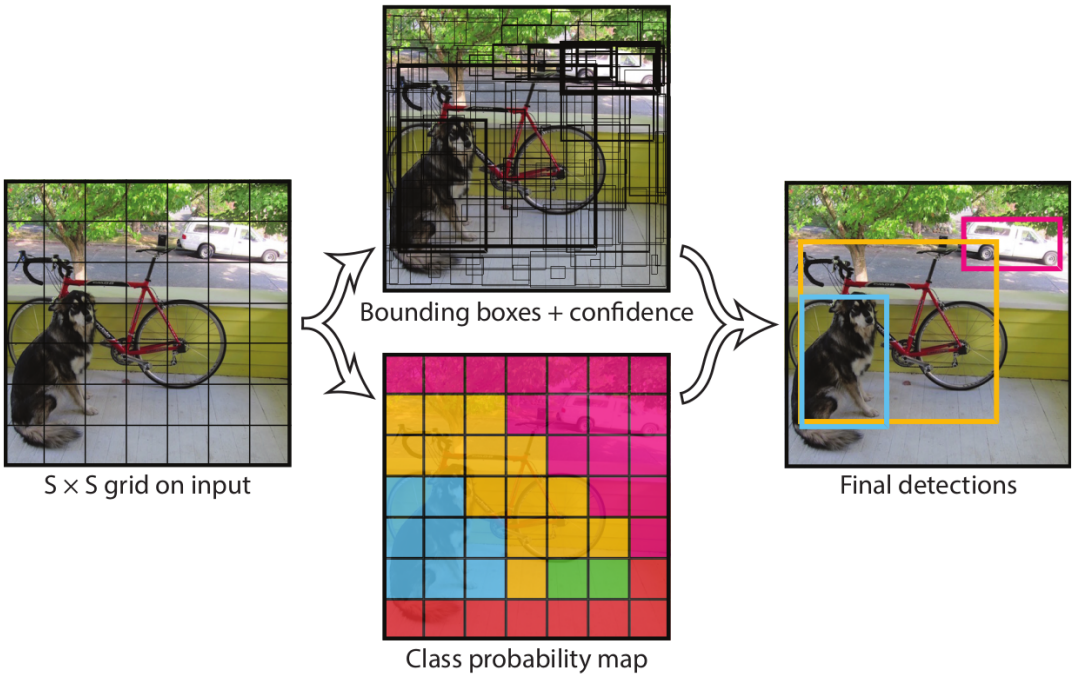


Figura 2.2: Proceso de detección de YOLO.

Este proceso genera una gran cantidad de predicciones, así que la forma en que el modelo trata con ellas es con el método de la intersección sobre la unión (*Intersection over Union*, IoU). Se calcula de la siguiente manera:

$$IoU = \frac{Area_{intersection}}{Area_{union}}$$

En la figura 2.3 se puede observar de forma más visual esta fórmula: teniendo dos áreas diferentes, en este caso las cajas verde y roja, $Area_{intersection}$ es el área que estas dos comparten. Por otra parte, $Area_{union}$ es el área que las dos ocupan en total. $Area_{union}$ puede calcularse como la suma de las áreas menos su intersección.

En el caso de tener tres *bounding boxes* que se solapen para el mismo objeto, no se puede simplemente coger la que tenga la mayor confianza para esa clase, ya que puede haber otros objetos, y eso eliminaría otras predicciones. La solución que plantearon los autores de YOLO consiste en agrupar las *bounding boxes* según el valor de IoU que comparten. Una vez agrupadas, seleccionan solo la de mayor confianza.

Figura 2.3: Cálculo de IoU para dos predicciones.

Arquitectura

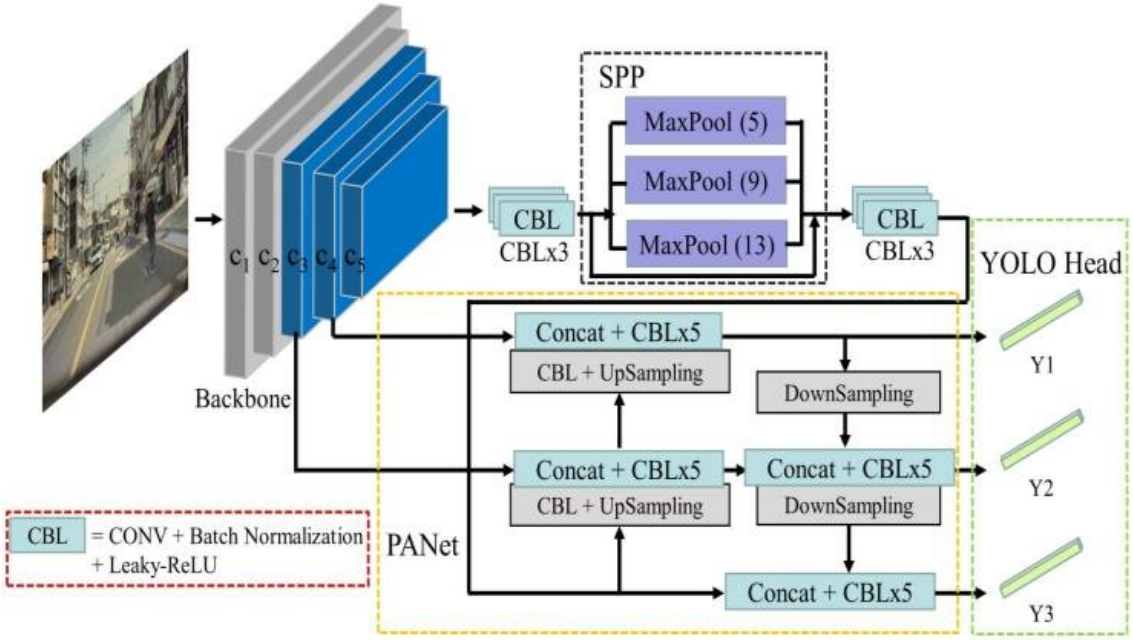


Figura 2.4: Arquitectura YOLOv5.

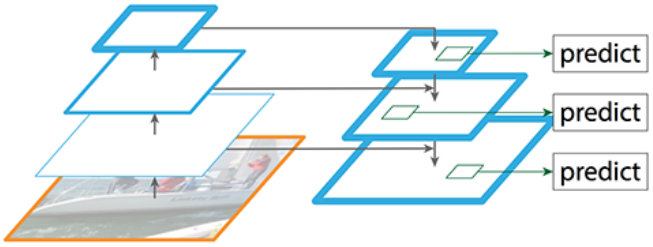


Figura 2.5: Funcionamiento de redes piramidales.

La arquitectura de YOLOv5 (vease la figura 2.4) se caracteriza por su capacidad para detectar objetos de manera rápida y precisa.

Para lograr esto, se basa en una arquitectura de red neuronal EfficientNet [45] como *Backbone*, que mejora la eficiencia computacional mientras la precisión se mantiene al mismo nivel o incluso superior. Además, se utilizan módulos de atención en el *Neck* para prestar atención a las áreas de la imagen que contienen objetos, lo que ayuda a mejorar la precisión de la detección. En el *Head* se utiliza una cuadrícula adaptativa para detectar objetos de diferentes tamaños con mayor precisión. Además, se utilizan técnicas de *data augmentation* para mejorar la capacidad de la red neuronal para generalizar a nuevas imágenes.

En conjunto, todas estas características permiten a YOLOv5 detectar objetos de manera rápida y precisa. Pero lo que realmente lo diferencia es el uso de un enfoque de pirámide espacial (vease la figura 2.5), donde se procesan diferentes escalas de la imagen a través de múltiples capas conectadas en serie, lo que permite detectar objetos de diferentes tamaños en la imagen. Esto es fundamental para detectar objetos pequeños o lejanos en una imagen.

2.2.3. Otras técnicas

En los últimos años se ha experimentado con el uso de *transformers*, una clase de modelos de lenguaje desarrollados por Google [47], para la detección de objetos en imágenes. Los *transformers* se basan en la atención, un mecanismo que permite a la red neuronal prestar atención a diferentes partes de la entrada, lo que les permite aprender patrones complejos. Esto los convierte en una opción atractiva para la detección de objetos, ya que las imágenes suelen tener características que se distribuyen de manera desigual.

En el campo de la detección de objetos, se han propuesto varias arquitecturas basadas en *transformers*. Un ejemplo es DETR [6], una arquitectura que utiliza un *transformer encoder-decoder* para detectar objetos en imágenes. La arquitectura se entrena de manera *end-to-end* y utiliza una técnica de atención global que permite a la red neuronal prestar atención a todas las partes de la imagen. El uso de un *transformer* permite a DETR aprender patrones complejos y detectar objetos de formas variadas, lo que ha llevado a resultados prometedores en diferentes conjuntos de datos de detección de objetos [52] [6] [47] [53]. Sin embargo, es un tipo de modelo bastante reciente, por lo que aún se requiere de más investigación para evaluar en profundidad su rendimiento y compararlo con los modelos tradicionales.

CAPÍTULO 3
Análisis del problema

El reconocimiento de dorsales (*BIBs*) es un problema que, como ya se ha mencionado, no se ha trabajado demasiado, o por lo menos con demasiado éxito. Esto puede deberse a la propia naturaleza del problema: el reconocimiento de dorsales consiste en detectar una pequeña region de texto en una imagen, que frecuentemente se encuentra en diferentes posiciones, torcida, o directamente tapada por un obstáculo como puede ser una mano o una persona.

A esto se le suma la propia naturaleza de las fotos con las que trabajamos. En ocasiones las imágenes se consiguen desde *frames* de video [14], pero incluso en fotografías, como las personas están corriendo es fácil que estén ligeramente borrosas. Para finalizar, los dorsales en sí tienen una superficie ligeramente reflectante, por lo que, dependiendo de cómo les refleje la luz, pueden resultar ilegibles.

3.1 Conjunto de datos

Set	Nº de imágenes	Nº de dorsales	Resolución
Set 1	92	100	342x512 - 480x720 px
Set 2	67	77	800x530 - 850x1260 px
Set 3	58	113	768x1024 px
Total	217	290	

Tabla 3.1: Estructura del *RBNR dataset*.

Hay que comenzar diciendo que no existe un gran conjunto de datos que sirva como referencia para el reconocimiento de dorsales. Lo más parecido es el conjunto del artículo RBNR [36], referenciado de ahora en adelante como RBNR *dataset*, que está formado por 217 imágenes divididas en tres grupos. Estas imágenes están anotadas con las coordenadas de los dorsales presentes y sus respectivos números. En la tabla 3.1 está la distribución exacta.

Aparte, se ha usado el conjunto de datos del artículo [14], referenciado de ahora en adelante como TGCRbNW *dataset*, cedido para realizar este trabajo. Este conjunto de datos está formado por 2530 imágenes en las que hay presentes 3232 dorsales, divididos en cinco grupos (RP1, RP2, RP3, RP4 y RP5). En la tabla 3.2 está la distribución exacta. Las imágenes de este *set* están sacadas de *frames* de vídeo de la competición, con un tamaño de 1920x1080 píxeles [14].

Una característica que diferencia el TGCRbNW *dataset* del RBNR *dataset* es que, aparte del tamaño del segundo, este último tiene imágenes más borrosas, y las caras de las per-

Set	Nº de dorsales	Nº de imágenes
RP1	1522	1033
RP2	706	637
RP3	478	407
RP4	241	209
RP5	285	244
Total	3232	2530

Tabla 3.2: Estructura del *dataset TGCRbNW*.

sonas están censuradas. Esto último tuvo relevancia en descartar el método de estimación de torso, ya que RetinaFace [7] no podía reconocer las caras.

CAPÍTULO 4
Identificación y análisis de soluciones posibles

4.1 OCR en la imagen completa

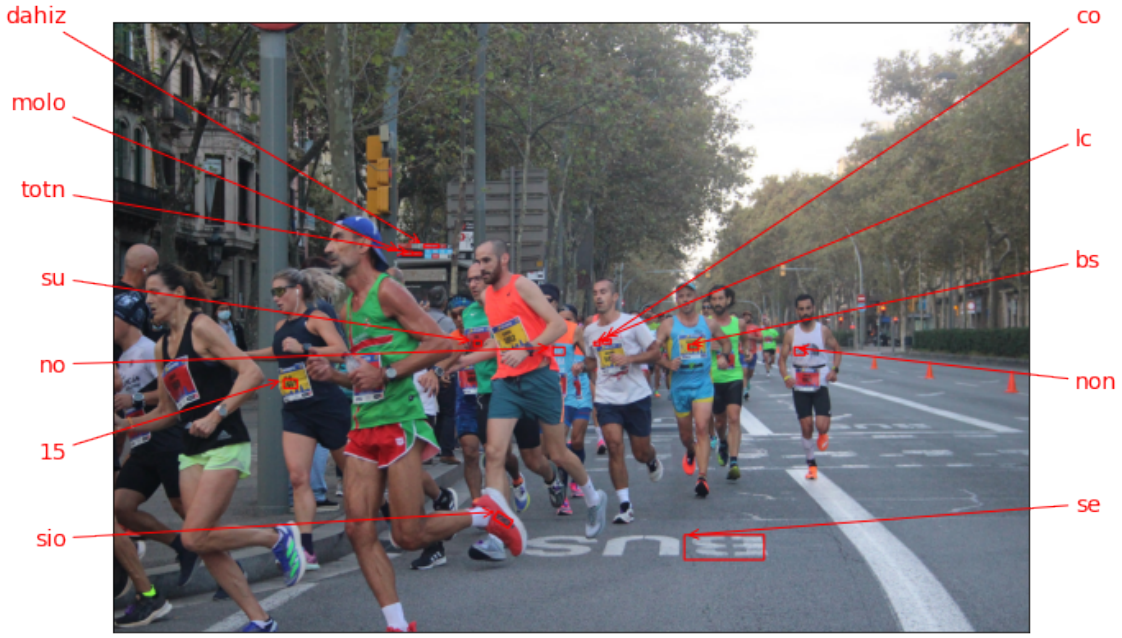


Figura 4.1: Reconocimiento de texto en toda la imagen.

La opción más directa para resolver el problema sería aplicar reconocimiento de texto a la imagen completa, pero esta solución no es desde luego óptima. Como podemos observar en la figura 4.1, el modelo detecta texto por toda la imagen (logos en camisetas, señales de tráfico, etc.), por lo que no resulta muy fiable. Este enfoque conllevaría un posterior tratamiento de los resultados para determinar si un texto corresponde a un dorsal o no.

4.2 Estimación de torso

En el artículo [36] plantean una solución interesante: primero se pasa la imagen a un modelo que detecta las caras en la imagen, y a partir de ahí se estima la región en la que

debería estar el torso. Una vez se obtiene la región del torso, se aplica el reconocimiento de texto utilizando una técnica llamada *Stroke Width Transform* (SWT), para "limpiar" la imagen y dejar (mayoritariamente) solo el texto.

Existen varios problemas con este método. El primero es que estamos estimando la region del torso, por lo que muchas veces, sobre todo si la persona está ligeramente girada, esta estimación suele crear una región erronea. En el RBNR *dataset* no es problema, pues todas las imágenes están relativamente bien posicionadas, pero en las fotos de la media maratón de Barcelona de 2021, un caso real, suele dar problemas. Aparte de esto, las imágenes del TGCRbNW *dataset* tiene las caras de los corredores censuradas, por lo que este método suele fallar. Con todo esto, este método acaba descartado a favor de uno más general.

(a) La región estimada en la persona de la izquierda no muestra por competo el dorsal.

(b) Al tener la cara pixelada, el modelo no puede reconocerla.

Figura 4.2: Casos de error de RetinaFace.

En la figura 4.2, se puede ver los dos fallos más comunes de RetinaFace: una mala estimación del torso, y no detectar una cara cuando está pixelada.

4.3 Solución propuesta

Al final usamos una forma de trabajo similar a la propuesta en el artículo [14]: a partir de una imagen, usamos el modelo YOLOv5s para detectar personas. El modelo base, que se puede importar desde Tensorflow [46], está preparado para detectar diversos objetos,

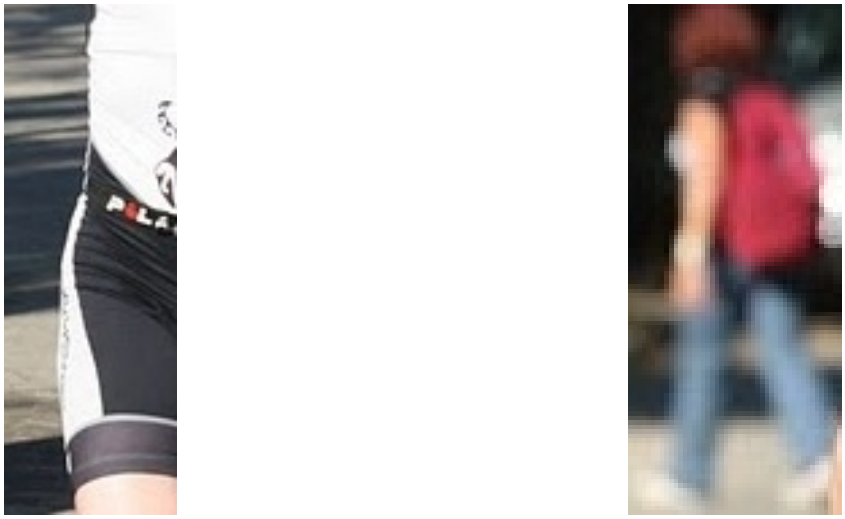


Figura 4.3: Falsos positivos al detectar personas.

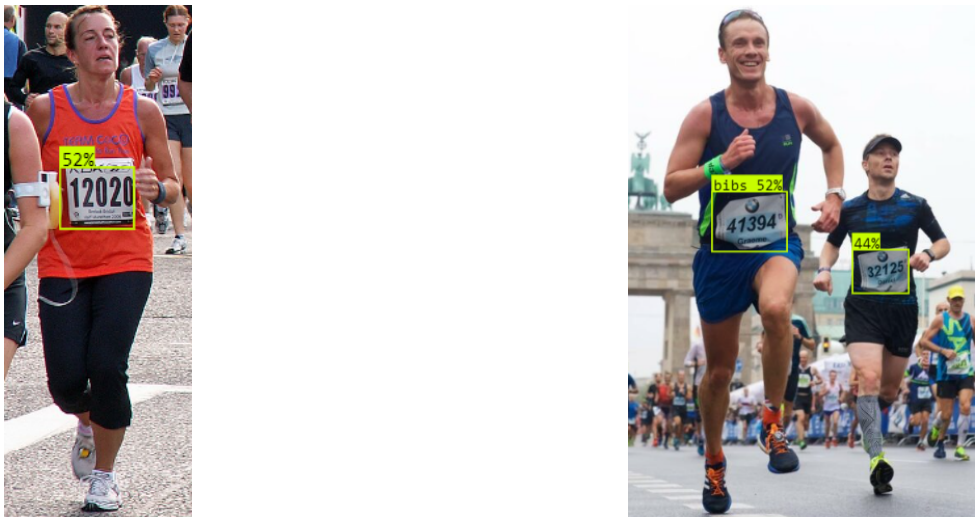


Figura 4.4: Reconocimiento de dorsales en dos fotos de internet.

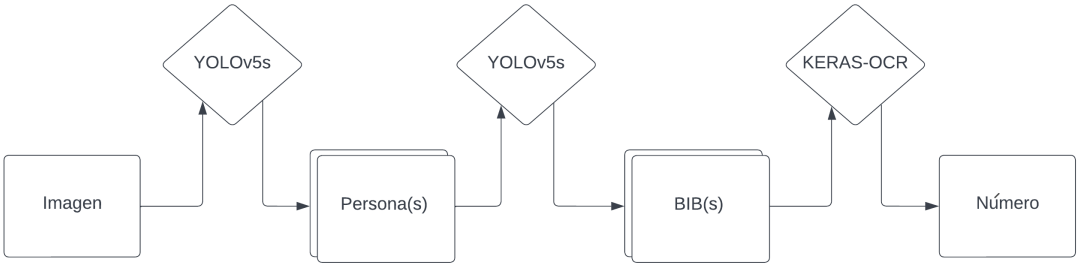


Figura 4.5: Flujo de trabajo.

entre ellos personas, por lo que es cuestión de filtrar los resultados, además de aplicar un *threshold* para mayor fiabilidad, devolviendo solo las regiones que tengan un mínimo de confianza. Aun con este *threshold* en ocasiones se devuelven falsos positivos, como personas en el fondo que no son corredores o, en ocasiones más raras, partes del cuerpo (figura 4.3). Aun teniendo falsos positivos, el modelo no falla en detectar personas.

En el siguiente paso volvemos a aplicar el modelo YOLOv5s, pero en este caso una versión entrenada en un conjunto de datos propio, usando imágenes del RBNR, la media maratón de Barcelona de 2021 y el TGCRbNW [14]. Para este entrenamiento se utiliza la página Roboflow (https://app.roboflow.com/), que ofrece soporte para el modelo YOLOv5, aplicando *data augmentation* a las imágenes. De esta forma conseguimos un modelo de YOLOv5s específicamente entrenado para reconocer dorsales. Una particularidad de este modelo es que, al estar entrenado con diferentes formatos de dorsales, y en el paso de *data augmentation* al haberse usado técnicas de cambios de color, contraste y luminosidad, es capaz de generalizar dorsales con bastante éxito (en la figura 4.4 aplicamos el modelo a imágenes aleatorias de Google). En una primera versión del código, un modelo bastante común era *EfficientNet* [45] para estimar la región del dorsal, pero el rendimiento de este modelo, así como la capacidad de realizar modificaciones, era bastante peor en comparación a YOLO.

Por último, las imágenes de los dorsales se pasan al modelo de lectura de texto. Se ha elegido el modelo Keras-OCR [23], ya que ofrece la posibilidad de entrenar un modelo y, además, es relativamente fácil de manejar y modificar. Más adelante se entrará en más detalles, pero en resumen, en la versión final se utiliza el detector de texto por defecto, y un lector de texto entrenado por mí mismo. Se puede ver un esquema del flujo de trabajo en la figura 4.5, y un ejemplo en la figura 4.6

Por último, mencionar que todas las imágenes generadas son guardadas para facilitar una posterior revisión manual en caso de que fuese necesario. La forma en la que se hace es primero añadir bandas negras a la imagen para hacerla cuadrada mientras se preservan las proporciones de la imagen, y después se reescala a un tamaño fijo: al principio 640x640 píxeles, y 512x512 píxeles en la última versión. Una vez todos los módulos han acabado, se generan archivos .csv en los que se almacenan todos los resultados generados.

(a) Imagen original.

(b) Persona detectada.

(c) Dorsal detectado.

(d) Texto detectado.

Figura 4.6: Flujo de trabajo con ejemplos.

CAPÍTULO 5
Tecnología utilizada

En este capítulo se explicarán modelos o herramientas de trabajo usados para este proyecto.

5.1 Kaggle

Kaggle [19] es una comunidad *online* dedicada a la ciencia de datos y *machine learning*. En ella se pueden encontrar competiciones que usuarios o compañías publican, en ocasiones con premios monetarios. Además de esto, Kaggle ofrece acceso a un entorno en la nube, donde la gente puede subir y ejecutar su código sin tener que ejecutarlo localmente.

Esta es una herramienta bastante valiosa para gente que quiera aprender sobre estos temas, o incluso como herramienta de trabajo, proporcionando un entorno bastante simple de utilizar, así como acceso a bases de datos de la comunidad y librerías, sin la necesidad de tener el equipo necesario para ello. La elección de usar Kaggle viene dada sobre todo por la familiaridad de uso, así como la posibilidad de usar una unidad de procesamiento gráfico (GPU), una herramienta indispensable para entrenar redes neuronales de alta complejidad en un tiempo razonable.

5.2 PyTorch

PyTorch [34] es un *framework open-source* basado en la librería Torch. Se puede utilizar para tareas de *machine learning* y está disponible en Python y C++.

PyTorch tiene una arquitectura de diseño basada en tensores, lo que permite una fácil implementación de operaciones en paralelo en GPU. En cuanto a la detección de objetos, PyTorch ofrece una gran cantidad de herramientas y recursos para desarrollar y entrenar modelos de detección de objetos, incluyendo la capacidad de importar modelos preentrenados de redes populares como YOLOv5 y R-CNN. Además, PyTorch cuenta con una amplia comunidad de desarrolladores y usuarios, lo que significa que hay un gran número de recursos y soluciones en línea disponibles para resolver problemas y mejorar modelos.

Los dos modelos principales usados en este proyecto (además de EfficientNet) utilizan modelos de PyTorch.

5.3 Roboflow

Roboflow [42] es un *framework* que permite a los usuarios configurar un conjunto de datos, etiquetarlo y añadir pasos para realizar *data augmentation*. Proporciona herramientas para facilitar el procesamiento, anotación y almacenamiento de datos de visión, así como una interfaz de programación de aplicaciones para la integración en flujos de trabajo existentes. Esto incluye la compatibilidad con varios *frameworks* populares de visión por ordenador, como TensorFlow [46] y PyTorch [34], lo que permite a los usuarios utilizar sus modelos favoritos con facilidad.

Además, ofrece una interfaz de usuario para la visualización y análisis de datos, lo que facilita el proceso de mejora y depuración de modelos.

5.4 OpenCV

OpenCV [15] es una librería *open-source* con código para los lenguajes de Python, C++, Java y Matlab. Esta librería se utiliza ampliamente en aplicaciones de visión por ordenador, como detección de objetos, seguimiento de objetos, análisis de imágenes y visión 3D. OpenCV proporciona una amplia variedad de algoritmos y herramientas para el procesamiento de imágenes y vídeos.

5.5 EfficientNet

EfficientNet es un modelo de clasificación de imágenes [44] y detección de objetos [45]. Su principal característica es la poca cantidad de parámetros que contiene si lo comparamos con modelos como ResNet [13]. Esto permite modelos que trabajan más rapido, y con resultados a la par de modelos mucho más complejos [44]. En el artículo además se plantean métodos de escalado (por anchura, profundidad y resolución) para modelos que mejoran el rendimiento de éstos.

Su reducido tamaño lo hace perfecto para entrenar en Kaggle, ya que es importante tener la limitación de memoria en cuenta. Otros modelos pueden ser entrenados también, pero se necesitan técnicas avanzadas para evitar esa limitación.

En nuestro caso, se intentó entrenar el modelo EfficientNet-B0, el más básico, para la detección de dorsales, y aunque en el conjunto de datos de entrenamiento daba buenos resultados, a la hora de probarlo en un caso real no llegó a funcionar.

5.6 RetinaFace

RetinaFace es un modelo publicado en 2018 [7], en el que los autores alcanzaron rendimiento de estado del arte. Este modelo es capaz de detectar si una imagen es una cara o no, detectar caras en una imagen, devolver una lista con puntos importantes de referencia de la cara (ojos, nariz y boca), y, por último, devuelve un mapa 3D que puede ser utilizado para generar un renderizado 3D de la cara.

En este caso nos interesa solo la localización de la cara, que utilizamos para la estimación del torso [36]. Para estimar la región del torso, calculamos una región de $2w$x$3h$, siendo w el ancho de la región de la cara, y h su altura. En la figura 5.1 podemos ver los resultados.

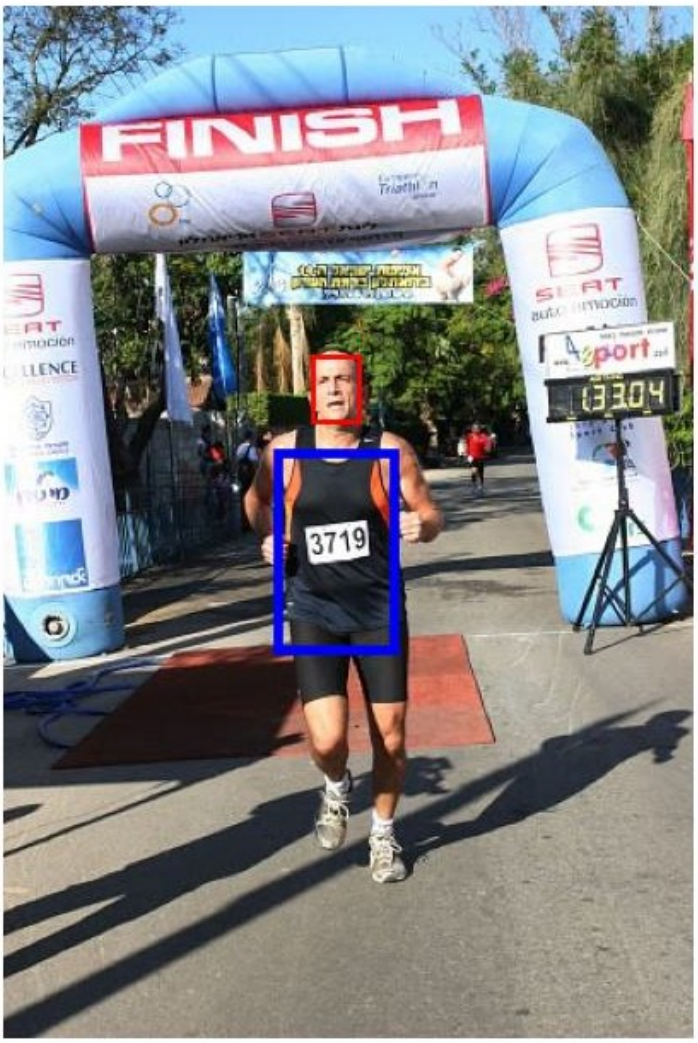


Figura 5.1: Estimación del torso dada la cara. En rojo la localización de la cara. En azul el torso estimado.

5.7 Modelos usados

5.7.1. Modelo para la detección de personas

El modelo de YOLO que se puede conseguir desde Tensorflow por defecto puede diferenciar hasta 80 clases de objetos, entre ellos personas, coches, bicicletas, etc. Por razones obvias, a nosotros solo nos interesan las personas, por lo que filtramos solo esos resultados.

En el artículo [50] los autores crean una solución en la que me inspiré para realizar mi modelo: utilizan YOLO (YOLOv3 en este caso) para buscar corredores, dorsales y números, y luego otro modelo para leer texto. En este caso los autores entrenan un modelo con su conjunto de datos para detectar corredores, pero consiguen una precisión de, como máximo, el 97 % (87 % al entrenar desde cero), por que hay corredores que no son detectados. El método que se utiliza, por el contrario, devuelve personas que no debería, como espectadores de fondo, pero se considera que es preferible a que no detecte personas.

Figura 5.2: Rango de colores (Hue) de una imagen en grados (º).

5.7.2. Modelo para la detección de dorsales

Para esta parte del modelo se utilizó la herramienta Roboflow para procesar el conjunto de datos. El tutorial en mayor detalle se encuentra aquí: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data.

En total se utilizaron 158 imágenes (con 177 dorsales anotados en total) del conjunto de datos RBNR [36] e imágenes de la media maratón de Barcelona de 2021. La división exacta fueron 109 imágenes de entrenamiento, 37 de validación y 12 de test. Estas imágenes se anotaron manualmente, con la herramienta integrada en la página web, y se utilizaron algunas técnicas de *data augmentation*:

- Ajuste de color a escala de grises (10 % de probabilidad)
- Ajuste de rango de color aleatorio (entre -40º y 40º), figura 5.2
- Ajuste de saturación (entre -25 % y 25 %)
- Ajuste de brillo (entre -25 % y 25 %)
- Ajuste de brillo localizado en la zona a detectar (entre -10 % y 10 %)

Al final el conjunto de datos acabó en 376 imágenes. La propia página especifica que a partir de cada imagen se generan tres, aplicando las técnicas de *data augmentation* mencionadas, pero lamentablemente no se puede acceder al número de dorsales anotados para este conjunto, ni observar las imágenes.

Para más información, los detalles sobre el *dataset* se pueden consultar en mi página personal de Roboflow https://universe.roboflow.com/tocaloupvtfg/bibs-rmyuk/health.

Precisión	Recall	MAP@5	MAP
0.95	0.98	0.97	0.60

Tabla 5.1: Tabla de resultados del modelo de reconocimiento de dorsales.

Cabe destacar la facilidad y rapidez para hacer este proceso, ya que la mayor dificultad radicó en colocar bien las carpetas para evitar errores. La libreta de Kaggle se ejecutó en 7 minutos, y el modelo estuvo entrenando durante 20 *epochs*, alcanzando el máximo en el número 12. Los resultados se encuentran en la tabla 5.1.

Aparte de los buenos resultados en los conjuntos de entrenamiento, este modelo es capaz de generalizar con bastante facilidad incluso en imágenes que no ha visto nunca, como es el caso de la figura 4.4.

Figura 5.3: Ejemplos de uso de Keras-OCR estándar.

5.8 Keras-OCR

Keras-OCR es una API que combina el detector de texto *CRAFT* [4] y la red neuronal CRNN [43]. Se puede encontrar en el siguiente enlace: `https://github.com/faustomorales/keras-ocr`.

Esta API ofrece además modelos preentrenados y la posibilidad de crear un *pipeline* entre estos modelos de forma bastante sencilla. En la figura 5.3 podemos ver que, usando el modelo base, el detector funciona de forma bastante fiable, pero la lectura de texto tiene problemas detectando números.

Esto podría deberse a un motivo muy simple: el lector de texto usa los pesos del modelo que está entrenado en conjuntos de datos compuestos mayoritariamente por letras, no números. Los conjuntos de datos específicos se encuentran en el artículo [4].

Para resolver este problema, Keras-OCR ofrece la opción de, al declarar el lector de texto, especificar un alfabeto o conjunto de símbolos concreto. De esta manera se puede configurar para que solo reconozca en este caso números. Sin embargo hay otro problema: ahora la "cabeza" del lector de texto no concuerda con el alfabeto, por lo que asigna pesos aleatorios. Como podemos ver en la figura 5.4, los resultados son directamente aleatorios.

Por suerte es posible entrenar un modelo de Keras-OCR en Kaggle. Esta es la opción que se decidió utilizar. Un punto positivo es que Keras-OCR ofrece realizar un entrenamiento completamente automatizado, y aunque los resultados no son tan buenos como el modelo existente para leer texto, los resultados obtenidos son satisfactorios. Los detalles de las diferentes versiones del modelo del lector de texto y los parámetros de su entrenamiento están desarrolladas en más detalle en la sección 6.2.1.

5.8.1. Alternativas a Keras-OCR

Por suerte, en el repositorio se puede encontrar una guía para realizar el entrenamiento de ambos, el detector y el lector de texto. Lamentablemente, a pesar de haber probado diferentes formas de entrenamiento, el detector no ha podido ser entrenado con éxito.

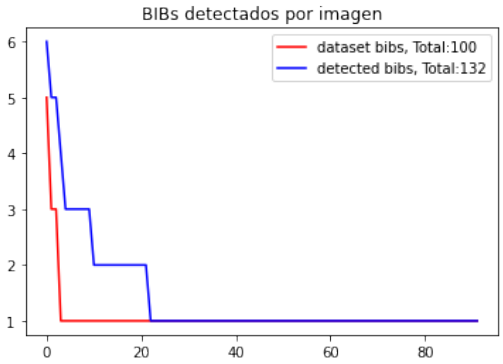
Aparte de Keras-OCR existen otras librerías para el reconocimiento de texto, como PyTesseract [33], pero los resultados de éste no eran muy favorables, además de no dar

Figura 5.4: Pesos aleatorios en la cabeza del lector.

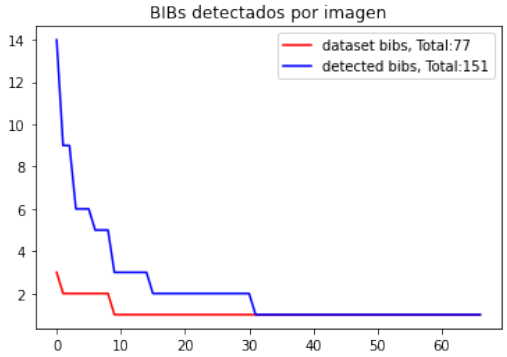
muchas opciones para modificar parámetros. Otra opción podía haber sido entrenar un modelo desde cero, pero PyTesseract no ofrece tantas facilidades de entrenar un modelo como Keras-OCR, por lo que al final decidí usar este último.

CAPÍTULO 6
Resultados

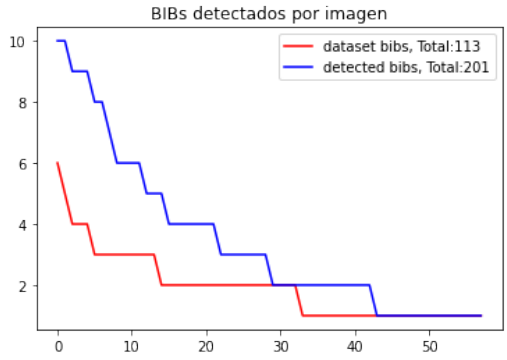
6.1 Reconocimiento de dorsales



(a) *Set* 1.



(b) *Set* 2.

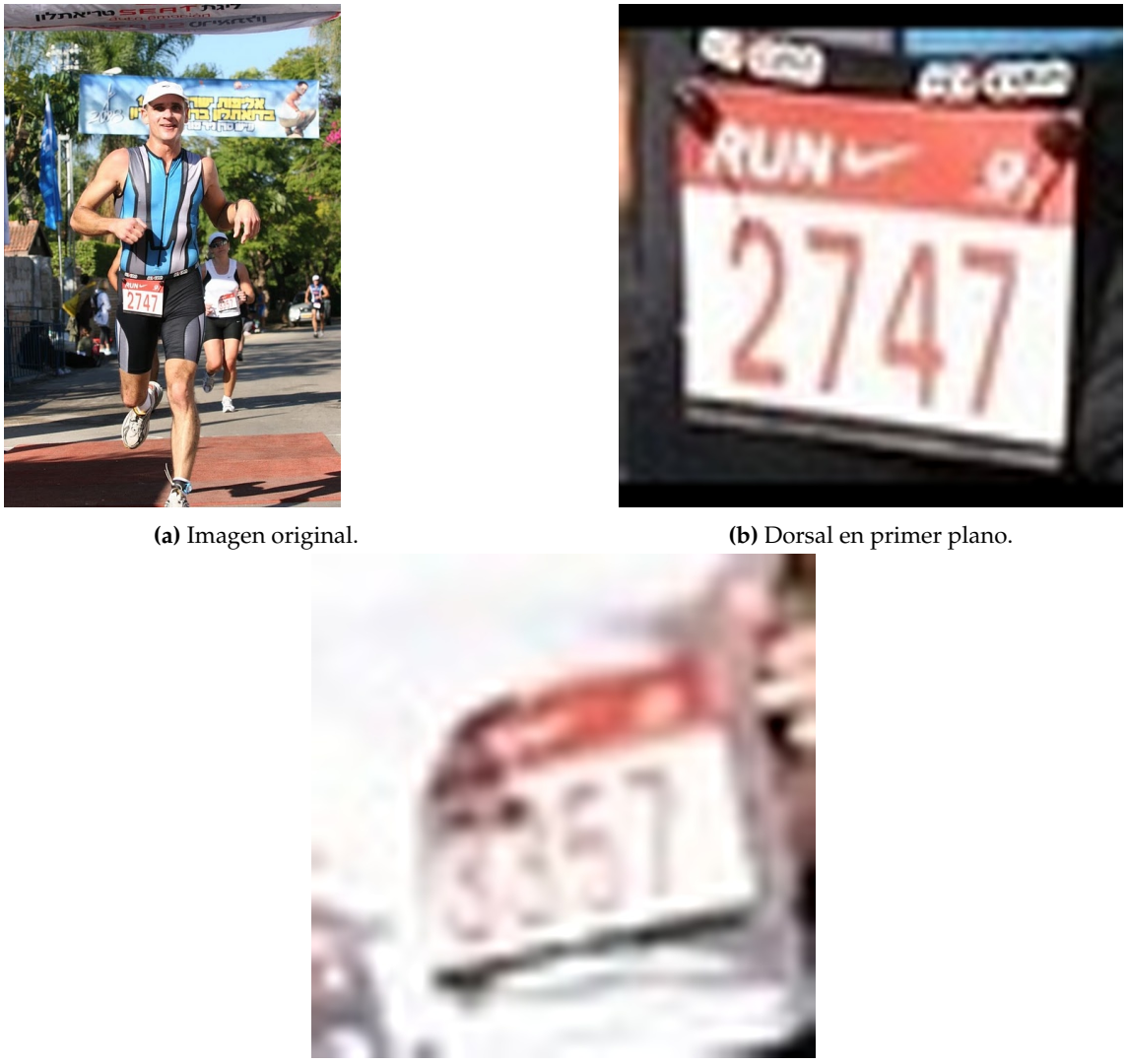


(c) *Set* 3.

Figura 6.1: Diferencia de número de dorsales detectados por *set* del RBNR *dataset*, en contraposicion del número de imágenes anotados originalmente.

En las gráficas de la figura 6.1 podemos observar en rojo el número de dorsales por imagen en el RBNR *dataset*, y en azul el número de dorsales que detecta nuestro modelo por foto. En un principio puede parecer que el modelo detecta demasiados falsos positivos, pero un análisis manual de los resultados revela casos como el de la figura 6.2. En este caso, el único dorsal anotado es el de la persona en primer plano, pero nuestro modelo detecta también el de la persona del fondo.

Con estos resultados como base, el siguiente paso es probar el modelo en el TGCRbNW *dataset*. Al no haber entrenado nuestro modelo en este conjunto de datos, no había datos. La solución fue usar el modelo (sin la parte de reconocimiento de texto para aumentar



(a) Imagen original.

(b) Dorsal en primer plano.

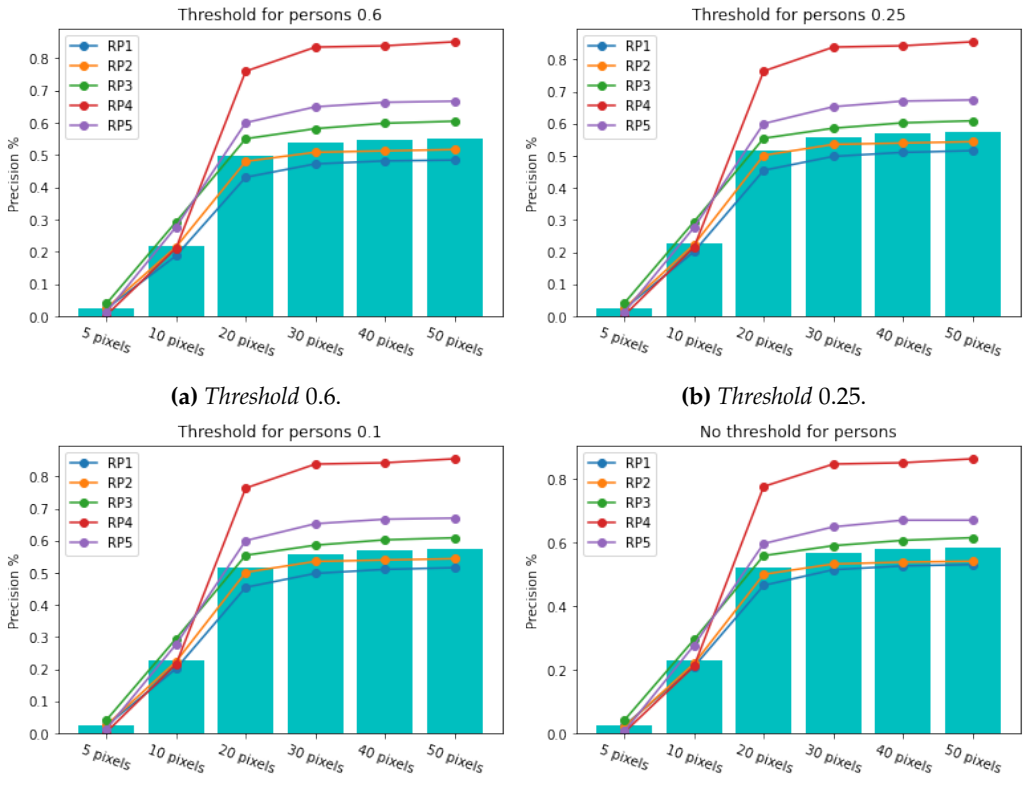
(c) Dorsal de la persona del fondo (No anotado en el *dataset* original).

Figura 6.2: Ejemplo de detección de dorsales en RBNR *dataset*.

la velocidad) en el conjunto de datos, y añadir un margen de error de x píxeles. Si las predicciones del modelo entraban en ese margen, se contaba como reconocido ese dorsal. Aparte de esto, este conjunto de datos tiene imágenes algo borrosas, con personas bastante pequeñas en relación a la imagen, por lo que además de darle un margen a las predicciones, se prueban diferentes *thresholds* para considerar válida una persona detectada (vease sección 5.7.1). En la tabla 6.1 se puede observar la precisión del modelo según el margen (en píxeles) y el mínimo de confianza.

***Threshold*/Margen**	**5p**	**10p**	**20p**	**30p**	**40p**	**50p**	**Tiempo de ejecución**
0.60	0.02	0.21	0.49	0.53	0.54	0.55	0.35 (h)
0.25	0.02	0.22	0.51	0.55	0.56	0.57	0.44 (h)
0.11	0.02	0.22	0.51	0.55	0.56	0.57	0.63 (h)
0	0.02	0.23	0.52	0.56	0.57	0.58	0.7 (h)

Tabla 6.1: Resultados de precisión del detección de dorsales con márgenes de error en el TGCRbNW *dataset* usando diferentes márgenes mínimos de confianza para las personas.



(a) *Threshold* 0.6.

(b) *Threshold* 0.25.

(c) *Threshold* 0.1.

(d) *Threshold* 0.

Figura 6.3: Resultados totales y por grupos en el TGCRbNW *dataset*.

Los resultados de la tabla no difieren los unos de los otros al cambiar el umbral, lo único relevante es el tiempo de ejecución, que aumenta bastante al bajar el *threshold*. Esto tiene sentido, ya que al ser menor hay que comprobar más personas en busca del dorsal en la imagen.

Otro detalle para mencionar es el rendimiento por grupos en el propio *dataset*. En la figura 6.3 podemos ver que el modelo funciona mejor en los conjuntos RP4 y RP5, que son fotos tomadas de día, mientras que conseguimos unos peores resultados en los conjuntos RP1, RP2 y RP3, que fueron tomadas de noche. En el artículo original [14] también se puede observar este patrón.

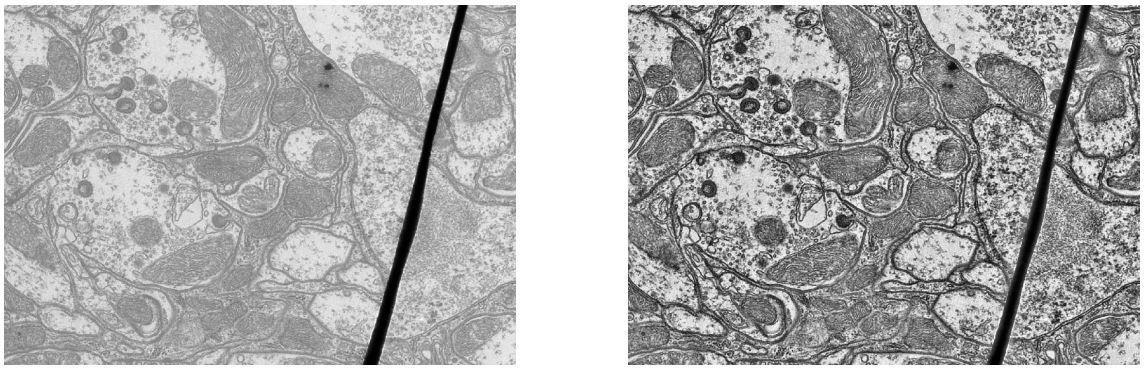
6.2 Reconocimiento de texto

En esta parte de los resultados nos centramos solo en el RBNR *dataset*, ya que las imágenes del TGCRbNW *dataset* se encuentran bastante borrosas, y los resultados conseguidos son bastante malos. Para el reconocimiento de texto le pasamos los dorsales detectados por el modelo de detección de dorsales al modelo de reconocimiento de texto. Este modelo se compone de dos partes: el detector de texto y el lector de texto. El detector de texto es el estándar [4], mientras que el lector de texto es uno entrenado específicamente para la detección de números, partiendo de pesos ya entrenados para acelerar el proceso [43].

En total hay dos lectores de texto: un primero, que fue entrenado según el tutorial, y un segundo detector en el que se usaron técnicas de *data augmentation*: *Gaussian blur* [9], adición de ruido, ajuste de contraste y ajuste aleatorio de color. Otra diferencia notable es que el primer modelo solo entrenó la cabeza del modelo, no las capas de extracción

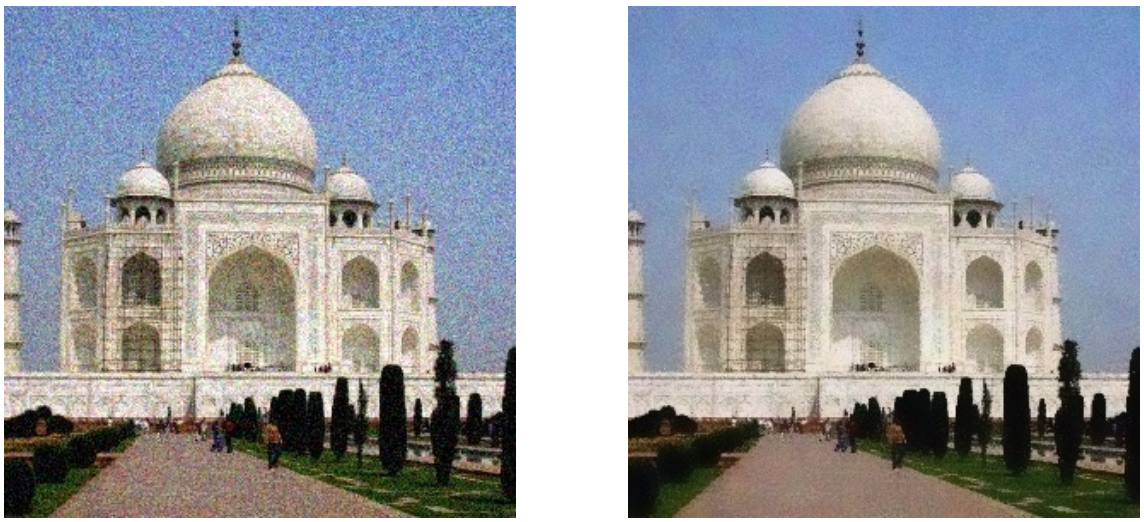
de características, mientras que el segundo fue actualizando todos los pesos del modelo. Al comenzar a usar el primer modelo se experimentó con dos técnicas de preprocesado: *Contrast Limited Adaptive Histogram Equalization* (*CLAHE*) y *Bilateral filtering*, que se explican a continuación.

6.2.1. Preprocesado de las imágenes



(a) Imagen base. **(b)** Imagen después de aplicar *CLAHE*.

Figura 6.4: Preprocesado usando *CLAHE*.

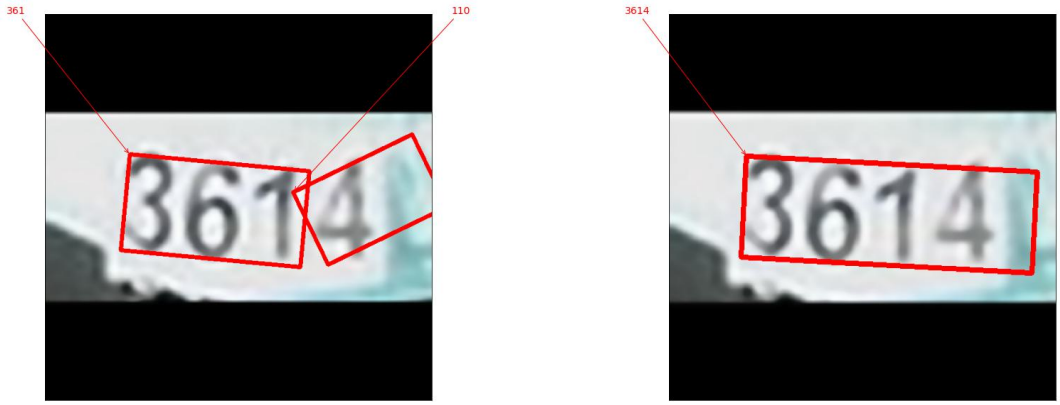


(a) Imagen base. **(b)** Imagen después de aplicar *Bilateral filtering*.

Figura 6.5: Preprocesado usando *Bilateral filtering*.

La técnica de *CLAHE* sirve para aumentar la diferencia de contraste en las imágenes, lo cual en principio parecía una buena idea, ya que habría mejor contraste entre los números y el fondo, como puede verse en la figura 6.4. En la práctica, se pudo observar que usar *CLAHE* de hecho nos da peores resultados. Esto puede deberse a que por defecto convierte la imagen a escala de grises, así como a la aparición de una mayor cantidad de ruido en la imagen. Por tanto, en las versiones finales no se ha usado esta técnica.

La segunda técnica por el contrario sí que nos proporciona una mejora. El *Bilateral filtering* es una forma de eliminar ruido en imágenes, con la ventaja de que mantiene los bordes bien definidos, como se puede ver en la figura 6.5. Como se puede ver en la tabla 6.2, parece que usar *Bilateral filtering* mejora los resultados, por lo que las versiones finales solo usan este preprocesado (a partir de la tercera versión). La tabla muestra el porcentaje



(a) Número mal detectado (640x640). **(b)** Número bien detectado (512x512).

Figura 6.6: Diferentes detecciones de texto segun el tamaño.

de aciertos respecto al total de imágenes. Solo tiene en cuenta si el número leído es el que aparece en la imagen.

Como último detalle, al comprobar manualmente los resultados se podía observar en algunos casos cómo el detector separaba los números en dos. Haciendo pruebas, se puede observar como al aumentar la calidad a la que se reescalan las imágenes de los dorsales, el detector empieza a dar problemas. Teniendo esto en cuenta, la versión final renderiza las imágenes con un tamaño de 512 por 512 píxeles en lugar de 640 por 640, ya que el lector no suele dar el mismo problema, y la imagen tiene una calidad suficiente para ser leída con éxito.

6.2.2. Resultados

A continuación se indica una lista de las versiones para facilitar la lectura de la tabla 6.2.

v1: Primer lector, usando *CLAHE*

v2: Primer lector, usando *CLAHE+Bilateral filtering*

v3: Primer lector, usa solo *Bilateral filtering*

v4: Segundo lector, *data augmentation*

v5: Segundo lector, tamaño de la imagen reducido de 640x640 a 512x512

	v1	**v2**	**v3**	**v4**	**v5**
set_1	25 %	27 %	41 %	52 %	61 %
set_2	22 %	50 %	59 %	58 %	57 %
set_3	15 %	22 %	30 %	33 %	49 %
Total	**20 %**	**31 %**	**42 %**	**46 %**	**55 %**

Tabla 6.2: Tabla de resultados sobre el conjunto de datos RBNR, el *set* 1, *set* 2 y *set* 3 tienen 100, 77 y 113 dorsales repectivamente. Los datos de la tabla son el porcentaje de aciertos sobre los diferentes *sets* de imágenes.

Los resultados anteriores tan solo reflejan los aciertos absolutos del modelo. Por ejemplo, en el caso de que el modelo lea el dorsal **"1112"** siendo el número real **"1111"**, si lo evaluamos de esta manera, sería un fallo del modelo, aunque haya leído bien tres de cuatro números.

Con esto en mente, se procede a hacer otro análisis de los resultados usando la distancia de Levenshtein (distancia de edición) como métrica.

$Precision = 1 - (dL/L_{real})$

$aciertos = 1 - errores$

La nueva precisión se calcula de la siguiente forma: dL es la distancia de Levenshtein entre la predicción del modelo y el valor real; como dL son números enteros, necesitamos una forma de normalizarla, por lo que la dividimos entre la longitud del valor real. En el caso de un acierto completo, la distancia de Levenshtein sería cero, por lo que la precisión para esa imagen sería 1, un 100 %, pero en el otro caso, predecir **"1112"** siendo el número real **"1111"**, sería $Precision = 1 - (1/4) = 0{,}75$, un 75 %. Los resultados despues de aplicar esta nueva métrica pueden observarse en la tabla 6.3.

	v1	**v2**	**v3**	**v4**	**v5**
set_1	72 %	65 %	63 %	78 %	84 %
set_2	83 %	63 %	81 %	84 %	85 %
set_3	71 %	62 %	59 %	74 %	83 %
Total	**74 %**	**64 %**	**67 %**	**79 %**	**85 %**

Tabla 6.3: Tabla de resultados sobre el conjunto de datos RBNR, el *set* 1, *set* 2 y *set* 3 tienen 100, 77 y 113 dorsales repectivamente. Los datos de la tabla son las precisiones calculadas usando la distancia de Levenshtein sobre los diferentes *sets* de imágenes.

CAPÍTULO 7
Conclusiones

7.1 Reconocimiento de dorsales

En este aspecto, los resultados de las dos primeras partes del modelo, detección de personas y dorsales, obtienen unos resultados bastante buenos. En el RBNR *dataset* no solo es capaz de detectar todos los dorsales, sino que además detecta dorsales que no están anotados (figura 6.1). Esto no quiere decir que no haya falsos positivos, pero podría solucionarse aplicando un filtro de confianza a los dorsales detectados.

En el TGCRbNW alcanza una precisión del 55 % si usamos un margen de 30 píxeles, una medida razonable si consideramos la resolución de las imagenes, que son de 1920x1080 píxeles [14]. Aparte de esto, ninguna imagen de este *dataset* fue usada para el entrenamiento del modelo, lo que demuestra la capacidad de generalización de éste.

7.2 Reconocimiento de texto

Esta es sin duda la parte en la que más sufre el modelo. En total se obtiene un porcentaje de aciertos absolutos del 55 % en el RBNR *dataset*, y tan solo en el *set* 1 conseguimos resultados medianamente comparables (61 % frente a 66 %). Los resultados muestran que los peores resultados del modelo se dan en el *set* 3, independientemente de la versión. Un análisis manual de este conjunto no revela ningún detalle importante a simple vista, como una peor calidad de las imágenes, o mala visibilidad de los dorsales. Una posible explicación podría ser el tipo de fuente usada: los "1"s se leen como "I"s, pero al margen de esta teoría, no hay resultados concluyentes.

Sin embargo, como ya se ha comentado en la sección 6.2.2, estos primeros resultados pueden crear dudas sobre la actual precisión del modelo. Si utilizamos la distancia de Levenshtein como métrica, los resultados finales alcanzan un 85 % de precisión (tabla 6.3), teniendo en cuenta que ciertos carácteres no son leídos correctamente.

7.3 Trabajos futuros

Como ya se ha mencionado, de cara al futuro podría ser interesante probar diferentes formas de lectura de texto, ya que los resultados aún tienen margen de mejora.

Muchos modelos actualmente tienen buen rendimiento, pero suelen estar centrados en documentos de texto, no en secuencias cortas de texto, como puede ser leer una serie de números. Aparte, éstos suelen estar entrenados para reconocer carácteres, siendo los números algo secundario. Como ya tenemos los datos procesados, un proyecto para

seguir desarrollando podría ser entrenar un modelo desde cero, construyendo la propia arquitectura, probablemente imitando modelos similares al CRNN [43].

Aparte, una buena contribución a la comunidad podría ser la elaboración de un conjunto de datos para el reconocimiento de dorsales. Ya que nuestro sistema en este aspecto da buenos resultados, sería un proceso de ir "limpiando" errores, como falsos positivos o ajustando predicciones. Después de este paso habría que anotar a mano los números de los dorsales.

CAPÍTULO 8
Relación con los estudios cursados

La realización de este trabajo ha servido para poner en práctica los conocimientos obtenidos en este grado.

Para empezar, todas las asignaturas de programación, que asentaron las bases, como pueden ser "Introducción a la informática y a la programación" y "Programación".

"Ingeniería del software" y "Gestión de proyectos" dieron una perspectiva más practica a la hora de afrontar proyectos, con técnicas de gestión del tiempo y estructura de proyectos.

A la hora de depurar el código, lo aprendido en "Computación paralela", "Estructuras de datos y algoritmos" y "Computabilidad y complejidad" ayudó a la hora de reducir el tiempo de ejecución.

"Sistemas inteligentes" y "Percepción" ofrecieron una introducción a las redes neuronales y a sistemas de visión por ordenador, sin los cuales no habría podido realizar este trabajo.

Personalmente, las asignaturas que curse en la Radboud University en el curso 2021-2022 fueron también de gran ayuda para este proyecto. "Statistical Machine Learning", "Advanced Machine Learning" y "Research Seminar Data Science" fueron realmente importantes para establecer las bases del *machine learning*, mientras que "Machine Learning in Practice" y "Deep Learning" ofrecen una perspectiva más práctica a la hora del aprendizaje.

Agradecimientos

Recuerdo que al empezar a escribir este trabajo se me hizo una montaña, pero poco a poco, con la ayuda de muchas personas, pude avanzar hasta este punto.

En primer lugar a mi tutor, que a pesar de todos los problemas e imprevistos, siempre estuvo ahí para echarme una mano. Perdón por todas las faltas de ortografía que se me pasaban cada vez que tocaba hacer una revisión.

Luego por supuesto a mi familia, que me han visto unos pocos demasiados días metido en mi habitación para sacar este proyecto adelante.

Aparte, a mis amigos, que me hicieron enfocarme en este trabajo en la misma medida que me distrajeron de trabajar.

Jamas pensé que un número fuese a significar tanto para mí, pero a todo el 149, gracias por estar ahí.

Agradecer por último a Pablo Hernández Carrascosa, Adrian Penate Sanchez, Javier Lorenzo Navarro, David Freire Obregón y Modesto Castrillón Santana por permitirme utilizar su conjunto de datos para la realización de este trabajo

"Open your eyes and then open your eyes again." - Terry Pratchett

Bibliografía

[1] Sahla Muhammed Ali. «Comparative Analysis of YOLOv3, YOLOv4 and YOLOv5 for Sign Language Detection». En: *Internation Journal Of Advance Research And Innovative Ideas In Education* 7 (4 2021), págs. 2393-2398. URL: http://ijariie.com/AdminUploadPdf/Comparative_Analysis_of_YOLOv3__YOLOv4_and_YOLOv5_for_Sign_Language_Detection_ijariie15253.pdf.

[2] Md. Zahangir Alom et al. «The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches». En: *CoRR* abs/1803.01164 (2018). URL: http://arxiv.org/abs/1803.01164.

[3] Anuntapat Anuntachai, Wanatphong Chaorattana y Jutatip Boonchoay. «Runner BIB number recognition system». En: *International Conference on Control, Automation and Systems* 2017-October (2017). ISSN: 15987833. DOI: 10.23919/ICCAS.2017.8204235.

[4] Youngmin Baek et al. «Character Region Awareness for Text Detection». En: *CoRR* abs/1904.01941 (2019). URL: http://arxiv.org/abs/1904.01941.

[5] Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao. «YOLOv4: Optimal Speed and Accuracy of Object Detection». En: *CoRR* abs/2004.10934 (2020). URL: https://arxiv.org/abs/2004.10934.

[6] Nicolas Carion et al. «End-to-End Object Detection with Transformers». En: *CoRR* abs/2005.12872 (2020). URL: https://arxiv.org/abs/2005.12872.

[7] Jiankang Deng et al. «RetinaFace: Single-stage Dense Face Localisation in the Wild». En: *CoRR* abs/1905.00641 (2019). URL: http://arxiv.org/abs/1905.00641.

[8] *Detecta texto en imágenes | API de Cloud Vision | Google Cloud*. URL: https://cloud.google.com/vision/docs/ocr.

[9] Pascal Getreuer. «ASurvey of Gaussian Convolution Algorithms». En: *Image Processing on Line* 3 (dic. de 2013), págs. 286-310. DOI: 10.5201/ipol.2013.87. URL: https://doi.org/10.5201%2Fipol.2013.87.

[10] Javad Ghofrani et al. *Machine Vision in the Context of Robotics: A Systematic Literature Review*. 2019. DOI: 10.48550/ARXIV.1905.03708. URL: https://arxiv.org/abs/1905.03708.

[11] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2013. DOI: 10.48550/ARXIV.1311.2524. URL: https://arxiv.org/abs/1311.2524.

[12] Ross B Girshick. «Fast R-CNN». En: *CoRR* abs/1504.08083 (2015). URL: http://arxiv.org/abs/1504.08083.

[13] Kaiming He et al. «Deep Residual Learning for Image Recognition». En: *CoRR* abs/1512.03385 (2015). URL: http://arxiv.org/abs/1512.03385.

[14] Pablo Hernández-Carrascosa et al. «TGCRbNW: A dataset for runner bib number detection (and recognition) in the wild». En: *Proceedings - International Conference on Pattern Recognition* (2020), págs. 9445-9451. ISSN: 10514651. DOI: 10.1109/ICPR48806.2021.9412220.

[15] *Home - OpenCV*. URL: https://opencv.org/.

[16] Gao Huang, Zhuang Liu y Kilian Q Weinberger. «Densely Connected Convolutional Networks». En: *CoRR* abs/1608.06993 (2016). URL: http://arxiv.org/abs/1608.06993.

[17] *Impact of Computer Vision Applications on the World*. URL: https://brighterkashmir.com/impact-of-computer-vision-applications-on-the-world.

[18] Joel Janai et al. *Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art*. 2017. DOI: 10.48550/ARXIV.1704.05519. URL: https://arxiv.org/abs/1704.05519.

[19] *Kaggle: Your Home for Data Science*. URL: https://www.kaggle.com/.

[20] Saranya Kanjaruek y Noppakun Boonsim. «Racing BiB number localization based on region convolutional neural networks». En: jun. de 2018.

[21] Dimosthenis Karatzas et al. «ICDAR 2013 Robust Reading Competition». En: Proceedings of ICDAR 2013, 2013, págs. 1484-1493. ISBN: 9780769549996. DOI: 10.1109/ICDAR.2013.221. URL: https://doi.org/10.1109/ICDAR.2013.221.

[22] Muhammed Can Keles et al. «Evaluation of YOLO Models with Sliced Inference for Small Object Detection». En: *ArXiv* abs/2203.04799 (2022). URL: https://arxiv.org/abs/2203.04799.

[23] *keras ocr documentation*. URL: https://keras-ocr.readthedocs.io.

[24] Seungkwan Lee, Suha Kwak y Minsu Cho. «Universal Bounding Box Regression and Its Applications». En: *CoRR* abs/1904.06805 (2019). URL: http://arxiv.org/abs/1904.06805.

[25] Tsung-Yi Lin et al. «Feature Pyramid Networks for Object Detection». En: *CoRR* abs/1612.03144 (2016). URL: http://arxiv.org/abs/1612.03144.

[26] Tsung-Yi Lin et al. «Focal Loss for Dense Object Detection». En: *CoRR* abs/1708.02002 (2017). URL: http://arxiv.org/abs/1708.02002.

[27] Shu Liu et al. «Path Aggregation Network for Instance Segmentation». En: *CoRR* abs/1803.01534 (2018). URL: http://arxiv.org/abs/1803.01534.

[28] Simon M. Lucas et al. «ICDAR 2003 robust reading competitions: Entries, results, and future directions». En: *International Journal on Document Analysis and Recognition* 7 (2-3 jul. de 2005), págs. 105-122. ISSN: 14332833. DOI: 10.1007/S10032-004-0134-3. URL: https://www.researchgate.net/publication/220163548_ICDAR_2003_Robust_Reading_Competitions_Entries_Results_and_Future_Directions.

[29] Akshansh Mishra. «Contrast Limited Adaptive Histogram Equalization (CLAHE) Approach for Enhancement of the Microstructures of Friction Stir Welded Joints». En: *CoRR* abs/2109.00886 (2021). URL: https://arxiv.org/abs/2109.00886.

[30] Anand Mishra, Karteek Alahari y C. V. Jawahar. «Scene text recognition using higher order language priors». En: *BMVC 2012 - Electronic Proceedings of the British Machine Vision Conference 2012* (2012). DOI: 10.5244/C.26.127. URL: https://www.researchgate.net/publication/255565360_Scene_Text_Recognition_using_Higher_Order_Language_Priors.

[31] *MNIST Dataset | Papers With Code*. URL: https://paperswithcode.com/dataset/mnist.

[32] Sauradip Nag et al. «A New Unified Method for Detecting Text from Marathon Runners and Sports Players in Video». En: *CoRR* abs/2005.12524 (2020). URL: https://arxiv.org/abs/2005.12524.

[33] *pytesseract · PyPI*. URL: https://pypi.org/project/pytesseract/.

[34] *PyTorch*. URL: https://pytorch.org/.

[35] Mahdi Rad, Markus Oberweger y Vincent Lepetit. «Feature Mapping for Learning Fast and Accurate 3D Pose Inference from Synthetic Images». En: *CoRR* abs/1712.03904 (2017). URL: http://arxiv.org/abs/1712.03904.

[36] Muhammad Rayhan y Kemas Lhaksmana. «Racing Bib Number Recognition Method using Deep Learning». En: *JURNAL MEDIA INFORMATIKA BUDIDARMA* 4 (sep. de 2020), pág. 815. DOI: 10.30865/mib.v4i3.2270.

[37] Joseph Redmon y Ali Farhadi. «YOLOv3: An Incremental Improvement». En: *CoRR* abs/1804.02767 (2018). URL: http://arxiv.org/abs/1804.02767.

[38] Joseph Redmon et al. «You Only Look Once: Unified, Real-Time Object Detection». En: *CoRR* abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640.

[39] Shaoqing Ren et al. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks». En: *CoRR* abs/1506.01497 (2015). URL: http://arxiv.org/abs/1506.01497.

[40] Hamza Riaz y Alan F. Smeaton. *Vision Based Machine Learning Algorithms for Out-of-Distribution Generalisation*. 2023. DOI: 10.48550/ARXIV.2301.06975. URL: https://arxiv.org/abs/2301.06975.

[41] *Roboflow dataset personalizado*. URL: https://universe.roboflow.com/tocaloupvtfg/bibs-rmyuk/health.

[42] *Roboflow: Give your software the power to see objects in images and video*. URL: https://roboflow.com/.

[43] Baoguang Shi, Xiang Bai y Cong Yao. «An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition». En: *CoRR* abs/1507.05717 (2015). URL: http://arxiv.org/abs/1507.05717.

[44] Mingxing Tan y Quoc V Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». En: *CoRR* abs/1905.11946 (2019). URL: http://arxiv.org/abs/1905.11946.

[45] Mingxing Tan, Ruoming Pang y Quoc V Le. «EfficientDet: Scalable and Efficient Object Detection». En: *CoRR* abs/1911.09070 (2019). URL: http://arxiv.org/abs/1911.09070.

[46] *TensorFlow*. URL: https://www.tensorflow.org.

[47] Ashish Vaswani et al. «Attention Is All You Need». En: *CoRR* abs/1706.03762 (2017). URL: http://arxiv.org/abs/1706.03762.

[48] Chien-Yao Wang, Alexey Bochkovskiy y Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. Jul. de 2022. URL: https://github.com/WongKinYiu/yolov7.

[49] Chien-Yao Wang et al. «CSPNet: A New Backbone that can Enhance Learning Capability of CNN». En: *CoRR* abs/1911.11929 (2019). URL: http://arxiv.org/abs/1911.11929.

[50] Yan Chiew Wong et al. «Deep learning-based racing bib number detection and recognition». En: *Jordanian Journal of Computers and Information Technology* 5 (3 dic. de 2019), págs. 181-194. ISSN: 24151076. DOI: 10.5455/jjcit.71-1562747728.

[51] Saining Xie et al. «Aggregated Residual Transformations for Deep Neural Networks». En: *CoRR* abs/1611.05431 (2016). URL: http://arxiv.org/abs/1611.05431.

[52] Michael Ying Yang. «Visual Transformer for Object Detection». En: *ArXiv* abs/2206.06323 (2022).

[53] Xiang Zhang et al. «Text Spotting Transformers». En: *arXiv e-prints* (abr. de 2022), arXiv:2204.01918.

[54] Zhong Qiu Zhao et al. *Object Detection with Deep Learning: A Review*. 2019. DOI: 10.1109/TNNLS.2018.2876865.

APÉNDICE A
Glosario de términos

BIB: Dorsal.

Perceptrón: Algoritmo para clasificación binaria (dos clases) de datos.

Red neuronal: Sistema compuesto por nodos o "neuronas" capaces de realizar predicciones en base a unos datos de entrada.

Red neuronal convolucional: Red neuronal en la que las neuronas reciben una "zona" de datos. Esta zona puede ser, por ejemplo, una matriz de n por n valores. Esta forma de procesar datos permite un mayor rendimiento a la hora de trabajar con imágenes, que pueden considerarse una matriz (o tres, si se está trabajando con valores RGB), pasando así secciones de la imagen.

Red neuronal recurrente: *RNN*, red neuronal en la que las diferentes capas de nodos están interconectadas, por lo que el estado del sistema actual condiciona la siguiente respuesta, permitiendo al sistema tener "memoria". Estos modelos se utilizan para tratar secuencias de datos en los que el orden es importante, como texto o audio.

LSTMs: (*Long Short Term Memory*), red neuronal recurrente diseñada para poder procesar secuencias de datos, como video, texto o audio. Esto se consigue añadiendo conexiones entre los diferentes valores. En el caso de procesamiento de texto, se tienen en cuenta las palabras anteriores antes de procesar la actual.

Dataset: Conjunto de datos, en este caso, imágenes.

Threshold: Valor mínimo necesario para que algo ocurra.

Data augmentation: Técnicas para aumentar artificialmente la cantidad de datos de entrenamiento. En imágenes normalmente se utilizan cambios de color, rotaciones de imagen, recortes aleatorios, etc.

Epoch: A la hora de entrenar redes neuronales, un *epoch* hace referencia a un paso o ciclo de entrenamiento.

Bounding Box: En detección de objetos, se refiere al área que delimita al objeto identificado.

APÉNDICE B

Ejemplos de imágenes de los conjuntos de datos utilizados

B.1 RBNR

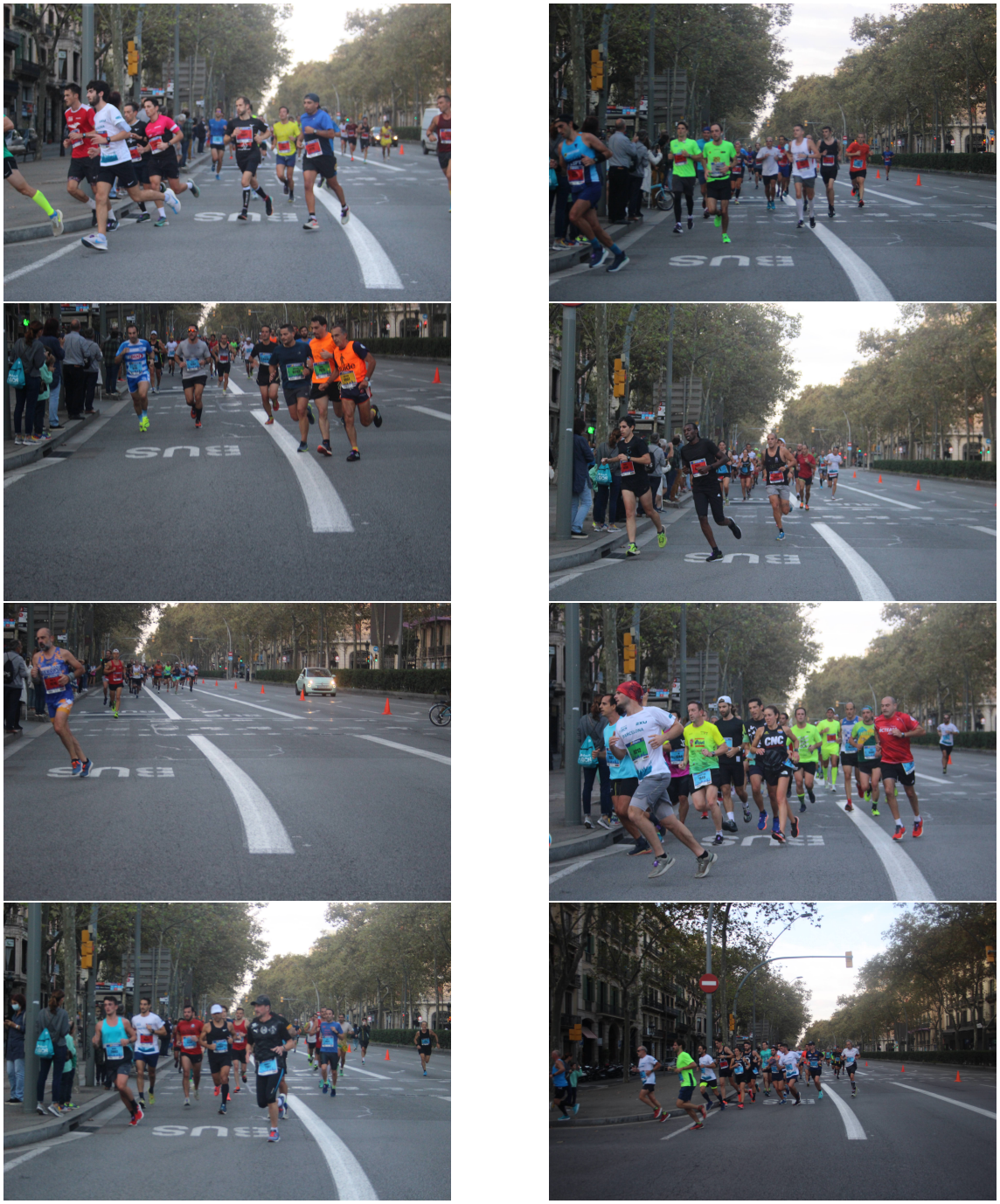
Ejemplos de imágenes del *RBNR dataset* [36]

B.2 TGCRbNW

Ejemplos de imágenes del *TGCRbNW dataset* [14]

B.3 Media maratón de Barcelona 2021

Ejemplos de imágenes de la media maratón de Barcelona 2021



APÉNDICE C
Objetivos de Desarrollo Sostenible

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**			X	
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**			X	
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**			X	
ODS 7. **Energía asequible y no contaminante.**			X	
ODS 8. **Trabajo decente y crecimiento económico.**			X	
ODS 9. **Industria, innovación e infraestructuras.**			X	
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**			X	
ODS 12. **Producción y consumo responsables.**			X	
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**			X	
ODS 15. **Vida de ecosistemas terrestres.**			X	
ODS 16. **Paz, justicia e instituciones sólidas.**			X	
ODS 17. **Alianzas para lograr objetivos.**			X	

El reconocimiento de dorsales es una tarea importante en el procesamiento de imágenes deportivas, ya que permite la identificación y clasificación de las fotografías de los participantes de un evento deportivo en particular. Esto es especialmente útil en carreras populares, donde cientos o miles de personas corren y quieren encontrar sus fotos al finalizar la carrera.

El uso de un modelo de reconocimiento de dorsales puede ayudar a identificar a los corredores de manera más eficiente y precisa, lo que facilita el proceso de búsqueda de fotos y puede mejorar la experiencia del participante en el evento.

El desarrollo de este trabajo puede tener un impacto indirecto positivo en la salud de los participantes de la carrera. Al poder encontrar fácilmente sus fotos, los corredores pueden revivir su experiencia en el evento y celebrar su logro con amigos y familiares. Esto puede mejorar la autoestima y motivación de los participantes para continuar su entrenamiento y participar en futuros eventos deportivos, lo que propiciaría beneficios positivos para la salud y el bienestar.